UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSTGRADO
MAESTRÍA EN COMPUTACIÓN

ANÁLISIS DE LAS TÉCNICAS DE LOCALIZACIÓN DE OBJETOS MÓVILES

MÉRIDA, febrero 2010

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSTGRADO
MAESTRÍA EN COMPUTACIÓN

ANÁLISIS DE LAS TÉCNICAS DE LOCALIZACIÓN DE OBJETOS MÓVILES

Autor: Carmen Janeth Parada
Tutor: Leandro León

MÉRIDA, febrero 2010

ÍNDICE GENERAL

	Página
INTRODUCCION	1
1. TRABAJOS RELACIONADOS CON LA LOCALIZACIÓN DE OBJETOS	3
2. TÉCNICAS DE LOCALIZACIÓN	6
2.1 DESCRIPCION DE LAS TÉCNICAS DE LOCALIZACIÓN DE RECURSOS	6
2.1.1 DIFUSIÓN.	6
2.1.2 LOCALIZACIÓN MEDIANTE CACHES.	7
2.1.3 PREFETCHING.	8
2.1.4 PIGGYBACK.	9
2.1.4.1 Prioridades de los piggybacks.	9
2.1.4.2 Gestión de un piggyback.	9
2.1.4.3 Control de congestión de piggybacks	10
2.2 CARACTERIZACIONES DE UNA FALLA DE ACCESO	11
2.2.1 Fallas Transparentes (F_i).	11
2.2.2 Fallas Simples (Fs_i).	11
2.2.3 Fallas Notables (Fn_i).	12
2.3 BÚSQUEDA DE RECURSOS.	12
2.3.1 Modos de Búsqueda	12
2.3.2 Tipos de Búsqueda.	14
2.4 MIGRACIÓN DE UN RECURSO.	14
3 MODELO	16
3.1 CREACIÓN DE UN RECURSO	16

Página

3.2. COMUNICACIÓN. 17

3.3 FLUJO DE ACCESO 19

3.3.1 Emisión de un *request*. 20

3.3.2 Recepción de un *request*. 20

3.3.3 Recepción de respuesta a un *request* 21

3.4. CACHES 22

3.5. PREFETCHING. 22

3.6. INVOCACIONES O ACCESOS 22

3.7. MIGRACIONES 24

4. DISEÑO Y ANÁLISIS DE EXPERIMENTOS 25

4.1 DISEÑO Y ANÁLISIS DE EXPERIMENTOS PARA VALIDAR EL MODELO 25

4.1.1 Validación del modelo usando valores con y sin multiplicidad 25

4.1.2 Validación del modelo usando valores con multiplicidad y variando los valores aleatorios 27

4.1.3 Validación del modelo usando valores con multiplicidad y variando los valores aleatorios 30

4.2 DISEÑO Y ANÁLISIS DE ESPERIMENTOS PARA DETERMINAR LA EFICIENCIA Y EL COSTO DE LA TÉCNICA DIFUSIÓN 35

4.3 DISEÑO Y ANÁLISIS DE EXPERIMENTOS PARA DETERMINAR LA EFICIENCIA Y EL COSTO DE LOS CACHE*S ARRIVING*, *LIVE* Y *DEAD* 37

4.3.1 *Cache Arriving* 38

4.3.2 *Cache live* 43

4.3.3 *Cache Dead* 49

4.4 DISEÑO Y ANÁLISIS DE EXPERIMENTOS PARA DETERMINAR LA EFICIENCIA Y EL COSTO DEL *CACHE STOP* 54

4.4.1 Experimentos para determinar la utilidad del *cache stop* 54

4.4.2 Experimento para determinar el comportamiento del *cache stop* 57

Página

4.5 DISEÑO Y ANÁLISIS DE EXPERIMENTOS PARA DETERMINAR LA EFICIENCIA Y EL COSTO DEL USO DEL *PIGGYBACK* 62

5. CONCLUSIÓN 68

REFERENCIAS BIBLIOGRÁFICAS 69

ÍNDICE DE FIGURAS

Página

Figura 1 Localización de un recurso usando difusión 7
Figura 3 Búsqueda sin parada 12
Figura 4 Búsqueda con parada 13
Figura 5 Migración de un recurso 15
Figura 6 Proceso de emisión de un *reques* 20
Figura 7 Proceso de recepción de un *request* 21
Figura 8 Proceso de recepción de respuesta a un *request* 21

ÍNDICE DE TABLAS

Página

Tabla 1 Estructura de una cola del *piggyback* 10

Tabla 2 Variables de respuesta referentes a la comunicación 17
Tabla 3 Tipos de mensajes de acceso al recurso 18
Tabla 4 Tipos de mensajes de control de localización. 19
Tabla 5 Factores referentes al *prefetching* 22
Tabla 7 Valores definidos para los factores a_3 y a_5 26
Tabla 8 Valores definidos para los factores a_3 y a_5 con multiplicidad 28
Tabla 9. Valores definidos para los factores a_3 y a_5 sin multiplicidad 31
Tabla 10 Niveles del tamaño del *arriving* 38
Tabla 11 Factores de análisis para el *Live* 44
Tabla 12 Factores de análisis para el *Dead* 49
Tabla 13 Niveles de los tiempo de migración del objeto del *stop* 58
Tabla 14 Niveles del tamaño del *arriving* 63

ÍNDICE DE GRÁFICAS

		Página
Grafica 1	Invocaciones con multiplicidad de los factores	27
Grafica 2	Invocaciones sin multiplicidad de los factores	27
Gráfica 3	Porcentaje de diferencia en las invocaciones usando valores con multiplicidad de los factores	29
Gráfica 4	Porcentaje de diferencia en las objetos creados usando valores con multiplicidad de los factores	29
Gráfica 5	Porcentaje de diferencia en las referencias creadas usando valores con multiplicidad de los factores	30
Gráfica 6	Porcentaje de diferencia en las invocaciones usando valores sin multiplicidad de los factores, variando valores aleatorios a3>a5	32
Gráfica 7	Porcentaje de diferencia en las invocaciones usando valores sin multiplicidad, variando valores aleatorios, a5>a3	32
Grafica 8	Porcentaje de diferencia de objetos creados sin multiplicidad de los factores, variando valores aleatorios, a3>a5	33
Grafica 9	Porcentaje de diferencia de objetos creados sin multiplicidad de los factores variando los valores aleatorios, a5>a3	33
Grafica 10	Porcentaje de diferencia de objetos creados sin multiplicidad de los factores variando los valores aleatorios, a5>a3	34
Grafica 11	Porcentaje de diferencia de objetos creados sin multiplicidad de los factores variando los valores aleatorios, a5<a3	34
Gráfica 12	Fallas transparentes con solo difusión y	36
Gráfica 13	Fallas Notables con difusión y	36
Gráfica 14	Niveles de fallas notables con difusión y	36
Gráfica 15	Invocaciones realizadas realizando solo difusión y	37
Gráfica 16	Invocaciones realizadas variando el tamaño del *arriving* y la frecuencia de acceso del objeto.	39
Gráfica 17	Mensajes fuertes realizados variando el tamaño del *arriving* y la frecuencia de acceso del objeto	39
Gráfica 18	Mensajes débiles realizados variando el tamaño del *arriving* y la frecuencia de acceso del objeto	40
Gráfica 19	*Broadcast* fuertes realizados variando el tamaño del *arriving* y la frecuencia de acceso del objeto	40
Gráfica 20	*Broadcast* débiles realizados variando el tamaño del *arriving* y la frecuencia de acceso del objeto	41
Gráfica 21	Fallas generada variando el tamaño del *arriving* y la	41

		Página
	frecuencia de acceso del objeto	
Gráfica 22	Niveles de fallas generadas variando el tamaño del *arriving* y la frecuencia de acceso del objeto	42
Gráfica 23	Invocaciones realizadas usando el *live*	45
Gráfica 24	Mensajes fuertes enviados y recibidos usando el *live*	45
Gráfica 25	Mensajes fuertes enviados y recibidos usando el *live*	46
Gráfica 26	*Broadcast* fuertes enviados y recibidos usando el *live*	46
Gráfica 27	*Broadcast* débiles enviados y recibidos usando el *live*	47
Gráfica 28	Fallas generadas usando el *live*	47
Gráfica 29	Niveles de fallas generadas usando el *live*	48
Gráfica 30	Invocaciones realizadas usando el *dead*	50
Gráfica 31	Mensajes fuertes enviados y recibidos usando el *dead*	50
Gráfica 32	Mensajes débiless enviados y recibidos usando el *dead*	51
Gráfica 33	*Broadcast* fuertes enviados y recibidos usando el *dead*	51
Gráfica 34	*Broadcast* débiles enviados y recibidos usando el *dead*	52
Gráfica 35	Fallas generadas usando el *dead*	52
Gráfica 36	Niveles de fallas generadas usando el *dead*	53
Gráfica 37	Invocaciones realizadas usando el *stop*	55
Gráfica 38	Migraciones realizadas usando el *stop*	55
Gráfica 39	Broadcast ejecutados usando el *stop*	56
Gráfica 40	Fallas realizadas usando el *stop*	56
Gráfica 41	Niveles de fallas realizadas usando el *stop*	57
Gráfica 42	Mensajes enviados usando el *stop*	59
Gráfica 43	*Broadcast* fuertes realizados usando el *stop*	59
Gráfica 44	*Broadcast* débiles realizados usando el *stop*	60
Gráfica 45	Fallas generadas usando el *stop*	60
Gráfica 46	Invocaciones realizadas usando el *stop*	61
Gráfica 47	Migraciones realizados usando el *stop*	61
Gráfica 48	Niveles de fallas generados usando el *stop*	62
Gráfica 49	Invocaciones realizadas usando el *piggyback*	64
Gráfica 50	Broadcast realizados usando el *piggyback*	64
Gráfica 51	Fallas generadas usando el *piggyback*	65
Gráfica 52	Mensajes fuertes enviados usando el *piggyback*	65
Gráfica 53	Mensajes débiles enviados usando el *piggyback*	66
Gráfica 54	Niveles de fallas usando el *piggyback*	66

RESUMEN

La escalabilidad de los sistemas distribuidos ha hecho que existan redes que constantemente incremente el número de computadores conectados, generando a su vez un incremento considerable de los recursos que se movilizan por el sistema; la migración constante de estos recursos, como por ejemplo, un dispositivo de hardware, un computador, una impresora, una pagina *web* o un agente, hace que su localización sea un problema latente el cual se puede convertir en un gran inconveniente en la evolución de estos tipos de sistemas.

Debido a la constante migración de los recursos, y a su invocación, se han desarrollado una serie de técnicas que permiten su localización; entre las técnicas mas conocidas están difusión, *prefetching*, *piggyback* y *caches*. Estas técnicas han sido modeladas para analizar el comportamiento de cada una de estas técnicas y poder determinar su eficiencia; es decir, determinar si las técnicas garantizan que un recurso solicitado sea localizado en un tiempo acotado; y el costo computacional que se genera.

En el proyecto se diseñan experimentos, donde se definen los factores que permiten realizar el estudio de las técnicas de localización de un objeto cuando este migra en el sistema, se determinan las variables de respuesta que evidencien el costo en que incurre el sistema en el momento de usar las técnicas y determinar si estas técnicas son eficientes cuando son usadas solas, o por lo contrario, ellas se complementan para hacer que el costo en que incurre el sistema se disminuya y se incremente la eficiencia.

Palabras claves: localización de objetos, migración objetos, sistema distribuido, difusión, piggyback, preffeching, caches.

INTRODUCCIÓN

Un sistema distribuido se define como una colección de computadores autónomos conectados por una red, y con el software distribuido adecuado para que el sistema sea visto por los usuarios como una única entidad capaz de proporcionar facilidades de computación [1]; la conexión de las computadores que hacen parte del sistema, permite la movilidad de los recursos[1], siendo su localización un problema complejo y costoso.

La constante migración de los recursos; esto es, el hecho que el recurso puede cambiar de sitio donde esté ubicado, hace que el sistema sea muy dinámico; esta migración se da por la búsqueda del uso óptimo de los mismos, la administración, la tolerancia a fallos, o para efectuar una tarea computacional, siendo esta última de vital importancia en un sistema distribuido [3].

Es así, como la escalabilidad de los sistemas distribuidos ha hecho que existan redes que constantemente incremente el número de computadores conectados, generando a su vez un incremento considerable de los recursos que se movilizan por el sistema; la migración constante de estos recursos, como por ejemplo, un dispositivo de hardware, un computador, una impresora, una pagina *web* o un agente, hace que su localización sea un problema latente el cual se puede convertir en un gran inconveniente en la evolución de estos tipos de sistemas.

Debido a la constante migración de los recursos, y a su invocación, se han desarrollado una serie de técnicas que permiten su localización; entre las técnicas mas conocidas están difusión, *prefetching, piggyback* y *caches*. Estas técnicas fueron modeladas en el proyecto "Localización de objetos distribuidos móviles" realizado por Andrés Arcia y Leandro León [3].

El propósito de este proyecto es analizar el comportamiento de cada una de estas técnicas para determinar su eficiencia; es decir, determinar si las técnicas garantizan que un recurso solicitado sea localizado en un tiempo acotado; y el costo computacional que se genera.

El análisis se desarrolló basado en los resultados arrojados por el modelo que se implementó en *OMNeT*[2] + + el cual es extensible, modular, basada en componentes C + + y el marco de simulación, con un IDE basado en Eclipse y un entorno de ejecución de gráficos.

---

[1] Los recursos que pueden migrar son: un agente, un dispositivo, una página web, ect

[2] **OMNeT++** es un simulador modular de eventos discretos de redes orientado a objetos, usado habitualmente para modelar el tráfico de redes de telecomunicaciones, sistemas multiprocesadores y distribuidos

El documento se estructura en cinco capítulos. El esquema se organiza de forma que se parte de los aspectos genéricos y la descripción del modelo, hasta el análisis de resultados y las conclusiones

En el capitulo uno se presentan los trabajos realizados con la localización de objetos.

En el capitulo dos se presentan los aspectos relacionados con las técnicas de localización de recursos, tipos de búsqueda y tipos de fallas. Su fin es establecer una base adecuada para extraer las nociones que son contempladas en el modelo.

El capitulo tres proporciona una descripción del modelo de simulación a eventos discretos sobre el cual se ejecutan las simulaciones, que observan variables de respuesta pertinentes para el análisis del comportamiento de las técnicas de localización.

El capitulo cuatro constituye el núcleo del trabajo. Tras considerar brevemente el problema de localización de un recurso, se presenta el diseño de los experimentos que permiten el análisis de las variables de respuesta de cada una de las técnicas de localización de objetos en un sistema distribuido, para determinar si las técnicas son eficientes.

Finalmente, en el capitulo cinco se presentan las conclusiones del trabajo.

1. TRABAJOS RELACIONADOS CON LA LOCALIZACIÓN DE OBJETOS

En esta sección se relacionan las técnicas que han sido propuestas en los diferente proyectos que se han desarrollado hasta el momento en lo que tiene que ver con la localización de objetos. A continuación se indican algunos de ellos:

- En la técnica de difusión, el ente migrante efectúa una difusión informando de su nueva ubicación. Para informar de su nueva localización lo puede hacer informando a todos los sitios del sistema o efectuando la difusión solo hacia los clientes del objeto que está migrando; esta última forma de hacer la difusión implica que se debe conocer todos los objetos enlazados al objeto que está migrando. En los proyectos *Demo/MP* [17] y *System V* [21], se consideró el uso de la difusión, mas no se implemento, debido a que el número de clientes debe ser pequeño.

- En los proyectos *Emerald* [12], SOS [18] y *Amber* [22] se utilizó el enfoque de prueba y actualización para determinar si la referencia era local o no. Esto es, antes que se realice la migración, el cliente verifica si el servidor se encuentra en la localización esperada. Si el servidor está ausente, el cliente pregunta por la nueva localización a un servicio especial de localización. Igualmente, Esmerald utilizó los lazos de persecución, la cual consiste en una indicación dejada en el sitio de origen de la migración y que apunta al sitio destino. Cuando el cliente realiza una invocación, ésta llega al sitio de origen y desde allí es enviada al sitio destino. La cadena de lazos se incrementa a medida que se incrementa el número de sitios nuevos que se han visitado en las migraciones sucesivas.

- En el sistema operativo *Galaxy* [23] se usan lazos de persecución e identificadores únicos para todos los objetos del sistema. El acceso es transparente a la localidad, lo que facilita la actualización de los lazos. Antes del congelamiento[3], un servidor de mensajes es informado de la nueva localidad, lo cual facilita la entrega del mensaje a los sitios destinos.

- En el sistema *DC*$^{++}$ [24] se usó los lazos de persecución y se implementó un enfoque de cadena de compresión. Se introdujo el concepto de sitio de nacimiento, el cual permitía ofrecer enlaces tolerantes a fallas. Esto consiste en mantener sistemáticamente un enlace entre el servidor y el primer sitio desde donde el servidor fue llamado por primera vez.

[3] El congelamiento es un término usado en la migración, el cual consiste en detener la ejecución de un proceso de manera que su estado puede ser extraído y migrado.

- En el sistema *Soul* [16] se implantó las cadenas pss[4] en el cual, una rama es equivalente a un tronco del lado del servidor. Cuando un servidor migra, un nuevo par tronco-rama se crea en el sitio origen haciendo una conexión o *binding* con el sitio destino de la migración. La rama ancestral del sitio se actualiza haciéndola apuntar al nuevo tronco. De esta manera, el nuevo par puede ser visto como una forma de lazo de persecución [18]. Este enfoque actualiza la nueva localidad del servidor de manera parecida a la del corto circuito. La diferencia se da en que un tronco puede apuntar a dos ramas diferentes. A uno de los apuntadores se le denomina débil y al otro fuerte. El apuntador fuerte forma parte de la cadena originada como consecuencia de las migraciones y el apuntador débil es aquél que se actualiza cuando una invocación descubre una nueva actualización.

- En el sistema *Larchant* [3, 9] se implantó la recolección de lazos, el cual consiste en que los troncos son agrupados por recolectores distribuidos. Las ramas son recolectadas mediante un protocolo distribuido de recolección que utiliza la información provista por los recolectores locales. Se asumió una comunicación débil donde los mensajes pueden perderse arbitrariamente, retardarse o ser entregados en mal orden.

- En el sistema *Amoeba* [20] se utilizó la actualización de los enlaces por recuperación de mensajes perdidos. En este enfoque, un mensaje enviado a un proceso falla si el proceso ha migrado. Se inicia entonces la recuperación del mensaje perdido. La recuperación consiste en localizar el proceso migrante y hacerlo llegar al mensaje perdido. Varias técnicas son usadas para detectar la falla de la pérdida del mensaje: expiración de un tiempo límite de transmisión; respuesta de un sitio antiguo que diga que el proceso requerido no ha sido encontrado; y excepción emitida por el sistema de comunicaciones.

- En el Sistema COOL-LDM [13] se usa una variante para la actualización de los enlaces por recuperación de mensajes perdidos llamada “actualización bajo excepción”, la cual consiste en iniciar la localización de un objeto cuando ocurre una excepción del sistema de comunicación.

De las técnicas empleadas en los proyectos desarrollados se puede indicar que:

- Los métodos de actualización cándidos están basados en la difusión. Todos los tipos de difusión existentes presentan el problema de la sobrecarga de la red, lo cual solo es recomendable ser usada en sistemas pequeños donde hay pocos sitios, pocos objetos, y el movimiento de objetos es bajo.

  A pesar que la técnica de difusión tiene problemas de sobrecarga, es la técnica a usar cuando otras técnicas fallan en la ubicación de un objeto.

- Los lazos de persecución, presentan inconvenientes bastante importantes. El primero es la presencia de dependencias residuales que disminuyen el desempeño y aumentan la sensibilidad a fallas conforme crece la cadena de lazos. El segundo inconveniente es la necesidad de implementar un mecanismo de recolección de basura.

[4] Stub-scion: pares de tronco-rama

Todos los mecanismos de actualización basados en lazos de persecución tienen la desventaja de que tienen que ser considerados por el protocolo de migración; esto es, el protocolo de migración no puede reanudar la ejecución del objeto en el sitio destino hasta que no se haya creado el lazo de persecución.

2. TÉCNICAS DE LOCALIZACIÓN

Un recurso puede ser localizado por medio de una "referencia", que es una especie de puntero que contiene la información para acceder al recurso; esto hace que el sistema mantenga actualizada la información referente a la ubicación real del recurso para que esta se realice en forma efectiva.

Para localizar un recurso se han implementado una serie de técnicas que permiten cumplir este objetivo; entre estas técnicas están la difusión, los lazos de persecución (*forwarding)*, actualización por fracaso de invocación, etc. Lo que busca cada una de estas técnicas es que la localización de un recurso sea transparente, tenga un alto grado de fiabilidad, rendimiento, eficacia, eficiencia y escalabilidad; pero el costo[5] de implementación de algunas de estas técnicas es muy alto y son deficientes en el desempeño, escalabilidad y sensibilidad a fallas.

En la actualidad existen investigaciones relacionadas con la localización de objetos [3,4,6,8,9,11], empleados en dos grandes paradigmas como son los objetos distribuidos y los agentes móviles.

2.1 DESCRIPCION DE LAS TÉCNICAS DE LOCALIZACIÓN DE RECURSOS DE ESTE TRABAJO

La localización de un recurso se puede realizar usando las técnicas de difusión, caches, *prefetching y piggyback*. A continuación se describe cada una de estas técnicas.

2.1.1 DIFUSIÓN.

Esta técnica difunde por el sistema la nueva locación del recurso, pudiéndolo hacer de dos formas: difundiendo su nueva localización a todos los sitios del sistema o solo hacia los clientes del objeto que migra. Una variante de esta técnica fue utilizada por el sistema Charlotte [4], en el cual cuando ocurría una migración, esta no era realizada hasta que todas las referencias no fueran actualizadas y reconocidas. En la figura 1 se puede observar la forma como opera la difusión, en

[5] El costo se determina por el número de fallas notables que se producen en el momento en que el sistema busca el recurso

la que desde un sitio1 se busca en el sistema un recurso; el mensaje se envía a todos los sitios y cuando este es encontrado se le informa a quien lo solicita.

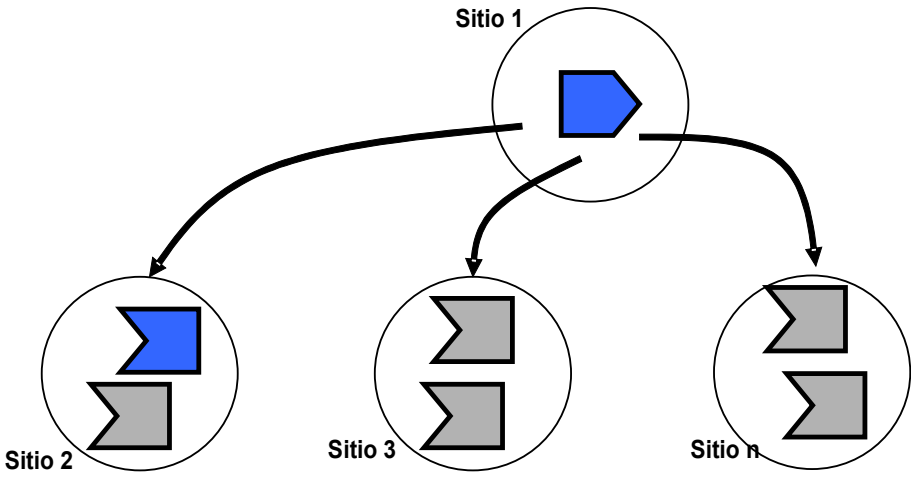


Figura 1. Localización de un recurso usando difusión. (Tomada de [3])

2.1.2 LOCALIZACIÓN MEDIANTE CACHES.

Esta técnica consiste en activar diferentes tipos de caches en cada localizador para almacenar los cálculos de la localización [3]. Los caches que se activan son: *arriving, live, dead* y el *stop.*

El *cache dead***,** permite guardar identificadores únicos de recursos que ya no existen; de este modo, un sitio, y consecuentemente una referencia, puede percatarse que un recurso que ya no existe, generando un ahorro de subsecuentes búsquedas. Este se actualiza en el sitio donde ocurre la eliminación de un recurso o cuando otro sitio recibe una notificación portando el A_{id} del recurso eliminado.

El *cache arriving,* guarda duplas de la forma $< A_{id}, l >$, donde A_{id} es un identificador de recurso y l es una lista finita de sitios referenciantes al recurso. Cada entrada de la lista l es un par $< i, t >$, donde i es el sitio donde se encuentran referencias hacia A_{id} y t es el tiempo físico, no lógico, en que el sitio i accedió a Ai_d. El tiempo físico t sirve para manejar la vigencia de la entrada en el sitio. Se actualiza desde el lado recurso cuando se le accede. Su principal función es para el "prefetching"; esta técnica consiste en actualizar referencias antes de que intenten realizar accesos.

El *cache live***,** guarda cuartetas de la forma $< A_{id}, i, lt, t_{stop} >$, donde A_{id} es un identificador de recurso, i es el sitio donde se encuentran referencias en búsqueda del recurso, lt es el último tiempo lógico conocido del recurso y t_{stop} es el tiempo físico durante el cual la entrada es vigente.

El *cache stop*, tiene como objetivo detener la migración de un recurso "rápido" hasta que

éste sea accedido por las referencias que lo están buscando. El tiempo físico t_{stop} acota la duración de ésta detención. Un recurso que intente migrar teniendo una entrada en este debe esperar hasta que el recurso sea accedido por el sitio *i* o hasta que expire el tiempo t_{stop}.

Con el fin de gestionar los caches, atender las peticiones de búsqueda, y mantener el estado de control de localización, cada sitio del sistema debe poseer un ejecutivo, tipo servidor o demonio, con las siguientes características:

- Capacidad de interceptación de mensajes de comunicación: cualquier mensaje de comunicación inherente al sistema, entrante o saliente, en particular los que concierna accesos a recursos, es interceptado y eventualmente inspeccionado, modificado o redirigido.

- Exportación de interfaz local a servicios de localización: el ejecutivo, en forma de servidor por sitio, ofrece servicios de registro, localización y de registro.

- Capacidad de intercambio de mensajes de control de localización entre otros localizadores.

Estas características son fácilmente implantables en cualquier arquitectura de sistema conocida. Al ejecutivo que implanta estas características se le denomina "localizador".

Un recurso móvil *A* asocia tres atributos de localización $<A_{id}, i,lt>$. Donde A_{id} es un identificador de recurso único en tiempo y espacio; *i* es el número del sitio donde reside el recurso y *lt* es una estampilla de tiempo lógico correspondiente al número de veces que ha migrado el recurso.

Una referencia también asocia un identificador de referencia único R_{id} , así como su sitio de residencia *j* de este modo, una referencia se interpreta como la tripleta $<R_{id}, j, A_i>$, donde A_i es la tripleta de localización del recurso $<A_{id}, i,lt>$.

2.1.3 *PREFETCHING.*

Prefetching es el acto de enviar una nueva localización hacia un sitio a efectos de prevenir que un acceso sea fallido. Existen dos eventos que causan *prefetching*. El primero ocurre cuando un recurso A_{id} migra desde el sitio i hacia un sitio j; en este caso, se toma un número máximo de entradas del *cache arriving* del sitio *i* que referencia el recurso A_{id}. Las entradas del *cache arriving* tienen una vigencia expresada en tiempo físico *t* después del último acceso; para cada sitio conteniendo una entrada vigente, se le envía un mensaje a efectos de que este se actualice en su *cache live*.

La segunda forma de *prefetching* ocurre periódicamente en cada sitio mediante revisión de su *cache arriving*. En este caso, se revisan *n* recursos con una vigencia física y se envía un mensaje de actualización a los sitios referenciantes. Tanto n como la vigencia física son

parámetros del sistema. A efectos de que no haya alta congestión de mensajes de red por el hecho de que varios sitios envíen simultáneamente mensajes de actualización, es importante que la periodicidad del *prefetching* este intercalada entre los sitios. Por esta razón, el periodo se determina según una distribución exponencial.

2.1.4 *PIGGYBACK.*

Esta técnica consiste en aprovechar implícitamente un mensaje explícito para enviar otro mensaje; el mensaje contendría el mensaje explícito, más una sección destinada al *piggyback*. La idea es aprovechar el envío de un mensaje explícito para transportar, a bajo costo, uno o más mensajes de control de localización.

La mayoría de los protocolos de red de segunda, tercera y cuarta capa cuantifican el tamaño máximo de paquetes en una cifra comúnmente llamada MTU[6]. Según el tipo de red, configuración y circunstancias de desempeño, puede o no ser conveniente llenar el MTU[7]

El sentido de los *piggybacks* para localización es propagar información de actualización y búsqueda. Desde la perspectiva de un sitio, sólo se propagaría la información concerniente a sus sitios de interés; es decir, respecto a los sitios con los cuales se interactúa.

Aparte de la información concerniente al control de localización, la trama de un *piggyback* contiene los siguientes atributos adicionales: tipo de *piggyback*, tiempo físico de creación (*tc*), sitio de creación (*src*) y tiempo lógico del *piggyback* (*plt*).

Los *piggybacks* se procesan exactamente de la misma forma que su correspondiente mensaje explícito; el localizador es el encargado de su gestión.

El localizador revisa todo mensaje entrante y extrae sus *piggybacks*; análogamente, antes de partir un mensaje, el localizador selecciona un conjunto de *piggybacks* a adjuntar al mensaje.

2.1.4.1 Prioridades de los *piggybacks*. Es posible que dos o más *piggybacks* distintos, pero referentes al mismo recurso, lleguen a un localizador. Para decidir cual de ellos debe procesarse, se mantiene una prioridad según el siguiente orden: *resource-dead-pb, stop-resource-pb, resource-search-pb, resource-found-pb, resource-update-pb.*

2.1.4.2 Gestión de un *piggyback*. Cada localizador maneja cinco (5) colas finitas de *piggybacks*, una por cada tipo, ordenadas según su inserción; es decir, el primer elemento de la lista corresponde al más reciente insertado. La longitud máxima de la cola es parametrizable y se denomina *mpq*.

---

[6] Unidad Máxima de Transferencia

[7] Algunos sistemas o protocolos de capas superiores esperan a que hayan más mensajes para colapsarlos en un solo paquete de tamaño próximo al MTU

Cada elemento de una de las colas anteriores se estructura como sigue:

Atributo	Descripción
Piggyback	*Piggybac*k recibido o creado
sl	Lista de sitios a los cuales se ha enviado
rl	Lista de sitios desde los cuales se ha recibido

Tabla 1. Estructura de una cola del *piggyback*

Las listas *sl* y *rl* pemiten evitar, en un primer nivel, que se envíe un piggyback hacia un sitio donde ya es conocido. Su longitud máxima es parametrizable de modo tal que los recursos que ocupen sean acotados.

Una inserción en una cola de piggyback ocurre cuando se recibe remotamente el piggyback, o cuando el localizador lo crea en función de álgun evento. Existe un número máximo de piggybacks llamado *máx_p*, parametrizable, que puede transportar un mensaje explícito.

2.1.4.3 Control de congestión de *piggybacks*. Los *piggybacks* ocupan recurso de red, pues aumentan el tamaño de los mensajes explícitos. Por otra parte, también consumen recursos como cpu y memoria en cada sitio. Se requieren entonces, considerar métodos para acotar su duración dentro del sistema. La adopción de tales métodos representa un compromiso; mientras más tiempo permanezca un *piggyback* dentro del sitema más posibilidades habrá de refrescar pero más recursos se consumirán. Contrariamente , mientras menos tiempo dure un *piggyback* en el sistema, menos recursos consume pero menos oportunidades tiene de refrescar los caches.

A los anterior hay que considerar que un *piggyback* en el sistema en cierto modo compite con otros *piggybacks*.

Hay tres maneras de acotar los piggybacks; la primera, ya mencionada, la compone el límite *mpq* sobre la longitud máxima de cada cola; la segunda consiste en acotar la cantidad de sitios por los cuales puede circular un piggyback. Este es el sentido del tiempo lógico *plt*, el cual representa el número de sitios que se han recorrido. Cuando parte un *piggyback*, se incrementa el contador. El sitio que lo recibe lo procesa y si el contador ha alcanzado el máximo, entonces se elimina del localizador. Esta estampilla representa una cadena parcial de circulación del piggyback, pero no el total de sitios en los cuales éste se conoce.

El máximo tiempo lógico de un *piggyback* es un parámetro del sistema y se denomina *mplt*.

La última forma de acotamiento la constituye la duración máxima de un *piggyback* dentro de un localizador, esta duración se denomina *mct*, y se contrasta con el atributo *ct* de cada *piggyback*. Si *t* es el tiempo físico actual de un localizador y $t >= ct + mct$, entonces el *piggyback* se elimina del localizador.

Envios de *piggybcks*.

Cuando un localizador envía un mensaje cualquiera hacia el sitio *i*, éste revisa cada cola según el orden de prioridad para cargar los *max_p* "primeros *piggyback*s". La revisión inicia desde la cola *resource-dead-pb* hasta la cola *resource-update-pb* según su prioridad. Cada cola se revisa secuencialmente para buscar piggybacks que no se hayan enviado hacia ni recibido desde *i*: esto se determina mediante las listas *s*l y *rl*, repectivamente. Este proceso de revisión se detiene cuando se han cargado *max_p* piggybacks o cuando se han revisado enteramente todas las colas.

Recepción de piggybacks.

Cuando se carga un *piggyback* para envío, éste se puede rotar; esto consiste en colocar el piggyback al principio o al final de la cola. Colocarlo al principio aumenta la posibilidad de que el piggyback se propague más rápidamente, pero tiende a dejar de procesar otros *piggybacks* que quizá sean más importantes. Colocarlo al final favorece la propagación de otros *piggyback*, pero aumenta la vida del *piggyback* en el sistema y el consumo de recursos; además, si la lista ya ha alcanzado su longitud máxima, entonces el *piggyback* recien cargado será eliminado.

2.2 CARACTERIZACIONES DE UNA FALLA DE ACCESO

A efectos de facilitar la comprensión se han caracterízado las siguientes fallas que pueden suceder cuando se intenta acceder a un recurso *Aid.*

2.2.1 Fallas Transparentes (F_i). Esta falla ocurre cuando el localizador emisor del acceso encuentra una referencia más reciente en el *cache live* y redirige transparentemente el acceso hacia la nueva localización.

Una falla transparente cuyo acceso es exitoso no acarrea ningún costo adicional de comunicación.

2.2.2 Fallas Simples (Fs_i). Esta falla ocurre cuando el localizador emisor del acceso recibe como respuesta un *reply-update* conteniendo una localización más reciente. En este caso, el localizador transparentemente repite el acceso redirigido hacia la nueva localidad dada por el *reply-update*.

El sistema puede reintentar un acceso hasta un máximo n_{stop} veces. Después de n_{stop} intentos se declara una falla notable. Cualquier falla simple cuesta *i* mensajes, donde $i <= n_{stop}$ es la cantidad de reintentos realizados.

Dada una referencia, Fs_i, $i <= n_{stop}$ es la cantidad de fallas simples que ha tenido la referencia en el *i*-ésimo intento.

Las fallas están ordenadas según su costo para el sistema. Una falla transparente, que no tiene costos comunicacionales, es la menos costosa. Una falla simple cuesta en reintentos

de acceso. La falla notable es la más costosa, pues aparte de los reintentos de acceso, requiere emprender una búsqueda a través de todo el sistema.

2.2.3 Fallas Notables (Fn_i). Esta falla puede ocurrir por dos razones. La primera es por expiración de los n_{stop} intentos de acceso de una falla simple. La segunda es que el emisor del acceso reciba un *reply-not-found.* Toda falla notable requiere una solicitud de búsqueda al localizador. La búsqueda arroja una nueva localidad con la cual se repetirá el intento de acceso. Este intento puede ser exitoso o devenir en cualquiera de las fallas presentadas.

Dada una referencia, se define Fn_i con la cantidad de fallas notables que ha tenido una referencia en el *i*-ésimo intento seguido sin lograr acceder el recurso. La recurrencia de una falla notable es indeseable en el sentido que refleja una especie de "*thrashing"* en el sistema.

2.3 BÚSQUEDA DE RECURSOS.

Como se indico en §2.2.3, una falla notable requiere solicitar una búsqueda al localizador. Para esto, se le trasmite al localizador el *Aid* del recurso y su último tiempo lógico conocido *lt*. Este par permite a cualquier localizador determinar si una localidad conocida es más reciente o no.

2.3.1 Modos de Búsqueda. Según se solicite o no la detención del recurso (mediante el stop) existen dos modos de búsqueda que determinan el procesamiento en el localizador de una solicitud de búsqueda.

Búsqueda sin parada.

Una búsqueda sin parada realiza difusiones débiles *de resource-search o piggybacks reource-search-pb*. Su procesamiento en cualquier localizador receptor se puede observar en la figura 3.

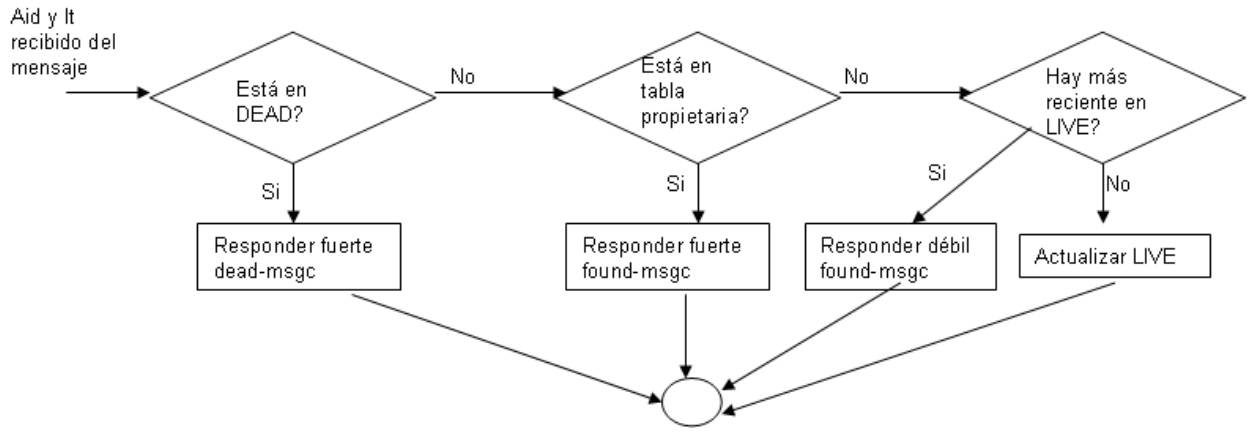


Figura 3. Búsqueda sin parada

Esta búsqueda se emprende con mensajes débiles *resource-search* o con *piggybacks resource.search-pb*. Por esto, a la excepción del localizador propietario, la respuesta es débil, pues se presume que si llegó el mensaje débil o el *piggyback*, entonces es factible que la respuesta tambien llegará.

Una búsqueda sin parada puede tener varias respuesta provenientes de los sitios que encontraron referencias más reciente en su *cahe live*. Esto conlleva la posibilidad de esperar un poco antes de reintentar el acceso de tal manera que se seleccione la localidad más vigente. Si el recurso se mueve durante la espera, entonces ocurrirá, al menos, una falla simple. El tiempo de espera antes de reintentar el acceso se denomina t_r que es un parámetro del sistema que se seleccionaría de acuerdo a sus características; esto es, si el recurso es considerados lento o rápido.

Búsqueda con parada.

Una búsqueda con parada usa difusiones fuertes de *resource-stop o piggybacks resource-stop-pb.* Su procesamiento en cualquier sitio receptor se ilustra en la figura 4.

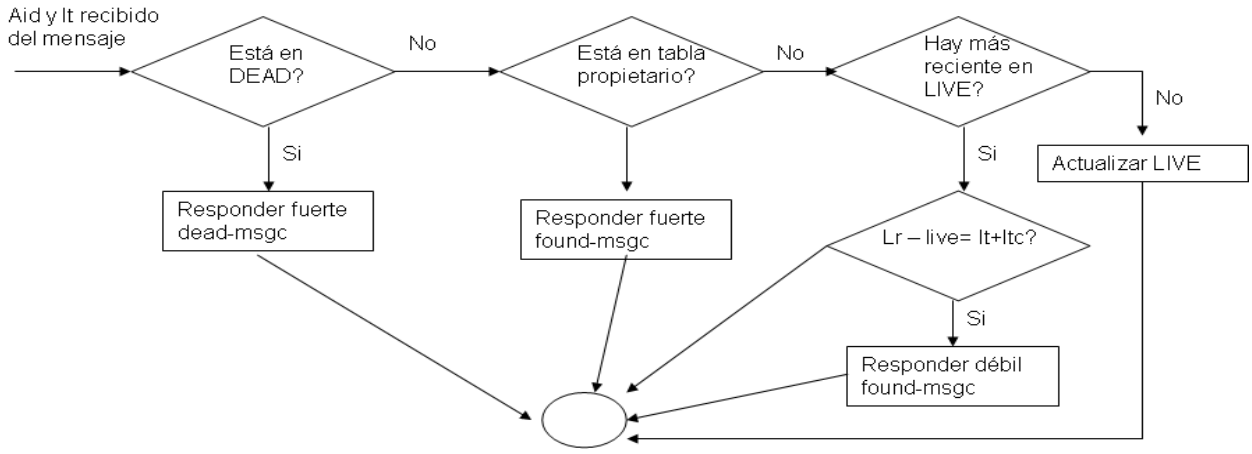


Figura 4. Búsqueda con parada

La diferencia notable con la búsqueda sin parada es la semántica de respuesta del *resource-found*. Si se encuentra una localidad más reciente en el *cache live*, entonce, se responde sólo si ésta es más reciente que el *lt+ltc*; de lo contrario no responde.

La búsqueda con parada sólo debe emplearse para resolver fallas notables recurrentes; es decir, que han ocurrido varias veces seguidas sin poder acceder al recurso. En este sentido, se efecturá una búsqueda con parada si ocurre *ltc* fallas notables. Esto es, tiempos lógicos menores a *lt+ltc* serían aún antiguos. Además, a diferencia de la búsqueda sin parada, en este caso se responde con un *resource-found* fuerte.

2.3.2 Tipos de Búsqueda.

La emisión de una solicitud de búsqueda se tipifica según el tipo de mensajes de búsqueda y su modo de propagación a través del sistema (*piggybacks*, difusión débil o difusión fuerte).

Búsqueda Explícita.

Este tipo de búsqueda propaga por el sistema *piggybacks* de solicitud de búsqueda. Si no han ocurrido n_{stop} fallas notables continuas entonces se propaga *resource-search-pb*, de contrario, propaga *stop-resource-pb*.

A la excepción de los mensajes resource-found, una búsqueda implícita no tiene coste en mensajes adicionales.

Búsqueda Débil.

Esta búsqueda realiza una difusión débil de *resource-search* o *resource-stop* según que la búsqueda sea de parada o no. Una búsqueda débil con parada se activa despues de nstop intentos fútiles de búsqueda sin parada.

Búsqueda Fuerte.

Esta búsqueda difunde fuertemente un *resource-stop*. Como tal, es muy costosa, pero desembocará en una actualización efectiva o en la conclusión de que el recurso ya no existe.

Para no perder una búsqueda fuerte debido a otra migración de recursos, la duración de la parada t_{stop} en el *stop* debe ser lo suficientemente grande como para impedir la migración.

2.4 MIGRACIÓN DE UN RECURSO.

La migración de un recurso A_{id} de un sitio origen a uno destino se procesa en cada sitio, según se observa en la figura 5.

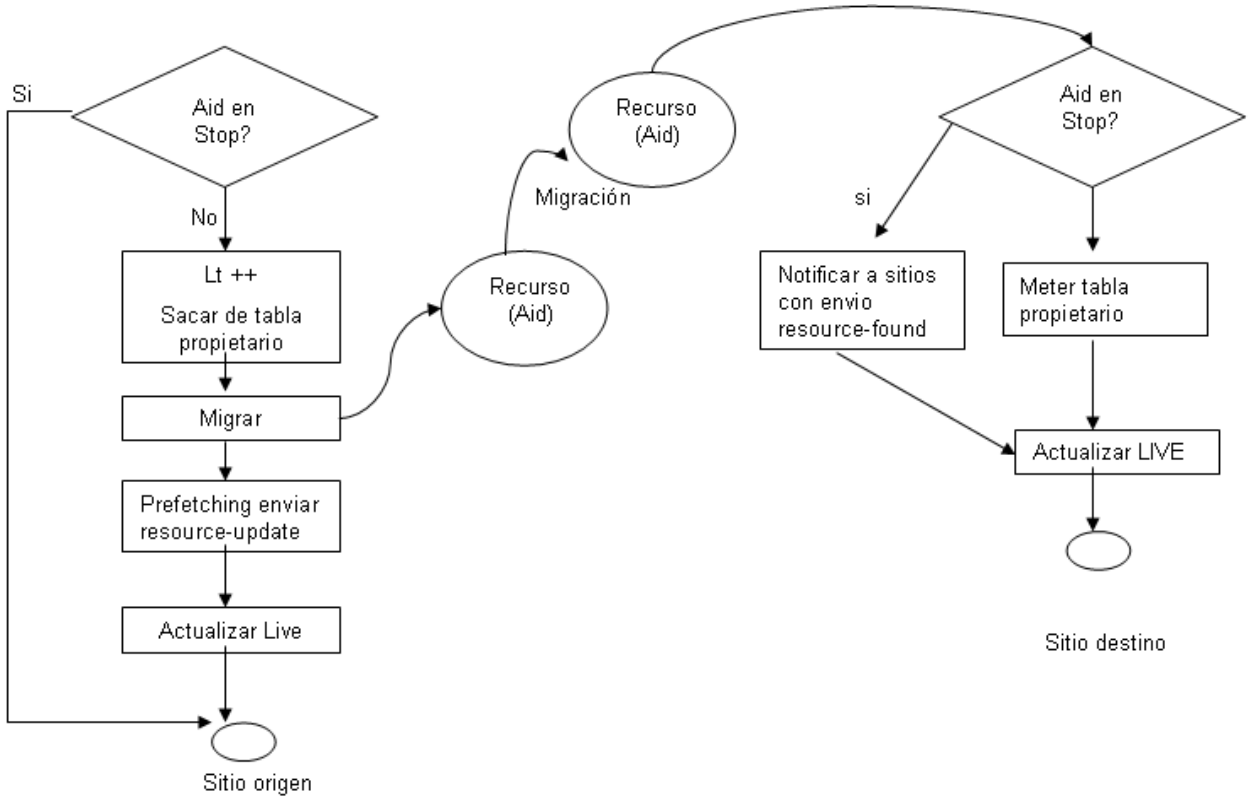


Figura 5. Migración de un recurso

Si el recurso se encuentra en s*top* en el sitio origen, entonces se aborta la migración, debido a que hay referencias pendientes por accederlo.

Si en el sitio destino hay una entrada s*top*, entonces se búscan los sitios asociados a esta entrada y se le notifica con un mensaje *resource-found*

3 MODELO

El modelo de simulación a eventos discretos representa las condiciones de un sistema de localización de recursos; abstrae la noción de sitio ("*host*"), el cual representa un sitio conectado a red, que contiene referencias y recursos, que accede y localiza recursos mediante las técnicas definida[8].

Cada sitio identifica los objetos y funcionalidades correspondientes a las técnicas referenciadas y puede configurarse independientemente. Por ejemplo, dos sitios pueden tener distintos tiempos de llegada de objetos, frecuencia de acceso y tamaño de caches, entre otros posibles parámetros. Cada sitio contabiliza un conjunto de métricas del sistema de nombre *num_hots*.

3.1 CREACIÓN DE UN RECURSO

Los recursos son creados en cada sitio según una distribución exponencial con media *obj_creation_iat*. Cuando éste es creado, se le instancia *num_ref_by_obj* referencias iniciales y son repartidas uniformemente entre los sitios del sistema.

A partir del momento de creación de un recurso, se crean otras nuevas referencias dinámicas según una distribución exponencial con media *ref_creation_ia*t. El sitio de la nueva referencia se escoge aleatoriamente entre los sitios; mientras el recurso exista en el sistema, se le siguen creando referencias dinámicas.

Cada referencia intenta accesos según un tiempo exponencial con parámetros *acces_time*. Los intentos de acceso se realizan independientemente de los anteriores.

Un recurso migra cada cierto tiempo según una distribución exponencial con media *migration_time*.

Un recurso tiene una duración *obj_duration,* el cual al finalizar este es eliminado del sistema; el tiempo de duración es exponencial.

[8] Técnicas de localización del recurso: difusión, *piggyback*, *prefetching* y caches

3.2. COMUNICACIÓN.

La comunicación se realiza a través de los mensajes fuertes y débiles; los mensajes fuertes son aquellos en la cual se asegura la entrega de los mismos con un costo de red significativo, mientras que los mensajes débiles pueden perderse pero los costos de red son muy bajos.

La transmisión de los mensajes débiles se somete a una distribución uniforme de media *lost_rate*, el cual modela la probabilidad de pérdida, mientras que la latencia de transmisión, *latency,* es una media de distribución exponencial.

El consumo de ancho de banda debido a la longitud de los mensajes no está modelado debido a que no es parte de este estudio, el cual solo corresponde la localización del recurso. Sin embargo, los costos de red se miden en cantidad de mensajes circulantes, usandose las siguientes variables de respuesta: un mensaje débil perdido se contabiliza en *Total_unreliable_lost_msg*, mientras que uno exitoso en *Total_unreliable_sent_msg.*

En la tabla 2 se definen las variables de respuesta referentes a las comunicación.

Variable de respuesta	Descripción
total_reliable_sent_msg	Mensajes fuertes enviados
total_reliable_received_msg	Mensajes fuertes recibidos
total_unreliable_sent_msg	Mensajes débiles enviados
total_unreliable_received_msg	Mensajes débiles recibidos
total_unreliable_lost_msg	Mensajes débiles perdidos
total_reliable_broadcast_sent	Difusiones fuertes enviadas
total_reliable_broadcast_received	Difusiones fuertes recibidas
total_unreliable_broadcast_sent	Difusiones débiles enviadas
total_unreliable_broadcast_lost	Difusiones débiles perdidas
total_unreliable_broadcast_received	Difusiones débiles recibidas

Tabla 2. Variables de respuesta referentes a la comunicación

Una difusión cuenta como *num_host – 1 ;* esto es, un mensaje a cada sitio a excepción del emisor.

Tipos de Mensajes

El sistema maneja mensajes que tiene que ver con el acceso a un recurso y con mensajes de control de localización.

En los mensajes de acceso al recurso, el acceso se realiza bajo el modelo *request/reply* usado en los paradigmas cliente/servidor, RPC, invocación o métodos remotos (RMI) o

llamadas a agentes. Es así como en el modelo el intercambio *request/reply* se maneja usando los mensajes definidos en la tabla 3.

Mensaje	Parámetros	Descripción
request $<A_{id}, R_{id}, lt>$	**A_{id}**: identificador del recurso **R_{id}**: identificador de la referencia **lt**: último tiempo lógico conocido del recurso *Aid*	se usa cuando se requiere acceder a un recurso remoto
Reply $<A_{id}, R_{id}, lt>$	**A_{id}**: identificador del recurso **R_{id}**: identificador de la referencia **lt**: último tiempo lógico conocido del recurso *Aid*	respuesta a *request* que indica que el recurso fue normalmente accedido
Reply-update $<A_{id}, i, R_{id}, lt>$	**A_{id}**: identificador del recurso i : identificador de la nueva localización **R_{id}**: identificador de la referencia **lt**: último tiempo lógico conocido del recurso *Aid*	respuesta a *request* que indica que el recurso no se encuentra en el sitio pero se conoce un paradero *i* con un tiempo lógico mas reciente
Reply-not-foud $<A_{id},R_{id}>$	**A_{id}**: identificador del recurso **R_{id}**: identificador de la referencia	respuesta a *request* que indica que el recurso no se encuentra en el sitio y no se sabe su localización
Reply-dead $<A_{id},R_{id}>$	**A_{id}**: identificador del recurso **R_{id}**: identificador de la referencia	Respuesta a *request* que indica que el recurso fue eliminado del sistema.

Tabla 3. Tipos de mensajes de acceso al recurso

Igualmente para el control de localización el sistema usa los mensajes definidos en la tabla4.

Mensaje	Parámetros	Descripción
resource-search<A_{id}, *i, lt*>	**A_{id}**: identificador del recurso **R_{id}**: identificador de la referencia **lt**: último tiempo lógico conocido del recurso A_{id}	indica que al menos una referencia en el sitio *i* busca el recurso A_{id} cuyo último tiempo lógico es *lt*
resource-stop<A_{id}, *i, lt*>	**A_{id}**: identificador del recurso **i** : identificador de la nueva localización **lt**: último tiempo lógico conocido del recurso A_{id}	indica que al menos una referencia en el sitio *i* busca el recurso A_{id} cuyo último tiempo lógico es *lt* y solicita que el recurso sea parado durante una duración finita y parametrizable.
resource-found<A_{id}, *i,* lt_j>	**A_{id}**: identificador del recurso **i** : identificador de la nueva localización **lt_j :** último tiempo lógico mas reciente conocido del recurso A_{id}	respuesta a un *resource-search o resource-stop*, informando que se conoce un tiempo lógico *l*t mas reciente para el recurso A_{id} en un sitio *i*
resource-update<A_{id}, *i,* lt_j>	**A_{id}**: identificador del recurso **i** : identificador de la nueva localización **lt_j :** último tiempo lógico mas reciente conocido del recurso A_{id}	mensaje asincrónico notificando una actualización de localización del recurso
resource-dead<A_{id}, *i,* lt_j>	**A_{id}**: identificador del recurso **i** : identificador de la nueva localización **lt_j :** último tiempo lógico mas reciente conocido del recurso A_{id}	respuesta a un *resource-search o resource-stop*, informando que el recurso ya no existe

Tabla 4. Tipos de mensajes de control de localización.

3.3 FLUJO DE ACCESO

Para acceder a un recurso se requiere una referencia que contiene las coordenadas del recurso. Cuando se realiza el acceso se envía un *request* hacia el sitio donde se supone debe estar el recurso; si el recurso se encuentra en este sitio se hace el acceso y se responde

con un *reply.*

Todos los mensajes del sistema, salientes del propio sitio y remotos provenientes de otros localizadores, son interceptados en el localizador. Este tipo de esquema no impacta el costo de comunicación debido a la gran diferencia entre la latencia de un mensaje local y uno de red [3].

3.3.1 Emisión de un *request.*

En la figura 6 se puede ver el proceso de emisión de un *request* al recurso A_{id}.

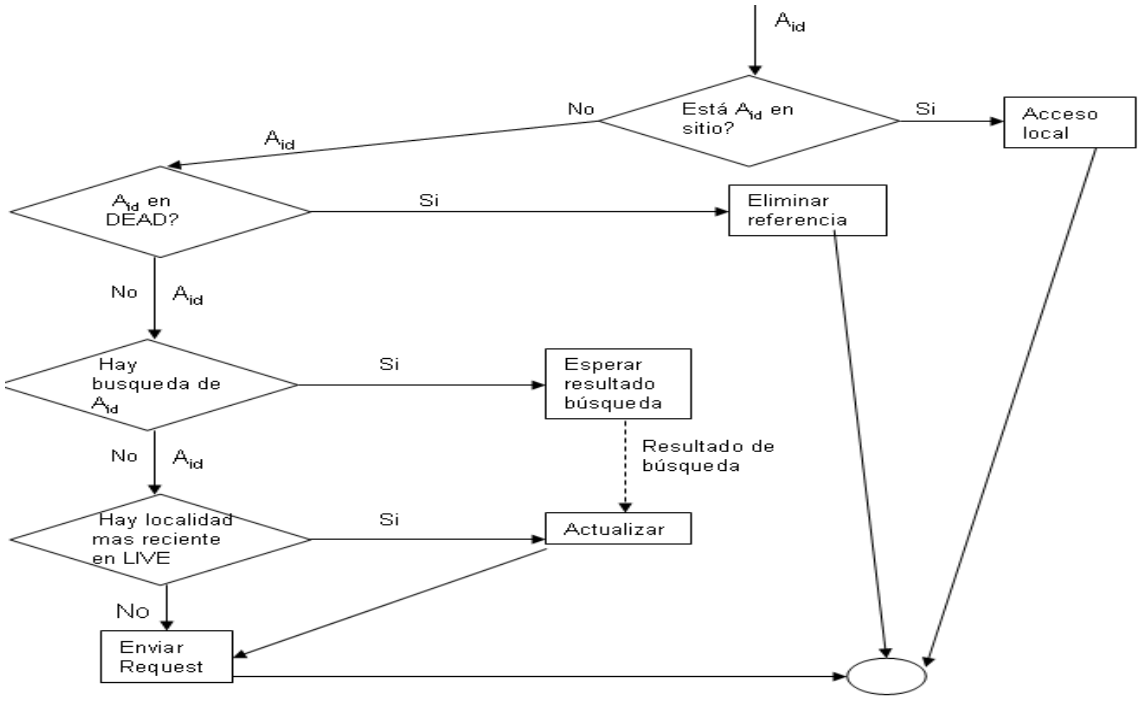


Figura 6. Proceso de emisión de un *request*

Se debe tener en cuenta que la inspección del *cache live* puede hacer que el *request* se actualiza transparentemente con una dirección mas reciente, y por tanto redirigirse a una localidad diferente a la guardada en la referencia. Igualmente, la inspección en la tabla de búsqueda puede detectar que el recurso A_{id} se está buscando; cuando esto ocurre, el *request* se retarda hasta que llegue una nueva localidad.

3.3.2 Recepción de un *request.*

Cuando llega un *reques*t $<A_{id}, R_{id}, lt >$, el localizador realiza el proceso definido en la figura 7.

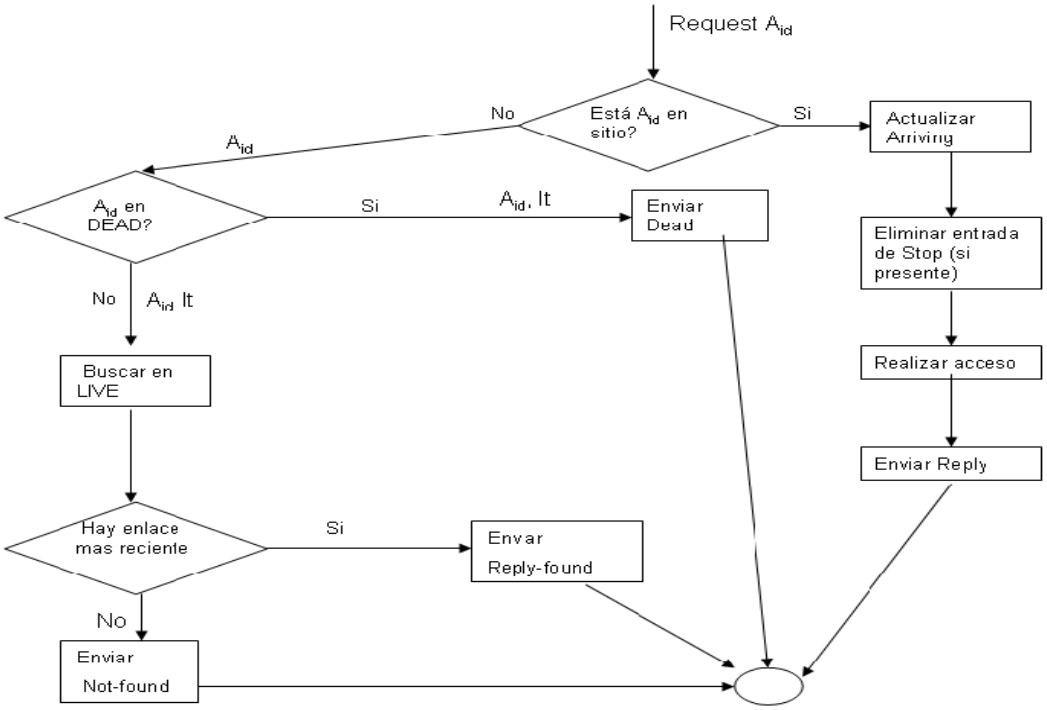


Figura 7. Proceso de recepción de un *request*

3.3.3 Recepción de respuesta a un *request.*

La recepción de un *request* puede generar cuatro diferentes tipos de respuestas; esto se puede analizar en la figura 8.

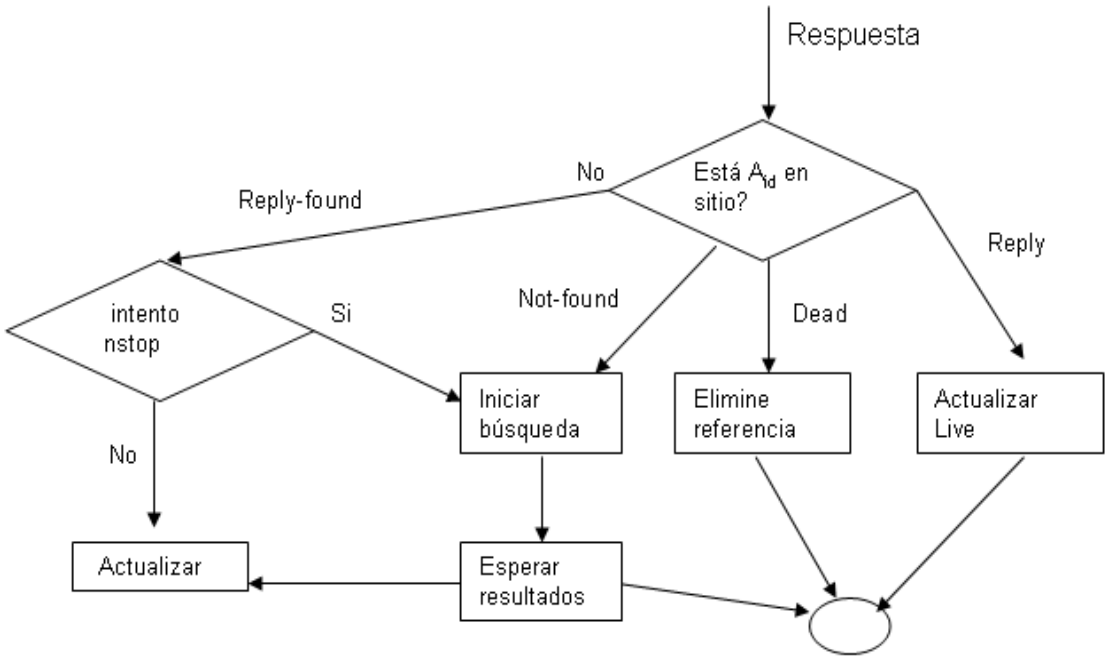


Figura 8. Proceso de recepción de respuesta a un *request*

3.4. CACHES

Respecto a los caches, estos son modelados como en una implantación real: una tabla asociativa, rápida, de tamaño finito, con una política de reemplazo LRU. El único factor que concierne a los caches es su tamaño *cache_size.*

Las variables de respuesta de cada cache de un sitio son: total de inserciones, *ins* y total de hits, *hits*.

3.5. *PREFETCHING.*

Como ya se determinó anteriormente, hay dos momentos para el *prefetching*: periódicamente y durante la migración. En la tabla 4 se enumera los diferentes factores según la discriminación anterior.

Factor	Descripción	Niveles
pre_type	Tipo de *prefetching* periódico	- sin *prefetching* - *prefetching* débil - *prefetching* fuerte
pre_iat	Intervalo exponencial entre *prefetching*	
pre_duration	Vigencia física de un *prefetching*	
pre_max	Máxima cantidad de *prefetchings* en una onda	
pre_mig_type	Tipo de *prefetching* en migración	- sin *prefetching* - *prefetching* débil - *prefetching* fuerte
pre_mig_duration	Vigencia física de *prefetchin* en migración	
pre_mig_max	Máxima cantidad de *prefetching* en migración	

Tabla 5 Factores referentes al *prefetching*

Los factores *pre-type* y *pre_mig_type* permiten estudiar el sistema sin alguna de las acciones de *prefetching*. La única variable de respuesta concerniente al *prefetching* es la cantidad de *prefetching* enviados desde el sitio.

3.6. INVOCACIONES O ACCESOS.

Al momento que nace un recurso, se crean *n* referencias iniciales según una distribución de *Poisson* con parámetros λ. Con la excepción del sitio de nacimiento del recurso, las referencias iniciales se reparten uniformemente a través de los sitios; cada referencia efectúa accesos a su recurso según una distribución exponencial con parámetro a_2.

El tiempo de vida de una referencia es el doble del tiempo de vida de un recurso, permitiendo modelar y verificar que las referencias se percaten de la muerte del mismo.

Tanto el tiempo de vida de un recurso, como el tiempo de vida de una referencia es exponencial, con un parámetro a_3, y $2a_3$ respectivamente.

La cantidad de accesos que se realizan a un recurso se caracteriza por :

$$n_i = P(1/a2)\ exp(a3)$$

donde P(1/a2) es una distribución de Poisson.

En promedio se tiene entonces: $\overline{n_i} = a_3 / a_2$, accesos por referencia.

El número promedio de accesos causados por las diferencias creadas inicialmente se caracteriza por:

$$\overline{n_i} = a_3 \lambda / a_2 \qquad (1)$$

A lo largo de vida del recurso se crean referencias dinámicas, cuyos tiempos de nacimiento obedecen a una distribución exponencial con parámetro a_4. Sean t_c y t_x los tiempos de creación y muerte de un recurso en el sistema tal que $t_x - t_c = exp(a_3)$; es decir, la duración del recurso dentro del sistema. Para la j-ésima referencia dinámica, la cantidad de accesos a su recurso está definida por: $n_j = Poisson(1/a_2)\ ((t_x - (tc + j\ exp(a_4)))$; cuyo promedio está dado por: $n_j = 1/a_2\ ((t_x - (tc + j\ exp(a_4))) = (a_3 - j\ a_4)/ a_2$.
El número esperado de accesos causados por las referencias dinámicas requerirían contar la cantidad de referencias dinámicas, la cual puede describirse como:

$$n_D = \sum_{\forall j \in N \mid a_3 - ja_4 > 0} (a_3 - j\ a_4) / a_2 \qquad (2)$$

Es así como el número total de accesos causados por un recurso estará dado por (1) + (2):

$$\overline{n_I} + \overline{n_D} = a_3\ \lambda / a_2 + \lambda T \sum_{\forall j \in N \mid a_3 - ja_4 > 0} (a_3 - j\ a_4)$$

$$= 1 / a_2 \left(a_3\ \lambda + \sum_{\forall j \in N \mid a_3 - ja_4 > 0} (a_3 - j\ a_4) \right)$$

Los recursos tienen un tiempo de llegada exponencial con parámetro a_5. Si T es el tiempo total de ejecución del sistema, entonces, la cantidad total esperada de acceso a recursos está dada por:

$$n_T = T / a_2 a_5 \left(a_3\ \lambda + \sum_{\forall j \in N \mid a_3 - ja_4 > 0} (a_3 - j\ a_4) \right) \qquad (3)$$

Esta cantidad representa una referencia de desempeño del sistema, en el sentido de que la

cantidad de invocaciones debe estar lo más cerca posible de este valor. Un modelo o implantación de las técnicas debe arrojar una cantidad de invocaciones, cuando no ocurra migración, aproximadamente a este valor.

3.7. MIGRACIONES.

Los recursos migran según una frecuencia exponencial con parámetro a6. A lo largo de la vida del recurso, la cantidad de migraciones estará dada por:

$$n_m = P(1/a_6)\, a_3;$$

cuyo valor esperado es, entonces:

$$\overline{n_m} = a_3 / a_6 .$$

Lo que, contabilizando la cantidad de recursos y el tiempo total de ejecución, arrojará la cantidad total esperada de migraciones siguientes:

$$\overline{n_m} = (T / a_5)\, (a_3 / a_6) \qquad (4)$$

4. DISEÑO Y ANÁLISIS DE EXPERIMENTOS

Para realizar el análisis de las técnicas de localización: difusión, *prefetching*, *cache* y *piggyback*, se diseñaron una serie de experimentos los cuales inicialmente permitieron identificar errores que tenía el modelo; los errores identificados tenían que ver con el calculo de algunas variables de respuesta, tales como el número de fallas, el número promedio de accesos, los cuales fueron corregidos para poder ejecutar los diferentes experimentos.

Corregido el modelo, se diseñaron experimentos los cuales se clasifican de acuerdo al objetivo planteado en cada uno de ellos; es así como se puede determinar los siguientes tipos de experimentos que permiten:

1. validar el modelo
2. determinar la eficiencia y el costo del uso de la técnica difusión
3. determinar la eficiencia y el costo del uso de los *cache arriving, live y dead*
4. determinar la eficiencia y el costo del uso del *cache stop*
5. determinar la eficiencia y el costo del uso de la técnica *piggyback*

4.1 DISEÑO Y ANÁLISIS DE EXPERIMENTOS PARA VALIDAR EL MODELO

Para validar el modelo se diseñan una serie de experimentos cuyo objetivo es determinar la confiabilidad del mismo. Se toma como base para el proceso de validación del modelo el tiempo de creación del objeto (*obj_creation_iat*) y el tiempo de duración del objeto en el sistema (*obj_duration_time*) y de diferentes valores aleatorios generados a partir de la ejecución de la simulación (archivo seed.ini).

Cada uno de los experimentos tiene la siguiente estructuración:

- Objetivo: donde se especifica lo que se pretende con el experimento
- Simulación realizada: aquí se determinan los niveles que tiene cada factor
- Variables de respuesta: son las variables que se analizarán a partir de los factores definidos.

4.1.1 Validación del modelo usando valores con y sin multiplicidad. Los experimentos 1 y 2 nos permiten validar el modelo a partir de los factores *obj_creation_iat* y *obj_duration_time* con multiplicidad de los valores usando los niveles alto y bajo.

- **Objetivo**: El objetivo de cada uno de los experimentos es determinar si el número de invocaciones sin migración que se realizan en el modelo en un tiempo de simulación *T*= 1600m, bajo determinados valores de los factores tiempo de vida de un recurso (a_3), tiempo de creación de un nuevo recurso en el sistema (a_5) y tiempo de creación de nueva referencia (a_4), corresponde al número de invocaciones esperadas.

  El tiempo de creación de la referencia (a_4), será 4m. y el número de referencias por recursos es 1.

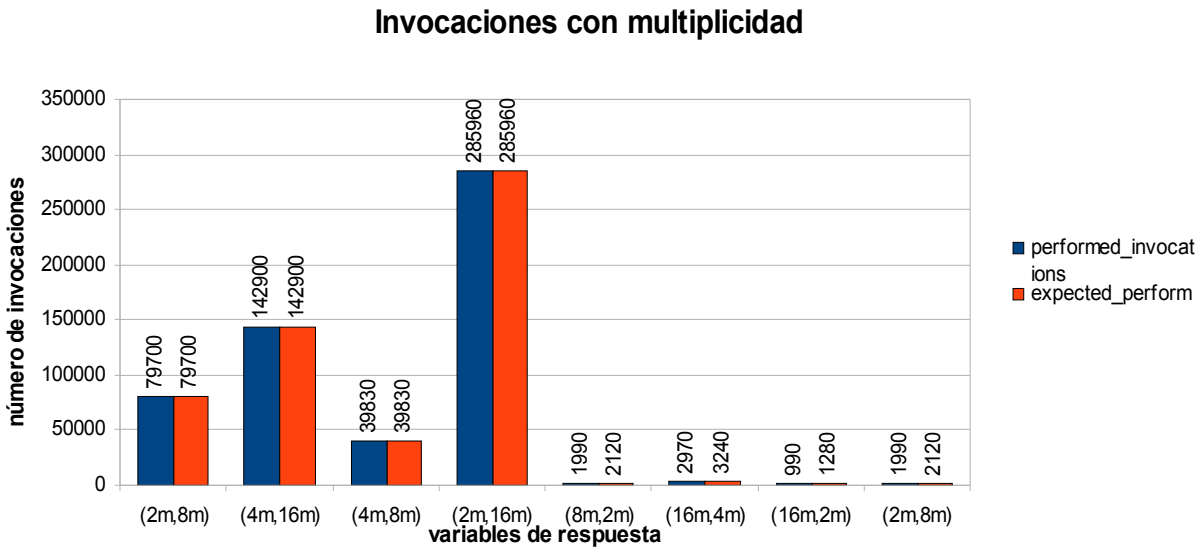
- **Simulación realizada**: Se analizará las posibles combinaciones de $a_3 < a_5$ y $a_3 > a_5$ con multiplicidad de los valores (ver Tabla 7) y sin multiplicidad[9] de los valores (ver tabla 8), en los niveles bajo y alto.

Multiplicidad		**Factor**			
		$a_3 < a_5$		**$a_3 > a_5$**	
		a_5	**a_3**	**a_5**	**a_3**
Con	bajo – bajo	2m	8m	8m	2m
	alto – alto	4m	16m	16m	4m
	alto - bajo	4m	8m	16m	2m
	bajo – alto	2m	16m	8m	4m
Sin	bajo – bajo	2m	5m	5m	2m
	alto – alto	3m	7m	7m	3m
	alto - bajo	3m	5m	7m	2m
	bajo – alto	2m	7m	5m	3m

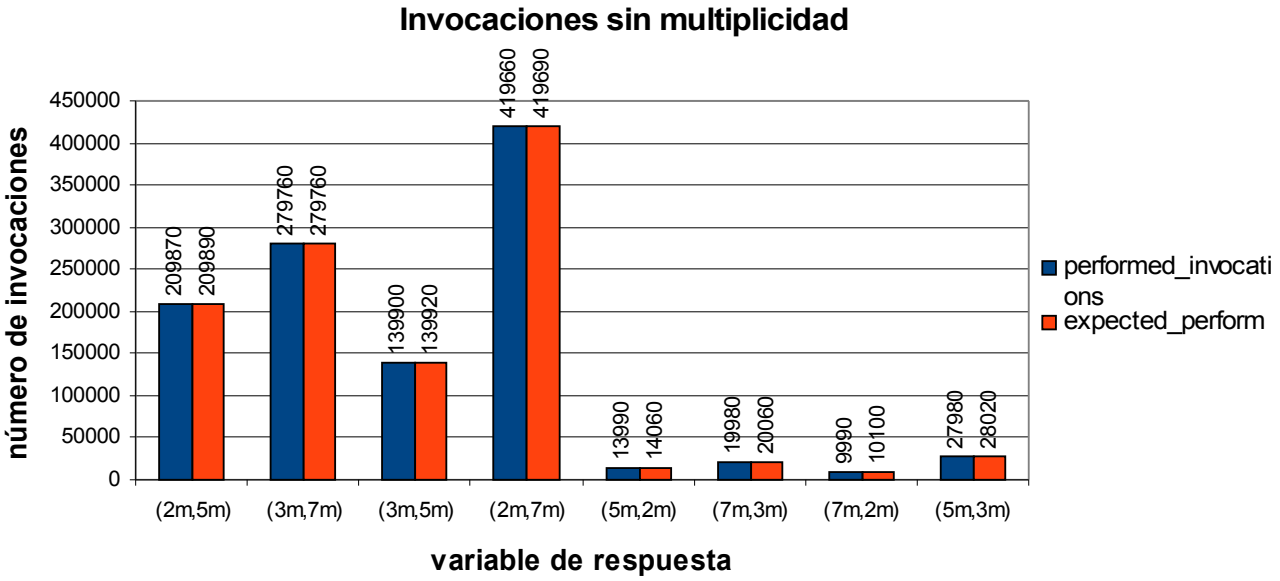
Tabla 7. Valores definidos para los factores a_3 y a_5

- **Variables de respuesta.** Ejecutados los programas experimento1.ini (Anexo 1) y experimento2.ini, (Anexo1)se determinan los valores de las variables de respuesta número de invocaciones ejecutada (*performed_invoctions*) y número de invocaciones esperadas (*expected_performed*), que se observan en las gráficas 1 y 2.

[9] La multiplicidad de un número está dado por los factores de descomposición primos del mismo.



Grafica 1. Invocaciones con multiplicidad de los factores



Grafica 2. Invocaciones sin multiplicidad de los factores

4.1.2 Validación del modelo usando valores con multiplicidad y variando los valores aleatorios. El experimento 3 permite validar el modelo a partir de los factores *obj_creation_iat* y *obj_duration_time* con multiplicidad de los factores usando los niveles alto y bajo, y variando los valores aleatorios que usa el mismo para efectuar las simulaciones.

- **Objetivo** : El objetivo del experimento es determinar si existe la necesidad de realizar replicas de cada uno de ellos para tener mayor certeza en los resultados generados.

- **Simulación a realizar:** Se realizaron simulaciones en un tiempo de simulacion T= 16000m, bajo determinados valores de los factores tiempo de vida de un recurso (a3), tiempo de creación de un nuevo recurso en el sistema (a5), tiempo de creación de nueva referencia (a4) y usando ciertos valores aleatorios. En la tabla 8 se puede observar los valores dados a a_3 y a_5.

  El tiempo de creación de la referencia (a4) será 4m, y el número de referencias por recursos es 1.

  Valores Aleatorios definidos en el archivo seed.ini

  - **Grupo aleatorio1 (E1):** 1768507984 33648008 1082809519
    703931312 1856610745 784675296
    426676692 1100642647
  - **Grupo aleatorio2 (E2):** 72358415 271666411 1281145440 991747913
    137375739 614387732 69416058 194900596
  - **Grupo aleatorio3 (E3):** 1486252914 1187433110 1820194104
    898589432 725502823 1428095551
    424107619 1845654421

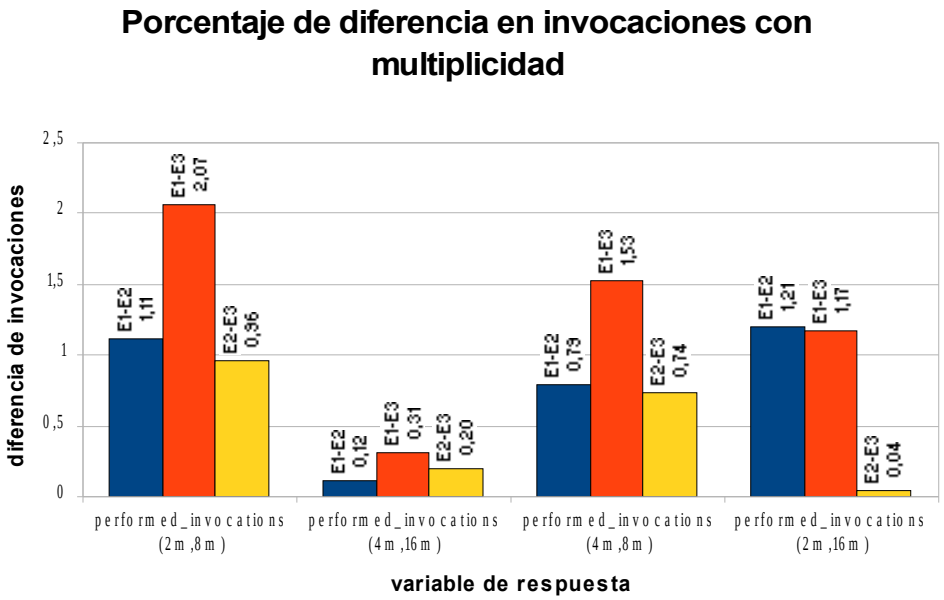
Multiplicidad		**Factor**	
		$a_3 < a_5$	
		a_5	**a_3**
con	Bajo – bajo	2m	8m
	Alto – alto	4m	16m
	Alto – bajo	4m	8m
	Bajo – alto	2m	16m

Tabla 8. Valores definidos para los factores a_3 y a_5 con multiplicidad

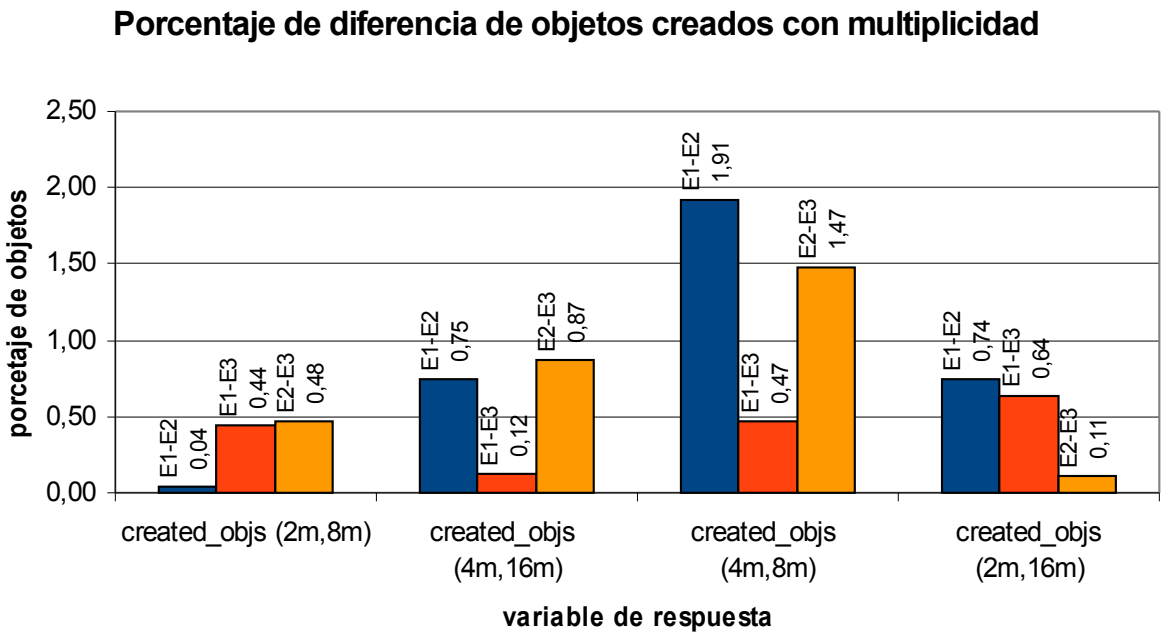
- **Variables de respuesta**. Ejecutado el programas experimento3.ini (Anexo 1) con los diferentes grupos de valores definidos en el archivo seed.ini, se determinan las diferencias entre los diferentes experimentos para las variables de respuesta :

Totals_performed_invocations	(total de invocaciones realizadas)
Totals_expected_performed_invocations	(total de invocaciones esperadas a ser ejecutadas)
Total_created_objs	(total de objetos creados)
Total_created_references	(total de referencias creadas)

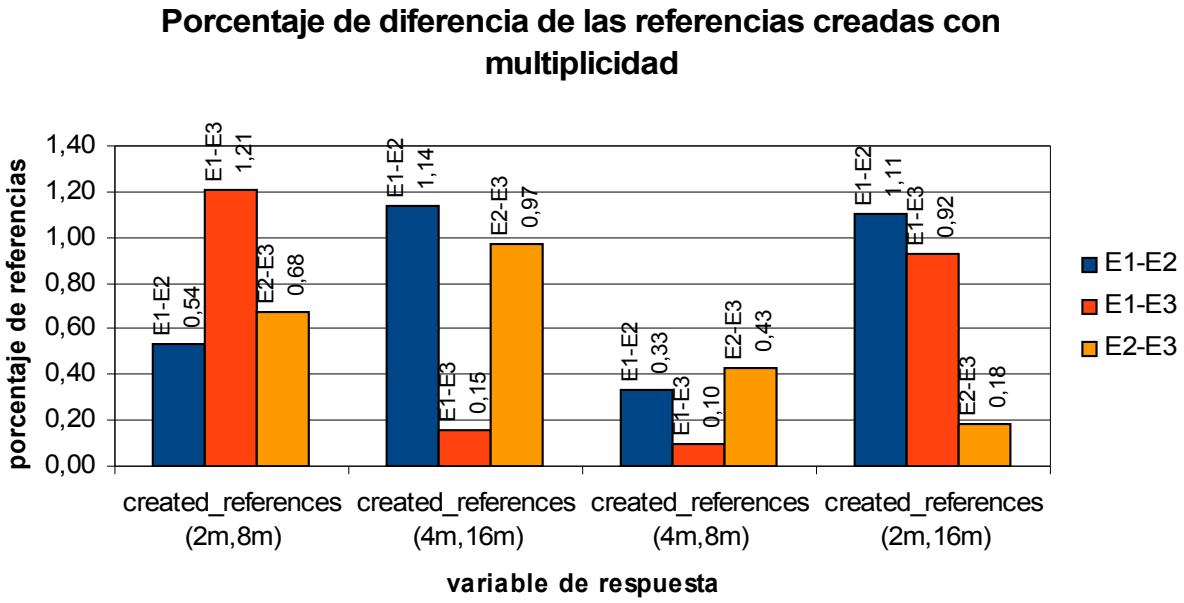
En las gráficas 3,4 y 5 se observa el comportamiento de las variables de respuesta



Gráfica 3. Porcentaje de diferencia en las invocaciones usando valores con multiplicidad de los factores



Gráfica 4. Porcentaje de diferencia en los objetos creados usando valores con multiplicidad de los factores



Gráfica 5. Porcentaje de diferencia en las referencias creadas usando valores con multiplicidad de los factores

4.1.3 Validación del modelo usando valores sin multiplicidad y variando los valores aleatorios. El experimento4 permite validar el modelo a partir de los factores *obj_creation_iat* y *obj_duration_time* sin multiplicidad de los factores usando los niveles alto y bajo y variando los valores aleatorios que usa el mismo para efectuar las simulaciones.

- **Objetivo** : El objetivo del experimento es determinar si existe la necesidad de realizar replicas de cada uno de los experimentos para tener mayor certeza en los resultados generados.

- **Simulación a realizar:** Se realizaron simulaciones en un tiempo de simulacion T= 16000m, bajo determinados valores de los factores tiempo de vida de un recurso (a3), tiempo de creación de un nuevo recurso en el sistema (a5), tiempo de creación de nueva referencia (a4) y usando ciertos valores aleatorios. En la tabla 9 se puede observar los valores dados a a_3 y a_5.

  El tiempo de creación de la referencia (a4), será 4m y el número de referencias por recursos es 1.

  Valores Aleatorios definidos en el archivo seed.ini

  - **Grupo aleatorio1 (E1):** 1768507984 33648008 1082809519
    703931312 1856610745 784675296
    426676692 1100642647

- **Grupo aleatorio2 (E2):** 72358415 271666411 1281145440
  991747913 137375739 614387732
  69416058 194900596

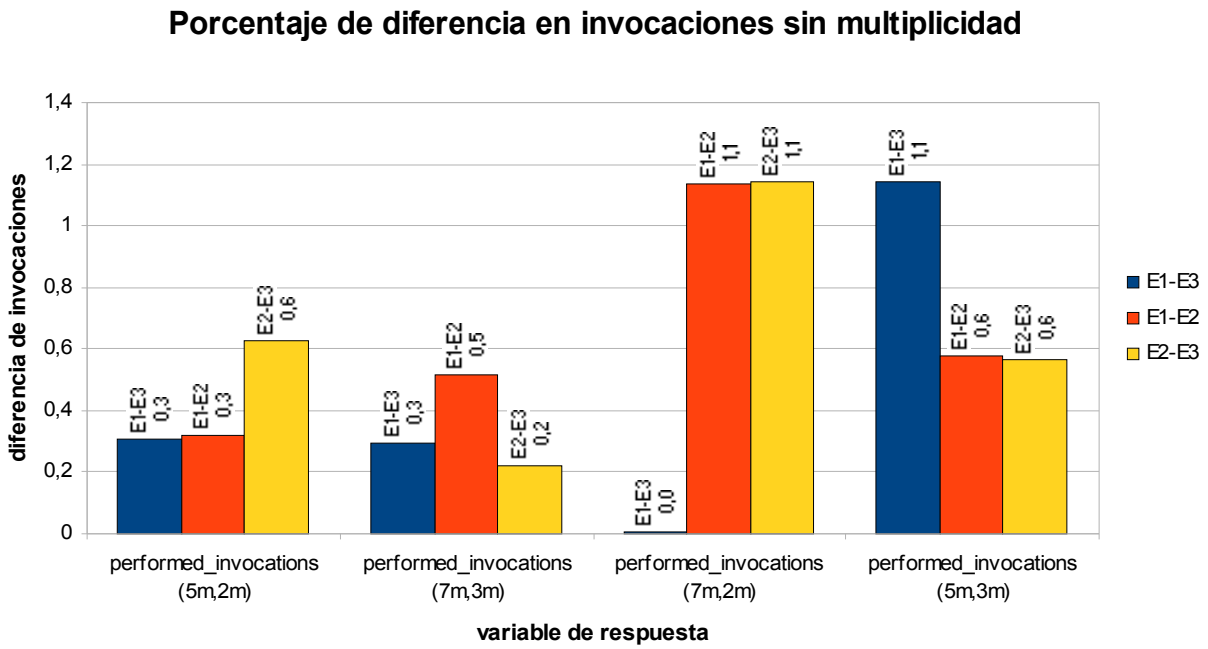
- **Grupo aleatorio3 (E3):** 1486252914 1187433110 1820194104
  898589432 725502823 1428095551
  424107619 1845654421

Multiplicidad		**Factor**			
		$a_3 < a_5$		**$a_3 > a_5$**	
		a_5	**a_3**	**a_5**	**a_3**
Sin	Bajo – bajo	2m	5m	5m	2m
	Alto – alto	3m	7m	7m	3m
	Alto - bajo	3m	5m	7m	2m
	Bajo – alto	2m	7m	5m	3m

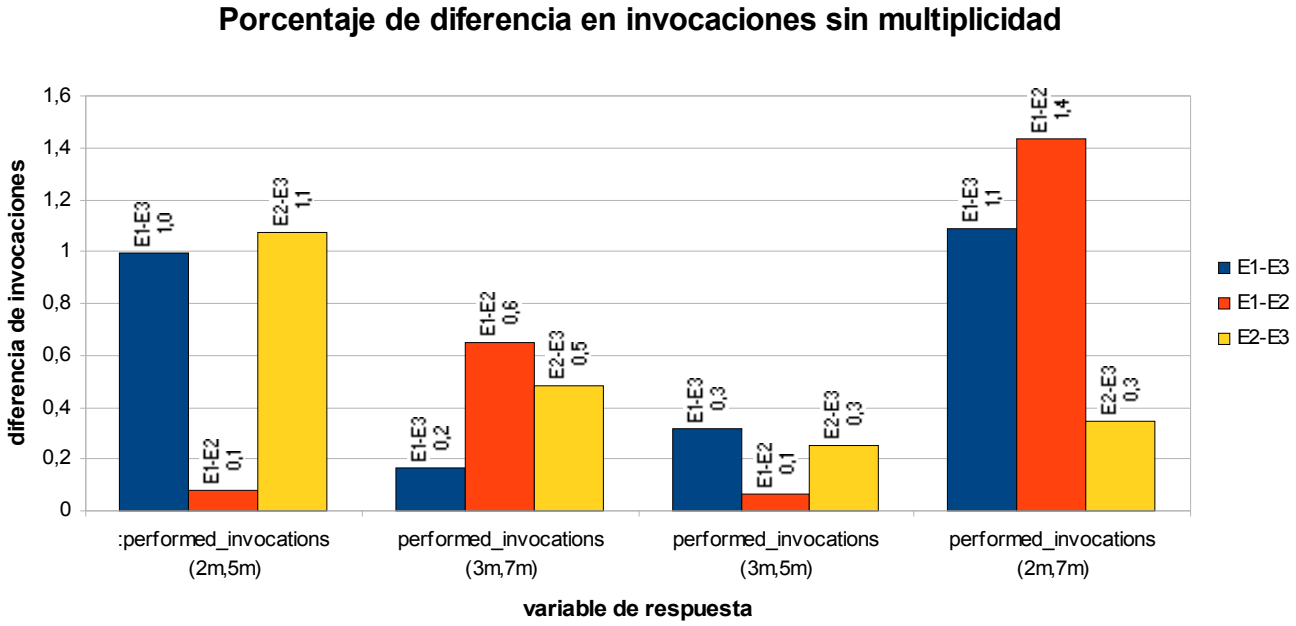
Tabla 9.. Valores definidos para los factores a_3 y a_5 sin multiplicidad

- **Variables de respuesta**. Ejecutado el programas experimento4.ini (Anexo 1) con los diferentes grupos de valores definidos en el archivo seed.ini, se determinan las diferencias entre los diferentes experimentos para las variables de respuesta :*Totals_performed_invocations* (total de invocaciones realizadas)
  Totals_expected_performed_invocations (total de invocaciones esperadas a ser ejecutadas)
  Total_created_objs (total de objetos creados)
  Total_created_references (total de referencias creadas)

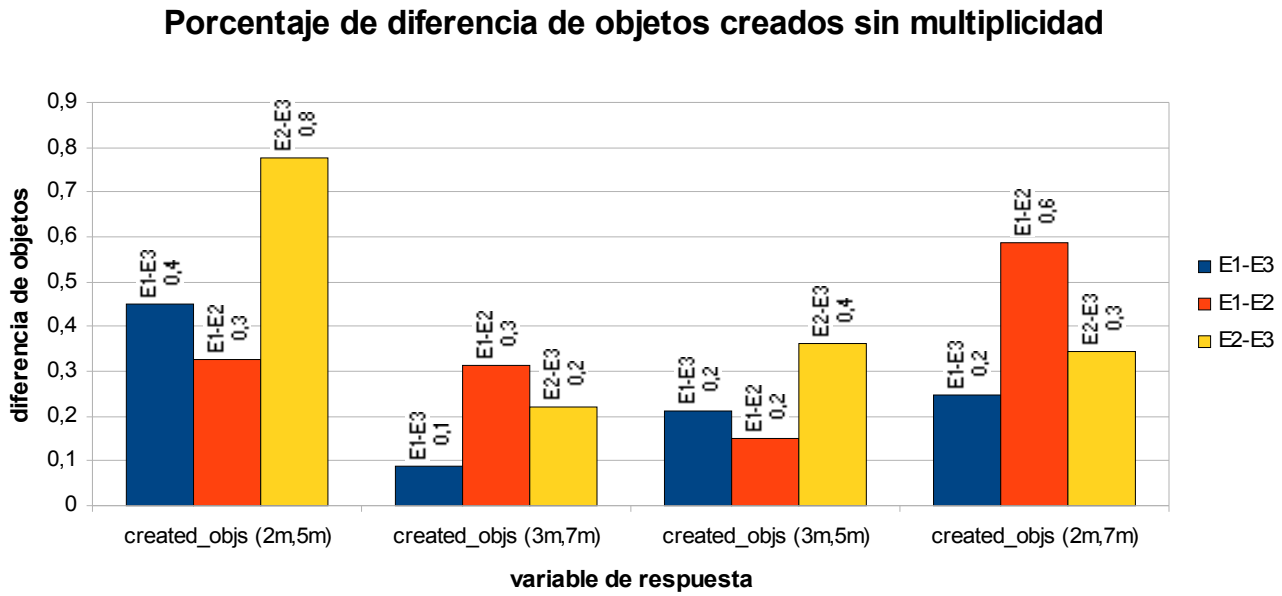
  En las gráficas del 6 al 11 se observa el comportamiento de las variables de respuesta.



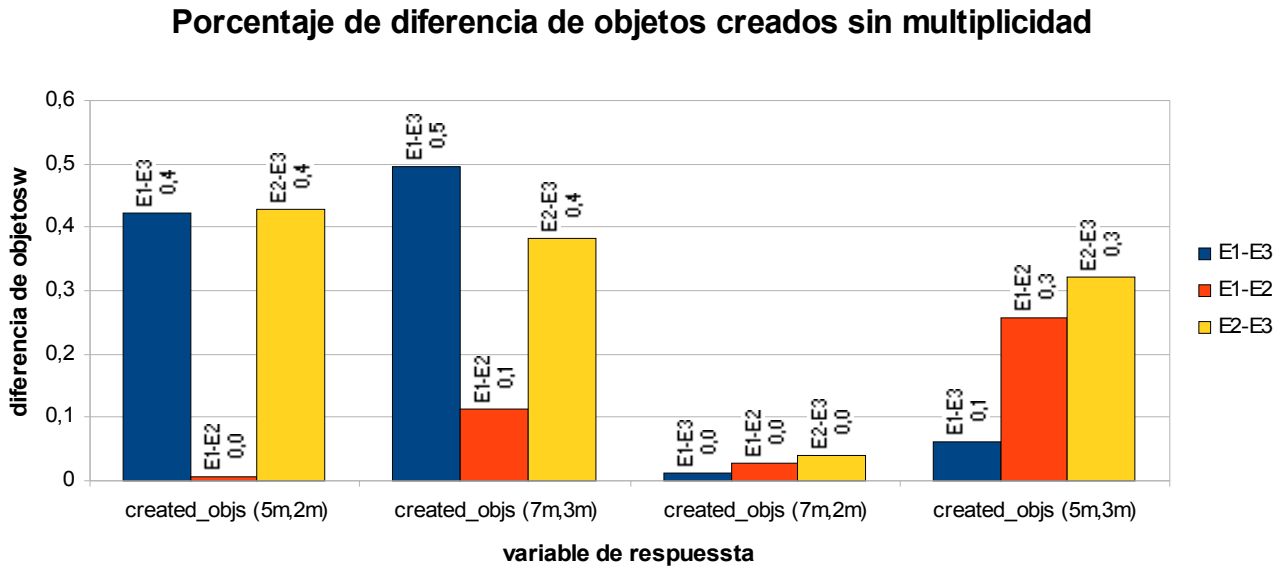
Gráfica 6. Porcentaje de diferencia en las invocaciones usando valores sin multiplicidad de los factores, variando valores aleatorios a3>a5



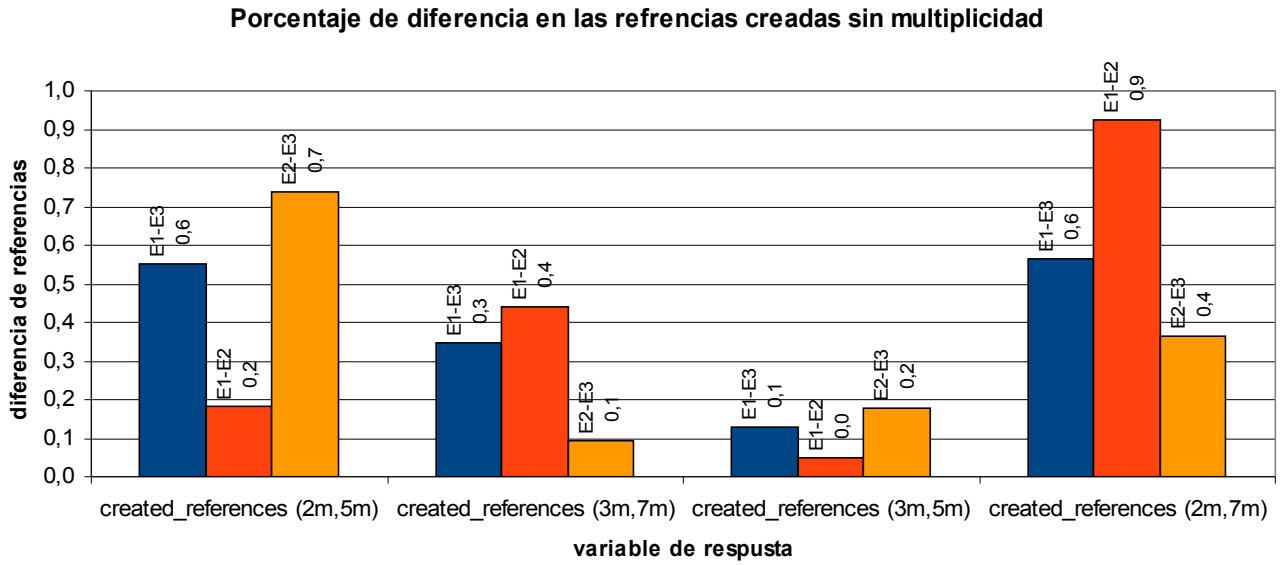
Gráfica 7. Porcentaje de diferencia en las invocaciones usando valores sin multiplicidad, variando valores aleatorios, a5>a3



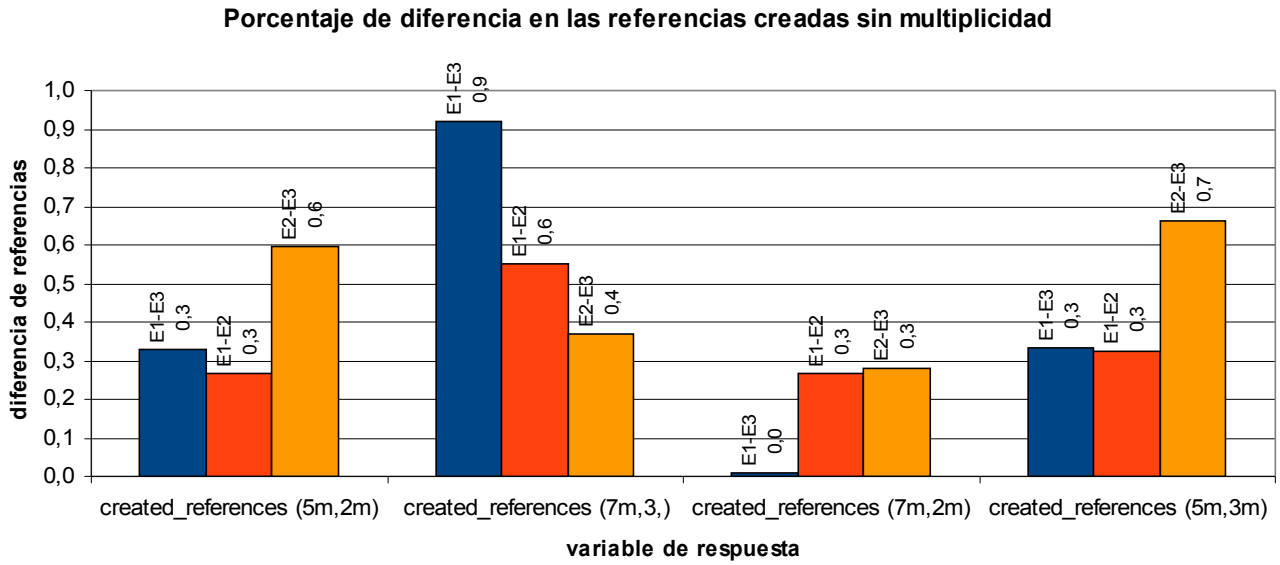
Grafica 8. Porcentaje de diferencia de objetos creados sin multiplicidad de los factores, variando valores aleatorios, a3>a5



Grafica 9. Porcentaje de diferencia de objetos creados sinn multiplicidad de los factores variando los valores aleatorios, a5>a3



Grafica 10. Porcentaje de diferencia de objetos creados sin multiplicidad de los factores variando los valores aleatorios, a5>a3



Grafica 11. Porcentaje de diferencia de objetos creados sin multiplicidad de los factores variando los valores aleatorios, a5<a3

- **Análisis de Resultados.**

En las gráficas 1 y 2 se puede observar los resultados de los experimentos, en donde se valida que el número de invocaciones efectuadas se aproxima al número de invocaciones

esperadas, determinando que el modelo es confiable debido a que no es significativo el número de invocaciones que no se ejecutan, esto cuando no existe migración de objetos en el sistema y sin importar cada cuanto se crea un objeto y el tiempo de permanencia en el mismo. Esto es, en el momento en que nace un recurso, se crean n referencias, con la excepción del sitio de nacimiento, según una distribución Poisson y cada referencia realiza accesos a los recursos según una distribución exponencial; a partir de esto se calcula el número de invocaciones esperadas sobre el recurso; si una invocación no se ejecuta, se pierde en el sistema y se contabiliza como invocación perdida.

Igualmente se puede observar en las graficas del 3 al 11 que representan los resultados de los experimentos 3 y 4, que el porcentaje de diferencia entre los datos arrojados en cada una de estas ejecuciones no es mayor al 5%. Esto es, tanto la diferencia en las invocaciones realizadas, los objetos y referencias creadas son poco significativos respecto a las esperadas de acuerdo al modelo.

El comportamiento de las variables de respuesta permite concluir que los cambios de los factores en cada uno de los experimentos no generan grandes cambios, por consiguiente no amerita el realizar replicas de los experimentos.

4.2 DISEÑO Y ANÁLISIS DE ESPERIMENTOS PARA DETERMINAR LA EFICIENCIA Y EL COSTO DE LA TÉCNICA DIFUSIÓN

Se diseña un experimento a partir del factor *obj_inv_iat* y del uso de los s que permitan realizar comparaciones para determinar la eficiencia y el costo computacional de la técnica difusión.

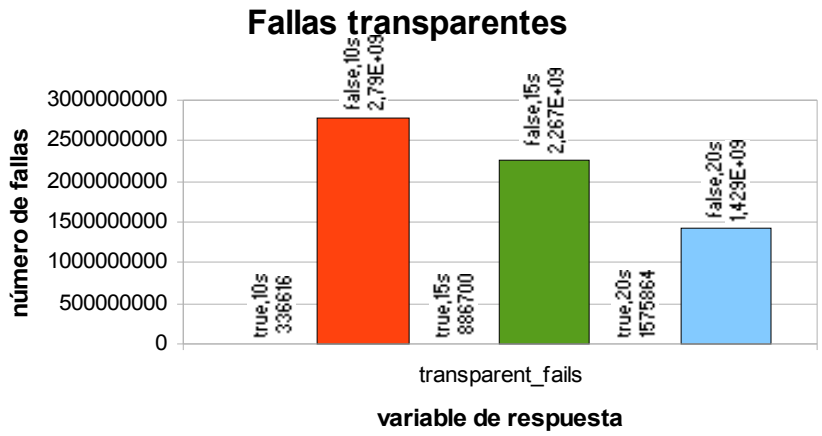
- **Objetivo:** El objetivo del experimento es analizar el comportamiento de las fallas transparentes, fallas notables y los niveles de las fallas notables que se realizan en el modelo .

- **Simulación a realizar:** se analizará el factor *obj_inv_iat*, en los niveles del *with_caching* en *true* y *false y* en un tiempo de simulación *T*= 1600m.

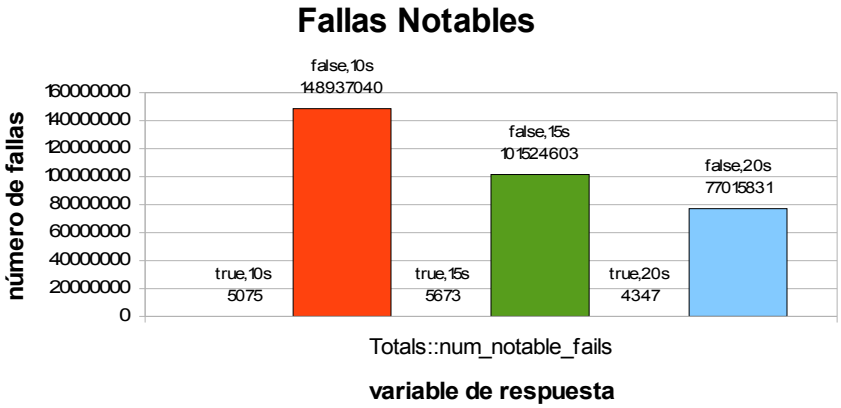
- **Variables de respuesta**. Ejecutado el programas experimento5.ini (Anexo 1) se determinan los valores de las variables de respuesta

transparent_fails	*(*total de fallas transparentes)
num_notable_fails	(total de fallas notables)
Nested_notable_fails	(total de niveles de fallas notables)
Perfomed_invocatins	(total de invocaciones realizadas)
Expected_perfomen_invocations	(total de invocaciones que se esperan ser ejecutadas)

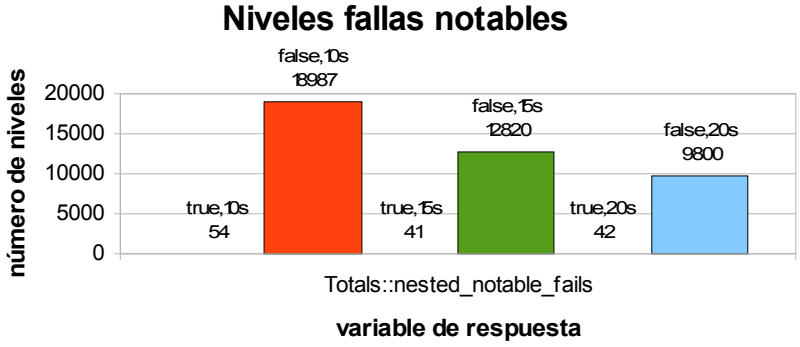
En las gráficas del 12 al 15 se observa el comportamiento de las variables de respuesta



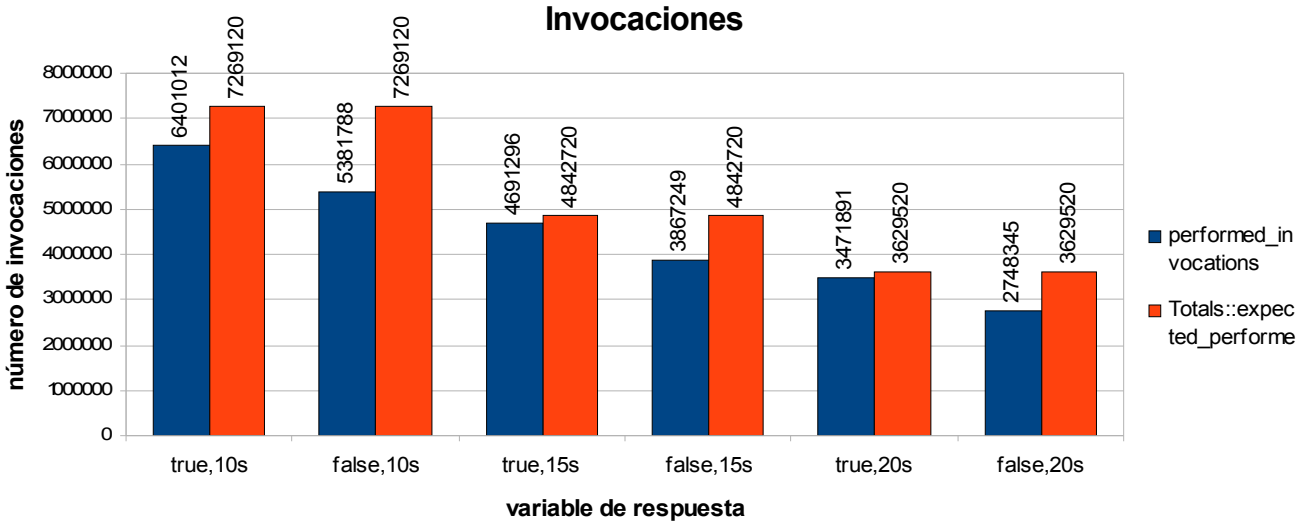
Gráfica 12. Fallas transparentes con solo difusión



Gráfica 13. Fallas Notables con difusión



Gráfica 14. Niveles de fallas notables con difusión



Gráfica 15. Invocaciones ejecutadas realizando solo difusión

- **Análisis de resultados.**

Tomando las gráficas 12, 13 y 14, lo que da como resultado que el número de fallas transparente[10] es alto cuando solo existe difusión; esto da la impresión que el uso de esta técnica no tiene un costo comunicacional; pero analizando la fallas notables y los niveles que se generan de esta fallas, se puede concluir que esto no es cierto, debido a que este tipo de fallas es la más costosa en virtud que además de realizar intentos de acceso, requiere emprender una búsqueda a través de todo el sistema realizando solicitudes al localizador el cual arroja una nueva localidad que hace que se repetirá el intento de acceso; este intento puede ser exitoso o generar cualquiera de las fallas.

En la gráfica 15 se puede observar que disminuye la eficiencia del sistema cuando el *with_caching* se encuentra inactivo, debido a que el número de invocaciones que se realizan es menor y las invocaciones perdidas son mas representativas; esto es, existe una mayor diferencia entre las invocaciones esperadas y las realizadas, ocasionando perdida de las mismas.

4.3 DISEÑO Y ANÁLISIS DE EXPERIMENTOS PARA DETERMINAR LA EFICIENCIA Y EL COSTO DE LOS S ARRIVING, LIVE Y DEAD

Para analizar los *cache arriving*, *live* y *dead,* se ha diseñado una serie de experimentos que permitan examinar el comportamiento de los mismos, donde se definen dos factores objetos de estudio : el tamaño del *cache* (*arriving_size, live_size* y *dead_size)* y el tiempo de invocación del objeto (*obj_inv_iat).*

[10] Falla que se genera cuando el localizador emisor del acceso encuentra una referencia reciente y redirige el acceso hacia la nueva localización

4.3.1 ***Cache Arriving.*** Para realizar el análisis del *cache arriving* se toma el experimento 6 que se plantea a continuación.

- **Objetivo**. Determinar el comportamiento del *cache arriving* bajo los factores *cache_size* y *obj_inv_iat*

- **Simulación a realizar**: Se determinan tres (3) niveles para el tamaño del cache y dos (2) para los tiempos de invocación del objeto (ver tabla 9)

Tañamo de la lista del : 3
Número de referencias por objeto: 8
Número de hosts: 25
Tiempo de creación de una nueva referencia : 12000m

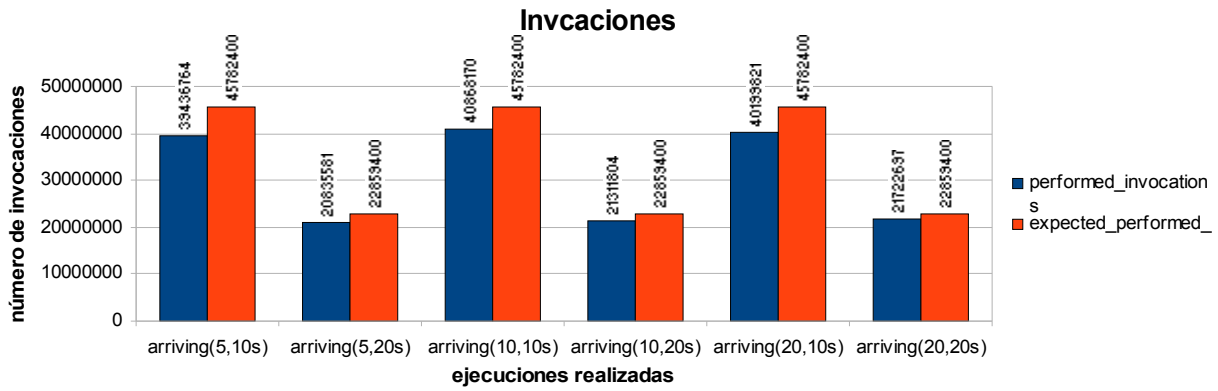
Cache_size	***obj_inv_iat***
5	10s
5	20s
10	10s
10	20s
20	10s
20	20s

Tabla 10 . Tamaños del *cache arriving*

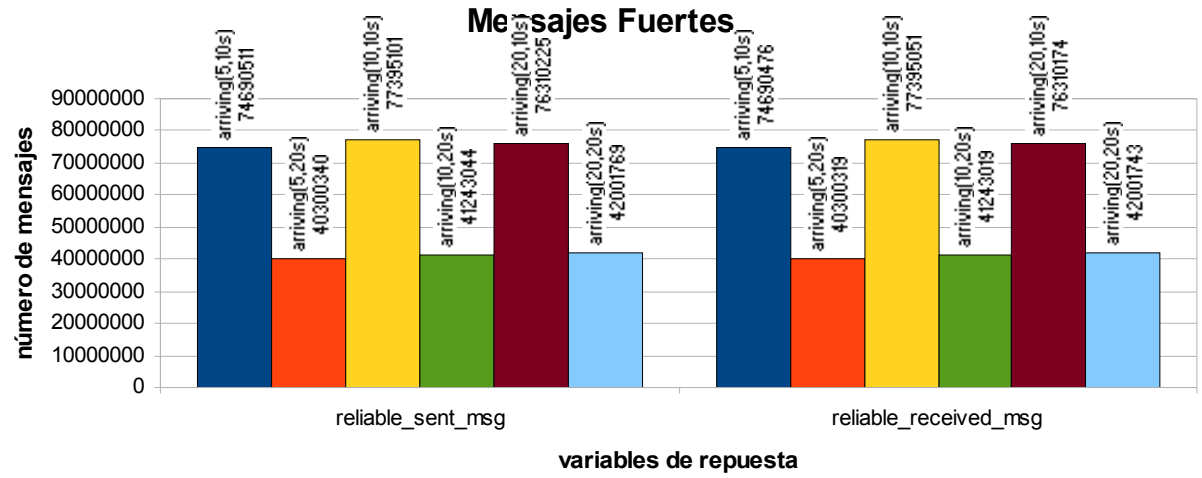
- **Variables de respuesta**. Ejecutado el programas experimento6.ini (Anexo 1) se determinan los valores de las variables de respuesta

transparent_fails	*(*total de fallas transparentes)
num_notable_fails	(total de fallas notables)
Nested_notable_fails	(total de niveles de fallas notables)
Perfomed_invocatins	(total de invocaciones realizadas)
Expected_perfomen_invocations	(total de invocaciones que se esperan ser ejecutadas)
Reliable_sent_msg	*(total de mensajes fuertes enviados)*
unreliable_sent_msg	*(total de mensajes débiles enviados)*
reliable_broadcast_sent	*(total de broadcast fuertes)*
un reliable_broadcast_sent	*(total de broadcast débiles)*
simples_fails	*(total de fallas simples)*
notable_fails	*(total de fallas notables)*
nested_simples_fails	*(notal de niveles de fallas simples)*
nested_notable_fails	*(notal de niveles de fallas notables)*

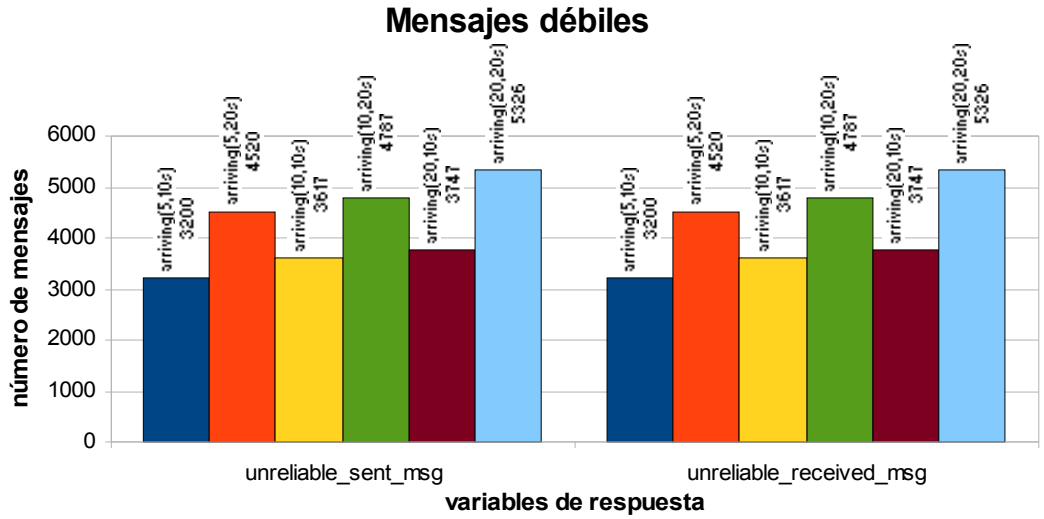
En las gráficas del 15 al 22 se observa el comportamiento de las variables de respuesta



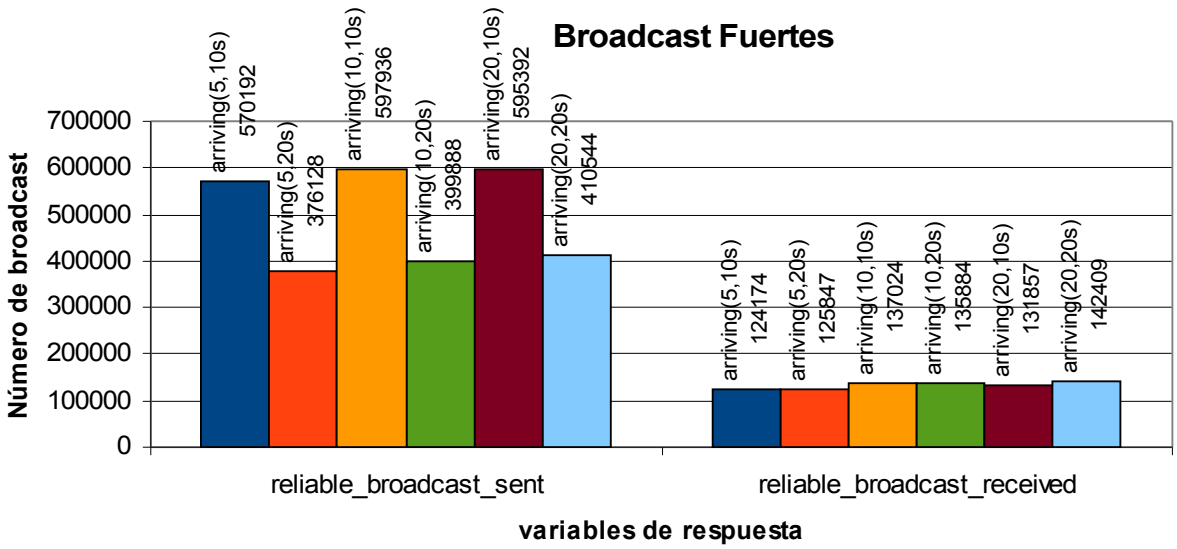
Gráfica 16. Invocaciones realizadas variando el tamaño del *cache arriving* y la frecuencia de acceso del objeto.



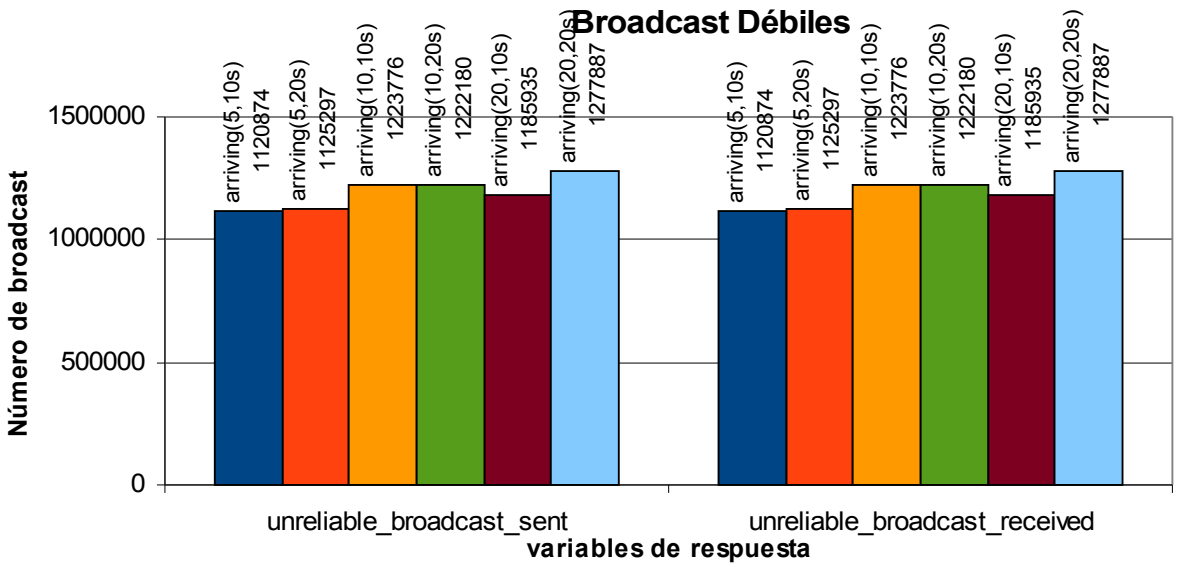
Gráfica 17. Mensajes fuertes realizados variando el tamaño del *arriving* y la frecuencia de acceso del objeto



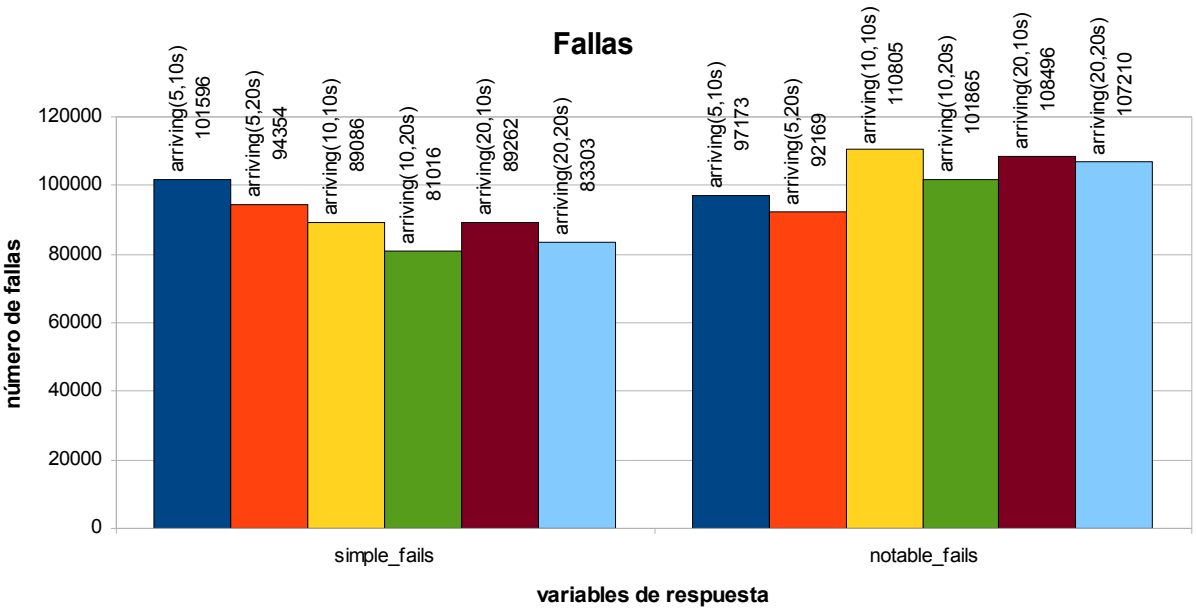
Gráfica 18. Mensajes débiles realizados variando el tamaño del *cache arriving* y la frecuencia de acceso del objeto



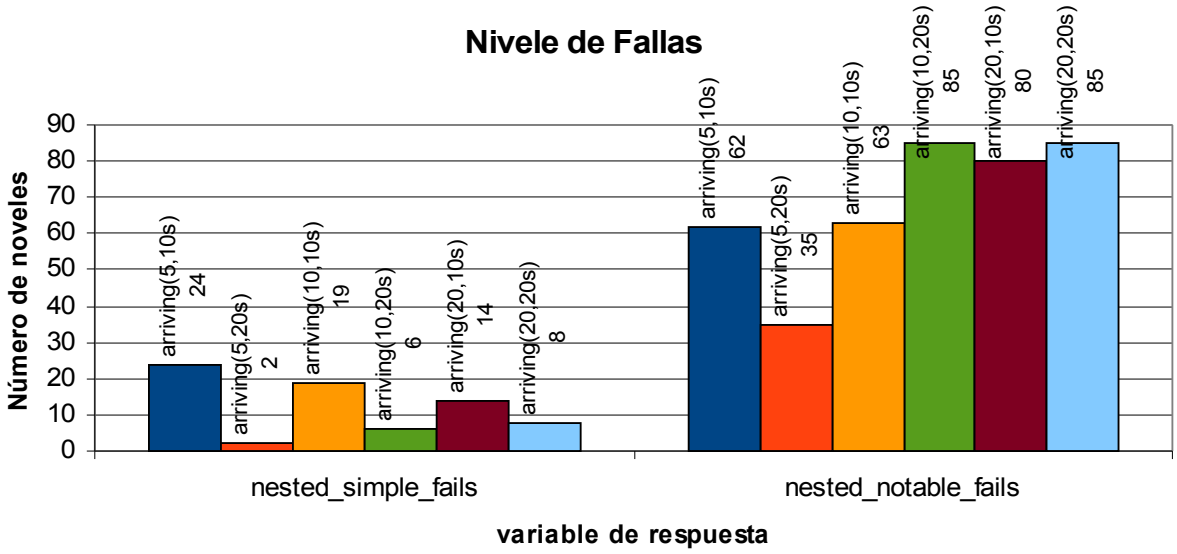
Gráfica 19. *Broadcast* fuertes realizados variando el tamaño del *cache arriving* y la frecuencia de acceso del objeto



Gráfica 20. *Broadcast* débiles realizados variando el tamaño del *cache arriving* y la frecuencia de acceso del objeto



Gráfica 21. Fallas generada variando el tamaño del *cache arriving* y la frecuencia de acceso del objeto



Gráfica 22. Niveles de fallas generadas variando el tamaño del *cache arriving* y la frecuencia de acceso del objeto

- **Análisis de resultados**.

Analizando las variables de respuesta cuando se usa el *cache arriving* se pueden evidenciar las siguientes situaciones:

1. *Entre mayor sea la frecuencia de acceso, las fallas simples y notables se incrementan* (ver gráfica 21); esto es, para las fallas simples el sistema debe repetir el acceso redirigido para la localización del recurso dada por *reply-update, nstop* veces; en este caso la frecuencia de acceso es importante pero no es significativa, generando un costo comunicacional muy bajo. Por otro lado las fallas notables se generan debido a que el número de intentos de acceso *nstop,* es mayor al definido para generarse una falla simple; el sistema ha tratado de ubicar el recurso con la dirección mas reciente de localización del mismo y no ha sido posible encontrarlo; igualmente se puede crear cuando no se ha dado una respuesta de *reply-not-found* por parte del emisor del acceso; es decir, no se tiene la información de ubicación del recurso.

   De manera similar si se observa la gráfica 22, se puede analizar el comportamiento de la variable de respuesta *nested_notables_fails*, en donde dichos valores son mas altos cuando la frecuencia de acceso es mayor; esto acarrea un mayor costo comunicacional debido a que se activa una búsqueda al localizador creándose un nuevo acceso, el cual no garantiza la ubicación del recurso.

2. *Entre mayor sea la frecuencia de acceso se producen mas difusiones y mensajes fuertes*. Esto es, se están produciendo más búsquedas fuertes que generan costos muy altos de comunicación, pero se tiene mayor certeza que el recurso es encontrado o que definitivamente éste ya no se encuentra en el sistema. Por otra

parte la diferencia entre los mensajes enviados y los recibidos es mínima; esto es, el número de mensajes perdidos usando difusiones fuertes y mensajes fuertes no es significativo.

En las gráficas 17 y 19, se puede ver que las difusiones recibidas difieren considerablemente de las enviadas, pero esto no se ve reflejado en los mensajes, puesto que el número de mensajes enviados y recibidos tiende a ser el mismo; esto es, el número de mensajes enviados y recibidos no dependen directamente del número de difusiones que se produzcan pero si del tipo de búsqueda que esté haciendo; en este caso una búsqueda fuerte.

3. *La frecuencia de acceso al recurso y el tamaño del cache influyen en el número de mensajes débiles enviados y recibidos*. Cuando la frecuencia de invocación del recurso es alta el número de mensajes tiende a ser estable, mas no así el número de difusiones, ya que estas se incrementan ocasionando mayor número de fallas notables y de difusiones perdidas, siendo esto significativo para el sistema; pero si se produce un incremento en el tamaño del caché y se disminuye la frecuencia de acceso, el número de fallas notables tiende a estabilizarse, con un valor menor a la frecuencia de acceso alta. En las gráficas 18 y 20, se puede ver esta situación.

   De acuerdo a los datos arrojados en los experimentos y que se ven en la gráfica 17, se puede observar que el sistema tiende a mantener la eficiencia del mismo en cuanto al número de mensajes enviados y recibidos; la diferencia entre ellos no es significativa.

4. *La frecuencia de acceso del recurso tiene relación directa con la eficiencia del sistema.* Cuando se invoca el recurso más rápidamente, si el tamaño del cache se incrementa hace que la eficiencia del sistema mejore levemente produciendo igualmente eficiencia en las migraciones. Esto no ocurre cuando la frecuencia es baja pero el tamaño del cache es mayor; en este caso la eficiencia tiende a disminuir pero sigue siendo mejor que en altas frecuencia (ver gráfica 16).

5. *A mayor tamaño del cache, más accesos.* Cuando se aumenta el tamaño del *cache*, la eficiencia del sistema se incrementa y disminuye el número de fallas notables.

4.3.2 ***Cache live.*** Para realizar el análisis del *cache live* se toma el experimento 7 que se plantea a continuación.

- **Objetivo**. Determinar el comportamiento del *cache live* bajo los factores *cache_size* y *obj_inv_iat*

  - **Simulación a realizar**: Se determinaron dos (2) niveles para el tamaño de y tres (3) para los tiempos de invocación del objeto (ver tabla 11)

.

Live_size **(Tamaño del)**	***obj_inv_iat*** **(tiempo de invocación del objeto)**
10	10s
10	15s
10	20s
15	10s
15	15s
15	20s

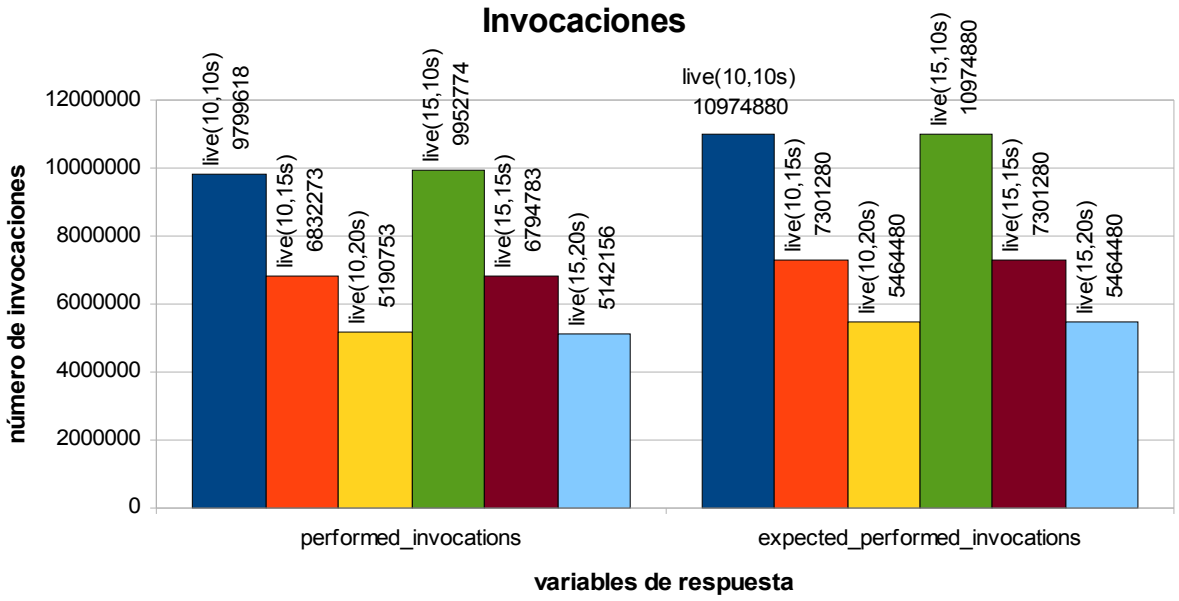
Tabla 11. Factores de análisis para el *Live*

Tañamo de la lista del : 3
Número de referencias por objeto: 8
Número de hosts: 20
Tiempo de creación de una nueva referencia : 12000m

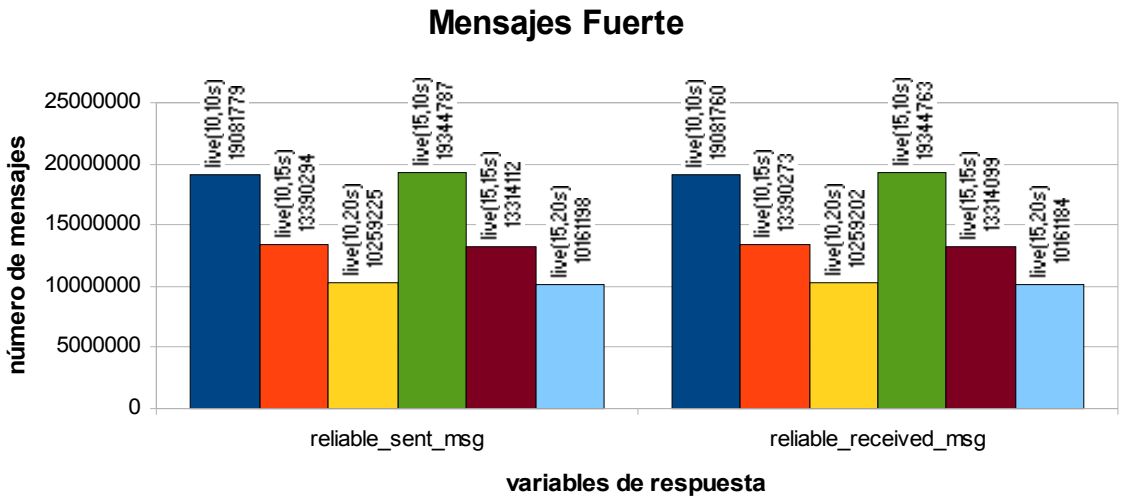
- **Variables de respuesta**. Ejecutado el programa experimento7.ini (Anexo 1) se determinan los valores de las variables de respuesta

transparent_fails *(*total de fallas transparentes)
num_notable_fails (total de fallas notables)
Nested_notable_fails (total de niveles de fallas notables)
Perfomed_invocatins (total de invocaciones realizadas)
Expected_perfomen_invocations (total de invocaciones que se esperan ser ejecutadas)
Reliable_sent_msg *(total de mensajes fuertes enviados)*
unreliable_sent_msg *(total de mensajes débiles enviados)*
reliable_broadcast_sent *(total de broadcast fuertes)*
un reliable_broadcast_sent *(total de broadcast débiles)*
simples_fails *(total de fallas simples)*
notable_fails *(total de fallas notables)*
nested_simples_fails *(notal de niveles de fallas simples)*
nested_notable_fails *(notal de niveles de fallas notables)*

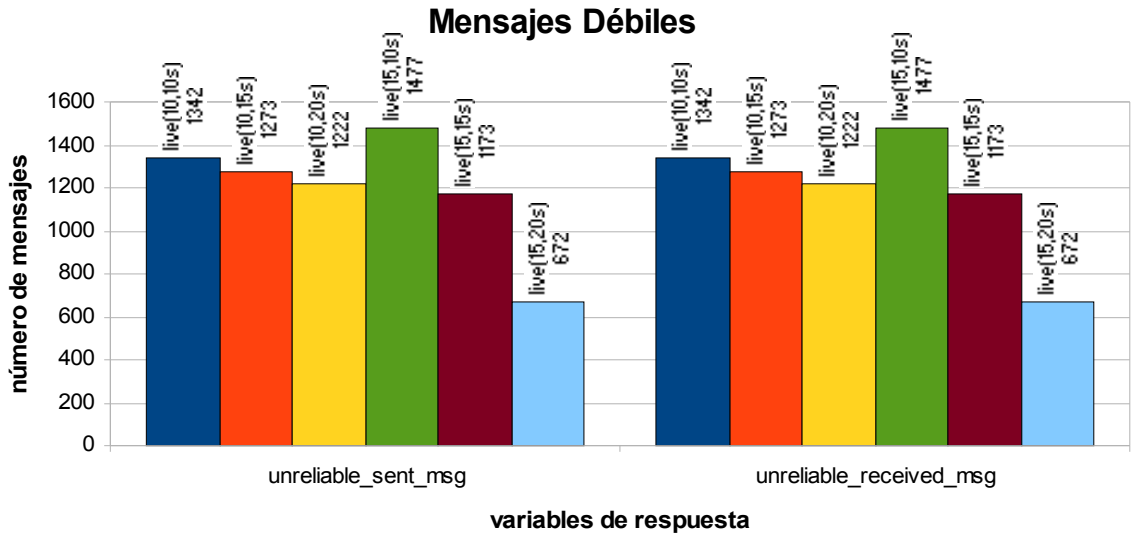
En las gráficas del 23 al 29 se observa el comportamiento de las variables de respuesta



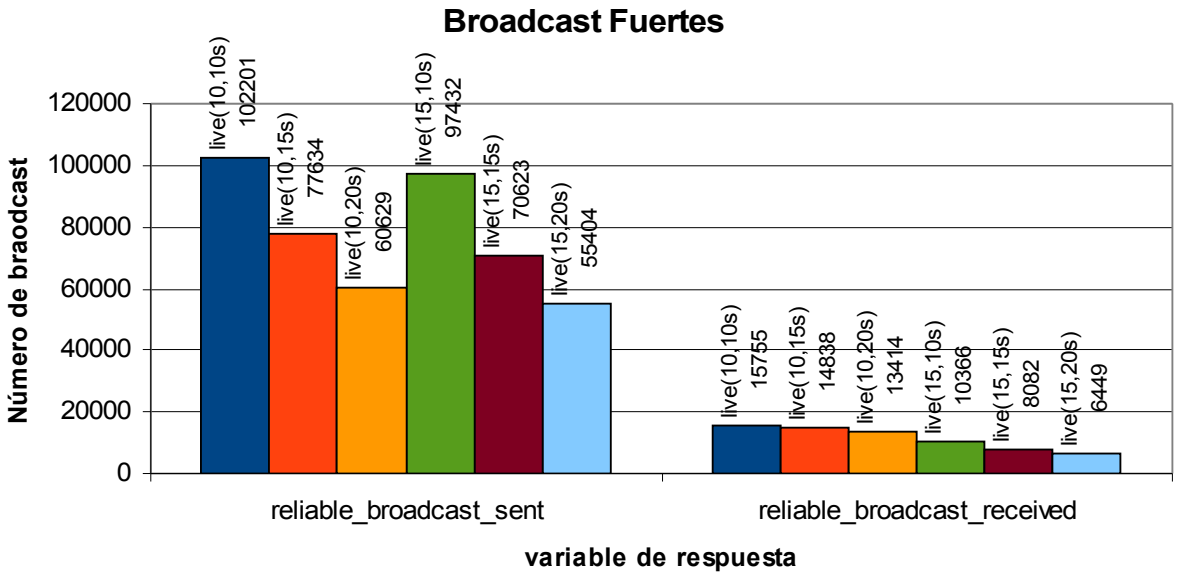
Gráfica 23. Invocaciones realizadas usando el *cache live*



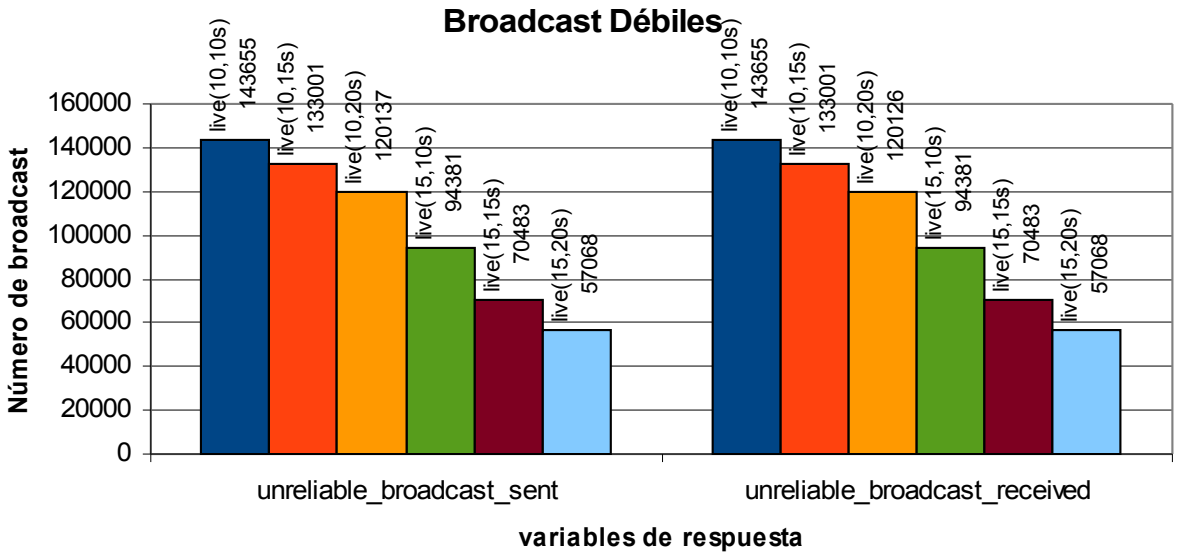
Gráfica 24. Mensajes fuertes enviados y recibidos usando el *cache live*



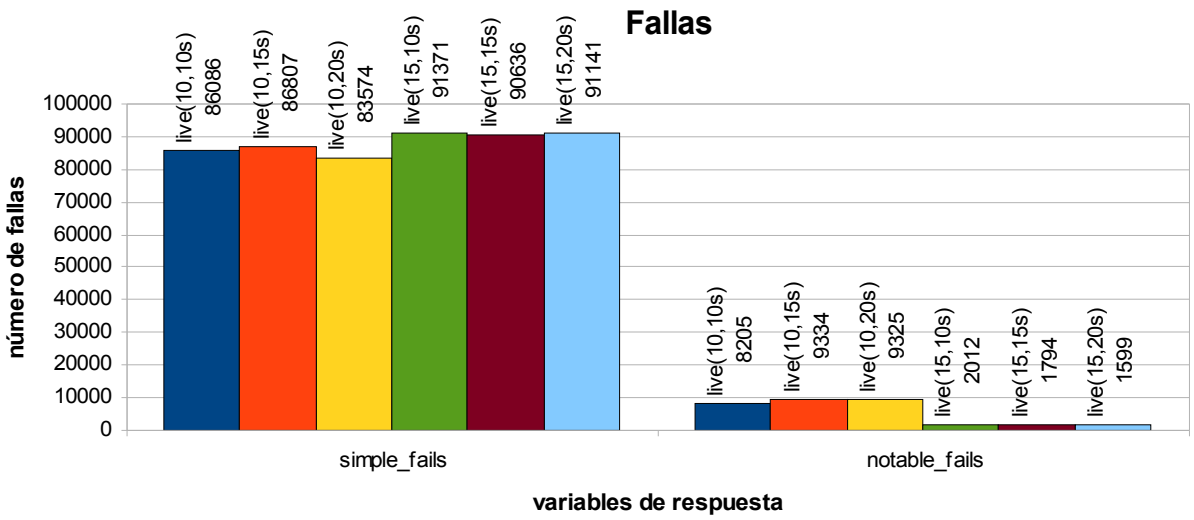
Gráfica 25. Mensajes fuertes enviados y recibidos usando el *cache live*



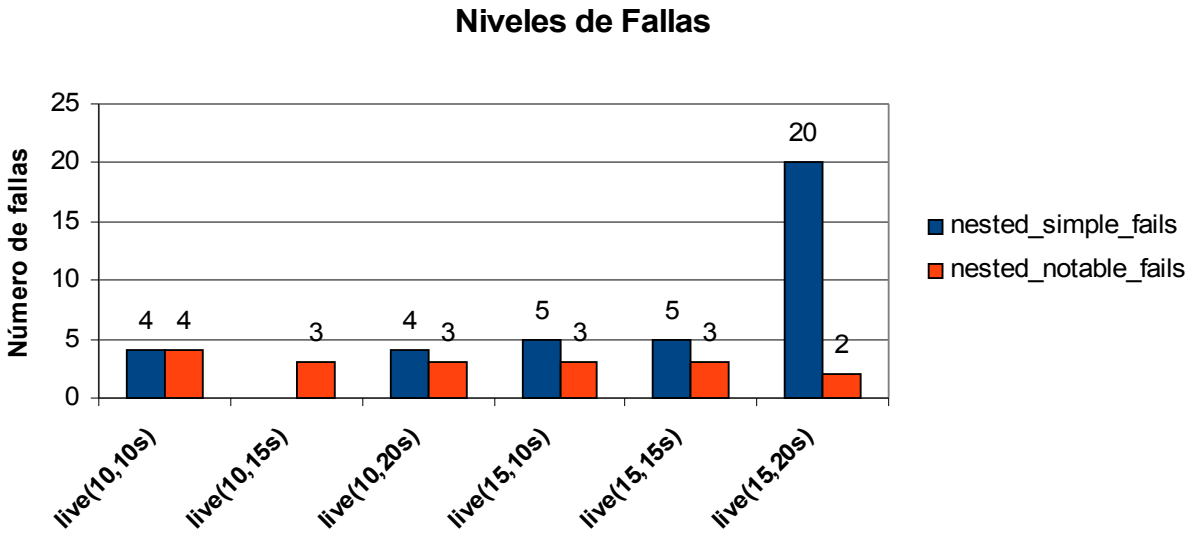
Gráfica 26. *Broadcast* fuertes enviados y recibidos usando el *cache live*



Gráfica 27. *Broadcast* débiles enviados y recibidos usando el *cache live*



Gráfica 28. Fallas generadas usando el *cache live*



Gráfica 29. Niveles de fallas generadas usando el *cache live*

- **Análisis de Resultados.** Realizado el análisis correspondiente a las variables de salida se puede precisar las siguientes observaciones:

1. *No es crítico el tamaño del cache ni la frecuencia de invocación del recurso para la generación de fallas.* La frecuencia de acceso al recurso es significativa para la generación de fallas simples; a menor frecuencia de acceso, mayor es la generación de este tipo de fallas; lo anterior representa un costo comunicacional mínimo para el sistema. Respecto a las fallas notables, se genera un número bajo de este tipo de falla el cual está determinado por el tamaño del cache, haciendo que el costo comunicacional del uso de este sea bajo. El recurso es encontrado dentro del límite establecido en *nstop* accesos.

   La frecuencia de invocación del recurso y el tamaño del cache no son críticos para el manejo del *live* (ver gráfica 29)

2. *Entre mayor sea la frecuencia de acceso se producen mas difusiones fuertes y por consiguiente mas mensajes fuertes***.** Se realizan más difusiones fuertes cuando la frecuencia de acceso al recurso es alta; el número de difusiones fuertes, tiende a ser la misma cuando el tamaño del es igual; si el tamaño de éste es incrementado el número de difusiones disminuye no siendo significativa esta diferencia (ver gráficas 24 y 26).

3. *El número de difusiones débiles depende del tamaño y la frecuencia de acceso*. Entre mas grande sea el tamaño del menor es el número de mensajes enviados y recibidos variando este número de acuerdo a la frecuencia de acceso; igualmente la perdida de mensajes tiende a disminuir si se incrementa el tamaño del cache, no siendo tan significativa en comparación al número de mensajes enviados y recibidos. El número de difusiones débiles dependen de los dos factores. (gráficas

25 y 27).

4. *La eficiencia del se incrementa cuando se aumenta el tamaño del cache y se disminuye la frecuencia de acceso.* Cuando el valor del factor *live_size* es pequeño pero la frecuencia de acceso es alta, la eficiencia del disminuye; el número de invocaciones realizadas no son las que se espera que se generen. Esto no es tan significativo si se disminuye el tamaño del y la frecuencia de acceso; en este caso la eficiencia aumenta, pero se produce un número menor de invocaciones (gráfica 23).

4.3.3 ***Cache Dead***

- **Objetivo**. Determinar el comportamiento del *cache dead* bajo los factores *cache_size* y *obj_inv_iat*

- **Simulación a realizar**: Se determinaron tres (3) niveles para el tamaño de y dos (2) para los tiempos de invocación del objeto (ver tabla 12)

Dead_size **(Tamaño del)**	***obj_inv_iat*** **(tiempo de invocación del objeto)**
5	10s
5	20s
10	10s
10	20s
20	10s
20	20s

Tabla 12. Factores de análisis para el *Dead*

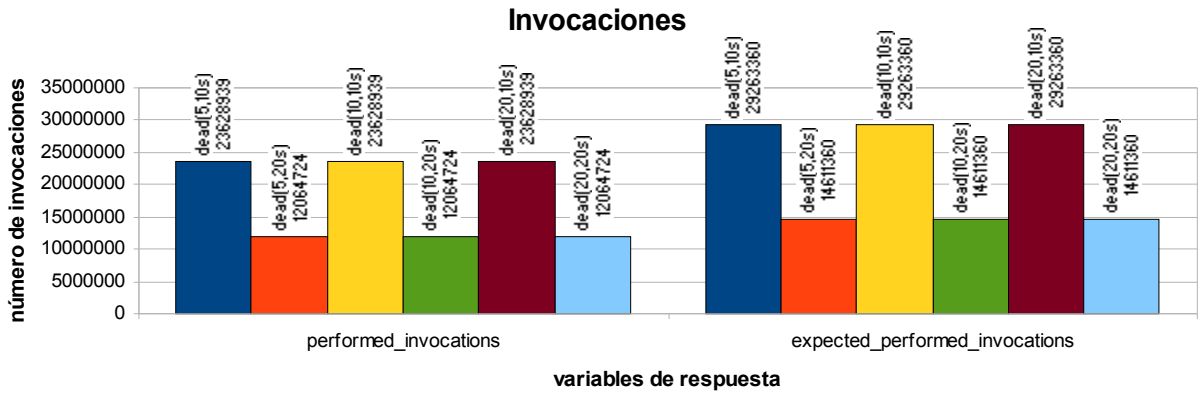
Tañamo de la lista del : 3
Número de referencias por objeto: 8
Número de hosts: 20
Tiempo de creación de una nueva referencia : 12000m

- **Variables de respuesta**. Ejecutado el programa experimento8.ini (Anexo 1) se determinan los valores de las variables de respuesta

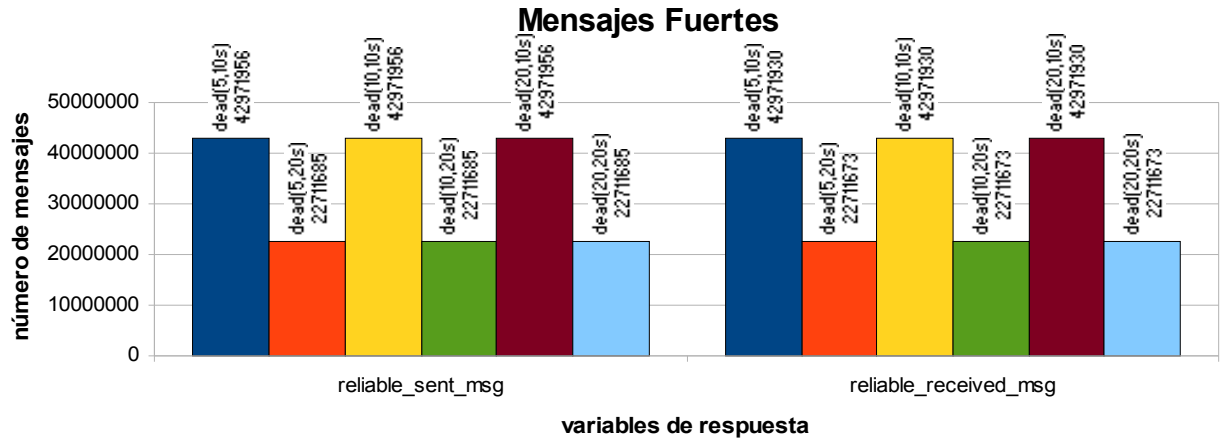
transparent_fails *(*total de fallas transparentes)
num_notable_fails (total de fallas notables)
Nested_notable_fails (total de niveles de fallas notables)
Perfomed_invocatins (total de invocaciones realizadas)
Expected_perfomen_invocations (total de invocaciones que se esperan ser ejecutadas)
Reliable_sent_msg *(total de mensajes fuertes enviados)*

unreliable_sent_msg	*(total de mensajes débiles enviados)*
reliable_broadcast_sent	*(total de broadcast fuertes)*
un reliable_broadcast_sent	*(total de broadcast débiles)*
simples_fails	*(total de fallas simples)*
notable_fails	*(total de fallas notables)*
nested_simples_fails	*(notal de niveles de fallas simples)*
nested_notable_fails	*(notal de niveles de fallas notables)*

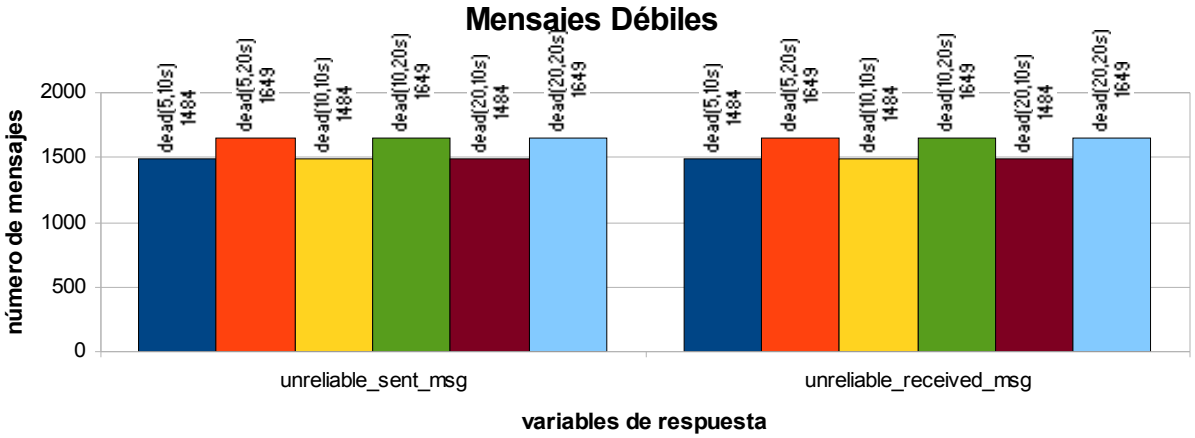
En las gráficas del 30 al 36 se observa el comportamiento de las variables de respuesta.



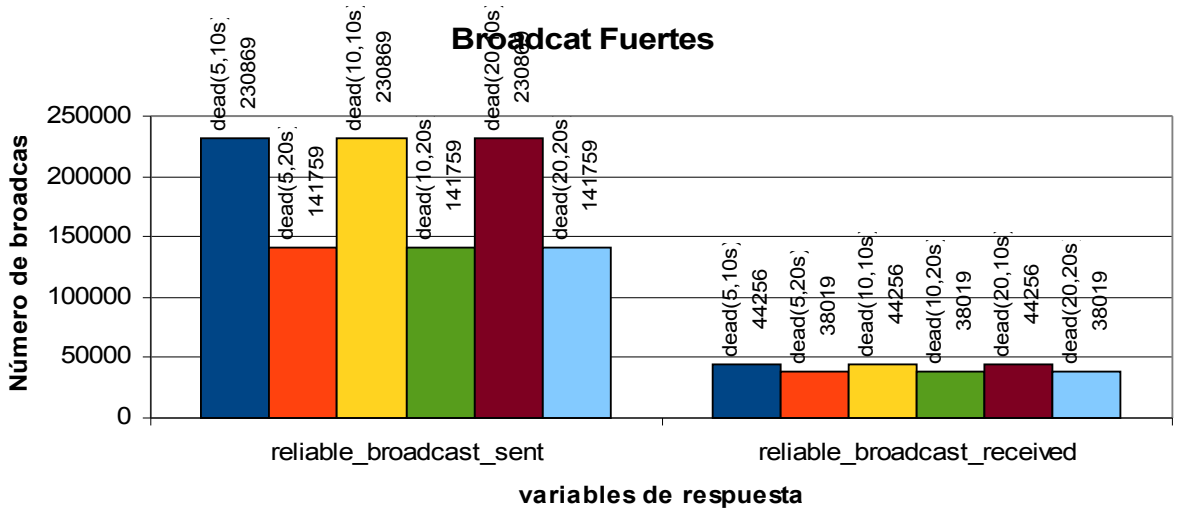
Gráfica 30. Invocaciones realizadas usando el *cache dead*



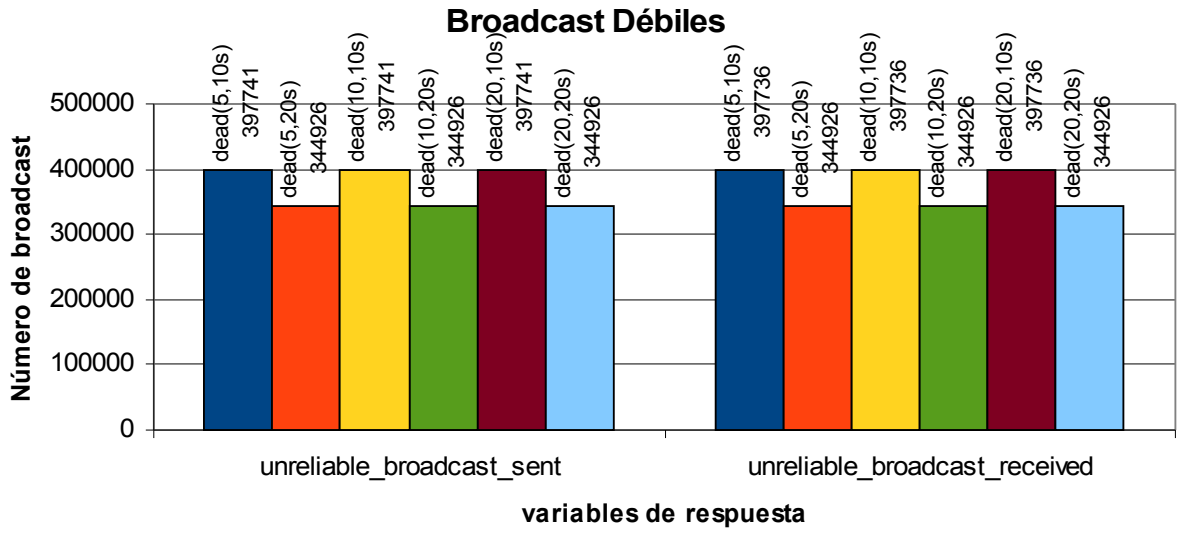
Gráfica 31. Mensajes fuertes enviados y recibidos usando el *cache dead*



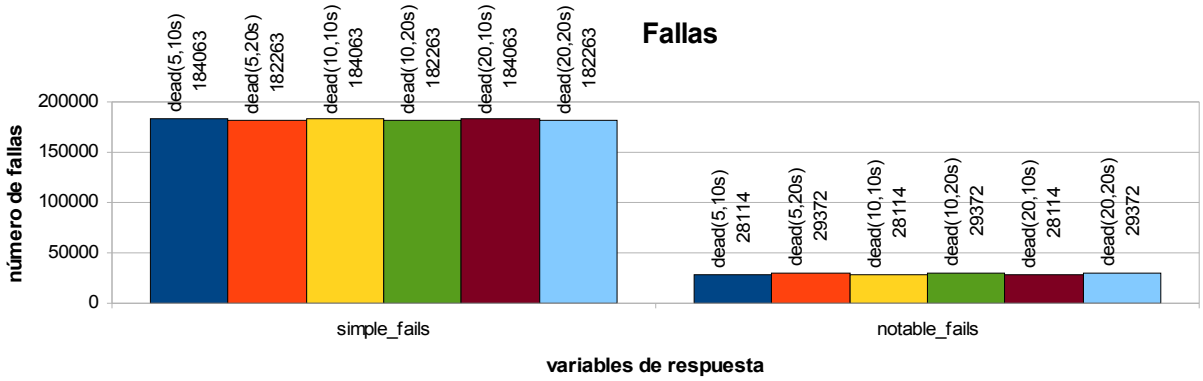
Gráfica 32. Mensajes débiles enviados y recibidos usando el *cache dead*



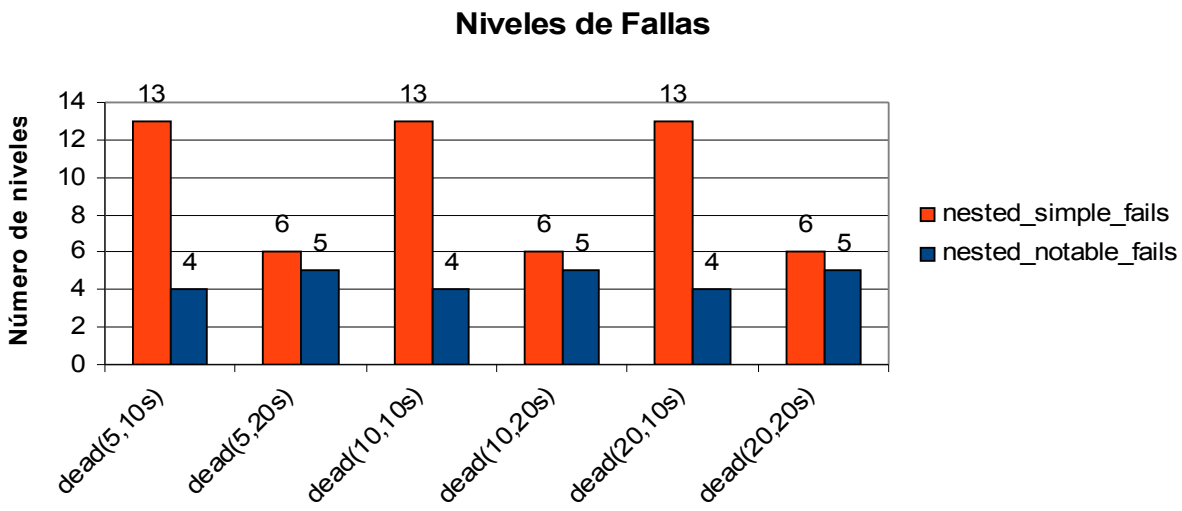
Gráfica 33. *Broadcast* fuertes enviados y recibidos usando el *cache dead*



Gráfica 34. *Broadcast* débiles enviados y recibidos usando el *cache dead*



Gráfica 35. Fallas generadas usando el *cache dead*



Gráfica 36. Niveles de fallas generadas usando el *cache dead*

Análisis de Resultados. Realizado el análisis correspondiente a las variables de salida se puede precisar las siguientes observaciones:

1. *El comportamiento de las fallas notables está directamente relacionadas con la frecuencia de acceso***.** El tamaño del no afecta el número de fallas notables que se producen en éste, pero si la frecuencia de acceso. Si observamos las gráficas 35 y 36, podemos ver que si es cierto que la frecuencia incrementa el número de fallas notables, estas no son tan significativas para determinar que el sistema incurra en costos comunicacional alto debido a que está localizando el recurso principalmente generando fallas transparentes y simples. Lo anterior permite determinar que el *cache dead* no tiene incidencia sobre el comportamiento de las falla notables que se producen.

2. *La frecuencia de acceso define el número de mensajes y de difusiones fuertes*; a mayor frecuencia de acceso mayor el número de mensajes y de difusiones realizadas; esto ocasiona un incremento en el costo comunicacional debido a que el sistema debe garantizar la recepción del mensaje. (ver graficas 31 y 33).

3. *La frecuencia de acceso determina el número de difusiones y mensajes débiles*. A mayor frecuencia de acceso menor es el número de difusiones y de mensajes que se generan (ver gráficas 32 y 34).

4. *La eficiencia del caché depende solo de la frecuencia de acceso*; a mayor frecuencia mayor la eficiencia sin que sea relevante el tamaño del caché. (ver gráfica 30)

4.4 DISEÑO Y ANÁLISIS DE EXPERIMENTOS PARA DETERMINAR LA EFICIENCIA Y EL COSTO DEL *CACHE STOP*

Para analizar el *cache stop,* se ha diseñado unos experimentos que permitan examinar la eficiencia y el comportamiento del mismo. Se definen los factores objetos de estudio : el tamaño del *cache* (*stop_size)*, el tiempo de invocación del objeto (*obj_inv_iat),* retraso débil de la emisión *(*wea*k_broadcast_delay*), y retraso fuerte de la emisión (*strong_broadcast_delay).*

4.4.1 Experimentos para determinar la utilidad del *stop.* El experimento permite determinar si es útil la activación del *stop*

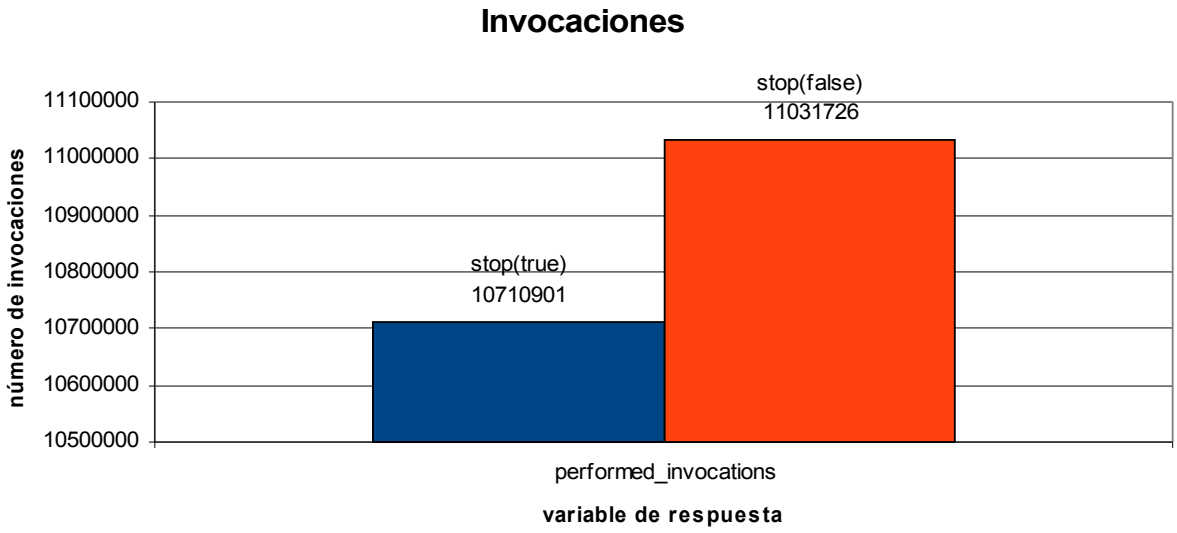
- **Objetivo**. Determinar la utilidad que pueda tener el uso del *cache stop*

- **Simulación a realizar**: Se determinaron los valores *true* y *false* para el factor *with_stop.*

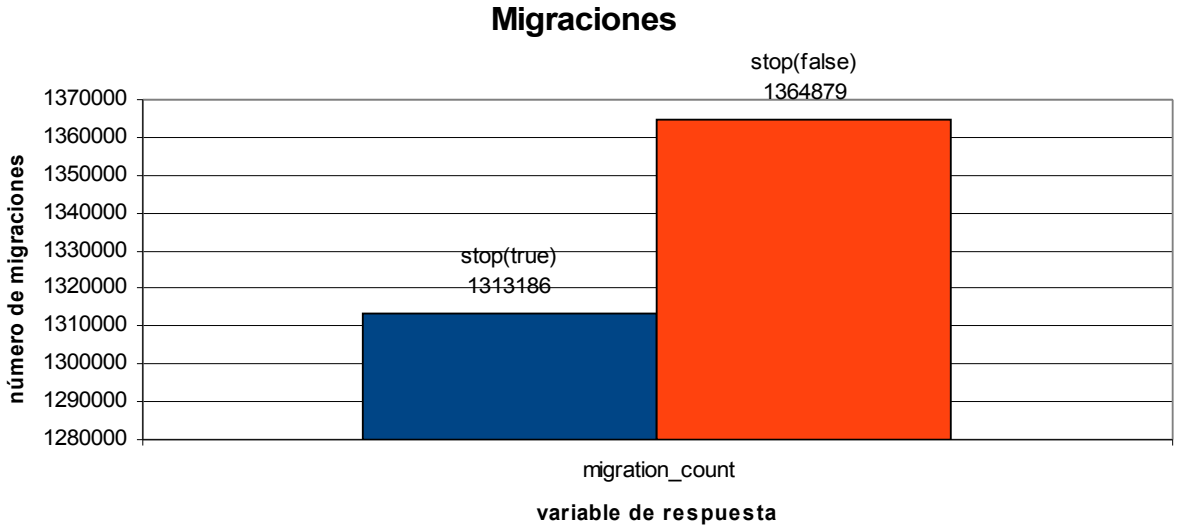
- **Variables de respuesta**. Ejecutado el programa experimento8.ini (Anexo 1) se determinan los valores de las variables de respuesta:

Nested_notable_fails	(total de niveles de fallas notables)
Nested_simple_fails	(total de niveles de fallas notables)
Perfomed_invocatins	(total de invocaciones realizadas)
Migration_count	(total de migraciones realizadas)
reliable_broadcast_sent	*(total de broadcast fuertes)*
simples_fails	*(total de fallas simples)*
notable_fails	*(total de fallas notables)*

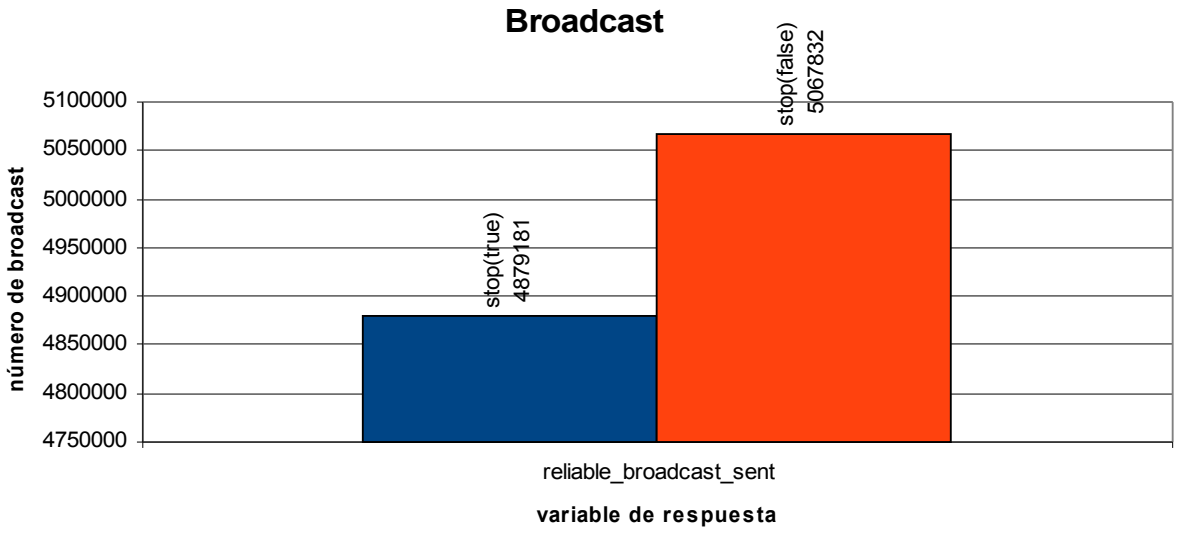
En las gráficas del 37 al 41 se observa el comportamiento de las variables de respuesta.



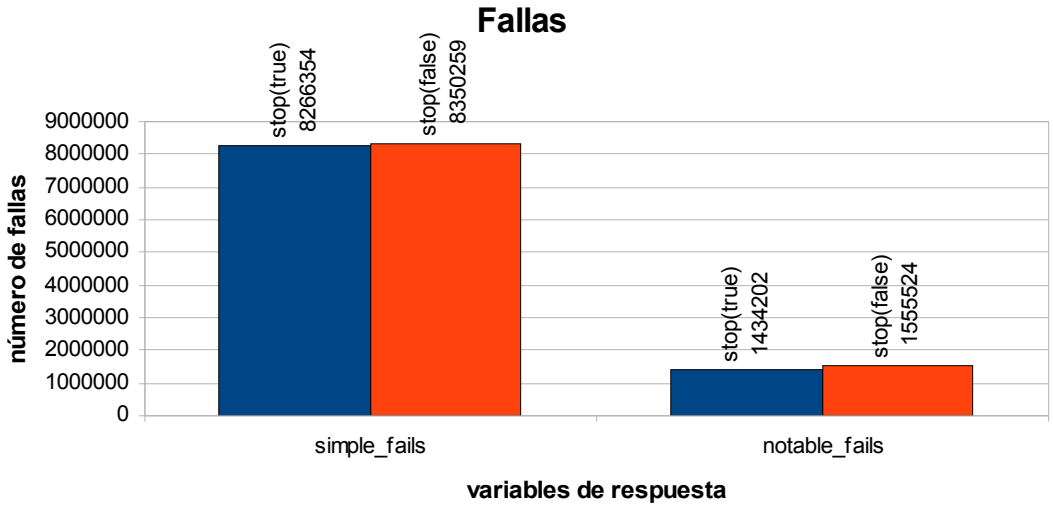
Gráfica 37. Invocaciones realizadas usando el *cache stop*



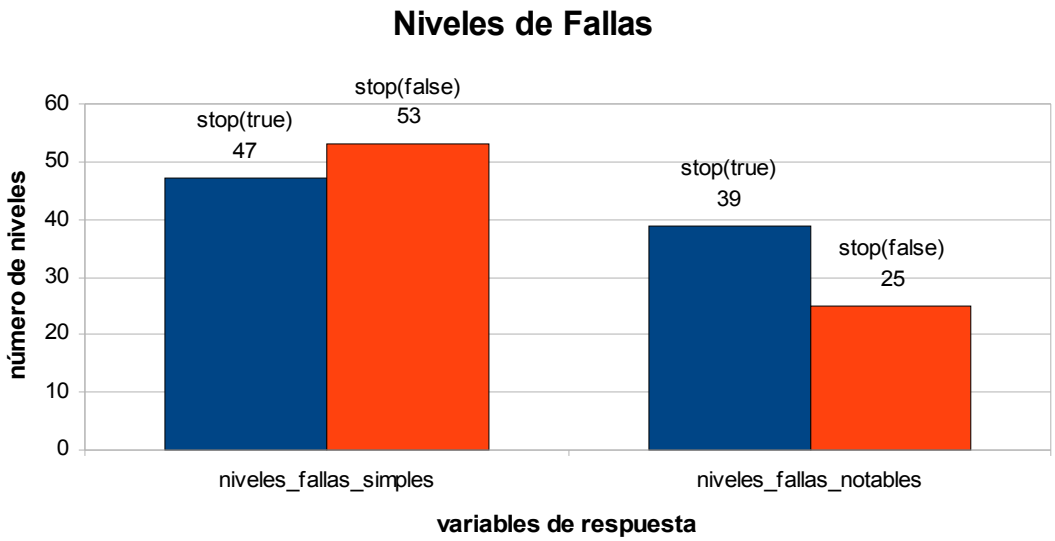
Gráfica 38. Migraciones realizadas usando el *cache stop*



Gráfica 39. *Broadcast* ejecutados usando el *cache stop*



Gráfica 40. Fallas realizadas usando el *cache stop*



Gráfica 41. Niveles de fallas realizadas usando el *cache stop*

- **Análisis de Resultados.** Realizado el análisis correspondiente a las variables de salida se puede precisar las siguientes observaciones:

  1. El *cahe stop* hace que se generen menos invocaciones y migraciones debido a que el objeto es retenido el tiempo definido para estar en el cache, lo cual hace que el objeto sea encontrado en un menor tiempo y por consiguiente no genere costo mayor. (ver gráficas 37 y 38).

  2. Si se activa el *cahe stop,* se generan menos *broadcast* fuertes (ver gráfica 39) ocasionando menos fallas notables y menos niveles de fallas notables (ver gráficas 40 y 41); esto es, el sistema realiza menos difusiones debido a que el recurso permanece quieto en el cache un tiempo definido y no migra rápidamente generando un mayor número de fallas, las cuales son costosas para el sistema.

4.4.2 Experimento para determinar el comportamiento del *cache stop*. El experimento permite analizar el comportamiento que tiene el stop cuando se determina retrasos en la migración del objeto.

- **Objetivo**. Determinar el comportamiento del *cache stop* bajo los factores tiempo de migración del objeto, *obj_mig_iat*; retraso débil de la emisión, *weak_broadcast_delay*; y retraso fuerte de la emisión, *strong_broadcast_delay*

5 **Simulación a realizar**: Se determinan tres (3) niveles para el tiempo de migración del objeto *(obj_mig_iat)* y dos (2) para los tiempos de retraso fuerte *(weak_broadcast_delay)* y retrazo débil del *broadcast (strong_broadcast_delay)*. (ver tabla 13)

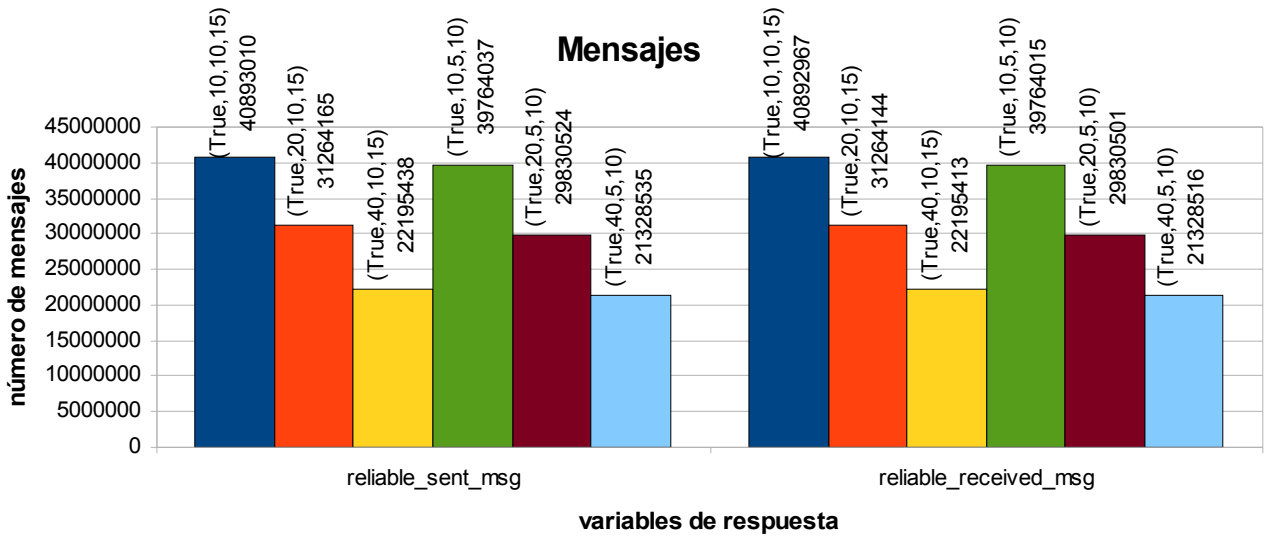
obj_mig_iat	*weak_broadcast_ delay*	*strong_broadcast_ delay*
10s	10s	15s
20s	10s	15s
40s	10s	15s
60s	10s	15s
10s	5s	10s
20s	5s	10s
40s	5s	10s
60s	5s	10s

Tabla 13 . Niveles de los tiempo de migración del objeto del *cache stop*

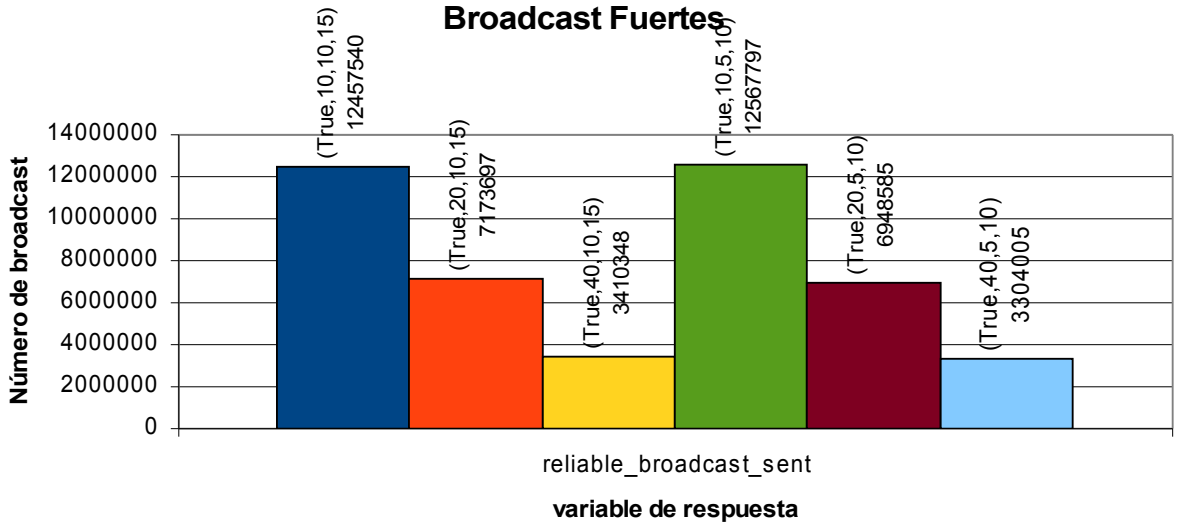
- **Variables de respuesta**. Ejecutado el programas experimento9.ini (Anexo 1) se determinan los valores de las variables de respuesta :

Perfomed_invocatins	(total de invocaciones realizadas)
Expected_perfomen_invocations	(total de invocaciones que se esperan ser ejecutadas)
Reliable_sent_msg	(total de mensajes fuertes enviados)
reliable_received_msg	(total de mensajes débiles enviados)
reliable_broadcast_sent	(total de broadcast fuertes)
unreliable_broadcast_sent	(total de broadcat débiles)
migration_count	(total de migraciones realizadas)
Simple_fails	(total de fallas simples)
Notable_fails	(total de fallas notables)
Nested_simple_fails	(total de niveles de fallas simples)
Nested_notable_fails	(total de niveles de fallas notables)

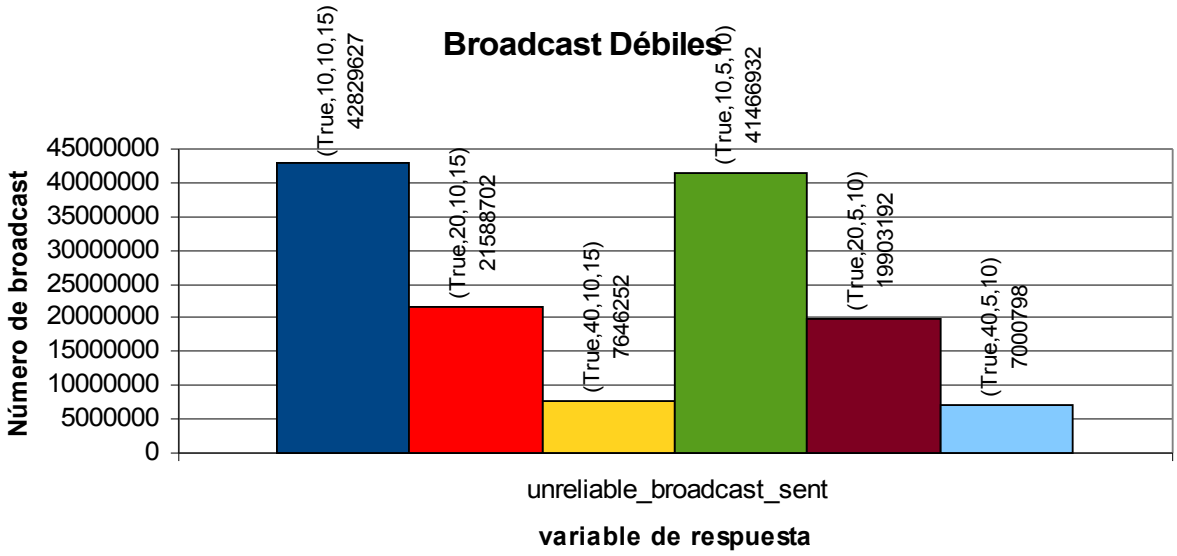
En las gráficas del 42 al 48 se observa el comportamiento de las variables de respuesta.



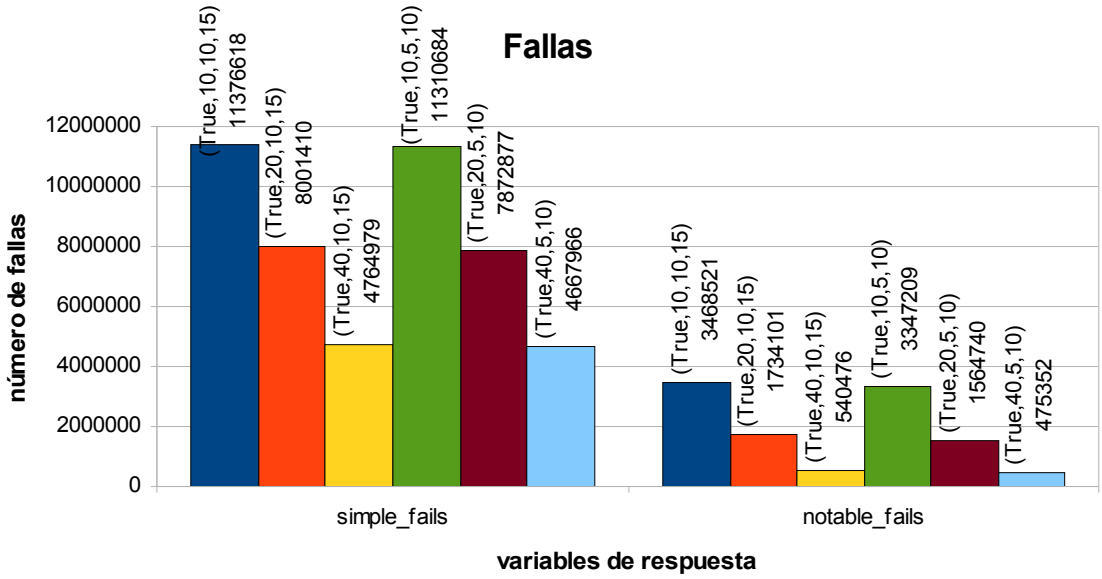
Gráfica 42. Mensajes enviados usando el *cache stop*



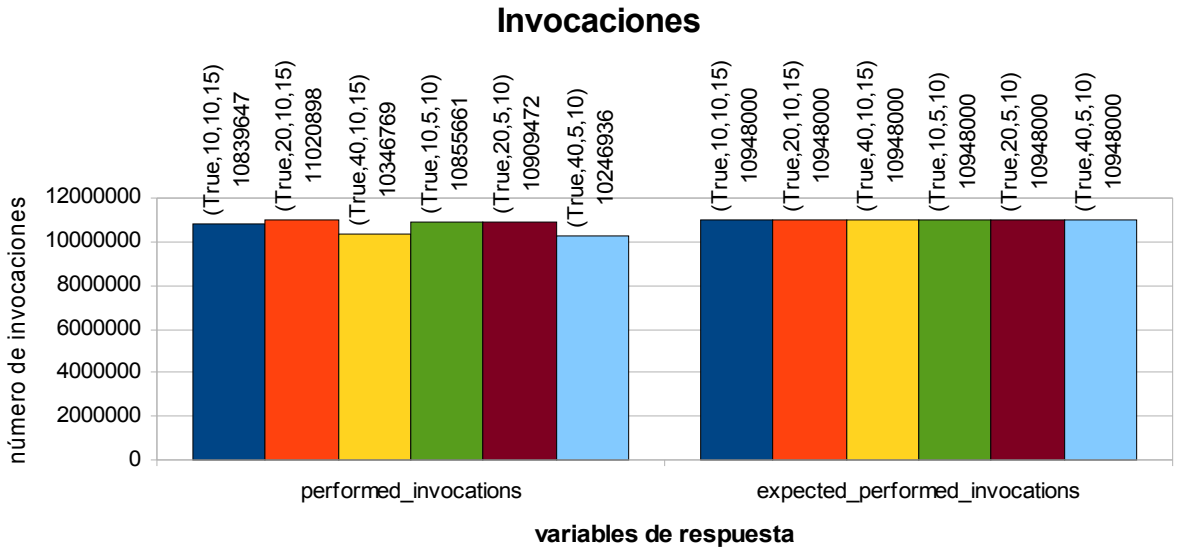
Gráfica 43. *Broadcast* fuertes realizados usando el *cache stop*



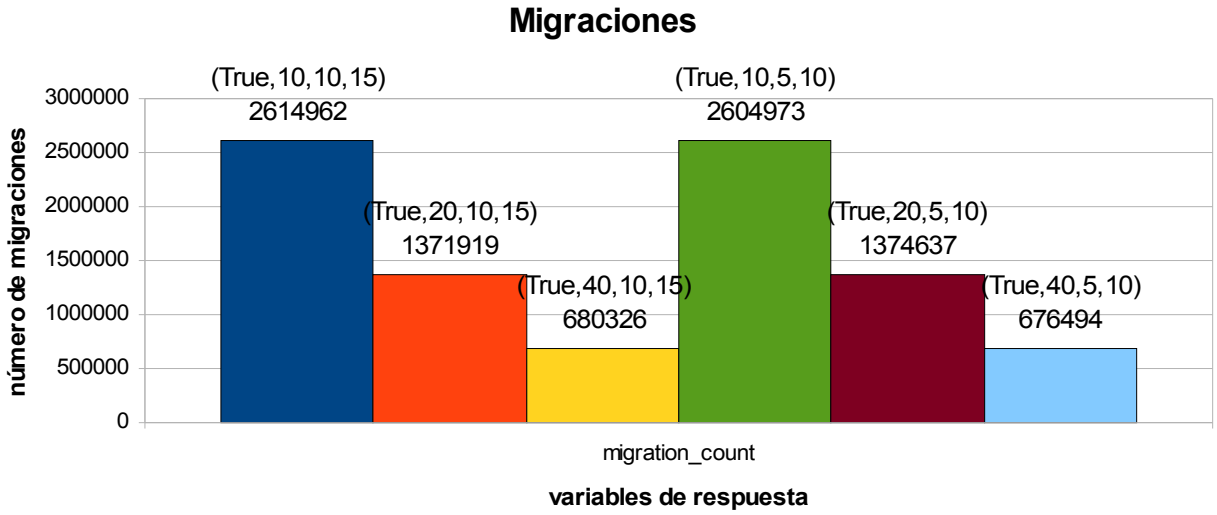
Gráfica 44. *Broadcast* débiles realizados usando el *cache stop*



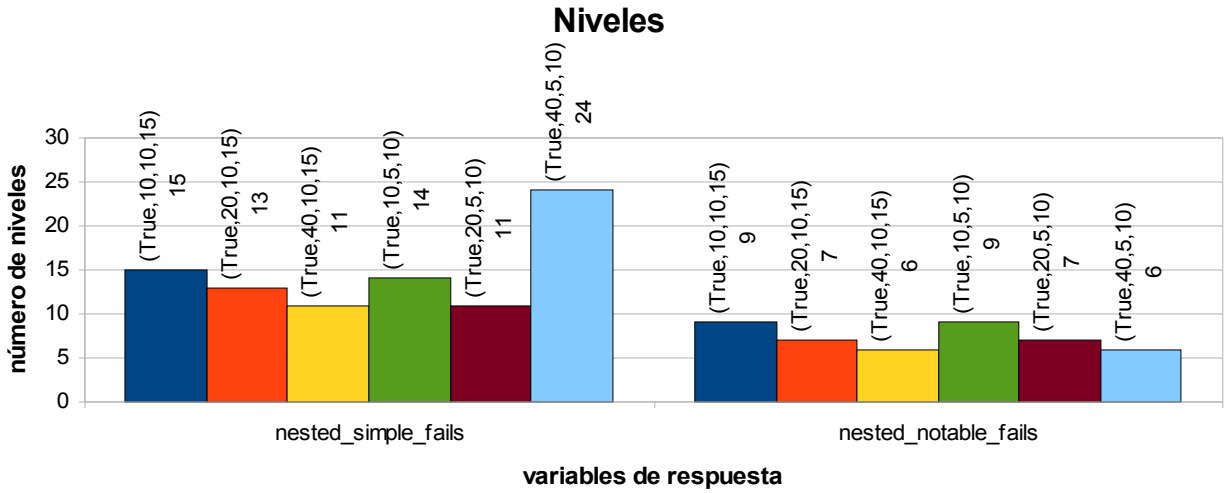
Gráfica 45. Fallas generadas usando el *cache stop*



Gráfica 46. Invocaciones realizadas usando el *cache stop*



Gráfica 47. Migraciones realizados usando el *cache stop*



Gráfica 48. Niveles de fallas generados usando el *cache stop*

- **Análisis de Resultados. .** Realizado el análisis correspondiente a las variables de salida se puede precisar las siguientes observaciones:

1. Cuando el objeto permanece mas tiempo en el se produce menos *broadcast* debido a que el objeto no migra y hay una mayor probabilidad que este pueda ser localizado sin que el sistema tenga que realizar una búsqueda generando menos fallas notables y por consiguiente menos costo. (ver gráficas 43, 44, 45 y 47)

2. Cuando se incrementa el tiempo de retraso débil del *broadcast* (*weak_broadcast_delay*) y el tiempo fuerte del *broadcast* (*strong_broadcast_delay*), el comportamiento del sistema varia en un porcentaje que oscila entre el 0.88 y el 9% aproximadamente, siendo esto no muy significativo; por ejemplo, la diferencia del número de broadcast fuertes realizados teniendo los tiempos del *weak_broadcast_delay* en 10s y 5s y *strong_broadcast_delay* en 15s y 10s es del 3,11%.

4.5 DISEÑO Y ANÁLISIS DE EXPERIMENTOS PARA DETERMINAR LA EFICIENCIA Y EL COSTO DEL USO DEL *PIGGYBACK*

Para analizar el uso del *piggyback,* se han diseñado experimentos que permitan examinar el comportamiento del mismo, donde se definen dos factores objetos de estudio : *with_piggybacks y num_hosts.*

- **Objetivo**. Determinar el comportamiento del *piggyback* bajo los factores *with_piggybacks y num_hosts.*

- **Simulación a realizar**: Se determinan tres (3) niveles para el número de *host* (2) en los estado *true* y *false* del *with_piggyback* (ver tabla 14)

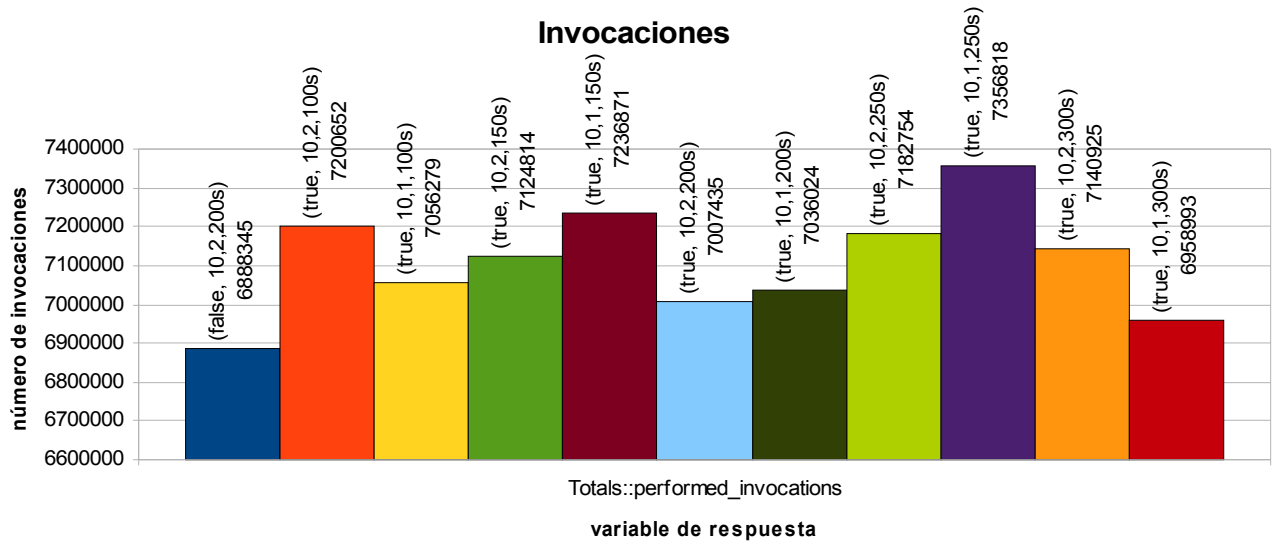
With_piggyback	***Num_host***
true	10
True	20
Trae	40
false	10
false	20
false	40

Tabla 14 . Número de *host* con el *piggyback* activo e inactivo

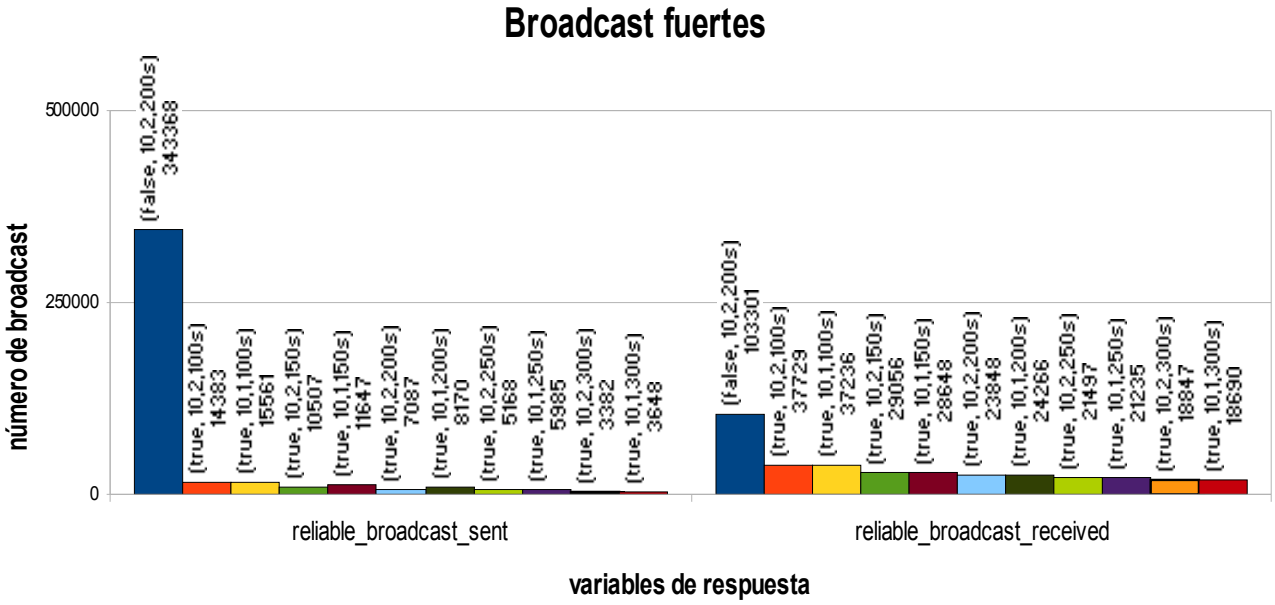
- **Variables de respuesta**. Ejecutado el programas experimento10.ini (Anexo 1) se determinan los valores de las variables de respuesta :

Perfomed_invocatins	(total de invocaciones realizadas)
Reliable_Broadcast_sent	(total de broadcast fuertes enviados)
Reliable_sent_msg	*(total de mensajes fuertes enviados)*
unreliable_sent_msg	*(total de mensajes débiles enviados)*
simples_fails	*(total de fallas simples)*
notable_fails	*(total de fallas notables)*
nested_simples_fails	*(notal de niveles de fallas simples)*
nested_notable_fails	*(notal de niveles de fallas notables)*

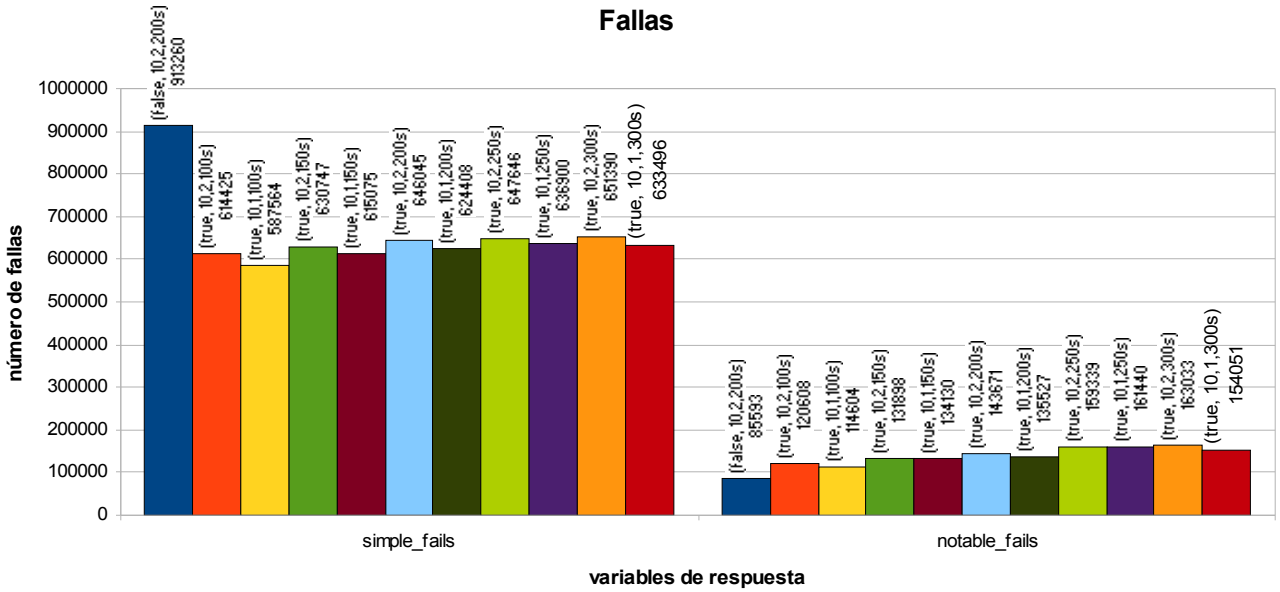
En las gráficas del 49 al 54 se observa el comportamiento de las variables de respuesta.



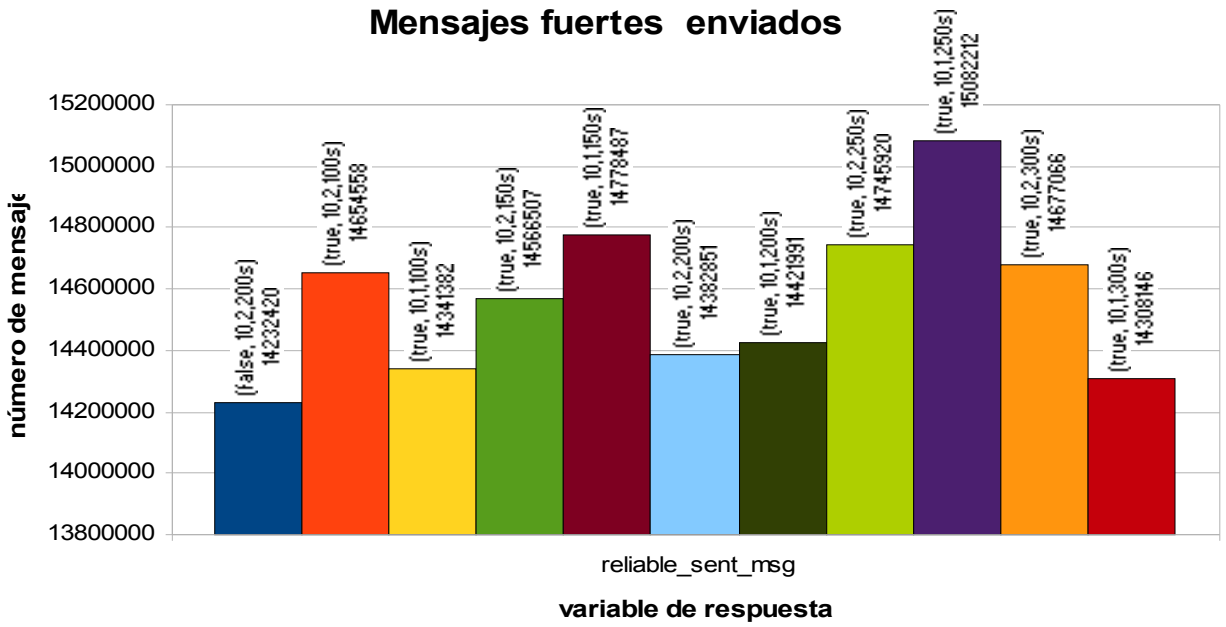
Gráfica 49. Invocaciones realizadas usando el *piggyback*



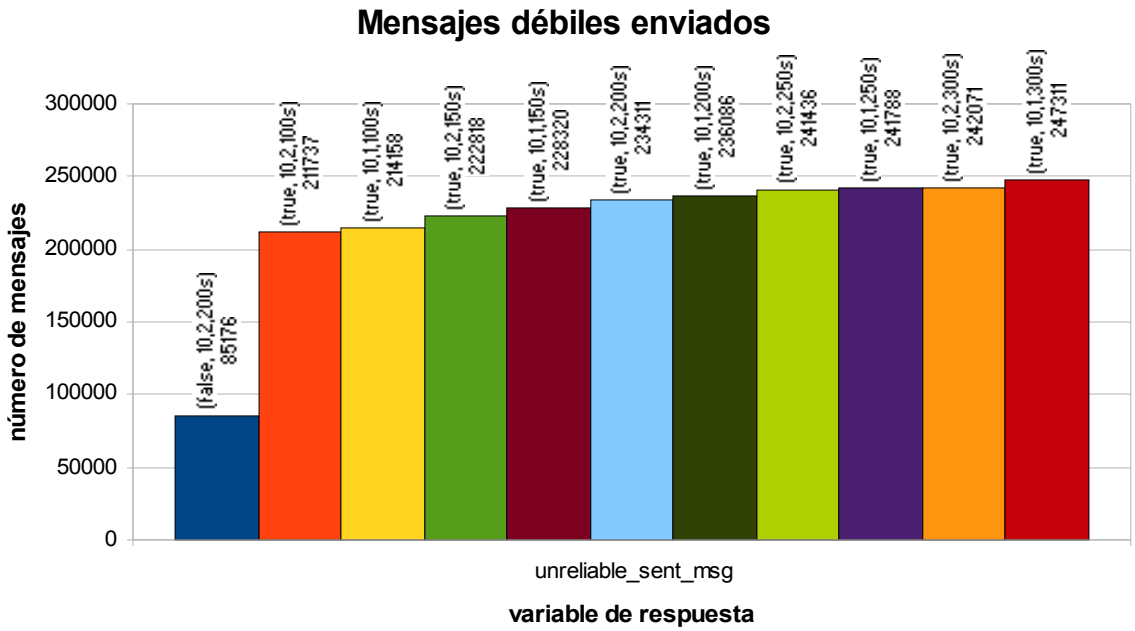
Gráfica 50. *Broadcast* realizados usando el *piggyback*



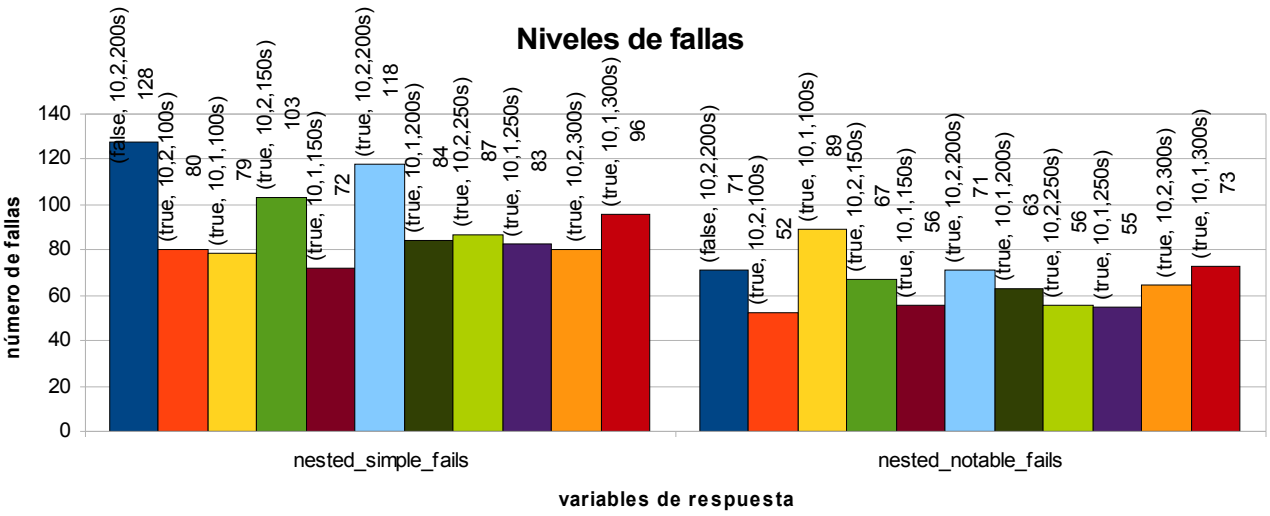
Gráfica 51. Fallas generadas usando el piggyback



Gráfica 52. Mensajes fuertes enviados usando el *piggyback*



Gráfica 53. Mensajes débiles enviados usando el *piggyback*



Gráfica 54. Niveles de fallas usando el *piggyback*

- **Análisis de resultados**. Realizado el análisis correspondiente a las variables de salida se puede precisar las siguientes observaciones:

1. Al estar activado el *piggyback*, se logran mas invocaciones al recurso;

2. Cuando se usa la técnica del *piggyback*, los *broadcast* generados son mucho más bajos en comparación a si el sistema no usa esta técnica. Esto es, el uso del *piggyback* hace que el costo de la difusión sea bajo y permite mantener los caches actualizados.

5. CONCLUSIÓN

Durante el desarrollo del trabajo se identificó:

- Que las técnicas de localización de un objeto implementadas en proyectos anteriores, tales como los sistemas, *System V* ,*Emerald, Amber, Galaxy, DC*$^{++}$, *Soul, Larchant* entre otros, son difusión y lazos de persecución. Estas técnicas presentan inconvenientes como es el caso de la sobrecarga de la red cuando se usa difusión; esta técnica es recomendable solo en sistemas pequeños, donde el número de sitios es bajo, hay pocos objetos y éstos migra poco.

- Si se usa la técnica de lazos de persecución, el problema principal que se presenta es la presencia de dependencias residuales, el cual hace que se disminuya el desempeño y se incrementen las fallas a medida que crece la cadena de lazos.

La contribución de este trabajo se circunscribe en dos ámbitos principales:

1. Un modelo de simulación completamente programado, comprobadamente confiable, que permite estudiar ampliamente las técnicas de localización presentadas: uso de *caches, prefetching y piggybacks*. El modelo es versátil, rápido[11] y contempla la mayoría de factores y variables de respuesta. En este sentido, este modelo es el punto de partida para estudios más profundos de estas técnicas.

2. Una comprobación, sustentada en resultados experimentales, de que las técnicas de localización aquí presentadas son buenas. Esto es, que su uso combinado permite localizar efectivamente recursos móviles, independientemente de su frecuencia de acceso y de movilidad. Esto da pie a futuras investigaciones que se basen en estas técnicas para las futuras generaciones de sistemas distribuidos, las cuales se presume tendrán alta movilidad de recursos.

[11] El modelo permite ser ejecutado en un PC de características regulares con unos tiempos de respuesta aceptables

REFERENCIAS BIBLIOGRÁFICAS

[1] COULOURIS, G., DOLLIMORE, J., KINDBERG, T. *Distributed Systems – Concepts and Design. Second Edition.* Addison-Wesley (1994)

[2] ALOUF S F., ÑAIN P.. Forwarders vs centralized server: An evaluation of two approaches for locating mobile agents (extended abstract). In Proc. Of ACM SIGMETRICS `02, Marina del Rey, Californaia, número special de performance Evaluation Review, volumen 30(1):278-279, june 2002.

[3] ARCIA Andrés, LEÓN Leandro. Localización de objetos distribuidos móviles. Noviembre de 2000.

[4] ARTSY Y, FINKEL R. Designing a process migration facility – the charlotte experience IEEE, 22(9):47-56, 1989.

[5] BAUDE Francoise, COROMEL Denis, HUET Fabrice, and VAYSSIERE Julie. Objets actifs mobiles et communicants. Tecnique et Science Informatiques, 21(6):823-849, 2002.

[6] DOUGLIS Frederik, . OUSTERHOUT Jhon K. Process Migration in the spriteoperating system. In 7th International Conference on Distributed Computer systems, sept 1987.

[7] BAUDE Francoise, COROMEL Denis, HUET Fabrice, and VAYSSIERE Julie.. Communicate mobiles active Objets in java. October 30 2000..

[8] MOSSIÈRE Jacques, CHEVALIER Pierre-Yves, HAGIMOT Daniel and ROUSSET DE PINA Xavier. Le système rèparti à objets Guide. Technical report, 1995.

[9] FERRERIRA Paulo. Larchant: ramasse-miettes dans une mèmoire partagè repartee avec persistence par atteignabilitè. phD thesis, Universitè Paris VI, may 1996.

[10] FOWLER Robert. The complexity of using forwarding addresses for decentralized object finding. In 5th ACM SIGOPS Conference on Principles of Distributed

Computing, page 108-120, aug 1986.

[11] GOSCINSKI A. Distributed Operating systems. The Logical Desing. Addison-Wesley, 1991. ISBN 0-201-41704-9.

[12] JUL Eric, LEVY Henry, HUTCHISON Norman, and BLACK Andrew. Fines-grained mobility in the emerald system. ACM Transaction on Computer System, 6(1):109-113, feb 1988.

[13] LEÓN Leandro. Une architecture pour les systémes repartis á objets mobiles. PhD thesis, Université Paris VI, feb 1998.

[14] MOLOJICIC D., DOUGLIS F., and WHEELER R., editors. Mobility: Processes, Computers, and agents. ACM Press, 1999.

[15] MULLENDER R.J., VAN ROSSUM G.,. TANENBAUM A.S,. Van RESESSE R, and VAN STAVEREN H.. Amoeba: A distributed operating system for the 1990s. IEEE Computer, 23:44-53, apr 1990.

[16] PLAINFOSSÉ David. Distributed Garbage Collection and Referencing Management in the Soul System. PhD thesis, Universitè Paris VI, jun 1994.

[17] POWEL M. and MILLER B.. Process migration in DEMOS/MP. In 9th Symposium on Operating Systems Principles, page 110-119, ct 1983.

[18] SHAPIRO Marc, GOURHENT Ivon, HABERT Sabine, MOSSERI Laurence, RUFFIN Michael, and VALOT Céline. Sos: and object-oriented operating system – assessment and perspectives. Computing System, 2(4): 287-337, 1989

[19] ALOUF S, and ÑAIN P. Forwarders vs centralized server: An evaluation of two approaches for locating mobile agents. Proc. Of performance `02, special issue of performance evaluation, 49:299-319, September 2002.

[20] . STEKETEE C,. ZHU W, and MOSELEY P.. Implementation of process migration in amoeba. In 14th International Conference on Distributed System, pages 194-203, 1994.

[21] THEIMER Marvin, LANTZ Keith, CHERITON David. Preemtable Remote execution facilities for the v-system. In 10 th ACM symposium on Operating System Principles, page 2-14, dec 1985

[22] CHASSE Jeffrey, AMADOR Franz, LAZOWSKA Edward, LEVY Henry, and LITLEFIELD Richard. The Amber system: parallel programming on a network of multiprocessors. In 12th ACM Symposium on Operating System Principles, pages 147-158, dec 1989.

[23] SINHA P, PARK K., JIAN X., SHIMUZE K., and MAEKAWA M. Process migration in the galaxy distributed system. In 5th International parallel processing symposium, pages 611-618, 1991.

[24] SCHILL Alexander and MOCK Markus. DC++ : Distributed object-oriented system support on top of OSF DCE. Distributed System Engineering, pages 112-125, dec 1993.

ANEXO 1

Experimento1.ini

```
[General]
## Experimento1 :  para validar el modelo usando factores con multiplicidad.
network = red
sim-time-limit   = 1600m
cpu-time-limit = 4h
debug-on-errors = true
num-rngs = 8
include seed.ini

[Cmdenv]
runs-to-execute = 0-7
express-mode=yes

[Tkenv]
use-mainwindow = yes
print-banners = yes
breakpoints-enabled = yes
update-freq-fast =
update-freq-express = 500
animation-delay = 0.3s
animation-enabled = yes
animation-msgnames = yes
animation-msgcolors = yes
animation-speed = 1.0


[Parameters]
red.num_hosts = 10
red.with_caching = true
red.latency      = 300ms
red.jitter       = 1.0
red.lost_rate   = 0.1
red.weak_broadcast_delay   = 60s
red.strong_broadcast_delay = 120s

red.pref_type       = 1
red.pref_iat        = 1h
red.pref_duration    = 10m
red.pref_max_entries = 8
red.pref_mig_type       = 1
red.pref_mig_max_entries = 20
red.pref_mig_duration    = 10m

red.piggyback_garbage_period = 30m
```

```
red.max_piggyback_list_len   = 500;
red.piggyback_sample_period  = 30s;
red.piggyback_age_limit      = 5
red.piggyback_duration       = 5m
red.piggyback_rotation       = 2
red.num_piggyback_by_msg     = 3

red.with_stop   = true
red.simple_fails_to_stop  = 5
red.notable_fails_to_stop = 10
red.dlt                   = 1
red.ltc             = 2
red.stop_duration         = 5m
red.obj_hash_size  = 100
red.ref_hash_size  = 200
red.cache_size     = 20
red.host_list_size = 20
red.h[*].obj_mig_iat    = 20m
red.h[*].obj_inv_iat    = 1m
red.h[*].ref_creation_iat = 4m
red.h[*].num_ref_by_obj   = 1

red.pref_type = 0
red.with_piggybacks = false
red.with_migration = false;

[Run 0]
description="a3 < a5 con Multiplicidad; a5 bajo; a3 bajo"
red.h[*].obj_creation_iat  = 2m
red.h[*].obj_duration_time = 8m

[Run 1]
description="a3 < a5 con Multiplicidad; a5 alto; a3 alto"
red.h[*].obj_creation_iat  = 4m
red.h[*].obj_duration_time = 16m

[Run 2]
description="a3 < a5 con Multiplicidad a5 alto; a3 bajo"
red.h[*].obj_creation_iat  = 4m
red.h[*].obj_duration_time = 8m

[Run 3]
description="a3 < a5 con Multiplicidad; a5 bajo; a3 alto"
red.h[*].obj_creation_iat  = 2m
red.h[*].obj_duration_time = 16m

[Run 4]
description="a3 > a5 con Multiplicidad; a5 bajo; a3 bajo"
```

red.h[*].obj_creation_iat = 8m
red.h[*].obj_duration_time = 2m

[Run 5]
description="a3 > a5 con Multiplicidad; a5 alto; a3 alto"
red.h[*].obj_creation_iat = 16m
red.h[*].obj_duration_time = 4m

[Run 6]
description="a3 > a5 con Multiplicidad a5 alto; a3 bajo"
red.h[*].obj_creation_iat = 16m
red.h[*].obj_duration_time = 2m

[Run 7]
description="a3 > a5 con Multiplicidad; a5 bajo; a3 alto"
red.h[*].obj_creation_iat = 8m
red.h[*].obj_duration_time = 4m

Experimento2.ini

[General]
Experimento2 : para validar el modelo usando factores sin multiplicidad.
network = red
sim-time-limit = 7000m
cpu-time-limit = 4h
debug-on-errors = true
num-rngs = 8

include seed.ini

[Cmdenv]
runs-to-execute = 0-7
express-mode=yes

[Tkenv]
use-mainwindow = yes
print-banners = yes
breakpoints-enabled = yes
update-freq-fast =
update-freq-express = 500
animation-delay = 0.3s
animation-enabled = yes
animation-msgnames = yes
animation-msgcolors = yes
animation-speed = 1.0

```
[Parameters]
red.num_hosts = 10
red.with_caching = true
red.latency      = 300ms
red.jitter       = 1.0
red.lost_rate    = 0.1
red.weak_broadcast_delay   = 60s
red.strong_broadcast_delay = 120s

red.pref_type = 0
red.pref_iat         = 1h
red.pref_duration    = 10m
red.pref_max_entries = 8
red.pref_mig_type        = 1
red.pref_mig_max_entries = 20
red.pref_mig_duration    = 10m

red.piggyback_garbage_period = 30m
red.max_piggyback_list_len   = 500;
red.piggyback_sample_period  = 30s;
red.piggyback_age_limit      = 5
red.piggyback_duration             = 5m
red.piggyback_rotation             = 2
red.num_piggyback_by_msg     = 3
red.with_stop                = true
red.simple_fails_to_stop  = 5
red.notable_fails_to_stop = 10
red.dlt                      = 1
red.ltc              = 2
red.stop_duration        = 5m
red.obj_hash_size  = 100
red.ref_hash_size  = 200
red.cache_size     = 20
red.host_list_size = 20
red.h[*].obj_mig_iat    = 20m
red.h[*].obj_inv_iat    = 1m
red.h[*].ref_creation_iat = 2m
red.h[*].num_ref_by_obj   = 1

red.with_piggybacks = false
red.with_migration = false;

[Run 0]
red.h[*].obj_creation_iat  = 2m              # este es a5
red.h[*].obj_duration_time = 5m              # este es a3
[Run 1]
red.h[*].obj_creation_iat  = 3m              # este es a5
red.h[*].obj_duration_time = 7m              # este es a3
```

```
[Run 2]
red.h[*].obj_creation_iat  = 3m          # este es a5
red.h[*].obj_duration_time = 5m          # este es a3

[Run 3]
red.h[*].obj_creation_iat  = 2m          # este es a5
red.h[*].obj_duration_time = 7m          # este es a3
[Run 4]
red.h[*].obj_creation_iat  = 5m          # este es a5
red.h[*].obj_duration_time = 2m          # este es a3

[Run 5]
red.h[*].obj_creation_iat  = 7m          # este es a5
red.h[*].obj_duration_time = 3m          # este es a3

[Run 6]
red.h[*].obj_creation_iat  = 7m          # este es a5
red.h[*].obj_duration_time = 2m          # este es a3

[Run 7]
red.h[*].obj_creation_iat  = 5m          # este es a5
red.h[*].obj_duration_time = 3m          # este es a3
```

Experimento3.ini

```
[General]
## Experimento 3: validar el modelo con diferentes valores aleatorios y multiplicidad
network = red
sim-time-limit          = 16000m     # este es T
cpu-time-limit = 4h
debug-on-errors = true
num-rngs = 8

include seed.ini

[Cmdenv]
runs-to-execute = 0-3
express-mode=yes

[Tkenv]
use-mainwindow = yes
print-banners = yes
breakpoints-enabled = yes
update-freq-fast =
update-freq-express = 500
```

```
animation-delay = 0.3s
animation-enabled = yes
animation-msgnames = yes
animation-msgcolors = yes
animation-speed = 1.0

[Parameters]
red.num_hosts = 10
red.with_caching = true
red.latency      = 300ms
red.jitter       = 1.0
red.lost_rate    = 0.1
red.weak_broadcast_delay   = 60s
red.strong_broadcast_delay = 120s
red.pref_iat         = 1h
red.pref_duration    = 10m
red.pref_max_entries = 8
red.pref_mig_type       = 1
red.pref_mig_max_entries = 20
red.pref_mig_duration    = 10m
red.piggyback_garbage_period = 30m
red.max_piggyback_list_len   = 500;
red.piggyback_sample_period  = 30s;
red.piggyback_age_limit      = 5
red.piggyback_duration           = 5m
red.piggyback_rotation           = 2
red.num_piggyback_by_msg     = 3
red.with_stop                = true
red.simple_fails_to_stop  = 5
red.notable_fails_to_stop = 10
red.dlt                      = 1
red.ltc              = 2
red.stop_duration        = 5m
red.obj_hash_size  = 100
red.ref_hash_size  = 200
red.cache_size     = 20
red.host_list_size = 20

red.h[*].obj_mig_iat    = 20m
red.h[*].obj_inv_iat    = 1m          # este a2
red.h[*].ref_creation_iat = 4m                # este a4
red.h[*].num_ref_by_obj   = 1                 # este es l1

red.pref_type = 0
red.with_piggybacks = false
red.with_migration = false;
```

```
[Run 0]
red.h[*].obj_creation_iat  = 2m              # este es a5
red.h[*].obj_duration_time = 8m              # este es a3
[Run 1]
red.h[*].obj_creation_iat  = 4m              # este es a5
red.h[*].obj_duration_time = 16m   # este es a3
[Run 2]
red.h[*].obj_creation_iat  = 4m              # este es a5
red.h[*].obj_duration_time = 8m              # este es a3
[Run 3]
red.h[*].obj_creation_iat  = 2m              # este es a5
red.h[*].obj_duration_time = 16m   # este es a3
```

Experimento4.ini

```
 [General]
## validar el modelo usando valores sin multiplicidad y diferentes valores aleatorios
network = red
sim-time-limit          = 7000m      # este es T
cpu-time-limit = 4h
debug-on-errors = true
num-rngs = 8

include seed.ini

[Cmdenv]
runs-to-execute = 0-7
express-mode=yes

[Tkenv]
use-mainwindow = yes
print-banners = yes
breakpoints-enabled = yes
update-freq-fast =
update-freq-express = 500
animation-delay = 0.3s
animation-enabled = yes
animation-msgnames = yes
animation-msgcolors = yes
animation-speed = 1.0

[Parameters]
red.num_hosts = 10
red.with_caching = true
red.latency     = 300ms
red.jitter      = 1.0
red.lost_rate   = 0.1
```

```
red.weak_broadcast_delay   = 60s
red.strong_broadcast_delay = 120s

red.pref_iat         = 1h
red.pref_duration    = 10m
red.pref_max_entries = 8

red.pref_mig_type       = 1
red.pref_mig_max_entries = 20
red.pref_mig_duration    = 10m

red.piggyback_garbage_period = 30m
red.max_piggyback_list_len   = 500;
red.piggyback_sample_period  = 30s;
red.piggyback_age_limit      = 5
red.piggyback_duration            = 5m
red.piggyback_rotation            = 2
red.num_piggyback_by_msg     = 3
red.with_stop                = true

red.simple_fails_to_stop  = 5
red.notable_fails_to_stop = 10
red.dlt                   = 1
red.ltc             = 2
red.stop_duration        = 5m
red.obj_hash_size  = 100
red.ref_hash_size  = 200
red.cache_size     = 20
red.host_list_size = 20

red.h[*].obj_mig_iat    = 20m
red.h[*].obj_inv_iat    = 1m          # factor a2
red.h[*].ref_creation_iat = 2m                # factor a4
red.h[*].num_ref_by_obj   = 1                 # este es l1

red.pref_type = 0
red.with_piggybacks = false
red.with_migration = false;

[Run 0]
red.h[*].obj_creation_iat  = 2m               # este es a5
red.h[*].obj_duration_time = 5m               # este es a3

[Run 1]
red.h[*].obj_creation_iat  = 3m               # este es a5
red.h[*].obj_duration_time = 7m               # este es a3

[Run 2]
```

```
red.h[*].obj_creation_iat  = 3m          # este es a5
red.h[*].obj_duration_time = 5m          # este es a3
[Run 3]
red.h[*].obj_creation_iat  = 2m          # este es a5
red.h[*].obj_duration_time = 7m          # este es a3

[Run 4]
red.h[*].obj_creation_iat  = 5m          # este es a5
red.h[*].obj_duration_time = 2m          # este es a3

[Run 5]
red.h[*].obj_creation_iat  = 7m          # este es a5
red.h[*].obj_duration_time = 3m          # este es a3

[Run 6]
red.h[*].obj_creation_iat  = 7m          # este es a5
red.h[*].obj_duration_time = 2m          # este es a3

[Run 7]
red.h[*].obj_creation_iat  = 5m          # este es a5
red.h[*].obj_duration_time = 3m          # este es a3
```

Experimento 5.ini

```
[General]
network = red
sim-time-limit = 4d    # este es T
cpu-time-limit = 4h
debug-on-errors = true
num-rngs = 8

include seed.ini

[Cmdenv]
express-mode=yes

[Tkenv]
use-mainwindow = yes
print-banners = yes
breakpoints-enabled = yes
update-freq-fast =
update-freq-express = 500
animation-delay = 0.3s
animation-enabled = yes
animation-msgnames = yes
animation-msgcolors = yes
animation-speed = 1.0
```

```
[Parameters]
red.latency      = 500ms
red.jitter       = 1.0
red.lost_rate    = 0.1
red.weak_broadcast_delay   = 60s
red.strong_broadcast_delay = 120s

red.pref_type       = 0
red.pref_iat        = 1h
red.pref_duration   = 10m
red.pref_max_entries = 8
red.pref_mig_type       = 0
red.pref_mig_max_entries = 20
red.pref_mig_duration   = 10m

red.piggyback_garbage_period = 30m
red.max_piggyback_list_len   = 500;
red.piggyback_sample_period  = 30s;
red.piggyback_age_limit      = 5
red.piggyback_duration           = 5m
red.piggyback_rotation           = 2
red.num_piggyback_by_msg     = 3
red.with_stop                = false

red.simple_fails_to_stop  = 5
red.notable_fails_to_stop = 10
red.dlt                   = 1
red.ltc             = 2
red.stop_duration       = 5m

red.obj_hash_size  = 100
red.ref_hash_size  = 200

red.h[*].obj_mig_iat   = 30m  #fijo
red.with_piggybacks = false
red.with_migration = true;

red.h[*].obj_creation_iat     =45m          # este es a5
red.h[*].obj_duration_time    =120m         #este es a3
red.h[*].ref_creation_iat     = 12000m      # factor a4
red.host_list_size            = 3
red.h[*].num_ref_by_obj       = 8      # este es l1

red.num_hosts                 =20
red.cache_size                =200
red.dead_size                 =10
red.live_size                 =10
```

```
red.stop_size                =10
red.arriving_size            = 10

[Run 0]
red.with_caching = false
red.h[*].obj_inv_iat         = 10s          # factor a2

[Run 1]
red.with_caching = false
red.h[*].obj_inv_iat         = 15s          # factor a2

[Run 2]
red.with_caching = false
red.h[*].obj_inv_iat         = 20s          # factor a2
[Run 3]
red.with_caching = true
red.h[*].obj_inv_iat         = 10s          # factor a2

[Run 4]
red.with_caching = true
red.h[*].obj_inv_iat         = 15s          # factor a2

[Run 6]
red.with_caching = true
red.h[*].obj_inv_iat         = 20s          # factor a2
```

Experimento6.ini

```
[General]

##experimentos para verificar la funcionalidad del cache arriving

network = red
sim-time-limit = 10d  # este es T
cpu-time-limit = 4h
debug-on-errors = true
num-rngs = 8

include seed.ini

[Cmdenv]
express-mode=yes

[Tkenv]
use-mainwindow = yes
print-banners = yes
breakpoints-enabled = yes
```

update-freq-fast =
update-freq-express = 500
animation-delay = 0.3s
animation-enabled = yes
animation-msgnames = yes
animation-msgcolors = yes
animation-speed = 1.0

[Parameters]
red.with_caching = true
red.latency = 500ms
red.jitter = 1.0
red.lost_rate = 0.1

red.weak_broadcast_delay = 60s
red.strong_broadcast_delay = 120s

red.pref_type = 0
red.pref_iat = 1h
red.pref_duration = 10m
red.pref_max_entries = 8
red.pref_mig_type = 1
red.pref_mig_max_entries = 20
red.pref_mig_duration = 10m

red.piggyback_garbage_period = 30m
red.max_piggyback_list_len = 500;
red.piggyback_sample_period = 30s;
red.piggyback_age_limit = 5
red.piggyback_duration = 5m
red.piggyback_rotation = 2
red.num_piggyback_by_msg = 3
red.with_stop = false

red.simple_fails_to_stop = 5
red.notable_fails_to_stop = 10
red.dlt = 1
red.ltc = 2
red.stop_duration = 5m

red.obj_hash_size = 100
red.ref_hash_size = 200

red.h[*].obj_mig_iat = 30m
red.with_piggybacks = false

red.with_migration = true;

```
red.h[*].obj_creation_iat     =45m          # este es a5
red.h[*].obj_duration_time    =120m         #este es a3
red.h[*].ref_creation_iat     = 12000m      # factor a4
red.num_hostsred.num_hosts = 20
red.host_list_size            = 3
red.h[*].num_ref_by_obj       = 8      # este es l1

red.num_hosts                 =25
red.cache_size                =200
red.dead_size                 =10
red.live_size                 =10
red.stop_size                 =10

#experimentos:  tamaños del cache_arriving= 5,10 y 20; obj_inv_iat=5s y 10s

[Run 0]
red.arriving_size             = 5
red.h[*].obj_inv_iat          = 10s         # factor a2

[Run 1]
red.arriving_size             = 5
red.h[*].obj_inv_iat          = 20s         # factor a2

[Run 2]
red.arriving_size             = 10
red.h[*].obj_inv_iat          = 10s         # factor a2

[Run 3]
red.arriving_size             = 10
red.h[*].obj_inv_iat          = 20s         # factor a2
[Run 4]
red.arriving_size             = 20
red.h[*].obj_inv_iat          = 10s         # factor a2

[Run 5]
red.arriving_size             = 20
red.h[*].obj_inv_iat          = 20s         # factor a2
```

Experimento7.ini

```
[General]
##verificar cahe_live con tamaño 10 y 15; tiempo 10, 15 y 20 seg

network = red
sim-time-limit = 4d    # este es T
cpu-time-limit = 4h
debug-on-errors = true
num-rngs = 8
```

```
include seed.ini

[Cmdenv]
#runs-to-execute = 0-5
express-mode=yes

[Tkenv]
use-mainwindow = yes
print-banners = yes
breakpoints-enabled = yes
update-freq-fast =
update-freq-express = 500
animation-delay = 0.3s
animation-enabled = yes
animation-msgnames = yes
animation-msgcolors = yes
animation-speed = 1.0

[Parameters]

red.with_caching = true
red.latency      = 500ms
red.jitter       = 1.0
red.lost_rate    = 0.1

red.weak_broadcast_delay   = 60s
red.strong_broadcast_delay = 120s

red.pref_type       = 0
red.pref_iat        = 1h
red.pref_duration   = 10m
red.pref_max_entries = 8

red.pref_mig_type       = 0
red.pref_mig_max_entries = 20
red.pref_mig_duration   = 10m

red.piggyback_garbage_period = 30m
red.max_piggyback_list_len   = 500;
red.piggyback_sample_period  = 30s;
red.piggyback_age_limit      = 5
red.piggyback_duration           = 5m
red.piggyback_rotation           = 2
red.num_piggyback_by_msg     = 3
red.with_stop                = false

red.simple_fails_to_stop  = 5
red.notable_fails_to_stop = 10
```

```
red.dlt                 = 1
red.ltc         = 2
red.stop_duration       = 5m

red.obj_hash_size  = 100
red.ref_hash_size  = 200

red.h[*].obj_mig_iat    = 30m  #fijom


red.with_piggybacks = false
red.with_migration = true;

red.h[*].obj_creation_iat  = 20m            # este es a5
red.h[*].obj_duration_time = 40m            #este es a3
red.h[*].ref_creation_iat  = 12000m                 # factor a4

red.host_list_size          = 3
red.h[*].num_ref_by_obj     = 8     # este es l1
red.num_hosts               = 20

red.cache_size              = 200
red.dead_size               = 10
red.stop_size               = 10
red.arriving_size           = 10

[Run 0]
red.live_size               = 10
red.h[*].obj_inv_iat        = 10s           # factor a2

[Run 1]
red.live_size               = 10
red.h[*].obj_inv_iat        = 15s           # factor a2

[Run 2]
red.live_size               = 10
red.h[*].obj_inv_iat        = 20s           # factor a2

[Run 3]
red.live_size               = 15
red.h[*].obj_inv_iat        = 10s           # factor a2

[Run 4]
red.live_size               = 15
red.h[*].obj_inv_iat        = 15s           # factor a2

[Run 5]
red.live_size               = 15
```

red.h[*].obj_inv_iat = 20s # factor a2

Experimento 8

##verificar cahe_stop

[General]

network = red
sim-time-limit = 8d # este es T
cpu-time-limit = 4h
debug-on-errors = true
num-rngs = 8

include seed.ini

[Cmdenv]
#runs-to-execute = 0-5
express-mode=yes

[Tkenv]
use-mainwindow = yes
print-banners = yes
breakpoints-enabled = yes
update-freq-fast =
update-freq-express = 500
animation-delay = 0.3s
animation-enabled = yes
animation-msgnames = yes
animation-msgcolors = yes
animation-speed = 1.0

[Parameters]

red.with_caching = true
red.latency = 500ms
red.jitter = 1.0
red.lost_rate = 0.1

red.pref_type = 0
red.pref_iat = 1h
red.pref_duration = 10m
red.pref_max_entries = 8

red.pref_mig_type = 0
red.pref_mig_max_entries = 20

```
red.pref_mig_duration    = 10m

red.piggyback_garbage_period = 30m
red.max_piggyback_list_len   = 500;
red.piggyback_sample_period  = 30s;
red.piggyback_age_limit      = 5
red.piggyback_duration           = 5m
red.piggyback_rotation           = 2
red.num_piggyback_by_msg     = 3

red.simple_fails_to_stop  = 5
red.notable_fails_to_stop = 10
red.dlt                   = 1
red.ltc           = 2
red.stop_duration       = 20s

red.obj_hash_size  = 100
red.ref_hash_size  = 200

red.with_piggybacks = false
red.with_migration = true;

red.h[*].obj_creation_iat  = 20m              #este es a5
red.h[*].obj_duration_time = 40m              #este es a3
red.h[*].ref_creation_iat  = 12000m           # factor a4

red.host_list_size              = 3
red.h[*].num_ref_by_obj      = 8      # este es l1


red.cache_size                  = 200
red.dead_size                   = 10
red.arriving_size               = 10
red.live_size                   = 10
red.num_hosts                   = 20
red.stop_size                   = 10

red.h[*].obj_mig_iat    = 20s
red.h[*].obj_inv_iat    = 20s

[Run 0]
red.with_stop               = true
red.weak_broadcast_delay   = 60s
red.strong_broadcast_delay = 120s

[Run 1]
red.with_stop               = true
red.weak_broadcast_delay   = 10s
```

red.strong_broadcast_delay = 20s

[Run 2]
red.with_stop = false
red.weak_broadcast_delay = 60s
red.strong_broadcast_delay = 120s

[Run 3]
red.with_stop = false
red.weak_broadcast_delay = 10s
red.strong_broadcast_delay = 20s

Experimento9.ini

[General]
##verificar cahe_stop variando week y el strong

network = red
sim-time-limit = 8d # este es T
cpu-time-limit = 4h
debug-on-errors = true
num-rngs = 8

include seed.ini

[Cmdenv]
#runs-to-execute = 0-5
express-mode=yes

[Tkenv]
use-mainwindow = yes
print-banners = yes
breakpoints-enabled = yes
update-freq-fast =
update-freq-express = 500
animation-delay = 0.3s
animation-enabled = yes
animation-msgnames = yes
animation-msgcolors = yes
animation-speed = 1.0

[Parameters]

red.with_caching = true
red.latency = 500ms
red.jitter = 1.0

```
red.lost_rate   = 0.1


red.pref_type       = 0
red.pref_iat        = 1h
red.pref_duration   = 10m
red.pref_max_entries = 8

red.pref_mig_type       = 0
red.pref_mig_max_entries = 20
red.pref_mig_duration   = 10m

red.piggyback_garbage_period = 30m
red.max_piggyback_list_len   = 500;
red.piggyback_sample_period  = 30s;
red.piggyback_age_limit      = 5
red.piggyback_duration           = 5m
red.piggyback_rotation           = 2
red.num_piggyback_by_msg     = 3


red.simple_fails_to_stop  = 5
red.notable_fails_to_stop = 10
red.dlt                   = 1
red.ltc           = 2
red.stop_duration       = 20s

red.obj_hash_size  = 100
red.ref_hash_size  = 200


red.with_piggybacks = false
red.with_migration = true;

red.h[*].obj_creation_iat  = 20m            #este es a5
red.h[*].obj_duration_time = 40m            #este es a3
red.h[*].ref_creation_iat  = 12000m         # factor a4

red.host_list_size              = 3
red.h[*].num_ref_by_obj     = 8     # este es l1


red.cache_size              = 200
red.dead_size               = 10
red.arriving_size                   = 10
red.live_size               = 10
red.num_hosts               = 20
red.stop_size               = 10
```

red.h[*].obj_inv_iat = 20s

red.with_stop = true

[Run 0]
red.h[*].obj_mig_iat = 10s
red.weak_broadcast_delay = 10s
red.strong_broadcast_delay = 15s

[Run 1]
red.h[*].obj_mig_iat = 20s
red.weak_broadcast_delay = 10s
red.strong_broadcast_delay = 15s

[Run 2]
red.h[*].obj_mig_iat = 40s
red.weak_broadcast_delay = 10s
red.strong_broadcast_delay = 15s

[Run 3]
red.h[*].obj_mig_iat = 60s
red.weak_broadcast_delay = 10s
red.strong_broadcast_delay = 15s

[Run 4]
red.h[*].obj_mig_iat = 10s
red.weak_broadcast_delay = 5s
red.strong_broadcast_delay = 10s

[Run 5]
red.h[*].obj_mig_iat = 20s
red.weak_broadcast_delay = 5s
red.strong_broadcast_delay = 10s

[Run 6]
red.h[*].obj_mig_iat = 40s
red.weak_broadcast_delay = 5s
red.strong_broadcast_delay = 10s

[Run 7]
red.h[*].obj_mig_iat = 60s
red.weak_broadcast_delay = 5s
red.strong_broadcast_delay = 10s

Experimento 10.ini

```
[General]
##verificar piggybacks

network = red
sim-time-limit = 8d    # este es T
cpu-time-limit = 4h
debug-on-errors = true
num-rngs = 8

include seed.ini

[Cmdenv]
#runs-to-execute = 0-5
express-mode=yes

[Tkenv]
use-mainwindow = yes
print-banners = yes
breakpoints-enabled = yes
update-freq-fast =
update-freq-express = 500
animation-delay = 0.3s
animation-enabled = yes
animation-msgnames = yes
animation-msgcolors = yes
animation-speed = 1.0

[Parameters]
red.weak_broadcast_delay   = 60s
red.strong_broadcast_delay = 120s

red.with_caching = true
red.latency      = 500ms
red.jitter       = 1.0
red.lost_rate    = 0.1

red.pref_type      = 0
red.pref_iat       = 1h
red.pref_duration    = 10m
red.pref_max_entries = 8
red.pref_mig_type      = 0
red.pref_mig_max_entries = 20
red.pref_mig_duration    = 10m

# variable piggyback
```

```
red.piggyback_garbage_period = 1m
red.max_piggyback_list_len   = 50;
red.piggyback_sample_period  = 10s;
red.piggyback_age_limit      = 5
red.piggyback_duration           = 30s
red.piggyback_rotation           = 2
red.num_piggyback_by_msg     = 3

red.simple_fails_to_stop  = 5
red.notable_fails_to_stop = 10
red.dlt                      = 1
red.ltc             = 2
red.stop_duration        = 20s

red.obj_hash_size  = 100
red.ref_hash_size  = 200

red.with_migration = true;

red.h[*].obj_creation_iat  = 20m              #45m          2; este es a5
red.h[*].obj_duration_time = 40m              #120m                 8; este es a3
red.h[*].ref_creation_iat  = 12000m                    # factor a4

red.host_list_size               = 3
red.h[*].num_ref_by_obj      = 8      # este es l1


red.cache_size                   = 200
red.dead_size                    = 10
red.arriving_size                = 10
red.live_size                    = 10

red.stop_size                    = 10

red.h[*].obj_mig_iat    = 200s
red.h[*].obj_inv_iat     = 20s

red.with_stop                = true
red.piggyback_rotation           = 2
red.piggyback_duration           = 10s
red.num_piggyback_by_msg     = 3

[Run 0]
red.with_piggybacks = true
red.num_hosts       = 10

[Run 1]
```

red.with_piggybacks = true
red.num_hosts = 20

[Run 2]
red.with_piggybacks = true
red.num_hosts = 40

[Run 3]
red.with_piggybacks = false
red.num_hosts = 10

[Run 4]
red.with_piggybacks = false
red.num_hosts = 20

[Run 5]
red.with_piggybacks = false
red.num_hosts = 40