VNIVERSIDAD NACIONAL AVPNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ALMACENAMIENTO DISTRIBUIDO TOLERANTE A FALLAS

T E S I S

QUE, PARA OBTENER EL GRADO DE

DOCTOR EN CIENCIAS,

P R E S E N T A

RICARDO MARCELÍN JIMÉNEZ

DIRECTOR DE LA TESIS: DR. SERGIO RAJSBAUM GORODEZKY

MÉXICO D.F., MAYO DEL 2004.

Con amor para Calú, Fer y Chrys.

Contenido

Lista de Figuras

Lista de Tablas

Presentación

En el año 490 a.C., Maratón era un pequeño emplazamiento semidesértico, en donde el ejército persa, al mando de su general Datis desembarcó a un promedio de 40.000 soldados con la consigna de invadir la ciudad de Atenas, capital de los pueblos griegos. Contra todo pronóstico, los escasos 12,000 griegos dirigidos por el general Milcíades derrotaron a los persas. Al ver esto, Datis decidió embarcarse y llegar a Atenas por otro punto, de tal forma que fuese sorprendida y creyendo que la batalla de Maratón estaba perdida, los habitantes entregaran la ciudad sin oponer fuerza. Milcíades se dió cuenta de la estrategia y decidió enviar a Fidípides, su corredor más veloz, para que avisara que no se rindieran y esperaran la llegada de los refuerzos. Llegó a la caída de la tarde gritando con sus últimas fuerzas: ¡Nike!¡Nike! (¡victoria! ¡victoria!) y se desplomó. Fidípides había recorrido en pocas horas más de cuarenta kilómetros: su hazaña es recordada cada cuatro años en las Olimpiadas modernas cuando atletas de todas las partes del mundo compiten sobre la distancia recorrida por él para ganarse el reconocimiento más codiciado y prestigioso, el de los corredores de maratón.

Para mí es motivo de gran satisfacción escribir esta parte de mi tesis que, aunque abre el trabajo y sirve de bienvenida, en realidad fue escrita después de que tuve completo el primer borrador y durante la etapa en que hice las últimas correcciones. Llego a este punto con la emoción de saberme en los metros finales de un maratón que comencé a correr cuando mi hija mayor estaba por nacer y que decidí terminar por vocación, por amor propio y por el compromiso con quienes siempre creyeron en mí, que son los únicos espectadores que quedan en el estadio. Al igual que aquel famoso corredor griego, yo he puesto la vida en este proyecto profesional que termina con este documento. A diferencia de él, creo que me tomó un "poco" más de tiempo y he guardado fuerzas para seguir corriendo.

A lo largo de estos años he tenido el placer de conocer a mucha gente que me ha enriquecido con sus conocimientos, consejos y puntos de vista. Con ellos tengo una deuda de gratitud.

En primer lugar debo reconocer a mi asesor, Sergio Rajsbaum, quien me inició en el mundo de los algoritmos distribuidos, también me sugirió los temas que indujeron mi trabajo de investigación, tuvo la sensibilidad para aceptar mi búsqueda personal y siempre ha sabido ofrecerme los puntos de vista y observaciones de un científico en plenitud.

A Mogens Bladt le debo haberme puesto en contacto con las heurísticas para resolver problemas combinatorios. Pero sobre todo, le debo la gentileza de su trato y todo el tiempo que me ofreció para escuchar con atención mis dudas.

Hanna Oktaba es una apasionada de su quehacer como formadora de muchas generaciones de graduados en computación, que le reconocemos su esfuerzo y cariño. Por ello mismo

agradezco todo el interés con que siguió cada paso de mi trabajo, el formato de esta tesis y el contenido del capítulo 5 deben mucho a las sugerencias de Hanna.

María Garza jugó varios roles durante mi estancia en el posgrado. Primero fue la coordinadora que me recibió en el programa, luego fue mi jurado de predoctoral (al igual que Sergio, Mogens, Hanna y Alberto) y, por último, fue sinodal en mi examen de grado. Siempre recibí de ella la más fina atención y el apoyo para superar cualquier problema administrativo.

Alberto Contreras me ayudó a poner al día muchos conceptos de series de tiempo y filtrado que enriquecieron mi caja de herramientas. Tenemos pendiente un trabajo sobre predicción de retardos en Internet. Sin embargo, el mérito que más le reconozco es que, siendo un joven investigador, tuvo la sensibilidad para empatizar con mis problemáticas de estudiante de doctorado y sugerirme estrategias para superarlas.

Jorge Urrutia es un investigador de prestigio internacional que confió en la calidad de mi trabajo. En los momentos que pudimos conversar, me hizo comentarios que mejoraron los resultados finales de mi tesis o que darán pie a nuevas direcciones de investigación.

Fabián García me sugirió cambios importantes en el orden temático de los capítulos iniciales, que sirvieron para darle su "acabado final".

Antonio Ramírez revisó este trabajo con el mayor detalle y aportó observaciones muy valiosas que mejoraron el capítulo 2. Siendo un experto en modelos de sistemas discretos pudo ofrecerme una visión muy crítica acerca de conceptos tales como la simulación de eventos discretos, la tolerancia a fallas y los estados globales.

Cristina Verde, como coordinadora del posgrado, me ayudó a financiar el viaje para asistir al congreso LADC'03.

Fausto Casco y Miguel Peña son dos queridos amigos de la UAM que, desde sus cargos administrativos, me apoyaron para asistir al congreso SIROCCO'03. Sin embargo, su ayuda generosa excedió con mucho las responsabilidades de sus puestos, siempre me brindaron su confianza y facilitaron mis tareas para que pudiera avocarme a mi trabajo de doctorado.

Hugo Rincón, Noé Gutiérrez y Adolfo Torres me iniciaron en la teoría de grupos y los campos finitos. Con los dos últimos compartó interés en los diseños combinatorios y fueron ellos quienes me pusieron en contacto con las geometrías finitas y los algoritmos de orden revolvente.

Lulú González, secretaria del posgrado, así como el resto del personal administrativo, han sido grandes amigas con cuya ayuda y experiencia pude sortear los aspectos burocráticos de mi estancia en el programa de doctorado. Les agradezco el trabajo que siempre han realizado con gran dedicación.

Mi amigo José Téllez fue muy amable en revisar los primeros capítulos de la tesis y hacerme algunas recomendaciones acerca del subjuntivo del español.

Cristina, mi compañera, me relevó de muchas tares domésticas aún durante las vacaciones y fines de semana de varios años, para poder dedicarme a mi trabajo. Por la misma razón, a mis hijas Clara Luz y Fernanda, les debo no sé cuantas salidas a la playa, balnearios, museos, pesca y un largo etcétera. Ellas son las atenienses por las que libro mis batallas y a quienes dedico mis maratones.

Agradezco todas las observaciones y correcciones que ayudaron a mejorar este documento, tanto en la forma, como en el fondo. No obstante, si aún con la ayuda y buena voluntad de

todos, algún error hubiera salido ileso de la cacería, entonces yo soy el único responsable.

Por último, quisiera comentar que disfrute mucho de la redacción de este trabajo, en buena medida gracias a que Sergio me ofreció sus comentarios y consejos para que, aún respetando la forma de una tesis doctoral, entendiera que esta debía reflejar mucho de lo que soy. Por tanto, decidí relajarme y contar las cosas a mi modo. Me parece que esto se puede observar especialmente en el capítulo 1, que tiene una forma autorreferencial y donde se hacen algunas disgresiones para contar otras historias al margen del hilo principal. En muchos de mis libros favoritos esto ocurre y no pude resistir a la tentación de emularlos. Por otro lado, cada capítulo inicia con una cita literaria. Son extractos de las lecturas que me acompañaron en todos estos años de trabajo y me pareció que decantaban el sabor de cada unidad temática.

Escribir es como lanzar una botella al mar con un mensaje adentro. Ojalá que ésta encuentre un receptor, en cuyo caso espero que le resulte interesante el mensaje y, si cabe el buen deseo, que la disfrute como yo.

R.M.J.

Capítulo 1

Introducción

En esta tesis se desarrollan herramientas que pueden emplearse en la construcción de simuladores distribuidos y dotarlos de la capacidad para tolerar fallas de paro. La contribución original puede dividirse en tres partes: en la primera se presenta un procedimiento de contingencia que, en caso de falla, permite restaurar una simulación distribuida a partir de un punto de su pasado reciente desde el cual se reinicia a cargo de los componentes que quedan en servicio. Esta medida preventiva consiste en registrar la "historia de la simulación" por medio de técnicas de almacenamiento distribuido. El reto es conseguir soluciones que logren un balance de carga (en tiempo y espacio) y toleren al mayor número de fallas. En la segunda parte se revisa la manera en que estos métodos pueden escalarse, es decir, aplicarse al caso en que la extensión del sistema y el costo de la comunicación representan importantes factores a considerar. En la última parte de este trabajo se presenta una plataforma de simulación para realizar experimentos relativos a los algoritmos desarrollados durante la investigación. Alternativamente, las técnicas que aquí se proponen pueden aplicarse al almacenamiento de trazas de una ejecución distribuida o bien, servir en la construcción de sistemas de almacenamiento distribuido de datos como los que se usan en Internet para el hospedaje de contenidos.

> Los Guardianes conversaban entre ellos. Me acerqué. Me esperaban; uno giró hacia mí y me sonrió, mientras los demás retornaban a su cuidadosa vigilancia, en apariencia indiferentes. -¿Qué es lo que vió? Decidí ser franco y se lo conté, con las limitaciones naturales del lenguaje . -¿Qué piensa de ello? - No lo sé, con exactitud, pero me hice una teoría: el Tiempo está cerrado en esta región.
>
> *R.C. De Marco en "Todos los Caminos del Universo"*

1.1 Contexto

Imaginemos una red de telecomunicaciones para la que tenemos la responsabilidad de estudiar sus operaciones. Esta red debe soportar desde los servicios convencionales de voz, hasta los nuevos servicios de Internet y los futuros servicios de video sobre demanda. Además, este sistema debe admitir usuarios fijos y móviles. Para optimizar los costos y garantizar la calidad de las operaciones, deberíamos realizar un estudio cuantitativo del que pudieran responderse preguntas como: ¿cuáles son las condiciones bajo las que se pueden presentar los cuellos de botella del sistema? ¿cómo planear su crecimiento? es decir, ¿dónde deben construirse las

centrales? ¿con qué capacidades? Igualmente, sería interesante poder prever situaciones de desastre y determinar los planes de contingencia más rápidos y eficaces.

Si bastara con resolver sus problemas presentes, entonces quizá sería suficiente con un monitoreo bien planeado con el que se detectaran las situaciones de conflicto en el sistema. Sin embargo, si la red existiera únicamente en planos, si se quisieran analizar situaciones hipotéticas, si se quisiera "ver" hacia el futuro o, incluso, si la acción de monitoreo interfiriera con la misma calidad de los servicios, entonces tendríamos que buscar un camino alternativo: un modelo abstracto que representara al sistema bajo estudio y sobre el que pudiéramos efectuar acciones cuyas consecuencias nos permitieran inferir las respuestas que nos interesan.

Construir un modelo como sustituto de un sistema real significa usar un lenguaje formal mediante el cual describimos las partes del sistema que parecen relevantes para nuestro estudio y establecemos las relaciones entre estos componentes. Las construcciones de un lenguaje formal pueden manipularse mediante un conjunto de herramientas (teóricas o concretas) para producir inferencias sobre las propiedades planteadas por el modelo. Por ejemplo, podríamos utilizar la teoría de filas de espera como lenguaje y representar al sistema como una red de servidores a los que llegan las solicitudes de los clientes con diferentes velocidades y estadísticas de atención. Usando esta teoría, en muchos casos es posible construir un modelo planteando una o varias ecuaciones diferenciales cuya solución (si existe) "revelaría", por ejemplo, los tiempos medios de servicio, el número de servidores necesario para garantizar una probabilidad de atención, etc. Eventualmente, estos parámetros inferidos deberían interpretarse para responder las preguntas que dieron inicio al estudio.

Por otro lado, cada método formal (es decir el lenguaje y sus herramientas) tienen un contexto de aplicación limitado y existen situaciones en las que su utilidad resulta rebasada. En el caso del método analítico que hemos planteado, hay que aceptar que sólo tiene aplicación cuando el sistema es muy simple y uniforme. En otras palabras, sólo es aplicable si se supone que las solicitudes de los clientes obedecen a unos cuantos patrones de llegada y estadísticas de servicio para los que existen modelos con solución. Es preciso mencionar que en la teoría de filas de espera, también existen métodos analíticos de tipo algorítmico que pueden ser una alternativa para resolver modelos. El problema con la red de telecomunicaciones bajo estudio es que debe estar preparada para nuevos tipos de servicios de los que apenas se conocen descripciones analíticas. En el caso de Internet, por ejemplo, no existe (usando esta teoría) una forma fidedigna de representar las ráfagas de datos que ocurren en algunas de sus aplicaciones. En este punto hay que mirar hacia otro lado en busca de soluciones alternativas.

En la simulación de eventos discretos, o DES (Discrete Events Simulation), el lenguaje de descripción es un lenguaje de computadora y la herramienta es la computadora misma. El modelo del sistema se describe mediante un programa que luego se compila y ejecuta para generar una "historia" del sistema que representa. El programa se construye bajo el supuesto de respetar las relaciones causa-efecto que se dan en el sistema real y esto le otorga a la "historia" artificial una validez y un sentido predictivo. Regresando a nuestro ejemplo, podemos programar eventos que representen a las solicitudes de servicio que llegan a la red, igualmente programaríamos las rutinas de atención (para voz, Internet y video sobre demanda). La computadora nos da también la facilidad de representar usuarios móviles que compiten por los recursos de la red con los usuarios fijos. Usando este método es posible construir un

modelo con cualquier nivel de detalle, sin embargo habría que considerar el costo en tiempo de máquina que pueda resultar de una simulación de "grano fino", como suele denominarse en la terminología especializada. Esto quiere decir que, para aceptar el carácter predictivo de la historia artificial, esta debería abarcar un tiempo suficientemente largo como para adquirir una validez estadística que acote la probabilidad de error en la predicción. Si nuestra red fuera del tamaño y/o la complejidad del sistema de telecomunicaciones de la Cd. de México, por ejemplo, requeriríamos varios meses de simulación continua con una sola computadora de capacidades medias, para poder confiar en el valor predictivo de nuestros resultados. ¿En qué es mejor un método analítico comparado con la DES? El primer método tiene la ventaja de producir resultados en término de parámetros de desempeño y por lo tanto sus soluciones son generales. Su inconveniente radica en el número limitado de sistemas reales cuyos modelos tienen solución. En contraste, la DES siempre produce una solución pero, cualquier cambio en los parámetros del sistema real requiere un nuevo programa que lo refleje (no es generalizable). Este método resulta más indicado cuando se necesita describir un sistema muy complejo y/o con mucho detalle. Sin embargo su mayor inconveniente es el tiempo de ejecución.

En una simulación distribuida de eventos discretos o DDES (Distributed Discrete Events Simulation), se tiene un conjunto de computadoras interconectadas mediante canales de alta velocidad. En esta ocasión, el modelo del sistema real se disgrega en subsistemas o subconjuntos disjuntos de componentes y cada uno de estos se convierte en un programa ejecutado por una de las computadoras disponibles. Las interacciones entre dos componentes que se simulan dentro de una misma máquina se resuelven como en el caso de la DES. Por otro lado, las interacciones entre dos componentes localizados en máquinas distintas se resuelven mediante mensajes transmitidos sobre los canales que las conectan. Usando DDES se puede estudiar un sistema como la red de telecomunicaciones de la Cd. de México, fragmentando su modelo en regiones disjuntas que luego se simulan sobre diferentes máquinas (una máquina por cada región).

El mayor problema de la DDES es la sincronización entre los componentes simulados. Pensemos, por ejemplo, en tres computadoras a cargo de tres regiones distintas del modelo bajo estudio. Desde una computadora A se envía a B un mensaje que, de acuerdo con el tiempo simulado o estampilla de tiempo, debe producir un efecto durante el instante t_0. Segundos después, desde la computadora C se envía también hacia B un mensaje con una estampilla de tiempo t_1, dirigido al mismo componente del evento anterior. ¿En qué orden se deben despachar los eventos recibidos en B? Obsérvese que el orden en que se reciben no es necesariamente el mismo en que deben simularse sus efectos correspondientes. Si se atendieran en el orden equivocado entonces podría estarse violando la relación de causalidad en la simulación y sus resultados perderían su valor predictivo. Supongamos que el mensaje con estampilla t_1 debe despacharse primero pero, ¿cómo sabe B que debe esperar su llegada? En realidad no lo sabe y puede proceder de dos formas: en el método optimista, supone que todos los mensajes que reciba tendrán una estampilla mayor que t_0. Si esto no ocurriera, entonces se restauraría la ejecución hasta un estado en el que todos los mensajes recibidos se hubieran despachado en orden. El segundo camino, denominado método conservador, consiste en esperar la llegada de un mensaje desde cada uno de sus canales de entrada para elegir al de menor estampilla y garantizar la atención ordenada de todos los mensajes que recibe.

1.2 Descripción de la contribución

Podríamos decir que ya contamos con una herramienta adecuada para estudiar el desempeño de los grandes sistemas de telecomunicaciones. En principio, si el sistema tiene un número masivo de componentes, basta con acomodar el modelo sobre un conjunto mayor de computadoras. Sin embargo, la experiencia nos dicta que mientras más máquinas participen en una ejecución, mayor es la probabilidad de que se presente una falla. Las fallas pueden tener muchos orígenes y manifestaciones. A nosotros nos interesan aquellas en donde la computadora se colapsa y cesan todas sus operaciones, a cuya consecuencia llamamos falla de paro. Adicionalmente, también nos interesan aquellas situaciones en las que la máquina ve degradada sus capacidades o velocidad de procesamiento, hasta el punto en que frena la simulación y resulta conveniente relevarla de su tarea.

En una ejecución distribuida, se denomina estado global, o instantánea (snapshot), al conjunto de variables que caracterizan a los procesos participantes, así como a los canales que los comunican, los cuales se registran en un punto del tiempo. Esto es semejante a suponer que un sistema distribuido se congela por un instante y se toma nota de todas las acciones que estaban efectuándose. Si tomáramos una instantánea de la ejecución distribuida antes de una falla de paro, sería como tomar una foto donde todos los participantes siguen activos. En caso de falla podríamos reponer la ejecución hasta este punto de su pasado y repartir las tareas entre los "sobrevivientes". De esta forma, ninguna tarea queda descuidada y la ejecución puede continuar, minimizándose el efecto del problema. Obsérvese que otro camino "posible" sería reponer la ejecución desde el inicio, pero si otro equipo fallara en el futuro, la ejecución podría quedarse estancada si se sigue aplicando este remedio de "fuerza bruta". Naturalmente, no basta con tomar una instantánea; regularmente debe dispararse esta acción preventiva. De otra forma, en caso de falla, la ejecución podría reponerse hasta un punto de su pasado remoto, que sería casi equivalente a la solución más burda.

En este punto nos encontramos con un nuevo problema de administración de recursos: ¿en dónde se almacenan los estados globales sucesivamente registrados? Por comodidad y consistencia, en adelante llamaremos bodegas a los sitios o depósitos donde se almacena esta información. Regresando a la pregunta, debemos observar que el archivo correspondiente puede ser muy "voluminoso", especialmente si contiene información acerca de un número grande de procesos y canales, como es el caso de una simulación distribuida de tamaño masivo. El almacenamiento de este archivo en una sola bodega tendría dos inconvenientes: puede saturarse muy rápidamente pero, sobretodo, el procedimiento de contingencia quedaría cancelado si la misma cayera en paro. Para resolver el riesgo de saturación podríamos pensar en designar un conjunto fijo de bodegas, fragmentar el archivo en tantos pedazos como bodegas se tengan y almacenar cada fragmento resultante en una bodega diferente. Sin embargo, bastaría con que una sola fallara para cancelar nuevamente el procedimiento de contingencia.

> Esta es la historia de v personas que poseen en propiedad mancomunada una riquísima mina de oro. Regularmente, el mineral debe colectarse y llevarse a la ciudad para su *almacenamiento*. Sin embargo, ésta es una tarea pesada y peligrosa de la que nadie se quiere encargar. Comisionar a un solo minero con todo el oro sería tanto como darlo por perdido (muchas cosas pueden pasarle en el

camino). Ir todos custodiando el cargamento sería abandonar la mina. Se trata de encontrar una solución que satisfaga múltiples restricciones: formaremos b comités con $k < v$ mineros cada uno y los programaremos de acuerdo con un calendario cíclico, i.e. en cada ocasión se designará a un comité diferente para el transporte y almacenamiento del oro. Cuando todos los comités hayan pasado, se volverán a comisionar en el mismo orden que la primera vuelta. Al llegar a la ciudad, el comité correspondiente deberá contratar una caja fuerte para almacenar su carga, anotar en un papel la combinación de la caja, romper el papel en k pedazos y dar un pedazo a cada minero del comité.

Muchos preguntas quedan por responderse: ¿cuál debe ser la composición de los comités? ¿cómo lograr que ningún minero trabaje en todos los comités? ¿cómo asegurarse que cada minero trabaje en tantos comités como cualquiera de sus compañeros? ¿cuál es el mínimo número de mineros cuya ausencia dejaría incompletos a todos los comités? lo que significa que ninguna caja podría abrirse ¿Cómo evitar este último problema sin darle demasiado poder a unos cuantos?

En la primera parte de este trabajo de investigación (capítulo 3) abordamos el problema de coordinar v sitios que participan en una ejecución distribuida y que también pueden fungir como bodegas. Para conseguir un equilibrio entre estas dos tareas se les organiza en b comités o bloques, que se calendarizan de manera cíclica. Cada bloque se encarga de guardar, en forma fragmentada, un estado global de la ejecución. La colección de bloques y su calendario definen un *esquema de almacenamiento*. Desarrollaremos métodos para su construcción y estableceremos medidas para evaluar cada solución en términos de requerimientos de almacenamiento, balance de carga y tolerancia a fallas.

Por otra parte, como podría esperarse, los costos de almacenamiento crecen con el número de sitios que participan en una ejecución distribuida, se trata de un problema de escalabilidad. Para abatir costos proponemos dividir al total de sitios en regiones de igual tamaño e instalar una réplica del mismo esquema de almacenamiento sobre cada una de ellas. La idea es que las bodegas de un esquema sólo guarden los fragmentos de su región y con ello se reduzca la capacidad requerida en cada bodega y se agilice el transporte de los fragmentos hasta su sitio de depósito. Sin embargo, la "regionalización" del sistema distribuido plantea un problema de optimización para el que no se conocen métodos de solución exacta.

Visto como un ser vivo, un hormiguero suele enfrentarse al problema de abastecerse de alimento buscando gastar en ello la mínima energía. Cada una de las hormigas que lo componen intenta recorrer por cuenta propia todos las fuentes de alimento y volver al nido. A su paso, una hormiga descarga en el suelo una sustancia química denominada feromona, con la que marca su rastro y establece un mecanismo de comunicación con el resto de sus hermanas. Por otro lado, la feromona se dispersa al cabo de un tiempo, por lo que su efecto tiene una duración limitada. Las hormigas tienden a seguir los rastros con feromona que han dejado sus predecesoras. Sin embargo, una ruta larga tiende a perder su carga de feromona más rápidamente que una ruta corta porque toma más tiempo construirla y recorrerla. En consecuencia, las rutas cortas prevalecen sobre las rutas largas.

> Al cabo de un tiempo, todas las hormigas avanzan en fila india sobre una misma ruta que resulta ser la más eficiente.

En los últimos años, la computación ha incorporado modelos inspirados en sistemas biológicos, para resolver problemas de tipo NP-completo. De manera muy elemental, esto significa que son problemas para los que no se conocen técnicas deterministas de solución, salvo para los casos en que el tamaño del problema sea muy pequeño. La mayoría de los métodos alternativos con que se les ataca, o técnicas heurísticas, parte de una representación del espacio de soluciones construida como una estructura de datos sobre la que se ensayan las soluciones hasta alcanzar un criterio de paro basado en calidad y tiempo de ejecución. Por ejemplo, el caso del hormiguero en busca de alimento se relaciona con un problema de optimización combinatoria conocido como "el agente viajero": Se tiene una gráfica formada por nodos y aristas con pesos, para la que se quiere encontrar un camino de peso mínimo que, partiendo de un nodo de origen, recorra todos los nodos sin repetir uno sólo, salvo el último que debe ser el mismo origen. Cuando el número de nodos a considerar excede la centena, no se conoce un algoritmo que resuelva exactamente este problema en tiempos razonables. Por otro lado, usando un enfoque heurístico denominado "sistema de hormigas", se puede construir una estructura de datos que represente a la gráfica del problema. Esta se almacena dentro de una computadora y luego el algoritmo ensaya aleatoriamente varios recorridos individuales a los que denomina "hormigas". Regularmente, se reúnen y evalúan estas soluciones, se escoge a las mejores y se vuelve a lanzar un nuevo ciclo de exploración, buscando "favorecer" la formación de caminos que tiendan a parecerse a los mejores hasta entonces construidos.

La regionalización del sistema distribuido también puede beneficiarse de un enfoque heurístico. En principio, hay que observar que se trata de un problema NP-completo, formalmente denominado "partición" o problema SP, que surge cuando crece la escala del sistema. Sin embargo, si usáramos un enfoque centralizado como en el ejemplo anterior, tendríamos que guardar en una sola máquina la representación de todo el sistema, resolver el problema sobre esa estructura de datos y luego, comunicar a cada máquina cuál es la región a la que quedaría asociada. A medida que crece el tamaño del problema, ésta puede ser una alternativa muy costosa pero, sobretodo, un desperdicio de las capacidades de cómputo distribuido del propio conjunto de recursos que se quiere administrar. En la segunda parte de este trabajo (capítulo 4), nuestra contribución construye un puente entre las heurísticas centralizadas y los algoritmos distribuidos, de forma que el mismo sistema utilice todos sus recursos para encontrar la solución que le conviene. Esta vez, cada "hormiga" realiza un recorrido aleatorio y exhaustivo sobre el sistema. Cada recorrido representa una solución del problema. Como en el caso del agente viajero, las hormigas se reúnen regularmente y se evalúa su resultado. Luego, se lanzan nuevos ciclos de exploración buscando "modular" los siguientes recorridos para que converjan hacia la solución deseada o la que se le aproxime dentro de un plazo razonable.

En computación distribuida existen otros problemas relacionados con la administración de recursos, que pueden abordarse con el enfoque propuesto en esta parte de la tesis. Por ejemplo, el balance y la (re)distribución de tareas. El primero se refiere a la asignación de tareas de cómputo de acuerdo con las capacidades disponibles de cada recurso participante. En tanto, el segundo se refiere a la repartición de las tareas de manera tal que se minimice la interacción

entre máquinas y por lo tanto se acelere su terminación (como en la DDES). Creemos que el enfoque que hemos propuesto puede ofrecer nuevos caminos de solución. Sin embargo, el trabajo con heurísticas distribuidas es de naturaleza experimental, porque los métodos pueden afinarse ajustando parámetros. Además, en el caso de muchos algoritmos distribuidos, sólo se conocen cotas superiores que evalúan su desempeño de manera aproximada sobre la base de suposiciones o simplificaciones teóricas que no siempre se cumplen.

> Imaginemos un sistema distribuido para el que tenemos la responsabilidad de estudiar sus operaciones. Igualmente importante sería prever situaciones de desastre y determinar los planes de contingencia más rápidos y eficaces. Sin embargo, si se quisieran analizar situaciones hipotéticas, si se quisiera "ver" hacia el futuro o, incluso, si la acción de experimentación interfiriera con la misma calidad de sus servicios, entonces tendríamos que buscar un camino alternativo...

En la última parte de este trabajo (capítulo 5), hemos construido un simulador de eventos discretos para evaluar algunos aspectos experimentales de la sección anterior. Consideramos la alternativa de emplear alguna utilería que pudiera adaptarse para nuestros objetivos, pero preferimos desarrollar una herramienta adhoc. Se trata de un conjunto de bibliotecas que otorgan al desarrollador la flexibilidad para experimentar con diferentes condiciones que intervienen en la operación de un sistema distribuido: comunicaciones, velocidades de procesamiento, eventos aleatorios, concurrencia, cooperación y fallas.

Curiosamente, se cierra un ciclo que comienza con el estudio de herramientas para evaluar desempeño y termina, en otro estado del conocimiento, planteando la aplicación de este mismo tipo de herramientas para la construcción de sistemas más avanzados. Creemos haber conseguido un balance entre los aspectos teóricos y los aspectos aplicados de nuestra investigación. Usando para ello un conjunto de disciplinas que van desde el álgebra abstracta hasta la simulación, pasando por la optimización combinatoria. Como fruto de esta investigación, cada uno de los capítulos mencionados, corresponden con un trabajo que fue publicado en un foro especializado de nivel internacional (el primero en una revista indexada [1] y los dos últimos en congresos [2, 3]).

Entre el momento en que se inició este trabajo y el momento de su defensa, han surgido varios proyectos industriales que apuntan en la dirección del almacenamiento distribuido como un tema importante de los próximos años. Esta coincidencia debe interpretarse como una confirmación del estado del conocimiento y de las condiciones que pueden materializar la tecnología. Sin embargo, hasta donde sabemos, la originalidad del trabajo se sostiene y radica en la aplicación de estos conceptos como estrategia para resolver contingencias en una ejecución distribuida.

1.3 Objetivos

Una vez que hemos descrito los problemas que se abordan en esta tesis, podemos precisar nuestras metas.

Objetivo general

Queremos desarrollar una colección de procedimientos de almacenamiento distribuido de la información, que puedan incorporarse en la construcción de sistemas distribuidos y, con ello, tengan la capacidad para tolerar fallas de paro.

Objetivos particulares

Adicionalmente, las soluciones que encontremos deben cumplir con las siguientes características que servirán para evaluar su calidad:

- Queremos que el procedimiento de contingencia en caso de paro esté basado en la alta disponibilidad de la información, garantizada por su almacenamiento distribuido.
- Tratándose de operaciones distribuidas, buscamos también un balance entre los recursos que cada componente individual deba comprometer y los beneficios colectivos que puedan conseguirse.
- Igualmente, buscamos un equilibrio entre el costo de las operaciones almacenamiento y el número de fallas de paro que puedan tolerarse.
- Por último, es importante que las soluciones puedan aplicarse sobre sistemas de mediana o gran escala manteniendo acotados los costos de su implementación.

1.4 El estado del conocimiento

En cada uno de los siguientes capítulos se revisa con detalle el "estado del arte" que enmarca cada unidad temática. Sin embargo, esta sección es una panorámica de los aspectos que guardan relación con nuestra investigación.

Sobre el almacenamiento distribuido, como ya mencionamos, en el último par de años han surgido propuestas que consideran esta tecnología como solución de base [4]. En todos los casos se le aplica como una alternativa para el resguardo de información con capacidad para tolerar fallas. En muchos incluso se hace referencia a los algoritmos de dispersión que aquí mismo se revisan(sec. 3.7). No obstante, la originalidad de nuestra investigación radica en proponer el almacenamiento de los estados globales y no de los datos. También hemos desarrollado una serie de esquemas de almacenamiento que representan una manera innovadora de coordinar a los dispositivos de almacenamiento.

Acerca del problema de partición (SP) relativo a la gestión de un sistema distribuido, debemos mencionar que nuestro enfoque tiende un puente entre dos metodologías bien conocidas y su originalidad radica en combinar los recursos de cada una de ellas. Por un lado, las heurísticas basadas en agentes han sido estudiadas desde hace más de diez años, en los que han demostrado sus ventajas (y limitaciones) en la solución de problemas de optimización [5]. Por otro lado, existe también una teoría de los algoritmos distribuidos, construida a lo largo de los últimos quince años (aprox.). Desde cada lado del puente se han trabajado algunas variantes

del problema SP. Sin embargo, el mayor inconveniente del enfoque centralizado [6] (pensando su aplicación bajo nuestras condiciones) es el desperdicio de los propios recursos de cálculo y las limitaciones en la escala del problema. En tanto, desde el lado distribuido se han resuelto problemas de partición donde no se exige un tamaño único en cada una de las regiones a las que se da origen [7]. Curiosamente, en ambos enfoques se utiliza una versión del algoritmo de búsqueda en profundidad (DFS centralizado ó distribuido, respectivamente) como algoritmo de exploración. Nosotros también basamos la simulación de cada "hormiga" en un algoritmo DFS distribuido, con el que recorremos aleatoriamente al sistema, construimos las particiones y medimos sus diámetros.

Finalmente, por lo que concierne a la construcción de herramientas DES, existe un conocimiento maduro sobre la ingeniería de estos sistemas, basado en el enfoque orientado a objetos [8]. La novedad de nuestro trabajo consiste en aplicar esta experiencia para la construcción de una solución hecha "sobre medida", que soporta modelos de autómatas y ofrece al usuario la flexibilidad para modificar los escenarios de comunicaciones sobre los que se quiere experimentar.

1.5 Metodología

Los fundamentos de la primera parte, referente a la definición de los esquemas de almacenamiento, se encuentran en la fuerte conexión entre el álgebra y la combinatoria [9, 10, 11]. Los resultados que se producen son de tipo analítico y establecen los métodos de construcción y los parámetros de desempeño de cada solución.

La segunda parte, referente a la escalabilidad de un esquema de almacenamiento, es de una naturaleza más experimental. Se sabe que la partición es un problema NP-completo. Nuestro enfoque utiliza modelos de cómputo inspirados en sistemas biológicos [12]. Proponemos un método distribuido que es una adaptación del algoritmo de búsqueda en profundidad (DFS) [13]. A continuación, demostramos que el algoritmo construye una solución heurística correcta, cuya calidad puede mejorarse en rondas sucesivas.

Por último, en la tercera parte referente a la plataforma de simulación, utilizamos un enfoque de diseño y programación orientados a objetos, sobre un modelo sencillo de simulador [8], que adaptamos para convertirlo en una herramienta adhoc.

1.6 Estructura de la tesis

El capítulo 2 es el antecedente teórico que contextualiza la investigación. En primer lugar, se presenta a la simulación como una metodología para manipular modelos cuantitativos, destacando los métodos de eventos discretos. Posteriormente, se describen los componentes que integran una simulación distribuida y se revisan los problemas abiertos o de vanguardia relacionados con la construcción de sistemas de alto rendimiento. Enseguida se revisan conceptos de computación confiable y modelos de falla en sistemas distribuidos. Por último, se presenta una lista de los sistemas actuales (o en proyecto) que usan el almacenamiento distribuido como componente clave de su construcción.

El capítulo 3 es un estudio en profundidad sobre el problema del almacenamiento distribuido de estados globales. Cuando se realiza una ejecución distribuida en la que los procesadores participantes pueden experimentar una degradación de sus capacidades, se sugiere almacenar regularmente el estado global del sistema para poder recuperar y restaurar la ejecución en un estado del que se tenga registro. El almacenamiento de la historia de una ejecución distribuida plantea un problema en la administración de los sitios designados para el depósito de la información.

Un *esquema de almacenamiento* es un procedimiento distribuido que se ejecuta sobre los sitios que componen una red de computadoras y se desarrolla por pasos, en instantes discretos de tiempo. Cada vez que se invoca, se selecciona un subconjunto de los sitios, denominado *bloque*, para participar con sus discos locales como espacio de almacenamiento o bodegas. Todos los participantes conocen los sitios que trabajan en este papel. Además, la composición de los bloques cambia con el tiempo hasta que se recicla. Los esquemas de almacenamiento que se presentan cumplen con las siguientes propiedades:

- El conjunto de bloques que va a usarse se define estáticamente al inicio de las operaciones. Los bloques se calendarizan de manera cíclica (uno por cada paso de almacenamiento) en un orden predefinido.
- Cada sitio debe participar en el menor número bloques que sea posible, pero también debe aparecer en tantos como cualquier otro sitio.
- Los bloques deben tener tantos sitios como sea posible para reducir la cantidad de espacio que se requiere en cada sitio individual.

Aquí se incluye el modelo formal que subyace en todos los esquemas de almacenamiento. Se introducen también los parámetros que caracterizan a cada solución: el *período de almacenamiento*, el *número crítico de fallas*, el *precio* y se define el concepto de *estrategia balanceada e incompleta*. Enseguida se desarrollan tres métodos alternativos, denominados estrategias simples, con los que puede construirse la colección de bloques de almacenamiento, se evalúan sus parámetros y se establece una comparación entre ellos. Por último, se desarrolla un cuarto método que consiste en una regla de composición de dos estrategias simples y se consideran posibles extensiones del trabajo como: calendarios intercalados/aleatorios, técnicas de dispersión de información [14], componentes de refacción o repuesto.

El capítulo 4 presenta un algoritmo distribuido para construir soluciones heurísticas del problema de partición (SP). En primer lugar se establece el modelo formal de sistema distribuido sobre el que se trabaja, denominado *red asíncrona*. Se define formalmente el problema y se revisan las propiedades del algoritmo de búsqueda en profundidad, al que se considera el bloque elemental de construcción. Enseguida, se describe cómo funciona un *sistema de hormigas*. Se trata de una estrategia para atacar problemas de optimización combinatoria, basada en el comportamiento de estos sistemas biológicos. En nuestra propuesta, los agentes, también llamados hormigas, se colocan en algún nodo de la red de comunicaciones. A continuación, cada uno desarrolla un camino aleatorio bajo el control de probabilidades de tránsito que van ajustándose. Cuando un agente ha visitado todos los nodos, se evalúa su viaje y se asignan valores de "feromona" (nuevas probabilidades) a las aristas de su camino, en una

cantidad proporcional a la calidad de la solución planteada por su recorrido. Este procedimiento puede repetirse varias veces para mejorar la solución. A lo largo del recorrido de un agente se eligen nodos denominados concentradores, encargados de administrar una región de la solución y medir su calidad local. Se demuestra que este algoritmo es correcto, pues cada nodo es visitado y asignado a una subgráfica, cada subgráfica tiene el mismo número de nodos, por cada una existe un concentrador y, al terminar el algoritmo, el nodo inicial dispone de la información completa para determinar la calidad de la partición que acaba de construirse. Creemos que varios problemas de optimización distribuida, como la (re)distribución y el balance de carga, pueden beneficiarse con este enfoque.

El capítulo 5 describe la construcción de una herramienta para simulación de eventos discretos, con la que desarrollamos un prototipo del sistema de hormigas presentado en el capítulo anterior. Para los fines de nuestra investigación construimos una utilería que reune las siguientes características

- Se trata de una plataforma flexible, de código abierto, que separa al algoritmo del entorno de comunicaciones sobre el que se despliega. Con ello se puede ejecutar el mismo algoritmo sobre diferentes topologías, sin modificar una línea de código.
- Se pueden simular eventos aleatorios, dejando al programador en libertad para especificar cualquier distribución de tiempo, ya sea en la duración de un paso de procesamiento dentro de un nodo, o en el retardo de transmisión de un mensaje sobre un canal.
- Se cuenta con un sencillo mecanismo de paso de mensajes, que el usuario puede extender para definir sus propias unidades de información.
- Cada entidad activa puede modelarse como un autómata simple o compuesto, dependiendo de la complejidad de su comportamiento.
- Se puede simular la ejecución simultánea del mismo algoritmo, para estudiar, por ejemplo, una operación de recorrido iniciada desde diferentes puntos de una red.
- Se puede simular la ejecución simultánea de diferentes algoritmos, para estudiar cooperación u otro tipo de interacciones.
- Se puede utilizar para investigación o docencia. Tiene una interfaz gráfica de usuario (GUI) programada con Tcl/Tk con la que puede desplegarse una simulación animada. Las funciones de la GUI incluyen edición de topologías, ventana de visualización, ventana para despliegue de trazas y controles de la simulación.

Primeramente, se establece que esta herramienta está orientada a la simulación basada en modelos de máquinas de estados, como los autómatas de entrada-salida y las máquinas comunicantes de estados finitos. En seguida, se describe la arquitectura del simulador, utilizando conceptos de diseño y programación orientados a objetos. Luego se presenta un ejemplo donde se implanta el sistema de hormigas y se resuelve una instancia del problema SP. Por último, se revisan algunos usos alternativos de nuestro simulador y se consideran las direcciones de trabajo futuro.

El capítulo 6 es una vista panorámica que revisa las contribuciones de esta tesis y señala una posible agenda de investigación y aplicaciones futuras.

Capítulo 2

Fundamentos

Para aprovechar la naturaleza concurrente de algunos sistemas estudiados mediante DDES, el modelo del sistema se disgrega en objetos de menor tamaño que se reparten sobre los diferentes procesadores de un sistema de cómputo distribuido. Sin embargo, en la medida que la DDES se utiliza para el estudio de sistemas de mayor escala y complejidad, surgen nuevos problemas de construcción. Entre los requerimientos que más se buscan en un sistema distribuido, se encuentra la tolerancia a fallas. Se espera que, de manera automática, el sistema reconozca un comportamiento erróneo y reponga el curso normal de su ejecución. Este capítulo ofrece una visión panorámica sobre la simulación distribuida de eventos discretos, desde sus conceptos básicos de construcción y operación, hasta sus problemas actuales que enmarcan nuestra investigación. Igualmente, incluye aquellas fuentes con las que se relacionan y le dan sustento teórico, como los modelos de falla y conceptos de cómputo confiable, así como los entornos en que se aplica el almacenamiento distribuido.

"Desde hace mucho tiempo la invención ha alcanzado sus límites, y me parece imposible que en el futuro puedan darse nuevos desarrollos".

Julius Frontinus, famoso ingeniero romano del siglo X d.C.

2.1 Simulación de eventos discretos (DES)

La simulación por computadora comprende varios métodos numéricos de análisis cuantitativo, incluída la simulación *dirigida por eventos* o simulación *de eventos discretos* (DES: Discrete Event Simulation). Los modelos en que ésta se sustenta, se caracterizan por un conjunto numerable de estados entre los que se pueden definir transiciones que pueden ocurrir sobre un eje de tiempo. Partiendo de ciertas condiciones iniciales, el modelo genera nuevos eventos sobre la marcha. En sistemas de cómputo, las simulaciones dirigidas por eventos son ampliamente preferidas por cuanto el sistema es fácilmente representado como un conjunto de estados discretos, por ejemplo: el número de tareas pendientes de atención en un sistema operativo, el número de paquetes que atraviesan un nodo de conmutación, un algoritmo distribuido como el de la fig. 2.1.

Todas las simulaciones de eventos discretos tienen una estructura común. Independiente del sistema que se modele, la simulación tendrá la mayoría de los elementos que se listan

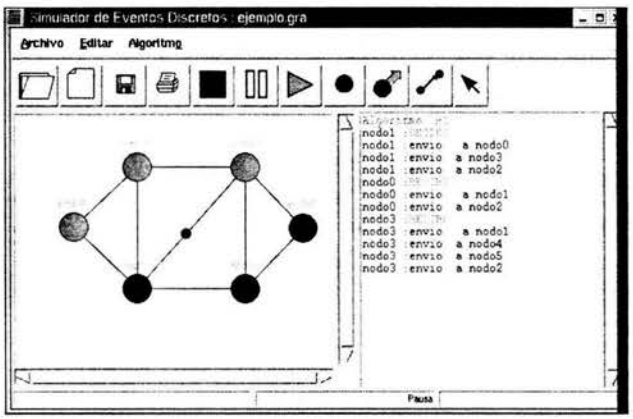

Figura 2.1: Simulación de un algoritmo distribuido usando DES.

enseguida. Si se utiliza un lenguaje de propósito general, todos los componentes tendrán que ser implementados por el analista. En cualquier otro caso, el software que se utilice proveerá estas partes [15]:

- Calendarizador de eventos.
- Reloj y mecanismo de actualización de tiempo.
- Variables de estado.
- Rutinas para manejo de eventos.
- Rutinas de entrada.
- Generador de reportes.
- Rutinas de inicialización.
- Rutinas para registro de trazas.
- Mecanismos para manejo dinámico de memoria.
- Programa principal.

En la simulación de eventos discretos una práctica común es 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 el estado en que se encuentre en función de su historia y de cada mensaje aceptado [16].

Partiendo de la red de procesos físicos $(pp_0, \ldots, pp_{n-1})$, se construye un programa de computadora denominado *sistema lógico* o simulador, compuesto a su vez por *procesos lógicos* $(lp_0, \ldots, lp_{n-1})$ encargados de representar a los procesos físicos.

Muchos sistemas reales pueden describirse en los términos de una red de procesos físicos (también llamado sistema físico, para diferenciarlo del sistema real): Un sistema de cómputo, por ejemplo, compuesto de un CPU, discos, memoria y terminales para ingreso de tareas, puede conceptualizarse como una red de procesos; el CPU interactúa con los discos enviando mensajes que soliciten o liberen espacio de almacenamiento; una terminal envía mensajes al CPU que representan tareas que deben ejecutarse.

Las técnicas tradicionales con que se desarrolla la simulación de eventos discretos, utilizan un algoritmo en el que el calendarizador administra una estructura de datos denominada *lista de eventos*. En ella se almacenan los mensajes (ordenados de acuerdo a sus tiempos de transmisión) que han sido programados para atenderse en el futuro (esto es, cuando el reloj de la simulación avance hasta habilitarlos). Se garantiza que un mensaje será enviado en el tiempo que tiene asociado, siempre que el transmisor no reciba, antes de este tiempo, otro mensaje que lo cancele. En cada ciclo, el mensaje asociado al menor tiempo futuro es removido de la lista de eventos y se simula la transmisión del mensaje correspondiente en el sistema físico. La transmisión de un mensaje puede, a su vez, causar el envío de otros mensajes en el futuro (que entonces son agregados a la lista de eventos) o la cancelación de mensajes previamente programados (que entonces se remueven de la lista). El reloj se avanza entonces, hasta el tiempo del mensaje cuya simulación acaba de completarse.

2.2 DES distribuida (DDES)

La naturaleza del mecanismo para manejo de la lista de eventos impone una simulación en serie de los eventos del sistema físico, puesto que en cada ciclo del algoritmo sólo se remueve un mensaje de la lista, se simulan sus efectos y, si así se requiere, se actualiza la lista. En consecuencia, el análisis de los grandes sistemas se ve limitado en el número de eventos que pueden simularse. En el caso de los nuevos sistemas de telecomunicaciones, por ejemplo, un experimento de simulación puede tomar varias horas e incluso días, antes de producir un solo resultado.

Si se tiene un conjunto de computadoras conectadas mediante canales rápidos y confiables, se puede plantear un método alternativo de procesamiento: la *Simulación Distribuida de Eventos Discretos* (DDES: Distributed Discrete Event Simulation). Con este nuevo enfoque se puede acelerar la producción de resultados al repartir la carga de trabajo entre los procesadores del sistema. El manejo de los objetos o variables compartidas de la simulación secuencial - el reloj y la lista de eventos - cede su lugar a un nuevo tipo de algoritmo en donde una máquina puede simular una parte del sistema físico y donde las interacciones entre componentes se simulan transmitiendo mensajes entre máquinas. Estos mensajes incluyen una estampilla de tiempo virtual, que representa el instante en que debe ocurrir la comunicación de las partes[1].

[1] Debe entenderse que hay una diferencia entre los mensajes intercambiados entre procesos lógicos y los

Las máquinas pueden operar concurrentemente mientras sus contrapartes físicas trabajen de forma autónoma, en tanto que deberán detener su ejecución y esperar la recepción de nuevos mensajes, para simular un evento de sincronización entre componentes. Sin embargo, la ejecución concurrente de varios eventos a lo largo del tiempo simulado, puede ser una fuente de problemas relacionados con la factibilidad del sistema físico. En un escenario de ejecución concurrente debe cuidarse que el procesamiento de eventos respete las relaciones de dependencias que puedan existir, de lo contrario, ocurrirán *errores de causalidad.*

Supóngase una simulación basada en el nuevo paradigma de ejecución distribuida. El sistema que se modela puede describirse como un conjunto de procesos físicos ($pp_0, pp_1, \ldots,$) que interactúan en varios instantes del tiempo simulado. Enseguida se construye una simulación del modelo, como un nuevo conjunto, esta vez de procesos lógicos ($lp_0, lp_1, \ldots,$), uno por cada proceso físico. Todas las interacciones entre los procesos físicos se simulan intercambiando mensajes con estampilla de tiempo entre los procesos lógicos correspondientes. Cada proceso lógico contiene una porción del estado global de la ejecución, correspondiente al proceso físico que representa, así como un reloj local que registra el progreso de éste. Se garantiza que no existirán errores de causalidad siempre que cada *lp* respete la *restricción de causalidad local*:

> *Una simulación de eventos discretos, consistente en una colección de procesos lógicos que exclusivamente se relacionan intercambiando mensajes, obedece la restricción de causalidad local, si y sólo si, cada proceso lógico ejecuta eventos en un orden no decreciente según sus estampillas de tiempo.*

Obsérvese cómo un proceso lógico debería detenerse hasta recibir los mensajes de todos los demás componentes simulados con los que se relaciona, para entonces procesar al de menor estampilla. Este último aspecto, sin embargo, representa una fuente de problemas al introducirse la posibilidad de interbloqueo o estancamiento (deadlock) en que pueden caer un conjunto de procesos lógicos que realizan un procedimiento de sincronización-interacción.

Los métodos para hacer frente al riesgo de interbloqueo en simulaciones distribuidas pueden agruparse en dos grandes familias. Por un lado, se encuentran los métodos *optimistas* en los que se evita a toda costa el riesgo de caer en el estancamiento, aún cuando se produzcan ejecuciones donde pueden violarse las relaciones de causalidad, que después deben corregirse. El protocolo de entrelazado de tiempo (time warp) es el procedimiento fundamental de esta familia. En contraste, se encuentran los métodos *seguros* (o conservadores), en los que es imposible ejecutar acciones fueran de orden cronológico (y causal) pero, a cambio, se requiere un procedimiento para prevenir ó corregir una situación de interbloqueo (para tratar con esta contingencia se prefiere la prevención sobre su corrección, por razones de costo y simplicidad). En los métodos seguros es necesario transmitir mensajes de control que no tienen ningún significado o correspondencia con el sistema físico que se simula y cuya tarea es el intercambio de los estados parciales de las máquinas, con los que se pueda inferir el estado global de la ejecución. El protocolo de mensajes nulos es quizás el más utilizado en esta clase de procedimientos [16, 17, 18].

---

mensajes intercambiados entre máquinas pero, ya que existe una asociación biyectiva entre los elementos de ambos conjuntos, se habla indistintamente de un tipo u otro.

1. *El entrelazado de tiempo.* Es un método para la prevención del interbloqueo entre máquinas, en donde se supone (optimistamente) que los mensajes llegarán a sus destinos en el tiempo y orden adecuados y, por tanto, las máquinas nunca pueden quedar estancadas, pues cada componente reacciona inmediatamente al mensaje que recibe sin esperarse a contar con una colección completa de eventos de donde pueda elegir al de menor tiempo (supone que siempre recibe en orden creciente de estampillas). Sin embargo, en la práctica puede ocurrir la recepción tardía de mensajes cuyo fechado precedía cronológicamente (y puede ser que causalmente también) a otros que ya fueron atendidos. Cuando esto ocurre se activa un *mecanismo de restauración* (rollback) que devuelve la ejecución hacia un *estado válido* (checkpoint) en el que se garantiza que todos los eventos habían sido despachados en orden cronológico.

2. *Los mensajes nulos.* En esta técnica, el mensaje de control contiene el estado del reloj local de un componente y se envía cuando éste no prevé interactuar con su vecino en los próximos instantes. De esta manera, el vecino puede disponer del conjunto completo de eventos que le indican cuál es el próximo componente con el que deberá sincronizarse.

Parece ser que el desempeño de los sistemas de tiempo entrelazado depende mucho de la latencia de los canales y de la granularidad de los eventos que se simulan. En general, esta técnica resulta más eficiente que su competencia, en escenarios asíncronos de grano grueso. Por otro lado, el inconveniente de los métodos seguros es que sólo funcionan en escenarios fijos, esto es, no puede modificarse la red de procesos físicos durante la simulación y, además, se debe garantizar que los canales de comunicación que se utilizan entregan sus mensajes en el orden en que se les expide (es decir, que son canales FIFO).

En todo caso, la simulación distribuida ofrece muchas ventajas sobre las técnicas secuenciales, además de la posible aceleración de todo el proceso: requiere muy poca memoria adicional; comparada con la simulación secuencial, cada máquina controla una parte relativamente pequeña de la ejecución; la simulación puede adaptarse a la disponibilidad de los recursos de hardware, esto es, con pocas máquinas se pueden simular varios procesos físicos en una sola computadora.

2.3 Problemas abiertos de la DDES

A pesar de los importantes avances en las tecnologías que dan sustento a la DDES 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 masivo de elementos de procesamiento que estarán involucrados. Se espera que los próximos simuladores sean capaces de decidir automáticamente 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 fallas [19].

Acerca de la DDES y sus temas de investigación pueden mencionarse algunas líneas bien reconocidas y que son objeto del interés de la comunidad científica:

1. Se requieren estudios analíticos y experimentales sobre la influencia de los diferentes parámetros del simulador en el rendimiento de la simulación: granularidad, latencia de los canales, intervalo de verificación (checkpoint), estabilidad de la restauración (rollback), distribución inicial de la carga [20, 21, 22, 23]. Esta parece una empresa de la que sólo pueden esperarse resultados parciales, toda vez que la simulación es una técnica de estudio cuantitativo, a la que se recurre precisamente cuando no es posible aplicar soluciones analíticas sobre los sistemas que se evalúan. En consecuencia, es difícil hablar de modelos completos lo suficientemente generales como para determinar el escenario óptimo de una simulación arbitraria.

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 [24, 25, 26].

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 una simulación en la que participen cientos o miles de computadoras, durante horas (o días), no puede tolerarse que la reducción en las capacidades de cálculo de una computadora (incluído el paro total) obligue a restaurar la ejecución desde cero. En estas condiciones deberán incorporarse procedimientos de recuperación en caso de contingencias tales como las fallas de paro o la sobrecarga, que puedan presentarse en algunos de los sitios participantes.

Hay que notar la diferencia entre la información que se almacena en caso de que ocurra una restauración (rollback), de aquella que se almacena para hacer frente a una falla de paro. En el primer caso, cada sitio guarda su estado en su propia memoria local. En tanto, en el segundo caso, la información se almacena en un sitio que no necesariamente es el mismo donde se genera. Si ocurre una restauración se trata de una falla del procedimiento de sincronización, que no obliga a redistribuir o migrar ninguna tarea. En cambio, si ocurre una falla de paro, será necesario reemplazar o localizar las tareas que se ejecutaban en el sitio descompuesto.

2.4 Estado global de una ejecución distribuida

Muchos problemas importantes del cómputo distribuido admiten soluciones que contienen una fase en la que se requiere la detección de una propiedad global, la cual puede entenderse como una instancia del problema de *Evaluación de un predicado global*. El objetivo consiste en evaluar una expresión booleana cuyas variables están referidas a un estado global del sistema[2] [28].

El estado global de un sistema distribuido es la unión de los estados individuales de los procesos que lo integran. Tratándose de un sistema asíncrono en el que no se dispone de algún recurso común para almacenamiento de información, la única forma en que un proceso puede conocer el estado de los demás componentes será intercambiando mensajes. Sin embargo, las limitaciones propias del sistema pueden llevar a un proceso a la construcción de un estado global obsoleto, incompleto ó inconsistente (informalmente se entiende que un estado es inconsistente cuando jamás pudo ocurrir). Al mismo tiempo, la variabilidad de los retardos de la comunicación podría producir, para un mismo instante de la ejecución, dos estados globales diferentes observados desde dos procesos diferentes.

Algunos ejemplos de problemas sobre sistemas distribuidos en los que la solución o una parte de ella puede codificarse como un predicado global pueden ser: la detección del bloqueo o estancamiento (deadlock), la detección de la terminación, la detección de la pérdida de una ficha, la recolección de basura, la construcción de puntos de verificación (checkpoints), la restauración de una ejecución y, en general, la depuración y el monitoreo.

> *Un programa distribuido que se ejecuta sobre un sistema asíncrono con intercambio de mensajes, se compone de un conjunto de n procesos* $p_1, \ldots, p_n$, *que no comparten ninguna memoria común y tampoco disponen de un reloj global al que se pueda leer instantáneamente, desde algún punto de la red. Se asume que los canales transportan sin pérdida los mensajes que viajan en su interior, con un retardo finito, pero impredecible, y que la ejecución de un proceso, como la transferencia de un mensaje, ocurren espontáneamente. Finalmente, se supone que los mensajes tienen una longitud finita [29], [30].*
>
> *Se definen tres tipos diferentes de* eventos *que un proceso puede ejecutar:* internos, de transmisión de mensajes *y* de recepción de mensajes. *Un evento interno sólo afecta el estado del proceso en que ocurre. Se trata, por tanto, de una acción local. Por otra parte, los eventos de transmisión y recepción representan el flujo de información entre procesos y establecen una relación causa-efecto entre la transmisión de un mensaje y su correspondiente recepción.*

La *historia local* del proceso p_i es una secuencia (posiblemente infinita) de eventos:

[2] Los conceptos que se revisan en esta sección puede abordarse también desde el enfoque de la teoría de los sistemas de eventos discretos [27], en donde se derivan nociones alternativas. Sin embargo, hemos preferido un tratamiento clásico del cómputo distribuido por varias razones importantes. En primer lugar se trata de un marco teórico aceptado por la comunidad científica dentro de la cual se dio este trabajo de investigación. Por otro lado, preferimos utilizar herramientas formales con las que estuvieramos familiarizados. Finalmente, para los fines de esta tesis, se da por hecho la existencia de un estado global que, independientemente de como se defina, registra un hecho del pasado y debe almacenarse.

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

ocurridos en p_i. El superíndice de la notación, denominada *numeración canónica*, corresponde con el orden total impuesto por la ejecución secuencial de los eventos locales. Sea $h_i^k = e_i^1, e_i^2, \ldots, e_i^k$ el prefijo de la historia local conteniendo sus primeros k eventos. Se define h_i^0, como la secuencia vacía. La *historia global* de la ejecución es el conjunto $H = h_1 \cup \ldots \cup h_n$, conteniendo la historia local de todos los participantes del sistema.

Obsérvese que la historia global no especifica los tiempos relativos entre sus eventos. En el contexto de un sistema asíncrono, los eventos pueden ordenarse en base a la noción de "causa y efecto", previamente mencionada. En otras palabras, dos eventos están obligados a suceder en cierto orden, sólo si la ocurrencia de uno puede afectar la existencia del otro. La información fluye de un evento a otro, ya sea porque ambos eventos ocurren en el mismo proceso, y por tanto inciden sobre un mismo estado local, o bien porque los eventos ocurren en procesos diferentes y corresponden al intercambio de un mensaje.

Se pueden formalizar estas ideas definiendo la relación $\rightarrow$ de *dependencia causal* sobre los eventos de una ejecución [31]:

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 $e \rightarrow e''$.

La dependencia causal permite caracterizar una *ejecución distribuida* como un conjunto parcialmente ordenado definido por la pareja $(H, \rightarrow)$. Hay que entender que la relación causa-efecto sólo se cumple cuando una pareja de eventos representan el segundo caso que establece a $\rightarrow$. En cualquier otra circunstancia, la única conclusión que puede inferirse de la relación $e \rightarrow e'$ es: la ocurrencia de e *podría* haber afectado la ocurrencia de e'.

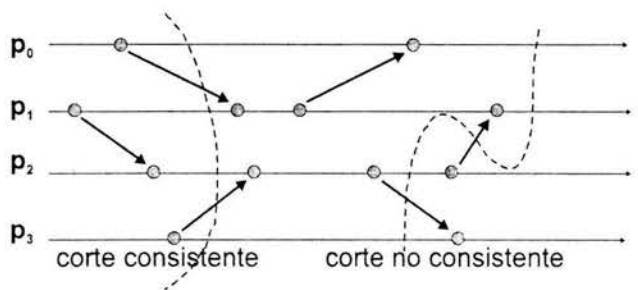

Figura 2.2: Dos cortes de una ejecución distribuida

Algunos eventos de la historia global pueden no estar causalmente relacionados. En otras palabras, es posible que para algunos e y e' no se cumpla que $e \rightarrow e'$ y tampoco $e' \rightarrow e$. En tal

circunstancia se dice que son eventos *concurrentes*, lo que se denota como $e||e'$. La figura 2.2, denominada diagrama espacio-tiempo, muestra una ejecución distribuida en cuatro procesos. Cada línea horizontal representa el progreso de un proceso diferente. Un punto indica un evento y una flecha indica la transferencia de un mensaje.

> Sea σ_i^k el estado local del proceso p_i inmediatamente después de ejecutar el evento e_i^k y σ_i^0 su estado inicial antes de ejecutar algún evento. El *estado global* de una ejecución distribuida será la n-tupla $\Sigma = (\sigma_1 \ldots \sigma_n)$ formada 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 cada una de las historias locales. Se puede especificar un corte $C = h_1^{c_1} \cup \ldots \cup h_n^{c_n}$ mediante la tupla de números naturales $(c_1 \ldots c_n)$ correspondiente al índice del último evento ocurrido en cada proceso. El conjunto $(e_1^{c_1} \ldots e_n^{c_n})$ de los últimos eventos incluidos en el corte se denomina la *frontera* del corte. Evidentemente, cada corte $(c_1 \ldots c_n)$ define un estado global $(\sigma_1^{c_1} \ldots \sigma_n^{c_n})$.

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.

Sea p_0 un proceso denominado *monitor*, encargado de construir el estado global del sistema en que participa. De primera instancia se puede proponer que el monitor envíe mensajes hacia todos los procesos del sistema, interrogándolos acerca de su estado local. Un proceso p_i que reciba este mensaje, contestará devolviendo su estado local σ_i. Una vez que todos los procesos del sistema hayan respondido, p_0 puede construir el estado global $(\sigma_1 \ldots \sigma_n)$. Esta solución inicial tiene una deficiencia seria, sabiendo que el monitor está sujeto a las mismas incertidumbres que el resto del sistema. Es evidente que el estado global que construya puede carecer de un significado real. Visto de otro modo, mientras que cada corte de una ejecución distribuida corresponde con un estado global, sólo algunos cortes corresponden con estados que *pudieron* ocurrir. En la fig. 2.2 se observan dos cortes realizados sobre una ejecución distribuida. En el primer caso, (de izquierda a derecha) el corte representa un estado global que sí ocurrió. Por otro lado, el segundo caso representa un estado global que nunca pudo ocurrir puesto que, en la frontera del corte, p_1 reporta la recepción de un mensaje del que p_2 no tiene registro. Esta clase de cortes pueden llevar a conclusiones equivocadas acerca de la condición del sistema.

La relación de precedencia causal $\rightarrow$ resulta ser una herramienta muy valiosa para distinguir entre los dos tipos de corte recién descritos. Se dice que un corte C es *consistente*, si para todos los eventos e y e'

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

y en cualquier otro caso es *inconsistente*. Luego, un *estado global consistente* será aquel que corresponda con un corte consistente. Igualmente, una corrida R se dice *consistente* si para todos los eventos $e \rightarrow e'$ implica que e aparece antes que e' en R. En otras palabras, el

orden total impuesto por R sobre los eventos es una extensión del orden parcial definido por la relación de precedencia causal. Se entiende que una corrida $R = e^1 e^2 \ldots$ produce una secuencia de estados globales $\Sigma^0\Sigma^1\Sigma^2 \ldots$, en la que Σ^0 representa el estado global inicial $(\sigma_1^0, \ldots, \sigma_n^0)$. Por ello, en lo sucesivo, se usa el término "corrida" para referirse a una secuencia de eventos, así como a la secuencia de estados a que se da lugar.

2.5 Conceptos de computación confiable

La garantía de funcionamiento de un sistema es la propiedad que le permite a un usuario mantener una confianza en el servicio que se le ofrece. Según las aplicaciones del sistema, las diferentes facetas de la garantía de funcionamiento vendrán a jugar un papel más o menos importante que, de acuerdo con el interés con el que se quiera sustentar un servicio, podrán clasificarse a partir de la siguiente lista de criterios [32]:

- Con relación a la rapidez con el que 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].

Laprie ofrece un conjunto de definiciones y clasificaciones que son aceptadas por la comunidad que trabaja en la denominada computación confiable (dependable computing) o tolerante a fallas (fault-tolerant) [33]:

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

La falla sobreviene porque el sistema tiene un comportamiento erróneo, cuya causa será un *deficiencia* (fault). Un *error* será entonces la manifestación de una deficiencia del sistema y una falla será el efecto de un error sobre el servicio[4].

Lo primero que se observa al emprender el estudio sistemático de las causas que producen las fallas en los sistemas (informáticos o de otro tipo), es que se trata de un importante conjunto de fenómenos que entran bajo la misma definición y que, además, pueden proceder de fuentes muy diversas. Se reconocen como los criterios más aceptados para clasificarlas, aquellos que las dividen de acuerdo con su naturaleza, origen y persistencia. Por su *naturaleza*, se dividen en accidentales e intencionales. Por su *origen*, se pueden ordenar a su vez, según su

[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.

[4] De manera semejante, existe un debate aún sin resolver sobre la traducción de la palabra "fault". No se acepta mucho el uso de "deficiencia" como su equivalente. En cambio se prefiere usar "falla" aún cuando la palabra "failure" también se traduzca con el mismo término.

causa fenomenológica (físicas ó humanas), el lugar donde se originen con relación a las fronteras del sistema (internas ó externas) y la fase de creación (de concepción u operacionales). Por su *persistencia*, se dividen en temporales y permanentes.

Por cuanto se refiere a los errores, estos pueden tener efectos o no, dependiendo de tres factores principales: La *composición del sistema*, esto es, la presencia de funciones redundantes o de respaldo. La *actividad del sistema*, que puede corregir errores antes de que generen desperfectos. La *percepción de la falla*, que depende de la subjectividad del usuario.

Por otro lado, en vista de la diversidad de formas en que se manifiestan, las fallas se pueden clasificar desde varios puntos de vista: por su dominio, por la percepción que de ellas tenga el usuario y por su gravedad.

El dominio de una falla conduce a distinguir entre fallas de valor y fallas temporales. Las primeras ocurren cuando los valores numéricos del servicio ofrecido no se apegan a la especificación. Las segundas sobrevienen cuando los instantes en que se ofrece el servicio no atienden a la especificación. También debe mencionarse el caso de las fallas por omisión, cuando no se entrega un servicio. Este tipo de falla se puede ver como un caso límite de la falla de valores (valor ausente) o la falla temporal (falla de retardo infinito).

Cuando un sistema sirve a varios usuarios, debe distinguirse, de acuerdo con la percepción que estos tengan, entre fallas coherentes y no coherentes. En el primer caso, todos las perciben de la misma forma mientras que, en el segundo, varios usuarios pueden tener distintas percepciones de una misma falla.

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 [34]:

- 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).

- 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:

- *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 contraste con el enfoque clásico de teoría de la confiabilidad, 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. La caracterización estadística de un sistema, usando medidas como el tiempo medio entre fallas (mean time to failure) y la probabilidad de fallas (probability of failure) son importantes para los usuarios. Sin embargo, existen ventajas reales al describir un sistema en término del máximo número de componentes deficientes que puede tolerar sobre un intervalo de interés. Afirmar que un sistema es f-tolerante es una medida de las capacidades inherentes a la arquitectura del sistema, en contraste con la tolerancia que se consigue usando componentes con un cierto grado de confiabilidad.

> *El propósito de un sistema tolerante a fallas es detectar y, si es posible, reparar los errores antes de que se manifiesten (en el interfaz del usuario). La clave del diseño consiste en el uso de la redundancia. El término* redundancia *se refiere a los recursos adicionales que se incluyen en un sistema, que no se usarían en una situación perfecta. La redundancia se requiere para detectar errores y enmascarar a las fallas en el mundo real. Se puede distinguir entre tres tipos de redundancia [35]: 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 trama, 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.

Los tres tipos de redundancia se emplean en el diseño de sistemas tolerante a fallas. La redundancia de la información se usa para proteger información relativa al estado de un sistema. La redundancia en el tiempo se utiliza para detectar y soportar la ocurrencia de fallas temporales. La redundancia de recursos se requiere para tolerar fallas permanentes en el hardware o software.

Se dice también que un sistema emplea *redundancia pasiva* (algunas veces llamada fría) si utiliza recursos físicos redundantes que se activan sólo cuando el déficit de un recurso primario puede producir una falla. Implícitamente, esto supone que el sistema incluye mecanismos de detección para capturar errores antes de que se produzca una salida incorrecta en el interfaz del usuario. Por otro lado, la *redundancia activa* (o caliente) se refiere a una organización que activa simultáneamente todos los recursos físicos redundantes, se requiere que todos los subsistemas replicados transiten por los mismos estados al mismo tiempo.

2.6 La evolución del almacenamiento distribuido

La memoria es un recurso fundamental de la computadora. Los sistemas de almacenamiento ofrecen esta capacidad a un costo inferior que la RAM y con una estabilidad de largo plazo. No obstante, también presentan ciertos inconvenientes, por ejemplo, anchos de banda que limitan las tasas de transferencia y latencias de acceso muy altas. Estos atributos - costo, persistencia, ancho de banda y latencia - son las métricas con las que tradicionalmente se evalúan los sistemas de almacenamiento. Sin embargo, el acentuado desarrollo de las redes de telecomunicaciones observado en las últimas décadas, ha hecho más complejo el escenario.

En la actualidad, la Internet es una parte integral de la computadora. Los usuarios consultan páginas Web, cuyo despliegue requiere la recuperación de información desde docenas de sitios diferentes alrededor del mundo. Este ha sido el primer sistema de almacenamiento distribuido de información con una escala global, lo cual ilustra el impacto tecnológico, económico y cultural del enfoque distribuido. Sin embargo, su fragilidad semántica y operacional impiden su aplicación para procesamiento de datos. Por ejemplo, los mensajes de error, o la

interrupción del despliegue son problemas comunes cuando la red experimenta una falla en algún punto, que vuelve inasequible una página o alguno de sus elementos.

Por otro lado, también se puede realizar otra forma muy simple de almacenamiento distribuido de archivos, cuando los usuarios de una red local accesan archivos depositados en algún sitio de su misma red. Entre las aplicaciones sobre una red de oficina y las aplicaciones Web (WWW), se encuentra el nuevo dominio del almacenamiento distribuido para empresas, cuya industria comienza a desarrollarse con rapidez.

En este demandante dominio del almacenamiento distribuido se requiere definir nuevas métricas y requerimientos de desempeño, más allá de los parámetros tradicionales. Esto incluye el acceso coherente de la información, la disponibilidad, la permanencia, la seguridad, la interoperabilidad, la búsqueda, el balance de carga y la escalabilidad. Se predice una necesidad de almacenamiento de proporciones inmensas. Esto a su vez implica que la tarea de administración de los grandes sistemas de almacenamiento deberá automatizarse para responder a esa necesidad.

Entre las decisiones que deben encararse para construir esta nueva tecnología, los diseñadores deben elegir a los dispositivos que trabajarán ofreciendo funciones de almacenamiento.

Los enfoques jerárquicos, que incluyen la tendencia hacia la virtualización del almacenamiento, utilizan capas de control y abstracción para salvar las diferencias entre los proveedores de almacenamiento e integrarlos a un espacio virtual. En el enfoque entre-pares ó P2P (peer-to-peer), los clientes mismos ofrecen la capacidad de almacenamiento unos a otros. No existe necesidad de un servidor en el sentido tradicional. En el caso ideal, estos sistemas son completamente simétricos sin un líder o coordinador central. Por otro lado, en el enfoque entre-servidores (server-to-server), relacionado con el anterior, los servidores trabajan de manera simétrica para ofrecer servicios de almacenamiento a sus clientes, los cuales no requieren instalar ningún software nuevo para invocar servicios básicos de almacenamiento.

Para tener una vista panorámica sobre el campo de almacenamiento distribuido, aquí se presenta una lista conteniendo los proyectos más recientes [4]. Por brevedad se omiten los primeros trabajos en el campo, pero se incluyen sus referencias para lecturas posteriores [36, 37, 38].

Past, Intermemory y Farsite son servicios de almacenamiento distribuido de datos, que buscan ofrecer alta disponibilidad, permanencia, desempeño y escalabilidad. Cada uno de estos servicios distribuye réplicas de datos o fragmentos resistentes al borrado, sobre un conjunto de nodos [14] para asegurar disponibilidad y persistencia de largo plazo. Se utilizan complejas estructuras de datos y algoritmos para seguir la pista a las réplicas. Estos sistemas comparten muchos objetivos con los proyectos OceanStore y el DSI de Internet-2. En el contexto de librerías digitales, merece mencionarse el trabajo de Crespo y García-Molina [39] sobre sistemas de almacenamiento de archivos basados en réplicas.

Past. Es una utilería global, cuyo desarrollo está a cargo de Microsoft Research de Cambridge, UK. Utiliza técnicas criptográficas para garantizar la persistencia de largo plazo de los contenidos que almacena [40]. Las réplicas de los archivos se depositan sobre un conjunto de nodos, mediante un algoritmo de enrutamiento tolerante a fallas. El conjunto de nodos que almacenan las réplicas se designa de forma seudoaleatoria, con lo que se consigue balance de carga, así como tolerancia a fallas y resistencia a ataques.

Intermemory. Su objetivo es el almacenamiento de largo plazo de librerías digitales. Puede entendérsele como un sistema P2P, ó bien, como un sistema entre servidores, donde las bibliotecas e instituciones cooperan para crear un sustrato robusto en el que almacenan su colecciones [41, 42]. A diferencia de otros proyectos con metas similares en cuanto a disponibilidad y persistencia, este proyecto implementa una capa de almacenamiento por bloques, que puede soportar estructuras de datos, incluyendo sistemas de archivos convencionales. Otra característica importante es que utiliza dos niveles de dispersión para producir fragmentos resistentes a borrado de tal suerte que, en el almacenamiento de un bloque participan 1024 nodos. El sistema utiliza sincronización de bases de datos entre parejas arbitrarias de nodos para propagar metadatos y fragmentos, con lo que ofrece un mecanismo de autorreparación.

Farsite. Es un sistema distribuido de archivos basado en replicación, donde los clientes contribuyen con recursos de almacenamiento a cambio de un servicio de archivos confiable y de alta disponibilidad [43]. Desde una perspectiva, es un sistema de archivos jerárquico, visible desde todos los puntos de acceso pero, por debajo, los archivos se replican y distribuyen entre las máquinas del sistema. No existe una administración central. Los derechos de escritura sobre los archivos se controlan mediante firmas electrónicas. Todos los archivos son encriptados antes de almacenarse, además los archivos son comprimidos a tiempo de escritura y descomprimidos a tiempo de lectura. Un directorio distribuido registra dinámicamente los sitios donde se almacenan las réplicas y las versiones de un archivo.

OceanStore. Su desarrollo corre a cargo de la Universidad de Berkeley (por encargo de la marina de los Estados Unidos). Se trata de un sistema de escala global que busca ofrecer propiedades tales como la persistencia, alta disponibilidad y desempeño, aún en presencia de fallas, ataques y cambios en las condiciones de red [44]. El tamaño del sistema reclama mecanismos de mantenimiento automático para: localización de objetos y enrutamientos de mensajes, para dispersión de fragmentos usando códigos resistentes a borrado, para actualización de los objetos mediante acuerdo bizantino y, para monitoreo/reconfiguración de operaciones.

I2-DSI (Internet-2 Distributed Storage Iniciative). Es un proyecto que busca construir una plataforma para alojar contenidos replicados [45]. Su estrategia consiste en almacenar las réplicas en las fronteras de la red para mejorar la latencia y reducir el consumo de ancho de banda. Los contenidos se asocian con canales que luego son publicados en un URL. Por su parte, los sitios participantes se suscriben a los canales de su interés y replican sus contenidos. Luego, un servicio de resolución redirige las solicitudes presentadas a un canal, hasta el sitio donde se encuentra su réplica más cercana. Los objetivos de DSI incluyen: protocolos de replicación, consistencia, replicación de contenidos activos y resolución de réplicas.

Napster, Gnutella, Mojo Nation y Freenet son sistemas P2P en los que cada nodo participante contribuye con espacio de almacenamiento para construir un espacio de archivos compartidos con alta disponibilidad. Cada nodo decide los archivos que quiere alojar. En

consecuencia, no existe una garantía de persistencia a largo plazo. Por su parte, Freenet, Publius y Free Haven son algunos de los proyectos que han buscado soportar anonimato de la escritura y resistencia a la censura.

Napster (http://www.napster.com) es un sistema para compartir archivos MP3. Los usuarios contribuyen con espacio de almacenamiento y recursos de red. Naturalmente, los archivos que resguardan sólo están disponibles si estos se encuentran en operación. La búsqueda de archivos se realiza mediante claves que se empatan con los metadatos asociados a cada archivo. Al igual que la búsqueda, la operación de indexado se realiza de forma centralizada desde un sitio Napster. Por otro lado, el sistema establece las conexiones entre los clientes, pero la transferencia final de datos ocurre directamente entre estos.

Gnutella (http://gnutella.wego.com) es un protocolo descentralizado construido sobre una estructura autoconfigurable. Aquí no existe un estándar para efectuar una búsqueda. La consulta difunde una cadena de datos que cada receptor es libre de interpretar como le parezca. Por ejemplo, algún nodo puede tratar de empatar la consulta con el nombre de un archivo, otro con su contenido y un tercero puede procesar la consulta de cualquier otra forma. La consulta y otras operaciones de solicitud se propagan usando un protocolo de inundación que previene la formación de ciclos. La transferencia tiene lugar directamente entre las partes interesadas.

Mojo Nation (http://www.mojonation.net) es un sistema donde los archivos son fragmentados utilizando códigos resistentes al borrado y luego se dispersan entre varias de las máquinas participantes. El propósito no es lograr una permanencia de largo plazo, sino mejorar el ancho de banda y lograr un balance de datos. Cuando se consulta un archivo, los fragmentos se descargan desde cada uno de sus sitios de almacenamiento y el archivo se reconstruye. Con ello se consigue que aún los usuarios con limitaciones en ancho de banda sean útiles y contribuyan con sus recursos de manera eficaz. A esta tecnología se le conoce como distribución de enjambre.

Otro aspecto sobresaliente de Mojo Nation es un sistema de contabilidad que ofrece incentivos a sus contribuyentes. Cada interacción P2P en el sistema tiene un costo en créditos, cuantificados en un valor de cambio digital denominado *mojo*. Los mojos se reciben cuando se contribuye con espacio de almacenamiento, capacidad de red y CPU. Las cuentas del servicio se llevan con la ayuda de un tercero confiable. Este esquema de contabilidad también ayuda a conseguir un balance de carga de grano fino. Por ejemplo, si un servidor se encuentra sobrecargado, el sistema da preferencia a los usuarios "ricos" y con ello se alienta a los demás para que emigren hacia otros servidores con mayor disponibilidad.

Freenet es un sistema totalmente descentralizado. Cada archivo se identifica con un nombre clave que tiene la apariencia de una cadena aleatoria [46]. Los objetivos de Freenet son: resistencia a la censura y anonimato para los usuarios. Las peticiones y respuestas son reexpedidas a través de una cadena de nodos. Los archivos transmitidos se pueden almacenar temporalmente en los nodos de esta cadena. Los documentos se encriptan y

sus llaves no están disponibles para los operadores de los nodos. Esto sirve para que el operador del nodo se deslinde del contenido que almacena.

Publius es un sistema para publicación de contenidos. Consiste en un conjunto fijo de servidores que almacenan contenidos encriptados [47]. El autor del contenido crea una llave secreta para encriptar su documento. La llave secreta se divide en n pedazos tales que bastan k de ellos, y no menos, para recuperarla. Los servidores se eligen de forma pseudoaleatoria para almacenar copias del documento encriptado y un pedazo de la llave.

La comunicación autor-servidor se soporta utilizando un canal anónimo. Se genera un URL que se mapea a un conjunto de (al menos k) servidores. Por otra parte, los documentos son consultados usando un navegador Web estándar. Un proxy local reune los fragmentos de la llave y una copia del documento encriptado, luego reconstruye la llave y desencripta el documento. El sistema ofrece también mecanismos para que el autor pueda actualizar o borrar documentos.

Free Haven es un sistema desarrollado por alumnos del MIT y Harvard, ofrece anonimato, resistencia a la censura y garantiza una disponibilidad del documento por un tiempo de vida que el propio autor especifica [48]. Este último atributo lo hace diferente de otros sistemas de su tipo, aunque se considera que puede tener algunos problemas de eficiencia.

2.7 La contribución al estado del conocimiento

En resumen, la propuesta de esta tesis consiste en fijar un marco teórico en el que pueden usarse tres formas de redundancia para construir un DDES que tolere fallas de paro: la redundancia en tiempo consiste en la restauración del sistema hasta un estado global previo, que se almacena en forma distribuida; la redundancia de los recursos físicos consiste en organizar a los componentes para que cada uno funcione como un respaldo de los demás, respetando criterios de equidad en la distribución de esta tarea; por último, es posible utilizar técnicas de redundancia de información para procesar un estado global antes de su almacenamiento. Con ello podría recuperarse esta información aún en presencia de errores de integridad.

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 servir como base para la construcción de servicios de almacenamiento distribuido de datos como los sistemas Past, Farsite e Intermemory.

El almacenamiento distribuido de la información llegará a convertirse en un procedimiento clave para los sistemas de cómputo, en la medida que garantice la continuidad de las operaciones de las mismas tecnologías y sirva como soporte para muchos de los servicios que están desarrollándose.

Capítulo 3

Esquemas de almacenamiento

En condiciones reales, los sitios activos de una ejecución distribuida pueden experimentar fallas de paro. Si se presentara una contingencia de este tipo, el sistema podría restaurarse desde el estado global más reciente del que se tuviera registro, siempre que se guardaran instantáneas de la ejecución [49]. Este capítulo, basado en [1], propone el uso de técnicas de almacenamiento distribuido para guardar los estados globales de una ejecución distribuida. Su contribución principal consiste en el desarrollo de varias soluciones de esta clase y su análisis a partir de las medidas críticas que se establecen para evaluar el balance de carga y la tolerancia a fallas.

Erick Lönnrot las estudió. Los tres lugares, en efecto, eran equidistantes. Simetría en el tiempo, simetría en el espacio, también . Sintió, de pronto, que estaba por descifrar el misterio.

J.L. Borges en "La muerte y la brújula"

3.1 Introducción

En su forma más sencilla, el almacenamiento distribuido consiste en el resguardo de una colección de archivos sobre los sitios de una red. Naturalmente, cada aplicación impone requerimientos especiales sobre este procedimiento.

El registro regular de los estados globales de una ejecución distribuida [50, 28] introduce un problema de gestión sobre los recursos seleccionados para este fin. En este capítulo se abordan los aspectos del desempeño de las soluciones balanceadas, es decir, aquellas que comprometen los recursos de la ejecución de una forma limitada y equitativa. Visto de otra forma, se busca que la responsabilidad de almacenamiento sea repartida por igual entre los mismos equipos a cargo de la tarea principal del sistema que se busca proteger. Existe un compromiso entre los sitios que fungen como bodegas y la cantidad de información resguardada en un solo sitio durante una operación de almacenamiento. Las soluciones que van a considerarse deben poseer las siguientes características:

- Cada vez que se invoque el procedimiento de almacenamiento, los componentes locales del estado global deben depositarse en un subconjunto de los sitios, denominado *bloque*.

- El conjunto de los bloques que van a usarse se define estáticamente al inicio de las operaciones. Los bloques se calendarizan de manera cíclica (uno por cada paso de almacenamiento) en un orden predefinido.
- Cada sitio debe participar en el menor número bloques que sea posible, pero también debe aparecer en tantos como cualquier otro sitio.
- Los bloques deben tener tantos sitios como sea posible para reducir la cantidad de espacio que se requiere de cada sitio individual.

Contribución

Un *esquema de almacenamiento* consiste de un conjunto ordenado de bloques como los que se definieron en el párrafo anterior. Es una solución distribuida por cuanto no requiere de un sitio central para coordinar el procedimiento. Esto se debe a que la lista de bloques se define a priori y, con ello, cada sitio puede utilizar una tabla para determinar, por medios locales, las bodegas designadas para cada paso de almacenamiento. En todos los esquemas de almacenamiento que van a desarrollarse, esta tabla explícita puede reemplazarse si cada sitio cuenta con una forma eficiente de calcular el calendario cíclico con que se designan los bloques.

En la sección 3.2 se presenta el modelo formal que subyace en todos los esquemas de almacenamiento. Se introducen también los parámetros que caracterizan a cada solución: el *período de almacenamiento* mide la cantidad total de espacio requerido por una solución, también mide el tiempo de reciclado de cualquier buffer participante; el *número crítico de fallas* mide la fortaleza de una esquema para tolerar fallas de paro. Por último, se define el concepto de estrategia *balanceada e incompleta*.

En la sección 3.3 se desarrolla el primer esquema denominado "selección de k de v", que puede considerarse como la solución obvia y elemental. Consiste en construir todos los subconjuntos de k elementos tomados de un conjunto de v sitios. Cada subconjunto corresponde con un bloque de almacenamiento. En esta sección, y las siguientes dos, el análisis se enfoca al estudio de los parámetros que caracterizan a los esquemas presentados. Por lo general, la dificultad surge cuando se trata de calcular el número crítico de fallas. En la sección 3.3, particularmente, los resultados se obtienen a partir de argumentos de conteo: dado un conjunto de v elementos, ¿cuántos de estos deben eliminarse para que sea imposible construir al menos un subconjunto de k elementos?

En la sección 3.4 se desarrolla el segundo esquema denominado "ranura deslizante". En este caso puede pensarse que los sitios están dispuestos alrededor de una mesa circular, se tienen un arco que puede girar alrededor de la circunferencia y abarca k plazas, de forma que los sitios bajo el arco quedan seleccionados dentro de un bloque de almacenamiento. El análisis y los resultados se basan en conceptos sobre grupos cíclicos finitos. Nuevamente, el reto aquí consiste en determinar el número crítico de fallas (véase [51] pp.62).

En la sección 3.5 se desarrolla el tercero y último esquema denominado "geometría finita". En este caso, cada bloque se conceptualiza como un conjunto de k puntos que definen una línea de una geometría finita, ya sea el plano proyectivo o el plano afín. Para determinar el

número de fallas que pueden soportarse. Se utilizan resultados conocidos acerca de *conjuntos de bloqueo* sobre diseños combinatorios,

La sección 3.6 incluye la comparación del desempeño entre los diferentes esquemas presentados. Se muestra que cualquier pareja de esquemas puede compararse sobre la base de dos parámetros: precio y tolerancia. El precio es un parámetro secundario, derivado del período de almacenamiento y el tamaño de bloque. En tanto, la tolerancia es una medida directa del número crítico de fallas. Cada una de las soluciones consideradas representa un compromiso entre estas dos figuras de mérito. La última parte de la sección muestra cómo combinando dos esquemas diferentes puede encontrarse una solución mejorada que pueda satisfacer algunos requerimientos particulares de almacenamiento.

Finalmente, la sección 3.7 contiene las conclusiones y un plan de trabajo futuro, incluyendo temas como calendarización y nuevos enfoques para mejorar la tolerancia a fallas.

Trabajo relacionado

El almacenamiento distribuido de la información se convertirá en una operación clave en muchos sistemas de cómputo y telecomunicaciones, ya que dependerán cada vez más de esta clase de soluciones para el soporte de sus servicios, así como de sus operaciones internas. El proyecto OceanStore [44] representa el ejemplo más contundente de las aplicaciones que pueden soportarse con esta nueva tecnología.

Se sabe de la fértil relación entre el álgebra y la teoría de la información [9, 10, 11] que sigue produciendo un vasto conjunto de soluciones con aplicaciones tecnológicas inmediatas [52]. Por lo que toca al presente trabajo, deben destacarse los códigos resistentes al borrado [53] y su utilización en los llamados arreglos redundantes de discos independientes o sistemas RAID [54] (Redundant Arrays of Independent Disks). No obstante, esta es una solución centralizada mientras que aquí se desarrolla un enfoque distribuido para guardar información. En este sentido, nuestra propuesta se halla más relacionada con los denominados arreglos redundantes de discos distribuidos (RADD) [6, 55], pero su originalidad se encuentra en que formaliza el concepto de esquema de almacenamiento que busca poner en relieve el balance en la distribución de las tareas de almacenamiento. Por otro lado, existen varios trabajos basados en diseños combinatorios para estudiar cooperación bajo restricciones o comunicación limitada. Puede consultarse [56] también, para un estudio sobre las aplicaciones de los diseños y, en general, sobre las técnicas para el balance de carga.

La dispersión de información [14] es un enfoque alternativo para implantar almacenamiento tolerante a fallas, aunque su costo en comunicaciones puede resultar caro, ya que la reconstrucción de un archivo implica varios accesos remotos. Sin embargo, puede ofrecer ventajas para el almacenamiento de copias de respaldo.

Existe una estrecha relación entre los denominados sistemas quorum y los esquemas de almacenamiento: ambos son colecciones de subconjuntos de sitios donde se guarda información [57]. El balance de carga es un aspecto muy importante en ambos casos. Por otro lado, en contraste con los sistemas quorum, la consistencia de la información no es un aspecto crítico de los esquemas de almacenamiento, mientras que la capacidad y la recuperación de fallas son los problemas más importantes. Esto significa que en los esquemas de almacenamiento no se requiere establecer una cadena de causalidad entre cualesquiera dos operaciones de al-

macenamiento y que la propiedad de intersección entre bloques puede relajarse. Cuando se trata de guardar volúmenes masivos de información, existen otras medidas que cuidar.

Adicionalmente, merece atención el concepto de jerarquía revolvente [58] desarrollado para conseguir una distribución equitativa de tareas sobre un ambiente distribuido. Al igual que los esquemas que aquí se presentan, puede entenderse como permutaciones sobre los roles desempeñados por los sitios de un sistema, a lo largo de una ejecución distribuida.

3.2 Estrategias de selección y esquemas de almacenamiento

Sea V un conjunto con v elementos. Una *estrategia de selección* $\mathcal{B}$ sobre V es una colección de b subconjuntos de V denominados *bloques*, donde

- $k_j = |B_j|$, $j = 1, \ldots, b$, es la cardinalidad del j-ésimo bloque, y
- $r_i = \sum_{j=1}^{b} |\{v_i\} \cap B_j|$, $i = 1, \ldots, v$, es el número de bloques en los que el elemento v_i aparece.

Para una estrategia $\mathcal{B}$, su matriz de incidencia $\mathbf{A} = (a_{ji})$ de tamaño $b \times v$ se define como

$$a_{ji} = \begin{cases} 1 & \text{si } v_i \text{ aparece en el bloque } j, \\ 0 & \text{en otro caso.} \end{cases}$$

Evidentemente, el renglón j corresponde con el bloque B_j. Alternativamente, la columna i muestra los bloques donde el elemento v_i participa.

Se dice que una estrategia es *balanceada* cuando todos sus bloques son del mismo tamaño y cualquier elemento de V aparece en un número fijo de bloques. De aquí se infiere que

$$k_j = k,\ j = 1, \ldots, b \quad \text{y}$$
$$r_i = r,\ i = 1, \ldots, v.$$

Para cualquier matriz de incidencias puede contarse el total de 1's que la componen de dos formas diferentes: por renglones ó por columnas. De lo anterior se concluye que, en una estrategia de este tipo

$$bk = vr. \tag{3.1}$$

Se dice que una estrategia es *incompleta* cuando ningún elemento de V aparece en todos los bloques, esto es

$$r_i < b,\ i = 1, \ldots, v.$$

Dada una estrategia $\mathcal{B}$ sobre un conjunto V, un *subconjunto generador* es un subconjunto V' de V tal que para cada bloque $B_j \in \mathcal{B}$, $V' \cap B_j \neq \emptyset$. Un *subconjunto generador mínimo* V'_{min}, de cardinalidad $f = |V'_{min}|$, es un subconjunto generador tal que, para cualquier subconjunto generador V', $f \leq |V'|$.

Un *esquema de almacenamiento* es un proceso distribuido que se ejecuta a cargo de los sitios conectados a una red y se desarrolla por *pasos*. Cada sitio administra su propio disco local. En cada paso, se designa un subconjunto de sitios para fungir como *bodegas*, y todos los participantes saben cuáles son los sitios con esta responsabilidad. Cada uno de los subconjuntos designados se programa para aparecer en un paso diferente y se recalendarizan de forma cíclica, una vez que todos han sido seleccionados.

Un paso comienza cuando cada sitio toma una instantánea de su estado local, denominada *fragmento*. Enseguida determina, de acuerdo con su esquema de almacenamiento, la bodega apropiada donde debe guardar esta información. Todos los fragmentos depositados en la misma bodega, durante un paso, definen un *contenido*. Por cada paso en el que participa como bodega, un sitio debe preparar el espacio de almacenamiento y una lista de los otros sitios a los que presta servicio. Se supone que cada participante tiene un número único $i \in (0, v-1)$ que sirve para identificarle. Durante la ejecución, los sitios quedan expuestos al riesgo de falla de paro. Supondremos que si un sitio cayera en esta condición, quedaría permanentemente fuera de operación [59, 32].

La evaluación de un esquema consiste en determinar dos parámetros básicos que le caracterizan: su *período de almacenamiento* y su *número crítico de fallas*. El período de almacenamiento se define como el mínimo número de pasos que deben transcurrir antes de poder reciclar un contenido. En tanto, el número crítico de fallas es el mínimo de sitios cuya falla haría imposible la recuperación de al menos un estado global integro, forzando a restaurar el sistema desde el instante cero.

Una vez que ha trabajado como bodega, un sitio debe limpiar y reutilizar el mismo espacio de almacenamiento que llenó en el ciclo anterior. En esta circunstancia podría ocurrir que una bodega vaciara su espacio sin alcanzar a reutilizarlo, por ejemplo, cuando queda pendiente la recepción del fragmento desde un sitio que cae en paro. Entonces, al menos una vez durante cada ciclo se debe echar a andar un procedimiento de verificación para comprobar la integridad de los contenidos almacenados, de otra forma, el registro de todos los estados globales podría perderse y con ello desaparecería la posibilidad de restaurar la ejecución en algún punto de su pasado reciente.

Un esquema de almacenamiento puede construirse a partir de una estrategia de selección. Sea V el conjunto de los sitios disponibles. Para cada paso de almacenamiento, el conjunto de sitios seleccionados puede obtenerse de la estrategia $\mathcal{B}$ sobre V. Un esquema de almacenamiento basado en una estrategia incompleta consigue un punto de equilibrio entre los recursos que cada sitio dedica como bodega y los que dedica para su tarea principal. Esto es, se garantiza que el sitio no trabaja como bodega durante todos los pasos de almacenamiento que dura un ciclo y, con ello, contribuye a la ejecución principal. Por otro lado, un esquema de almacenamiento basado en una estrategia balanceada distribuye uniformemente la tarea de almacenamiento entre todos los sitios disponibles. El balance garantiza que un sitio funge como bodega tantas veces como cualquier otro.

Supongamos ahora que una estrategia balanceada e incompleta (estrategia buena, para abreviar) se encuentra bajo el control de un calendario cíclico que designa a cada bloque en un orden fijo, antes de reprogramar su participación. Esto significa que el período de almacenamiento será igual al máximo número de bloques b que pueden usarse antes de repetir

alguno. Por último, basta con eliminar a los f elementos de cualquier subconjunto generador mínimo, para destruir a todos los bloques de $\mathcal{B}$.

Adicionalmente, algunas medidas pueden derivarse de los parámetros básicos para ofrecer una comprensión más profunda sobre el desempeño de un esquema: el *número de contenidos* mide los diferentes contenidos almacenados en un solo sitio, igualmente cuenta el número de pasos durante los que un sitio participa como bodega; el *cuello de botella* es la cantidad de fragmentos almacenados en cualquier bodega durante un solo paso de almacenamiento o, equivalentemente, el tamaño de un contenido. Ahora, para una estrategia buena, el número de contenidos r se sigue de la ec.(3.1). Al mismo tiempo, puesto que hay v fragmentos diferentes que deben almacenarse durante cada paso, y se suponen repartidos uniformemente entre las k bodegas disponibles, el cuello de botella η será $\frac{v}{k}$.

En la sección 3.6 se demostrará que, para una estrategia buena, varios de sus parámetros dependen del tamaño k de sus bloques. Un valor pequeño significa que cada sitio conserva un número reducido de contenidos pero, durante los pocos pasos en que trabaja como bodega, corre el riesgo de verse desbordado por el total de fragmentos η que recibe al mismo tiempo. En el caso contrario, cuando un bloque es grande, cada componente tiene que posponer su tarea principal para trabajar como bodega en la mayor parte de los pasos que dura un ciclo. Claro que, en cada ocasión, sólo maneja un número reducido de peticiones. Al parecer, existen dos medidas en conflicto que contribuyen con el *precio total* del esquema:

$$P = r + \eta = \frac{b}{v}k + \frac{v}{k}. \tag{3.2}$$

Por tanto, puede conseguirse un balance a condición de minimizar P, es decir, siempre que exista un valor de $k \in [1, v)$ que reduzca el precio total. El mejor esquema será aquel basado en una estrategia buena con el menor número de bloques, que exhiba un precio total mínimo, pero no sacrifique la robustez que se exige a esta solución de almacenamiento.

En síntesis, la calidad de las soluciones que van a considerarse queda determinada por dos medidas que son el precio total y el número crítico de fallas. Evidentemente, el precio tiene una expresión analítica que no depende de la forma en que se construye la estrategia de selección subyacente, sino que es una propiedad de las estrategias balanceadas. Algunas de ellas permiten manipular el valor de k y con ello puede optimizarse el precio. Por otro lado, el número crítico de fallas depende totalmente de la estrategia de selección, lo cual explica porqué la búsqueda de un método de construcción que ofrezca un tamaño de bloque cercano al óptimo y, al mismo tiempo, maximice el número de fallas que pueden tolerarse.

En cualquier esquema de almacenamiento es importante disponer de un procedimiento para determinar, con economía de recursos y de forma distribuida, el bloque que debe designarse en cada paso de almacenamiento. De otra forma, en todos los sitios tendría que registrarse la lista completa de bloques que componen a la estrategia subyacente. Todas las soluciones consideradas en este trabajo poseen esta importante propiedad que sustenta un procedimiento denominado *algoritmo de orden revolvente*, para determinar con el mínimo esfuerzo, la composición del siguiente bloque a cargo.

Para concluir esta sección es importante recalcar que, cualquier procedimiento de recuperación que se invocara en caso de falla, incluiría diferentes módulos o componentes, entre los que se hallaría un esquema de almacenamiento. Algunas otras partes serían: un módulo

de detección de fallas y un módulo de balance y redistribución de carga. También se usaría, al inicio de sus operaciones, un componente para repartir uniformemente el total de fragmentos entre las bodegas de cada bloque, de manera que se consiguiera optimizar la distancia entre la fuente y el sitio donde ésta guarda su información. Asumiendo que esta repartición es óptima en alguna medida es que se usa el término "mejor" sitio, a lo largo de los algoritmos presentados en las siguientes secciones.

3.3 Selección de k de v

```
< 1> Condiciones iniciales
< 2>    j = 0;
< 3>    M = v! / (k!(v-k)!);
< 4>    sea c = (c_1, c_2, ..., c_{k+1}) un vector,
        tal que c_i ∈ [1...v], 1 ≤ i ≤ k;
< 5>    sea c_{k+1} = v + 1, una coordenada constante de c
< 6> En el j-ésimo paso de almacenamiento
< 7>    l = 1;
< 8>    mientras (l ≤ k) y (c_l == l)
< 9>       l = l + 1;
<10>    si (k ≠ l mod 2)
<11>       si (j == 1);
<12>          c_1 = c_1 - 1;
<13>       otro
<14>          c_{l-1} = l;
<15>          c_{l-2} = l - 1;
<16>    otro
<17>       si (c_{l+1} ≠ c_l + 1);
<18>          c_{l-1} = c_l;
<19>          c_l = c_l + 1;
<20>       otro
<21>          c_{l+1} = c_l;
<22>          c_l = l;
<23>    sea B = {c_i ∈ c};
<24>    sea i - 1 el mejor sitio tal que i ∈ B;
<25>    envía fragmento[j] a i - 1;
<26>    j = (j + 1) mod M;
```

Figura 3.1: ¿Qué sitios serán las bodegas, si se "seleccionan k de v"?

El primer esquema de almacenamiento de este trabajo se denomina "selección de k de v" (kv) y se basa en la estrategia más simple: designa como bloques a todos los subconjuntos de tamaño k de un conjunto V con v elementos.

Lema 3.1 *Para valores fijos de v y k, los parámetros básicos de un esquema "selección de k de v" son los siguientes:*

$$b = \left(\begin{array}{c} v \\ k \end{array} \right),$$
$$f = v - k + 1.$$

Claramente, el número de bloques corresponde con el coeficiente binomial $\frac{v!}{k!(v-k)!}$. En cuanto al número crítico de fallas f, supóngase que se eliminan $v-k$ elementos de V. Entonces, con los k elementos que restan se puede construir aún un bloque en $\mathcal{B}$. Sin embargo, si se descontaran $v-k+1$ elementos, no sería posible.

Para la estrategia en que se basa este esquema, el número de bloques depende de k. Por otro lado, de la ec.(3.2) se desprende que el esquema consigue el óptimo de su precio total óptimo cuando $k = v$ y también para un valor no entero tal que $1 < k < 2$. En el primer caso, la estrategia subyacente no es incompleta, lo que significa que es una solución que requiere de la participación de cada sitio en todo paso de almacenamiento. En el segundo caso, el cuello de botella que se presenta es el mayor inconveniente.

La colección de bloques de este esquema se genera utilizando el procedimiento distribuido que se observa en la fig. 3.1. Se trata de un algoritmo que ordena a todos los bloques de forma tal que dos bloques consecutivos cualesquiera difieren en un número mínimo de elementos. También se le conoce como el algoritmo de la puerta giratoria (revolving door ordering algorithm) [60]. Se trata de un algoritmo que produce todos los bloques del esquema utilizando un mínimo de recursos (i.e. espacio y tiempo), ya que sólo requiere del último bloque y manipula dos coordenadas del vector que lo representa para determinar el siguiente bloque. Una vez que se conoce el siguiente k-subconjunto, cada sitio elige de éste a la "mejor" bodega donde guardará su nuevo fragmento.

3.4 Ranura deslizante

El segundo esquema de almacenamiento de este trabajo se denomina "ranura deslizante" (ss). Puede explicarse usando una analogía: supóngase una mesa circular alrededor de la cual se sientan v personas; se seleccionan k de ellas en posiciones consecutivas, para formar un bloque. Esta selección define una *ranura* que puede entenderse también como un arco de circunferencia que abarca a k personas. Enseguida, el arco gira γ *posiciones*, a la derecha por ejemplo, y las k personas que quedan bajo el arco en su nueva posición definen el siguiente bloque designado. De igual forma, el resto de los bloques se establecen mediante rotaciones sucesivas de la ranura.

Un sitio que participó en el último bloque, pero que no aparece en la siguiente selección, se dice que *abandona* la ranura. En cambio, un sitio que reaparece, al cabo de un ciclo, se dice que *entra* en la ranura.

Cada bloque generado de esta forma puede conceptualizarse como una v-tupla de la forma $(c_0, c_1, \ldots, c_{v-1})$, donde la i-ésima componente puede tomar un valor 1 ó 0, según el sitio i participe ó no en dicho bloque. Los bloques pueden determinarse de acuerdo con la siguiente regla:

```
< 1> Condiciones iniciales
< 2>      j = 0;
< 3>      d = mcd(v, γ);
< 4>      b = v/d;
< 5>      sea c_0 = (c_0, c_1, ..., c_{v-1}) un vector binario,
          tal que c_0 = ... = c_{k-1} = 1, y c_k = ... = c_{v-1} = 0;
< 6> En el j-ésimo paso de almacenamiento
< 7>      sea c_j = [c_0, c_1, ., c_{v-1}] la la j-ésima ranura, según la regla ss;
< 8>      sea B = {i|c_i ∈ c_j & c_i == 1};
< 9>      sea i el mejor sitio tal que i ∈ B;
<10>      envía el fragmento[j] a i;
<11>      j = (j + 1) mod b;
```

Figura 3.2: ¿Qué sitios serán las bodegas, si se usa "ranura deslizante"?

regla ss Si $\mathbf{c} = (c_0, c_1, \ldots, c_{v-1})$, donde $c_i \in F_2$ para $i = 0, 1, \ldots, v-1$, representa una ranura, entonces cada una de ellas tiene un polinomio asociado de la forma $c_0 + c_1 x + \ldots + c_{v-1} x^{v-1}$ en el anillo $F_2[x]/ < x^v + 1 >$ de los polinomios sobre F_2 módulo $x^n + 1$ tal que, si $\mathbf{c}_j$, $\mathbf{c}_{j+1}$ son dos ranuras consecutivas y $p_j(x)$, $p_{j+1}(x)$ sus polinomios respectivos $F_2[x]/ < x^v + 1 >$, entonces $p_2(x) = x^\gamma p_1(x)$. Para este esquema existe una ranura $\mathbf{c}_0 = (c_0, c_1, \ldots, c_{v-1})$, tal que $c_0 = c_1 = \ldots = c_{k-1} = 1$, $c_k = c_{k+1} \ldots = c_{v-1} = 0$ y todas las ranuras sucesivas pueden derivarse de ésta, siguiendo el procedimiento mencionado.

Claramente, para una ranura inicial $\mathbf{c}_0$, el conjunto de las rotaciones de $\mathbf{c}_0$ forma un grupo cíclico G de orden v tal que $G =< 1 >$. Esto significa que la rotación simple, o de una sola posición, genera al grupo completo. Además, G es isomorfo a Z_v. Cualquier otro elemento del grupo, por ejemplo γ, generara un subgrupo cíclico H, tal que $|H| = T = v/d$, donde $d = \text{mcd}(v, \gamma)$. A partir del algoritmo de Euclides puede interpretarse que el conjunto de las ranuras generadas aplicando rotaciones de γ posiciones sobre $\mathbf{c}_0$, sería el mismo conjunto que el de las ranuras generadas aplicando rotaciones de d posiciones sobre $\mathbf{c}_0$.

Supónganse, por ejemplo, valores fijos para v y γ, tales que $\text{mcd}(v, \gamma) = 1$. Entonces, los parámetros básicos de este esquema serían: $b = v$, $f = \lceil \frac{v}{k} \rceil$. Dada una ranura inicial $\mathbf{c}_0$ que se deslizara una posición en cada paso, tomaría v pasos para volver a la posición original. Ahora, si se cubriera a la circunferencia con el menor número de ranuras, estas deberían ser ranuras disjuntas (hasta donde fuera posible) y serían $\lceil \frac{v}{k} \rceil$. Si se eliminara el sitio de la primer posición en cada una de ellas, no existirían, entre las posiciones restantes, k sitios consecutivos donde pudiera acomodarse una ranura completa.

Para los esquemas de este tipo, se puede observar que el total de bloques $b = v$ es un número pequeño y no depende del tamaño de k. De acuerdo con la ec.(3.2), un esquema de ranura deslizante alcanza el óptimo de su precio total $P = 2\sqrt{v}$, cuando $k = \sqrt{v}$. De hecho, para algunos valores de v y $\gamma > 1$ es posible conseguir un precio inferior, pero estas soluciones ofrecerán una tolerancia menor a la del caso en que $\gamma = 1$.

Lema 3.2 *Para valores fijos de v, γ y k, tales que $d = \text{mcd}(v, \gamma)$, los parámetros básicos de*

un esquema de "ranura deslizante" son los siguientes:

$$b = \frac{v}{d},$$
$$f = \begin{cases} \lceil \frac{v}{d\lceil \frac{k}{d} \rceil} \rceil & si\ d \leq k, \\ \frac{v}{d} & en\ otro\ caso. \end{cases}$$

Si $d \leq k$, un segmento de longitud d cabe $\lceil \frac{k}{d} \rceil$ veces en un segmento de longitud k. Lo cual significa que entre el punto c_0, en la posición 1 de una ranura fija $\mathbf{c}$, y el punto $c_{k'}$, en la posición 1 de la ranura disjunta más próxima a $\mathbf{c}$, hay una distancia $k' = d\lceil \frac{k}{d} \rceil$. De aquí se infiere que el máximo número de ranuras disjuntas que se requieren para cubrir una circunferencia de longitud v es $f = \lceil \frac{v}{k'} \rceil$. Bastará con eliminar la primer posición de cada una de dichas ranuras para destruir todos los bloques de la estrategia subyacente. Ahora, supóngase que es posible eliminar $f' < f$ posiciones para afectar a toda la estrategia, por ejemplo eliminando uno de cada $k' + 1$ puntos, comenzando por c_0. Equivalentemente, esto significa que la circunferencia quedaría dividida en $\lceil \frac{v}{k'+1} \rceil$ segmentos disjuntos, cada uno de longitud $k' + 1$. Sin embargo, si existen al menos d segmentos consecutivos de esta longitud, habrá espacio suficiente para acomodar un bloque íntegro. Por otro lado, si se requiere que $f' < f$, habrá al menos k' segmentos de longitud $k' + 1$. Puesto que $k' > d$, habrá al menos un bloque sin destruir(!).

Si $d > k$, todas las ranuras serán disjuntas y será suficiente con tomar una posición de cada una para construir un subconjunto generador mínimo, i.e. $f = \frac{v}{d}$

La colección de bloques de este esquema se genera utilizando el procedimiento distribuido que se observa en la fig. 3.2. En resumen, se trata de un procedimiento distribuido que funciona de la siguiente manera: cuando se convoca un nuevo paso de almacenamiento, cada sitio genera por sí solo la ranura próxima. Entonces, una vez que se conoce la siguiente tupla, el sitio elige de ésta a la "mejor" bodega donde guardará su nuevo fragmento.

3.5 Geometrías finitas

El tercer esquema de este trabajo se denomina "geometrías finitas". Comprende dos estrategias relacionadas: "plano proyectivo finito" (pp) y "plano afín finito" (ap). Esta familia de soluciones tiene una motivación geométrica: los sitios son considerados como puntos y los bloques definen líneas que heredan propiedades particulares de la geometría a la que pertenecen. Existe una sólida teoría acerca de esta clase de objetos matemáticos [9, 10, 11].

Sean v, k, λ enteros positivos tales que $v > k > 2$. Un (v, k, λ)-*diseño de bloques balanceado e incompleto* o (v, k, λ)-BIBD es una pareja $(V, \mathcal{B})$ que satisface las siguientes propiedades:

1. V es un conjunto de v elementos denominados *puntos*.

2. $\mathcal{B}$ es una colección de b subconjuntos de V denominados *líneas* o *bloques*.

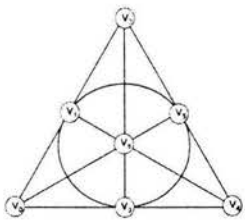

Figura 3.3: Plano proyectivo de orden 2

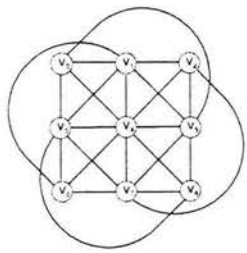

Figura 3.4: Plano afín de orden 3

3. Cada bloque está formado por k puntos.

4. Cada pareja de puntos distintos está contenida en λ bloques exactamente.

Un *plano proyectivo de orden* q es un $(q^2+q+1, q+1, 1)$-BIBD. Cualquier plano proyectivo es también un BIBD simétrico, lo que significa que $b = v$ y que cualquier par de bloques se intersecta en un punto único. Se sabe que un plano proyectivo de orden q existe si y sólo si q es potencia de un primo. La fig. 3.3 muestra un plano proyectivo de orden 2.

Se puede considerar una definición alternativa (y equivalente) [61, 62, 63]. Un plano proyectivo es una colección de puntos y líneas que satisfacen tres propiedades: (i) existe una línea única que contiene a cualesquiera dos puntos, (ii) cualesquiera dos líneas se intersectan en un punto único y (iii) existen cuatro puntos de los cuales, tres de ellos no son colineales. De las definiciones anteriores se obtienen los siguientes resultados:

Lema 3.3 *Para un valor fijo de* v, *tal que* $v = q^2 + q + 1$ *y* q *potencia de un número primo, los parámetros básicos de un esquema de "plano proyectivo" son los siguientes:*

$$b = q^2 + q + 1,$$
$$f = q + 1.$$

El primer parámetro es una propiedad inmediata del plano proyectivo. En cuanto al segundo parámetro, en un plano de orden q, un subconjunto generador puede contener o no a una línea. Si la contiene, entonces, puesto que todas las líneas se intersectan, los puntos de una línea B_1 producen un subconjunto generador mínimo con $q + 1$ puntos. Se sabe también que, para el mismo plano de orden q, un subconjunto de puntos que no contenga a una línea completa y que tenga una intersección no vacía con todas las líneas de la geometría deberá tener al menos $q + \sqrt{q} + 1$ puntos [61, 64]. Por tanto, el subconjunto generador mínimo contiene $q + 1$ elementos o, equivalentemente, comprende a una línea con $k = q + 1$ puntos.

Al igual que los esquemas de ranura deslizante, un plano proyectivo ofrece un número de bloques pequeño, $b = v$, con una estrategia balanceada e incompleta. Por otro lado, su

q	$D \subseteq \mathcal{Z}_{q^2+q+1}$
2	$\{1, 2, 4\}$
3	$\{1, 2, 6, 12\}$
4	$\{1, 2, 5, 15, 17\}$
5	$\{1, 2, 16, 20, 22, 25\}$

Tabla 3.1: Algunos conjuntos de diferencias.

tamaño de bloque es fijo y no puede manipularse. Sin embargo, este valor $k = q+1 = O(\sqrt{v})$ produce un precio total(3.2) cercano al óptimo.

Sea $(G, +)$ un grupo Abeliano (aditivo) finito de orden v, cuya identidad se denota por el símbolo 0. (p.e. $G = \mathcal{Z}_v$, los enteros módulo v). Un (v, k, λ)-*conjunto de diferencias* en G es un subconjunto $D \subseteq G$, que satisface las siguientes propiedades:

1. $|D| = k$,
2. La colección de valores $\{x - y : x, y \in D, x \neq y\}$ contiene a cada elemento de $G \backslash \{0\}$ exactamente λ veces.

Los conjuntos de diferencias pueden usarse para construir BIBDs simétricos. Sea D un (v, k, λ)-conjunto de diferencias en un grupo Abeliano $(G, +)$. Para todo $g \in G$ se define al conjunto

$$D + g = \{x + g : x \in D\},$$

al que se denomina *traducción* de D. Enseguida se define a

$$\mathrm{Dev}(D) = \{D + g : g \in G\},$$

que es el conjunto de todas las traducciones de D y se denomina *desarrollo* de D. Se sabe [11] que $(G, \mathrm{Dev}(D))$ es un (v, k, λ)-BIBD simétrico y que existe un $(q^2 + q + 1, q + 1, 1)$-conjunto de diferencias en $\mathcal{Z}_{q^2+q+1}$ para todo q potencia de número primo. Este resultado será fundamental para la construcción del procedimiento distribuido de coordinación asociado con los esquemas de plano proyectivo:

La tabla 3.1 muestra los conjuntos de diferencias para cuatro posibles valores de v.

La colección de bloques de este esquema se genera utilizando el procedimiento distribuido que se observa en la fig. 3.5, el cual funciona de la siguiente manera: comenzando por su conjunto de diferencias correspondiente, cada sitio genera por sí solo la siguiente línea de la geometría subyacente. Entonces el sitio elige de ésta a la "mejor" bodega donde guardará su nuevo fragmento.

Un *plano afín de orden* q es un $(q^2, q, 1)$-BIBD que contiene $b = q^2 + q$ bloques de tamaño $k = q$. Se sabe que un plano afín de orden q existe si y sólo si q es potencia de un primo. La fig. 3.4 muestra un plano afín de orden 3.

Supóngase que $(V, \mathcal{B})$ es un (v, k, λ)-BIBD. Una *clase paralela* en $(V, \mathcal{B})$ es un subconjunto de bloques disjuntos de $\mathcal{B}$ cuya unión es V. Debe observarse que una clase paralela contiene

```
< 1> Condiciones iniciales
< 2>      j = 0;
< 3>      sea D el conjunto de diferencias del pp de orden q;
< 4> En el j-ésimo paso de almacenamiento
< 5>      sea B = {x + j mod v|x ∈ D};
< 6>      sea i el mejor sitio tal que i ∈ B;
< 7>      envía el fragmento[j] to i;
< 8>      j = (j + 1) mod v;
```

Figura 3.5: ¿Qué sitios serán las bodegas, si se usa "plano proyectivo"?

$\frac{v}{k} = \frac{b}{r}$ y que un BIBD puede tener una clase paralela sólo si $v \equiv 0 \bmod k$. Una partición de $\mathcal{B}$ en una clase paralela se denomina *resolución* y se dice que $(V, \mathcal{B})$ es *resoluble* si $\mathcal{B}$ tiene al menos una resolución. Se sabe que cualquier plano afín es resoluble.

Se puede considerar una definición alternativa (y equivalente) [61, 62, 63]. Un plano afín o plano Euclideano es una colección de puntos y líneas que satisfacen tres propiedades: (i) existe una línea única que contiene a cualesquiera dos puntos, (ii) dada una línea L y un punto $p \notin L$, existe una línea única que pasa por p y que no se intersecta con L, (iii) existen tres puntos que no son colineales. Es muy importante destacar que un plano afín puede generarse a partir de un plano proyectivo del mismo orden borrando de él todos los puntos de una línea fija. Para este segundo tipo de geometría finita se pueden inferir los siguientes resultados:

Lema 3.4 *Para un valor fijo de v, tal que $v = q^2$ y q potencia de un número primo, los parámetros básicos de un esquema de "plano afín" son los siguientes:*

$$b = q^2 + q,$$
$$f = 2q - 1.$$

El primer parámetro es una propiedad del mismo plano afín. Por lo que se refiere al segundo parámetro, se sabe que para un plano afín de orden q, el mínimo conjunto de puntos que tiene una intersección no vacía con cada línea tendrá $2q - 1$ elementos [61, 65]. Ya sea que contenga una línea o no, se puede construir un subconjunto generador mínimo de este tamaño. Un ejemplo de subconjunto generador conteniendo una línea completa sería establecer una resolución del plano afín, tomar todos los puntos de una línea de dicha resolución y un punto de cada una de las otras líneas de la misma clase paralela. Un ejemplo de subconjunto generador que no contiene una línea sería el siguiente [61], considérense los puntos v_0, v_1, v_2, v_3 y las líneas:

- B_1 que incluye a v_0 y v_1,
- B_2 que incluye a v_0 y v_2,
- B_3 que incluye a v_0 y v_3,

- B_4 que incluye a v_2 y v_3,
- B_5 que incluye a v_1 y v_3,
- B_6 que incluye a v_1 y v_2,

de manera que B_1 es paralela a B_4 y B_2 es paralela a B_5. Se escoge un punto $v \in B_6 \setminus \{B_3 \cap B_6, v_1, v_2\}$. Entonces, $S = (B_1 \cup B_2 \cup \{v, v_3\}) \setminus \{v_1, v_2\}$ es un subconjunto generador que no contiene a una línea completa.

```
< 1> Condiciones iniciales
< 2>      j = 1;
< 3>      v' = q^2 + q + 1;
< 4>      sea D el conjunto de diferencias del pp de orden q;
< 5> En el j-ésimo paso de almacenamiento
< 6>      sea B = {x + j mod v' | x ∈ D};
< 7>      sea B' = B - D;
< 8>      sea i el mejor sitio tal que i ∈ B';
< 9>      envía el fragmento[j] a i;
<10>      j = (j + 1) mod v + 1;
```

Figura 3.6: ¿Qué sitios serán las bodegas, si se usa "plano afín"?

Un esquema de almacenamiento basado en plano afín ofrece un número de bloques casi tan pequeño como aquellos basados en plano proyectivo o ranura deslizante, es decir $b = O(v)$. Además, $k = \sqrt{v}$, lo que significa que el precio total que se obtiene(3.2) es cercano al óptimo. Con todo, su mejor característica es el número crítico de fallas $f = 2k - 1$, lo cual indica que tolera el doble del número de fallas que otros esquemas con precio semejante (ss and pp).

La colección de bloques de este esquema se genera utilizando el procedimiento distribuido que se observa en la fig. 3.6. Como se mencionó anteriormente, un plano afín puede construirse a partir de un plano proyectivo del mismo orden. De esta manera, todo sitio genera, en cada paso, la siguiente línea de un plano proyectivo y descarta de ella los puntos pertenecientes a un bloque fijo. Por conveniencia, este bloque será el mismo conjunto de diferencias que genera a la geometría proyectiva. Entonces, a partir de la línea que resulta, el sitio reconoce a la "mejor" bodega donde guardará su nuevo fragmento.

3.6 Discusión de resultados

En esta sección se busca evaluar los diferentes esquemas presentados sobre la base de dos parámetros directamente relacionados: precio y tolerancia. Con ellos, el usuario o diseñador de un esquema podrá decidir un punto de equilibrio entre la capacidad para tolerar fallas y el costo que esté dispuesto a pagar por esta prestación. Con estos criterios de evaluación podrá observarse cómo los esquemas desarrollados caracterizan al espacio donde caben todas las soluciones de almacenamiento distribuido que obedezcan los principios enunciados en la sección 3.2.

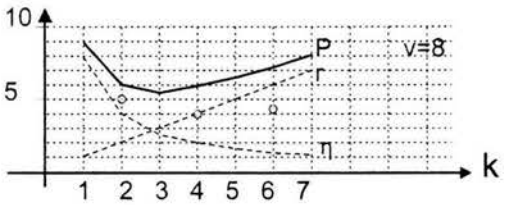

Figura 3.7: r, η y P, como funciones de k

Supóngase que se requiere instalar un esquema ss sobre un conjunto de v sitios. De las propiedades de este esquema se garantiza la viabilidad de la estrategia de selección subyacente con un número de bloques $b = v$, un paso de deslizamiento $\gamma = 1$ y cualquier tamaño de bloque $k \in [1, v-1]$. La figura 3.7 muestra con líneas discontinuas los parámetros $r = bk/v$ y $\eta = v/k$ como funciones de k para un valor fijo de $v = 8$, un número de bloques $b = v$, y un deslizamiento $\gamma = 1$.

Puede observarse que, cuando $k = 1$, r y η alcanzan su mínimo y máximo, respectivamente. En el extremo opuesto, cuando $k = v-1$, su situación relativa se intercambia. Estas tendencias contrarias pueden explicarse en términos del trabajo individual y colectivo realizado durante un paso de almacenamiento. En el primer caso, cada bloque está compuesto de un solo sitio, lo que significa que durante el paso que le corresponde, la única bodega designada debe cargar con todo el trabajo posible. En el segundo caso, casi todos los sitios funcionan como bodegas en cualquier paso de almacenamiento, lo que significa que el trabajo de cada bodega se reduce al mínimo. Desde esta perspectiva, se sugiere que η caracteriza al esfuerzo individual, mientras que r caracteriza al esfuerzo colectivo.

Por otra parte debe tomarse en cuenta que, en cualquiera de estas situaciones extremas, existe una ejecución distribuida que paga un costo excesivo. Ya que, en el primer caso, los sitios podrían verse obligados a detener su tarea principal durante un paso de almacenamiento muy largo, si requirieran coordinarse con el único sitio que funge como bodega. En tanto, para el caso contrario, cada componente se distrae de su tarea principal con mucha frecuencia, porque interviene como bodega en casi todos los pasos de almacenamiento.

Es interesante considerar que, para cualquier $k \in [1, v-1]$, $r\eta = b$, lo que implica que, para cualquier valor de k, la cantidad total de información almacenada en cada sitio es constante. ¿Significa esto que una solución arbitraria es tan buena como cualquier otra? La curva de precio que se muestra con línea continua (definida en la ec. 3.2) como una figura de desempeño que sintetiza los diferentes factores que contribuyen en la calidad de un esquema de almacenamiento: el espacio requerido en cada sitio, la posibilidad de ralentización de la tarea principal, el esfuerzo individual y colectivo. Por tanto, debe buscarse un punto de equilibrio que optimice la suma de estas contribuciones. Los puntos grises de la misma figura muestran como, para el mismo valor de $v = 8$, un paso de deslizamiento $\gamma = 2$ y ciertos valores de k, se puede construir una estrategia con un precio por debajo de la curva continua. Sin embargo, se trata de puntos aislados. Esto quiere decir que dicha curva señala la frontera hasta la que es posible construir una solución para cualquier entero $k \in [1, v-1]$).

La fig. 3.8 muestra todas las curvas de precio de los esquemas desarrollados en este trabajo.

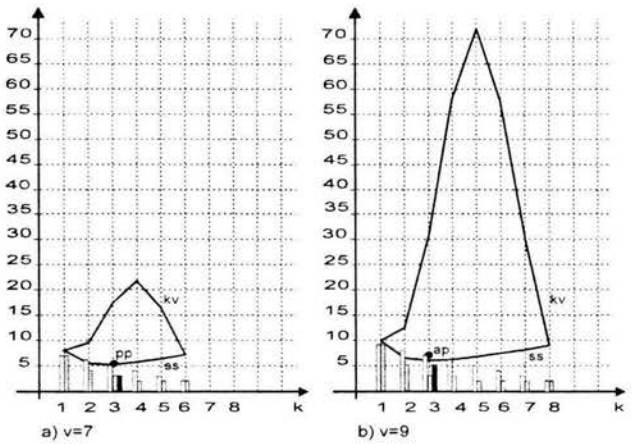

Figura 3.8: P y f como funciones de k, para kv, ss, pp y ap

Es importante señalar que kv es el único esquema en el que el número de bloques depende del tamaño del bloque. Es muy fácil demostrar que, bajo las mismas condiciones, cualquier esquema exhibirá un precio menor o igual que el correspondiente a kv, lo que lo hace el esquema más caro posible. En cuanto a la familia de geometrías finitas, existen como puntos aislados en el plano $k - P$ (mostrados en negro), y sólo para algunos valores de v. En otras palabras, sólo se puede instalar un esquema basado en plano proyectivo o afín, cuando el total de sitios participantes es $q^2 + q + 1$ o q^2, respectivamente (por ejemplo, $v = 7$ y $v = 9$, como en la fig. 3.8). En los casos en que estas soluciones son realizables y pueden compararse con los otros esquemas, se les puede ubicar muy cerca de la curva ss.

	kv	ss	pp	ap
	$k = 1$	$k = \sqrt{v}$	$k = \frac{1}{2} + \sqrt{v - \frac{3}{4}}$	$k = \sqrt{v}$
P	$v + 1$	$2k$	$k + \frac{v}{k}$	$2k + 1$
f	v	$\lceil k \rceil$	k	$2k - 1$

Tabla 3.2: Las condiciones de precio óptimo para cada esquema

Para terminar de evaluar un esquema de almacenamiento, además del precio, existe un segundo parámetro a considerar: la tolerancia, que se mide usando el valor f. Este parámetro, que en la fig. 3.8 se muestra en barras blancas, grises y negras, para kv, ss y pp/ap, respectivamente, evalúa la habilidad de un esquema para tolerar fallas de paro. En un párrafo anterior se mencionó que, con algunos valores de v y k, es posible construir estrategias de precio muy bajo pero, como es de esperarse, éstas serán también las soluciones más vulnerables. La tabla 3.2 compara los esquemas presentados y muestra cómo, el precio y la tolerancia se encuentran directamente relacionados. Esto es, puede observarse que el esquema más tolerante es también el de mayor precio (y viceversa).

Los esquemas presentados a lo largo de este trabajo muestran ventajas y limitaciones. El primero de ellos, kv, ofrece la estrategia más robusta y soporta diferentes valores de k. Por otro lado, el mismo tamaño de bloque determina el período y, en consecuencia, el precio, que resulta ser el más caro posible. El segundo esquema, ss, muestra una tolerancia que puede manipularse también, por medio de su tamaño de bloque. De hecho en los casos extremos, i.e. cuando $k = 1$ ó $k = v - 1$, ss y kv son indistinguibles en cuanto a precio y tolerancia se refieren. Sin embargo, en cualquier otra circunstancia, las figuras de desempeño de ss son inferiores. El período de almacenamiento que puede conseguirse con ss es una característica de la que puede sacarse provecho. Finalmente, las geometrías finitas son soluciones elegantes pero rígidas, que son viables para cierto número de sitios y bajo condiciones que no aceptan manipulación. El esquema basado en ap duplica la tolerancia que puede conseguirse con ss o pp, con un precio ligeramente superior.

En lo que resta de esta sección se presenta un enfoque alternativo para combinar estas soluciones y conseguir parámetros de desempeño mejorados. Del análisis correspondiente se concluye que es posible, bajo ciertas condiciones, tomar lo mejor de dos esquemas diferentes para encontrar un esquema adaptado a los requerimientos particulares de un sistema.

Sea un conjunto V de sitios a cargo de una ejecución distribuida, tal que $v = |V|$. Dos estrategias de selección $\mathcal{B}_1$ y $\mathcal{B}_2$ pueden combinarse e instalarse sobre V para producir la estrategia combinada $\mathcal{B}_1(\mathcal{B}_2)$, de acuerdo con la siguiente:

regla $\mathcal{B}_1(\mathcal{B}_2)$ Instálese $\mathcal{B}_1$ sobre V, para producir la colección de subconjuntos B_i, para $i = 1, \ldots, b_1$, tal que $v' = |B_i|$.

Instálese una replica de $\mathcal{B}_2$ sobre cada subconjunto B_i de $\mathcal{B}_1$, para producir la colección de bloques $B_{i,j}$, para $i = 1, \ldots, b_1$ y $j = 1, \ldots, b_2$, tal que $k' = |B_{i,j}|$.

Los bloques $B_{i,j}$ se calendarizan en orden lexicográfico.

Teorema 3.1 $\mathcal{B}_1(\mathcal{B}_2)$ *es una estrategia balanceada e incompleta.*

Puesto que $\mathcal{B}_1$ es balanceada e incompleta, cuando se instala sobre V, cualquier elemento $v_l \in V$ participa en r_1 subconjuntos, donde $r_1 < b_1$.

Sea B_i uno de los r_1 subconjuntos donde v_l aparece. Ahora, si se instala $\mathcal{B}_2$ sobre B_i, entonces v_l (como cualquier otro elemento en B_i) participa en r_2 de los bloques resultantes, donde $r_2 < b_2$. Por tanto, para todo $v_l \in V$, $r_l = r_1 r_2 < b_1 b_2$. Además, por la regla de construcción, $k_{ij} = k'$.

Ahora se considerarán dos combinaciones en las que se demuestra cómo, usando esta nueva regla, puede mejorarse la tolerancia bajo las misma condiciones de precio que la estrategia simple $\mathcal{B}_1$ en la que se basan nuestros casos compuestos.

ss(kv) En este caso, se utiliza un esquema ss con un valor de v' tal que $v'|v$, para dividir al conjunto V en $b_1 = \frac{v}{v'}$ subconjuntos *disjuntos* (ranuras) de tamaño v'. Enseguida, sobre cada uno de ellos se instala un esquema kv con tamaño de bloque $k' = v' - 1$. De las propiedades de kv se tiene que $b_2 = v'$. Además, cada esquema kv individual muestra un número crítico de fallas $f' = 2$. En consecuencia, el esquema combinado tiene un período $b = b_1 b_2 = v$ y un número crítico de fallas $f = b_1 f' = 2\frac{v}{v'}$.

ss(ap) En este caso, se utiliza un esquema ss con un valor de v' tal que $v'|v$, para dividir al conjunto V en $b_1 = \frac{v}{v'}$ subconjuntos *disjuntos* (ranuras) de tamaño v'. Enseguida, es posible instalar un esquema ap sobre cada uno de ellos, siempre que $v' = q^2$ para una potencia de número primo q. De las propiedades de ap se tiene que $b_2 = q^2+q = v'+\sqrt{v'}$. Además, cada esquema ap muestra un número crítico de fallas $f' = 2q - 1 = 2\sqrt{v'} - 1$. El esquema combinado tiene un período $b = b_1 b_2 = \frac{v}{v'}(v' + \sqrt{v'})$ y un número crítico de fallas $f = b_1 f' = \frac{v}{v'}(2\sqrt{v'} + 1)$.

	ss(kv)	ss(ap)
v'	$v'\|v$	$v'\|v$
k'	$v' - 1$	$\sqrt{v'}$
$b = b_1 b_2$	v	$(v' + \sqrt{v'})\frac{v}{v'}$
$f_{\mathcal{B}_1(\mathcal{B}_2)}$	$2\frac{v}{v'}$	$\sim 2\frac{v}{\sqrt{v'}}$
$f_{\mathrm{ss}\|v,k'}$	$\lceil \frac{v}{v'-1} \rceil$	$\sim \frac{v}{\sqrt{v'}}$
$f_{\mathrm{ss}\|b,k'}$	$\lceil \frac{v}{v'-1} \rceil$	$\sim \frac{v}{\sqrt{v'}}$

Tabla 3.3: Parámetros básicos de los esquemas combinados

Ahora vamos a comparar la tolerancia conseguida contra dos casos de "control" diferentes. En el primero, tenemos un esquema ss simple, con parámetros v y k', i.e. las mismas condiciones finales que resultaron en los esquemas combinados. En el segundo, suponemos un esquema ss simple otra vez, pero con tantos elementos como sea necesario para conseguir el mismo período b de los esquemas combinados. En ambos casos $\gamma = 1$. Hemos decidido estos "bancos" de prueba, porque siempre es posible construir un esquema ss para cualesquiera valores de v y k. Con el primero deseamos conocer la tolerancia conseguida si sólo aplicáramos una estrategia simple. En el segundo, deseamos conocer la tolerancia que conseguiríamos si pagáramos el mismo número de bloques que la solución combinada. En las columnas de la tabla 3.3 puede verse el valor de f para el caso de estudio y los dos casos de comparación.

Parece natural pensar que puede sacarse más provecho de la estrategia combinada ss(kv) si se selecciona una $k' < v' - 1$. Sin embargo, el sistema tendría que estar preparado para pagar una elevación en el número de bloques, que no siempre se justifica. Por ejemplo, teniendo $k' = v' - 2$ produciría una tolerancia $f_{\mathrm{ss(kv)}} = 3\frac{v}{v'}$, pero con un número de bloques $b = \frac{v(v'-1)}{2}$. En contraste, si se tuviera un esquema simple ss, con $\frac{v(v'-1)}{2}$ y un tamaño de bloque $k' = v' - 2$, conseguiríamos una tolerancia $f_{\mathrm{ss}} = \lceil \frac{v}{2} \rceil$.

3.7 Trabajo futuro

Los esquemas de almacenamiento pueden llegar a convertirse en importantes procedimientos para el soporte del cómputo distribuido de escala masiva, como los sistemas simuladores de eventos discretos de alto desempeño. En estos escenarios, sería demasiado costoso reestablecer una ejecución desde el principio, si llegara a ocurrir una falla de paro. Todo parecer indicar que almacenar la "historia" del sistema es la única salida.

En este trabajo se presentaron tres diferentes esquemas de almacenamiento. El primero de ellos, denominado "selección de k de v", no parece muy útil a menos que la aplicación esté preparada para pagar un precio muy alto para tolerar la máxima cantidad de fallas. El segundo esquema, "ranura deslizante", alcanza su precio óptimo con el menor número de bloques. En cierta forma, este esquema y el de "plano proyectivo", de la tercer familia, son muy semejantes en cuanto a sus parámetros de desempeño óptimo. Sin embargo, los planos proyectivos sólo existen para potencias de números primos, mientras que siempre es posible construir un esquema de ranura deslizante. El último esquema, "plano afín", también de la familia de estrategias geométricas, es muy cercano a los dos anteriores, pero tiene una tolerancia superior. En la última parte del trabajo se demostró que dos esquemas pueden combinar fuerzas para ofrecer un procedimiento mejorado de almacenamiento distribuido.

Llama la atención que algunas de las mejores soluciones, provienen de diseños con fuertes propiedades de intersección por pares, mientras que el problema original de cómputo distribuido no parece requerir estas propiedades (al menos de primera impresión). Sería interesante investigar esta relación, así como la existencia de otros esquemas.

El tiempo es una dimensión de los esquemas de almacenamiento escasamente considerada, pero que merece un análisis cuidadoso. Visto de otro modo, es evidente que los bloques son recalendarizados al final de un ciclo, pero esto no impide la posibilidad de intercalar dos o más estrategias, cada una con un calendario diferente, adaptado a sus condiciones de precio y desempeño. Por otra parte, es posible también desarrollar un calendario estocástico, en lugar de un calendario determinista. Para ello se requeriría construir una cadena de Markov cuyos estados fueran asociados con los bloques de la estrategia de selección. De esta forma, los sitios participantes podrían cooperar con distintas tasas de operación, adaptadas a sus características particulares, como capacidad, confiabilidad, rapidez, carga, etc.

Merece una mención especial, como un enfoque complementario para almacenar información, la técnica de dispersión de información desarrollada por Rabin [14]. Sea F un fragmento de un estado global, que requiere almacenarse. Usando el algoritmo de Rabin, F se transforma en $k-1$ archivos dispersos. Cada uno de $|F|/m$ bits, donde $k-1=m+l$. Luego, el archivo original se guarda en su lugar correspondiente, mientras que los archivos dispersos se almacenan en los demás $k-1$ sitios que forman el bloque designado. De las propiedades del algoritmo se garantiza que si el original se perdiera, podría reconstruirse a partir de cualesquiera m archivos dispersos. Esto significa que el bloque puede tolerar hasta $l+1$ fallas antes de que se le considere destruido. En estas condiciones, el costo de la reconstrucción se "amortiza" al dividirse entre los sitios que aportan sus capacidades de almacenamiento de respaldo y, por supuesto, debido a la tolerancia superior que se consigue.

Por último, supóngase que un sitio cae en paro después de que el esquema de almacenamiento donde trabaja ha estado en operación por al menos un ciclo. ¿Cómo afecta este evento a los sitios sobrevivientes? ¿qué pasaría si ocurrieran más fallas? ¿cómo puede minimizarse el daño? Al menos tres posibles trayectorias de acción pueden seguirse:

1. Avisar a los sobrevivientes la identidad del sitio con falla y conservar el mismo esquema, pero omitiendo los bloques donde éste tomaba parte.

2. Instalar un nuevo esquema sobre los sitios sobrevivientes con el mismo o con un nuevo tamaño de bloque.

3. Prevenir esta contingencia comenzando con v' sitios, pero instalando el esquema como si sólo hubieran v de ellos y guardando los otros $v' - v$ sitios como reservas. Si ocurriera una falla, un sitio de reserva tomaría el lugar del sitio caído.

En cualquiera de las acciones de contingencia se requiere una operación de difusión para sincronizar el paso correctivo. Sin embargo, algunas acciones pueden ser más efectivas: En el último procedimiento se preserva el esquema con todos sus parámetros. Esta solución soporta hasta $v' - v$ paros y, al mismo tiempo, es totalmente compatible con las otras alternativas. Es decir que, los casos 1 ó 2 podrían utilizarse cuando el caso 3 quedara fuera de aplicación. En conclusión, parece que es un buen principio de diseño instalar un esquema de almacenamiento sobre condiciones holgadas, dejando espacio suficiente para resolver contingencias de manera rápida y efectiva, mediante sitios de reserva, al mismo tiempo que se dispone de un plan secundario, por si se le requiriera.

Capítulo 4

El problema de escalabilidad

En un esquema de almacenamiento se requiere encontrar un punto de equilibrio entre el costo de las comunicaciones y la disponibilidad de la información. Para sistemas pequeños o medianos, el dispositivo de almacenamiento se encuentra en la "vecindad" de la fuente donde se origina el fragmento. Sin embargo, para un sistema de mayor escala, el almacenamiento puede resultar muy caro cuando la bodega se encuentra lejos de la fuente. Bajo esta premisa, podría ser conveniente particionar el conjunto de sitios en subconjuntos disjuntos e instalar un esquema de almacenamiento sobre cada uno, siempre que se mantenga un mecanismo que sincronice a todos los esquemas resultantes. En este capítulo, basado en [2], se desarrolla un algoritmo distribuido que resuelve el llamado problema de partición. *Se presentan resultados que demuestran la factibilidad de una solución heurística de calidad, obtenida por estos medios.*

Ha puesto orden en las galaxias y distribuye bien el tránsito en el camino de las hormigas.

Jaime Sabines en "Me encanta Dios"

4.1 Introducción

La partición de la gráfica que subyace en un sistema distribuido es un problema que se presenta con relativa frecuencia. En diferentes circunstancias se requiere cortar la gráfica en subconjuntos disjuntos e instalar un subsistema sobre cada una de las particiones que resulten. Para el caso particular que aquí se presenta, se necesita además garantizar que todas las particiones tienen el mismo número de nodos y que, para la subgráfica inducida por cada subconjunto, se optimiza una medida de distancia. Hasta ahora, el problema se ha trabajado con un enfoque centralizado, esto es, se construye una estructura de datos que modela a la gráfica, se resuelve el problema sobre este modelo y se difunde el resultado desde un punto central, para informar a los sitios sobre la partición a la que quedan adscritos. A medida que los sistemas de cómputo crecen en escala, este enfoque pierde viabilidad. Se requiere una alternativa distribuida mediante la cual cada sitio se agregue, por sí mismo, al subconjunto que defina su partición, basado solamente en la información local de que disponga.

Contribución

La idea central de esta investigación es establecer un puente entre los enfoques centralizado y distribuido con que se atacan los problemas difíciles (en un sentido de complejidad computacional). La diferencia entre ambos puede explicarse considerando la manera como cada uno modela la instancia de un problema: un método centralizado requiere una estructura de datos monolítica para representar el espacio de soluciones, mientras que los sitios o procesadores que participan en un método distribuido sólo disponen de información relativa a su condición local. Si una metaheurística centralizada debiera convertirse en un método distribuido, debería basarse preferentemente en entidades cooperativas que pudieran "viajar y negociar" sobre el espacio de soluciones, no monolítico, representado por el mismo sistema.

Nuestro enfoque está basado en conceptos de sistemas biológicos que han probado ser de utilidad en el dominio de la computación. Se sabe que las colonias de hormigas resuelven problemas de búsqueda de alimento que pueden modelarse como problemas de optimización combinatoria. Para ello, las hormigas utilizan un sencillo mecanismo biológico: cada individuo comunica a los demás el camino que ha seguido descargando en él una sustancia química. Las hormigas tienden a seguir estos rastros de feromonas. Al cabo de cierto tiempo, los caminos más cortos entre el nido y las fuentes de comida se recorren con más frecuencia y van acumulando más y más feromona, lo que a su vez refuerza el rastro anterior.

El sistema de hormigas (artificial) que aquí se presenta, se basa en principios similares: los agentes se colocan en algún nodo del sistema distribuido. Luego, cada uno desarrolla un camino aleatorio bajo el control de probabilidades de tránsito que van ajustándose. Cuando un agente ha visitado todos los nodos, se evalúa su viaje y se asignan valores de "feromona" (nuevas probabilidades) a las aristas de su camino, en una cantidad proporcional a la calidad de la solución planteada por su recorrido. Este procedimiento puede repetirse varias veces. Como medida para prevenir la acumulación excesiva de feromona, se cuenta con un mecanismo de "evaporación" que reduce periódicamente los valores de feromona. Gutjahr [66] demostró que, para su modelo de sistema hormiga y bajo ciertas condiciones, las soluciones convergen al óptimo con una probabilidad que puede hacerse arbitrariamente cercana a 1.

El algoritmo HORMIGA desarrollado en nuestro trabajo es una adaptación del algoritmo distribuido para búsqueda en profundidad. Las instancias de HORMIGA, también llamadas *agentes*, viajan atravesando la gráfica de comunicaciones produciendo caminos aleatorios a partir de una distribución de probabilidades. Cada agente recoge la identidad del nodo que visita y lo asigna al subconjunto en construcción. Cuando éste alcanza el orden que se requiere, la "bolsa" se cierra y se abre una nueva colección de nodos. Regularmente, todos los agentes se reúnen para evaluar la mejor solución hasta entonces encontrada. Luego, la distribución de probabilidades se modifica para aumentar la posibilidad de alcanzar mejores salidas.

En la sección 4.2 se revisa el modelo de sistema con el que trabaja, se define formalmente el problema de partición y se presentan las características de nuestro bloque de construcción, i.e. se resumen las propiedades del algoritmo DFS distribuido. La sección 4.3 describe el llamado procedimiento de recompensa, en el que se modifica la distribución de probabilidades, con la que se construyen los recorridos aleatorios de los agentes. En la sección 4.4 se presenta un sumario de los pasos seguidos por un sistema sobre el que se implante nuestro sistema hormiga, desde el momento que arranca hasta el momento en que el problema se considera

resuelto. La sección 4.5 explica cómo se hizo la adaptación de la ficha DFS para convertirla en un agente y cómo se colectan los resultados para evaluar su calidad. La sección 4.6 desarrolla una discusión de los resultados. Las conclusiones se presentan en la sección 4.7.

Trabajo relacionado

Nuestra investigación guarda relación muy estrecha con el trabajo de Mourad et. al. [6] y el trabajo de Peleg [7]. El primero desarrolló una heurística centralizada para resolver un problema de partición similar al que aquí se presenta. En tanto, el segundo ha estudiado varios problemas distribuidos bajo el enfoque de sensitividad local. Sin embargo, en ninguna de estas fuentes se estudia una solución para el problema que aquí nos interesa.

En la propuesta de Mourad existe una etapa de preprocesamiento sobre la gráfica de entrada para producir un árbol generador con un máximo de hojas. Entonces, se echa a andar un algoritmo de búsqueda en profundidad sobre dicho árbol y, a medida que se recorre, los nodos se van incorporando (al vuelo) al subconjunto en construcción. Cuando éste alcanza el número de nodos que se requiere, se da por terminado y se abre un nuevo subconjunto. Por su parte, en el enfoque de Peleg, el crecimiento de cada partición está a cargo de un algoritmo que acumula capas concéntricas de nodos sobre un núcleo inicial. Cuando la tasa de crecimiento de la "bola" cae por debajo de un umbral preestablecido, se inicia el crecimiento de una nueva partición a partir de otro núcleo. En cualquiera de los casos considerados, se utiliza la búsqueda en profundidad (DFS) para garantizar que, en todo momento, a lo más existe un subconjunto en construcción, i.e. las particiones son creadas una tras otra. También es interesante observar que ambos métodos incorporan un mecanismo para acotar el diámetro de cada partición. En el primer caso, la etapa de preprocesamiento cumple esta función, mientras que, en el segundo, el crecimiento concéntrico es semejante al de los sincronizadores γ [67] que, se sabe, tienen esta propiedad.

Los *sistemas de hormigas* se remontan al trabajo pionero de Dorigo [5], pero no fue sino hasta algunos años cuando los equipos científicos comenzaron a considerarles como candidatos promisorios para desarrollar metaheurísticas distribuidas, debido a su naturaleza inherentemente distribuida y cooperativa. Se trata de sistemas multiagentes inspirados en el comportamiento de las hormigas reales.

4.2 El modelo, el problema y el bloque de construcción

En este proyecto se trabaja con algoritmos distribuidos en un modelo de *red asíncrona*. Se trata de una red de comunicación punto a punto (store and forward) descrita mediante una gráfica no dirigida denominada *gráfica de comunicaciones* $G = (V, E)$, de orden $n = |V|$ y tamaño $m = |E|$. El conjunto de nodos o vértices V representa los sitios o procesadores de la red y el conjunto de aristas E representa los canales de comunicación que interconectan a los sitios. No existe una memoria común que los nodos puedan compartir y cada nodo tiene una identidad que lo distingue del resto. Cada nodo procesa un mensaje que recibe de sus vecinos, desarrolla un cómputo local y eventualmente devuelve mensajes a sus vecinos. Se supone que todas estas acciones se completan en un tiempo despreciable. Además, los mensajes tienen una longitud fija y sólo pueden transportar una cantidad acotada de información. Cada

mensaje transmitido llega a su destino al cabo de un tiempo finito, pero impredecible. Este modelo aparece en Awerbuch [67] y Segall [68], entre otros autores.

Para evaluar el desempeño de los algoritmos que operan sobre el modelo de red asíncrona se utilizan las siguientes medidas de complejidad: La *complejidad de la comunicación* es el total de mensajes intercambiados durante la ejecución del algoritmo. La *complejidad del tiempo* es el máximo tiempo que puede transcurrir durante la ejecución del algoritmo, suponiendo que el tiempo de entrega de cualquier mensaje es, a lo más, una unidad. Este retardo acotado sólo se asume para evaluar esta segunda medida de complejidad

Para los fines de esta propuesta, G es una gráfica ponderada, cada una de cuyas aristas $e \in E$ tiene un peso positivo $w(e)$. La *distancia* $d(u, v)$ entre un par de vértices de G es la mínima longitud (peso) de todas las trayectorias $u - v$ en G, si acaso existen; en otro caso $d(u, v) = \infty$.

Suponiendo que $n = pN$, el problema de partición puede formularse en los siguientes términos:

> *Problema (SP).* Encuéntrese una partición de V en p subconjuntos disjuntos $V_1, V_2, \ldots, V_p$ de cardinalidad N, cada uno y tales que, si $d(u, v)$ denota la distancia entre los vértices u y v, entonces $\sum_{i=1}^{p} \sum_{u \in V_i} \sum_{v \in V_i - \{u\}} d(u, v)$ es mínima.

Se sabe que el problema *SP* es *NP-completo* para un valor fijo $N \geq 3$ [6]. La solución heurística de SP, que presentamos, se basa en el algoritmo distribuido de búsqueda en profundidad (DFS) [13]. Un algoritmo de este tipo produce como salida un árbol DFS construido sobre la gráfica de comunicaciones G en donde cada nodo sólo conoce a las aristas que lo conectan con su ancestro inmediato y con sus descendientes directos. Sus medidas de complejidad son $O(m)$ y $O(n)$, para mensajes y tiempo respectivamente.

> Los conceptos de *ancestro*, *descendiente* y *nivel* pueden definirse de manera recursiva: la raíz es, axiomáticamente, considerada como su propio padre, con un nivel 0. El ancestro de un nodo es su padre o un ancestro de su padre. El nivel de un nodo es 1 más el nivel de su padre. El descendiente de un nodo es cualquiera de sus hijos o un hijo de cualquiera de sus descendientes.

4.3 El procedimiento de recompensa

Según Gutjhar [66], un *sistema hormiga* se compone de un conjunto $\{A_1, A_2, \ldots, A_Z\}$ de *agentes*. El *ciclo* del sistema es el período de tiempo durante el cual cada agente completa un camino sobre la gráfica de construcción $G = (V, E)$.

Sea $u = (u_0, u_1, \ldots, u_{t-1} = k)$ el camino que llega hasta el nodo k, recorrido por un agente fijo durante el ciclo corriente m. Para un nodo l de V, se denota como $l \in u$ si el nodo está contenido en el camino y como $l \notin u$, en otro caso. La probabilidad de que el agente se dirija del nodo k al nodo l, en el ciclo m, se denomina *probabilidad de transición* $p_{kl}(m, u)$ y se determina de la siguiente forma:

$$p_{kl}(m, u) = \frac{[\tau_{kl}(m)]^{\alpha}[\eta_{kl}(m)]^{\beta}}{\sum_{r \notin u,(k,r) \in E}[\tau_{kr}(m)]^{\alpha}[\eta_{kr}(m)]^{\beta}},$$

si $l \notin u$ y $(k,l) \in E$, o bien

$$p_{kl}(m,u) = 0,$$

en otro caso. Los números $\eta_{kl}(u)$ se denominan "valores de deseabilidad" y los números $\tau_{kl}(m)$ se denominan "valores de feromona". Los primeros pueden obtenerse a partir de la solución del problema en consideración mediante un algoritmo *voraz* (greedy). Por cuanto a los segundos, al iniciar el ciclo 1,$\tau_{kl}(1) = 1/$(número de vecinos de k). Al concluir el ciclo m se aplica la siguiente regla de actualización. Primero, para cada agente A_z y cada arista (k,l), se determina un valor $\Delta\tau_{kl}^{(z)}(m)$ como función de la solución asignada al camino de A_z durante el ciclo corriente. Supóngase que la solución tiene un costo f_z entonces, para cada arista (k,l):

$$\Delta\tau_{kl}^{(z)}(m) = \begin{cases} \phi(f_z) & \text{si el agente } A_z \text{ atravesó } (k,l), \\ 0 & \text{en otro caso,} \end{cases}$$

donde ϕ es una función no creciente que depende de los caminos completados durante los ciclos $1,\ldots,m-1$, y mide en cuánto mejora la solución más reciente, comparada con resultados previos. Sea R la *recompensa total*, tal que

$$R = \sum_{(k,l)\in E} \sum_{z=1}^{Z} \Delta\tau_{kl}^{(z)}(m).$$

Luego, si $R = 0$ (no se mejoró la solución durante el último ciclo m), el próximo valor de $\tau_{k,l}(m+1)$ es

$$\tau_{k,l}(m+1) = \tau_{k,l}(m)$$

En cambio, si $R > 0$,

$$\tau_{k,l}(m+1) = (1-\rho)\tau_{k,l}(m) + \rho\Delta\tau_{kl}(m+1),$$

donde ρ se conoce como *factor de evaporación*, mientras que el valor $\Delta\tau_{kl}(m+1)$ es:

$$\Delta\tau_{kl}(m+1) = \frac{1}{R}\sum_{z=1}^{Z} \Delta\tau_{kl}^{z}(m).$$

4.4 La jornada de una colonia de hormigas

Esta sección ofrece un resumen de los pasos que se siguen desde el momento que el sistema arranca, hasta el momento en que se considera resuelto el problema.

1. Primeramente, se efectúa una etapa de preprocesamiento, que consiste en una versión distribuida del algoritmo Ford-Fulkerson (FRD) [69]. Al final de este paso, cada nodo conoce la mejor distancia hasta cualquier otro nodo de la gráfica de comunicaciones.

```
< 1> al recibir DESCUBRE desde j efectúa
< 2>      sin_visitar ← sin_visitar −{j}
< 3>      si visitado == FALSO
< 4>           visitado ← VERDADERO
< 5>           padre ← j
< 6>           para todo k ∈ vecinos−{j}
< 7>                envía AVISO a k
< 8>           llama a continua_recorrido()

< 9> al recibir REGRESA desde j efectúa
<10>      llama a continua_recorrido()

<11> al recibir AVISO desde j efectúa
<12>      sin_visitar ← sin_visitar −{j}

<13> al recibir REPORTE desde j efectúa
<14>      si soy concentrador de la lista c/N
<15>           inserta id de la fuente en lista
<16>           si tamaño de lista == N
<17>                envía LISTA a mis descendientes en lista
<18>                diámetro = 0
<19>                candidatos_recibidos = 0
<20>      otro reexpide REPORTE a padre

<21> al recibir LISTA desde j efectúa
<22>      si estoy en la lista
<23>           candidato = max{d(i,v)|v ∈ lista}
<24>           envía CANDIDATO a padre
<25>      envía LISTA a mis descendientes en lista

<26> al recibir CANDIDATO desde j efectúa
<27>      si soy concentrador de la lista c/N
<28>           si candidato > diámetro
<29>                diámetro = candidato
<30>           candidatos_recibidos++
<31>           si candidatos_recibidos == N − 1
<32>                envía DIAMETRO a padre
<33>      otro reexpide CANDIDATO a padre
```

Figura 4.1: Algoritmo HORMIGA (parte 1).

2. Enseguida se crean y liberan Z agentes desde un nodo fijo de la gráfica, denominado *nido*. Cada agente, u hormiga, completará un recorrido DFS aleatorio, visitando cada nodo de la gráfica.

3. Una vez que una hormiga abandona un nodo, es decir, deja de utilizarlo como base para explorar la gráfica, el nodo recibe un valor denominado "mi_num" y queda adscrito a la partición en construcción. Cada vez que una partición alcanza un orden previamente establecido, se da por terminada y se designa a un nodo concentrador encargado de medir la calidad de esta solución parcial. Mientras existan nodos sin visitar (i.e. el recorrido DFS aún no termina), se inicia la construcción del siguiente subconjunto.

4. Un concentrador se encarga de difundir sobre la partición que administra una lista con los nodos para los que trabaja. Cada nodo que la recibe debe reportar a su concentrador la mayor distancia que observa dentro de su partición. El concentrador reúne estos reportes y con ellos elige al diámetro de la partición. Después, envía este valor hacia el nido. El algoritmo que se presenta en este trabajo, queda a cargo de los pasos 2...4.

5. La solución encontrada por cada agente queda caracterizada por el mayor diámetro de las particiones que construyó. Entonces, el nido difunde una lista de las hormigas que produjeron salidas que mejoraron los resultados previos. Con ello se indica a los nodos que favorezcan recorridos semejantes a los de las hormigas premiadas. Este procedimiento de recompensa utiliza un algoritmo de propagación de información con retroalimentación (PIF:Propagation of Information with Feedback) [68].

6. Esta secuencia se repite a partir del paso 2, por un número fijo de ciclos o hasta alcanzar un criterio de convergencia.

4.5 ¿Cómo convertir una ficha DFS en una hormiga?

Esta sección describe el algoritmo HORMIGA, se trata de una adaptación de la versión distribuida del algoritmo DFS. El siguiente código muestra el código del algoritmo en el nodo i. Durante su ejecución, éste intercambia los siguientes mensajes:

DESCUBRE. Un nodo envía este mensaje a un vecino al que aún no se ha explorado, para indicarle que deberá continuar con la construcción del árbol DFS. El receptor considerará como su padre al nodo de quien lo reciba.

REGRESA. Un nodo envía este mensaje a su padre, cuando ha concluido la exploración local DFS para la que fungió como base, i.e. no tiene más nodos que explorar.

AVISO. Un nodo envía este mensaje a sus vecinos inmediatos, excepto su padre, para anunciarles que ha comenzado su ejecución. Los que lo reciban deben tomar nota de la identidad del emisor y evitar su exploración.

REPORTE. Un nodo envía este mensaje sobre la ruta de sus ancestros, hasta alcanzar a su concentrador correspondiente, el cuál se encargará de reunir las identidades de todos los nodos que pertenezcan a la misma partición.

```
<34> al recibir DIAMETRO desde j efectúa
<35>      si soy el nido
<36>           si diámetro > máximo
<37>                máximo = diámetro
<38>           diámetros_recibidos++
<39>           si diámetros_recibidos == n/N
<40>                termina
<41>      otro reexpide DIAMETRO a padre

<42> procedimiento continua_recorrido()
<43>      si sin_visitar ≠ ∅
<44>           llama a soy_concentrador()
<45>           escoge aleatoriamente a un l ∈ sin_visitar
<46>           envía DESCUBRE a l
<47>           sin_visitar ← sin_visitar −{l}
<48>      otro mi_num= c++
<49>           llama a soy_concentrador()
<50>           si padre ≠ i
<51>                envía REGRESA a padre
<52>           envía REPORTE a padre

<53> procedimiento soy_concentrador()
<54>      si c > 0
<55>           npart = c/N
<56>           si primera_vez
<57>                primera_vez ← FALSO
<58>           otro si ((npart≠upart)o((mi_nummodN)== N − 1))
<59>                     crea lista para upart
<60>           upart = npart
```

Figura 4.2: Algoritmo HORMIGA (parte 2).

LISTA. Una vez que un concentrador ha reunido las identidades de los N nodos que administra, entonces transmite esta lista hacia sus descendientes que aparecen en ella.

CANDIDATO. Un nodo envía este mensaje sobre la ruta de sus ancestros, hasta alcanzar a su concentrador. El mensaje contiene la mayor distancia que el emisor ha medido al interior de su partición.

DIAMETRO. Una vez que un concentrador ha reunido los candidatos de los N sitios que administra, elige de entre ellos al diámetro de su partición y luego envía este valor hacia el nido (la raíz).

Al mismo tiempo, el nodo i administra un conjunto de variables con las que da cuenta de su estado local:

visitado: VERDADERO si i ya inició su ejecución local. Inicialmente es FALSO.

vecinos: Es el conjunto de nodos con los que i comparte un canal de comunicación.

sin_visitar: El conjunto de vecinos que pueden convertirse en sus hijos.

padre: El nodo que inició la ejecución del algoritmo en i.

candidato: La mayor distancia que i reporta a su concentrador.

candidatos_recibidos: El total de candidatos que un concentrador lleva procesados.

diámetro: El mayor de los N candidatos recibidos en el concentrador.

diámetros_recibidos: El total de diámetros procesados en el nido.

máximo: El mayor diámetro reportado desde cualquier concentrador.

mi_num: El número que se asigna a un nodo cuando concluye su ejecución local de DFS.

upart: La última partición en construcción de la que se tiene noticia.

npart: La nueva partición en construcción.

primera_vez: FALSO si i no conoce el valor de upart.

4.6 Discusión de resultados

Esta sección presenta una justificación formal del funcionamiento correcto de nuestro algoritmo. También fija algunas cotas en las medidas de complejidad del sistema.

Lema 4.1 *Cada nodo es visitado y asociado con una partición exactamente.*

El algoritmo DFS garantiza que, si G es conexa (lo cual se asume), cada nodo es visitado y puede reconocer el momento en que concluye su ejecución local del algoritmo. Entonces el nodo recibe (por única vez) un valor c (inicialmente igual a 0) que lo identifica como el (c mod N)-ésimo nodo de la (c/N)-ésima partición hasta ese momento construida. Inmediatamente después, el nodo incrementa c en 1 y lo reexpide sobre la ficha DFS que continua la exploración de G.

Lema 4.2 *Cada partición tiene exactamente el mismo número de nodos asociados.*

Por hipótesis, se supone que $N|n$ (i.e. el tamaño de las particiones divide al orden de la gráfica). El resultado anterior garantiza que cada nodo tiene un valor diferente en su parámetro local "mi_num". Además, las asignaciones de este valor se producen en forma consecutiva, comenzando por 0 hasta llegar a $n-1$. Entonces exactamente N nodos concentran su información hacia el nodo que administra la (mi_num/N)-ésima partición.

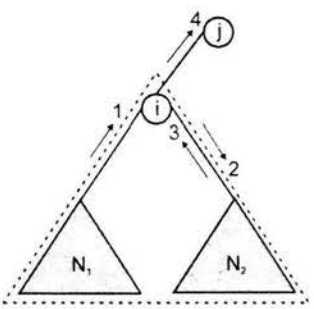

Figura 4.3: Un ancestro convertido en concentrador.

Lema 4.3 *Para cada partición construida existe exactamente un nodo, llamado concentrador que, sobre el árbol DFS construido, es el ancestro de mayor nivel para todo este subconjunto de nodos.*

Un nodo sabe que debe convertirse en concentrador, cuando la acción del algoritmo regresa a él y observa que la última partición de la que tenía noticia, se ha terminado. Supóngase que un nodo i (véase la fig. 4.3) recibe un mensaje para informarle que la construcción de su subárbol izquierdo se ha completado y contribuye con N_1 nodos a la partición que está en construcción. Al mismo tiempo, i aún tiene que construir su subárbol derecho que puede contener N_2 nodos (en un caso extremo $N_2 = 0$), a la partición corriente. Obsérvese también que i no puede ser una hoja. Los siguientes casos son los únicos que pueden ocurrir:

a: $N_1 = N - 1$ y $N_2 = 0$. En este caso i es el N-ésimo nodo que se incorpora a la partición corriente. Luego, i reconoce que es el ancestro de más alto nivel que puede convertirse en concentrador. Enseguida, se inicia la construcción de una nueva partición. Ningún otro ancestro tendrá noticias de la partición recién terminada y ningún nodo descendiente de i, podría haber tomado su lugar.

b: $N_1 + N_2 > N - 1$. En este caso i espera hasta que su subárbol derecho se termine y reconoce que la última partición de la que tenía noticia se ha cerrado, porque una nueva está en construcción. Luego, se convertirá en el concentrador de la partición que esperaba. Ningún otro ancestro tendrá noticias de la partición recién terminada y ningún nodo descendiente de i, podría haber tomado su lugar.

c: $N_1 + N_2 = N - 1$. Es el mismo caso que 'a'.

d: $N_1 + N_2 < N - 1$. En esta situación se deben replantear los 4 casos posibles, pero para el nodo j, padre de i.

Cualquiera de los casos anteriores puede presentarse en un nodo, que no sea una hoja, cuyo valor "mi_num" sea menor que $n-1-N$. Aquellas cuyo valor "mi_num" sea mayor o igual que $n - 1 - N$ y menor que $n - 1$, no llegarán a convertirse en concentradores porque la raíz del árbol, o nido, será la que tome este papel, de acuerdo con el caso 'a'.

Lema 4.4 *Un agente u hormiga termina su camino en el nido desde el que fue liberado.*

Se sigue de la propiedad de terminación de DFS.

Teorema 4.1 *Cuando una hormiga termina su camino, ha construido n/N particiones cuya calidad se evalúa con los concentradores designados, quienes se encargan de medir los diámetros de las particiones que administran.*

Inmediatamente que un nodo se convierte en concentrador, comienza a reunir los identificadores de los nodos que administra. Estos son transmitidos desde cada nodo que recibe el valor $mi\_num = c$ y ascienden por el subárbol recién construido. Una vez que el concentrador completa su lista, la difunde sobre el mismo subárbol. Luego, el concentrador recibe de regreso una serie de valores de donde escoge al diámetro de su partición. Entonces envía este valor hacia la raíz. Finalmente, todos los diámetros convergen en ese punto.

Podemos conjeturar que el proceso descrito es eficiente en su medida de tiempo, por los siguientes argumentos: el algoritmo FRD (paso 1), el algoritmo HORMIGA (pasos 2...4) y el algoritmo PIF (paso 5) tiene complejidades $O(n)$. Además, todos los Z agentes se ejecutan concurrentemente durante cada ciclo, esto es, el número de agentes no afecta (al menos teóricamente) la duración del sistema. Por último, creemos que es posible arrancar con más de un nido a fin de distribuir (y agilizar) la función de estos sitios especiales.

4.7 Trabajo futuro

Hemos presentado un algoritmo distribuido tipo hormiga para resolver la partición de sitios. Nuestra contribución tiende un puente entre las metaheurísticas centralizadas y el cómputo distribuido. También introdujimos el concepto de "concentrador" que administra una región de la solución y mide su calidad local. Creemos que varios problemas de optimización distribuida pueden beneficiarse con este enfoque, como pueden ser: la (re)distribución y el balance de carga, entre otros.

Como trabajo posterior, debe prepararse una agenda de experimentos para someter al sistema a diferentes escenarios y responder preguntas que quedan pendientes: Durante su ejecución, éste intercambia los siguientes mensajes: ¿cuál es el mejor número de agentes? ¿cuántos ciclos de búsqueda deben repetirse? ¿puede complementarse nuestra solución con algún otro enfoque heurístico?

Capítulo 5

Un simulador DES

En este capítulo, basado en [3], se presenta una herramienta para simulación de eventos discretos, orientada al estudio de algoritmos distribuidos cuyas entidades pueden modelarse usando máquinas de estados. El objetivo es desarrollar una plataforma experimental para verificar los aspectos empíricos de los algoritmos de tipo hormiga. Nuestro diseño ofrece flexibilidad para simular diferentes entornos de comunicación y ejecuciones concurrentes.

> Sobre la relación entre esos dos objetos se interrogó a un oráculo. Uno de los objetos - fue la respuesta - tiene la forma que los dioses dieron al cielo estrellado y a las órbitas en que giran los mundos; el otro no es más que su reflejo aproximado, como toda obra humana.
>
> *I. Calvino en "Las Ciudades Invisibles"*

5.1 Introducción

En computación distribuida, se han efectuado pocos estudios acerca del desempeño de los algoritmos bajo diferentes topologías de comunicación y modelos de retardo en los canales. En muchas de las veces, sólo se han publicado los análisis del peor caso respecto al número de mensajes y la complejidad en tiempo, esta última calculada bajo suposiciones de tiempo normalizado, i.e. se asume que el retardo sobre cada canal de transmisión es menor o igual que una unidad de tiempo. Otros estudios se efectúan solamente sobre ciertos tipos de topologías, como anillos o gráficas completas. En realidad, es muy difícil derivar resultados analíticos más finos para ciertos modelos de retardo o algunas topologías. Además, el tiempo que puede tomar la obtención de un resultado analítico puede que no se justifique, cuando sólo se busca tener una visión panorámica mediante algunos ejemplos selectos. En estos escenarios, parece indicado un estudio de simulación. La simulación es también una herramienta muy útil para enseñar o desarrollar algoritmos, debido a sus capacidades para visualizar y construir animaciones de una ejecución.

Contribución

En este trabajo se presenta una plataforma de software que ofrece un ambiente para la implantación, simulación y análisis de algoritmos distribuidos. Con nuestra herramienta, un programador puede desarrollar un algoritmo codificándolo en C++ y compilarlo con nuestras

bibliotecas. En un principio se pensó en una utilería para estudiar algoritmos distribuidos solamente, pero es posible aplicarla para representar otros modelos, tales como los agentes o los circuitos digitales. La razón de su amplio espectro de posibilidades radica en su diseño basado en un modelo de máquina de estados, que es una poderosa herramienta teórica del cómputo distribuido, con la que se describe la interacción entre entidades autónomas. En conjunto con una arquitectura muy estudiada en la comunidad de la DES. Nuestro enfoque de construcción permite ofrecer un producto con las siguientes características:

- Se trata de una plataforma flexible, de código abierto, que separa al algoritmo del entorno de comunicaciones sobre el que se despliega. Con ello se puede ejecutar el mismo algoritmo sobre diferentes topologías, sin modificar una línea de código.
- Se pueden simular eventos aleatorios, dejando al programador en libertad para especificar cualquier distribución de tiempo, ya sea en la duración de un paso de procesamiento dentro de un nodo, o en el retardo de transmisión de un mensaje sobre un canal.
- Se cuenta con un sencillo mecanismo de paso de mensajes, que el usuario puede extender para definir sus propias unidades de información.
- Cada entidad activa puede modelarse como un autómata simple o compuesto, dependiendo de la complejidad de su comportamiento.
- Se puede simular la ejecución simultánea del mismo algoritmo, para estudiar, por ejemplo, una operación de difusión iniciada desde diferentes puntos de una red.
- Se puede simular la ejecución simultánea de diferentes algoritmos, para estudiar cooperación u otro tipo de interacciones (e.g. sincronizadores o elección).
- Se puede utilizar para investigación o docencia. Tiene una interfaz gráfica de usuario (GUI) programada con Tcl/Tk con la que puede desplegarse una simulación animada. Las funciones de la GUI incluyen edición de topologías, ventana de visualización, ventana para despliegue de trazas y controles de la simulación (véase la fig. 5.1)

Trabajo relacionado

Cuando se considera simular un algoritmo distribuido, existen al menos dos caminos posibles. Por un lado se puede usar alguno de los poderosos ambientes que ya existen. También es posible diseñar un conjunto de macros o una capa que funcione encima de estas herramientas, con ello puede adaptarse a las condiciones particulares del estudio que se busca completar. Por ejemplo, [70] integra herramientas como OPNET y XPLOT para simular algoritmos para sistemas operativos distribuidos. Este enfoque aprovecha el conjunto de facilidades provistas por las herramientas sobre las que se construye (e.g. paquetes estadísticos, gráficas, distribuciones aleatorias, etc.). Nosotros tomamos un camino alternativo, diseñar una plataforma de simulación partiendo de cero. Esta decisión produjo una plataforma muy ligera, con la que es fácil adaptarse al dominio del problema y conseguir las características listadas con anterioridad.

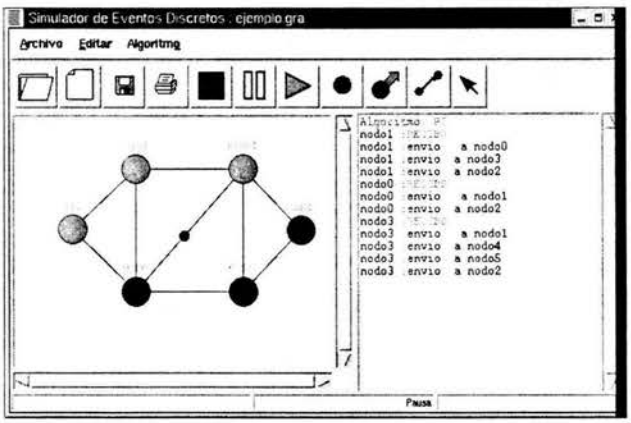

Figura 5.1: Simulación del algoritmo de propagación de información.

Las herramientas para simulación de algoritmos distribuidos se usan para dos tipos de aplicaciones: prueba o validación. La prueba se refiere a la evaluación del desempeño o propiedades cuantitativas. Nuestra plataforma pertenece a este tipo. Por otra parte, la validación se relaciona con propiedades cualitativas que garantizan una salida correcta y una terminación del algoritmo (correctness and liveness). Merecen atención especial los siguientes trabajos.

La plataforma para algoritmos distribuidos (DAP), actualmente en desarrollo, está pensada como una herramienta para desarrollo, simulación y prueba de algoritmos distribuidos para el soporte de redes guiadas o móviles. Está escrita en C++ como una colección de bibliotecas. Sólo se sabe de un par de aplicaciones desarrolladas con su ayuda.

La plataforma para sistemas distribuidos (DSP) se diseñó para ofrecer un modelo bien conocido de sistema distribuido, a fin de ofrecer un marco unificado para el diseño, análisis, verificación y programación de algoritmos distribuidos y protocolos. En DSP, cada proceso (nodo) es una máquina de estados finitos con una tabla local de transiciones. Se requiere que los usuarios aprendan un lenguaje algorítmico y un lenguaje para describir topologías.

Neko es un ambiente para simular y desarrollar prototipos de algoritmos distribuidos. Los usuarios pueden construir sus aplicaciones codificándolas en JAVA, para después instalarlas sobre escenarios reales. Sin embargo, se encuentran limitados por el número mínimo de sistemas de red que se soportan.

Por cuanto concierne a herramientas como el simulador para autómatas I/O, el ambiente de simulación SPIN/PROMELA, o el simulador VESTA, por mencionar sólo algunos de los más importantes, se les considera orientados a la validación automática y no guardan una fuerte relación con nuestro trabajo.

El resto del capítulo incluye las siguientes secciones. La sección 5.2 presenta una vista panorámica de los modelos formales soportados por el simulador. La sección 5.3 describe la arquitectura del simulador usando un enfoque orientado a objetos. La sección 5.4 muestra

un ejemplo de utilización. Por último, la sección 5.5 incluye las conclusiones y las direcciones de trabajo futuro.

5.2 El modelo subyacente

Este trabajo aborda sistemas que cambian su estado de una forma discreta. A continuación se definen brevemente los conceptos relevantes que serán utilizados en lo sucesivo:

- Sistema. Una colección de entidades que interactúan a lo largo del tiempo para alcanzar una o varias metas.
- Modelo. La representación en abstracto de un sistema, usualmente contiene relaciones matemáticas y/o lógicas que describen al sistema en término de entidades, atributos, conjuntos, eventos, actividades y retardos.
- Estado. Una colección de variables que contienen toda la información necesaria para describir al sistema en cualquier momento.
- Entidad. Cualquier objeto o componente del sistema que requiere representación explícita en el modelo.
- Atributos. Las propiedades de una entidad dada.
- Conjunto. Una colección de entidades asociadas (permanente o temporalmente).
- Evento. Una acción instantánea que cambia el estado del sistema.
- Actividad. Una acción caracterizada por un lapso de tiempo de duración específica (también puede definirse a partir de una distribución estadística).
- Retardo. Un lapso de tiempo de longitud no especificada.

La simulación de eventos discretos, o DES, es la construcción de modelos para sistemas cuyas variables de estado cambian en puntos discretos del tiempo. Los modelos se analizan mediante métodos numéricos; es decir que los modelos se "corren" para crear una historia artificial del sistema, basada en suposiciones acerca del sistema. Las observaciones se reúnen para su análisis y para estimar las medidas de desempeño del sistema real que se caracteriza.

Para los fines de nuestra investigación, las máquinas de estados finitos son la herramienta teórica indicada, puesto que sirven para modelar directamente la interacción entre entidades que se da con el intercambio discreto de información. De entre las muchas opciones disponibles, las que mejor se adaptan a nuestros objetivos son las *máquinas comunicantes de estados finitos* y los *autómatas I/O*, esto es, el tipo de entidades con las que buscamos trabajar han sido descritos usando estos formalismos:

Una *máquina comunicante de estados finitos* puede describirse como un "demonio" abstracto que acepta símbolos de entrada, genera símbolos de salida y cambia su estado interno de acuerdo con un plan predefinido [71]. Los demonios pueden comunicarse a través de canales FIFO de capacidad acotada que mapean la salida de una máquina sobre la entrada de otra.

Un *autómata I/O* modela los componentes de un sistema distribuido que interactúan con otros componentes del sistema. Es un tipo muy simple de máquina de estados en la que las transiciones están asociadas con *acciones* designadas. Las acciones se clasifican como *entradas*, *salidas* o *internas*. Las primeras dos se usan para la comunicación entre el autómata y su ambiente, mientras que las acciones internas sólo son visibles para el autómata mismo. Se supone que las acciones externas no están bajo el control del autómata - ocurren desde el exterior - mientras que el autómata mismo especifica las acciones internas y de salida, que deben efectuarse.

Es posible representar sistemas complejos con el enfoque de autómatas I/O, recurriendo a la operación de composición, para modelar componentes individuales del sistema. La composición identifica acciones con la misma designación, que ocurren en diferentes autómatas compuestos. Cuando algún autómata compuesto efectúa un paso que implica a la acción π, así lo hacen todos aquellos componentes (autómatas) que tienen a π en su rúbrica, esto es, en la descripción de sus acciones de entrada, salida o internas.

Nuestra plataforma de simulación puede trabajar con sistemas cuyas entidades pueden modelarse mediante máquinas de estado, como las mencionadas anteriormente. Por otro lado, para la interacción entre estas entidades activas se recurre al *modelo de red asíncrona*. Se trata de una red de comunicaciones punto a punto, descrita mediante una *gráfica de comunicaciones* $G = (V, E)$. El conjunto de nodos V representa los componentes activos de la red y el conjunto de enlaces E representa los canales bidireccionales, libres de interferencia, mediante los cuales se comunican los nodos. Cada nodo procesa los mensajes de sus vecinos, efectúa un cómputo local y envía mensajes a sus vecinos. Todos los mensajes son de longitud acotada y sólo pueden transportar una cantidad finita de información. Cada mensaje transmitido sobre un canal, llega a su destino al cabo de un retardo finito. Este es un modelo ampliamente aceptado entre la comunidad de los sistemas distribuidos [68].

Usando nuestra herramienta es posible estudiar la ejecución de uno o varios algoritmos, en forma simultánea. En el segundo caso, puede tratarse de varias instancias de un mismo algoritmo o bien, instancias diferentes de algoritmos complementarios. Al mismo tiempo, es muy sencillo ejecutar un mismo algoritmo sobre gráficas diferentes, ya que éstas se consideran condiciones iniciales del experimento que pueden proporcionarse durante su arranque, mediante un archivo que almacena la lista de adyacencias que caracteriza a una gráfica en particular.

5.3 La arquitectura del simulador

El simulador que presentamos fue diseñado y construido usando un enfoque orientado a objetos y tiene una fuerte influencia del trabajo de Preiss [8]. La reutilización es un concepto clave de nuestro diseño con el que fue posible construir un sistema "ultraligero", con apenas 2,000 líneas de código escrito en C++, aproximadamente.

Las piezas elementales de nuestra construcción son dos clases parametrizadas o templetes: "Linklist<>" y "AVLTree<>". Una instancia de "Linklist<>" puede funcionar como cola, pila o lista ordenada, cuyos items pueden ser tipos escalares simples o, incluso, instancias de cualquier clase. En tanto, una instancia de "AVLTree<>" es un objeto que puede manejar

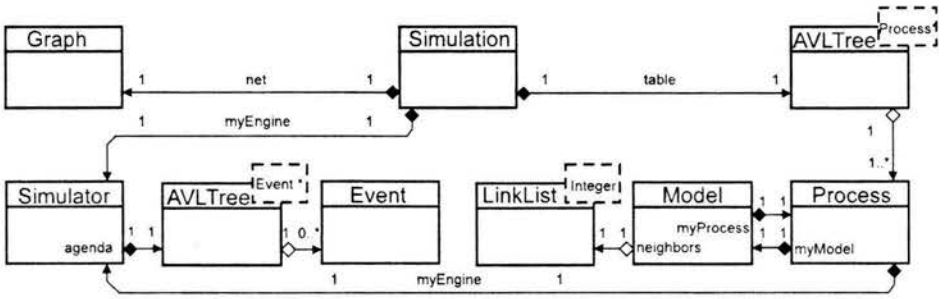

Figura 5.2: Diagrama de clases.

eficientemente miles de items que pueden generarse o accesarse en forma aleatoria.

El resto de las clases con que se construyó el sistema son las siguientes (véase la fig. 5.2):

1. *Event*: Representa un paquete de información intercambiado entre componentes.
2. *Model*: Representa la rutina para la atención de eventos, de una máquina de estados finitos. El estado global del sistema se codifica a través de los modelos de sus componentes. También efectúan funciones de reporte y generación de trazas.
3. *Process*: Representa la entidad activa a cargo de un modelo.
4. *Graph*: Representa la gráfica de comunicaciones. Puede ser una gráfica ponderada.
5. *Simulator*: Representa al calendarizador de eventos, también efectúa tareas de reloj.
6. *Simulation*: Representa al administrador que establece los canales entre procesos y coordina su comunicación. También efectúa funciones de inicialización.

Una instancia de la clase "Simulation", denominada *myExperiment*, consta de 3 atributos: una gráfica de comunicaciones, una tabla de procesos y un calendarizador o despachador de la simulación. En cada paso, *myExperiment* invoca al despachador para recoger al próximo evento que debe atenderse. Enseguida, determina la identidad del proceso al que va dirigido y lo busca en su tabla. Cuando ha recibido el evento que le corresponde, el proceso consulta su modelo para determinar las acciones que deben tomarse como respuesta. Si se requiriera, el modelo invocará los métodos de transmisión de su proceso, los que a su vez se traducirán en los métodos de inserción sobre la agenda del despachador, para programar eventos futuros(véase la fig. 5.3).

5.4 Usando el simulador

Para trabajar con nuestra herramienta, el desarrollador codifica en C++ su algoritmo distribuido, lo compila usando gpp (i.e. GNU C++) y lo liga con las bibliotecas de la plataforma

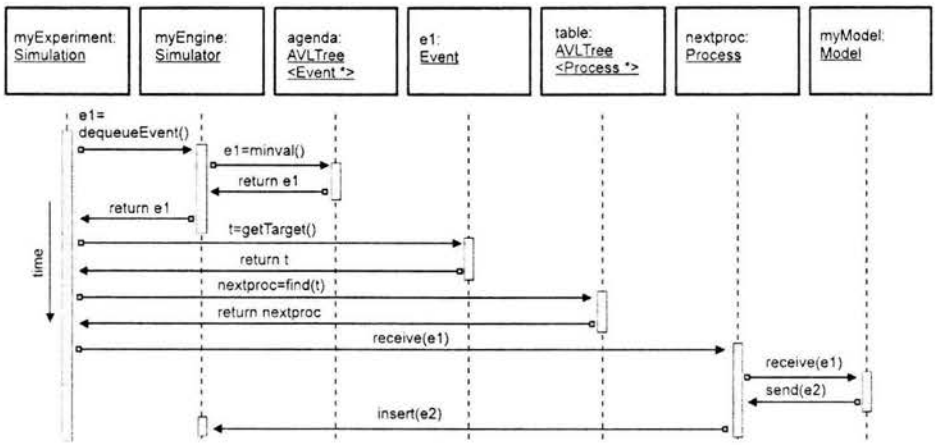

Figura 5.3: Diagrama de secuencia.

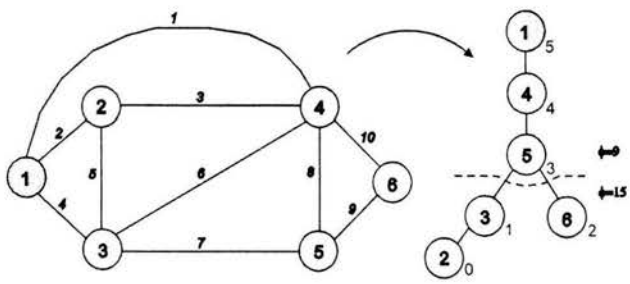

Figura 5.4: Solución de SP con 1 hormiga, ciclo 1 de 1

para producir un archivo ejecutable. Este último interactúa con la GUI para desarrollar la visualización. Las funciones de la GUI son: edición de topologías, visualización, despliegue de trazas y control de velocidades.

La figura 5.4 muestra una gráfica sobre la que se ejecutó el algoritmo desarrollado en el capítulo anterior. Para esta instancia particular del problema SP, deseamos particionar la gráfica en subconjuntos de 3 elementos cada uno, de forma que se optimicen las restricciones que definen a SP. Se sabe que un agente, u hormiga, construye un recorrido DFS aleatorio sobre la gráfica en que se simula el algoritmo. En el ejemplo, el recorrido se inicia en el nodo 1. El número a la derecha de cada nodo indica el orden en que se le exploró. De esta manera los primeros 3 nodos explorados quedan adscritos al primer subconjunto y los siguientes 3 pertenecen al segundo subconjunto.

Un autómata a cargo del algoritmo HORMIGA se compone de varios autómatas elementales. Primeramente, se echa a andar el algoritmo de enrutamiento Ford-Fulkerson(FORD) [69], al término del cual, cada nodo sabe el costo de la ruta óptima hasta cualquier otro punto

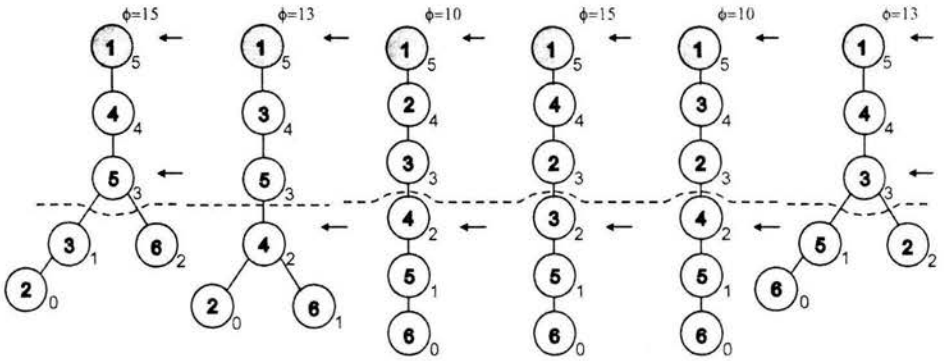

Figura 5.5: Solución de SP con 6 hormigas, ciclo 1 de 3.

de la red. Enseguida se ejecutan, simultáneamente, varias instancias del algoritmo de recorrido aleatorio, que es una modificación del algoritmo de búsqueda en profundidad [13]. Cada recorrido construye una solución heurística del problema SP, como se describe en el párrafo anterior. Cuando todos los recorridos se han completado, se realiza una evaluación de la calidad de sus soluciones y se efectúa una operación de premiación, por medio del algoritmo de propagación de información [68]. En tanto no se alcance el criterio de paro, se vuelven a ejecutar todas las instancias del recorrido aleatorio y se realiza la premiación.

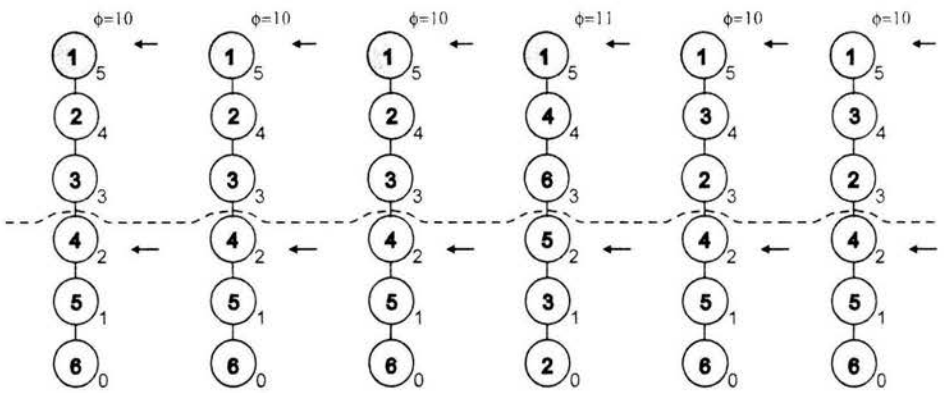

Figura 5.6: Solución de SP con 6 hormigas, ciclo 3 de 3

La figura 5.5 muestra otra simulación, esta vez con 6 agentes que construyen sus recorridos sobre la misma gráfica del ejemplo anterior. Al término del primer ciclo, puede observarse que se han producido 6 soluciones diferentes, cuya calidad se mide por el mayor diámetro, ϕ, de todas las subgráficas inducidas por un mismo agente. En contraste, la figura 5.6 muestra los recorridos construidos al cabo del tercer ciclo.

5.5 Trabajo futuro

Hemos presentado una herramienta DES flexible, ligera y portable. Con ella, los programadores no requieren aprender un nuevo lenguaje para desarrollar sus aplicaciones. Además, ofrecemos una importante colección de algoritmos distribuidos que pueden usarse como elementos de construcción para implementar algoritmos más complejos (ya que la plataforma soporta composición y cooperación).

La herramienta puede entenderse como compuesta de dos partes: las utilerías DES y la GUI, respectivamente. El tamaño de la primera es muy pequeño gracias al uso intensivo de templetes que simplifican la programación y reducen el código. Por lo que respecta a la segunda parte, hemos aprovechado un conjunto de bibliotecas gratuitas de mucha calidad, como Tcl/Tk, para acelerar la construcción de la interfaz gráfica. Estas decisiones nos permitieron obtener los primeros resultados en un plazo muy breve. Sin embargo, si buscamos que la comunidad acepte nuestro desarrollo, creemos que los usuarios desearían trabajar con él desde un navegador web.

Como se mencionó anteriormente, los principios de diseño de nuestra plataforma están inspirados en el simulador PARSIMONY de Bruno Preiss. Esto significa que varias de nuestras clases son versiones simplificadas de las clases de PARSIMONY. En contraste, hemos decidido un punto de equilibrio diferente entre desempeño y complejidad. Ello nos permite ofrecer un código libre que otros pueden adaptar con facilidad a sus necesidades, de manera incremental.

Para el futuro planeamos construir un simulador distribuido de eventos discretos, con el que se pueda estudiar sistemas de gran escala. Para ello, pensamos que es necesario desarrollar módulos de tolerancia a fallas y cómputo móvil.

Capítulo 6

Conclusiones

Hemos desarrollado un conjunto de herramientas básicas para la construcción de un sistema de almacenamiento tolerante a fallas de paro. Dicho conjunto incluye: la definición de los esquemas de almacenamiento, las heurísticas para resolver el problema de escalabilidad y la plataforma de experimentación. Una serie de extensiones interesantes y preguntas abiertas han resultado de esta investigación, que tienen que ver con la construcción de sistemas de archivos, bases de datos distribuidas, y la implantación de sistemas DDES de alto desempeño.

En ese momento el viejecillo se disolvió en la clara mañana. Pero el punto rojo siguió corriendo y saltando entre los rieles, imprudentemente, al encuentro del tren.

J.J. Arreola en "El guardagujas"

Al iniciar este trabajo sugeríamos evaluar las capacidades de una red de telecomunicaciones. En el escenario supuesto, se tenía un sistema que requería una simulacion DDES de larga duración y soportada por muchas máquinas. Entonces planteamos el riesgo de falla de paro, es decir, afirmamos que algún componente de la simulación podía experimentar una degradación de sus funciones, al punto de considerarlo fuera de servicio. A lo largo de la investigación desarrollamos un colección de soluciones o procedimientos de contingencia que incorporan tres tipos de redundancia: de tiempo, de recursos y de información. La redundancia en tiempo significa restaurar la ejecucion distribuida hasta un estado global de su "historia" reciente, a partir del cual corre a cargo de los componentes que siguen en servicio. La redundancia de recursos significa que la operación de almacenamiento de los estados globales es soportada por más de un bloque de bodegas, definidos estos últimos mediante un esquema de almacenamiento. Finalmente, la redundancia de información significa que el almacenamiento puede acompañarse por un procesamiento que proteja la integridad de la información depositada en cada bloque. Entre las ventajas de nuestra propuesta pueden subrayarse:

1. La disponibilidad de la información, o sea la capacidad para responder con rapidez a una solicitud de servicio, debido a que la información no está depositada en un sólo sitio y eso reduce los cuellos de botella. Puesto de otro modo, si se requiere recuperar un estado global almacenado, la velocidad a la que se reconstruye es proporcional al número de bodegas que forman el bloque donde se guardó.

2. El balance que se consigue entre los recursos dedicados a la acción preventiva y los recursos dedicados a la tarea principal. Visto de otra forma, podemos garantizar que cualquier componente de la ejecución distribuida debe distraerse de su tarea principal (la simulación que está ejecutando por ejemplo), durante una fracción reducida del ciclo de almacenamiento en la cual aporta una cantidad acotada de su espacio.

3. La posibilidad de configurar el número crítico de fallas, lo cual significa que el constructor queda en libertad para fijar la confiabilidad de su sistema según la importancia que asigne a la continuidad de las operaciones. Esto quiere decir que puede diseñar un esquema de almancenamiento que tolera hasta 3 fallas, por ejemplo, antes de que ocurra una catástrofe y se anule el procedimiento de contingencia.

4. La escalabilidad, es decir, la facilidad para implantar un procedimiento de contingencia con un costo linealmente proporcional al número de componentes sobre el que se despliega. Como vimos, los costos en capacidad de almacenamiento y transporte pueden dispararse cuando crece el tamaño del sistema, a menos que se particione en regiones más pequeñas y se implante una réplica del mismo esquema de almacenamiento sobre cada una de ellas.

Consideramos que un valor extra de esta investigación es la posibilidad de extender los resultados para su implantación en otras situaciones. En nuestra opinión, esto indica que fuimos capaces de reconocer problemas importantes del cómputo distribuido y desarrollar soluciones de aplicación general. En ese sentido, esta tesis puede interpretarse como los planos básicos de un proyecto de largo plazo.

6.1 Lo que aprendimos

Cuando iniciamos este trabajo, buscábamos un procedimiento de contingencia para hacer frente a fallas de paro en sistemas DDES. Propusimos desarrollar métodos de almacenamiento distribuido como solución de fondo. Aunque este enfoque ya se había aplicado en la construcción de bases de datos, nuestra contribución tiene características que la hacen original: 1) proponemos que la información que se almacene se refiera al estado de una ejecución y no a los datos con que ésta trabaja, 2) a diferencia de una base de datos, no se guarda información replicada, sino que se almacenan un número fijo estados globales sucesivos sobre diferentes subconjuntos o bloques de almacenamiento, con lo que se consigue una solución equitativa y económica, que no compromete la tarea principal del sistema que se busca proteger, 3) el procedimiento con que definimos la colección de bloques es revolvente y se presta para una implantación distribuida eficiente. Esto quiere decir que se requiere una cantidad mínima de información para mantener la agenda de almacenamiento. De hecho, sólo se necesita saber la composición del último bloque designado, para determinar la identidad de los almacenes que participan en el siguiente bloque.

Por otro lado, hemos demostrado que las plataformas DES y DDES son herramientas que facilitan la codificación de heurísticas basadas en el concepto de agentes. Al mismo tiempo, reconocimos una forma distribuida del problema de partición (SP) y, por medio de la DES,

también demostramos que un sistema distribuido puede aplicar sus propias capacidades para la solución del problema. Nos parece que este enfoque tomará impulso con el desarrollo de nuevas aplicaciones cooperativas sobre Internet, como el caso de las llamadas aplicaciones P2P.

Finalmente, hemos construido una plataforma DES adhoc sobre la que pueden efectuarse experimentos relacionados con sistemas distribuidos. Creemos que la flexibilidad de nuestra herramienta está basada en el soporte de los modelos de autómatas y la facilidad para definir las conexiones entre los mismos. Esto será la clave para extender su aplicaciones en el futuro. Por el momento, ya fue utilizada para simular agentes (hormigas) y protocolos.

6.2 Aplicaciones y trabajo futuro

Los esquemas de almacenamiento puede adaptarse y emplearse como sistemas de archivos y bases de datos. Imaginemos, por ejemplo, un archivo que deseamos almacenar con esta tecnología. Primeramente, lo dispersamos usando el algoritmo de Rabin (sec. 3.7), luego determinamos el número de bloque sobre el que se deposita y guardamos cada uno de los dispersos en las bodegas que forman el bloque correspondiente. Por otro lado, si somos capaces de construir un mecanismo de acceso en lecto-escritura sobre los dispersos, entonces estaremos en posibilidad de gestionar una base de datos distribuida (BDD). En contraste con una BDD convencional, nuestra solución no requiere almacenar réplicas, ya que sólo se tiene un conjunto de datos, pero dispersados y almacenados sobre un bloque de bodegas. Adicionalmente, si utilizamos componentes de refacción, esto posibilita la construcción de una base de datos tolerante a fallas. Esto es, si una bodega cae en paro, podemos reemplazarla y usar la información depositada en sus compañeras de bloque para recuperar los contenidos perdidos y guardarlos en la bodega de refacción que toma su lugar.

Por su parte, las soluciones heurísticas de tipo hormiga pueden ser la base para construir sistemas de almacenamiento distribuido que puedan desplegarse y (re)configurarse de manera automática. Supongamos un conjunto de 16 computadoras conectadas entre sí mediante una red de alta velocidad y sobre las que deseamos ejecutar una simulación DDES. Un sistema automático sería capaz de particionar al conjunto en dos particiones de 8 máquinas (por ejemplo), usando para ello el algoritmo que hemos presentado. Con esto se podría instalar sobre cada partición un esquema de almacenamiento con 7 bodegas activas y 1 de refacción.

Otro problema recurrente del cómputo distribuido es el cálculo de número óptimo de máquinas sobre las que se consigue la máxima velocidad de ejecución. Dicho problema también admite un tratamiento con el mismo enfoque heurístico con que trabajamos SP. Las circunstancias en que se presenta son las siguientes: se sabe que existen algunos tipos de ejecuciones que no siempre se aceleran cuando se les acomoda sobre un mayor número de computadoras. Por ejemplo, supongamos otra vez una aplicación DDES que puede ejecutarse sobre 4 ó 5 máquinas. De primera instancia parecería que puede conseguirse un desempeño superior en el segundo caso, sin embargo, es posible que entre los componentes del modelo bajo estudio exista una fuerte interrelación. Entonces, si se les simula sobre máquinas diferentes, podrían dar lugar a un intercambio de mensajes muy pesado entre las máquinas a cargo. Eventualmente esto derivaría en cuellos de botella y retardos. En tanto, también es posible que el

modelo pueda acomodarse sobre 4 máquinas de tal suerte que los componentes fuertemente relacionados se encuentren en un mismo sitio mientras que los componentes débilmente relacionados se encuentren en sitios diferentes. En una ejecución distribuida, este problema se presenta al inicio de las operaciones y durante su restauración, después de que ocurre una falla de paro. En suma, creemos que los problemas combinatorios que surgen de la administración de un sistema distribuido pueden resolverse aplicando su propia capacidad de cálculo, mediante nuestra propuesta metodológica.

Por cuanto concierne a las posibilidades futuras de la plataforma de simulación, consideramos que esta abre un vasto conjunto de aplicaciones que, por sí solas, pueden convertirse en una línea de investigación. Por principio, existen muchos algoritmos para los que se conocen solamente medidas holgadas que acotan su complejidad y se ignora su desempeño sobre escenarios particulares, i.e. topologías, retardos, fallas. Igualmente, es muy interesante cómo la experimentación puede servir para inferir y depurar las propiedades de un algoritmo, durante su fase de concepción.

Por otro lado, además de los sistemas distribuidos, hay otras ramas de la computación que pueden sacar provecho de la simulación. La realidad virtual requiere de sistemas DDES de alto rendimiento cuando se trata de construir, por ejemplo, aparatos para entrenar pilotos, tripulaciones completas o fuerzas de tarea.

Por último, existe un número creciente de grupos de investigación y desarrollo que ven en la simulación de eventos discretos a un recurso invaluable que tienden a incorporar en su trabajo cotidiano. Prácticamente cualquier dominio de conocimiento cuyos modelos cambian de estado en instantes discretos del tiempo, puede beneficiarse con esta herramienta: las comunicaciones, el transporte, la investigación de operaciones y la economía, son sólo algunos ejemplos.

En este contexto, la meta debe ser incorporar todas las propuestas de nuestra investigación, para concretar un producto flexible y sencillo de operar, capaz de desplegarse sobre diferentes ambientes cooperativos, como los clusters, los entornos de cómputo móvil o la propia Internet.

Índice

Bibliografía

[1] R. Marcelín-Jiménez and S. Rajsbaum. Cyclic strategies for balanced and fault-tolerant distributed storage. In R. De Lemos and T. Weber, editors, *1st. Latin-American Symposium on Dependable Computing*, LNCS, Sao Paulo, Brazil, Oct. 2003. Springer Verlag. accepted paper.

[2] R. Marcelín-Jiménez. Distributed site partitioning. In J. Sibeyn, editor, *10th. Int. Colloquium on Structural Information and Communication Complexity (SIROCCO 10)*, pages 249–257, Umeå, Sweden, Jun. 2003. Carleton Scientific.

[3] R. Marcelín-Jiménez, R. Esquivel-Villafaña, and S. Rajsbaum. A flexible simulator for distributed algorithms. In *Encuentro Nacional de Computación, ENC03*, Tlaxcala, México, Sep. 2003. accepted paper.

[4] P. Yianilos and S. Sobti. The evolving field of distributed storage. *IEEE Internet Computing*, pages 35–39, Sep/Oct 2001.

[5] M. Dorigo. *Optimization, Learning and Natural Algorithms.* PhD thesis, Dept. of Electronics, Politecnico di Milano, Italy, 1992.

[6] A.N. Mourad, K.W. Fuchs, and D.G. Saab. Site partitioning for redundant arrays of distributed disks. *Journal of Parallel and Distributed Computing*, (33):1–11, 1996.

[7] D. Peleg. *Distributed Computing A Locality-Sensitive Approach.* SIAM, 2000.

[8] B.R. Preiss. The parsimony project website. Technical report, University of Waterloo, http://www.pads.uwaterloo.ca/Bruno.Preiss, 1999.

[9] F.J. MacWilliams and N.J. Sloane. *The Theory of Error-correcting codes.* North-Holland, 8th edition, 1993.

[10] V. Tonchev. *Combinatorial Configurations Designs, Codes, Graphs.* Longman Scientific and Technical, 1988.

[11] D.R. Stinson. An introduction to combinatorial designs. Technical report, University of Waterloo, Dept. of Combinatorics and Optimization, Dec. 1999.

[12] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization.* McGraw Hill, 1999.

[13] I. Cidon. Yet another distributed depth-first-search algorithm. *Information Processing Letters*, 26:301–305, Jan. 1988.

[14] M.O. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the ACM*, 36(2):335–348, Apr. 1989.

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

[16] J. Misra. Distributed discrete-event simulation. *Computing Surveys of the ACM*, 18(1):39–65, Mar. 1986.

[17] Y. Lin and P.A. Fishwick. Asynchronous parallel discrete event simulation. *IEEE Trans. on systs., man and cibernetics*, 1995. ficha tomada de la web.

[18] A. Ferscha. Parallel and distributed simulation of discrete event systems. In *Handbook of Parallel and Distributed Computing.* McGraw Hill, 1995.

[19] D.M. Nicol, O. Balci, R.M. Fujimoto, P.A. Fishwick, P. L'Ecuyer, and R. Smith. Strategic directions in simulation research. In P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, editors, *Winter Simulation Conference*, pages 1509–1520, 1999.

[20] R. Shorey, A. Kumar, and K.M. Rege. Instability and performance limits of distributed simulators of feedforward queueing networks. *ACM Trans. on Modeling and Computer Simulation*, 7(2):210–238, Apr. 1997.

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

[22] C.D. Carothers, R.M. Fujimoto, and P. England. Effect of communication overheads on time warp performance. In *Winter Simulation Conference*, 1994. ficha tomada de la web.

[23] B.R. Preiss, W.N. Loucks, and I.D. MacIntyre. Effects of the checkpoint interval on time and space in time warp. *ACM Trans. on Modeling and Computer Simulation*, 4:223–253, Jul. 1994.

[24] A. Ferscha. Adaptive time warp simulation of timed petri nets. *IEEE Trans. on Software Engineering*, 25(2):237–257, Mar. 1999.

[25] R.D. Samir and R. Fujimoto. Adaptive memory management and optimism control in time warp. *ACM Trans. on Modeling and Computer Simulation*, 7(2):239–271, Apr. 1997.

[26] S. Srinivasan and P.F. Reynolds. Elastic time. *ACM Trans. on Modeling and Computer Simulation*, 8(2):103–139, Apr. 1998.

[27] W.M. Wonham. Notes on control of discrete-event systems. Technical report, University of Toronto, Dept. Electrical and Computer Eng., Jul. 2003.

[28] O. Babaoglu and K. Marzullo. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In S. Mullender, editor, *Distributed Systems*, chapter 4, pages 55–96. ACM, 2nd edition, 1993.

[29] C. Lavault. *Evaluation des Algorithmes Distribues.* Hermes, Paris, France, 1995.

[30] L. Lamport and N. Lynch. Distributed computing: Models and methods. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 18, pages 1158–1199. Elsevier and MIT Press, 1990.

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

[32] Y. Deswarte. Tolérance aux fautes, sécurité et protection. In R. Balter et. al., editor, *Construction des Systèmes d'exploitation Répartis.* INRIA, Paris, France, 1991.

[33] JC. Laprie. Dependability: Basics concepts and terminology. In *Dependable Computing and Fault Tolerant Systems*, volume 5. Spriger-Verlag, 1992.

[34] B. F. Schneider. What good are models and what models are good? In S. Mullender, editor, *Distributed Systems*, chapter 2, pages 17–26. ACM, 2 edition, 1993.

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

[36] E. Levy and A. Silverschatz. Distributed file systems: Concepts and examples. *ACM Computing Surveys*, 22(4):321–374, Dec. 1990.

[37] T. Anderson et. al. Serverless network file systems. In *15th. ACM Simposium on Operating Systems Principles*, pages 109–126. ACM, 1995.

[38] C. Thekkath, T. Mann, and E. Lee. Frangipani: A scalable distributed file systems. In *16th. ACM Simposium on Operating Systems Principles*, pages 224–237. ACM, 1997.

[39] A. Crespo and H. García-Molina. Archival storage for digital libraries. In *3rd. Int. Conf. on Digital Libraries*, pages 69–78. ACM, 1998.

[40] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale peer-to-peer storage utility. In *18th. ACM Simposium on Operating Systems Principles*, pages 224–237. ACM, 2001.

[41] A.V. Goldberg and P.N. Yianilos. Towards an archival intermemory. In *Int. Forum on Research and Technology Advances in Digital Libraries*, pages 147–156. IEEE, 1998.

[42] Y. Chen et. al. A prototype implementation of archival intermemory. In *4th. ACM Conf. on Digital Libraries.* ACM, 1999.

[43] W.J. Bolosky et.al. Feasibility of a serverless distributed file system deployed on an existing set of pcs. In *Int. Conf. on Measurement and Modeling of Computer Systems*, pages 34–43. ACM, 2000.

[44] S. Rhea et. al. Maintenance-free global data storage. *IEEE Internet Computing*, pages 40–49, Sep/Oct 2001.

[45] M. Beck and T. Moore. The internet2 distributed storage infrastructure project: An architecture for internet content channels. *Computer Networks and ISDN Systems*, 30(22/23):2141–2148, 1998.

[46] I. Clarke et. al. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. ICSI Workshop on Design Issues in Anonymity and Unobservability*. International Computer Science Institute, 2000.

[47] M. Waldman, A.D. Rubin, and L.F. Cranor. Publius: A robust, tamper-evident, censorship-resistant web publishing system. In *Proc. 9th. Usenix Security Symp.*, pages 59–72. Usenix Assoc., 2000.

[48] R. Dingledine, M.J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In *Proc. Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[49] A. Azagury, M.E. Factor, and J. Satran. Point-in-time copy: Yesterday, today and tomorrow. In *19th IEEE Symposium on Mass Storage Systems*, pages 259–270. IEEE, 2002.

[50] M. Chandy and L. Lamport. Distributed snapshots: Determining global states in distributed systems. *ACM Trans. on Computer Science*, 3(1):63–75, 1985.

[51] J.B. Fraleigh. *Álgebra Abstracta*. Addison-Wesley Iberoamericana, 1987.

[52] C.J. Colbourn, J.H. Dinitz, and D.R. Stinson. Applications of combinatorial designs to communications, cryptography, and networking. Technical report, University of Vermont, 2000.

[53] R. Bhagwan, D. Moore, S. Savage, and G. Voelker. Replication strategies for highly available peer-to-peer storage. In *International Workshop on Future Directions in Distributed Computing*, 2002.

[54] D. Kotz. Introduction to multiprocessor i/o architecture. In R. Jain, J. Werth, and J. C. Browne, editors, *Input/Output in Parallel and Distributed Computer Systems*. Kluwer Academic Publishers, 1996.

[55] P. Berenbrink, A. Brinkmann, and C. Scheideler. Design of the presto multimedia storage network. In *International Workshop on Communication and Data Management in Large Networks*, pages 2–12, 1999.

[56] A. Russell and A. A. Shvartsman. Distributed computation meets design theory: Local schedulling for disconnected operations. *Bulletin of the EATCS*, pages 120–131, Jun. 2002.

[57] D. Peleg and A. Wool. The availability of quorum systems. *Information and Computation*, 123(2):210–223, Dec. 1995.

[58] V.K. Garg and J. Ghosh. Repeated computation of global functions in a distributed environment. *IEEE Trans. on Parallel and Distributed Systems*, 5(8):823–834, Aug. 1994.

[59] N. Lynch. *Distributed Algorithms*. Morgan Kaufman Pub., 1996.

[60] D.L. Kreher and D.R. Stinson. *Combinatorial Algorithms*. CRC Press, 1998.

[61] L. M. Batten. *Combinatorics of finite geometries*. Cambridge University Press, 1997.

[62] P. Dembowski. *Finite Geometries*. Springer-Verlag, 1968.

[63] J.N. Cederberg. *A Course in Modern Geometries*. Springer-Verlag, 1989.

[64] A. A. Bruen and R. Silverman. Acs and blocking sets ii. *European Journal of Combinatorics*, 8(4):351–356, 1987.

[65] A.E. Brouwer and A. Schrijver. The blocking number of an affine space. *Journal of Combinatorial Theory ser. A*, 24(2):251–253, 1978.

[66] W. Gutjahr. A generalized convergence result for the grap-based ant system metaheuristic. Technical Report 99-09, University of Vienna, 1999.

[67] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, Oct. 1985.

[68] A. Segall. Distributed network protocols. *IEEE Trans. on Information Theory*, 29(1):23–35, Jan. 1983.

[69] L. Ford and D. Fulkerson. Flows in networks. Technical report, Princeton University, 1962.

[70] S. Khanvilkar and S. M. Shatz. Tool integration for flexible simulation of distributed algorithms. *ACM Software Practice and Experience*, 31(14):1363–1380, Nov. 2001.

[71] G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.