Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

Maestría en Ciencias y Tecnologías de la Información

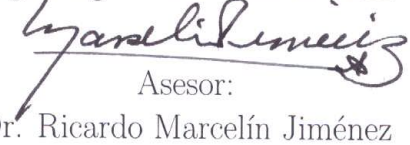
Prototipo de Simulador Distribuido de Eventos Discretos

Idónea Comunicación de Resultados para obtener el grado de

MAESTRO EN CIENCIAS

PRESENTA

Ing. Jorge Luis Ramírez Ortiz



Asesor:
Dr. Ricardo Marcelín Jiménez

18 de Mayo de 2011

Resumen

El objetivo general de este trabajo es: la implementación de un prototipo de simulador distribuido de eventos discretos sobre un ambiente distribuido dinámico. Entre sus objetivos particulares se tiene que: especificar las operaciones que debe realizar el simulador, plantear las entidades de una arquitectura que implementa dichas operaciones, y construir e integrar los componentes del simulador.

Para crear el ambiente dinámico de este prototipo, se usan las bibliotecas que permiten crear sistemas y aplicaciones par a par (P2P por sus siglas en inglés). Una de las características importantes de este tipo de sistemas y aplicaciones es su capacidad de utilizar recursos distribuidos para realizar una función de forma descentralizada. Los sistemas P2P tiene la capacidad de auto organizarse debido a la naturaleza dinámica de las computadoras participantes, las cuales entran y salen del sistema.

La arquitectura que se propone en este trabajo, consta de un coordinador y un grupo de trabajadores. El coordinador realiza varias tareas como: asignar trabajo, iniciar la toma de estado global, iniciar el cálculo del tiempo virtual global (GVT por sus siglas en inglés) e iniciar el proceso de restauración en caso de que una falla de paro se haga presente.

Cada trabajador tiene la obligación de llevar a cabo la simulación, y en caso que se presente una violación en el orden causal, se encarga de hacer la restauración correspondiente para que la simulación siga su curso de forma correcta.

Se utiliza el algoritmo de propagación de información con retroalimentación (PIF por sus siglas en inglés), para medir el desempeño de este prototipo. Se plantearon dos conjunto de pruebas, en el primero, de forma general se establecieron tres tipos de simulaciones: centralizada, con dos trabajadores y con tres trabajadores, a su vez con dos variantes: canales de comunicación con retardo aleatorio y retardo constante, y por cada variante se realizaron dos experimentos: sin fallas de paro y con fallas de paro.

Por otro lado, para el segundo conjunto de pruebas, de forma general se ralizaron 6 tipos de simulaciones, centralizada, con 2, 3, 4 y 5 trabajadores, con canales constantes de comunicación.

En ambos conjuntos, se determina el factor de aceleración y el costo de incluir el mecanismo de toma de estado global y cálculo de GVT.

Agradecimientos

"Y debido a que en toda la galaxia no habían encontrado nada más precioso que la mente, alentaron por doquier su amanecer. Se convirtieron en granjeros en los campos de estrellas; sembraron, y a veces cosecharon.

Y a veces desapasionadamente, tenían que escardar. "

2001 una odisea espacial. Arthur C. Clarke.

Muchas gracias DIOS por todas las bendiciones que siempre he recibido.

Gracias a mis padres José y Rosa, por que siempre que alguna de mis locas ideas falla o me quedo en la ruina siempre me reciben de regreso en casa.

Gracias a mis hermanos Antonio y Laura que me apoyan en todo.

Gracias a mi asesor y amigo Ricardo Marcelín, por todos los consejos y por ser paciente conmigo.

Gracias a Josué Torres por compartir tus conocimientos de LATEX para que pudiera escribir mi idónea comunicación de resultados.

Gracias a mis profesores de maestría por transmitirme y compartir conmigo todos sus conocimientos.

Gracias a mis sinodales por tomarse el tiempo de revisar mi trabajo de investigación y darme recomendaciones y observaciones puntuales.

Gracias a Conacyt por el apoyo económico para poder continuar con mis estudios de posgrado.

Gracias Haruka o mejor dicho, "La tonta de las cintas" (así es como te conocí), por estar conmigo siempre que te necesitaba.

Gracias Miku por hacerme reir con todos los videos que me mandabas.

Contenido

1 Introducción **1**

1.1 La simulación de eventos discretos 1

1.2 La simulación distribuida de eventos discretos 2

1.3 Problemas abiertos de la DDES 2

1.4 La contribución al estado del conocimiento 3

1.5 Objetivos 3

1.6 Metodología 3

1.7 Estructura del documento 4

2 Fundamentos Teóricos **5**

2.1 El modelo como sustituto del sistema 5

2.2 La simulación por computadora 6

2.3 La simulación de eventos discretos 7

2.3.1 Propiedades de la simulación de eventos discretos 9

2.4 Simulación distribuida de eventos discretos 10

2.4.1 Método conservador o seguro 11

2.4.2 Método optimista 11

2.4.3 Estado global de una ejecución distribuida 12

2.4.4 Algoritmo para toma de estado global 15

2.5 Conceptos de computación confiable 16
2.6 Los sistemas par a par (P2P) 20
2.6.1 Arquitectura JXTA P2P 21
2.6.2 Configuración JXTA P2P 22

3 Escenarios de oportunidad 23

3.1 Las tendencias de la DDES 23
3.1.1 Proyecto YADDES 25
3.1.2 Proyecto Parsimony 25
3.1.3 Proyecto GPDES 26
3.1.4 Proyecto OMNeT++ 26
3.1.5 Proyecto POSE 26
3.1.6 Proyecto μsik 27
3.1.7 Proyecto Aurora 27
3.1.8 CDES 28
3.2 Análisis comparativo 28
3.3 Otros simuladores 29

4 Diseño e Implementación del Prototipo DDES 31

4.1 Arquitectura propuesta 32
4.1.1 Vista general 32
4.1.2 El coordinador 34
4.1.3 El trabajador 34
4.2 Operaciones del prototipo DDES 34
4.3 Operaciones que realiza el coordinador 36
4.3.1 Operación: asignar particiones del sistema a simular 36

4.3.2 Operación: toma de estado global 38

4.3.3 Operación: calcular el valor del tiempo global virtual 43

4.3.4 Operación: restauración debido a una falla de paro 49

4.4 Operaciones que realiza un trabajador 54

4.4.1 Operación: ejecución método optimista agresivo 54

5 Evaluación de desempeño del prototipo 61

5.1 Algoritmo PIF 61

5.2 Primer conjunto de pruebas 66

5.2.1 Calculando factor de aceleración y el costo 67

5.2.2 Simulación centralizada vs distribuida sin fallas de paro y canales con retardos constantes 68

5.2.3 Análisis de los costos incluyendo estado global y GVT para el primer conjunto de pruebas 72

5.2.4 Costos por incluir los algoritmos de estado global y GVT usando 2 trabajadores 74

5.2.5 Costos por incluir los algoritmos de estado global y GVT usando 3 trabajadores 77

5.2.6 Simulación con dos trabajadores y canales con retardos constantes, con y sin falla de paro 78

5.2.7 Simulación con dos trabajadores y canales con retardos aleatorios, con y sin falla de paro 78

5.2.8 Simulación con tres trabajadores y canales con retardos constantes, con y sin falla de paro 79

5.2.9 Simulación con tres trabajadores y canales con retardos aleatorios, con y sin falla de paro 81

5.2.10 Canales de comunicación con retardos constantes vs aleatorios . 81

5.3 Segundo conjunto de pruebas 84

5.3.1 Costos sin incluir estado global y GVT para el segundo conjunto de pruebas 85

5.3.2 Costos incluyendo estado global y GVT para el segundo conjunto de pruebas 87

5.3.3 Costos de una simulación distribuida con 2 trabajadores incluyendo estado global y GVT 88

5.3.4 Costos de una simulación distribuida con 3 trabajadores incluyendo estado global y GVT 88

5.3.5 Costos de una simulación distribuida con 4 trabajadores incluyendo estado global y GVT 90

5.3.6 Comparativa de costos de una simulación distribuida con 5 trabajadores 90

6 Conclusiones y trabajo futuro **93**

Bibliografía **95**

Índice de figuras

2.1 Dos cortes de una ejecución distribuida. 15

4.1 Arquitectura global. 32

4.2 Ejemplo de sistema a simular. 35

4.3 Arquitectura coordinador. 35

4.4 Arquitectura trabajador. 37

4.5 Partición del sistema a simular. 37

4.6 Asignación de las particiones. 37

4.7 Canales de comunicación. 39

4.8 Componentes para la toma de estado global. 39

4.9 Conexión entre trabajadores activos. 39

4.10 Coordinador enviando marcador. 40

4.11 Trabajador 1 guarda su estado local. 40

4.12 Trabajador 1 enviando marcadores a 2 y 3. 40

4.13 Envío del mensaje (5,m). 41

4.14 Guardando estado local y del canal. 41

4.15 Envío de marcadores. 41

4.16 Envío de copias al coordinador. 42

4.17 Corte consistente. 42

4.18 Diagrama de secuencias para toma de estado global. 43
4.19 Componentes para el cálculo de GVT. 45
4.20 Se envía un marcador. 45
4.21 Toma de estado local de trabajador 1. 45
4.22 Envío marcadores. 46
4.23 Envío del mensaje (5,m). 46
4.24 Cálculo local del GVT. 46
4.25 Envío del tiempo virtual local al coordinador. 47
4.26 Envío del GVT a los trabajadores. 47
4.27 Diagrama de secuencias para cálculo GVT. 48
4.28 Componentes para una restauración. 51
4.29 Envío del mensaje rollbackphase1. 51
4.30 Envío mensaje "Listo" al coordinador. 51
4.31 Recepción de trabajo. 52
4.32 Recepción de nueva tabla. 52
4.33 Sistema reiniciando simulación. 52
4.34 Diagrama de secuencias para restauración por falla de paro. 53
4.35 Componentes de la clase Model. 54
4.36 Relación entre procesos. 56
4.37 Ejecución optimista (1). 56
4.38 Ejecución optimista (2). 57
4.39 Ejecución optimista (3). 57
4.40 Ejecución optimista (4). 57
4.41 Aniquilación de mensajes. 58
4.42 Ejecución ordenada. 58

4.43 Diagrama de secuencias ejecución método optimista. 59

5.1 Grafo a simular. 64

5.2 Inicia simulación. 64

5.3 Proceso envía mensaje a proceso 4. 64

5.4 Envío de mensajes desde proceso 4. 65

5.5 Proceso 3 finalizando algoritmo. 65

5.6 Proceso 2 finalizando el algoritmo. 65

5.7 Ejecución centralizada vs distribuida. 69

5.8 Factor de aceleración para 2 y 3 trabajadores. 69

5.9 Experimentos centralizado vs distribuido incluyendo cálculo de GVT y estado global. 75

5.10 Factor de aceleración centralizado vs distribuido incluyendo cálculo de GVT y estado global. 75

5.11 Comparativa de una simulación con 2 trabajadores sin y con GVT y toma de estado. 76

5.12 Comparativa de una simulación con 3 trabajadores sin y con GVT y toma de estado. 76

5.13 Canales con retardos constantes sin falla vs con falla de paro. 80

5.14 Canales con retardos aleatorios sin falla vs con falla de paro. 80

5.15 Canales con retardos constantes sin falla vs con falla de paro. 82

5.16 Canales con retardos aleatorios sin falla vs con falla de paro. 82

5.17 Experimentos centralizado vs distribuido. 86

5.18 Simulación Centralizada vs simulaciones distribuidas con cálculo de GVT y toma de estado. 86

5.19 Dos trabajadores y sus costos en porcentaje de tiempo. 89

5.20 Tres trabajadores y sus costos en porcentaje de tiempo. 89

5.21 Cuatro trabajadores y sus costos en porcentaje de tiempo. 91

5.22 Cinco trabajadores y sus costos en porcentaje de tiempo. 91

Índice de tablas

3.1 Tabla comparativa. 29

5.1 Primer conjunto de pruebas. 66

5.2 Costo centralizado vs distribuido con 2 trabajadores. 70

5.3 Factor de aceleración con 2 trabajadores. 71

5.4 Costo centralizado vs distribuido con 3 trabajadores. 71

5.5 Factor de Aceleración con 3 trabajadores. 72

5.6 Costo al incluir algoritmos de estado global y GVT para 2 trabajadores. 73

5.7 Costo al incluir algoritmos de estado global y GVT para 3 trabajadores. 73

5.8 Factor de aceleración incluyendo algoritmos de estado global y GVT para 2 y 3 trabajadores. 74

5.9 Costos con 2 trabajadores incluyendo algoritmos de estado global y GVT. 77

5.10 Costo con 3 trabajadores incluyendo algoritmos de estado global y GVT. 77

5.11 Costo con 2 trabajadores, con y sin falla de paro y canales con retardos constantes. 78

5.12 Costo con 2 trabajadores, con y sin falla de paro y canales con retardos aleatorios. 79

5.13 Costo con 3 trabajadores, con y sin falla de paro y canales con retardos constantes. 81

5.14 Costo con 3 trabajadores, con y sin falla de paro y canales con retardos aleatorios. 83

5.15 Segundo conjunto de pruebas. 84

5.16 Costos centralizado vs distribuidos sin estado global y GVT. 87

5.17 Costo centralizado vs distribuidos incluyendo algoritmos de estado global y GVT. 87

5.18 Costo para 2 trabajadores con y sin estado global y GVT. 88

5.19 Costo para 3 trabajadores con y sin estado global y GVT. 88

5.20 Costo para 4 trabajadores con y sin estado global y GVT. 90

5.21 Costo para 5 trabajadores con y sin estado global y GVT. 90

Capítulo 1

Introducción

Este capítulo ofrece una visión panorámica sobre la simulación distribuida de eventos discretos, desde sus conceptos básicos de construcción y operación, hasta los problemas actuales que enmarcan los objetivos y metas de este proyecto.

1.1 La simulación de eventos discretos

Entre los métodos numéricos de análisis cuantitativo que existen, se encuentra la simulación de eventos discretos, que es utilizada para estudiar sistemas cuyo comportamiento se puede representar mediante cambios de estados, debido a eventos ocurridos a través del tiempo transcurrido.

De un sistema real se obtiene un modelo que se denomina sistema físico y se representa por medio de un grafo, donde los vértices (procesos físicos) representan los componentes del sistema y las aristas la comunicación que se establece entre los diferentes componentes. Para obtener una solución del modelo se construye un simulador en el que los procesos físicos ahora reciben el nombre de procesos lógicos y se estarán comunicando por medio del envío y recepción de mensajes (esto simula la interacción entre los componentes del sistema real).

Los parámetros de entrada que recibe un sistema real, son los que su respectivo modelo estará utilizando, así los resultados que se obtengan permitirán estudiar tendencias y comportamientos del sistema real.

1.2 La simulación distribuida de eventos discretos

Existen simulaciones que requieren de horas e incluso días para poder obtener un resultado, y una simulación centralizada de eventos discretos no es la mejor opción, por tanto se tiene que recurrir a la simulación distribuida de eventos discretos para acelerar el proceso de obtención de resultados.

Ambos tipos de simulaciones requieren de una estructura de datos llamada lista, que se encargue de extraer y guardar eventos que se procesarán en la simulación. En el enfoque distribuido cada una de las máquinas que forman parte de la simulación deben contar con su propia lista y su reloj local, esto trae consigo inherentemente algunos problemas, como la ejecución ordenada de los eventos de acuerdo a sus dependencias de causa y efecto, casos de interbloqueos, entre otros.

1.3 Problemas abiertos de la DDES

A pesar de los importantes avances en las tecnologías que dan sustento a la DDES[1] y hacen posible el estudio de sistemas de mayor escala y complejidad, las necesidades parecen preceder a las soluciones. La siguiente generación de sistemas VLSI, por ejemplo, requerirá de herramientas de simulación de alto rendimiento con las que pueda estudiarse el comportamiento de circuitos formados por decenas o cientos de millones de transistores.

Además de sus problemas inherentes de sincronización, en los futuros escenarios de aplicación de la DDES se preven nuevos problemas de construcción, relacionados con la administración del número de elementos de procesamiento que pueden participar. Se espera que los próximos simuladores sean capaces de decidir la mejor técnica de sincronización que deban aplicar a cada simulación que se les presente, inicializar el experimento de acuerdo con criterios de eficiencia en tiempo y memoria, e incluso, reconfigurarlo sobre la marcha a partir de los mismos criterios o como respuesta ante contingencias [19].

Si se considera el comportamiento dinámico de los componentes que pueden participar en un sistema distribuido es importante incorporar técnicas de tolerancia a fallas, especialmente cuando se observa una tendencia hacia la simulación sobre ambientes heterogéneos (diferentes tipos de computadoras y redes, incluyendo Internet). En muchas aplicaciones de la DDES no puede tolerarse que la reducción en las capacidades de cálculo obliguen a restaurar la ejecución desde cero. Se espera que, de manera automática, un sistema con estas características sea capaz de reconocer un

[1]DDES son las siglas de Distributed Discrete Event Simulation, su traducción es: simulación distribuida de eventos discretos

comportamiento erróneo y reponer el curso normal de su ejecución.

1.4 La contribución al estado del conocimiento

En resumen, la propuesta de esta idónea comunicación de resultados, consiste en la utilización de técnicas para la administración de recursos redundantes con el objetivo de tolerar fallas de paro. En concreto se trata de utilizar un conjunto de componentes redundantes, provistos por una red P2P, para reemplazar a otras máquinas que pudieran quedar fuera de operación, restaurando el sistema hasta un estado global previo, desde el que se retoma la ejecución de la simulación.

Creemos que un valor adicional de nuestra propuesta es que tiene posibilidades de aplicación no solamente en el contexto de la simulación distribuida, sino que puede aplicarse sobre cualquier sistema distribuido basado en comunicación por paso de mensajes.

1.5 Objetivos

1. Objetivo general: Implementación de un prototipo de DDES sobre un ambiente distribuido dinámico.
2. Objetivos particulares:
   - El DDES tendrá soporte para una falla de paro.
   - Especificar las operaciones de un DDES.
   - Plantear las entidades de una arquitectura que implemente dichas operaciones.
   - Construir e integrar los componentes del DDES.

Nota: Si durante la ejecución de una simulación se incorporan recursos a la red de trabajo, éstos se considerarán como reservas en caso que una falla de paro esté presente. A esto se refiere el objetivo general con respecto a implementar un simulador sobre un ambiente distribuido dinámico.

1.6 Metodología

1. Revisión de la literatura en el tema.

2. Diseño de los elementos del sistema.
3. Selección de una herramienta de programación.
4. Implementación del prototipo
5. Evaluación de un caso de estudio.
6. Comunicación de resultados.

1.7 Estructura del documento

En el capítulo 2 se presenta los fundamentos teóricos de la simulación distribuida, en el capítulo 3 se lleva a cabo un estudio del estado del arte con respecto a algunos simuladores de eventos discretos, en el capítulo 4 se presenta la arquitectura propuesta para crear un prototipo de simulador distribuido con tolerancia a una falla de paro, en el capítulo 5 se presenta el diseño de experimentos y sus resultados, en el capítulo 6 se encuentran las conclusiones y el trabajo futuro.

Capítulo 2

Fundamentos Teóricos

Este capítulo ofrece una visión panorámica sobre los fundamentos de la simulación por computadora y, en especial, de la simulación distribuida de eventos discretos, así como los conceptos de computación tolerante a fallas, sobre los que se apoya la propuesta del presente proyecto.

2.1 El modelo como sustituto del sistema

Un sistema es un conjunto de partes organizadas funcionalmente para formar un todo [21]. Como ejemplo se puede citar una computadora, sus componentes son: memoria, CPU, disco duro, entre otros. Como ejemplo: un programa en la memoria puede solicitar al CPU que realice un cálculo y después el resultado se guarda en el disco duro.

Ahora bien, cuando se requiere estudiar un sistema, éste puede encontrarse ya sea en operación y resulta costoso intervenirlo o bien se encuentra en su fase de diseño. Para superar estos inconvenientes es posible recurrir al uso de un modelo.

El modelo es una abstracción del sistema y por tanto se debe especificar las características que resultan relevantes y desechar las que se consideran no necesarias. La elección correcta de las características suele ser a menudo una tarea difícil, por ello se requiere de personas con la experiencia y el conocimiento especializado acerca del objetivo y funcionamiento que tiene (si ya existe) o debe tener el sistema (si está en su fase de diseño).

Existen dos tipos de modelos, cualitativos y cuantitativos.

Con los modelos cualitativos se busca exhibir las propiedades de un sistema, por

otro lado los modelos cuantitativos sirven para evaluar el desempeño de las operaciones con el fin de mejorarlas.

Los métodos cuantitativos se agrupan en tres grandes categorías: la *medición*, el *análisis* y la *simulación*.

1. Con los métodos de medición es posible reunir, a partir de algunas comparaciones y relaciones elementales, un conjunto de criterios de desempeño (débito o caudal, tiempos de respuesta, tasas de servicio, etc.) entre los que pueden establecerse relaciones y hacerse inferencias. Su mayor inconveniente es el número limitado de mediciones, directas e indirectas, que pueden realizarse sobre el objeto de estudio.

2. Con los métodos analíticos se pueden derivar ecuaciones matemáticas que, siempre que puedan resolverse, implican un cálculo rápido. En este sentido son muy eficientes. Su mayor inconveniente reside en el número limitado de modelos que poseen una solución exacta. Sólo los casos más simples pueden resolverse por este método y bajo hipótesis muy limitadas. Los modelos más complejos sólo poseen soluciones aproximadas y la exactitud de ellas difícilmente puede cuantificarse.

3. Con los métodos de simulación, en contraste, se tienen soluciones aproximadas en cualquier ocasión. Sin embargo, la confianza o exactitud del método puede cuantificarse o incluso, controlarse. La simulación permite tomar en cuenta los detalles y describir con precisión el modelo, incluyendo restricciones. El precio que se paga por una mayor exactitud es el tiempo de simulación.

2.2 La simulación por computadora

La simulación por computadora es una técnica numérica de análisis cuantitativo, con la que pueden realizarse experimentos sobre un modelo, a fin de obtener salidas relacionadas con el rendimiento del sistema que representa [21].

Los resultados que se producen a partir de un programa para computadora, representan la solución del modelo y expresan las salidas del sistema como función de sus valores de entrada y sus parámetros de configuración. A partir de la comparación de los resultados de la simulación con un conjunto de medidas conocidas del sistema, el modelo puede validarse o redefinirse mediante un proceso iterativo de depuración hasta que pueda confiarse en el valor predictivo de los resultados.

Entre la variedad de técnicas de simulación descritas en la literatura, aquellas que pueden ser de interés para su uso en computadoras son: la emulación, la simulación Montecarlo, la simulación dirigida por trazas y la simulación dirigida por eventos [22].

Una simulación que utiliza hardware o firmware se llama *emulación*. Un emulador de terminal, por ejemplo, simula un tipo de terminal utilizando otra. Un emulador de procesador simula un conjunto de instrucciones de un procesador, mediante otro. Aunque la emulación es un tipo de simulación, sus objetivos están más relacionados con el diseño de hardware y no tanto con el análisis de desempeño.

Una simulación *Montecarlo* se utiliza para modelar fenómenos probabilistas cuyas propiedades no cambian en el tiempo, por lo que se le conoce también como simulación estática. Igualmente se utiliza para evaluar expresiones deterministas usando técnicas de probabilidad.

Una simulación *dirigida por trazas* utiliza como entrada un registro de los eventos que pueden presentarse en un sistema real, ordenados de manera cronológica. En este tipo de simulación, como en el que le sigue, un evento se entiende como una acción discreta que causa un cambio en el estado del modelo. La simulación por trazas es ampliamente usada para el análisis de sistemas de cómputo, se les ha empleado con mucho éxito para afinar o revisar los algoritmos que administran recursos, como en procedimientos para manejo de páginas, manejo de caché, de calendarización, de prevención de interbloqueo (deadlock) y de asignación dinámica de recursos, entre otros.

Una simulación *dirigida por eventos* o simulación *de eventos discretos* utiliza un modelo de estados discretos, en donde, a partir de ciertas condiciones iniciales el modelo genera nuevos eventos sobre la marcha (obedeciendo alguna propiedad estadística). Hay que observar que el término discreto no se aplica sobre los valores de tiempo usados en la simulación. Una simulación dirigida por eventos puede usar valores continuos o discretos de tiempo.

Cualquiera que sea la técnica de simulación, la selección de un lenguaje de computadora para su implantación es uno de los pasos más importantes del proceso. Existen, al menos, cuatro selecciones posibles: Un lenguaje de simulación, un lenguaje de propósito general, una extensión de un lenguaje de propósito general y un paquete de simulación.

2.3 La simulación de eventos discretos

Todas las simulaciones de eventos discretos tienen una estructura común. Independientemente del sistema que se modela, la simulación tendrá la mayoría de los elementos que se describen enseguida. Si se utiliza un lenguaje de propósito general, todos los componentes tendrán que ser implantados por el analista. En cualquier otro caso, el software que se utilice proveerá estas partes [22]:

Calendarizador de eventos Mantiene una lista o registro cronológico de los eventos que deben suceder y ofrece un conjunto de operaciones para su manipulación.

- Programa un evento e para que ocurra en el tiempo t.
- Retiene un evento e durante un intervalo de tiempo dt.
- Cancela un evento e, previamente programado.
- Retiene un evento e por tiempo indefinido.
- Reprograma un evento e que se había retenido por tiempo indefinido.

El calendarizador es uno de los componentes que se ejecutan con más frecuencia durante la simulación. Se le invoca antes de cada evento y puede llamársele varias veces cuando se atiende un evento, para programar, o cancelar, otros tantos. El diseño de un calendarizador eficiente es, en consecuencia, una tarea muy delicada que incide en el desempeño del simulador.

Reloj y mecanismo de actualización de tiempo

Cada simulación tiene al menos una variable que representa el tiempo simulado. El calendarizador es responsable de actualizarla y avanzar este tiempo. Existen dos maneras de completar esta operación: Puede utilizarse una unidad de tiempo que sirve para incrementar el reloj en cantidades fijas, cada vez que se actualiza el reloj el calendarizador debe revisar si existen eventos programados para ocurrir en este nuevo tiempo (a esta forma de operar el reloj también se le conoce como *simulación dirigida por tiempo*). Alternativamente, el calendarizador puede avanzar el reloj hasta el instante en que debe ocurrir el siguiente evento programado. Obsérvese cómo, en la primer técnica el reloj sólo puede tomar valores discretos, mientras que con la segunda, el reloj puede tomar valores continuos.

Variables de estado Este es un conjunto de variables, posiblemente globales, que describen el estado del sistema.

Rutinas para manejo de eventos Cada evento es simulado por una rutina que actualiza el estado del sistema y puede generar nuevos eventos.

Rutinas de entrada Sirven para ingresar los parámetros del modelo. Típicamente un experimento de simulación se diseña para repetir varias veces una ejecución modificando de manera controlada las condiciones iniciales y con ello elaborar ciertas inferencias estadísticas.

Generador de reportes Estas son las rutinas de salida que se ejecutan al final de la simulación. Calculan los resultados finales y los imprimen en un formato específico.

Rutinas de inicialización Sirven para determinar el estado inicial de las variables del sistema e inicializar la generación de números seudoaleatorios.

Rutinas para registro de trazas Sirven para el registro e impresión de los estados en que evoluciona la ejecución, puede usárseles para la depuración del programa. Se recomienda que estos mecanismos tengan la opción de apagarse y encenderse según se les requiera.

Mecanismos para manejo dinámico de memoria

El número de entidades presentes en una simulación cambia continuamente, conforme se generan nuevas y se desechan viejas instancias. Esta situación requiere de una recolección periódica de basura. Casi todos los lenguajes de simulación y muchos lenguajes de propósito general ofrecen este mecanismo automáticamente. En otro caso, el programador tendrá que cargar con esta responsabilidad.

Programa principal Llama a las rutinas de entrada, inicializa la simulación, controla el desarrollo de cada experimento y finalmente, invoca a las rutinas de salida.

En la simulación de eventos discretos es una práctica común [2] construir un modelo del sistema real, al que se denomina *red de procesos físicos*, que consiste en una gráfica cuyos vértices, o *procesos físicos*, representan unidades autónomas de procesamiento en tanto, las aristas representan las posibles interacciones entre procesos. Estas interacciones se sustentan en el intercambio de *mensajes* cuyo contenido depende del estado del proceso que transmite. Por su parte, el proceso que recibe verá afectado su estado corriente en función de su historia y de cada mensaje aceptado. Visto de otra forma, el mensaje es un evento que afecta el estado de un proceso posiblemente diferente del que lo genera. Partiendo de la red de procesos físicos $(pp_0, \ldots, pp_{n-1})$, se construye un programa para computadora denominado *sistema lógico* o simulador, compuesto por *procesos lógicos* $(lp_0, \ldots, lp_{n-1})$ encargados de representar a los procesos físicos.

2.3.1 Propiedades de la simulación de eventos discretos

En una simulación de eventos discretos, un proceso físico queda descrito por un conjunto de eventos y los tiempos asociados en que se supone estos deben ocurrir. Además, existe una *relación de dependencia* entre todos los eventos del sistema. Si la pareja de eventos (e, e') pertenece a esta relación, se dice que e' depende de e. De esta manera se captura el sentido intuitivo del orden causal en el que los eventos deben presentarse. El tiempo asociado con el evento e' no debe ser menor que el tiempo asociado a cualquier evento del que dependa e'.

Existen dos condiciones que un sistema físico debe reunir para poder simularse: *realizabilidad* y *predecibilidad* [2].

- Realizabilidad. Establece que un proceso físico (pp[1]) no puede adivinar que mensaje pueda llegar en el futuro.

- Predecibilidad. Garantiza que el sistema está "bien definido" en el sentido que la salida de cada pp para cualquier tiempo t puede ser calculado dado el estado inicial del sistema.

Se dice que *un sistema lógico simula un sistema físico, si es posible que el primero prediga la secuencia exacta de mensajes transmitidos por el segundo.* Esto es, si t_1, t_2, ... , t_i, ... son los tiempos en que el sistema físico debe transmitir los mensajes m_1, m_2, ... , m_i, ... y $t_1 \leq t_2 \ldots \leq t_i \leq \ldots$, entonces el sistema lógico debe poder generar la secuencia de salida $\langle (t_1, m_1), (t_2, m_2), \ldots, (t_i, m_i), \ldots \rangle$. Obviamente, la construcción del simulador supone que la velocidad a la que ocurren los eventos en éste será diferente de la velocidad con que suceden en el sistema físico.

2.4 Simulación distribuida de eventos discretos

El análisis de los grandes sistemas se ve limitado en el número de eventos que pueden simularse. En el caso de las nuevas redes de telecomunicaciones, por ejemplo, un sólo experimento de simulación puede tomar varias horas e incluso días, antes de producir resultados.

Para acelerar la producción de resultados se utiliza la simulación distribuida de eventos discretos, con la cual se pretende distribuir la carga de trabajo sobre los diferentes procesadores del sistema.

Un sistema distribuido[2] consiste en:

1. Un número finito de procesos lógicos (lp [2]).

2. Canales de comunicación que conectan a los lp entre si.

3. Cada lp puede enviar o recibir mensajes(se puede inferir que cada lp tiene canales de salida y de entrada).

4. Los mensajes se entregan en el orden que son enviados.

5. Un lp puede tener que esperar por el arribo de un mensaje a través de alguno de sus canales de entrada.

[1] pp por sus siglas en inglés.
[2] lp por sus siglas en inglés

La forma como se construye un sistema lógico es de la siguiente manera:

1. Se asocia un lp con cada pp.
2. Un lp simulará las acciones de un pp, en algún punto de la simulación.
3. Un lp debe haber recibido una secuencia correcta de mensajes a través de sus canales de entrada.
4. Un lp debe haber enviado una secuencia correcta de mensajes a través de sus canales de salida.

Esta descripción del sistema distribuido trae consigo una consecuencia inminente, la presencia de interbloqueos (deadlocks). Este problema consiste en que en algún punto se estanca la simulación y un lp se puede quedar esperando indefinidamente recibir un mensaje que nunca llega. Existe el método conservador y el optimista que contribuyen a que en una simulación, la secuencia de eventos se ejecute respetando la relación de dependencia, que pudiera existir entre ellos, así como también, la posible presencia de un interbloqueo.

2.4.1 Método conservador o seguro

Si un proceso tiene un mensaje m_1 con un tiempo asociado t_1 y no se ha procesado aún, y si el proceso puede determinar que no recibirá ningún otro mensaje con tiempo menor a t_1, entonces puede procesar de forma segura a m_1[23]. Para lograr esto hay que considerar que se requiere que los mensajes (internos y externos), sean procesados en orden cronológico, los canales de comunicación deben ser FIFO y tener asociado un reloj interno, además un proceso elegirá el canal con menor tiempo y obtendrá de ahí un mensaje cuya estampilla de tiempo sea mayor que el tiempo local virtual del proceso.

Si un canal vacío conecta dos procesos que se han bloqueado, y este tiene la menor estampilla de tiempo para ambos procesos, se puede inferir que los procesos se encuentran en un interbloqueo. Para evitar los interbloqueos en el método conservardor, se utiliza el envío de mensajes nulos, que incluyen un valor numérico que se encarga de establecer una promesa que el tiempo correspondiente al siguiente mensaje a enviar no será menor al valor numérico que acompaña al mensaje nulo (ver algoritmo 1).

2.4.2 Método optimista

El método optimista [14], en contraste con el conservador, puede avanzar su reloj local conforme procese mensajes que se encuentran pendientes, no requiere que el mensaje

Algorithm 1 Algoritmo Método Conservador

```
1: comienza
2:   Inicializa el reloj local y de cada canal de entrada
3:     reloj = 0;
4:     para todo canal de entrada C(j), t(j) = 0
5:
6:     Mientras(criterio de terminación == falso)
7:       sea C el canal de entrada con el reloj mínimo
8:       SI (C esta vacio), espera
9:       OTRO elimina(t_in, m_in) del frente de C
10:          simula el efecto de recibir m_in en t_in
11:          reloj = t_in
12:          para todo (t_out, m_out), mensaje de salida,
13:          SI (reloj >= t_out), transmite (t_out, m_out)
14:          transmite mensaje nulo por cada salida
15: termina
```

a procesar sea seguro, esto permite que la simulación pueda explotar el paralelismo entre sus componentes. Hay problemas que trae consigo este enfoque: la posibilidad de recibir mensajes rezagados y con ello la posibilidad de no respetar la restricción de dependencia entre mensajes (o eventos). Para superar este inconveniente, cada proceso lleva un historial de estados por los que ha pasado, los mensajes que ha procesado y los mensajes que ha enviado (por cada mensaje enviado se crea un antimensaje), con todo esto cuando llega un mensaje rezagado se desencadena un mecanismo de restauración que permitirá ejecutar en orden adecuado los mensajes (incluyendo el mensaje rezagado) así como regresar al proceso a un estado previo al instante que le corresponde al mensaje que llegó rezagado. También se enviarán antimensajes por cada mensaje enviado previo a la llegada del mensaje rezagado.

Se define el tiempo global virtual (GVT) como la estampilla de tiempo menor entre todos los mensajes no procesados y el menor tiempo de entre todos los relojes virtuales de los procesos. Este valor sirve como cota, que por un lado, no permitirá que se haga una restauración para tiempos menores a GVT y, por otro lado, realizará una recolección de fósiles, es decir el borrado de mensajes procesados, estados y antimensajes cuyas estampillas de tiempo son menores a GVT. Ver algoritmo 2.

2.4.3 Estado global de una ejecución distribuida

En el cómputo distribuido existen problemas como recolección de basura, detección de interbloqueos, detección de la terminación, entre otros, cuya solución se alcanza mediante la evaluación de un predicado global.

Algorithm 2 Algoritmo Método Optimista

```
1: comienza
2:   Inicializa el reloj local y de cada canal de entrada
3:     reloj = 0;
4:     para todo canal de entrada C(j), t(j) = 0
5:
6:     Mientras(criterio de terminación == falso)
7:       sea C el canal de entrada, no vacio, con el reloj mínimo
8:
9:       elimina (t_in, m_in) del frente de C
10:      SI (t_in < reloj), llama a restauración
11:      OTRO simula el efecto de recibir m_in en t_in
12:          reloj = t_in
13:          para todo (t_out, m_out), mensaje de salida,
14:          si reloj¿=t_out, transmite (t_out, m_out)
15: termina
```

Para encontrar dicha solución se requiere conocer el estado global del sistema. En un sistema distribuido no se cuenta con un tiempo global y la única forma que un proceso puede conocer el estado de otros procesos es mediante la comunicación por medio de mensajes.

Un sistema distribuido está formado por un conjunto de procesos. En cada proceso se ejecuta una serie de eventos. Por tanto la historia local de cada proceso p_i [38] se puede caracterizar por la serie de eventos que sucedieron.

$$h_i = e_i^1, e_i^2, \ldots$$

Un prefijo de la historia del proceso p_i se representa hasta el k-ésimo evento que sucedió.

$$h_k = e_i^1, e_i^2, \ldots, e_i^k$$

La historia global es la unión de las historias individuales de los procesos.

$$H = h_1 \cup \ldots \cup h_n$$

Nótese que la historia global no especifica los tiempos relativos entre los eventos que sucedieron en el proceso.

Un evento que se ejecuta en un proceso, puede representar un cambio de estado, el envío o bien la recepción de un mensaje a través de sus canales de comunicación. Y hay eventos que son la causa de la ejecución de otros, a esta noción de la causa y el efecto se le denomina relación *sucedió antes* [33], que señala lo siguiente:

1. Si $e_i^k, e_i^l \in h_i$ y $k < l$, entonces $e_i^k \rightarrow e_i^l$,
2. Si e_i es un evento de transmisión y e_j es su recepción correspondiente, entonces $e_i \rightarrow e_j$.
3. Si $e \rightarrow e'$ y $e' \rightarrow e''$, entonces $e \rightarrow e''$.

Sin embargo hay eventos que no tienen una relación *sucedió antes* y por tanto pueden ocurrir de forma concurrente, lo que se denota como $e||e'$.

Un conjunto de eventos que sucedieron en un proceso da como resultado un conjunto de estados por que los que éste pasó. Un estado global será formado por los estados locales de cada uno de los procesos del sistema.

Un corte de la ejecución es un subconjunto C de su historia global que contiene un prefijo inicial de las historias locales.

$$C = h_1^{c_1} \cup \ldots \cup h_n^{c_n}$$

Los eventos involucrados en un corte se denominan frontera del corte. Un corte resulta en un posible estado global que se presentó en un punto en el tiempo, por otro lado también puede resultar en un estado que nunca ocurrió. Por tanto para que un estado global pueda haber ocurrido, se requiere que un corte sea consistente.

Se define un corte consistente como aquel donde:

$$(e \in C) \wedge (e' \rightarrow e) \Rightarrow e' \in C,$$

Un estado global consistente es el resultado de un corte consistente.

De la figura 2.1 se muestra un caso en el que se tiene un corte consistente en el lado izquierdo de la figura, y del lado derecho un corte inconsistente.

Por último, una *corrida* es un orden total R que incluye a todos los eventos de la historia global y es consistente con cada historia local. Esto significa que, por cada proceso p_i, los eventos de p_i aparecen en R en el mismo orden que aparecen en h_i. Obsérvese que una corrida puede ó no ser igual a la ejecución que representa y, por otro lado, una misma ejecución puede representarse con más de una corrida.

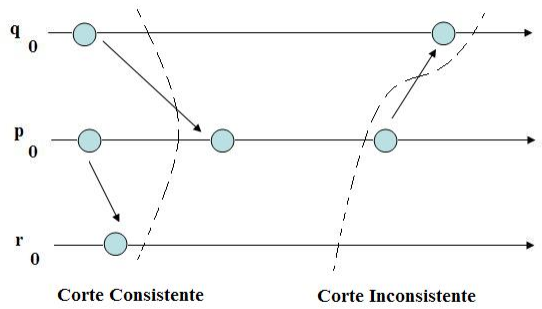


Figura 2.1: Dos cortes de una ejecución distribuida.

2.4.4 Algoritmo para toma de estado global

Se utiliza el algoritmo de Chandy-Lamport [13] para determinar estados globales consistentes. Un estado global consistente muestra un cierto estado en el que se encontraba una ejecución distribuida, y puede ser utilizado para no reiniciar una ejecución desde el principio.

El algoritmo es el siguiente:

El modelo de sistema distribuido en el que se basa este algoritmo tiene las siguientes premisas:

- Los canales de comunicación son confiables y tienen colas de entrada infinitos.
- Los canales de comunicación son del tipo FIFO.
- Hay un retardo finito entre el envío y la recepción de un mensaje.

Así como sus características son:

- El estado global resultante es consistente.
- Termina en un tiempo finito.
- Se puede ejecutar de forma conjunta con el trabajo que un proceso puede o no estar realizando, en ese instante.
- Cualquier participante de una ejecución distribuida puede iniciar este algoritmo.

Algorithm 3 Algoritmo Toma de Estado Global

1: Regla para el envio del marcador por el proceso p_i
2: a) Proceso p_i graba su estado.
3: b) Por cada canal de salida C, sobre el cual un marcador no ha sido enviado, p_i envía un marcador a través de C antes que p_i envíe más mensajes a través de C.
4:
5: Regla para recibir marcador por el proceso p_j
6: Al recibir un marcador a través del canal C':
7: SI p_j no ha grabado su estado ENTONCES
8: Graba su estado y
9: Graba el estado de C' como el conjunto vacío
10: Ejecuta la *Regla para el envio del marcador a partir del inciso b)*
11: OTRO
12: Graba el estado de C' como el conjunto de mensajes recibidos a través de C' después de que el estado de p_j fue grabado y antes de que p_j recibiera el marcador a través de C'

- No hay interbloqueos.

Para la implementación del presente prototipo se lleva a cabo la implementación de los algoritmos de toma de estado global, cálculo del tiempo global virtual (GVT por sus siglas en inglés), entrelazado de tiempo y restauración del sistema. Los detalles se encuentran en el capítulo 4.

2.5 Conceptos de computación confiable

Un sistema se considera confiable, siempre que ofrezca un servicio de acuerdo a las especificaciones establecidas con antelación, además de ser capaz de enmascarar las posibles fallas que se presenten y que éstas sean imperceptibles a los ojos de un usuario o bien presentadas de la forma más amigable posible.

Existen cierto criterios que se pueden considerar a la hora de la construcción de un sistema, estos criterios ofrecen una garantía en el funcionamiento de éste, dicha garantía permite que un usuario confie en el servicio que el sistema le ofrece. La lista de criterios se definen a continuación [8]:

- Con relación a la rapidez con que el sistema pueda ofrecer un servicio en un instante de tiempo, la garantía se denomina disponibilidad (*availability*).
- Con relación a continuidad del servicio, la garantía se denomina confiabilidad (*reliability*).

- Con relación a la prevención de desastres, la garantía se denomina seguridad (*safety*).
- Con relación a la integridad y la confidencialidad de la información, la garantía también se denomina seguridad (*security*)[3].

La falla (*failure*) de un sistema ocurre cuando el servicio que ofrece se desvía de su especificación, siendo ésta una descripción acordada del servicio.

Las causas por las cuales se producen fallas, se clasifican en tres, por su naturaleza, origen y persistencia:

1. Por su naturaleza se se dividen en accidentales o intencionales.
2. Por su origen se dividen en fenómeno que las origina (físicas o humanas), lugar donde se originan con relación a las fronteras del sistema (internas o externas), y la fase de creación (concepción u operacionales).
3. Por su persistencia (temporales y permanentes).

En la medida que un sistema distribuido crece, también crece el número de sus componentes y la probabilidad de que alguno falle.

Un componente cuyo servicio se desvía de su especificación puede clasificarse de acuerdo con un modelo de falla. En la literatura de sistemas distribuidos, los modelos de falla comúnmente aceptados son [20]:

- Paro (*failstop*). Un procesador incurre en esta falla si se detiene y permanece en ese estado. Los demás procesadores pueden detectar esta condición.
- Caída (*crash*). Un procesador incurre en esta falla si se detiene y permanece en ese estado. Los demás procesadores no pueden detectar esta condición.
- Caída y enlace (*crash+link*). Un procesador incurre en esta falla si se detiene y permanece en ese estado. Un enlace falla perdiendo algunos mensajes, pero no retrasa, duplica o corrompe mensajes.
- Omisión de recepción (*receive omission*). Un procesador incurre en esta falla si recibe sólo un subconjunto de los mensajes que se le han enviado (posiblemente porque se detiene y permanece en ese estado).

[3]Las palabras “safety” y “security” se traducen al español como “seguridad”, lo que puede resultar ambiguo, a menos que se explique el contexto en el que se les utiliza.

- Omisión de transmisión (*send omission*). Un procesador incurre en esta falla si transmite sólo un subconjunto de los mensajes que debía enviar (posiblemente porque se detiene y permanece en ese estado).

- Omisión general (*general omission*). Un procesador incurre en esta falla si recibe sólo un subconjunto de los mensajes que se le han enviado, envía un subconjunto de los mensajes que debía envíar y/o se detiene y permanece en ese estado.

- Fallas bizantinas. Un procesador incurre en esta falla si exhibe un comportamiento arbitrario.

Ante la posibilidad de que ocurran fallas (*faults*), para ofrecer un sistema que garantice un servicio conforme a una especificación, se cuenta con un conjunto de herramientas y métodos aplicables, desde la fase de concepción hasta la implantación. Según el momento de su aplicación, estos pueden clasificarse de la siguiente manera [8]:

- *Prevención.* Se trata de impedir, desde la construcción, la ocurrencia de fallas.

- *Tolerancia.* Se trata de ofrecer, conforme a la especificación, funciones redundantes que puedan entrar en operación en caso de fallas en los componentes primarios.

- *Eliminación.* Se trata de disminuir, mediante un proceso de verificación, el número y la gravedad de las fallas.

- *Previsión.* Se trata de estimar, mediante evaluación, la posible presencia de fallas y sus consecuencias.

En la práctica se pueden hacer combinaciones de estos métodos tratando de asegurar el funcionamiento del sistema en más de un frente. La utilización conjunta de técnicas de prevención y eliminación busca evitar la presencia de fallas. De manera semejante, con la aplicación de técnicas de prevención y tolerancia se persigue conseguir la garantía de funcionamiento. Finalmente, la eliminación y previsión se efectúan durante el proceso de validación.

En computación distribuida se cuenta el número de componentes que pueden causar fallas y no las ocurrencias de un comportamiento que se desvía de su especificación. Se habla de que un sistema es f-tolerante cuando puede continuar satisfaciendo su especificación siempre que no se presenten más de f componentes que puedan generar fallas [20].

EL objetivo de un sistema distribuido es detectar, enmascarar o eliminar errores, antes de que sean evidentes a un usuario (comunmente desde una interfaz gráfica). Para

lograr estos objetivos se utiliza lo que se denomina redundancia. Existen tre tipos de redundancia [31]: de recursos físicos, de tiempo y de información.

La *redundancia de recursos físicos* se refiere a la replicación de los componentes físicos del sistema. Por ejemplo, si se provee un sistema con tres computadoras en vez de una, se pueden comparar los resultados y seleccionar el resultado de la mayoría, de esta forma se puede enmascarar la falla de un componente.

La *redundancia de tiempo* se refiere a la repetición de una acción de cómputo o de comunicaciones, sobre el dominio del tiempo. Por ejemplo, es una operación bien conocida en sistemas de comunicación, reenvíar un mensaje (en un instante diferente) si el receptor no devuelve un acuse al cabo de cierto plazo.

La *redundancia de información* se refiere a una técnica de codificación. Un texto fuente se transforma en un texto objeto que pertenece a un espacio de codificación más grande y, con ello, hace posible la detección y corrección de errores que pudieran introducirse en el texto objeto. En sistemas de trasmisión de datos, por ejemplo, se utiliza una cadena de bits denominada, secuencia de verificación, que consiste en una secuencia redundante de información, con la que el receptor de un mensaje puede inferir si éste ha sufrido algún error de integridad.

Para soportar fallas en un simulador distribuido, se cuenta con varios tipos de algoritmos, de los cuales se presenta una reseña a continuación [16]:

1. Basados en puntos de control:

   - Requiere de tomas periódicas de puntos de control [4].
   - Coordinados: simplifican la recuperación o restauración y la recolección de basura. Presentan buen rendimiento en la práctica.
   - No coordinados: propensos a tener el efecto dominó [5]. Se complica la recuperación o restauración. Requiere de coordinación para la recolección de basura.
   - Punto de control inducido por comunicación
     - Depende de los patrones de comunicación de las aplicaciones para la toma de checkpoints.
     - La naturaleza no determinista de estos protocolos complica la recolección de basura y degrada el desempeño.

[4] Un punto de control (checkpoint por su nombre en Inglés)se puede considerar como la última toma de estado global consistente realizada sobre un sistema distribuido.

[5] Un sistema presenta una falla y desea volver a un estado consistente previo a la falla,se dice que presenta un efecto dominó, si del conjunto global de cortes de un sistema distribuido no se tiene ningún corte consistente, y por tanto su ejecución tendrá que volver a su estado inicial.

2. Restauración-Recuperación basada en registro (Log Based Rollback-Recovery)

   - Son la elección natural para aplicaciones que interactúan con mucha frecuencia con el mundo exterior.
   - Hay tres categorías: pesimista, optimista y causal.
     - Pesimista: es el más atractivo para muchas aplicaciones reales. Simplifica la recuperación, , protege a los procesos sobrevivientes de tener que hacer una restauración (rollback). Presenta sobrecarga en el rendimiento.
     - Optimista: reduce la sobrecarga, hay mensajes huérfanos [6], esto resulta en una complicación para la recuperación, y para la recolección de basura.
     - Causal: reduce sobrecarga, está libre de mensajes huérfanos. La recuperación se vuelve compleja así como también la recolección de basura.

Se recomienda que el lector consulte las referencias citadas a lo largo de este capítulo para un estudio y comprensión más profundo de los temas expuestos.

2.6 Los sistemas par a par (P2P)

Los sistemas par a par (peer to peer por su nombre en inglés), son una clase de sistemas y aplicaciones que emplean recursos distribuidos para realizar una función de forma descentralizada [9]. Cada nodo tiene potencialmente la misma responsabilidad. Los sistemas P2P tienen una interesante capacidad de autoorganización, que sirve para afrontar los frecuentes cambios de la topología debido a la naturaleza dinámica de las computadoras participantes, los cuales entran y salen del sistema. El enfoque P2P proveé ciertas ventajas sobre los sistemas convencionales lo que permite justificar su uso en la construcción de sistemas distribuidos.

Algunas de las características principales de un sistema P2P son [9]:

1. Escalabilidad: Es deseable que ante el incremento de los nodos en la red, el sistema siga funcionado correctamente, esto implica que ante la llegada o salida de nodos, el sistema debe mantenerse estable.

2. Robustez: Se refiere a seguir en funcionamiento aún con fallos en los nodos de la red.

[6] Un proceso transmisor envía un mensaje a un receptor, el transmisor presenta una falla de paro, por consiguiente el mensaje que llega al transmisor se considera como un mensaje huérfano.

3. Descentralización: Quiere decir que cada uno de los nodos puede tomar el rol de cliente o de servidor y que la comunicación entre pares es simétrica.

4. Costos distribuidos: Al compartirse los recursos, los costos también se reparten entre los nodos participantes en la red.

5. Anonimato: En algunas aplicaciones es deseable mantener oculta la información acerca del autor de un contenido, el lector, el servidor que lo alberga y la petición para encontrarlo.

6. Seguridad: Identificar y evitar nodos con contenido malicioso.

7. Persistencia de información: El sistema debe ser capaz de proveer acceso a los datos, y se debe asegurar que los datos almacenados en el sistema están disponibles y protegidos.

8. Balance de carga: Se refiere a que el sistema debe ser capaz de hacer una distribución eficiente de los recursos, basada en la capacidad y disponibilidad de los nodos.

9. Localización rápida de los recursos: Es una de las características más importantes, pues como los recursos están distribuidos en diversos pares se necesita de un mecanismo eficiente para la localización y recuperación del recurso solicitado.

En contraste con el esquema cliente-servidor, los sistemas P2P ofrecen inherentemente escalabilidad y disponibilidad de recursos.

La redundancia de componentes físicos que ofrece un sistema par a par, mediante el soporte para entrada y salida de nodos, brinda la capacidad de ofrecer tolerancia a fallas de paro en un simulador distribuido. Además de soportar un ambiente dinámico. Estas cualidades son consideradas para la elaboración del presente trabajo.

2.6.1 Arquitectura JXTA P2P

JXTA es la especificación de una plataforma P2P cuyo desarrollo fue iniciado por Sun Microsystems.

Dentro de JXTA P2P las máquinas involucradas en la red, reciben el nombre de pares (peers). A continuación se mencionan los diferentes modos de configuración para los pares, dentro de la arquitectura JXTA, seguida de la implementación de esos modos de configuración dentro de JXSE.

2.6.2 Configuración JXTA P2P

De acuerdo a [15], en la arquitectura de JXTA se tienen diferentes modos de configuración para los pares, los cuales se enumeran a continuación:

1. Pares hoja mínimos (Minimal Edge Peer): Típicamente envían y reciben mensajes, pero no guardan copias de anuncios o rutas de mensajes. Este tipo de configuración es adecuada para sensores, teléfonos móviles, etc. Son considerados como consumidores puros de la red JXTA.

2. Pares hoja completamente equipados (Full-Featured Edge Peer): Pueden enviar y recibir mensajes, pueden guardar anuncios y participar en el proceso de descubrimiento iniciado por otros pares respondiendo a sus solicitudes cuando puedan. Contribuyen en gran medida a la red JXTA, dado que ayudan a otros pares a encontrar u obtener acceso a recursos sobre la red.

3. Pares de punto de encuentro (RENDEZVOUS): son similares a los pares completamente equipados, sólo que pueden tomar más responsabilidades en organizar la red JXTA. Guardan un mapa de todos los pares conectados en un grupo de pares y ayudan a reenviar mensajes y solicitudes. El par punto de encuentro es considerado como un par de infraesructura.

4. Par de relevo (Relay Peer): formalmente conocidos como pares de ruta en los inicios de JXTA. Su propósito es ayudar a los pares ocultos por alguna característica de la red, a que se conecten con otros pares. También son considerados como pares de infraestructura.

Capítulo 3

Escenarios de oportunidad

Este capítulo describe cómo la DDES[1] requiere incorporar escenarios en los que participen un número escalable de máquinas. Al mismo tiempo, esta tendencia obliga a considerar capacidades para soportar el comportamiento dinámico de los elementos que colaboran en la simulación, incluyendo la posibilidad de tolerar fallas de paro. Por último se hace una revisión panorámica y un estudio crítico sobre las diferentes herramientas de simulación que se conocen al momento de escribir este documento.

3.1 Las tendencias de la DDES

En los más 20 años que tiene como tema de investigación, la simulación distribuida ha demostrado ser una rica fuente de interesantes problemas teórico/prácticos y tener un extenso campo de aplicaciones. Desde sus usos más convencionales como herramienta para analizar el desempeño de sistemas de tratamiento de información o redes de vehículos, hasta su más reciente utilización como instrumento de realidad virtual para entrenamiento o videojuegos [24].

Con el advenimiento de canales de comunicación con bajas tasas de error e incremento de velocidad, así como la madurez en los lenguajes orientados a objetos, se tienen ventajas entre las cuales se puede mencionar:

- La posibilidad de abordar problemas hasta ahora pendientes debido a su escala y complejidad.
- La reutilización del software y sus patrones de diseño.

[1]DDES significa Distributed Discrete Event Simulator por su nombre en Inglés.

- La transparencia con que el usuario final puede utilizar las aplicaciones.

Con todo, aún quedan tareas por resolver o mejorar. En el futuro, se espera que un simulador pueda decidir automáticamente la mejor técnica de sincronización que debe aplicarse a cada simulación que se le presenta, inicializar el experimento de acuerdo con criterios de eficiencia en tiempo y memoria, e incluso, reconfigurarlo sobre la marcha a partir de los mismos criterios o como respuesta ante fallas [19].

1. Existen algunos parámetros que afectan el rendimiento de una simulación: granularidad, latencia de los canales, intervalo de "punto de control" (*checkpoint*), estabilidad en la "restauración" (*roll-back*) , distribución inicial de carga [25, 26, 27, 28]. Sin embargo, todavía no se puede hablar de modelos terminados que sirvan para predecir el escenario óptimo de una simulación.

2. Por otro lado, no existe hasta ahora una conclusión definitiva acerca de la superioridad de una técnica de sincronización sobre otra. Se conocen diferentes enfoques que buscan encontrar un compromiso entre los métodos conservadores y los optimistas: 1) relajando los protocolos conservadores, 2) limitando los protocolos optimistas, 3) adaptando dinámicamente el manejo de la sincronización. Esta última alternativa plantea una solución continua que abarca a los dos tipos de métodos como sus casos extremos. Entre sus ventajas más importantes está su capacidad para reconocer el intervalo de sincronización correcto sin la intervención de un programador, lo cual es muy importante cuando se quiere alcanzar a los usuarios no especializados o bien, cuando la simulación exhibe patrones de comunicación que varían con el tiempo [29, 35, 36].

3. También es importante incorporar técnicas de computación móvil y tolerancia a fallas, especialmente cuando se observa una tendencia hacia la simulación sobre ambientes heterogéneos (diferentes tipos de computadoras y redes, incluyendo Internet). En lo que se refiere al entrelazado de tiempo, es importante revisar los algoritmos con que se determina el tiempo virtual global (GVT), ya que esta operación puede tener un fuerte impacto en el costo de las comunicaciones y el tiempo de ejecución [32, 34].

Se presenta a continuación un conjunto representativo de simuladores de eventos discretos. Se desea conocer de cada simulador si es capaz de:

1. Soportar fallas de paro.

2. Trabajar en una red dinámica.

3. Ser de propósito general.

3.1.1 Proyecto YADDES

YADDES son las siglas de su nombre en Inglés: Yet Another Distributed Discrete Event Simulator.

Entre sus características se puede mencionar [7]:

1. La especificación de un lenguaje y un compilador para la simulación.
2. La nueva versión de una simulación se tiene que distribuir a mano entre los recursos computacionales.
3. Una biblioteca para recolección de estadísticas y generación de reportes.
4. La red de simulación debe ser fija mientras la simulación se lleva a cabo.
5. No ofrece ningún mecanismo de tolerancia a fallas de paro con redundancia de componentes físicos.

3.1.2 Proyecto Parsimony

Entre sus características se puede mencionar [3]:

1. Utiliza el lenguaje de programa java para implementar los componentes del sistema.
2. Trabaja bajo el esquema maestro-esclavo.
3. Para la simulación distribuida utiliza las bibliotecas de RMI (Remote Method Invocation por su nombre en Inglés).
4. La red de simulación debe ser fija mientras la simulación se lleva a cabo.
5. No ofrece ningún mecanismo de tolerancia a fallas de paro con redundancia de componentes físicos.

Este proyecto ofrece ocho simuladores, tres de los cuales son distribuidos. Se soportan los enfoques conservador y optimista.

3.1.3 Proyecto GPDES

GPDES significa General Purpose Discrete Event Simulator.

Entre sus características se puede mencionar [4]:

1. Utiliza el lenguaje de programa java para implementar los componentes del sistema y establecer la comunicación entre los recursos computacionales.
2. Trabaja bajo el esquema cliente-servidor.
3. Toda la comunicación y distribución de trabajo lo hace el servidor. No se permite comunicación entre clientes.
4. La red de simulación debe ser fija mientras la simulación se lleva a cabo.
5. No ofrece ningún mecanismo de tolerancia a fallas de paro con redundancia de componentes físicos.

3.1.4 Proyecto OMNeT++

Entre sus características se puede mencionar [5]:

1. Utiliza un lenguaje llamado NED, para implementar los componentes del sistema.
2. Soporta los métodos optimista y conservador.
3. Trabaja bajo el esquema maestro-esclavo.
4. Para llevar a cabo las simulaciones distribuidas este simulador depende de MPI [6].
5. La red de simulación debe ser fija mientras la simulación se lleva a cabo.
6. No ofrece ningún mecanismo de tolerancia a fallas de paro con redundancia de componentes físicos.

3.1.5 Proyecto POSE

POSE: Parallel Object Oriented Simulation Environment. El proyecto POSE, permite utilizar C++, y cada proceso lógico es asignado a un objeto de C++.

Entre sus características se puede mencionar [10]:

1. Utiliza el lenguaje de programación C++, para implementar los componentes del sistema.
2. Utiliza el método optimista, para respetar la restricción de causalidad.
3. POSE mejora la simulación de grano fino, ya que cuando un LP recibe el control, ejecuta un grupo entero de eventos pendientes.
4. La red de simulación debe ser fija mientras la simulación se lleva a cabo.
5. No ofrece ningún mecanismo de tolerancia a fallas de paro con redundancia de componentes físicos.

3.1.6 Proyecto μsik

Entre sus características se puede mencionar [11]:

1. Utiliza el lenguaje de programación C++, para implementar los componentes del sistema.
2. Ofrece servicios esenciales como soporte para la comunicación, generación aleatoria de números, entre otros.
3. Utiliza los métodos conservador y optimista.
4. La red de simulación debe ser fija mientras la simulación se lleva a cabo.
5. No ofrece ningún mecanismo de tolerancia a fallas de paro con redundancia de componentes físicos.

3.1.7 Proyecto Aurora

Entre sus características se puede mencionar [12]:

1. Utiliza los estándares de servicios web, para llevar a cabo simulaciones a gran escala.
2. Trabaja bajo el esquema cliente-servidor.
3. El servidor asigna trabajo a un cliente, una vez termina el cliente de procesar la información, devuelve los resultados al servidor.
4. Está prohibida la comunicación entre clientes.

5. Utiliza un método conservador.

6. La red de simulación puede cambiar con el tiempo gracias al esquema cliente-servidor.

7. El mecanismo de tolerancia a fallas de paro que ofrece es mediante la reasignación de trabajo, que no se completó previamente, a otros clientes .

3.1.8 CDES

CDES: Centralized Discrete Event Simulator.

Entre sus características se puede mencionar [1]:

1. Se pueden realizar simulaciones centralizadas.

2. Utiliza el lenguaje de programación C++ para implementar los componentes del sistema a simular.

3. No requiere usar ningún método conservador u optimista.

4. Tiene una interfaz gráfica de usuario (GUI) con la que puede visualizarse la animación de una traza de ejecución, editar topologías y controlar la velocidad de la animación.

5. Dado que es un simulador que trabaja de forma centralizada, no utiliza una red trabajo y no cuenta con ningún mecanismo de tolerancia a fallas paro.

3.2 Análisis comparativo

A continuación se comparan las prestaciones que ofrecen los proyectos descritos anteriormente.

La tabla 3.1 muestra un análisis comparativo de los simuladores que se han mencionado en este capítulo.

En términos generales, los simuladores:

1. Son de propósito general, esto indica que todo sistema que se pueda representar mediante el enfoque de eventos discretos, se puede programar en cualquiera de esos simuladores.

Proyecto	Propósito General	Red Dinámica	Tolerancia
YADDES	Si	No	No
Parsimony	Si	No	No
GPDES	Si	No	No
OMNeT++	Si	No	No
POSE	Si	No	No
μsik	Si	No	No
Aurora	Si	Si	Si
CDES	Si	No	No

Tabla 3.1: Tabla comparativa.

2. No presentan soporte para tolerancia a fallas de paro, excepto el proyecto Aurora.

3. No soportan cambios dinámicos en la topología de la red de trabajo, excepto el proyecto Aurora[2].

3.3 Otros simuladores

En esta sección se presentan simuladores cuyas propuestas son interesantes. No obstante cabe mencionar que su enfoque es para tolerancia a fallos en procesamiento en paralelo.

La propuesta "Tolerancia a fallos de paro de tipo proactivo " [17] tiene las siguientes características:

1. Que las fallas sean predecibles

2. Hace uso del protocolo SMART para predecir futuros fallos en un disco duro.

3. Hace uso de sensores de temperatura de las tarjetas principales[3] para poder predecir una falla en un procesador.

4. Hace uso del lenguaje Charm++.

5. Hace uso de una plataforma de paso de mensajes para sistemas paralelos, conocida como AMPI.

[2]El cambio dinámico consiste en que hace uso del esquema cliente/servidor, y no permite la comunicación entre trabajadores. Lo que realmente se busca es que haya comunicación entre los diferentes componentes del simulador distribuido

[3]motherboards por su nombre en Inglés.

La propuesta consiste en predecir con antelación que puede ocurrir una falla y antes de que se haga presente reacomodar los procesos de la simulación en el resto de procesadores.

La propuesta "Protocolo de tolerancia a fallos con restauración rápida"[18] tiene las siguientes características:

1. Hace uso del lenguaje Charm++.
2. Hace uso de una plataforma de paso de mensajes para sistemas paralelos, conocida como AMPI.
3. Utiliza las capacidades del lenguaje Charm++ y AMPI[4] para realizar balance de carga entre procesadores.
4. La toma de estado global (snapshot) se guarda en memoria.

La forma como trabaja es: cuando un procesador falla, se hace un balance de carga entre los procesadores restantes que participan en la simulación.

Ambas propuestas [17, 18] no utilizan el esquema de redundancia en componentes físicos, para tolerar una falla de paro, simplemente realizan un nuevo balance de carga.

[4]AMPI Adaptive MPI, es una plataforma que se utiliza para procesamiento en paralelo, que a su vez usa el lenguaje de programación Charm++.

Capítulo 4

Diseño e Implementación del Prototipo DDES

En este capítulo se presenta el diseño y la implementación del prototipo de simulador distribuido de eventos discretos. Se abordan por un lado los objetivos del presente trabajo:

- El DDES tendrá soporte para una falla de paro[1].
- Especificar las operaciones de un DDES.
- Plantear las entidades de una arquitectura que implemente dichas operaciones.
- Diseño e implementación de los componentes del DDES.

y por otro lado, de la metodología propuesta se cubren los siguientes puntos:

- Diseño de los elementos del sistema.
- Selección de una herramienta de programación.
- Implementación del prototipo.

A partir de este momento cuando se mencione método optimista se hace referencia al mecanismo de entrelazado de tiempo [14]. Ver sección 4.4.1.

Las siglas GVT se utilizan para hacer referencia al tiempo global virtual (Global Virtual Time por su nombre en inglés). Ver sección 4.4.1.

[1]Es importante recordar que el soporte para una falla de paro, es posible, siempre y cuando exista una toma de estado global que se calculó y almacenó previamente.

4.1 Arquitectura propuesta

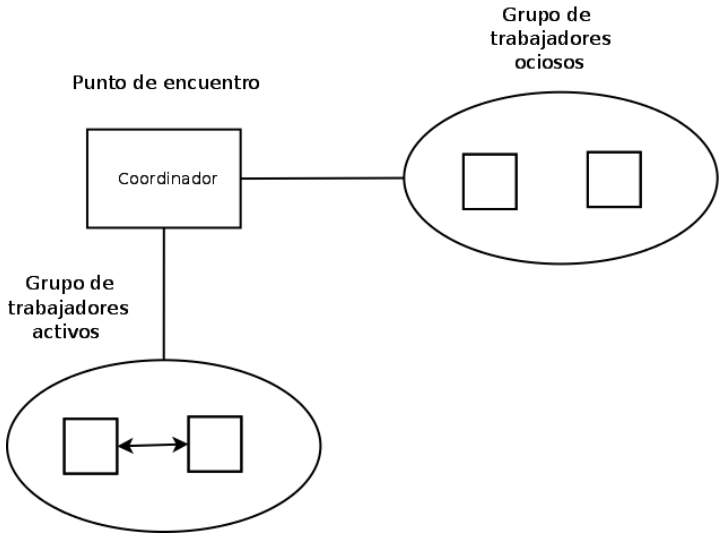


Figura 4.1: Arquitectura global.

La figura 4.1, muestra la arquitectura general propuesta para el prototipo de simulador que se pretende construir.

Se utiliza una red de tipo par a par (P2P), en donde se tiene un coordinador que manejará dos grupos, primero, el grupo de trabajadores ociosos, y segundo, el grupo de trabajadores activos. El grupo de trabajadores ociosos es el grupo al cual pertenecen todos los trabajadores que están listos para formar parte de la simulación. También es importante destacar que un coordinador es del tipo "punto de encuentro", y los trabajadores son del tipo "hoja". Ver sección 2.6 para recordar los tipos de pares en una red P2P.

4.1.1 Vista general

La forma general como trabaja este simulador es la siguiente:

1. Al inicio debe haber un coordinador y los trabajadores que se vayan incorporando a la red P2P se añaden al grupo de trabajadores ociosos.
2. Un coordinador sabe la cantidad de trabajadores que están disponibles para llevar a cabo la simulación.
3. Cuando se inicia la simulación, el coordinador conoce el grafo que representa el sistema que va a simular, como un ejemplo se puede apreciar la figura 4.2

4. El coordinador particiona el sistema que se desea simular y asigna las particiones resultantes a los trabajadores.

5. Los trabajadores elegidos pasarán a formar parte del grupo de trabajadores activos.

6. El coordinador envía un mensaje al primer trabajador que tiene en su lista de registro, para iniciar la simulación.

7. En algún momento el coordinador envía el mensaje *snapshot* al primer trabajador que tiene en su lista de registro, para iniciar la toma de estado global.

8. Cuando finaliza la toma de estado global, cada trabajador envía una copia de su estado local al coordinador.

9. En algún momento el coordinador envía el mensaje *calGVT*, al primer trabajador que tiene en su lista de registro, para iniciar el cálculo de GVT

10. Una vez que finaliza el algoritmo, cada trabajador envía su correspondiente cálculo de GVT al coordinador.

11. El coordinador selecciona el más pequeño de los valores para el GVT y lo envía a todos los trabajadores.

12. Cada trabajador recibe y guarda el valor del nuevo GVT para la recolección de fósiles.

13. Se tienen dos mecanismos de restauración, el primero se desencadena cuando se viola la restricción de "sucedió antes", y el segundo se ejecuta cuando hay una falla de paro[2].

14. El coordinador tiene un hilo de ejecución que se llama ConnectoPeer y se encarga de verificar que los trabajadores se encuentren activos. Si algún trabajador deja de funcionar o abandona el grupo, ConnectoPeer le avisa al coordinador y éste tiene que iniciar el proceso de restauración, y asignará los procesos que se perdieron a algún trabajador de reserva (ocioso).

15. De acuerdo a [14], cuando GVT alcanza el valor de +inf[3], se concluye que la simulación ha finalizado.

16. El coordinador recibe las trazas que contienen los resultados parciales de la simulación distribuida.

17. El coordinador construye la traza total.

[2]En este trabajo se considera como falla de paro: cuando un trabajador sale de la red de simulación o bien deja de funcionar.

[3]+inf significa que se alcanza un valor infinito positivo, en términos prácticos se utilizó un valor entero igual a $2^{31} - 1$

4.1.2 El coordinador

En la figura 4.3, se puede apreciar la arquitectura del coordinador. Una instancia del coordinador (Coordinator):

1. Usa una instancia de la clase Graph para tener acceso al sistema a simular, éste último tiene que estar representado en forma de grafo como se puede apreciar en la figura 4.2.
2. Crea una tabla de procesos (tableProcess) a partir de Graph, utilizando instancias de la clase Process.
3. Cada instancia de Process usa instancias de Model y de Simulator. Una instancia de la clase Model (modelo) tiene el algoritmo que se estará ejecutando durante la simulación. Una instancia de la clase Simulator, es el motor de simulación que todo proceso estará utilizando.

4.1.3 El trabajador

Una instancia del trabajador (Worker):

1. Usa una tabla de procesos (tableProcess), dicha tabla es asignada por el coordinador antes de dar inicio a la simulación. Por tanto ya contiene información acerca del conjunto de instancias de Process, Model y Simulator.
2. El motor de simulación (instancia de Simulator) en un trabajador, tiene que realizar dos tareas con la agenda: guardar y extraer un evento.
3. La agenda es una lista donde se guardan tanto los eventos que pertenecen a mensajes positivos como también a los antimensajes.
4. El trabajador se encarga de verificar si el evento que extrae Simulator, va dirigido a un proceso que se encuentra de forma local o remota. Si es de forma local entonces continúa con la ejecución normal del algoritmo, si es remota establece una conexión con el trabajador que contiene el proceso al cual va dirigido el evento.

4.2 Operaciones del prototipo DDES

En esta sección se presentan las operaciones que realiza tanto el coordinador como el trabajador.

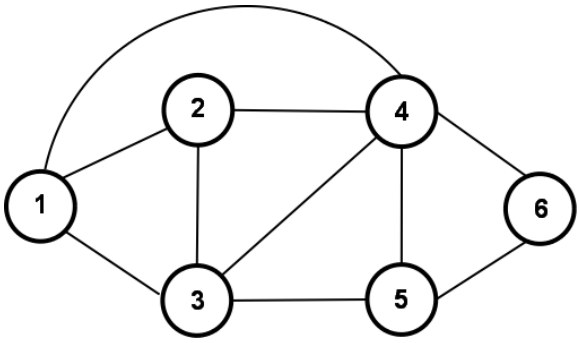


Figura 4.2: Ejemplo de sistema a simular.

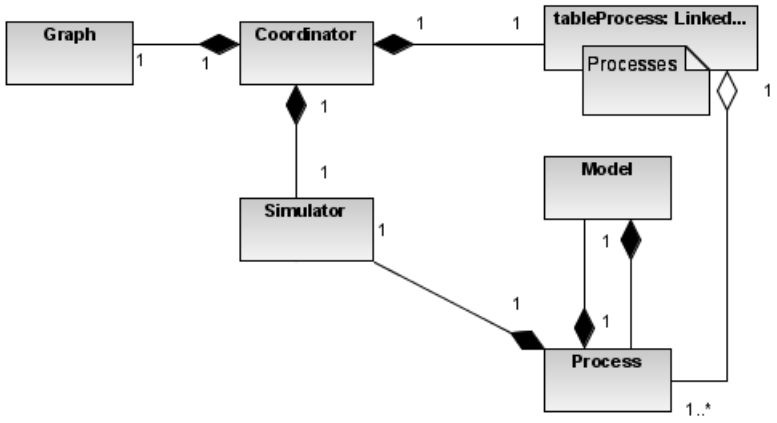


Figura 4.3: Arquitectura coordinador.

4.3 Operaciones que realiza el coordinador

Las operaciones que el coordinador debe realizar son las siguientes:

1. Particionar el sistema que se desea simular y asignar las particiones resultantes a los trabajadores.
2. Toma de Estado Global.
3. Calcular GVT
4. Operación de Restauración (Rollback) debido a una falla de paro en un trabajador.

4.3.1 Operación: asignar particiones del sistema a simular

Para explicar esta operación, se supone que se tienen dos trabajadores ociosos, y el sistema a simular es el de la figura 4.2.

1. El Coordinador realiza una partición en el sistema a simular (ver figura 4.5).
2. El Coordinador asigna una partición a cada trabajador (ver figura 4.6).
3. El Coordinador envía un mensaje al trabajador que se encuentra como primero en su lista de registro, para iniciar la simulación.

La forma en la que un coordinador asigna una partición es por medio de una operación aritmética, se divide el número de nodos del grafo entre el número de trabajadores, y como resultado se obtiene el número de nodos que corresponde a cada trabajador. Los trabajadores están registrados en una lista del coordinador, así que al primer trabajador le corresponde el primer conjunto de nodos y así sucesivamente.

De la figura 4.6, se puede apreciar que los procesos 1,2 y 3, tienen como vecino al proceso 4, pero el proceso 4 se encuentra en el trabajador 2. Las líneas discontinuas reciben el nombre de conexiones lógicas. [4]

Cuando el proceso 1 necesite mandar un mensaje a 4, no se va a hacer uso de su conexión lógica, sino que utilizará un canal de comunicación, como se muestra en la figura 4.7, este canal se denota por la flecha que va del trabajador 1 al 2, de igual forma si el proceso 4 necesita enviar un mensaje a 1, se utilizará el canal que conecta al trabajador 2 con 1.

[4]Se le llama conexión lógica ya que de acuerdo a la figura 4.2 el proceso 4 se conecta con 1,2 y 3 pero físicamente sólo habrá una conexión real entre dos trabajadores.

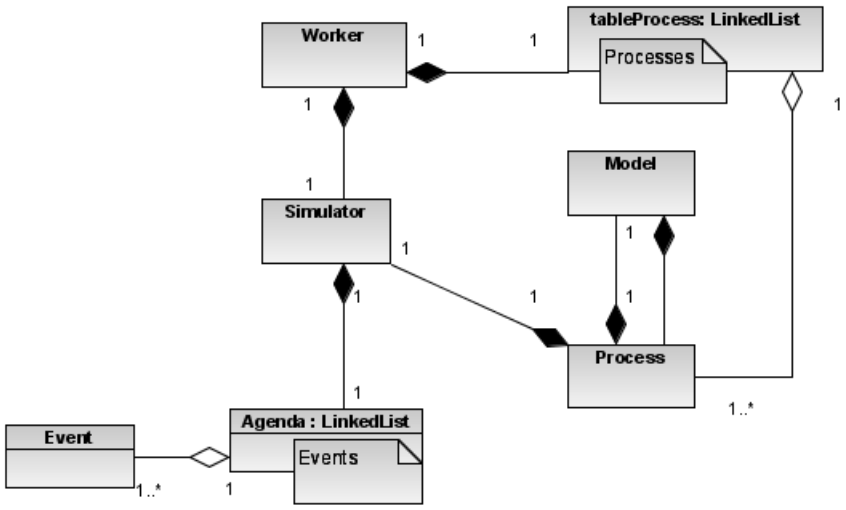


Figura 4.4: Arquitectura trabajador.

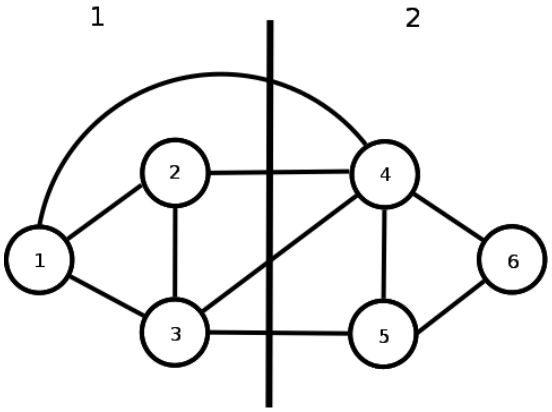


Figura 4.5: Partición del sistema a simular.

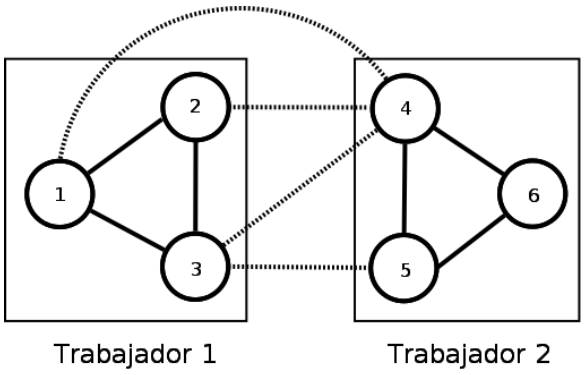


Figura 4.6: Asignación de las particiones.

4.3.2 Operación: toma de estado global

La figura 4.8 muestra que el coordinador crea un hilo de ejecución utilizando la clase CalcSnapshot, esta instancia se encargará de iniciar la operación de toma de estado global cada cierto tiempo, mediante el envío de un mensaje al primer trabajador que se encuentra registrado en la lista del coordinador.

El lector puede remitirse al capítulo 2.4.4, para revisar el algoritmo nuevamente.

Se presenta a continuación un ejemplo que muestra la ejecución de la operación toma de estado global.

La figura 4.9 muestra tres trabajadores activos que están conectados entre sí, y se consideran como base para este ejemplo.

La operación de toma de estado global se comportaría de la siguiente manera:

1. CalcSnapshot inicia enviando un marcador *snapshot*, al primer trabajador de la lista. Ver figura 4.10.
2. El trabajador 1 guarda su estado local. Ver Figura 4.11.
3. El trabajador 1 envía marcadores a través de todos sus canales de salida, es decir hacia el trabajador 2 y 3. Ver figura 4.12.
4. Supongamos que trabajador 3 envía un mensaje antes de que llegue el marcador procedente de trabajador 1. Ver figura 4.13.
5. Los trabajadores 2 y 3 reciben el marcador y guardan su estado local. El mensaje con estampilla de tiempo 5, se guarda como parte del estado del canal para el trabajador 1. Ver figura 4.14.
6. Los trabajadores 2 y 3 envían los marcadores a través de todos sus canales de salida. Ver figura 4.15.
7. Se termina la operación de *toma de estado global* una vez que cada trabajador recibe un marcador a través de todos sus canales de entrada.
8. Cada uno de los trabajadores envía una copia de su estado local y de canales al coordinador. El coordinador guarda toda esa información. Ver figura 4.16.

En este ejemplo se tiene como resultado el corte consistente que se muestra en la figura 4.17. Se puede apreciar que la historia de cada trabajador queda determinada por los eventos precedentes a la línea discontinua y para el trabajador 1 se tiene además el mensaje (5,m) como parte del estado de su canal, los trabajadores 2 y 3 tienen un estado vacío en sus canales.

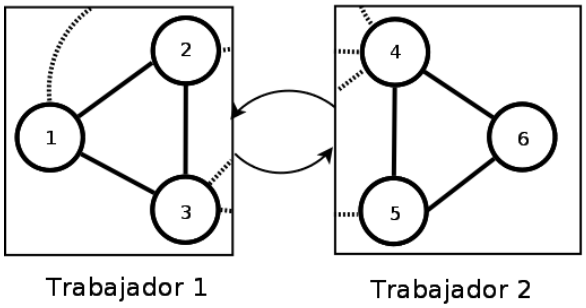


Figura 4.7: Canales de comunicación.

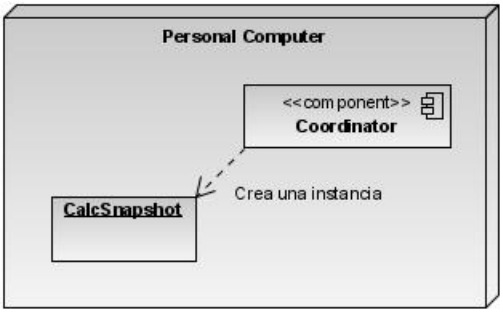


Figura 4.8: Componentes para la toma de estado global.

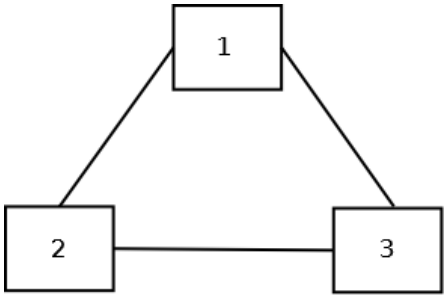


Figura 4.9: Conexión entre trabajadores activos.

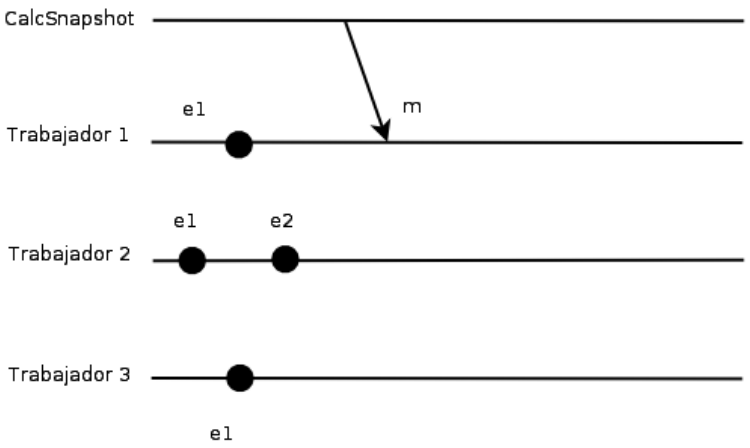


Figura 4.10: Coordinador enviando marcador.

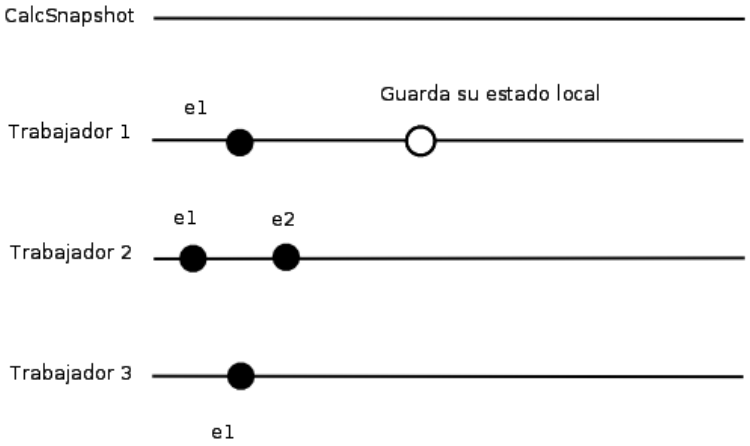


Figura 4.11: Trabajador 1 guarda su estado local.

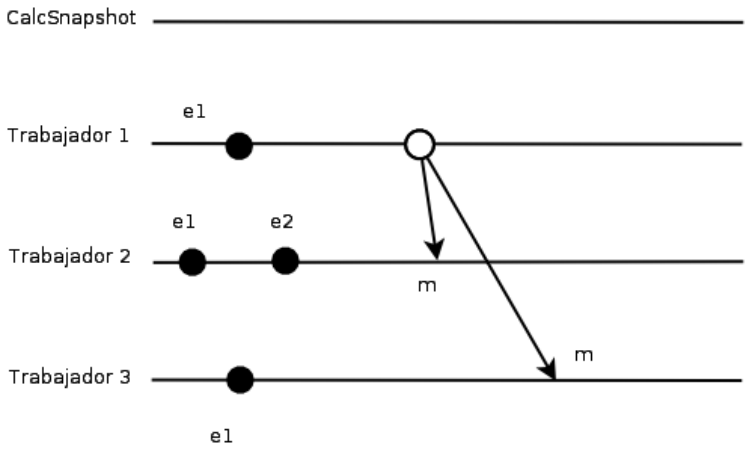


Figura 4.12: Trabajador 1 enviando marcadores a 2 y 3.

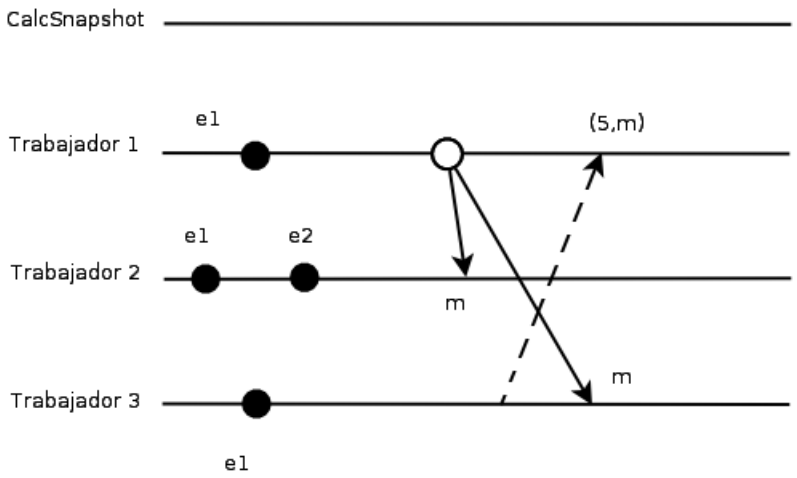


Figura 4.13: Envío del mensaje (5,m).

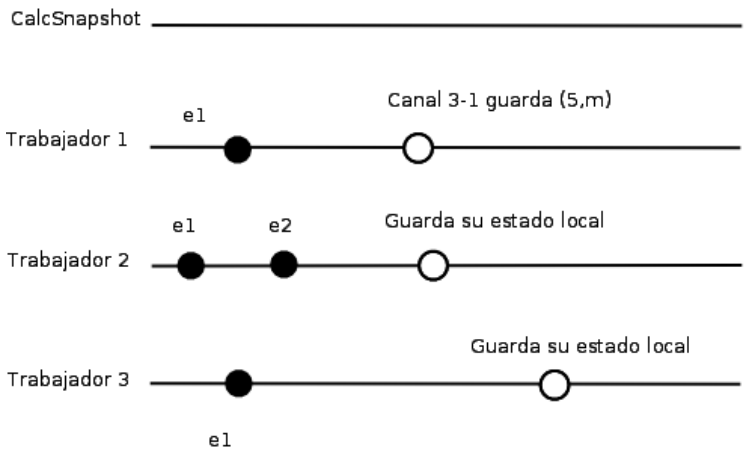


Figura 4.14: Guardando estado local y del canal.

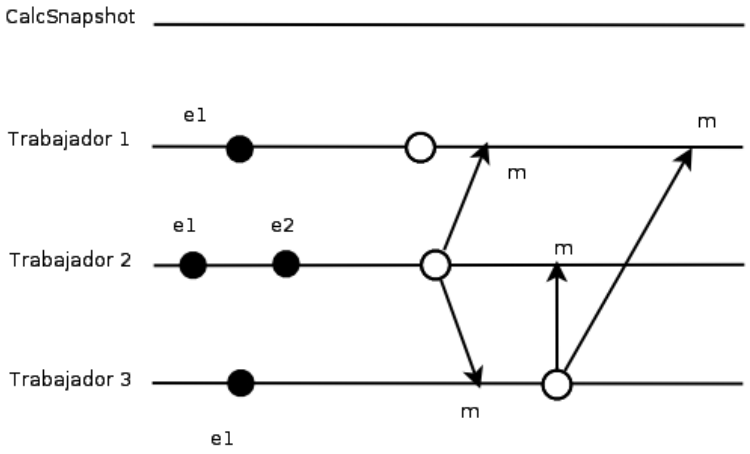


Figura 4.15: Envío de marcadores.

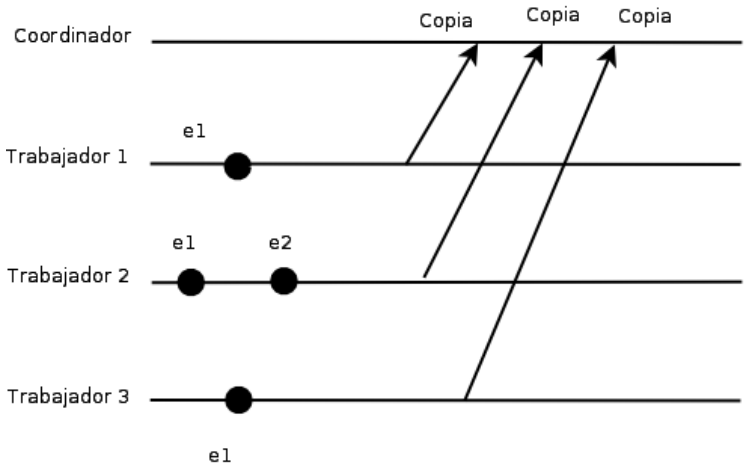


Figura 4.16: Envío de copias al coordinador.

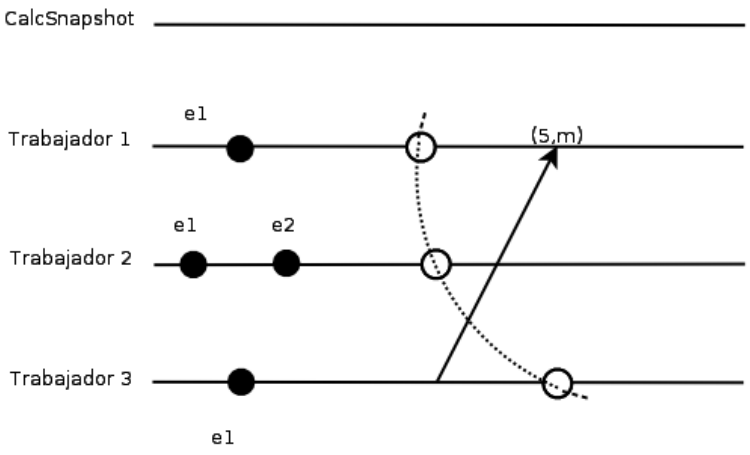


Figura 4.17: Corte consistente.

El diagrama de secuencias de la figura 4.18, muestra de forma general el funcionamiento de la operación de *toma de estado global*.

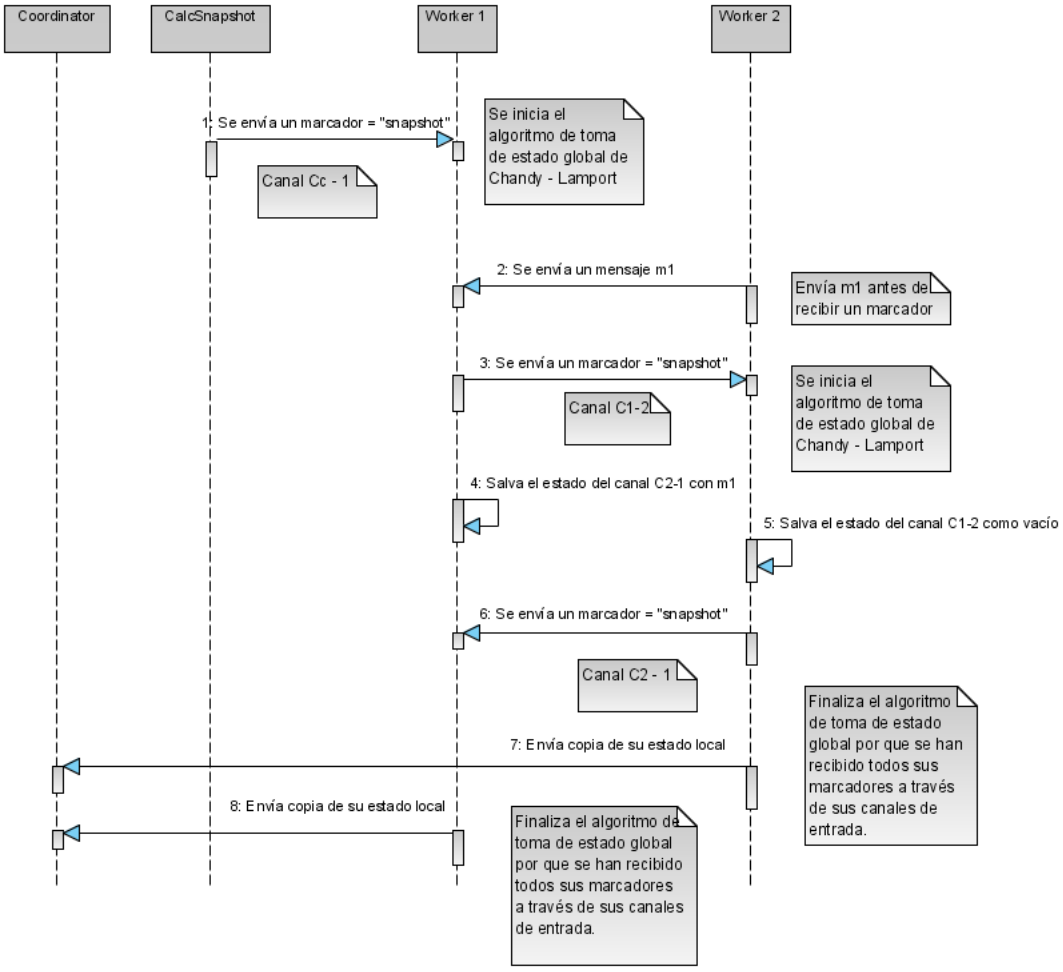


Figura 4.18: Diagrama de secuencias para toma de estado global.

4.3.3 Operación: calcular el valor del tiempo global virtual

El GVT se calcula utilizando el algoritmo de Chandy-Lamport [13].

La figura 4.19 muestra que el coordinador crea un hilo de ejecución con CalcGVT, esta instancia se encargará de iniciar la operación de cálculo de GVT cada cierto tiempo, mediante el envío de un mensaje al primer trabajador que se encuentra registrado en la lista del coordinador.

Como ejemplo se considera una conexión entre tres trabajadores activos, como se

muestra en la figura 4.9. Así la operación para calcular el valor de GVT funciona de la siguiente manera:

1. CalcGVT envía el marcador *calGVT*, al primer trabajador de la lista (ver figura 4.20).

2. El trabajador 1 recibe el marcador. Ejecuta la operación de *Cálculo de GVT*, que consiste en elegir el valor de tiempo más pequeño que puede encontrar en su agenda y la tabla de procesos. Ver figura 4.21.

3. El trabajador 1 envía marcadores a través de todos sus canales de salida. Ver figura 4.22.

4. Cada trabajador que recibe el marcador *calGVT*, elige el valor de tiempo más pequeño que puede encontrar en su agenda y la tabla de procesos. En este ejemplo hay un mensaje con estampilla de tiempo 5, que se envía antes que Trabajador 3 reciba el marcardor *calGVT*. Véase figura 4.23.

5. Mientras no reciba ningún marcador, todo mensaje que llega en un canal de entrada se considera para el cálculo del GVT, tal y como se muestra en la figura 4.24.

6. Se termina la operación de *Cálculo de GVT* una vez que cada trabajador recibe en todos sus canales de entrada un marcador.

7. Cada trabajador envía una copia de su tiempo calculado de GVT al coordinador, ver figura 4.25.

8. El coordinador selecciona de entre todos los tiempos virtuales, el valor más pequeño, y eso determinará el valor actual de GVT.

9. Una vez el coordinador determina el valor de GVT, éste se transmite a todos los trabajadores que participan en la simulación. Ver figura 4.26.

El valor de GVT es importante, ya que con él se realizará la recolección de fósiles, un fósil es un estado previo al valor que tiene GVT. Estos estados son guardados por cada proceso tras la ejecución de un evento, y por tanto llega un momento en el que se debe liberar espacio. El valor de GVT también es utilizado para determinar si ya finalizó una simulación.

La figura 4.27, corresponde al diagrama de secuencias que muestra de forma general el Cálculo de GVT.

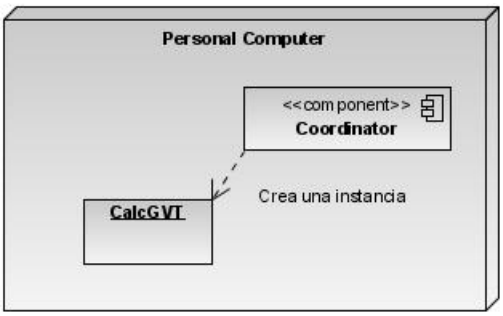


Figura 4.19: Componentes para el cálculo de GVT.

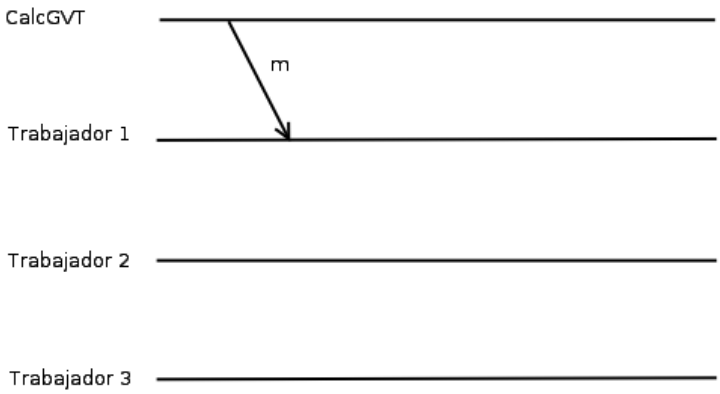


Figura 4.20: Se envía un marcador.

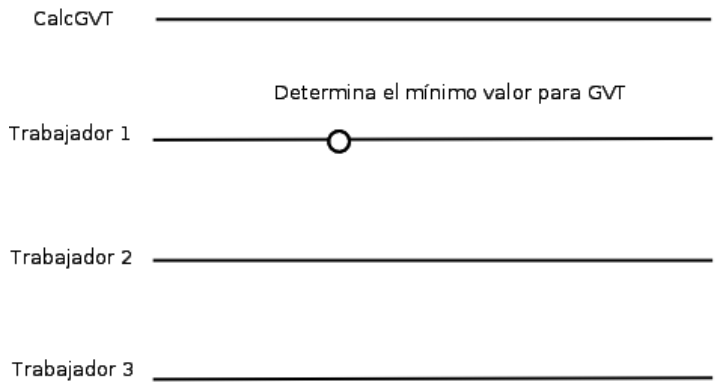


Figura 4.21: Toma de estado local de trabajador 1.

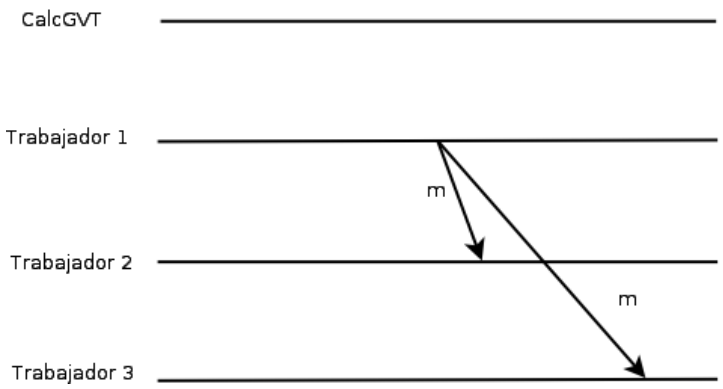


Figura 4.22: Envío marcadores.

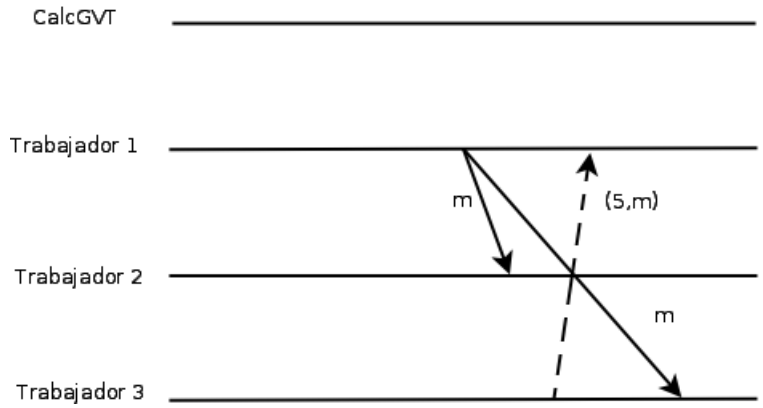


Figura 4.23: Envío del mensaje (5,m).

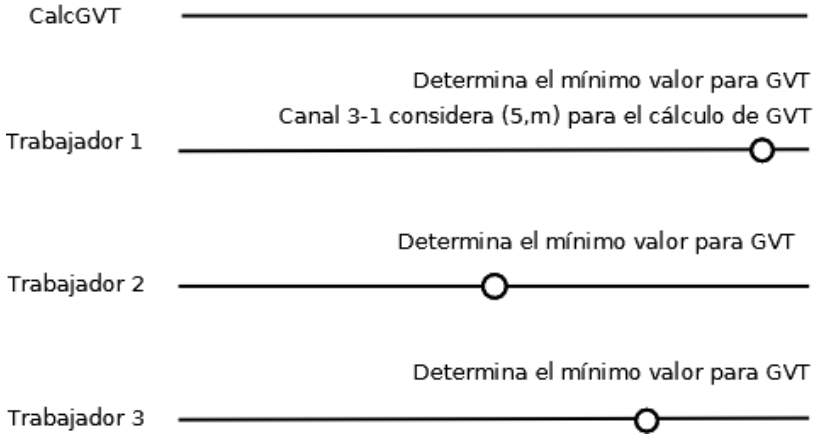


Figura 4.24: Cálculo local del GVT.

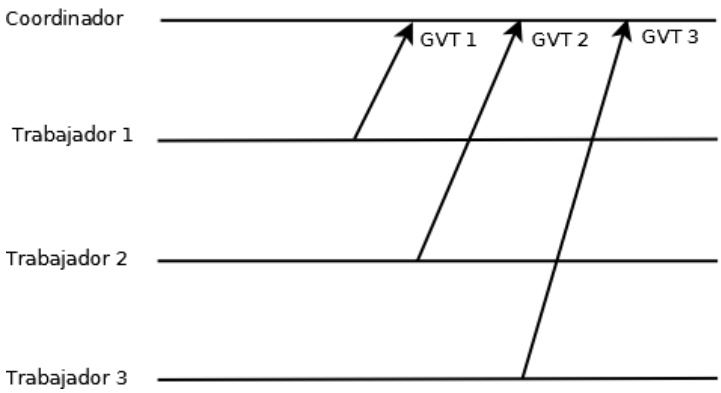


Figura 4.25: Envío del tiempo virtual local al coordinador.

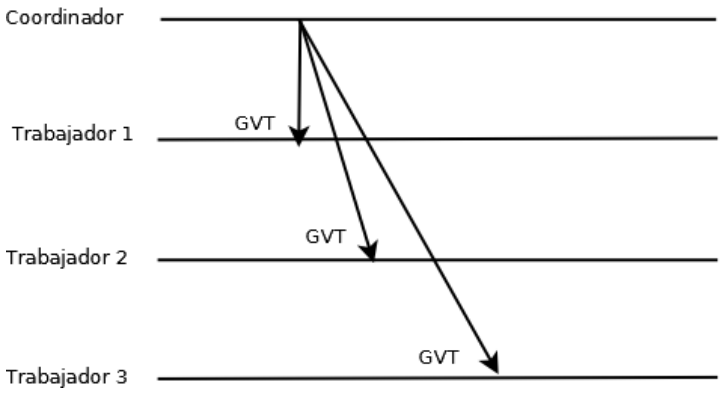


Figura 4.26: Envío del GVT a los trabajadores.

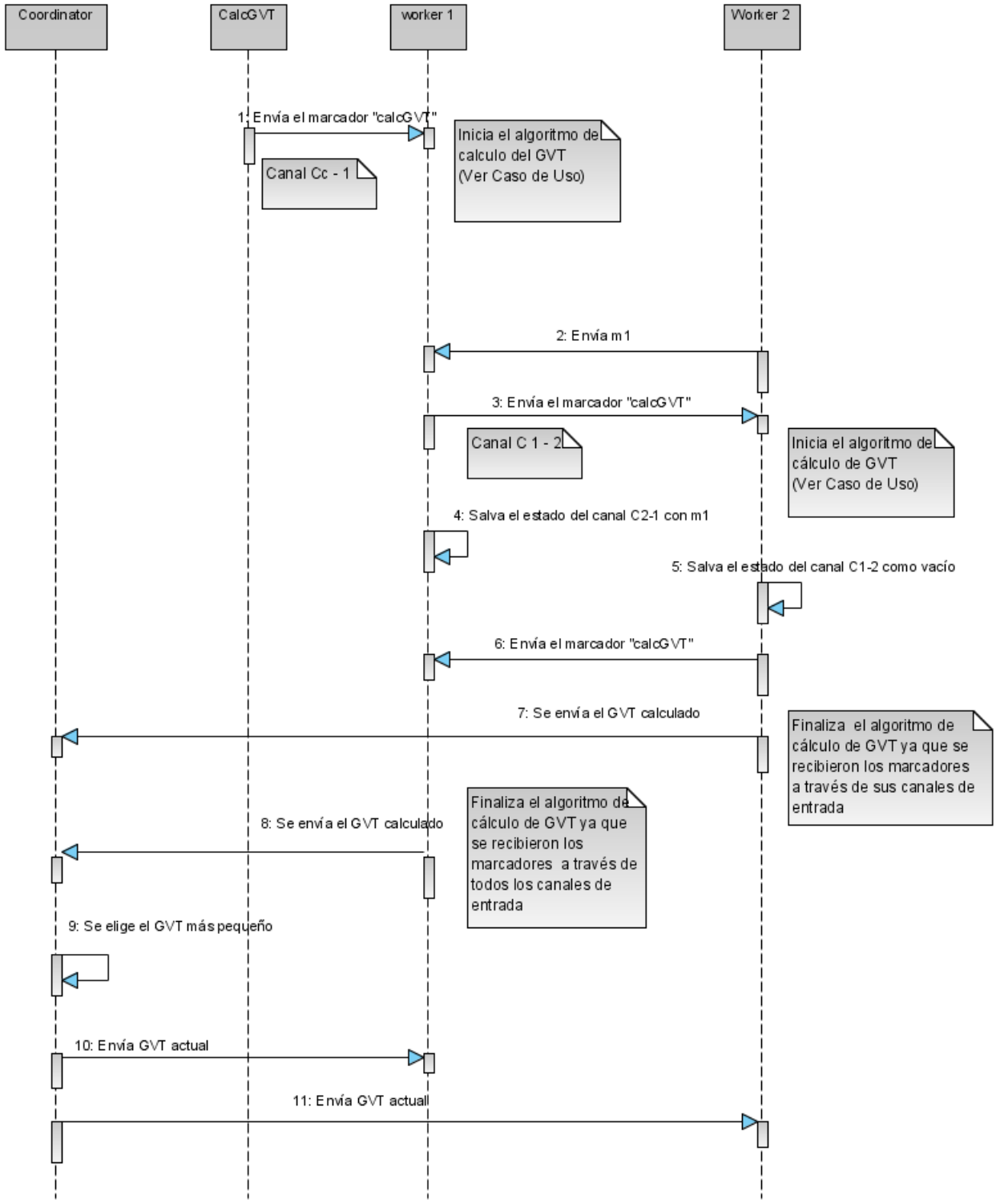


Figura 4.27: Diagrama de secuencias para cálculo GVT.

4.3.4 Operación: restauración debido a una falla de paro

Antes de mostrar un ejemplo de cómo se ejecuta esta operación es importante mencionar dos cosas:

Primero el número de fallas de paro que se pueden soportar queda determinado por la siguiente relación:

número de reservas >= número de nodos que presentan fallas de paro.

Segundo: Hay que considerar el caso cuando se está tomando un estado global y/o calculando el valor de GVT y se presenta una falla de paro. En el caso que se ejecute el cálculo de GVT y se presente una falla de paro, se descarta esta operación. Para la toma de estado global, si se presenta una falla de paro hay que considerar dos escenarios:

Escenario 1: Hay una versión previa de toma de estado global.

El escenario se describe mejor de la siguiente forma:

1. El simulador crea una carpeta con el nombre new, para guardar la toma de estado global más reciente.
2. El coordinador inicia la nueva toma de estado global.
3. El coordinador crea una nueva carpeta con el nombre old, y realiza una copia de la versión que se encuentra en la carpeta new.
4. La carpeta new queda vacía.
5. Ocurre una falla de paro durante la toma de estado global.
6. El coordinador descarta la operación actual de toma de estado global y recurre a ejecutar la operación de restauración, utilizando la versión antigua de la toma de estado global.
7. Si se estaba llevando a cabo el cálculo de GVT éste también se descarta.
8. Una vez que el sistema se recupera de la falla de paro, procede a continuar con la simulación, y también se ralizarán los siguientes cálculos para GVT en la simulación.

Escenario 2: No hay una versión previa de toma de estado global.

Los siguientes pasos describen el escenario:

1. El sistema no ha realizado ninguna toma de estado global.

2. Por primera vez inicia la toma de estado global.
3. Ocurre una falla de paro.
4. Se descarta la toma de estado global y se procede a realizar una restauración del sistema.
5. El sistema colapsa dado que no existe ninguna versión antigua de alguna toma de estado global.
6. En caso que se realice un cálculo de GVT, éste se descarta.

El siguiente ejemplo explica cómo se lleva a cabo la operación: restauración debido a una falla de paro.

Considerando la figura 4.28, se aprecia que el Coordinador crea un hilo de ejecución llamado ConnectoPeer que se encarga de verificar que los trabajadores estén conectados.

A manera de ejemplo supongamos que se cuenta con una conexión de tres trabajadores activos (1,2 y 3) como se muestra en la figura 4.9, y también supongamos que se tiene un trabajador de reserva (ocioso) y el trabajador tres presenta una falla de paro.

Esta operación funciona de la siguiente manera:

1. En el momento que ConnectoPeer detecta que algún trabajador activo no está conectado avisa al coordinador para que se lleve a cabo el proceso de restauración.
2. El coordinador inicia el proceso de restauración (rollback) enviando el mensaje *rollbackphase1* a los trabajadores activos. Ver figura 4.29.
3. Cada trabajador que recibe el mensaje de *rollbackphase1*, detiene su ejecución, y avisa al coordinador que está listo para hacer la restauración. Ver figura 4.30.
4. El coordinador verifica si hay más trabajadores disponibles.
5. El coordinador elige un trabajador ocioso, y le asigna una copia del estado local y la partición de procesos que tenía el trabajador que tuvo una falla de paro.
6. El nuevo trabajador recibe la copia. Ver figura 4.31.
7. Todos los trabajadores reciben una nueva tabla con la información de los participantes que ahora están presentes en la simulación, incluido el nuevo trabajador. Ver figura 4.32.
8. El sistema reinicia la simulación a partir del estado global consistente que se registró antes que se presentara la falla de faro.

La figura 4.34, muestra el diagrama de secuencias para esta operación.

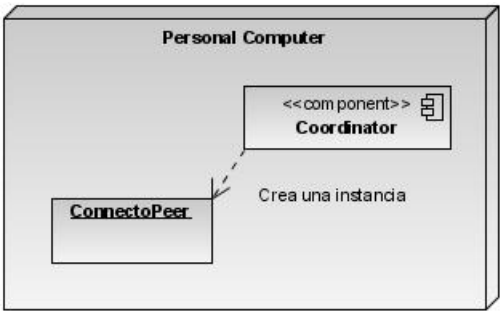


Figura 4.28: Componentes para una restauración.

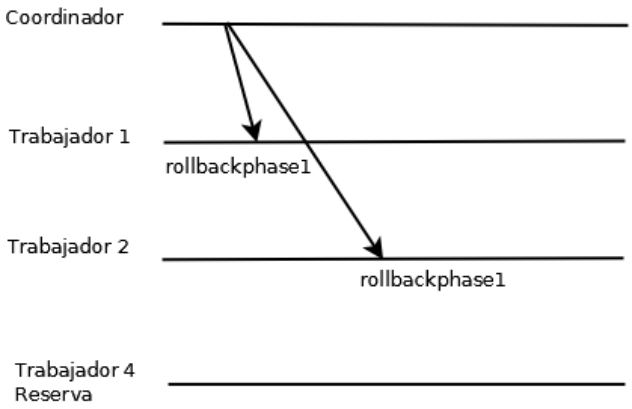


Figura 4.29: Envío del mensaje rollbackphase1.

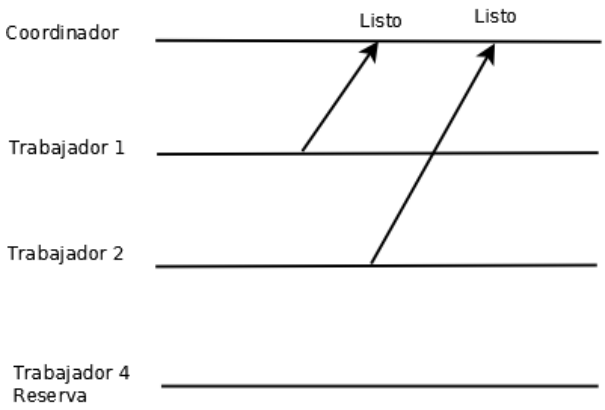


Figura 4.30: Envío mensaje "Listo" al coordinador.

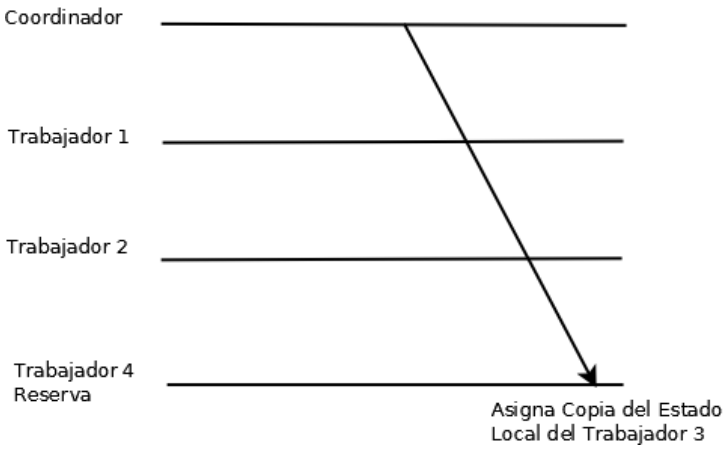


Figura 4.31: Recepción de trabajo.

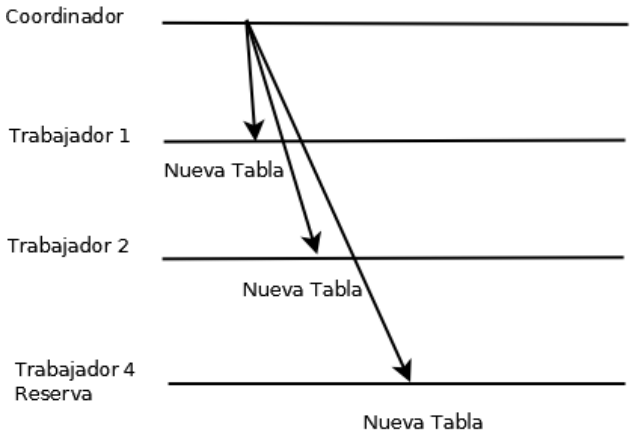


Figura 4.32: Recepción de nueva tabla.

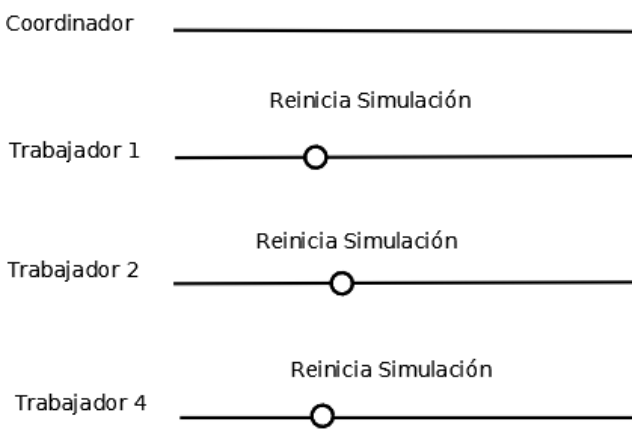


Figura 4.33: Sistema reiniciando simulación.

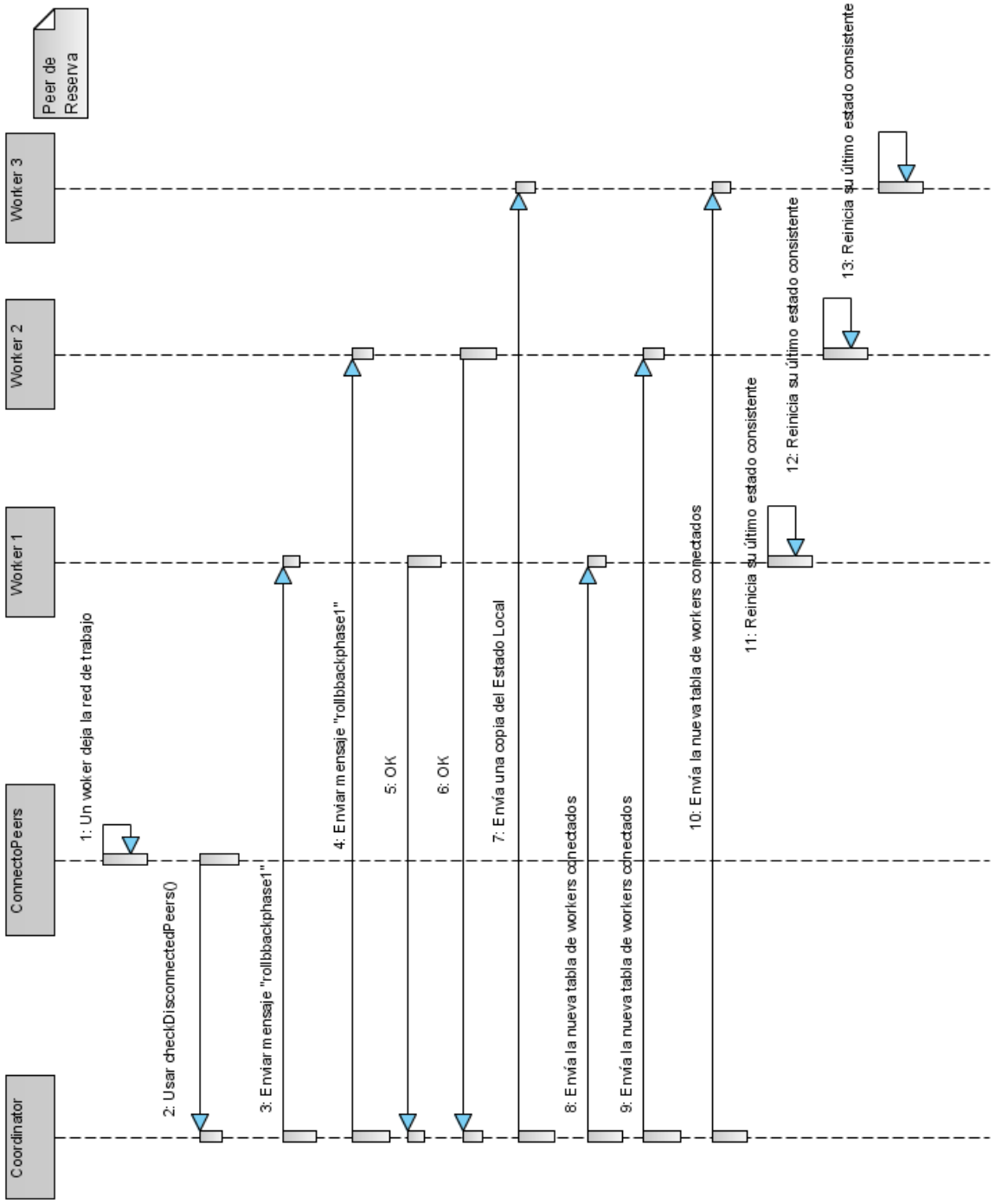


Figura 4.34: Diagrama de secuencias para restauración por falla de paro.

4.4 Operaciones que realiza un trabajador

En esta sección se detalla la única operación que realiza un trabajador dentro de una simulación distribuida: ejecución del método optimista.

La figura 4.35, muestra que una instancia de la clase Model, creará la instancia de una cola de entrada, una de salida y una de estados, que son requeridas para la óptima ejecución del método optimista.

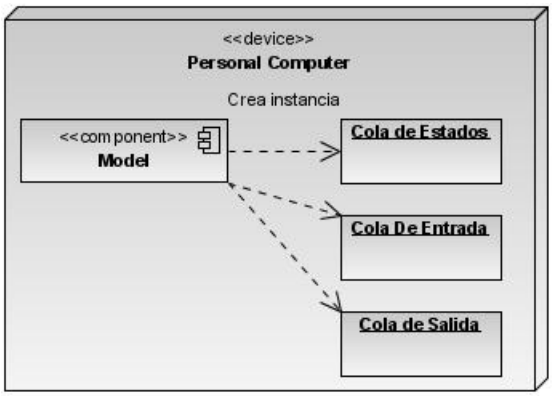


Figura 4.35: Componentes de la clase Model.

4.4.1 Operación: ejecución método optimista agresivo

El método optimista señala que se deben tener en cuenta los siguientes componentes a nivel proceso [14]:

1. Número del Proceso: Este número es el identificador del proceso.

2. Reloj Virtual Local: Es un valor numérico que indica el progreso de un proceso dentro de la simulación.

3. Estado: Es el estado en el que se encuentra un proceso después de haber ejecutado un evento. Este estado es guardado en la cola de estados.

4. Cola de estados: Contiene todos los estados previos por los cuales ha pasado este proceso. Este simulador salva el estado del proceso actual después de la ejecución de cada evento.

5. Cola de entrada (Input Queue): Contiene todos los mensajes recientemente ejecutados.

6. Cola de salida (Output Queue): Contiene copias de antimensajes que corresponden a los mensajes positivos enviados recientemente. Se usan en caso de restauración por el arribo tardío de un mensaje.

Es importante aclarar que en este prototipo todos los componentes menos el número del proceso, se utilizan en la clase Model. También hay que tomar en consideración los siguientes casos que se pueden presentar en el uso de antimensajes:

1. Si el mensaje positivo ya llegó, pero no ha sido procesado aún, su tiempo de recepción virtual debería ser más grande que el valor del reloj virtual del receptor. El antimensaje, teniendo el mismo tiempo de recepción virtual, al momento de su arribo causará aniquilación con el mensaje positivo dejando al receptor sin ningún registro de estos mensajes.

2. La segunda posibilidad es que el mensaje positivo tiene un tiempo de recepción virtual que está ahora en el presente o pasado con respecto al reloj virtual del receptor y puede haber sido ya parcialmente o completamente procesado, causando efectos colaterales sobre el estado del receptor. En este caso el antimensaje llegará y causará que el receptor haga una restauración al instante inmediato anterior en el que se recibió el mensaje positivo. Nótese que como resultado de la restauración, el proceso actual puede enviar antimensajes a otros procesos.

3. Un antimensaje puede también llegar en el destino antes que el mensaje positivo. En este caso, es encolado y será aniquilado cuando el mensaje positivo llegue.

En la arquitectura del trabajador (figura 4.4) cada instancia de la clase Model tiene tres estructuras de datos que reciben el nombre de inputQueue, outputQueue y stateQueue, que corresponden a la cola de entrada, de salida y de estados respectivamente. Estas estructuras son una instancia de la clase TreeMap (incluida en el lenguaje estándar de Java) y se encargan de guardar pares llave−valor. TreeMap ofrece métodos para poder recorrer esta estructura incluso en orden inverso, hacer consultas, etc.

La figura 4.36, muestra la conexión entre tres procesos y se toma como base para este ejemplo.

La forma como trabaja el método optimista es la siguiente:

1. En el proceso 1 se ejecuta el evento (5,m), ver figura 4.37. De la ejecución resultan dos mensajes cuyos destinos son el proceso 2 y proceso 3 respectivamente. El reloj virtual local de proceso 1 se ajusta a el valor 5.

2. El proceso 2 envía un mensaje con estampilla de tiempo menor que el valor del reloj virtual del proceso 1. Ver figura 4.38.

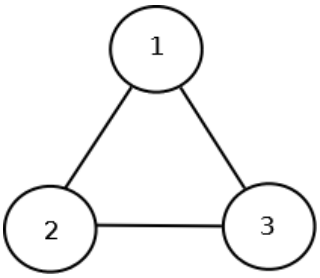


Figura 4.36: Relación entre procesos.

3. El mensaje que llega al proceso 1 tiene una estampilla de tiempo menor al reloj virtual local, por tanto se considera que llega retrasado.

4. El proceso 1, inicia el procedimiento de restauración, ya que se ha violado el orden causal; detiene la simulación y busca, en la cola de estados, el estado inmediato anterior cuya estampilla de tiempo es menor a la estampilla de tiempo del mensaje que llegó retrasado; una vez encontrado, el proceso 1 adquiere ese estado. Proceso 1 guarda en la cola de entrada tanto el mensaje que llegó retrasado como también los mensajes que deben reejecutarse. Ver figura 4.39.

5. El proceso 1 se encarga de enviar (si los hay) antimensajes. Los antimensajes se encuentran en lo que se conoce como la cola de salida. En este ejemplo se envían antimensajes a los procesos 2 y 3 para deshacer los cambios provocados por los mensajes anteriormente enviados. Ver figura 4.40.

6. En este ejemplo se supone que los mensajes originales aún no han sido procesados y por lo tanto se cancelan con la llegada de los antimensajes. Ver figura 4.41.

7. Por último la ejecución se reanuda en el orden adecuado. Ver figura 4.42.

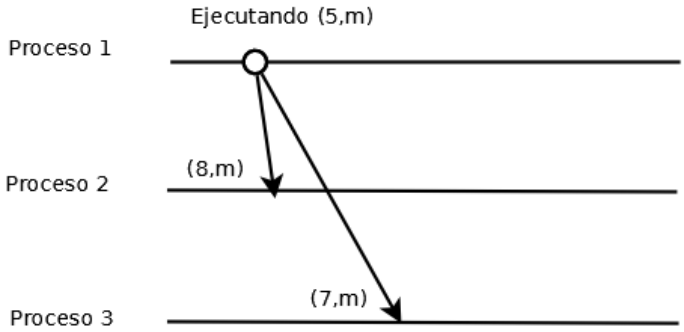


Figura 4.37: Ejecución optimista (1).

La figura 4.43, muestra el diagrama de secuencias para esta operación.

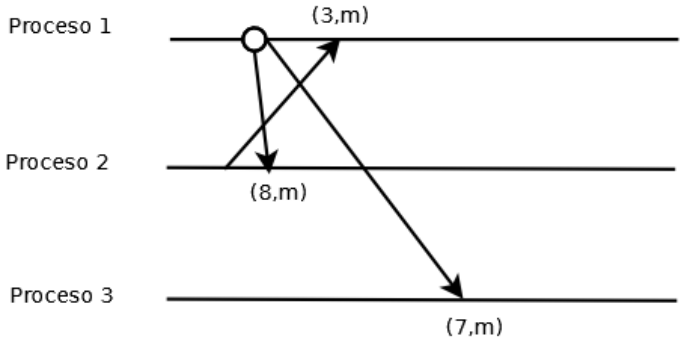


Figura 4.38: Ejecución optimista (2).

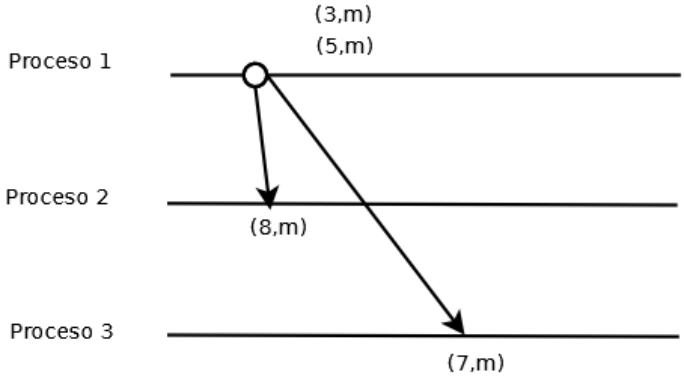


Figura 4.39: Ejecución optimista (3).

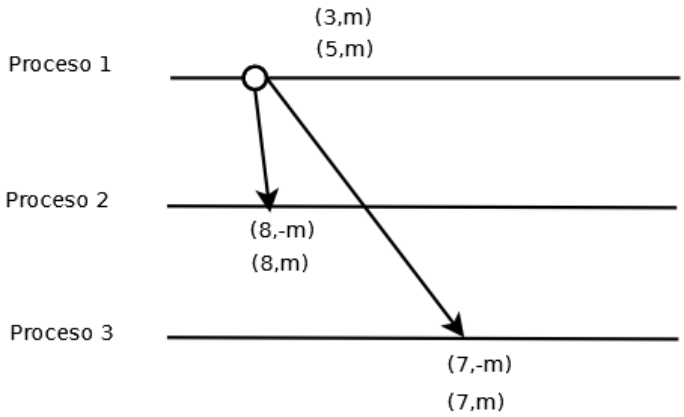


Figura 4.40: Ejecución optimista (4).

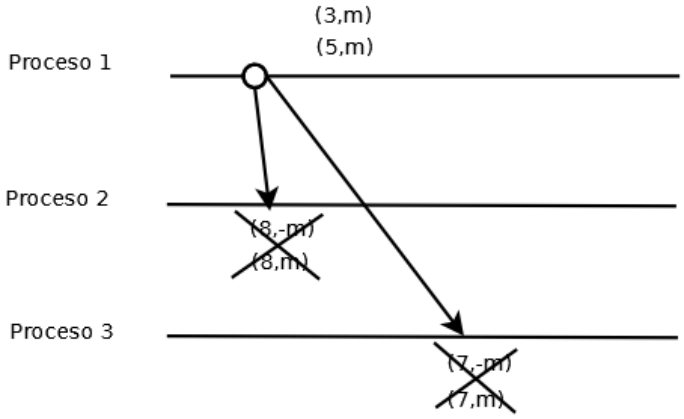


Figura 4.41: Aniquilación de mensajes.

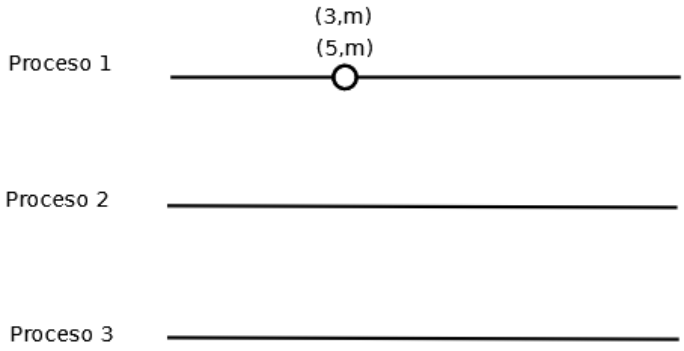


Figura 4.42: Ejecución ordenada.

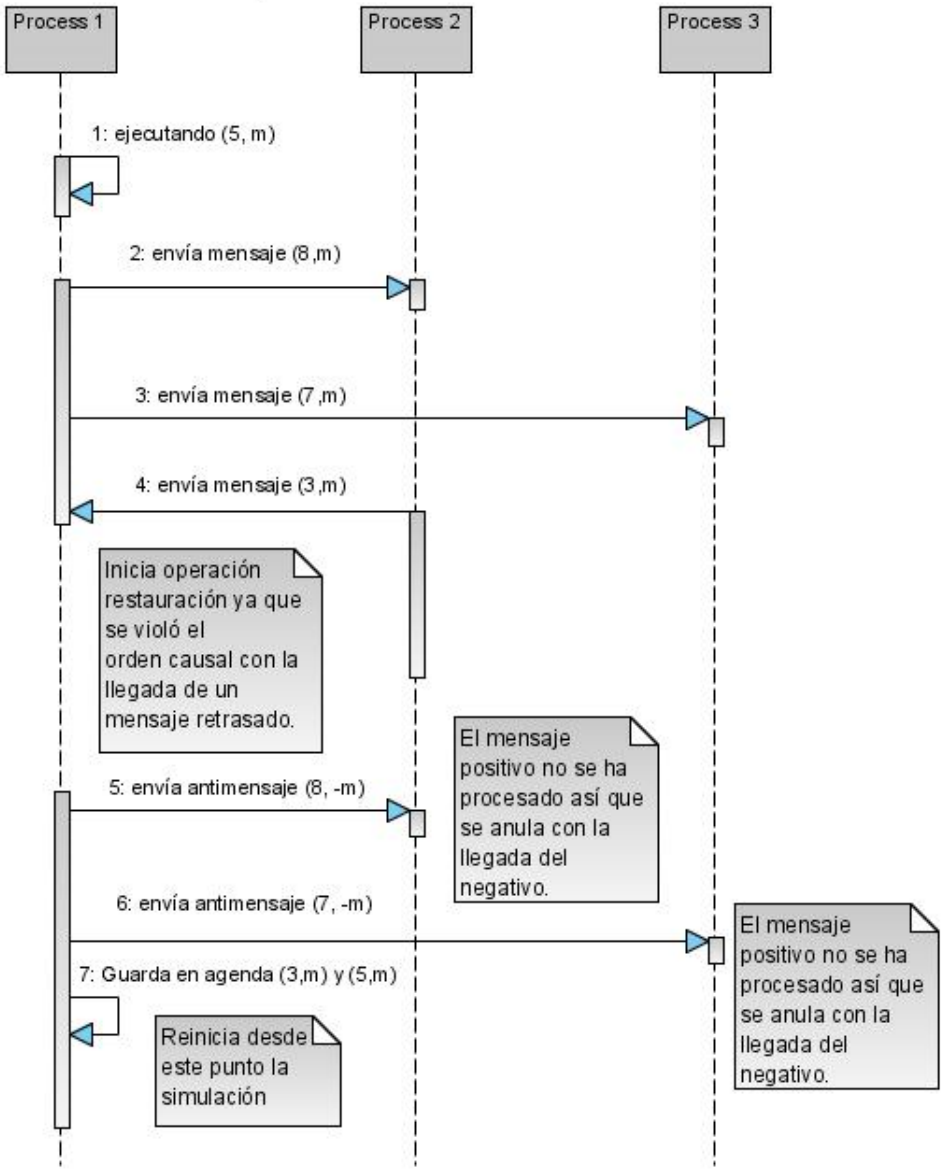


Figura 4.43: Diagrama de secuencias ejecución método optimista.

Capítulo 5

Evaluación de desempeño del prototipo

En este capítulo se describe el diseño de experimentos y las condiciones de prueba a las que fue sometido el prototipo DDES. De igual forma, se presentan los resultados de esta evaluación de desempeño.

5.1 Algoritmo PIF

Se utiliza el algoritmo de propagación de información con retroalimentación (PIF por sus siglas en inglés), para probar el desempeño del prototipo de simulador distribuido.

Una característica sobresaliente de este algoritmo es la etapa de retroalimentación con la cual un proceso tendrá conocimiento de cuando ha finalizado la propagación de información.

Este algoritmo es utilizado a menudo en los sistemas distribuidos para [30]:

1. Calcular la ruta más corta entre dos sitios arbitrarios.
2. Pruebas de conectividad.
3. Reconocer cambios en la topología de la red.

Para los algoritmos 4 y 5, se define lo siguiente:

1. Se utiliza un grafo donde se ejecutará el algoritmo.

2. M es el mensaje que se propaga en el grafo.

3. s, es el nodo del grafo donde inicia el algoritmo.

4. i,j son dos nodos arbitrarios pertenecientes al grafo.

5. N(i,j), es el registro donde i sabrá si ya recibió mensajes desde j.

6. padre(i),es el enlace por el que i recibe M la primera vez.

7. vecinos(i), son todos los nodos que comparten una arista con i

Algorithm 4 Algoritmo PIF en s

```
1:  Al recibir COMIENZA efectúa
2:    visitado(s)<-VERDADERO
3:    envia M a todo k en vecinos(s)
4:  Al recibir M, desde j, efectúa
5:    N(i,j)=1
6:    SI N(i,l)==1 para todo l en vecinos(s)
7:      entonces termina
```

Algorithm 5 Algoritmo PIF en i $\neq$ s

```
1:  Al recibir M, desde j, efectúa
2:  N(i,j)<-1
3:  SI visitado(i)==FALSO
4:      ENTONCES visitado(i)<- VERDADERO
5:      padre(i)<-j
6:      envía M a todo k en vecinos(i)-padre(i)
7:      SI N(i,l)==1 para todo l en vecinos(i)
8:      entonces envía M a padre(i)
9:      termina
10: OTRO
11:     SI N(i,l)==1 para todo l en vecinos(i)
12:     ENTONCES envía M a padre(i)
13:     termina
```

De forma general se enumeran a continuación las propiedades del algoritmo:

1. Todos los procesos envían el mensaje M a todos sus vecinos, salvo a su padre, en tiempo finito y exactamente una vez.
2. Todos los procesos reconocen cuando terminan.
3. Exactamente un mensaje es enviado por cada arista, en cada dirección.
4. Tiene un costo de $O(c)$ en mensajes. Donde c representa el número de canales de comunicación. La duración de la propagación es una función lineal del diámetro del grafo, es decir, de la máxima distancia que existe entre una pareja arbitraria de vértices que se comunican por la vía más corta.

A continuación se presenta una ejecución de ejemplo para el algoritmo PIF. Se usa el grafo de la figura 5.1 y se supone que tiene canales de comunicación con retardos aleatorios.

Este es el comportamiento del algoritmo:

1. Inicia el algoritmo en el proceso 1 enviando mensajes a sus vecinos, se envían los mensajes (1.7,M), (9.5,M) y (9.1,M) a los procesos 2,3 y 4 respectivamente, véase figura 5.2.
2. El proceso 2 recibe el mensaje y de acuerdo al algoritmo tiene que enviar a sus vecinos un mensaje, excepto a su padre, por tanto envía (4.3,M) al proceso 4. Ver figura 5.3.
3. El proceso 4 después de recibir el mensaje desde proceso 2, también recibe el mensaje (9.1,M) desde proceso 1. Proceso 4 envía el mensaje (11,M) para el proceso 1 y para el proceso 2 envía el mensaje (12,R) que indica que proceso 4 ya terminó la ejecución del algoritmo, nótese que ahora el mensaje es R y no M. Ver figura 5.4. Este mensaje R es la parte de retroalimentación del algoritmo PIF.
4. El proceso 3 finaliza el algoritmo y regresa el mensaje correspondiente a proceso 1. Ver figura 5.5.
5. El proceso 2 al recibir el mensaje de proceso 4, da por terminado el algoritmo PIF, y envía un mensaje a proceso 1. Ver figura 5.6.
6. Una vez que el proceso 1 recibe un mensaje a través de todos sus canales de entrada entonces da por terminado el algoritmo PIF.

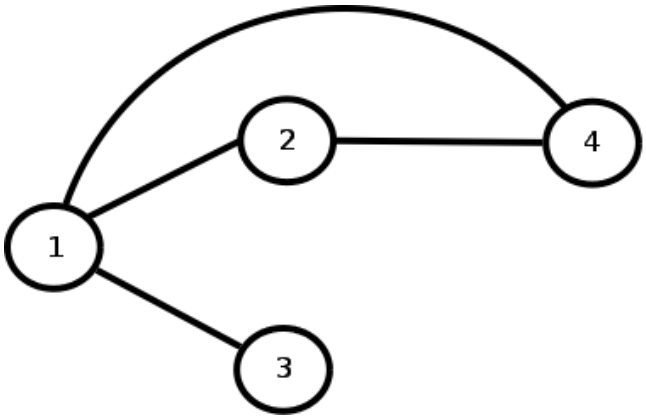


Figura 5.1: Grafo a simular.

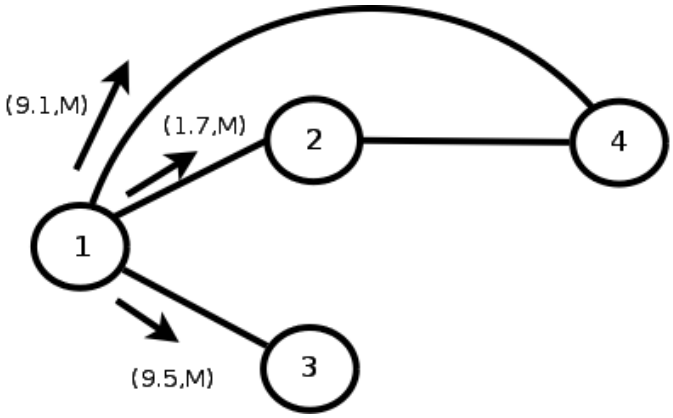


Figura 5.2: Inicia simulación.

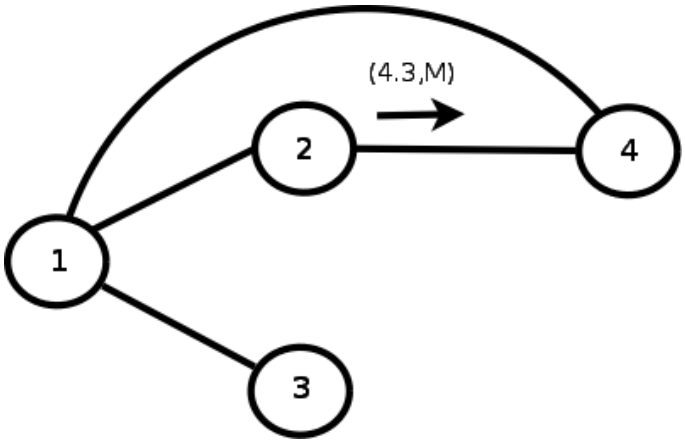


Figura 5.3: Proceso envía mensaje a proceso 4.

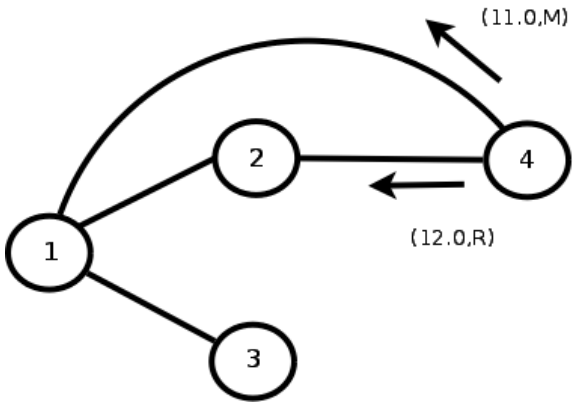


Figura 5.4: Envío de mensajes desde proceso 4.

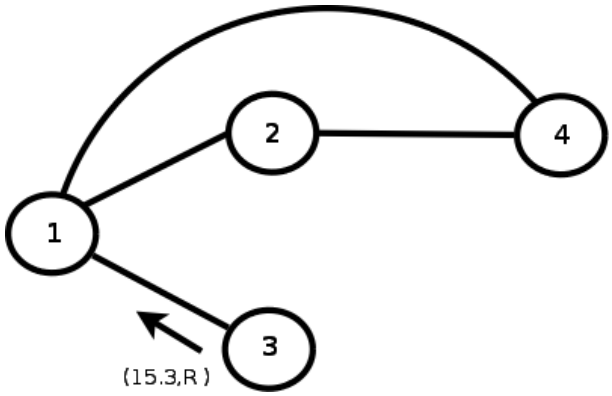


Figura 5.5: Proceso 3 finalizando algoritmo.

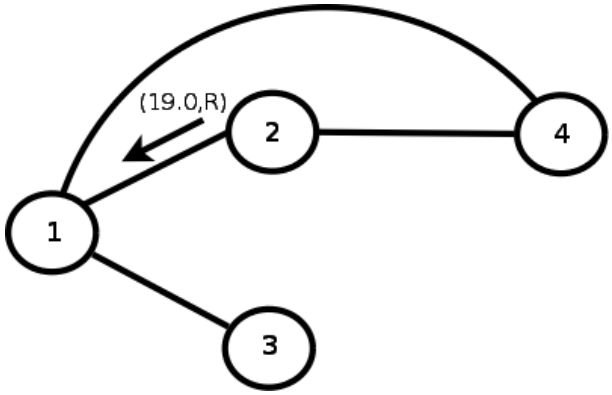


Figura 5.6: Proceso 2 finalizando el algoritmo.

5.2 Primer conjunto de pruebas

Se realizaron pruebas del simulador distribuido en un ambiente totalmente controlado, en un red local y utilizando computadoras personales.

La tabla 5.1 muestra el primer conjunto de pruebas que se aplicaron al simulador.

Algoritmo	Simulación	# Trabajadores	Fallas	Retardo	#Grafos	# Vértices
PIF	Centralizada	1	NO	constante	10	100
					⋮	⋮
					10	1000
	Distribuida	2	NO	constante	10	100
					⋮	⋮
					10	1000
				aleatorio	10	100
					⋮	⋮
					10	1000
			SI	constante	10	100
					⋮	⋮
					10	1000
				aleatorio	10	100
					⋮	⋮
					10	1000
		3	NO	constante	10	100
					⋮	⋮
					10	1000
				aleatorio	10	100
					⋮	⋮
					10	1000
			SI	constante	10	100
					⋮	⋮
					10	1000
				aleatorio	10	100
					⋮	⋮
					10	1000

Tabla 5.1: Primer conjunto de pruebas.

La tabla 5.1 se entiende de la siguiente manera: para los experimentos se utiliza el algoritmo PIF, se realizan dos tipos de simulación, centralizada y distribuida. Para la centralizada sólo se utiliza un trabajador, para sus contrapartes distribuidas se utilizan 2 y 3 trabajadores. De forma centralizada no se considera una falla de paro y el retardo en los canales es constante, por otro lado, para 2 y 3 trabajadores se tienen dos escenarios de prueba, sin y con falla de paro y para cada escenario se utilizan canales con retardos aleatorios o constantes. Y en general para todos los experimentos se utilizan 10 grafos distintos de 100 vértices cada grafo y así hasta 10 grafos de 1000 vértices cada uno.

Este primer conjunto de pruebas consta de 4 partes. Primero: se realizó un experimento de tipo centralizado, y también de tipo distribuido, sin utilizar los mecanismos de toma de estado global y cálculo de GVT, con la finalidad de calcular y comparar costos y factor de aceleración.

Segundo: se ejecuta de nuevo el conjunto de pruebas incluyendo los mecanismos de GVT y toma de estado global, con la finalidad de comparar costos y factor de aceleración, y determinar si es conveniente o no incluir dichos mecanismos.

Tercero: se realizaron experimentos de tipo distribuido sin y con los mecanismos ya mencionados, para poder comparar los costos que éstos implican.

Cuarto: se realizaron experimentos de tipo distribuido con y sin falla de paro y se determinó que tan costoso es recuperarse de una falla.

Para este conjunto de pruebas todos los grafos fueron creados de forma aleatoria, y no se garantiza que sean 100% conexos. Además se realizó una sola medición por cada grafo del experimento.

5.2.1 Calculando factor de aceleración y el costo

Para este primer conjunto de pruebas la forma en la que se determinó el costo [1] fue de la siguiente manera:

Para cada uno de los grupos de 10 grafos, se obtuvo la media, tanto para el caso centralizado como para los casos distribuidos (2 y 3 trabajadores).

$$costo = \frac{(media_c - media_d) * 100}{media_d}(\%)$$

Donde $media_c$ es la media para el caso centralizado y $media_d$ es la media para el

[1] Hay que entender por costo un porcentaje de tiempo que determinará que tan rápido se realiza una simulación distribuida en comparación con una centralizada. También se entiende como cuánto tiempo más se requiere, si se incluye la ejecución de los algoritmos de toma estado global y GVT.

caso distribuido.

El factor de aceleración[2] formalmente se describe como [39]:

$$Speedup = T(1)/T(N)$$

Donde:

1. T(1) es el mejor tiempo para una ejecución centralizada.
2. T(N) representa el tiempo de ejecución sobre N procesadores.

Para el caso de este primer conjunto de experimentos, T(1) corresponde a la media de un grupo de 10 grafos en el caso centralizado, de igual forma T(N) es la media pero para el caso distribuido.

5.2.2 Simulación centralizada vs distribuida sin fallas de paro y canales con retardos constantes

La figura 5.7, permite observar una comparativa entre la simulación centralizada y las simulaciones distribuidas. Se usa un trabajador para llevar a cabo una simulación centralizada y se distingue de sus contrapartes porque le corresponde una figura en forma de signo $'+'$. En este caso no se calcula GVT y tampoco se realiza una toma de estado global durante la simulación. Se puede observar que en una simulación de tipo centralizada, conforme crece el número de nodos en el grafo a simular, más tiempo se tarda la simulación, este comportamiento se observa a partir de los grafos que tienen 300 nodos y se hace más evidente en grafos con 400 nodos en adelante. Todo esto se debe a que si se incrementa la carga de trabajo, se requerirá de mayor tiempo de procesamiento y también se depende de otro factor, de la topología que tiene el grafo que se usa en la simulación lo cual implicará establecer mayor o menor número de canales de comunicación entre trabajadores.

Se puede apreciar, para el primer conjunto de pruebas, que se requiere menos tiempo en la simulación cuando se hace de forma distribuida.

No obstante, se puede observar en la gráfica 5.7[3] que no hay mucha diferencia en los tiempos de simulación al momento de usar 2 o 3 trabajadores, los tiempos son cercanos

[2]El factor de aceleración muestra que tanto mejora el rendimiento de una simulación distribuida comparada con una centralizada.

[3]En todas las gráficas de este trabajo, se utiliza SN que se refiere a la toma de estado global en una simulación distribuida.

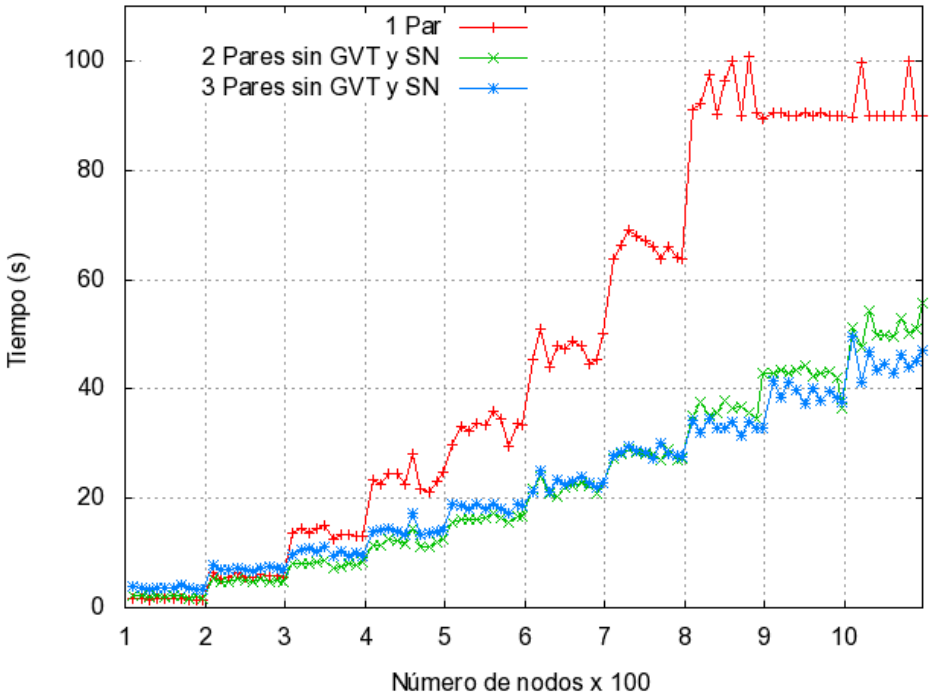


Figura 5.7: Ejecución centralizada vs distribuida.

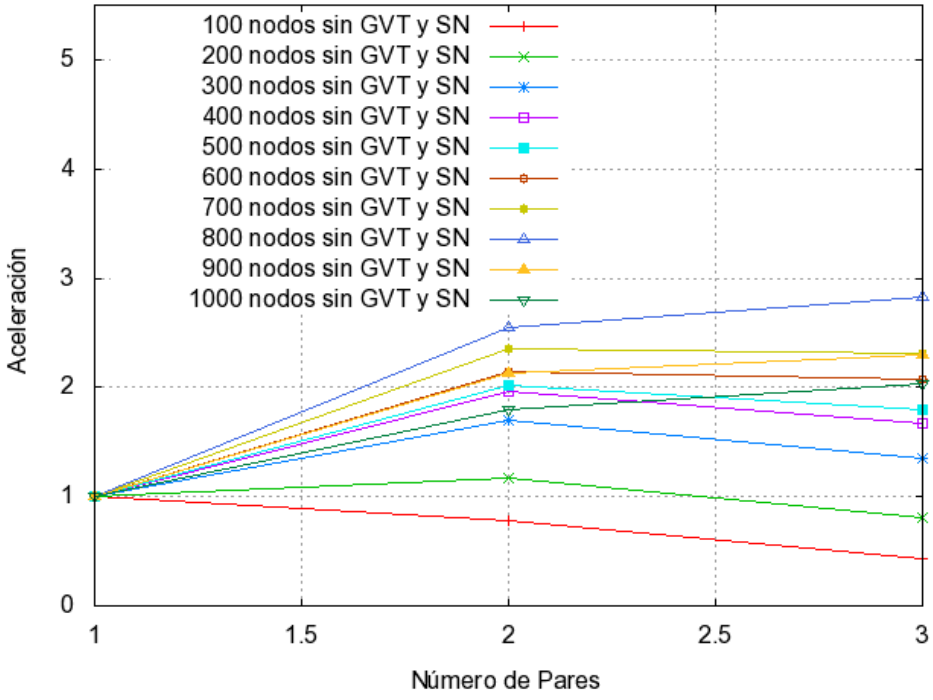


Figura 5.8: Factor de aceleración para 2 y 3 trabajadores.

aunque no iguales. Un comportamiento interesante de observar es que una simulación de 3 trabajadores, en el rango de 100 a 600 nodos, requiere un par de segundos más que su contraparte de 2, esto se debe a que se crean menos canales de comunicación entre 2 trabajadores que su contraparte de 3.

En el rango de 800 a 1000 una simulación con 2 trabajadores requiere de un par de segundos más que su contraparte de 3. Esto se debe a que conforme crece el número de nodos en el grafo, 2 trabajadores tendrán más carga de trabajo que su contraparte de 3. De igual forma se depende de la topología del grafo.

Para mostrar el factor de aceleración y el costo, se utiliza como base los resultados que muestra la figura 5.7, y se consideran dos casos:

1. Simulación centralizada vs distribuida con 2 trabajadores.
2. Simulación centralizada vs distribuida con 3 trabajadores.

Los canales de comunicación tienen retardos constantes y durante la simulación no se calcula GVT ni tampoco hay toma de estado global.

Caso: Simulación centralizada vs distribuida con 2 trabajadores

En esta sección se hace un análisis del factor de aceleración y del costo que implica una simulación centralizada comparada con una distribuida.

# Nodos	Costo (%)
100	-22.14
200	17.57
300	69.52
400	96.11
500	101.64
600	114.45
700	135.79
800	155.21
900	112.59
1000	79.39

Tabla 5.2: Costo centralizado vs distribuido con 2 trabajadores.

La tabla 5.2, muestra qué tan rápida es la simulación distribuida con respecto a la centralizada, primero, existe un dato que se debe analizar, para grafos de 100 nodos se tiene un valor negativo de -22.14 %, esto significa que para este tipo de grafos es mejor

realizar una simulación de tipo centralizada. Por otro lado, para los demás grafos se recomienda que sean simulados con 2 trabajadores ya que por ejemplo para grafos de 1000 nodos, la simulación con 2 trabajadores es un 79.39 % más rápida. Para los demás datos se puede razonar de la misma forma.

# Nodos	Aceleración
100	0.77
200	1.17
300	1.69
400	1.96
500	2.01
600	2.14
700	2.35
800	2.55
900	2.12
1000	1.79

Tabla 5.3: Factor de aceleración con 2 trabajadores.

La tabla 5.3 muestra el factor de aceleración utilizando 2 trabajadores. Por ejemplo para un grafo de 500 nodos, la simulación se acelera al doble, es decir, la simulación distribuida es dos veces más rápida que la centralizada. Un razonamiento similar se aplica a los demás datos. De hecho para el grafo con 100 nodos conviene más que sea realizado de forma centralizada.

Caso: Simulación centralizada vs distribuida con 3 trabajadores

# Nodos	Costo (%)
100	-56.19
200	-18.68
300	34.38
400	66.77
500	79.00
600	107.03
700	131.70
800	182.53
900	129.72
1000	103.80

Tabla 5.4: Costo centralizado vs distribuido con 3 trabajadores.

La tabla 5.4, muestra qué tan rápida es la simulación distribuida con respecto a la centralizada, primero existe un par de datos que se deben analizar, para grafos de 100

y 200 nodos se tiene un valor negativo, esto significa que para este tipo de grafos es mejor realizar una simulación de tipo centralizada. Es conveniente usar una simulación distribuida (con 3 trabajadores) a partir de grafos de 300 nodos en adelante. Como un ejemplo se puede apreciar que para grafos de 800 nodos es 182.53 % más rápida la simulación distribuida.

# Nodos	Aceleración
100	0.43
200	0.81
300	1.34
400	1.66
500	1.79
600	2.07
700	2.31
800	2.82
900	2.29
1000	2.03

Tabla 5.5: Factor de Aceleración con 3 trabajadores.

Comparando la tabla 5.5 con la tabla 5.3, la aceleración para 3 trabajadores no presenta una mejora notable comparado con la simulación con 2 trabajadores. En caso que se busque optimizar recursos, es una excelente opción utilizar 2 trabajadores. Es importante notar que para grafos de 100 y 200 nodos es mejor realizar una simulación centralizada.

Análisis: gráfica factor de aceleración

La gráfica de la figura 5.8, corresponde a los factores de aceleración, para los dos casos mencionados recientemente. Hay que observar que para todo el primer conjunto de pruebas (sin calcular estado global y GVT), el factor de aceleración aumentó muy poco cuando se usan 3 trabajadores. En resumen, de la figura 5.7 y la 5.8, se infiere que se recomienda utilizar 2 trabajadores para una simulación con grafos de 200 a 500 nodos, de 600 a 700 se puede utilizar 2 o 3 trabajadores, y de 800 a 1000 nodos se recomienda utilizar 3 trabajadores.

5.2.3 Análisis de los costos incluyendo estado global y GVT para el primer conjunto de pruebas

En esta sección se muestra el costo de incluir los algoritmos de toma de estado global y cálculo de GVT en una simulación distribuida y se compara con la centralizada que no

incluye dichos algoritmos. Y se determina si conviene o no incluir dichos mecanismos.

# Nodos	Costo (%)
100	-67.62
200	-46.28
300	-23.84
400	-4.42
500	-0.78
600	13.27
700	20.50
800	42.09
900	14.78
1000	0.58

Tabla 5.6: Costo al incluir algoritmos de estado global y GVT para 2 trabajadores.

La tabla 5.6 muestra el costo de incluir dichos mecanismos utilizando 2 trabajadores. La figura 5.9, muestra el comportamiento al momento de incluir los algoritmos ya mencionados. Por tanto, de la tabla 5.6 y la figura 5.9, se puede inferir que para grafos con nodos de 600 a 1000 ya se recomienda utilizar la simulación con 2 trabajadores.

# Nodos	Costo (%)
100	-62.68
200	-35.53
300	4.85
400	32.62
500	39.57
600	63.45
700	88.52
800	124.90
900	81.30
1000	64.28

Tabla 5.7: Costo al incluir algoritmos de estado global y GVT para 3 trabajadores.

La tabla 5.7 y la figura 5.9, muestran los costos y el comportamiento respectivamente de una simulación al utilizar 3 trabajadores. Y se puede inferir que para grafos de 100 a 200 nodos se recomienda una simulación centralizada. En cambio, para grafos con nodos de 300 a 1000 ya conviene utilizar la simulación distribuida con 3 trabajadores.

En la figura 5.9, se aprecia que a pesar de incluir los mecanismos de GVT y de estado global, la simulación con 3 trabajadores presenta un mejor rendimiento. Esto se debe a que 2 trabajadores tienen una mayor carga de trabajo que su contraparte de 3.

En la tabla 5.8, se muestra el factor de aceleración para 2 y 3 trabajadores.

Por otro lado, cabe mencionar que realizando una comparativa entre las tablas 5.6 y 5.7 y apoyándose en las figuras 5.9 y 5.10, se puede inferir que para este primer conjunto de pruebas es conveniente utilizar la simulación distribuida con 3 trabajadores, incluyendo la ejecución de los algoritmos de toma de estado global y cálculo de GVT, para simulaciones con grafos de 300 a 1000 nodos.

# Nodos	2 Pares	3 Pares
100	0.32	0.37
200	0.53	0.64
300	0.76	1.04
400	0.95	1.32
500	0.99	1.39
600	1.13	1.63
700	1.20	1.88
800	1.42	2.24
900	1.14	1.81
1000	1.00	1.64

Tabla 5.8: Factor de aceleración incluyendo algoritmos de estado global y GVT para 2 y 3 trabajadores.

5.2.4 Costos por incluir los algoritmos de estado global y GVT usando 2 trabajadores

En esta sección se determina que tan caro resulta incluir la ejecución de los algoritmos de toma de estado global y cálculo de GVT, durante una simulación distribuida dónde no hay falla de paro, se compara contra una simulación distribuida que no ejecuta dichos algoritmos. Los algoritmos se ejecutan una sola vez durante la simulación.

El costo se calcula de la siguiente manera:

$$costo = \frac{(media_{dsn} - media_d) * 100}{media_d}(\%)$$

Donde $media_{dsn}$ es la media para el caso cuando se calcula el GVT y la toma de estado global y $media_d$ es la media para el caso distribuido que no incluye dichos mecanismos. Esta fórmula se utiliza para el caso de 3 trabajadores también.

La figura 5.11 muestra una comparativa entre dos simulaciones de tipo distribuido con dos trabajadores, con y sin toma de estado global y cálculo de GVT.

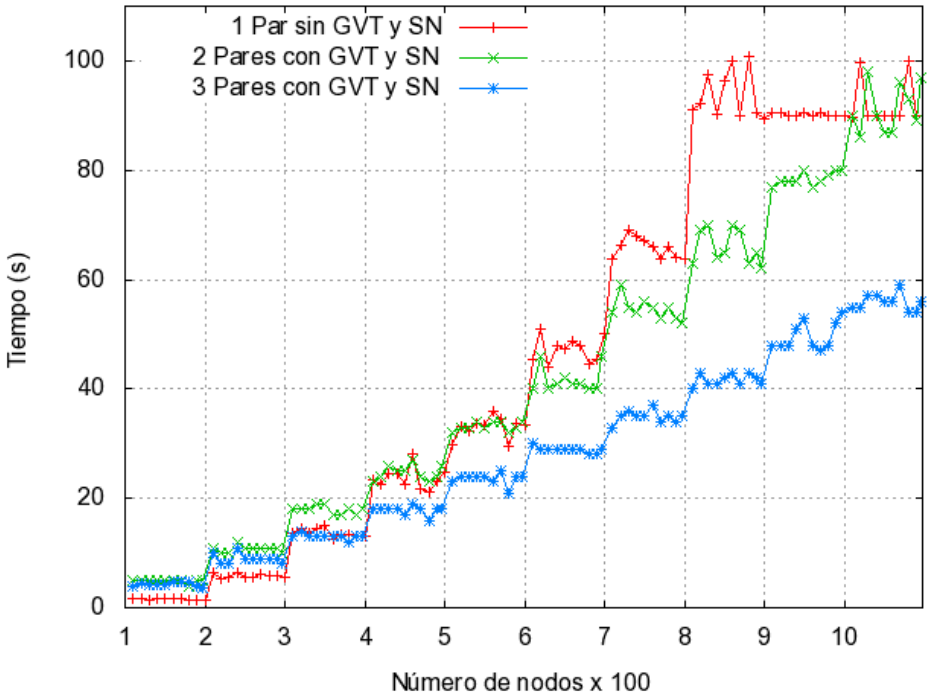


Figura 5.9: Experimentos centralizado vs distribuido incluyendo cálculo de GVT y estado global.

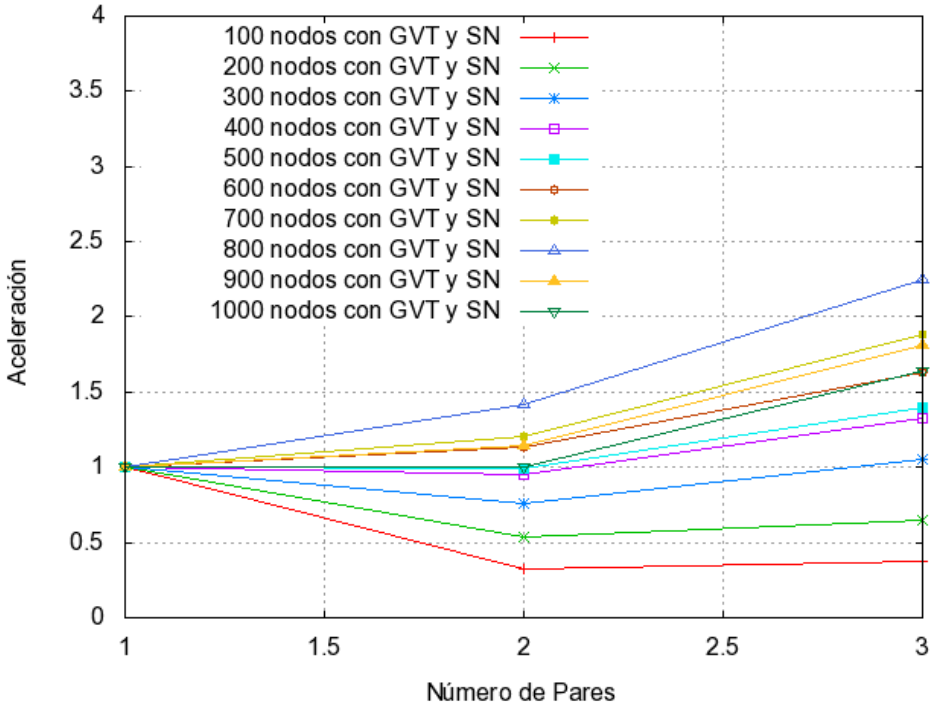


Figura 5.10: Factor de aceleración centralizado vs distribuido incluyendo cálculo de GVT y estado global.

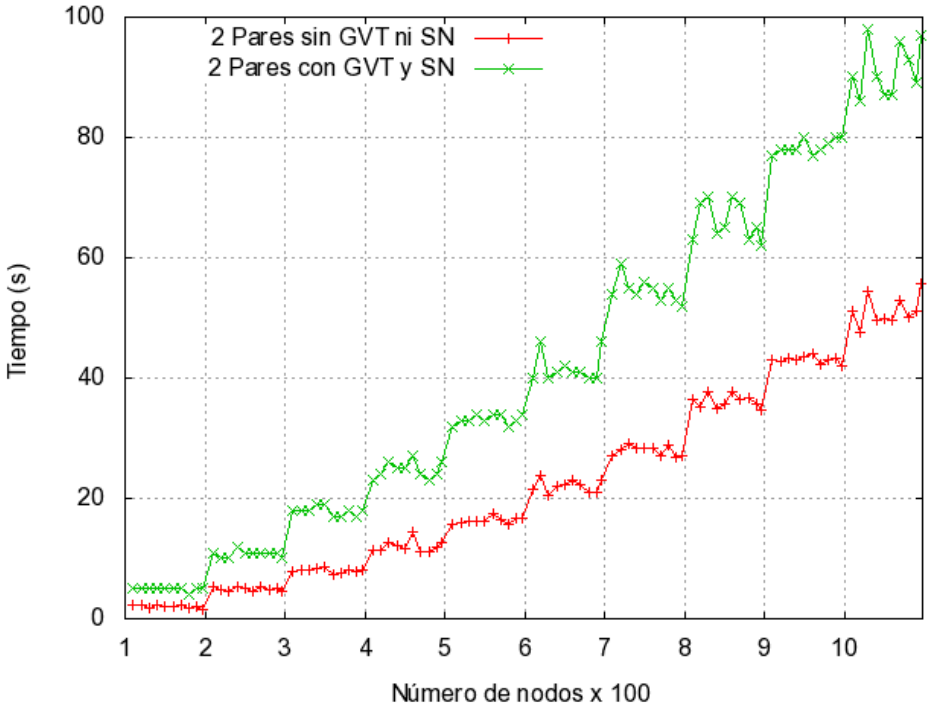


Figura 5.11: Comparativa de una simulación con 2 trabajadores sin y con GVT y toma de estado.

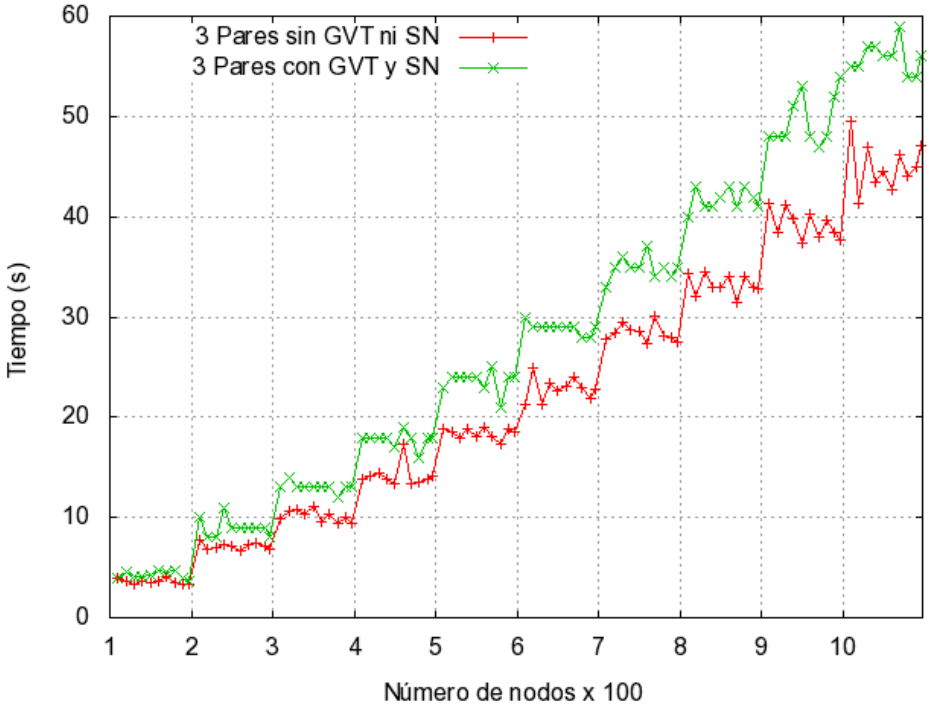


Figura 5.12: Comparativa de una simulación con 3 trabajadores sin y con GVT y toma de estado.

De la tabla 5.9, se puede observar por ejemplo, que para un grafo de 800 nodos se requerirá de un 82.82 % más de tiempo para finalizar la simulación.

# Nodos	Costo (%)
100	139.69
200	118.75
300	122.71
400	105.04
500	103.17
600	89.32
700	95.57
800	82.82
900	82.51
1000	78.31

Tabla 5.9: Costos con 2 trabajadores incluyendo algoritmos de estado global y GVT.

5.2.5 Costos por incluir los algoritmos de estado global y GVT usando 3 trabajadores

En esta sección se determina que tan caro resulta incluir los mecanismos de toma de estado global y cálculo de GVT dentro de una simulación que ocupa 3 trabajadores y no hay falla de paro. Se compara con una simulación que no incluye dichos mecanismos.

La figura 5.12 muestra este caso. Como ejemplo de la tabla 5.10, un grafo de 300 nodos requerirá de 28.24 % más de tiempo al incluir estos mecanismos.

# Nodos	Costo (%)
100	17.47
200	26.16
300	28.24
400	25.75
500	28.25
600	26.66
700	22.90
800	25.63
900	26.77
1000	24.05

Tabla 5.10: Costo con 3 trabajadores incluyendo algoritmos de estado global y GVT.

5.2.6 Simulación con dos trabajadores y canales con retardos constantes, con y sin falla de paro

En esta sección se analiza que tan costosa es una falla de paro cuando se presenta en una simulación distribuida, utilizando 2 trabajadores.

En este experimento se realizaron simulaciones utilizando dos trabajadores, y canales con retardos constantes. El primer experimento consistió en llevar a cabo todo el conjunto de pruebas sin que se presentara una falla de paro, por otro lado la segunda ejecución del conjunto de pruebas se realizaron con una falla de paro. La falla de paro ocurrió después de la primera toma de estado global.

Los experimentos que presentan fallas de paro en la figura 5.13 se representan mediante el símbolo 'x'. Se observa que una simulación necesitará de más tiempo para ser completada una vez que la falla de paro está presente. Por el contrario cuando no hay alguna falla de paro la simulación requiere de menos tiempo para su finalización. De la tabla 5.11, por ejemplo, para un grafo de 500 nodos si se presenta una falla de paro, la simulación requerirá de 293.16 % más de tiempo para terminar. Otra forma como se puede entender esta tabla es: para grafos de 500 nodos si no hay una falla de paro, la simulación será un 293.16 % más rápida.

# Nodos	Costo (%)
100	2277.03
200	943.78
300	579.56
400	386.35
500	293.16
600	218.99
700	175.86
800	129.65
900	104.85
1000	86.91

Tabla 5.11: Costo con 2 trabajadores, con y sin falla de paro y canales con retardos constantes.

5.2.7 Simulación con dos trabajadores y canales con retardos aleatorios, con y sin falla de paro

En este experimento se realizaron simulaciones utilizando dos trabajadores, y canales con retardos aleatorios. El primer experimento consistió en llevar a cabo todo el conjunto de pruebas sin que se presentara una falla de paro, por otro lado la segunda

ejecución se realizó con una falla de paro. La falla de paro ocurrió después de la primera toma de estado global.

La figura 5.14, muestra un comportamiento muy parecido al que se describió en la sección 5.2.6, la única diferencia en el experimento es que se utilizaron canales con retardos aleatorios. De la tabla 5.12, por ejemplo, para un grafo de 500 nodos, si se presenta una falla de paro, la simulación requerirá de 257.81 % más de tiempo para terminar.

# Nodos	Costo (%)
100	2134.30
200	898.66
300	555.97
400	372.35
500	257.81
600	209.44
700	162.62
800	141.00
900	121.12
1000	96.63

Tabla 5.12: Costo con 2 trabajadores, con y sin falla de paro y canales con retardos aleatorios.

5.2.8 Simulación con tres trabajadores y canales con retardos constantes, con y sin falla de paro

En esta sección se analiza que tan cara es una falla de paro cuando se presenta en una simulación distribuida, utilizando 3 trabajadores.

En este experimento se realizaron simulaciones utilizando tres trabajadores y canales con retardos constantes. El primer experimento consistió en llevar a cabo todo el conjunto de pruebas sin que se presentara una falla de paro, por otro lado en la segunda ejecución se realizaron con una falla de paro. La falla de paro ocurrió después de la primera toma de estado global.

Los experimentos que presentan fallas de paro en la figura 5.15 se representan mediante el símbolo 'x'. Se observa que una simulación necesitará de más tiempo para ser completada una vez la falla de paro está presente. Por el contrario cuando no hay alguna falla de paro la simulación requiere de menos tiempo para su finalización. Los costos se pueden observar en la tabla 5.13.

Por ejemplo, para un grafo con 1000 nodos se requiere de un 81.80 % más de tiempo

Figura 5.13: Canales con retardos constantes sin falla vs con falla de paro.

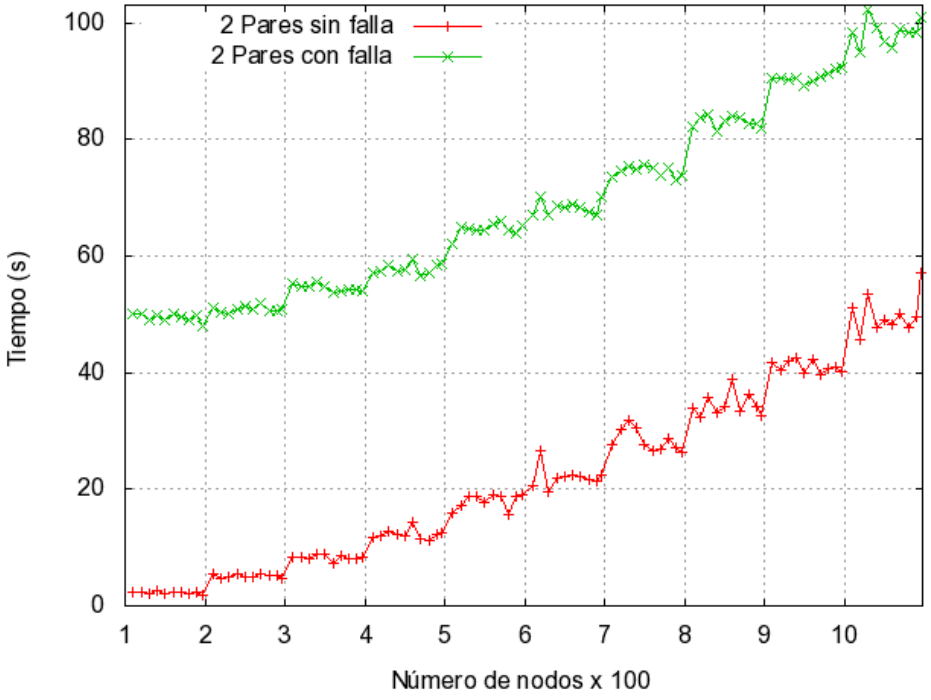


Figura 5.14: Canales con retardos aleatorios sin falla vs con falla de paro.

para que termine la simulación, cuando ocurre una falla de paro.

# Nodos	Costo (%)
100	1241.04
200	614.68
300	426.46
400	298.20
500	223.56
600	183.12
700	139.97
800	117.38
900	102.30
1000	81.80

Tabla 5.13: Costo con 3 trabajadores, con y sin falla de paro y canales con retardos constantes.

5.2.9 Simulación con tres trabajadores y canales con retardos aleatorios, con y sin falla de paro

En este experimento se realizaron simulaciones utilizando tres trabajadores, y canales con retardos aleatorios. El primer experimento consistió en llevar a cabo todo el conjunto de pruebas sin que se presentara una falla de paro, por otro lado en la segunda ejecución se realizaron con una falla de paro. La falla de paro ocurrió después de la primera toma de estado global.

Los experimentos que presentan fallas de paro en la figura 5.16 se representan mediante el símbolo 'x'. Se observa que una simulación necesitará de más tiempo para ser completada una vez la falla de paro esté presente. Por el contrario cuando no hay alguna falla de paro la simulación requiere de menos tiempo para finalizar.

De la tabla 5.14, por ejemplo, para un grafo de 1000 nodos, si se presenta una falla de paro, la simulación requerirá de 87.47 % más de tiempo para terminar.

5.2.10 Canales de comunicación con retardos constantes vs aleatorios

El análisis que se presenta a continuación, aplica por un lado a una comparativa entre las gráficas 5.13 y 5.14, y por otro lado entre las gráficas 5.15 y 5.16.

Es posible apreciar que tienen como resultado tiempos muy parecidos, a pesar de

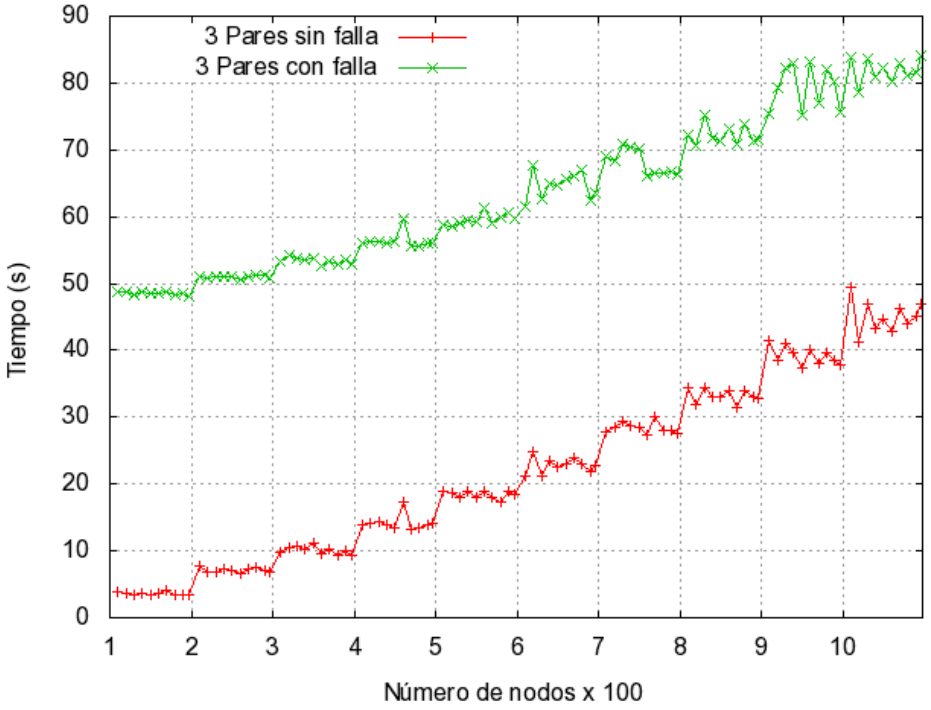


Figura 5.15: Canales con retardos constantes sin falla vs con falla de paro.

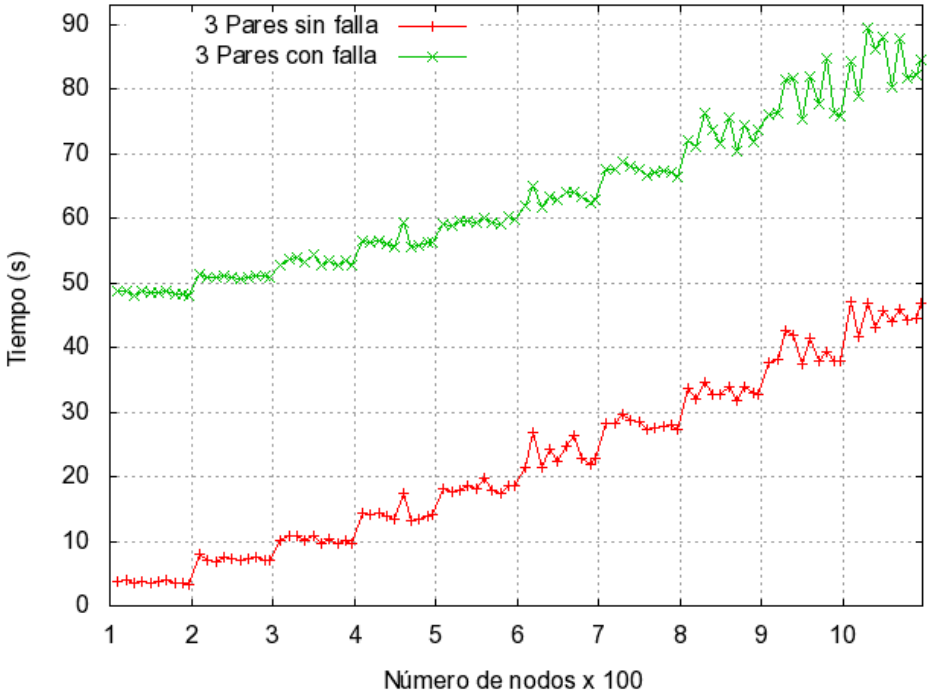


Figura 5.16: Canales con retardos aleatorios sin falla vs con falla de paro.

# Nodos	Costo (%)
100	1218.63
200	602.81
300	420.38
400	297.05
500	225.97
600	168.63
700	139.62
800	120.61
900	100.75
1000	87.47

Tabla 5.14: Costo con 3 trabajadores, con y sin falla de paro y canales con retardos aleatorios.

haber involucrado la diferencia de usar canales de comunicación con retardos constantes y aleatorios. Esto se debe a que ambas ejecuciones se realizaron en igualdad de condiciones.

También se observa que una simulación necesitará de más tiempo, para ser completada, cuando una falla de paro esté presente. Para explicar este comportamiento hay que considerar que una instancia de ConnectoPeer revisa, cada cierto tiempo, que los trabajadores estén conectados, después que ConnectoPeer detecta la falla de paro, avisa al coordinador y se realiza la restauración, aquí hay que tener en cuenta dos casos: primero cuando la falla de paro se presenta inmediatamente después de que ConnectoPeer revisó la presencia de los trabajadores. Segundo, cuando ConnectoPeer inicia la verificación inmediatamente después que se presentó una falla de paro. Todo esto permite concluir que se necesitará de más o menos tiempo a partir del instante en el que se presentó la falla de paro. Además hay que agregar el tiempo restante para finalizar la simulación. Considerando todo esto es posible explicar el por qué del incremento de tiempo para finalizar la simulación cuando hay una falla de paro.

Por otro lado hay que mencionar también que en los experimentos que no presentan falla de paro se omitió la toma de estado global, esto resta algún tiempo en los resultados de esa simulación, hay que tomar en cuenta que la toma de estado global implica envío de mensajes y además detiene la simulación hasta que termine la toma de estado global, después deja continuar la simulación.

Otro factor importante es que el GVT se calcula de la misma forma que se realiza una toma de estado global, con la única diferencia que no se guarda nada en un medio estable, con lo cual se tendrá la creación de más canales de comunicación y envío de mensajes.

5.3 Segundo conjunto de pruebas

En esta sección se presenta un segundo conjunto de pruebas a las que se sometió el prototipo de simulador, se utilizaron grafos de 1400, 1800, 2200, 2600 y 3000 nodos. Los canales de comunicación tienen retardos constantes y no se garantiza que los grafos sean 100 % conexos.

Algoritmo	Simulación	Trabajadores	Fallas	Retardo	#Grafos	# Vértices
PIF	Centralizada	1	NO	constante	1	1400
					1	1800
					1	2200
					1	2600
					1	3000
	Distribuida	2	NO	constante	1	1400
					1	1800
					1	2200
					1	2600
					1	3000
		3	NO	constante	1	1400
					1	1800
					1	2200
					1	2600
					1	3000
		4	NO	constante	1	1400
					1	1800
					1	2200
					1	2600
					1	3000
		5	NO	constante	1	1400
					1	1800
					1	2200
					1	2600
					1	3000

Tabla 5.15: Segundo conjunto de pruebas.

La tabla 5.15, se entiende de la siguiente manera, para el segundo conjunto de pruebas se utiliza el algoritmo PIF, se realizan dos tipos de simulaciones una centralizada y 4 distribuidas. Las simulaciones distribuidas se realizaron con 2,3,4 y 5 trabajadores. Para todas las simulaciones no hay fallas de paro y los canales tienen retardos constantes de tiempo. Y por cada tipo de simulación se usa 1 grafo de 1400, 1800, 2200, 2600 y 3000 nodos.

Este segundo conjunto de pruebas consta de 3 partes. Primero: se realizó un experimento de tipo centralizado sin toma de estado global ni GVT porque no tienen sentido estos mecanismos en una simulación que ocurre sobre una sola máquina. Estos experimentos son con la finalidad de poder comparar un escenario centralizado contra varios escenarios distribuidos, sin la presencia de dichos mecanismos. Se determina solamente el costo.

Segundo: se realizaron experimentos de forma distribuida con toma de estado global y cálculo de GVT, con la finalidad de compararlos con el experimento de tipo centralizado y determinar si es conveniente ejecutar o no dichos mecanismos. También se determina el costo.

Tercero: se realizan experimentos de forma distribuida, primero se lleva a cabo una ejecución de los mecanismos ya mencionados, y segundo se lleva acabo una ejecución sin los mecanismos, con la finalidad de poder calcular el costo que esto implica.

Se realizó una sola medición por cada grafo del experimento.

El costo se calcula de la misma forma como se realizó en el primer conjunto de pruebas.

5.3.1 Costos sin incluir estado global y GVT para el segundo conjunto de pruebas

La figura 5.17 muestra una comparativa entre una simulación centralizada y sus contrapartes para 2,3,4 y 5 trabajadores. Y hace evidente para este segundo conjunto de pruebas, que la simulación distribuida ofrece un mejor tiempo en las simulaciones. No está incluido el cálculo de GVT ni la toma de estado global.

De hecho en términos generales la simulación distribuida es más conveniente, en este segundo conjunto de pruebas, que la centralizada, y esto se puede inferir al revisar la tabla 5.16. Es posible observar que el mejor escenario consiste en utilizar 5 trabajadores [4], ya que por ejemplo para un grafo de 3000 nodos, la simulación resulta en un 854.16 % más rápida que la centralizada.

[4]Por razones de espacio, en las columnas de las tablas se utiliza el término par en lugar de trabajador.

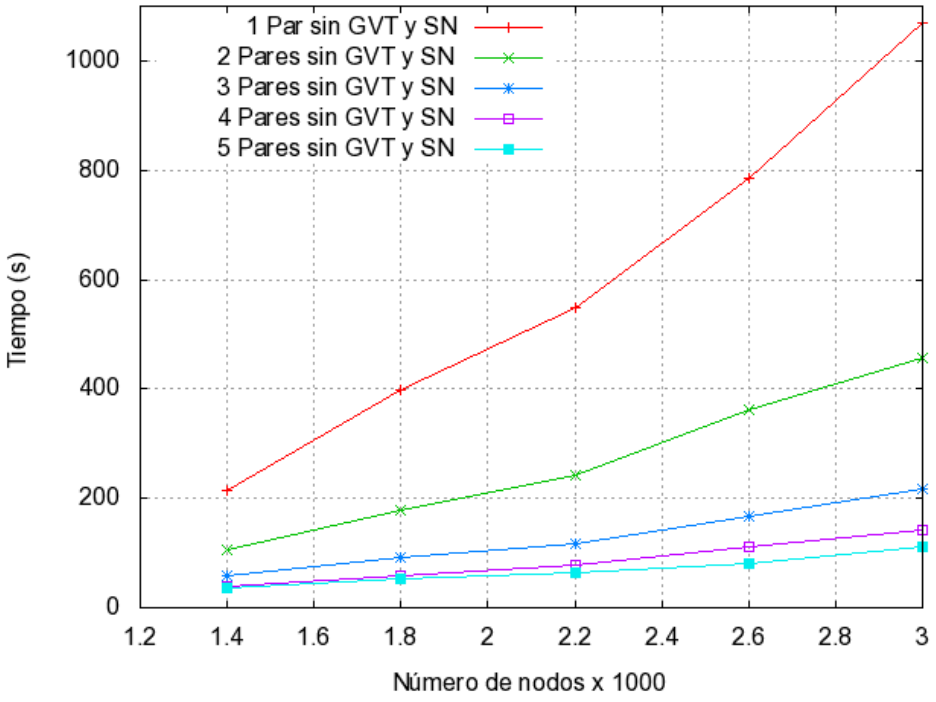


Figura 5.17: Experimentos centralizado vs distribuido.

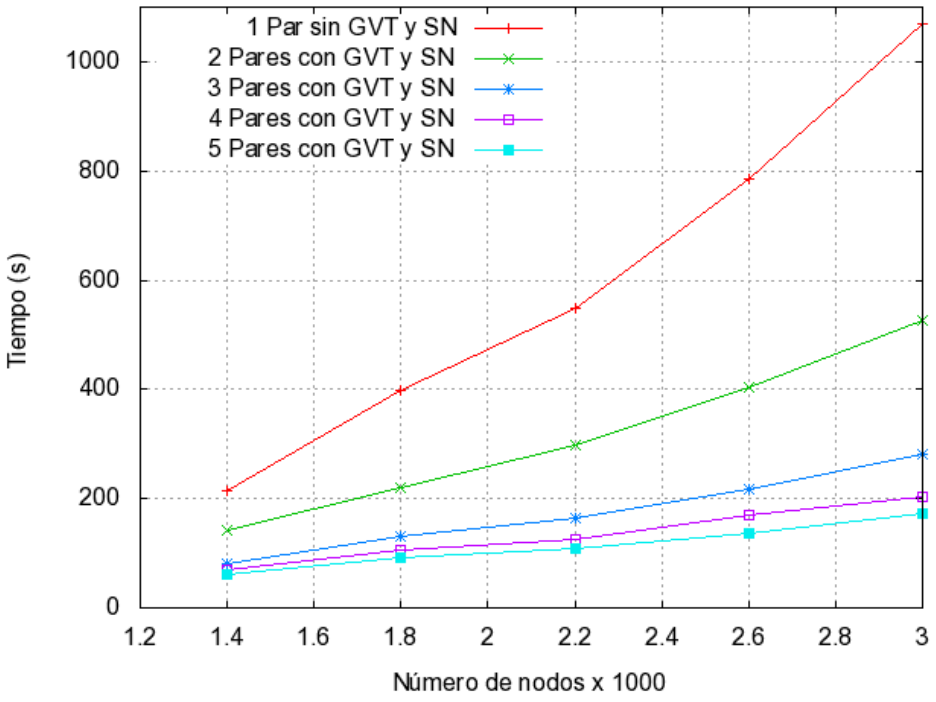


Figura 5.18: Simulación Centralizada vs simulaciones distribuidas con cálculo de GVT y toma de estado.

Costos % / # Nodos	2 Pares	3 Pares	4 Pares	5 Pares
1400	100.62	271.09	463.15	511.42
1800	122.16	338.60	570.22	669.67
2200	125.03	363.38	591.17	770.37
2600	117.11	371.99	610.84	863.26
3000	134.01	393.23	652.58	854.16

Tabla 5.16: Costos centralizado vs distribuidos sin estado global y GVT.

5.3.2 Costos incluyendo estado global y GVT para el segundo conjunto de pruebas

En esta sección, se compara la simulación distribuida contra la centralizada y se determina si conviene o no, en la simulación distribuida, incluir los algoritmos de toma de estado global y cálculo de GVT.

Costos % / # Nodos	2 Pares	3 Pares	4 Pares	5 Pares
1400	50.70	166.39	205.71	245.16
1800	79.66	205.11	273.98	330.68
2200	84.21	235.03	336.33	407.71
2600	94.55	263.63	364.56	477.01
3000	102.52	281.21	427.30	516.53

Tabla 5.17: Costo centralizado vs distribuidos incluyendo algoritmos de estado global y GVT.

De la tabla 5.17, se puede inferir en términos generales, que es conveniente usar cualquier tipo de simulación distribuida, incluyendo los algoritmos de estado global y GVT. El mejor escenario es tener a disposición 5 trabajadores. Por ejemplo, para un grafo de 3000 nodos en una simulación con 5 trabajadores es 516.53 % más rápida comparada con la simulación centralizada.

Por otro lado, si sólo se tuvieran 2 trabajadores, es también buena opción, ya que esta simulación es 102.52 % más rápida para un grafo de 3000 nodos.

En resumen, el mejor escenario es: 5 trabajadores para cualquier grafo de éste segundo conjunto de pruebas.

5.3.3 Costos de una simulación distribuida con 2 trabajadores incluyendo estado global y GVT

La figura 5.19 muestra una comparativa entre dos simulaciones de tipo distribuido con 2 trabajadores, con y sin toma de estado global y cálculo de GVT. Durante la simulación no hay falla de paro.

# Nodos	Costo (%)
1400	33.12
1800	23.64
2200	22.16
2600	11.59
3000	15.54

Tabla 5.18: Costo para 2 trabajadores con y sin estado global y GVT.

De la tabla 5.18, se observa el precio que se tiene que pagar en una simulación distribuida con 2 trabajadores, por incluir los algoritmos de toma de estado global y cálculo de GVT. Por ejemplo para un grafo de 2600 nodos se requiere de un 11.59 % más de tiempo para que finalice la simulación.

5.3.4 Costos de una simulación distribuida con 3 trabajadores incluyendo estado global y GVT

La figura 5.20 muestra una comparativa entre dos simulaciones de tipo distribuido con 3 trabajadores, con y sin toma de estado global y cálculo de GVT. Durante la simulación no hay falla de paro.

# Nodos	Costo (%)
1400	39.30
1800	43.74
2200	38.30
2600	29.79
3000	29.38

Tabla 5.19: Costo para 3 trabajadores con y sin estado global y GVT.

De la tabla 5.19, se observa el precio que se tiene que pagar en una simulación distribuida con 3 trabajadores, por incluir los algoritmos de toma de estado global y cálculo de GVT. Por ejemplo para un grafo de 2600 nodos se requiere de un 29.79 % más de tiempo para que finalice la simulación.

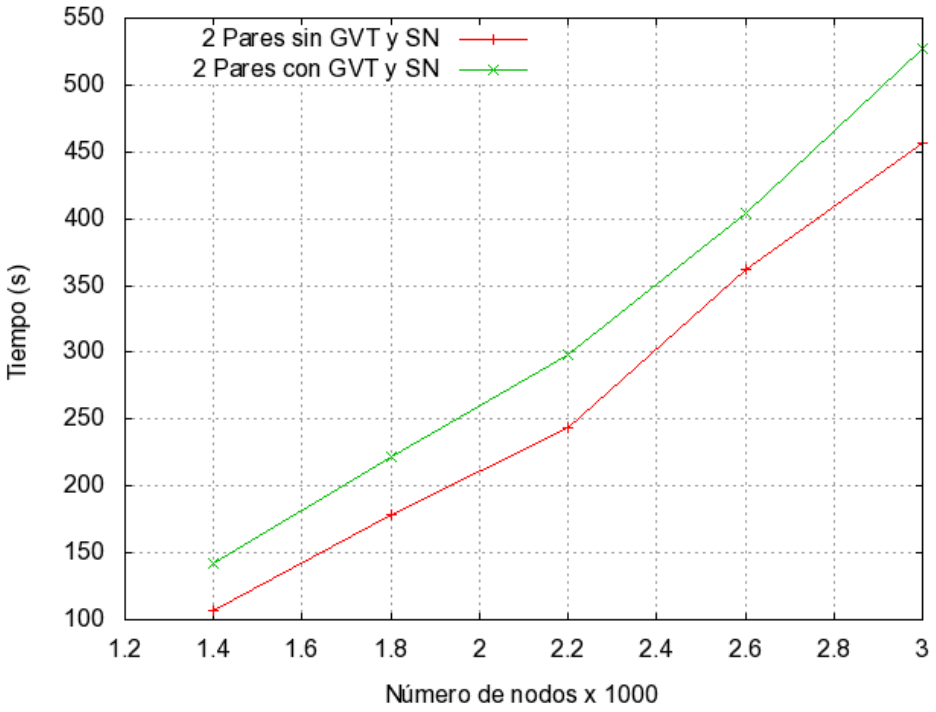


Figura 5.19: Dos trabajadores y sus costos en porcentaje de tiempo.

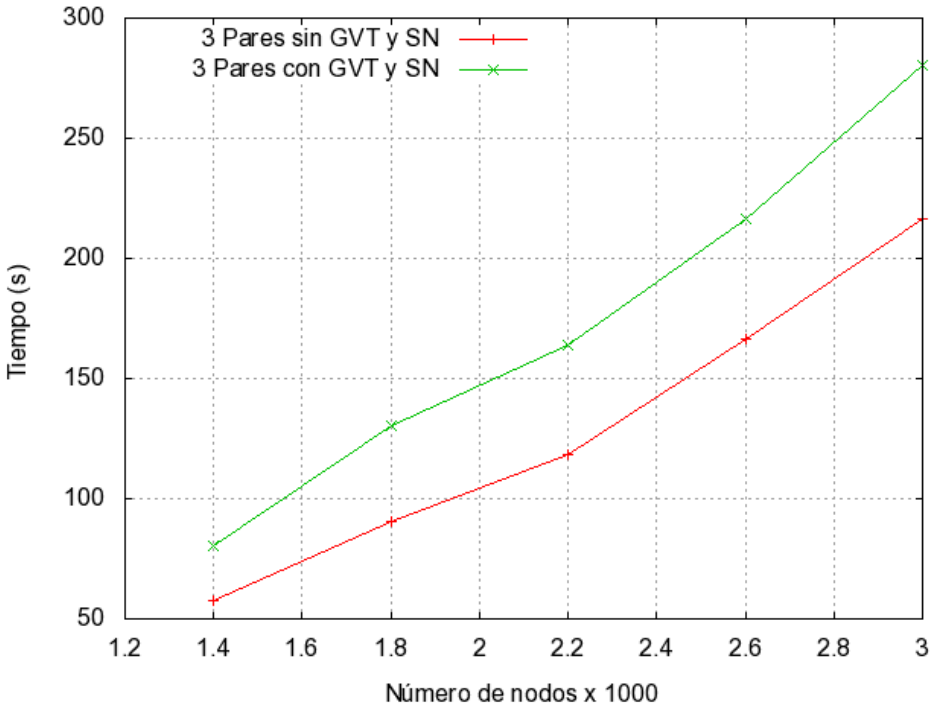


Figura 5.20: Tres trabajadores y sus costos en porcentaje de tiempo.

5.3.5 Costos de una simulación distribuida con 4 trabajadores incluyendo estado global y GVT

La figura 5.21 muestra una comparativa entre dos simulaciones de tipo distribuido con 4 trabajadores, con y sin toma de estado global y cálculo de GVT. Durante la simulación no hay falla de paro.

# Nodos	Costo (%)
1400	84.21
1800	79.21
2200	58.40
2600	53.01
3000	42.72

Tabla 5.20: Costo para 4 trabajadores con y sin estado global y GVT.

De la tabla 5.20, se observa el precio que se tiene que pagar en una simulación distribuida con 4 trabajadores, por incluir los algoritmos de toma de estado global y cálculo de GVT. Por ejemplo para un grafo de 2600 nodos se requiere de un 53.01 % más de tiempo para que finalice la simulación.

5.3.6 Comparativa de costos de una simulación distribuida con 5 trabajadores

La figura 5.22 muestra una comparativa entre dos simulaciones de tipo distribuido con 5 trabajadores, con y sin toma de estado global y cálculo de GVT. Durante la simulación no hay falla de paro.

# Nodos	Costo (%)
1400	77.14
1800	78.70
2200	71.42
2600	66.93
3000	54.76

Tabla 5.21: Costo para 5 trabajadores con y sin estado global y GVT.

De la tabla 5.21, se observa el precio que se tiene que pagar en una simulación distribuida con 5 trabajadores, por incluir los algoritmos de toma de estado global y cálculo de GVT. Por ejemplo para un grafo de 2600 nodos se requiere de un 66.93 % más de tiempo para que finalice la simulación.

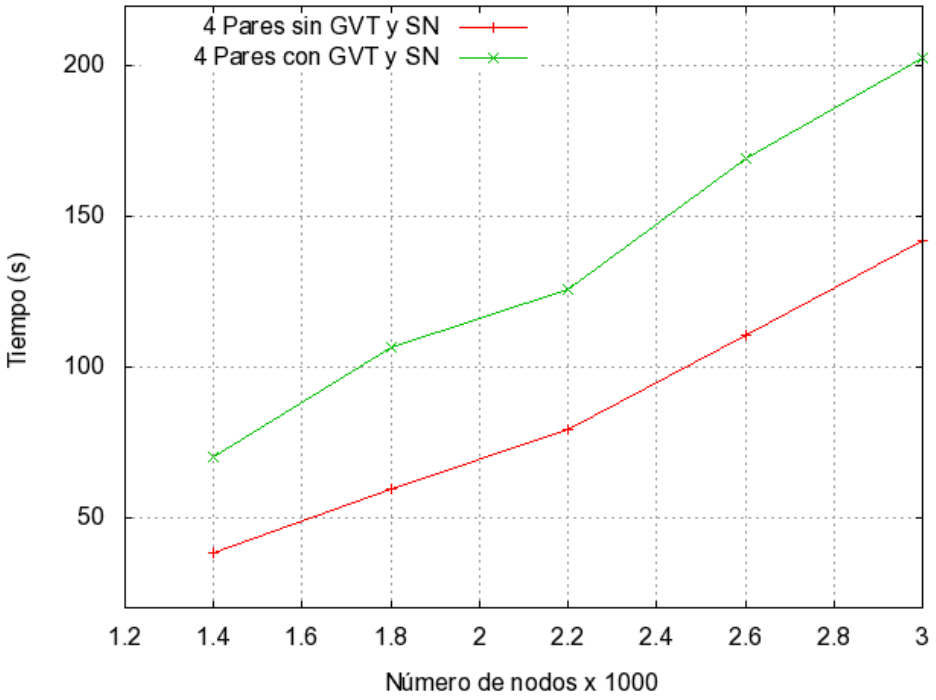


Figura 5.21: Cuatro trabajadores y sus costos en porcentaje de tiempo.

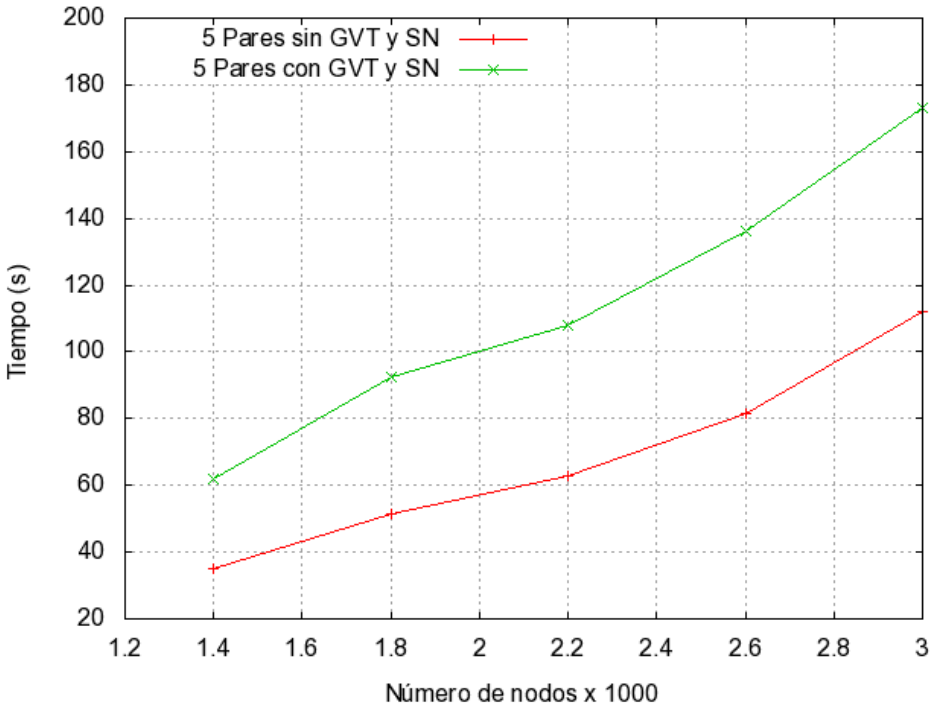


Figura 5.22: Cinco trabajadores y sus costos en porcentaje de tiempo.

Capítulo 6

Conclusiones y trabajo futuro

Se han cumplido todos los objetivos para la construcción de este prototipo de simulador, por un lado el objetivo general: implementación de un prototipo sobre un ambiente distribuido dinámico, por otro lado, los objetivos particulares: especificar las operaciones, plantear las entidades de una arquitectura que implemente dichas operaciones, construir e integrar los componentes y el simulador tendrá soporte para una falla de paro.

Se ha establecido una base para el funcionamiento de este prototipo que podrá ser utilizado para estudios de sistemas que se pueden representar mediante el enfoque de eventos discretos, y también para la enseñanza y estudio de algoritmos que resuelvan problemas correspondientes a los sistemas distribuidos.

Por otro lado, para el primer conjunto de pruebas, si no se ejecutan los algoritmos para toma de estado global y el cálculo de GVT, se puede utilizar 2 o 3 trabajadores. Eso depende de la elección del usuario. En caso que se incluya la ejecución de los algoritmos ya mencionados, es muy conveniente usar 3 trabajadores a partir de grafos desde 300 a 1000 nodos, justo como se puede apreciar en la figura 5.9.

Los resultados de las secciones 5.2.6, 5.2.7, 5.2.8 y 5.2.9, para el primer conjunto de pruebas demuestran cómo en una simulación en donde se hace presente una falla de paro, resulta muy costoso recuperarse, sin embargo, ese costo se justifica para lograr que la simulación finalice satisfactoriamente. Es preciso notar que sólo para simulaciones donde se involucran grafos con 1000 nodos es conveniente dejar que el mecanismo de restauración se lleve a cabo, para el resto de datos conviene más reiniciar la simulación.

Para el segundo conjunto de pruebas, el mejor escenario es utilizar 5 trabajadores, con cualquiera de los grafos. A pesar de que se utilicen los mecanismo de toma de estado global y cálculo de GVT, 5 trabajadores sigue siendo la mejor opción. Si se cuenta con menos trabajadores, cualquiera de las simulaciones es aceptable, con 2,3 y 4 trabajadores.

Por otro lado, el presente simulador sólo se ejecuta bajo un medio controlado como lo es una red local y requiere de la presencia de una persona para iniciar una simulación.

Como trabajo futuro se pueden mencionar los siguientes puntos:

1. Determinar la frecuencia con la que puede realizarse la toma de estado global y cálculo de GVT, de tal manera que no afecte el rendimiento de una simulación cuya duración sea de horas o incluso días.

2. Determinar el número óptimo de trabajadores para el primer y segundo conjunto de pruebas.

3. Utilizar algún conjunto de pruebas propuesto por la comunidad científica para evaluar el desempeño de este simulador.

4. Implementación del inicio automático del simulador al iniciar la computadora.

5. Implementación de una interfaz gráfica para un uso más amigable por parte del usuario.

6. Sería conveniente introducir algún algoritmo de balance de carga, que ya exista en la literatura de los sistemas distribuidos o bien proponer uno nuevo.

7. El coordinador guarda todas las tomas de estado global de la simulación y en caso que el coordinador sufriera una falla de paro, no hay nada implementado que ayude a la recuperación del coordinador, habría que proponer una arquitectura donde se tome en cuenta este tipo de falla orientada al coordinador.

8. Proponer una nueva arquitectura que permita que este prototipo pueda llevar a cabo su ejecución e interacción con redes externas. Es decir que este simulador se pueda ejecutar en varias redes independientes y lograr así una interacción y hacer experimentos de mayor escala, con casos de estudio más grandes.

9. Evaluar la posibilidad de realizar una implementación de este prototipo orientada a lo que se conoce bajo el nombre de infraestructura como servicio (IaaS por sus siglas en inglés), donde este tipo de sistemas ofrecen al usuario la habilidad para ejecutar y controlar instancias de máquinas virtuales desplegadas a través de una variedad de recursos físicos [37].

Bibliografía

[1] Ricardo Marcelín Jiménez. *Manual Simulador Eventos Discretos (CDES)* Paquete Computacional. Universidad Autónoma Metropolitana. 2005.

[2] Misra, J. Distributed discrete event simulation. *Computing Surveys*, vol 18 Issue 1, pp 39-65, Mar. 1986.

[3] Bruno R. The Parsimony Project: A Distributed Simulation Testbed in Java In *Proc. 1999 International Conference On Web-Based Modelling and Simulation*, volume 31 of Simulation Series, pages 89-94, San Francisco, CA, January 1999. Society for Computer Simulation.

[4] L.M. Campos. A General-Porpuse Discrete Event Simulator. In *Symposium on Performance Evaluation of Computer and Telecomunication Systems*, 2001.

[5] OMNET++ *User Manual*. Internet: http://www.omnetpp.org/doc/manual/usman.html#sec378, Jun. 2009.

[6] Francisco García. Biblioteca de Programación Paralela. Internet: http://flanagan.ugr.es/oep/node34.html, Dec. 14, 2000 ,Mar. 2009.

[7] Bruno R. Preiss, Ian MacIntyre. Yaddes Yet Another Distributed Discrete Event Simulator User Manual. Internet: http://www.brpreiss.com/page61.html, Oct. 14, 2008, Jun. 2009.

[8] Marcelín Jiménez, R. *Almacenamiento distribuido tolerante a fallas*. Tesis de doctorado, UNAM, 2004.

[9] Hassan, R. Anwar, Z. Yurcik, W. Brumbaugh, L. Campbell, R.. A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems. *In Proceedings of IEEE International Conference on Information Technology (ITCC)*, Apr. 2005.

[10] Wilmarth, T.L. & Kale, L.V. (2004). POSE: Getting Over Grainsize in Parallel Discrete Event Simulation. In Proceedings of the 2004 *International Conference on Parallel Processing*. Los Alamitos, CA: IEEE Computer Society.

[11] Perumalla, K.S. (2005). μsik a microkernel for parallel distributed simulation systems. *In Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*. Los Alamitos, CA: IEEE Computer Society.

[12] Park, A. & Fujimoto, R.M. (2006). Aurora: an approach to high throughput parallel simulation. *In Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*. Los Alamitos, CA: IEEE Computer Society.

[13] Chandy & Lamport, Distributed Snapshots: Determining Global States of Distributed Systems, *ACM Transactions on Computer Systems*, Feb. 1985, Num 1, Vol 3, Page(s) 63 - 75.

[14] Jefferson D.R, Virtual Time. *ACM Transactions on Programming Languages and Systems*, 404 - 425 Jul. 1985, Num 3, Vol 7.

[15] Jerome Verstrynge, *Practical JXTA cracking the puzzle*, Dawning Streams, Inc.,Lulu Enterprises, Inc., Netherlands, 2008.

[16] Alvisi Lorenzo, A Survey of Rollback-Recovery Protocols in Message-Passing Systems, *ACM Computing Surveys*, September 2002, Num 3, Vol 34, Page(s) 375-408.

[17] Sayantan Chakravorty & Celso Mendez & Laxmikant Kalé, *Proactive Fault Tolerance in Large Systems*, Departament of Computer Science, University of Illinois at Urbana-Champaing.

[18] Sayantan Chakravorty & Laxmikant Kalé, *A Fault Tolerance Protocol with Fast Fault Recovery*, Departament of Computer Science, University of Illinois at Urbana-Champaing.

[19] Nicol, D.M. and Balci, Strategic Directions in Simulation Research , *Winter Simulation Conference*, 1999 Farrington, P.A. and Nembhard, H.B. and Sturrock, D.T. and Evans, G.W. Page(s) 1509-1520.

[20] Schneider, B. F., What Good are Models and What Models are Good?, *Distributed Systems* , ACM, 1993, Mullender, S. pages 17-26, Second Edition.

[21] Beltrao-Moura, J.A., *Redes locales de computadoras*, McGraw Hill, 1990.

[22] Jain, R. *The art of computer systems performance analysis*,John Wiley and sons, 1991.

[23] Ferscha, A., *Parallel and Distributed Simulation of Discrete Event Systems*, Handbook of Parallel and Distributed Computing, McGraw Hill, 1995.

[24] Fujimoto, R.M., *Parallel and Distributed Simulation*, Winter Simulation Conference, 1999, Farrington, P.A. and Nembhard, H.B. and Sturrock, D.T. and Evans, G.W., pages 122-131.

[25] Shorey, R. and Kumar, A. and Rege, K.M., Instability and Performance Limits of Distributed Simulators of Feedforward Queueing Networks,*ACM Trans. on Modeling and Computer Simulations*, Vol. 7, Num 2, pages 210-238, 1997.

[26] Shwartz, A. and Weiss, A., *Large Deviations for Performance Analysis*,Chapman & Hall, 1995.

[27] Carothers, C.D. and Fujimoto, R.M. and England, P., Effect of Communication Overheads on Time Warp Performance, *Winter Simulation Conference*, 1994.

[28] Preiss, B.R. and Loucks, W.N. and MacIntyre, I.D.,Effects of the Checkpoint Interval on Time and Space in Time Warp, ACM Trans. on Modeling and Computer Simulation, Vol. 4, Num 3, pages 223-253, Jul. 1994.

[29] Ferscha, A., Adaptive Time Warp Simulation of Timed Petri Nets, *IEEE Trans. on Software Engineering*, Vol. 25, Num 2, pages 237-257, Mar. 1999.

[30] A. Segall. Distributed network protocol, *IEEE Trans. on Information Theory*, 29(1):23-35, Jan. 1983.

[31] H. Kopetz and P. Veríssimo. Real tiem and dependability concepts. In S. Mullender, editor, *Distributed Systems* , chapter 16, pages 411-446. ACM, 2 edition, 1993.

[32] T. Park adn H.Y. Yeom. Application controlled checkpointing coordination for fault-tolerant distribuited computing systems. *Parallel Computing*, 26:467-482,2000.

[33] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558-564, 1978.

[34] J.M. Helary, R.H.B. Netzer, and M. Raynal. Consistency issues in distributed checkpoints. *IEEE Trans. on Software Engineering*, 25(2):274-281, Mar. 1999.

[35] Samir, R.D. and Fujimoto, R., Adaptive Memory Management and Optimism Control in Time Warp, *ACM Trans. on Modeling and Computer Simulation*, Vol 7, Num 2, pages 239-271, Apr, 1997.

[36] Srinivasan, S. and Reynolds, P.F., Elastic Time, *ACM Trans. on Modeling and Computer Simulation*, Vol 8, Num 2, pages 103-139, Apr, 1998.

[37] Nurmi, D., *The Eucalyptus Open-source Cloud-Computing System*. Computer Science Department, University of California, Santa Barbara, California 93106.

[38] Coulouris., *Distributed Systems*. Computer Science Department, University of California, Santa Barbara, California 93106.

[39] D. Thiébaut, *Parallel Programming in C for the Transputer*. Internet: http://maven.smith.edu/∼thiebaut/transputer/chapter8/chap8-1.html, 1995.

Index

f-tolerante, 18

algoritmo
    método conservador, 11
    método optimista, 12
    PIF, 61
    toma de estado global, 15
algoritmo PIF
    propiedades, 63
arquitectura
    coordinador, 34
    general, 32
    trabajador, 34

calculando
    costo, 67
    factor de aceleración, 67
computación confiable, 17
confiabilidad, 16
conjunto de pruebas
    primero, 66
    segundo, 84
conjunto finito de estados, 1

disponibilidad, 16
Distributed Discrete Event Simulation, 2

Eliminación, 18
errores
    de causalidad, 2
    de integridad, 3
estado global, 3, 12
estampilla(s) de tiempo, 2
eventos
    lista de, 2

falla(s)
    accidentales, 17
    bizantinas, 17
    de caída, 17
    de caída y enlace, 17
    de concepción, 17
    de omisión de recepción, 17
    de omisión de transmisión, 17
    de omisión general, 17
    de paro, 17
    externas, 17
    físicas, 17
    humanas, 17
    intencionales, 17
    internas, 17
    operacionales, 17
    permanentes, 17
    temporales, 17
    tolerante a, 17

historia
    global, 13
    local, 13

operación
    calcular GVT, 36
    clasificar y asignar tareas, 36
    ejecución método optimista, 54
    falla de paro, 36
    toma de estado global, 36
operaciones
    coordinador, 36
    trabajador, 54

predicado global, 12

Prevención, 18
Previsión, 18
proyecto
    μsik, 25
    Aurora, 25
    CDES, 26
    GPDES, 23
    OMNeT++, 24
    Parsimony, 23
    POSE, 24
    YADDES, 22

redundancia
    de información, 19
    de recursos físicos, 19
    de tiempo, 19
    en información, 3
    en recursos físicos, 3
    en tiempo, 3
reloj, 2

seguridad, 16
simulación
    de alto rendimiento, 2
    dirigida por eventos, 1
    en serie, 2
sistema distribuido, 12
sistemas P2P
    características, 28

tiempo simulado, 2
Tolerancia, 18