Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Utilizando la calidad de las respuestas como política de distribución de la información de recursos en Grid Computing

Tesis presentada para optar al título de
Licenciada en Ciencias de la Computación

Paula Verghelet

Director: Dr. E. Mocskos

Buenos Aires, 2014

Resumen

La computación de alto rendimiento (HPC, High Performance Computing) es un término que engloba un conjunto de procedimientos y estrategias computacionales para resolver *eficientemente* problemas complejos que demandan un gran poder de cómputo. Una de las tecnologías que ha emergido y se ha consolidado en los últimos años es *Grid Computing*, que permite el acceso de manera remota a diversos recursos distribuidos geográficamente.

Debido a su naturaleza altamente distribuida, uno de los puntos más importantes a considerar dentro de las Grids es saber dónde están los recursos y su disponibilidad de manera de poder coordinar su utilización, siendo ésta una problemática común a Cloud Computing y a sistemas distribuidos en general. Cómo se obtiene y disemina la información sobre estos recursos es lo que se denomina *política de distribución de la información de recursos.*

Una clasificación posible de las políticas de distribución de la información es considerarlas centralizadas o descentralizadas, y a estas últimas a su vez como estructuradas o no-estructuradas. En esta tesis se estudian algunos aspectos de cuatro políticas de distribución de la información que resultan representativas dentro de esta clasificación: Jerárquica, Super Peer, Random y Best Neighbor. En particular para la política Best Neighbor, algunos resultados previos no fueron los esperados sino similares a los obtenidos al utilizar la política Random, a pesar de utilizar información histórica obtenida en base a consultas previas para guiar la política de distribución.

El principal aporte de esta tesis es la propuesta de una serie de mejoras a la política Best Neighbor de las que finalmente se obtuvieron dos variaciones que mostraron muy buena performance, siendo a su vez escalables y distribuidas.

High Performance Computing refers to a set of techniques and computational strategies devised to solve complex problems needing high computing power. One technology that emerged during last years is *Grid Computing*, which allows the access to globally distributed resources.

Due to its highly distributed nature, one of the key points to be considered in Grid Computing is knowing the resources location and their status. The way this information is obtained and distributed is known as *Resource Information Distribution Policy*.

One possible classification of the resource information distribution policies is if they are centralized or decentralized, which can be further divided in structured or unstructured. In this thesis, some relevant aspects of four policies are treated: Hierarchical, Super Peer, Random and Best Neighbor.

In previous works, Best Neighbor presented some unexpected results, similar to those obtained by Random policy, despite of being designed to use historical communication information to improve its performance.

The main contribution of this work is the proposal of improvements to Best Neighbour policy. Two variations are presented with good performance, scalability and fully distributed behavior.

Gracias a Laila y Esteban, porque esta paciencia no siempre es fácil.
Gracias a Juanma y a mis papis.
Gracias a mi director por llegar hasta acá.
Gracias a los compañeros y a los amigos, por no tener orden.
Gracias a los alfabetizadores por seguir.
Gracias a la gente del LSC y a la gente del DC por el apoyo.
Gracias al Gordo Arturo y la Gallega por encontrarse.

Para los que no necesitan fuerza para mover elefantes.

Índice general

1. INTRODUCCIÓN

La simulación numérica es una herramienta con la cual se estudian diversos fenómenos dentro de una gran variedad de ramas de la ciencia y de la ingeniería. Uno de los ejemplos paradigmáticos resulta ser la predicción del clima. Esta disciplina no solo necesita de modelos matemáticos que logren capturar la complejidad de los diversos procesos que lo gobiernan, sino de la tecnología y las técnicas para que estos puedan ser utilizados de manera de poder obtener resultados en tiempo y forma.

Por otro lado, durante los últimos años, el continuo aumento de la velocidad de los procesadores se ha detenido, siendo reemplazado por un aumento en la cantidad de núcleos de procesamiento cuya velocidad individual no presenta incrementos notables [1]. Este hecho ha vuelto indispensable el desarrollo de aplicaciones que realicen el cómputo de manera cooperativa, es decir, utilizando *paralelismo*.

Hoy en día, se cuenta con grandes clusters cuya tecnología no resulta mucho más específica que la que podemos encontrar en las computadoras de nuestros propios escritorios. Sin embargo, las supercomputadoras más poderosas del mundo[1] no son accesibles para la mayoría de los centros investigación o universidades. La computación de alto rendimiento (HPC, High Performance Computing) es un término que engloba un conjunto de procedimientos y estrategias computacionales para resolver *eficientemente* problemas complejos que demandan un gran poder de cómputo.

Una de las tecnologías que ha emergido y se ha consolidado en los últimos años es *Grid Computing*, que permite el acceso a prestaciones de supercómputo (como clusters para procesamiento o unidades de almacenamiento) de manera remota, así como la utilización de instrumentos de medición de alta complejidad que se encuentran on-line o aplicaciones de cómputo científico que usualmente se encuentran distribuidas geográficamente [2].

Como describe Ranjan et al. [3] y Navimipour et al. [4], pueden distinguirse distintas clases de Grids según su prioridad de uso:

- *Cómputo:* sistemas con gran capacidad de cómputo disponible para aplicaciones.
- *Datos:* proveen infraestructura para la síntesis de información de grandes repositorios de datos como bibliotecas.
- *Inalámbrica:* permite a usuarios de móviles y tecnología inalámbrica compartir recursos, servicios e información.
- *Multimedia:* provee infraestructura para aplicaciones multimedia de tiempo real.

Para compartir recursos como el acceso directo a computadoras, software y datos necesarios para resolver problemas de manera cooperativa, es necesario contar con reglas claras que definan cuidadosamente qué es lo que se comparte, quién tiene acceso a ello y bajo qué condiciones, como se describe en Foster et al. [5]. Una *Organización Virtual* (Virtual Organization, VO), es un conjunto de individuos y/o instituciones definido por dichas reglas.

Las VOs difieren en sus propósitos, alcance, tamaño, duración, estructura, comunidad que la conforma, etc. Por ejemplo: i) proveedores de servicio de aplicación, almacenamiento,

[1] Se pueden consultar en www.top500.com.

tiempo de cómputo y consultores contratados por una fábrica de autos para el desarrollo de escenarios de prueba, ii) un equipo de gestión de crisis y las bases de datos y simulación de sistemas que utilizan para planificar una respuesta a una situación de emergencia, iii) los miembros de un gran proyecto de colaboración internacional, de varios años, en física de altas energías [5].

El trabajo en investigación y desarrollo en Grid Computing ha producido protocolos, servicios y herramientas que permiten afrontar los desafíos que surgen de intentar contruir VOs de gran escala. Los *middlewares* son piezas de software diseñados para la administración de sistemas distribuidos. Permiten al usuario la utilización de los recursos distribuidos de manera transparente. Un middleware Grid es un conjunto de servicios y bibliotecas de software que da soporte a las aplicaciones Grid, y a la Grid como sistema en sí mismo, en cuanto a lo que concierne a seguridad, búsqueda de información, administración de recursos, administración de datos, comunicación, detección de fallas y portabilidad [6].

Debido a su naturaleza altamente distribuida, uno de los temas más importantes a considerar dentro de las Grids es saber dónde están los recursos y su disponibilidad de manera de poder coordinar su utilización. Es decir, resulta clave contar con información sobre el estado de los recursos disponibles que posee el sistema y que esta información se encuentre lo suficientemente actualizada, tal como se menciona en Iamnitchi et al. [7] y en Pipan [8].

Cómo se obtiene y distribuye la información sobre estos recursos es lo que se denomina *política de distribución de la información de recursos*. Esta problemática tiene puntos de contacto con la tecnología Cloud Computing [9–11] y, en general, con los sistemas distribuidos.

En la arquitectura descripta en Foster et al. [5], la capa de recursos (*resources*) es la que incluye los procesos de descubrimiento y localización de recursos. Los recursos en las Grids de Cómputo (de aquí en adelante, simplemente Grids) suelen tener dos clases de atributos: estáticos (sistema operativo, velocidad de procesador, capacidad de almacenamiento, ancho de banda de la red) y dinámicos (utilización del procesador, utilización de la memoria física, costo de uso, carga de la red, etc.) [3]. La planificación de una tarea o un *job* requiere la coordinación de recursos de distinta índole, para lo cual resulta necesario poder resolver consultas (queries) multidimensionales. Los superschedulers[2] utilizan para este fin la información sobre recursos que proveen los servicios de información.

En Grid Computing, los middlewares utilizados (i.e. `Globus Toolkit` [12]) delegan las funciones de descubrimiento y monitoreo de recursos a los GRIS (Grid Resources Information Services). Por ejemplo, dentro de Globus, se puede encontrar el componente MDS (Monitoring and Discovering Service), cada VO (Virtual Organization) asigna un recurso para hostear el servicio de información de recursos que luego es utilizada por MDS. Aplicaciones de monitoreo como `Ganglia` [13], tienen un tipo de servicio similar en el cual se suele asignar uno de los nodos de un cluster para recolectar la información de estado del resto del sistema.

Los servicios de indexación y recolección de información de estado de los recursos pueden ser organizados de varias maneras. Las propuestas iniciales incluían modelos centralizados y jerárquicos [3,14]. Estos modelos centralizados tienen como principal problema la baja tolerancia a fallas (*single point of failure*), así como una probable congestión de la red.

El crecimiento en tamaño y recursos de las Grids con el correr del tiempo trajo consigo

[2] Planificadores de tareas grid (literalmente: supercalendarizadores).

la búsqueda de métodos más eficientes y robustos, despertándose un creciente interés en lo desarrollado en el paradigma P2P [2–4, 7, 15, 16].

Uno de los principales aportes del trabajo de Ranjan et al. [3] es la definición de diferentes taxonomías para los distintos componentes involucrados en el monitoreo y descubrimiento de recursos. En un primer acercamiento identifica las siguientes:

- *Taxonomía de organización de los recursos:* Centralizada, Jerárquica o Descentralizada. Distinguen cuatro desafíos para los modelos de organización: escalabilidad, adaptabilidad, disponibilidad y manejabilidad.
- *Taxonomía de los atributos de los recursos:* Estáticos y dinámicos.
- *Taxonomía de las consultas de recursos:* Distingue dos tipos básicos de consultas, `look-up` y `update`. Introduce la idea de *Consulta* d-*dimensional puntual* y *Consulta* d-*dimensional por rango (o ventana)*. Estas, a su vez, pueden ser de matching exacto (todos los atributos), parcial (algunos de los atributos), de rango (se consulta por atributos con valores en un cierto rango), booleanas (alguno o todos los atributos satisfacen ciertas condiciones booleanas).

Dentro de los alcances de este trabajo, resulta de utilidad la *Taxonomía según la organización de recursos* (*GRIS organization*). Ranjan trata los P2P-GRIS de manera independiente, previo a lo cual analiza y clasifica los sistemas P2P. En esta taxonomía propone observar por separado la organización de red, es decir de qué modo se comunican los peers con los otros participantes, qué estructuras de datos utilizan y de qué manera se realiza el ruteo de las consultas *d*-dimensionales.

Por otro lado, en el reciente trabajo de Navimipour et al. [4], su taxonomía clasifica la organización de la información de recursos en cinco categorías: (i) Centralizada, (ii) Descentralizada, (iii) P2P, (iv) Jerárquica, (v) Agent-Based. A su vez, divide la categoría P2P en las siguientes subcategorías: Estructurado, No-estructurado, Híbrido y Super Peer.

Estas subcategorías son análogas a las propuestas por Ranjan et al. [3] en la taxonomía de organización de red de los sistemas P2P. Ambas propuestas difieren en la clasificación de algunos sistemas: por ejemplo el trabajo de Iamnitchi [7] es considerado en Ranjan et al. como P2P y en Navimipour et al. como descentralizado.

En Trunfio et al. [16] se analizan los sistemas Grids basados en P2P No-estructurado. También se presenta una revisión de los sistemas Grids más prometedores entre aquellos que comenzaron a utilizar técnicas P2P para facilitar el descubrimiento de recursos. De esta manera, se busca hacer una comparación cualitativa de los enfoques existentes y delinear conclusiones sobre sus ventajas y debilidades. En este artículo, se realiza un análisis comparando las comunidades P2P y GRID. Se plantea un escenario donde tanto P2P como Grid son sistemas para compartir recursos que tienen como fin último su aprovechamiento a través de múltiples dominios de administración. En referencia a las características que ambos tipos de sistema presentan, se menciona el comportamiento dinámico y la heterogeneidad de los componentes involucrados. Por otro lado, algunas diferencias esenciales se reflejan mayormente en el comportamiento de los usuarios involucrados, la naturaleza dinámica de los recursos Grid (en oposición al intercambio de archivos). Adicionalmente, en los sistemas Grid existen las aplicaciones *data critical* y/o *time critical*, tolerancia a fallas y requerimientos de seguridad, mientras que en los sistemas P2P se ofrece una operación del tipo *best effort* (mejor esfuerzo). Considera que ambos, Grid y P2P, constituyen un exitoso paradigma para compartir recursos. Describe los diferentes modelos P2P

(estructurado, no-estructurado e híbrido) y los compara en lo básico de su performance y capacidades. Una característica importante que destaca es la escalabilidad, dado que los sistemas P2P están globalmente extendidos. Supone para su estudio un sistema no estructurado P2P de N nodos con una red generada aleatoriamente de diámetro $O(log(N))$. Los sistemas no estructurados no destacan en este aspecto dado el tráfico generado por el *flooding*. Se han propuestos mejoras como el random walk [7], a expensas de incrementar el tiempo de respuesta y reducir el alcance (network coverage).

Los sistemas estructurados hacen posible la escalabilidad pero, por otro lado, son más difíciles de mantener bajo *high churn* (alta tasa de deserción), lo cual no es muy bueno en un sistema inherentemente inestable como P2P, es decir, son sistemas menos robustos. Además las notificaciones periódicas incrementan el tráfico si el periodo de refresco es muy pequeño o la información deja de ser válida si el periodo es muy largo.

Aunque el tiempo de validez de la información no resulta un problema vital en los sistemas P2P, se torna crítico cuando se trata de sistemas Grid. En el primer caso, los recursos son relativamente estáticos (i.e. archivos) mientras que en Grid hay recursos a compartir cuya naturaleza es altamente dinámica (CPU load, free memory, etc.).

En Trunfio et al. [16], se menciona que los enfoques estructurados podrían ser utilizados para la información relativamente estática y los no estructurados para información dinámica. Más aún, se sugiere que el uso de la arquitectura Super Peer con cualquiera de las dos estrategias (protocolos estructurados y no estructurados) podría ser adoptado para el descubrimiento de recursos dentro de las VOs y entre distintas VOs.

Se menciona también que los enfoques para monitoreo y descubrimiento de recursos en los sistemas P2P pueden ser útiles para sistemas Grid escalables basados en semántica y, en particular, para construir una base de conocimiento distribuida para las descripciones de los recursos así como almacenamiento distribuido para las ontologías[3]. La búsqueda de coincidencias y el ruteo basado en semántica prometen mejoras en la precisión y costo del proceso de descubrimiento.

En Mastroianni et al. [15], Iamnitchi et al. [7], Trunfio et al. [16] se mencionan algunas políticas que pueden ser utilizadas para la distribución de la información en grids o grandes sistemas distribuidos.

En Iamnitchi et al. [7], se definen cuatro ejes para el estudio del problema de descubrimiento de recursos:

1. *Integración:* define el procedimiento por el cual los nodos se agregan al sistema y cómo conocen al resto[4].

2. *Función de construcción de la red lógica:* permite elegir a partir de ciertas métricas los nodos (peers) más adecuados para tener contacto.

3. *Preprocesamiento:* utilización de técnicas para hacer más eficiente la búsqueda, como reconectar la red lógica o anunciar los recursos locales a otras áreas de la red, etc.

[3] La información semántica está definida en términos de conceptos y relaciones especificados en una ontología. Una ontología es una especificación explícita de una conceptualización que sirve como fundamento para una representación formal del conocimento. Especifica formalmente cómo se representan los objetos, conceptos y otras entidades que se suponen que existen en algún área de interés y sus relaciones. En descubrimiento de recursos basados en semántica, ontologías comunes facilitan la comunicación entre proveedores de servicio y consumidores.

[4] Membership protocol.

4. *Procesamiento de solicitudes:* (i) Procesamiento local, por ejemplo dividir una consulta n-dimensional en n consultas de un sólo tipo de recurso, o descartar consultas a partir de un cierto TTL (*Time To Live*). (ii) Procesamiento remoto, mecanismo para el reenvío de la consulta (que ya puede haber sido procesada localmente) a otros peers (i.e. Flooding, Forwarding, Epidemic Communication, etc.).

En cuanto a cada uno de estos items, en el modelo que se utiliza en Iamnitchi et al. [7], los nodos se integran a la red haciendo contacto con algún nodo que ya pertenece a la misma y, de manera pasiva, conoce más integrantes de la red a medida que recibe mensajes de nodos que hasta ese momento resultaban desconocidos. La construcción de la red se inicia con una jerarquía y va creciendo a medida que se conocen otros nodos integrantes, no hay un límite prefijado en la cantidad de nodos. Suponen que no se realiza ningún tipo de preprocesamiento para optimizar las búsquedas. El procesamiento local sólo acepta matching exacto, en caso contrario se decrementa el TTL y se reenvía. Para la elección del destino al cual propagar, se proponen cuatro estrategias:

1. *Random walk:* el nodo seleccionado para realizar el reenvío se elije de manera aleatoria.

2. *Learning Based:* el nodo aprende de su experiencia. Reenvía al nodo, o los nodos, que respondieron previamente solicitudes similares. Si no hay experiencia previa se reenvía de forma aleatoria.

3. *Best Neighbor:* se registra la cantidad de respuestas recibidas de cada nodo, sin importar de que tipo se trate. Se reenvía al nodo que haya respondido más cantidad de solicitudes.

4. *Learning Based + Best Neighbor:* Similar a Learning Based, pero en el caso en que no haya experiencia previa se reenvía al nodo que seleccionaría la estrategia best neighbor. En este trabajo, la política Best Neighbor implementada se basa en esta estrategia.

De las cuatro estrategias analizadas en este trabajo, Iamnichi menciona que Random resulta la menos eficiente, mientras que las basadas en aprendizaje obtienen una buena performance en los escenarios tratados. Sin embargo, la estrategia Learning Based + Best Neighbor resulta más inestable, mostrando una performance menos predecible. La performance (medida en cantidad de hops per request) para Learning Based es la más sensible al tipo de distribución de las solicitudes de usuarios y, para Best Neighbor, se muestra influenciada por la distribución de recursos.

En Mastroianni et al. [15], se introducen dos claves para la utilización de P2P en sistemas Grid: Integración y estrategias para el descubrimiento de recursos[5]. En dicho trabajo, se analiza Super Peer, una propuesta híbrida P2P basada en el trabajo de Yang et al. [17]. En esta política, un nodo distinguido denominado super-peer actúa como nodo central de un sistema centralizado formado por un subconjunto de peers. Este super-peer, a su vez, se comunica con otros super-peers, pero no se comunica con los nodos asignados a otros super-peers. De esta manera se genera una red de dos niveles. En esta publicación de Mastroianni se remarca que este enfoque resulta especialmente adecuado para Grids de gran escala. Estas Grids de gran escala están compuestas por Grids pequeñas, es decir

[5] Membership management y resource discovery.

un conjunto de hosts bajo un mismo dominio de administración (a las que denomina *Physical Organizations*, PO, y no necesariamente coincide con las VOs). Cada PO podría, entonces, tener uno o más hosts actuando como super-peer y el resto de los hosts puede utilizarlo para acceder a la Grid y buscar servicios y recursos. En este trabajo los peers se comunican por medio de *gossiping* para elegir dinámicamente al super-peer. Una vez integrado al sistema, el super-peer envía las solicitudes a uno o varios de sus super-peers vecinos para que estos consulten sus Servicios de Información (i.e. Globus Toolkit 3 Index Sevices, o MDS-2 de Globus Toolkit 2).

En la descripción de las simulaciones que realizan en [15] puede observarse la variedad y clases de parámetros que intervienen y que sería necesario tener en cuenta para la utilización de Super Peer. Seleccionan dos de estos parámetros, TTL y cantidad de super-peers vecinos a los que se efectúa forwarding. Se fija uno y se varía el otro para distintos tamaños de cluster (1 indica que todos los nodos son super-peers, al aumentar el tamaño del cluster disminuye la cantidad de super-peers). Evalúa las siguientes métricas: (i) probabilidad de éxito de una solicitud realizada por un peer, (ii) cantidad promedio de recursos que un nodo descubre con una consulta, (iii) frecuencia de mensajes recibidos por un nodo, (iv) porcentaje de efectividad (cantidad de hits / cantidad de mensajes), (v) distintas variantes de tiempo de respuesta esperado. El artículo de Mastroianni et al. [15] contiene gran cantidad de resultados que pueden ser de interés al momento de la utilización de Super Peer, por ejemplo, que para clusters de 100 peers (100 super-peers en una red de 10000 nodos) el porcentaje de efectividad es máximo cuando el TTL es 3, mientras que para clusters de 500 es máximo cuando es sólo 2.

Como se dijo anteriormente, la manera cómo se obtiene y distribuye la información sobre los recursos se denomina *política de distribución de la información de recursos*, lo que en el artículo de Iamnitchi, [7], se denomina *Resource Discovery Problem*. Pensando los GRIS como peers y, teniendo en cuenta lo expuesto hasta ahora, existe un amplio abanico de posibilidades para organizar una política de distribución de la información.

Si bien, la política jerárquica es una de las más utilizadas, la necesidad de la intervención manual para su configuración y mantenimiento la vuelve menos atractiva ante un escenario de una creciente cantidad de recursos disponibles. Esta situación genera un especial interés en las políticas distribuidas surgidas del paradigma P2P.

En este contexto, resulta indispensable la caracterización del comportamiento de las diferentes propuestas de políticas de distribución de la información y la correcta evaluación de su performance.

Dentro de la bibliografía consultada, se encontraron diversas métricas para la evaluación de una política de distribución de recursos. Entre los parámetros que habitualmente se consideran podemos encontrar: tiempos de lookup, tamaño de las redes, tráfico y workload extras requeridos en función de la cantidad de mensajes.

La mayoría de estas métricas son descriptivas, dan información de las características de los escenarios y del comportamiento del sistema de manera muy general. Estas características deben tenerse en cuenta, pero para realizar un estudio de la perfomance de políticas de distribución de la información de recursos es necesario hilar más fino. Es decir, si bien es importante conocer el índice de aumento de tráfico en función de la cantidad de mensajes, esto nos dará información en función de las características de la red, la topología, el mecanismo de control de congestión, etc. Sobre la performance esto sólo nos dice que si aumenta la cantidad de mensajes aumenta el tráfico y según la capacidad de la red y el tipo de control de congestión que utilice, puede verse afectada. Esta característica

depende del escenario analizado y no permite observar el comportamiento de las políticas de manera general.

En cambio, conociendo la cantidad de recursos disponibles en el sistema, la probabilidad de éxito de una solicitud realizada por un peer, Mastroianni et al. [15], da información sobre la performance del sistema: si los recursos están disponibles en el sistema, pero la probabilidad de éxito al realizar una solicitud es baja, aún cuando el incremento en el tráfico de mensajes es aceptable, sería síntoma de una política poco eficiente.

En esa misma dirección encontramos en trabajos previos como los de Yabo [18], Mocskos et al. [14] y González Márquez et al. [19, 20] algunas métricas y herramientas que nos permiten este tipo de análisis para políticas de distribución de la información sobre recursos en general, las mismas se detallan en la Sección 2.3.

Siguiendo el trabajo presentado en Mocskos et al. [14], se estudian en este trabajo las políticas Jerárquica (Hierarchical), Super Peer, Random y Best Neighbor. La Figura 1.1 ejemplifica el esquema de intercambio de mensajes en términos de la organización de los nodos, en tanto que a continuación se presenta una breve descripción de ellas:

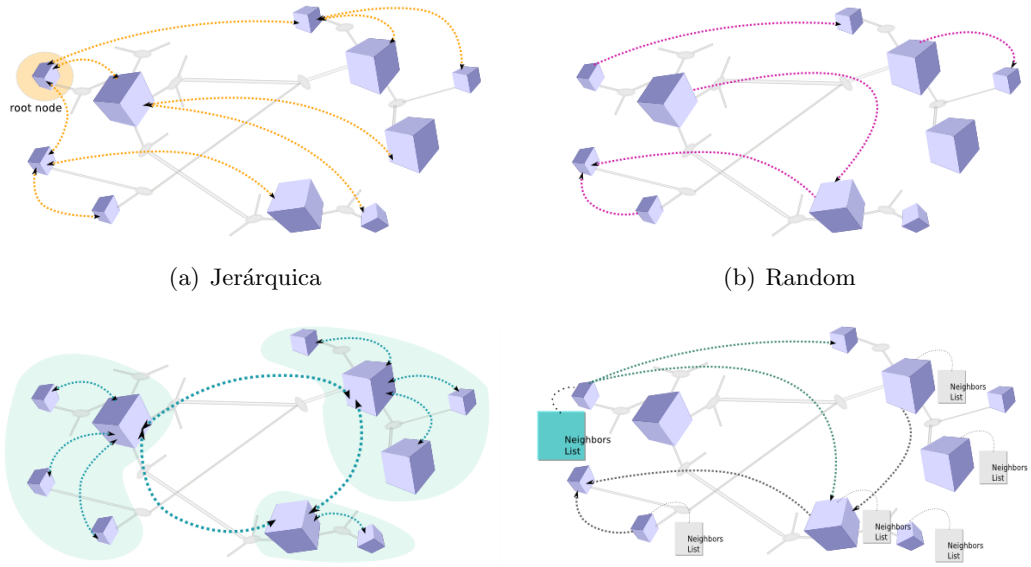
- **Hierarchical:** En esta clase de política, una jerarquía entre los nodos se establece con antelación. Todo intercambio de información entre los nodos que integran la red (solicitudes y publicaciones de recursos disponibles) utiliza esta estructura. Los nodos intercambian información solo con los del nivel inmediato inferior o inmediato superior de la jerarquía establecida, Figura 1.1(a). Esta política es la utilizada por MDS [14].

- **Random:** En las políticas random no existe ningún tipo de estructura. Cada nodo elige de manera aleatoria otro nodo de la red del cual obtener información, Figura 1.1(b). Suele utilizarse para comparar comportamiento de peor caso [14].

- **Super Peer:** Las redes Super Peer están desarrolladas como un sistema híbrido entre los sistemas completamente distribuidos y los *cache-based* (basados en la utilización de cache local para aminorar la carga general de la red). Una red Super Peer opera como una P2P (Peer to Peer) no estructurado [16], pero algunos nodos son definidos como super-peers (sp) trabajando como servidores de un subconjunto de nodos (peers) y como peers en la red de super-peers, quedando así definida una estructura de dos niveles, Figura 1.1(c). Esta distinción en dos tipos de nodos es utilizada por la política de distribución de la información Super Peer de la siguiente manera: los nodos peers se comunican directamente con un único super-peer y a través de él con los demás nodos. Al recibir una solicitud de un peer, el super-peer, primero consulta a los peers que están en contacto directo con él, en caso de no obtener respuesta satisfactoria reenvia esta solicitud a los demás super peers para que estos, a su vez, la reenvíen al subconjunto de peers que están en contacto directo con cada uno de ellos.

- **Best Neighbor:** Se almacena información a partir de cada respuesta recibida y el siguiente nodo al que se le enviará una solicitud será elegido según la "calidad" de la misma (cantidad de información recibida y tiempo de respuesta), Figura 1.1(d). Inicialmente, el nodo posee un desconocimiento total acerca de los demás nodos en la red y selecciona alguno de manera aleatoria. A medida que recibe respuestas, genera una lista de vecinos (neighbors) para luego seleccionar al que mejor satisface sus requerimientos (provee el mejor servicio, tiene mayor disponibilidad de recursos,

etc.). Cada nodo mantiene también una pequeña probabilidad de elegir de manera aleatoria a quien consultar, aún cuando ya tiene la información de todo el sistema, de manera de poder adecuarse a eventuales cambios en la topología de la red.



(a) Jerárquica (b) Random

(c) Super Peer (d) Best Neighbor

Fig. 1.1: Esquemas del intercambio de mensajes de diferentes políticas de distribución de la información. (a) Hierarchical (Jerárquica), (b) Random (Aleatoria), (c) Super Peer y (d) Best Neighbor (Mejor vecino).

Cabe aclarar que de las políticas mencionadas sólo Random y Best Neighbor pueden considerarse completamente distribuidas (descentralizadas y no-estructuradas).

Para la política Super Peer, los nodos necesitan identificar a sus correspondientes Super Peer (y los nodos Super Peer, a su vez, identificarse entre sí). Sin embargo, al utilizar la política Best Neighbor esto no resulta necesario. Con Best Neighbor los nodos no necesitan ningún tipo de información previa sobre el resto de la red.

El presente trabajo analiza estas cuatro políticas en general y se centra en el estudio de la política de distribución de la información de recursos Best Neighbor.

Siguiendo la política Best Neighbor, cada nodo atraviesa dos etapas bien diferenciadas: *Aprendizaje* y *Aplicación*[6]:

1. **Etapa de Aprendizaje**: Durante esta etapa, el nodo utiliza una estrategia Random para obtener información sobre los otros nodos de la grid que pueden proveerle recursos.

2. **Etapa de Aplicación**: utiliza la información que pudo acopiar durante la etapa anterior para seleccionar el nodo al que le efectuará la consulta. Esta selección la realiza a partir de su lista de vecinos (neighbors), eligiendo el mejor de ellos según un ranking que cada nodo confecciona en base a la información adquirida. Este ranking

[6] Learning Stage y Working Stage.

se genera en base a una función que utiliza distintos parámetros de la red (i.e. Round Trip Time del enlace o `RTT`) y/o de los otros nodos (i.e. poder de cómputo). Esta función se denomina *función de scoring*.

Es esperable que la información global con la que cuenta cada nodo sea *mejor* que si se estuviera utilizando la política Random, dado que incluye mayor esfuerzo computacional en la generación del *score* para guiar la selección del nodo a consultar, y que esto se vea reflejado en una mejor performance.

Contrariamente a lo esperado, los resultados iniciales utilizando la política Best Neighbor en diversos escenarios mostraron un desempeño pobre, similar a la política Random, por lo que se decidió buscar las razones causantes de este comportamiento.

De los análisis realizados a partir de los resultados de las simulaciones surgen las observaciones que permiten el desarrollo del siguiente informe y la posterior implementación de dos nuevas políticas de distribución basadas en Best Neighbor: pBN y xBN.

La organización de este trabajo es la siguiente: en el Capítulo 2 se puede encontrar un detalle de la metodología utilizada y algunas consideraciones respecto de los escenarios utilizados en las simulaciones.

En el Capítulo 3, se incluye un análisis de algunas de las características de las políticas de distribución estudiadas.

En el Capítulo 4 se presentan los resultados obtenidos y su posterior análisis.

Finalmente, en el Capítulo 5 se dan las conclusiones finales y algunas posibles líneas de trabajo que pueden explotarse a partir de lo realizado.

Los parámetros asociados a los resultados y otros detalles de los escenarios de simulación se detallan en el Apéndice A.

2. METODOLOGÍA

En este Capítulo se describe la metodología que se siguió para realizar el estudio de la política Best Neighbor y su comparación con las políticas de distribución de la información mencionadas en la Sección anterior: Jerárquica, Super Peer y Random.

Para el análisis y comparación de la performance de estas políticas de distribución se consideran los resultados de un conjunto de métricas, obtenidos mediante la simulación en diversos escenarios, tanto desde el punto de vista de la topología de red subyacente como de la cantidad de nodos del sistema.

2.1. Métricas

Entre la bibliografía consultada, el procedimiento para evaluar las distintas propuestas de políticas de distribución de la información se basa en un procedimiento similar al realizado en el paradigma P2P: tiempos de lookup, tamaño de las redes, tráfico y workload extras requeridos en función de la cantidad de mensajes. En Trunfio et al. [16], por ejemplo, las métricas que se utilizan son el análisis asintótico de peor caso para la escalabilidad (de tiempo y de tráfico), tamaño de la red, cantidad de nodos, grado de conectividad y diámetro de la red.

Usualmente, las métricas utilizadas para la evaluación de los sistemas de información de recursos se basan en: (i) throughput, (ii) tiempo de respuesta promedio, (iii) carga promedio de la CPU, (iv) cantidad de recursos conocidos, (v) cantidad de consultas exitosas, (vi) uso de los recursos, (vii) overload del sistema de información de recursos [14]. Esta clase de métricas no captura adecuadamente la dinámica de grandes sistemas distribuidos. Esto se debe en parte a la alta frecuencia de variación de cierto tipo de información (por ejemplo, cantidad de procesadores libres) y, en parte, porque varias de estas métricas proveen información que resulta demasiado local, es decir, no describen el comportamiento global del sistema.

El índice LIR (*Local Information Rate*) definido en Mocskos et al. [14] es una propuesta que captura la dinámica de este proceso al obtener una medida de la cantidad de información sobre recursos con la que cada nodo cuenta en un momento determinado incorporando, además, qué tan nueva resulta dicha información.

Este índice, junto con los índices GIR (*Global Information Rate*) y GIV (*Global Information Variability*) que capturan respectivamente el cambio en la información en la totalidad del sistema y la uniformidad de la misma entre los nodos, que también son propuestos en Mocskos et al. [14] y utilizados en González Márquez et al. [19, 20], serán los que se utilizarán en este trabajo.

Algunas características de los mismos:

- **LIR (Local Information Rate)**: Es un cociente que toma valores entre 0 y 1 que indica cuánta información tiene un nodo en particular sobre toda la red en un cierto momento, teniendo en cuenta el tiempo de expiración de dicha información.

  Alcanza un valor de 1 cuando el nodo tiene información sobre cada nodo de la red y ésta es lo más actual posible. N es la cantidad de nodos que componen la red, $expiration_h$ es el tiempo de expiración de la información sobre el nodo

h en el nodo k, age_h es el tiempo transcurrido desde que se obtuvo esa información. Mientras que $resourceCount_h$ es la cantidad de recursos que el nodo h provee y $totalResourceCount$ la cantidad total de recursos de la red. La expresión $expiration_h - age_h$ es similar al parámetro *time-to-live* del protocolo IP. Así, el LIR correspondiente al nodo k se obtiene mediante la siguiente expresión:

$$LIR_k = \frac{\sum_{h=1}^{N} \frac{expiration_h - age_h}{expiration_h} \cdot resourceCount_h}{totalResourceCount} \tag{2.1}$$

- **GIR (Global Information Rate)**: También toma valores entre 0 y 1 e indica la cantidad de información con la que cuenta el sistema sobre el total de los recursos disponibles. Se obtiene por medio del promedio de los **LIR** de cada nodo.
- **GIV**: corresponde a la variabilidad del **GIR** en el sistema. Se calcula como la desviación estándar del **GIR**.

Además se tendrán en cuenta los resultados presentados en González Márquez et al. [20] sobre la relación entre el tiempo de refresco y el tiempo expiración de la información sobre recursos, denominada índice R, para la selección de algunos de los parámetros en la configuración de los escenarios utilizados para las simulaciones que se describen luego, en la Sección 2.3.

Este índice se define como:

- **R:** cociente que indica la relación entre el tiempo de expiración de la información que intercambian los nodos y cada cuánto se actualiza la misma:

$$R = \frac{\text{tiempo de Expiración}}{\text{tiempo de Refresco}}$$

2.2. Simulador: Gridmatrix / SimGrid

Existe una gran variedad de herramientas de simulación que podrían utilizarse, como las mencionadas por Quétier et al. [21]. En particular, para el estudio de políticas de distribución de la información se cuenta con la herramienta de simulación `Gridmatrix` [14, 18].

El framework sobre el que se desarrolló, y que actualmente utiliza `Gridmatrix`, es `SimGrid` [22], un simulador para aplicaciones distribuidas en escenarios heterogéneos.

`SimGrid` requiere dos archivos: una red (`platform file`) que describe el entorno de links y hosts en el que se efectuará la simulación, y un detalle de qué procesos se ejecutan en cada host y con qué parametros (`deploy file`). En este último se define qué política de distribución de la información sobre recursos se utilizará. Cabe aclarar aquí que ciertos valores que se utilizan en las simulaciones con `Gridmatrix` dependen directamente del valor asignado en la configuración de la red subyacente (en el `platform file`). Por ejemplo la cantidad de recursos en los hosts, o el valor del RTT que depende del valor de la latencia en los enlaces, sólo que en este último caso debe considerarse además la descripción del modelo de red utilizado por Gridmatrix, es decir el modelo de red de `SimGrid`. El modelo de red se describe en profundidad en el trabajo de Casanova et al. [23].

Los procesos definidos en el `deploy file` deben ser implementados por el usuario. `Gridmatrix` incluye soporte para simular el comportamiento de las políticas Jerárquica,

Super Peer, Best Neighbor y Random, así como la posibilidad de incorporar nuevas políticas o modificarlas a partir de scripts en `Python`. En las secciones referidas a cada una de estas políticas en el Capítulo 3 se describe, en líneas generales, la implementación de cada una de ellas que será utilizada en este trabajo.

La interfaz gráfica de `Gridmatrix` permite elegir para cada una de estas políticas los valores de los parámetros, algunos de ellos específicos de la política y otros comunes a todas ellas. Entre los primeros se encuentran la cantidad de niveles para la jerarquía de la política Jerárquica o la cantidad de super-peers en el caso de la política Super Peer. Entre los segundos se encuentran el valor de `poll`[1], el valor de `push`[2] y el tiempo de expiración[3].

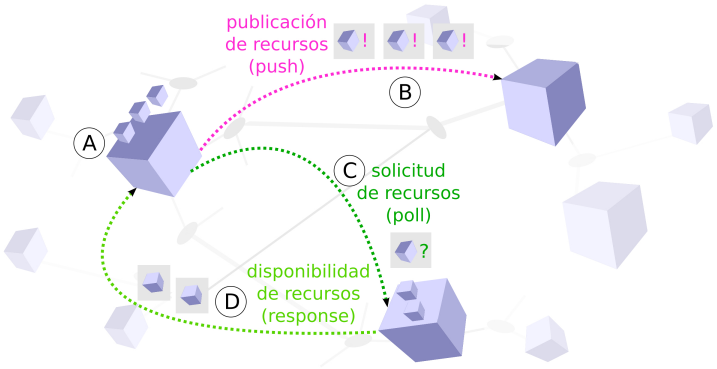


Fig. 2.1: Eventos de `push` (publicación de disponibilidad de recursos) y poll (solicitud de recursos). A - El host cuenta con tres recursos disponibles. B - El host publica la disponibilidad de estos recursos enviando un *push* a otro host. C - El host necesita un recurso, probablemente con características distintas de los que tiene disponible (i.e. mayor almacenamiento). Envía un *poll* a otro host solicitando el recurso y, a su vez, informa de la disponibilidad de sus recursos. D - Se responde la solicitud de recursos.

Además, permite la generación de redes con diversas topologías (incluso redes libres de escala), dando gran facilidad para la selección de parámetros (como ser el poder de cómputo de los nodos y la latencia para los enlaces), y permite obtener como resultado de las simulaciones los valores de LIR, de cada nodo y del GIR para distintas políticas. En su versión actual (`2.0+/simgrid3.5`), es posible generar redes de gran tamaño utilizando menor capacidad de cómputo que la utilizada por versiones anteriores.

La Figura 2.1 esquematiza los eventos de `push` y `poll` que se utilizan en la aplicación. En la primer etapa del esquema (A), uno de los hosts del sistema tiene tres recursos disponibles.

Al efectuar un `push` (punto B), el host da a conocer los recursos que puede proveer (tanto los suyos propios, como los de los nodos que conoce). Cada host que reciba un `push` actualizará en ese momento la información que tiene de los recursos disponibles en la grid.

Al efectuar un `poll` (punto C), el host solicita a algún host los recursos que necesita, informando a la vez los recursos propios que tiene disponibles en ese momento. Al recibir un poll el host responde con los recursos que tiene disponibles (punto D) y en ese momento también puede actualizar la información que tiene de los recursos disponibles en el sistema a partir de la información que provee el host que originalmente realizó la solicitud.

[1] Cada cuanto se efectúa una solicitud de recursos o servicios.
[2] Cada cuánto se efectúa una publicación.
[3] Durante cuánto tiempo se considera válida la información obtenida.

2.3. Descripción del escenario considerado y modelo utilizado

En los escenarios utilizados durante este trabajo, la cantidad de nodos se mantiene constante a lo largo del tiempo de cada simulación. Incluir escenarios dinámicos en los cuales los nodos pueden incorporarse o abandonar el sistema agrega un nivel muy interesante y atractivo de riqueza en el comportamiento del sistema, sin embargo este estudio excede los alcances que plantea la presente tesis.

Para poder seleccionar valores de referencia con los que comparar y poder definir los parámetros para el estudio de Best Neighbor (testigos para los resultados), previamente se realizaron algunas simulaciones en sistemas de 30 y 400 nodos para tres topologías: 2.2(b) Anillo; 2.2(d) Clique; 2.2(f) Exponencial (ver Sección 2.4).

En la Figura 2.2(a), 2.2(c) y 2.2(e) se incluyen a modo de ejemplo los resultados del GIR obtenidos para cada una de estas topologías para las políticas Jerárquica, de dos y tres niveles, Random, Super Peer (20 nodos y 40 nodos sp, 5 % y 10 % respectivamente) en una red con 400 nodos y 300 routers en distintas topologías.

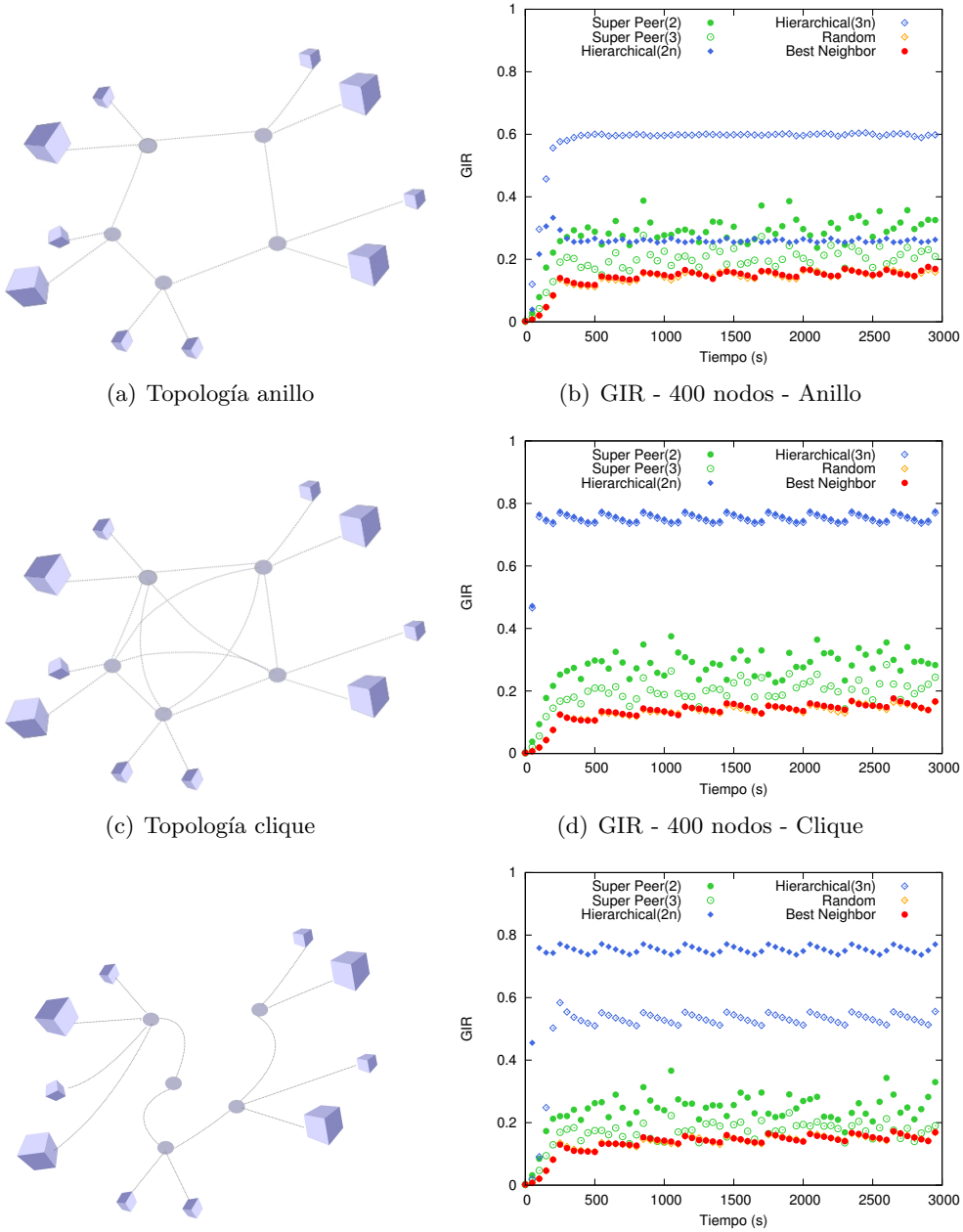
En las topologías Clique y Exponencial, la política Jerárquica de dos niveles (centralizada) obtiene los valores más altos de GIR, mientras que para la topología de Anillo los resultados de esta política son peores que los de Super Peer (5 %). Super Peer presenta un desempeño ligeramente superior a Random en las tres topologías, siendo mejor cuando menor es el porcentaje de super-peers. Random en todos los casos presenta el peor desempeño, mientras que Best Neighbor presenta valores de GIR semejantes a Random para las tres topologías.

Estas simulaciones no son determinísticas, aunque la dimensión del problema, es decir, la cantidad de variables involucradas, el poco conocimiento sobre la independencia entre ellas y la forma en que podrían estar afectando la perfomance del sistema, hacen que un estudio estadístico que permita tratar con rigurosidad la performance general de cada una de las políticas exceda el alcance de este trabajo. Un avance en esa dirección motivó, por ejemplo, algunos de los análisis incluidos en el Capítulo 3, a modo de estudio preliminar que permitiera reducir el espacio de pruebas a la hora de realizar experimentos en escenarios reales. En la sección inicial del Capítulo 3 se profundiza el análisis sobre las características de las políticas de distribución Jerárquica, Super Peer y Random para poder ajustar estos parámetros.

La Figura 2.3 permite observar la relación entre las métricas definidas en las Sección 2.1: la evolución del GIR en función de R para distintas políticas de distribución de la información. Cada punto en estas curvas corresponde a la simulación de un sistema en la cual los tiempos de expiración y de refresco de la información se han fijado de manera que se cumpla el valor de R correspondiente.

Atentos a los resultados presentados en [20] sobre la relación entre el tiempo de refresco y el tiempo de expiración de las solicitudes y publicaciones de información sobre recursos, verificamos que cuando los valores del índice R son menores que 10, el GIR presenta una mayor variación para todas las políticas estudiadas. Según estos resultados, cuando el valor de R es menor a 5, se considera que el sistema se encuentra trabajando con baja o muy baja performance, mientras que el GIR llega a un punto de estabilidad para valores más altos, es decir, cuando R supera el valor de 10.

Para valores de R en el rango entre 5 y 10, el GIR muestra diferencias mayores que en otros intervalos. Para todas las políticas de distribución de la información estudiadas se cumple que si el valor (promedio) del GIR es mayor en una política que en otra para



(a) Topología anillo

(b) GIR - 400 nodos - Anillo

(c) Topología clique

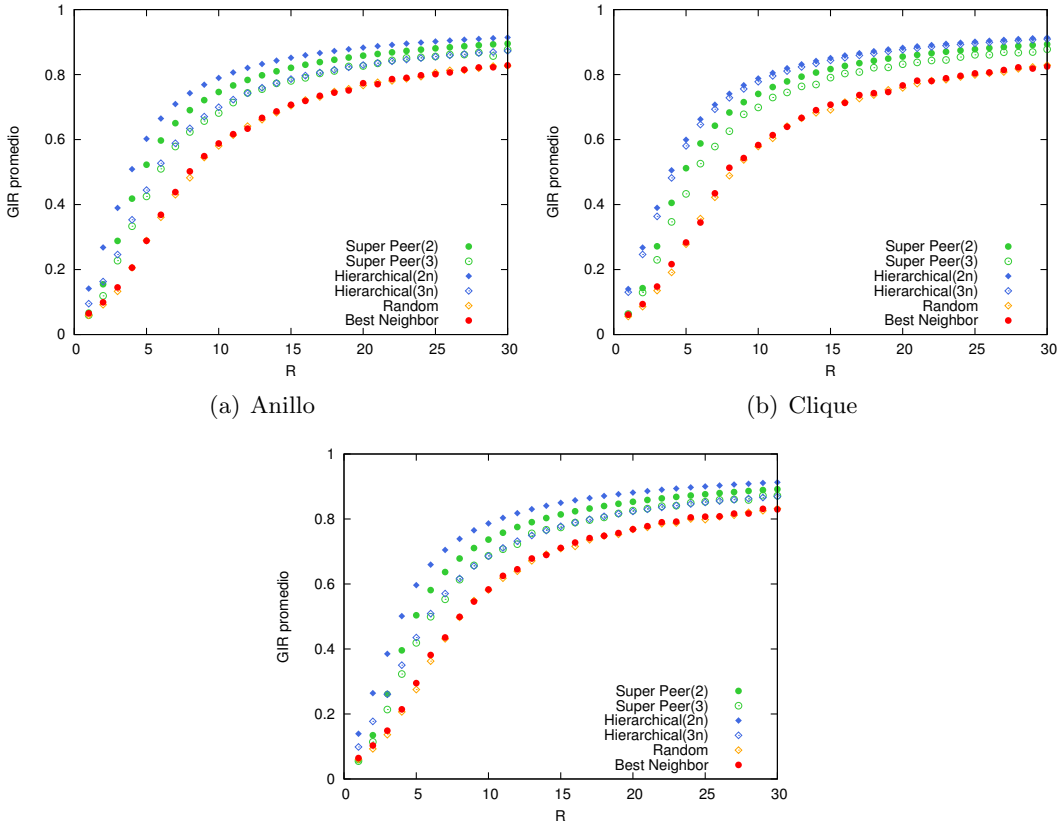
(d) GIR - 400 nodos - Clique

(e) Topología exponencial

(f) GIR - 400 nodos - Exponencial

Fig. 2.2: Las Figuras (a), (c) y (e) ejemplifican las distintas topologías de red para el caso de 5 routers y 10 nodos. En las Figuras (b), (d) y (f) se encuentran los resultados de la evolución del GIR en función del tiempo (tomando una ventana temporal de 10) para las políticas Jerárquica de dos y tres niveles, Random, Super Peer (20 nodos y 40 nodos sp, 5 % y 10 % respectivamente) en una red con 400 nodos y 300 routers en distintas topologías: (b) Anillo; (d) Clique; (f) Exponencial. En las topologías Clique y Exponencial, la política Jerárquica de dos niveles (centralizada) obtiene los valores más altos de GIR, mientras que para la topología de Anillo los resultados de esta política son peores que los de Super Peer (5 %). Super Peer presenta un desempeño ligeramente superior a Random en las tres topologías, siendo mejor cuando menor es el porcentaje de super-peers. Random en todos los casos presenta el peor desempeño. Best Neighbor presenta valores de GIR semejantes a Random para las tres topologías.

un R dado, entonces lo es para cualquier valor de R. Esto es común a las tres topologías



(a) Anillo

(b) Clique

(c) Exponencial

Fig. 2.3: Valores de GIR promedio en función de R de las políticas de distribución de la información Jerárquica (dos niveles), Jerárquica (tres niveles), Super Peer (dos nodos como super-peers), Super Peer (tres nodos como super-peers), Best Neighbor y Random, para sistemas de 30 nodos y 15 routers de topologías Anillo(a), Clique (b) y Exponencial (c).
Una política jerárquica de dos niveles equivale a una política centralizada. En el caso de Super Peer 1,5 nodos representan el 5 % de los nodos y 3 nodos representan el 10 %.

según puede observarse en la Figura 2.3: Anillo (a), Clique (b) y Exponencial (c). Estas condiciones llevan a elegir el mismo valor de R para todas las políticas.

Se configuran los escenarios de simulación de manera tal que el sistema se encuentre trabajando con valores de R en el intervalo $5 \leq R \leq 10$.

Se supondrá en este trabajo que la información que se obtiene en respuesta a una solicitud de recursos es siempre válida, es decir, si un nodo responde se supone que este puede satisfacer el pedido realizado en la solicitud.

No hay requerimientos sobre QoS (Quality of Service), aunque en el caso de Best Neighbor se registra la cantidad de veces que no se obtuvo respuesta de un nodo o ésta fue tardía.

Se configuran todos los enlaces con el mismo valor de latencia[4] y todos los nodos con el mismo valor de poder de cómputo (tipo de recurso que proveen) con lo que será indistinto elegir entre dos nodos salvo por la distancia a la que se encuentran y los recursos que proveen.

El ancho de banda se define como varios órdenes de magnitud mayor que la latencia en

[4] La distancia entre dos nodos queda simplificada a la cantidad de hops, o saltos, entre ellos.

los enlaces. La relación entre el tamaño de los mensajes y la cantidad de nodos en la grid, es decir la cantidad de *streams* posibles, es mucho menor que la capacidad en cada enlace, con lo que no se espera congestión. Si bien el simulador permite elegir el tipo de control de congestión (`TCPReno`, `TCPLasVegas`, etc.) no resulta útil la configuración del mismo (ver detalle de los parámetros en el apéndice A en la página 56). Se utiliza el modelo de red `CM02`, la configuración por defecto para `SimGrid` que se basa en *MaxMin fairness + AIMD*[5].

Finalmente, la cantidad de recursos que provee cada nodo es aleatoria con distribución uniforme entre un valor mínimo y uno máximo. Para todos los escenarios utilizamos los mismos valores mínimos y máximos.

2.4. Topologías utilizadas

En este trabajo, se seleccionaron tres topologías para analizar el comportamiento de las distintas políticas: Anillo, Clique y Exponencial. En las Figuras 2.2(a), 2.2(c) y 2.2(e) se muestran tres redes de cinco routers y diez nodos ejemplificando las topologías mencionadas.

Tal como se ejemplifica en la Figura 2.2(c), en una topología de Clique cada nodo se encuentra conectado directamente con todos los demás, mientras que en una topología de Anillo cada nodo tiene grado 2, es decir, tiene conexión únicamente con otros dos nodos formando, como su nombre lo indica, una estructura con forma de anillo (tal cual se muestra en la Figura 2.2(a)). Finalmente, la Figura 2.2(e) muestra un ejemplo de la topología Exponencial, la cual corresponde a un modelo de red compleja más cercano a las redes libres de escala como la World Wide Web (Internet), estudiada por Albert et al. [24] o distintos tipos de redes colaborativas o redes sociales [25–27].

En las redes libres de escala cada nuevo nodo se agrega de manera preferencial eligiendo los nodos con mayor conectividad (el vértice con mayor grado). En cambio en las redes exponenciales los nodos se agregan eligiendo un nodo de entre todos de manera equiprobable. En la Figura 2.4 se grafican ambas distribuciones.

En Newman et al. [27], puede encontrarse la descripción formal de grafos con distribución exponencial y grafos con distribución *power-law*, siendo estos últimos los que se corresponden con el modelo de estudio en el trabajo de Bárabasi et al. [26]. La topología Exponencial, al igual que los modelos power-law mencionados, está relacionada generalmente con escenarios dinámicos y se utilizan para modelar un variado conjunto de redes complejas.

2.5. Tamaño del sistema

Para los resultados presentados en este trabajo se utilizaron grids de entre 10 y 1000 nodos, aunque se analizaron sistemas de hasta 4000 nodos.

Los resultados de la Figura 2.5 fueron obtenidos mediante el promedio de la evolución del GIR durante 3000s en sistemas entre 100 y 1000 nodos. Se observa que al aumentar el tamaño de la red, la performance disminuye para todas las políticas estudiadas. Para la política Jerárquica disminuye más lentamente que para las otras a medida que aumenta la cantidad de nodos, en azul en la Figura 2.5. Tanto la política Random como Best Neighbor muestra una perfomance muy pobre y un decaimiento muy rápido respecto al tamaño del

[5] Puede encontrarse la documentación de `SimGrid` en http://simgrid.gforge.inria.fr.

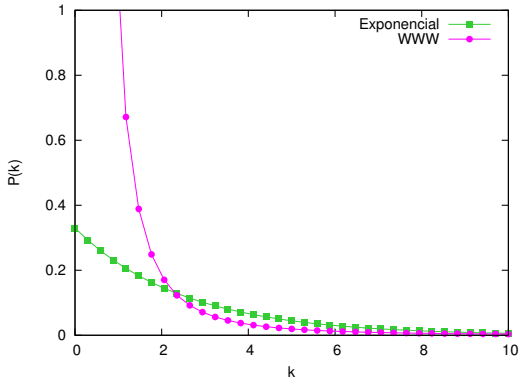


Fig. 2.4: Distribuciones de probabilidad que modelan el grado de conectividad de cada nodo descriptas por Albert et al. y Barabási et al. en [25, 26] y por Newman et al. en [27]. Describen la probabilidad de que un nodo esté conectado con k otros nodos de la red. Por ejemplo WWW(World Wide Web) sigue una distribución $P(k) = k^{-\gamma}$ donde para $P_{out}(k)$ se tiene $\gamma = 2{,}45$ y representa la probabilidad de que un sitio de Internet tenga k links externos (outgoing links). La distribución exponencial sigue una ley tipo $p(k) = (1 - e^{-\frac{1}{2{,}5}})e^{-\frac{k}{2{,}5}}$.

sistema. Finalmente, Super Peer también decae con el tamaño del sistema, pero lo hace de una manera más suave. Por otro lado para estas políticas también se reduce el desvío respecto del promedio lo que indica que el sistema funciona de manera más estable, esto no sucede con la política Jerárquica que muestra el mayor desvío.

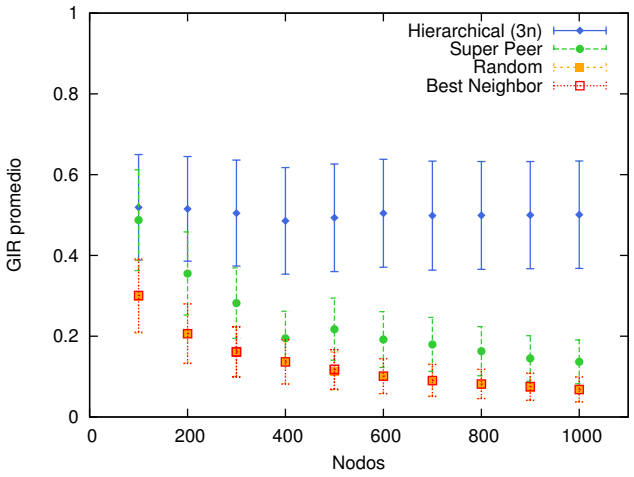


Fig. 2.5: GIR promedio en 3000s de las políticas de distribución Jerárquica, Super Peer (5 % de nodos como sp), Random y Best Neighbor, para redes de topología Exponencial con distintas cantidades de nodos (100 a 1000), y sus respectivos desvíos.

Si bien excede el alcance de este Capítulo, cabe mencionar que se considera que esta diferencia entre la performance de la política de distribución Jerárquica y la performance de las otras tres políticas se debe a que en estas últimas, en cada mensaje se obtiene una menor cantidad de información sobre recursos. De este modo, para contar con la misma cantidad de información (sin tener en cuenta cuán actual es), sería necesaria una mayor cantidad de mensajes.

Se observa que el índice R también se ve levemente afectado por el tamaño de la red, postergando la estabilidad en la performance para valores un poco mayores que 10, los resultados relacionados no se presentan porque no resultan concluyentes.

Estos temas no serán profundizados aquí, pero se retoman en el siguiente Capítulo.

Independientemente de la cantidad de nodos, se observarán los resultados de las políticas de distribución de la información en función de los obtenidos por la política Jeráquica de tres niveles para la misma cantidad de nodos.

3. CONSIDERACIONES PREVIAS

Previamente a analizar el comportamiento de la política Best Neighbor y de buscar las potenciales mejoras a realizar, resulta importante caracterizar las políticas de distribución de la información sobre recursos que se utilizarán para comparar a medida que se vayan presentando las diferentes propuestas.

Como se describió en la Sección 2.2, suponemos para este trabajo que cada nodo actualiza la información de los recursos en cada `push` que recibe y por cada respuesta dirigida hacia él a partir de la información que el otro nodo posee.

Se debe intentar mantener la información sobre el estado del sistema lo más actualizada posible para lograr una selección eficiente de los recursos. Para que esto ocurra, una vez que el host tiene información actual del sistema, cada nodo debería recibir un mensaje a cada instante con la información actualizada de cada uno de los recursos disponibles en el sistema, es decir en cada intervalo de tiempo Δt de duración lo suficientemente corta, de manera que la información no se desactualice. Claramente, un planteo que se basa en que todos los nodos del sistema envían y reciben un mensaje con la actualización sobre la información de los recursos en cada instante de tiempo resulta impracticable desde el punto de vista de la congestión, la escalabilidad y el uso eficiente de los recursos.

En el otro extremo, encontramos un escenario en el que cada nodo envía un sólo mensaje con información sobre sus recursos a otro nodo durante el intervalo Δt de tiempo. En este caso tardaría, como mínimo, N intervalos en comunicar su estado actual a los demás $N-1$ nodos de la red. La actualidad de la información dependerá del tamaño de la red y de cuán corta sea la duración del intervalo de actualización. Ante esta situación, es comprensible que la performance del sistema, en cuanto a la información sobre recursos, disminuya al aumentar el tamaño de la red.

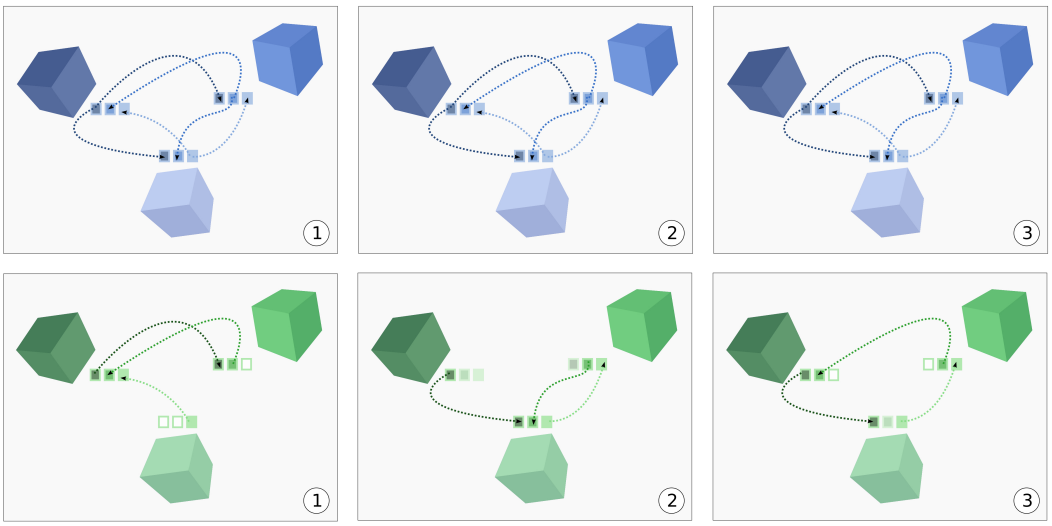


Fig. 3.1: Mecanismo de comunicación *push* en tres intervalos consecutivos (1, 2, 3). En la primer fila, todos los nodos envían a todos los nodos, luego de un intervalo de tiempo Δt (período de actualización) todos los nodos tienen información sobre todos los demás nodos, aunque desactualizada Δt tiempo. Los siguientes tres esquemas representan lo que sucede si cada nodo sólo envía un mensaje por vez a otro nodo: sería necesario un tiempo $(N-1)\Delta t$ para que todos los nodos tengan información sobre los demás nodos.

En la Figura 3.1 se esquematizan ambas situaciones: la primer fila corresponde a una situación en la que todos los nodos se envían mensajes entre sí, luego de un intervalo de tiempo Δt (período de actualización) todos los nodos tienen información sobre todos los demás nodos, aunque desactualizada Δt tiempo. En tanto que la segunda fila representa un escenario en el que cada nodo envía un solo mensaje por vez: es necesario un tiempo $(N-1)\Delta t$ cómo mínimo para que todos los nodos tengan alguna información sobre los demás.

Esta información tiene distintos grados de desactualización, por ejemplo, la información recibida sobre un nodo en un tiempo t_0 estará $(N-1)\Delta t$ tiempo desactualizada para cuando en $t_1 = t_0 + N\Delta t$ llegue nuevamente información de ese nodo con su estado actual (suponiendo que cada nodo efectúa los `push`s en round robin para simplificar la explicación).

Entre uno y otro extremo, cada una de las políticas de distribución de la información sobre recursos mencionadas en este trabajo provee distintas propuestas para la diseminación de la información sobre el estado de los recursos. Cómo influyen las mismas en la performance del sistema es analizado en las siguientes secciones.

Otros factores que pueden influir en la performance son, por ejemplo, los presentados en resultados previos que sugieren que los valores del GIR no son independientes de la elección del host que será la raíz (*root*) de la jerarquía en la política Jerárquica, ni de la cantidad de super-peers, en el caso de Super Peer [20].

Los factores adicionales que pueden influir en el desempeño de la política Super Peer incluyen, entre otros, cantidad de mensajes requerida entre super-peers, tipo de política de distribución que se utiliza entre los super-peers, tipo de política entre los peers, o el método de selección del super-peer [15]. Algunos de estos factores son analizados en la Sección 3.2, otros quedan fuera del alcance del presente trabajo.

De manera similar, no serán tratados ni el algoritmo para la generación de la jerarquía en la política Jerárquica, ni la búsqueda de la jerarquía óptima respecto de una topología dada; aunque sí se presenta en la Sección 3.1 un criterio con el que seleccionar el root que permite caracterizar la performance de la política Jerárquica en grids de topología exponencial.

La Sección 3.2 se centra en el impacto del aumento de la cantidad de super-peers en la política Super Peer.

En la Sección 3.3 se analiza el impacto del incremento de mensajes de actualización observando su efecto en la performance de la política de distribución Random.

Finalmente, en la Sección 3.4 se comienza con el estudio de Best Neighbor propiamente dicho.

3.1. Política Jerárquica

En cuanto a la política de distribución Jerárquica (**H**), el algoritmo utilizado por Gridmatrix genera un *overlay* arbóreo para la topología dada. Una vez seleccionado un nodo como raíz (*root*), se distribuye el resto de los nodos en los n niveles indicados por el usuario, utilizando un algoritmo que considera las distancias mínimas y su sobrecarga. En este trabajo se utilizan jerarquías de tres niveles en las que en el primer nivel sólo se encuentra el root, o nodo generador. Luego, partiendo de los niveles de la red física el algoritmo va plegando niveles hasta que sólo quedan n y por cada nivel asigna los nodos en el nivel inferior al nodo más cercano con menos nodos asignados.

En la política Jerárquica, el root en la jerarquía recibe `poll`s y `push`s de todos sus hijos, que corresponden con los nodos ubicados en el segundo nivel. Por otro lado, el root sólo puede responder a pedidos de `poll`s que le han sido realizados previamente por otros nodos.

Es decir, que en cada intervalo de intercambio el nodo raíz cuenta con la información actual de cada uno de sus hijos y la que estos, a su vez, poseen sobre los nodos en el siguiente nivel. Supongamos un intervalo de tiempo Δt en el que todos los nodos en el segundo nivel realizan un `push` (sólo pueden hacerlo al root).

En el segundo nivel, durante ese mismo intervalo, los nodos hoja realizaron un `push` a su respectivo nodo padre, por lo que en el siguiente intervalo de tiempo los nodos en el segundo nivel efectuarán un `push` al root y este contará con información de toda la red que ha envejecido, a lo sumo, $2\Delta t$.

En el siguiente intervalo las respuestas del nodo root a los nodos en el segundo nivel tendrán esa información que ha envejecido $3\Delta t$, y, en el siguiente, los nodos hoja tendrán información de todo el sistema envejecida, a lo sumo, $4\Delta t$.

La duración del intervalo Δt, que puede asociarse al tiempo de refresco, dependerá del tiempo de procesamiento, la estabilidad del sistema y la cantidad de nodos.

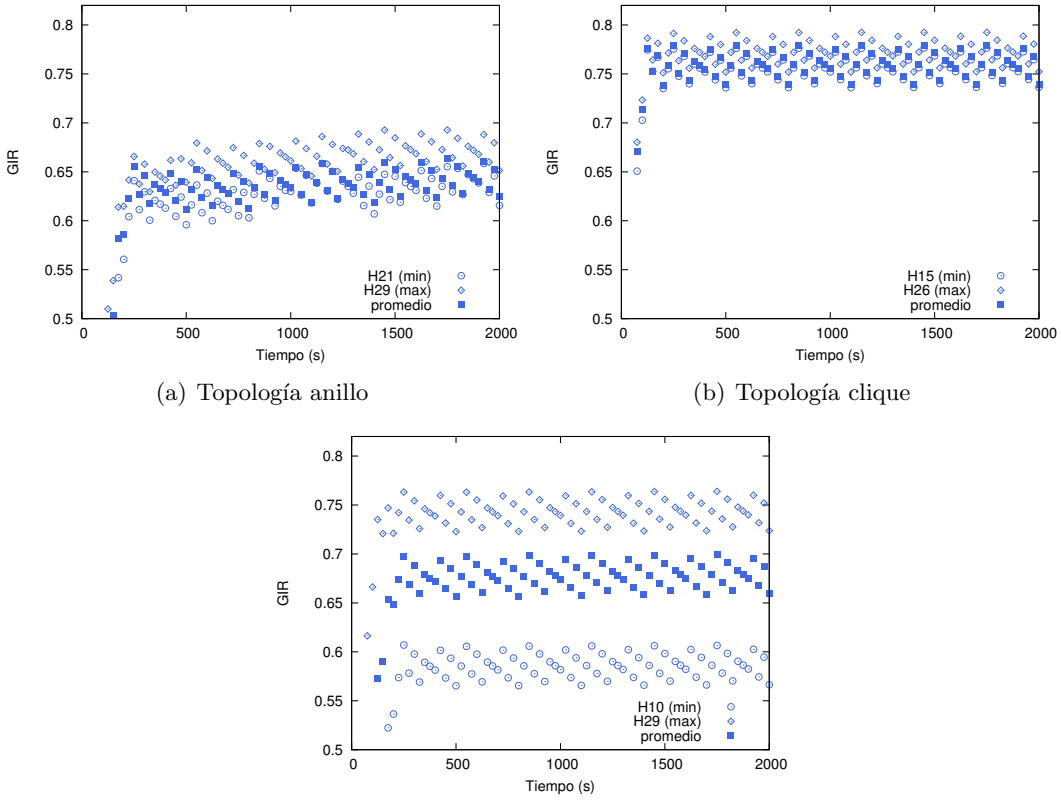
El nodo root y, en menor medida, los nodos en el segundo nivel envían gran cantidad de mensajes de respuesta (poll response) que sirven de mensajes de actualización. El tiempo que deben esperar los hosts para recibir información actualizada, junto con el hecho de que el root concentra la información de todo el sistema en poco tiempo, colabora en el obtener valores altos de LIR en los hosts e, indirectamente, en una buena performance.

A pesar de la buena performance que puede presentar la política Jerárquica, se observó en trabajos previos que la selección aleatoria del host que actúa como root del *overlay* jerárquico puede producir resultados significativamente diferentes. En este contexto, resulta fundamental encontrar un criterio con el cual elegirlo. Probar cada uno de los nodos del sistema no es una metodología escalable y no es sencillo hallar un criterio confiable para seleccionar el root de modo que la performance de la política jerárquica sea razonablemente buena.

Para los resultados de la Figura 3.2 se utilizó cada host como generador de una jerarquía en tres grids de 400 nodos con distintas topologías. Al evaluar el GIR para cada jerarquía generada, destacamos tres jerarquías paradigmáticas: mejor, peor y performance promedio para cada una de las topologías analizadas. En las Figuras 3.2(a) y 3.2(b) se observa que para una estructura de tres niveles en topologías en las cuales los nodos tienen igual grado (anillo y clique, respectivamente), resulta indistinto cuál host es seleccionado como root, mientras que para redes libres de escala la decisión deja de ser trivial, Figura 3.2(c).

En el caso de la topología exponencial el host elegido como generador de la jerarquía tiene una fuerte influencia en la performance, como muestra la diferencia entre los resultados del Host_29 (mayor GIR) y los del Host_10 (menor GIR) en la Figura 3.2(c). Elegir el nodo generador de manera aleatoria no parece adecuado. Sin embargo, es difícil encontrar propiedades que permitan distinguir, a priori, aquellos candidatos que determinan una buena performance de aquellos que no. Realizar una prueba para evaluar cada nodo a fin de determinar cuál produce la mejor performance podría llegar a tener sentido en un sistema con 30 nodos, pero no es un procedimiento escalable para dimensiones mayores.

La forma en que se distribuye la información sobre recursos puede entenderse como un caso particular del problema de Minimum Broadcast Time (MBTime), dado que lo que se intenta es mantener la información lo más actual posible entre un conjunto de



(a) Topología anillo

(b) Topología clique

(c) Topología exponencial

Fig. 3.2: Evolución del GIR para la política Jerárquica en grids de 30 nodos con distintas topologías y para las ditintas jerarquías de tres niveles generadas por cada uno de sus hosts al ser elegidos como root. Sólo se muestran los resultados de máxima performance, mínima performance y el promedio, se indica el host utilizado como root en cada caso. (a) Topología anillo, el utilizar distintas jerarquías no tiene gran influencia en la performance. (b) Topología clique, la diferencia en la performance al utilizar distintas jerarquías es casi indistinguible. (c) Topología exponencial, la performance depende fuertemente de la jerarquía utilizada.

nodos. El problema general de determinar el MBTime para un conjunto de nodos es NP completo [28]. Sería interesante la generación de overlays jerárquicos de mínimo tiempo de broadcast, y puede considerarse un caso particular de MBTime, en el que la comunicación entre los nodos se restringe a las ramas de un árbol.

En el trabajo de Proskurowski [29] se caracterizan los Minimum Broadcast Trees (MBTs), cuya estructura permite el broadcasting a los demás nodos en tiempo mínimo, y se presenta un algoritmo que permite la construcción de MBTs para un número determinado de vértices, y una fórmula para contar los posibles árboles MBTime para ese conjunto de nodos. En el trabajo de Farley et al. [30] se propone un algoritmo tal que para una estructura jerárquica (un árbol) permite determinar el conjunto mínimo de subárboles que cubren toda la red en un tiempo dado y, lo que está más relacionado con nuestro trabajo, estudia la influencia del nodo, o los nodos, que origina el mensaje y el esquema de reenvío para cada subárbol.

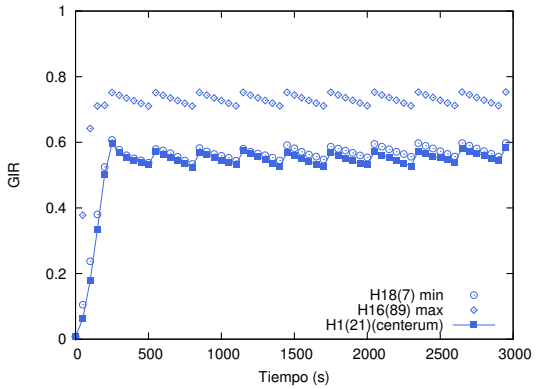
Podría ser muy rico un análisis de las heurísticas para esta clase de problemas, así como explorar los algoritmos utilizados para la generación de las jerarquías pero queda fuera del alcance de este trabajo.

Para poder determinar un nodo generador que permita obtener una buena performance, se implementaron dos técnicas con resultados poco alentadores:

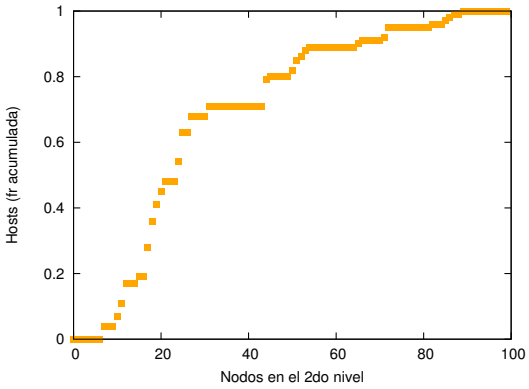
i) utilizar el nodo de mayor grado como nodo generador.

ii) utilizar el nodo con el menor camino máximo a cualquier otro nodo (el de menor excentricidad).

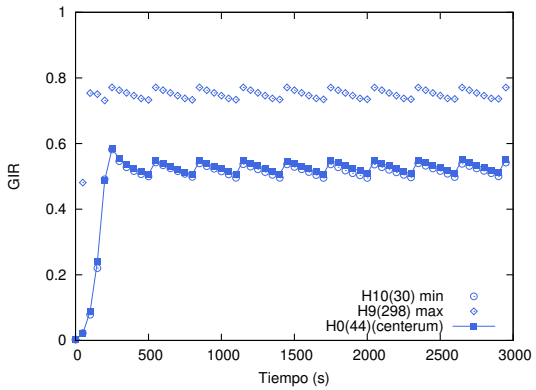
Se define **centerum** como aquel nodo con suma mínima de distancias a los demás nodos, y se observan los valores del GIR cuando es utilizado como root de la jerarquía.



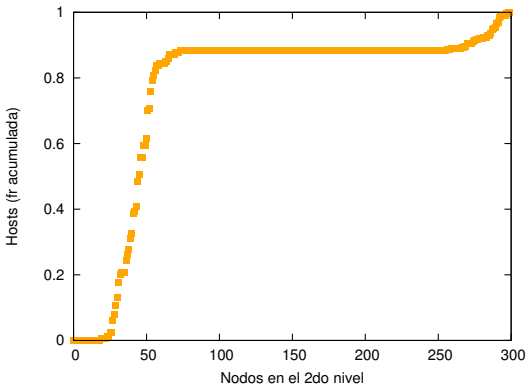
(a) 100 nodos



(b) 100 nodos. Frecuencia relativa acumulada.



(c) 300 nodos



(d) 300 nodos. Frecuencia relativa acumulada.

Fig. 3.3: En las Figuras (a) y (c) se grafica la evolución del GIR para 100 y 300 nodos, para el host con máxima cantidad de nodos en el segundo nivel, el centerum y el nodo con mínima cantidad de nodos en el segundo nivel (entre paréntesis se indica la cantidad exacta de nodos en el segundo nivel de la jerarquía generada por cada uno de los hosts). En las Figuras (b) y (d) se grafica para sistemas de 100 y 300 nodos la frecuencia relativa acumulada de hosts que generan jerarquías con la misma cantidad de nodos en el segundo nivel. Para 100 nodos el 50 % genera jerarquías con menos de 23 nodos en el segundo nivel, el centerum genera una jerarquía con 21 nodos en el segundo nivel. Para 300 nodos el 50 % genera jerarquías con menos de 45 nodos en el segundo nivel y el 80 % con menos de 55 nodos en el segundo nivel, en este caso el centerum genera una jerarquía con 44 nodos en el segundo nivel.

Si para un sistema determinado de N nodos generamos la jerarquía a partir de cada uno de sus nodos utilizando el algoritmo descripto en la Sección 3.1 obtenemos, en principio, N distintas jerarquías.

En la Figura 3.3(a), se encuentran los resultados del GIR para la jerarquía generada a partir del centerum y para las jerarquías generadas a partir del `host18` y el `host16` de un

sistema con 100 nodos, entre paréntesis se indica la cantidad de nodos en el segundo nivel. La jerarquía generada a partir del `host16` obtiene valores más altos de GIR, aunque la cantidad de nodos en el segundo nivel indicaría que se asemeja a un sistema centralizado.

En la Figura 3.3(c) se encuentran los resultados del mismo experimento pero para un sistema de 300 nodos. Las Figuras 3.3(a) y 3.3(c) muestran que el desempeño mejora cuando la jerarquía generada se asemeja a una de dos niveles.

Se analiza para sistemas de 100 y 300 nodos la cantidad de nodos en el segundo nivel de cada una de las jerarquías generadas a partir de cada uno de los hosts. Es posible agrupar estos resultados según la cantidad de hosts que generaron una jerarquía con determinada cantidad de nodos en el segundo nivel y obtener la frecuencia relativa de cantidad de hosts en el segundo nivel. En la Figura 3.3(b) se presentan los resultados de la frecuencia relativa acumulada de hosts que generan jerarquías con la misma cantidad de nodos en el segundo nivel para cada posible cantidad de nodos. Es posible observar la probabilidad de que un host genere una jerarquía con determinada cantidad de nodos en el segundo nivel. Para 100 nodos el 50 % genera jerarquías con menos de 23 nodos en el segundo nivel, el centerum genera una jerarquía con 21 nodos en el segundo nivel. Para 300 nodos el 50 % genera jerarquías con menos de 45 nodos en el segundo nivel y el 80 % con menos de 55 nodos en el segundo nivel, en este caso el centerum genera una jerarquía con 44 nodos en el segundo nivel.

Al aumentar a más del doble la cantidad de nodos se vuelve aún más notoria esta tendencia. La Figura 3.3(d) revela que la mayoría de los hosts generan jerarquías con pocos nodos en el segundo nivel y que se polariza mucho más el tipo de jerarquía generada.

La performance se maximiza cuando la cantidad de nodos en el segundo nivel de la jerarquía generada aumenta, pero la mayor parte de los nodos generan jeraquías similares a la generada por el centerum.

El overlay generado con el centerum como root, si bien no tiene una excelente performance resulta representativo de la mayoría de las jerarquías, por lo que se utilizará como criterio para los resultados de la política jerárquica la utilización del centerum como root .

3.2. Política Super-Peer

En la implementación de la política Super Peer (**SP**), cada nodo (*peer*) se comunica con un *super-peer* (**sp**) local y esta información es distribuida a los demás super-peers eligiendo otro super-peer de manera aleatoria, como puede verse en el esquema de la Figura 1.1(c) (página 8).

La implementación de SP particiona la topología de red subyacente utilizando la herramienta `metis` [31]. Dentro de cada partición se selecciona al super-peer de manera tal que minimice el total de distancias a otros nodos dentro de la partición.

Existen otras variantes para la selección de los super-peer, por ejemplo en el trabajo de Mastroianni se utiliza *gossiping* entre los peers, mientras que en Lin et al. [32] los clusters de peers se conforman según la información que poseen sobre cierto tipo de atributo. Para preservar localidad en la resolución de las *queries*, cada super-peer representa cierto tipo de atributo (i.e. capacidad de almacenamiento), el mecanismo propuesto por Lin prevee balanceo de carga, "reasignando" a los peers con atributos de bajo requerimiento a otro super-peer en un cluster con mayor carga.

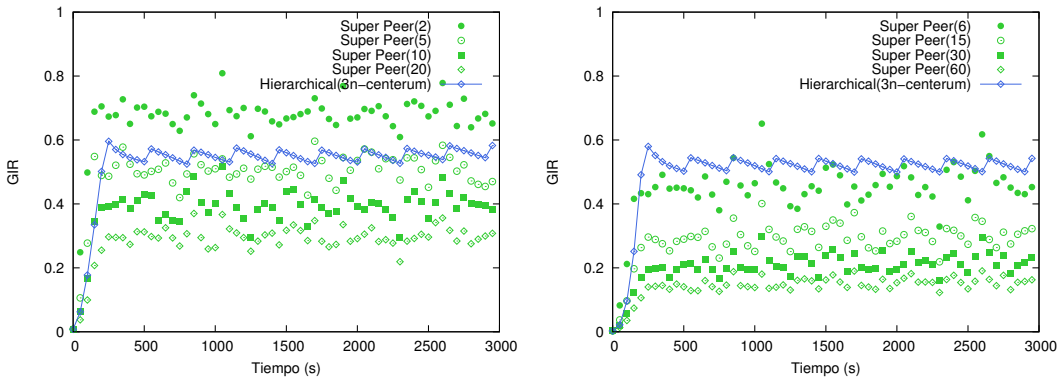
En SP, los nodos peers realizan `push` y `poll` a los super-peers, cada super-peer tiene

la información de todos sus peers en un sólo intervalo Δt. A su vez, el nodo super-peer reenvía, a partir de $2\Delta t$, los `polls` a otro super-peer. Al recibir una respuesta de otro super-peer, obtendrá información de todos los peers de ese super-peer, envejecida $3\Delta t$. Al responder a sus propios peers estos tendrán la información actualizada de los peers en ese cluster y de los peers correspondientes al otro super-peer envejecida, a lo sumo, $4\Delta t$.

Si se disminuye la cantidad de super-peers, disminuye también la cantidad de intervalos de tiempo necesaria para que todos los peers obtengan información del sistema. Con sólo dos super-peers, cada uno contará con información de la mitad del sistema (suponiendo un particionado parejo). En tres o cuatro intervalos de tiempo, cada peer contará con información de toda la red. Si fueran cuatro super-peers y, suponiendo que cada super-peer realiza un poll a un super-peer distinto en cada intervalo, se necesitarían no más de tres intervalos más, es decir $6\Delta t$. Sin embargo, la información sería bastante menos actual.

La política super-peer muestra un comportamiento más estable en la topología exponencial. Aunque puede verse en la Figura 3.4 que la performance de esta política depende fuertemente de la cantidad de nodos que actúan como super-peers (sps). El valor del GIR mejora al disminuir la cantidad de sps, superando, en una grid de 100 nodos y para una mínima cantidad de sps la performance de Jerárquica (centerum), Figura 3.4(a). Resultados relacionados fueron presentados por González Márquez et al. en [20].

Como puede verse en la Figura 3.4(b), la performance también se ve afectada por el tamaño de la red.



(a) 100 nodos (b) 300 nodos

Fig. 3.4: Evolución del GIR de redes de topología exponencial formadas por 100 nodos y 300 nodos para distintos tamaños de cluster (distintos porcentajes de super peers)

En cada intervalo de `push` todos los peers envían la información actual sobre sus recursos al sp en ese cluster, es decir que en cada intervalo de `push` todos los super-peers cuentan con la información más actual posible de una porción de la red. Al disminuir la cantidad de sps aumenta el tamaño del cluster, por lo que cada sp cuenta con información actual sobre una mayor cantidad de nodos, lo que como se mencionó previamente disminuye también la cantidad de intervalos de tiempo necesaria para que todos los peers obtengan información del sistema. Esto explicaría por qué la performance disminuye al aumentar la cantidad de sps.

Las solicitudes son enviadas y recibidas entre super-peers. Al recibir respuesta el sp cuenta con la información actual del sp que respondió, es decir de todos los peers de ese cluster. Dado que en la implementación utilizada para las simulaciones cada solicitud

recibe una respuesta, existe una relación uno a uno entre la cantidad de solicitudes y la cantidad de respuestas. Al disminuir el tiempo de envío de solicitudes se está disminuyendo, indirectamente el tiempo entre respuestas y así la información recibida mantendría un poco más su actualidad, aumentando los valores de GIR, lo que indica una mejora en la performance.

En la Figura 3.5(a) se encuentran los resultados del GIR promedio al variar el tiempo entre `polls` en una topología exponencial de 100 nodos, para distintos tamaños de cluster. Al disminuir el tiempo entre `polls` entre sps (el tiempo de `push` y `poll` no se modificó para los peers) el GIR promedio mejora notablemente, reduciendo las diferencias introducidas por el aumento de super-peers.

Debe tenerse en cuenta que en este caso se trata de incrementar el intercambio sobre un bajo porcentaje de los nodos de la red, por lo que la influencia de esta mejora en el incremento de tráfico no debería ser relevante. Aunque sería importante su estudio en relación a la cantidad de sps, se deja para trabajos futuros.

Retomando el ejemplo de una grid con tamaño de cluster N/2 (2 sps), si en cada intervalo de intercambio se realiza una solicitud a un sólo super-peer, alcanzará con un intervalo para que cada uno de esos super-peers tenga la información del otro. Es decir al cabo de dos períodos de actualización cada uno de ellos contará con información muy actual de todos los nodos.

Al aumentar la cantidad de super-peers, y utilizando el mismo criterio de enviar una sóla solicitud a otro super-peer elegido al azar, la cantidad de información con la que cuenta cada sp al cabo del mismo intervalo de tiempo es la de dos cluster, el propio y el del sp que responde. Por ejemplo, en el caso de tener 5 sps, la información sería de 2/5 del total de los nodos. En un caso ideal en el que cada destinatario de una solicitud (y respuesta) no se repitiera y se eligiera el sp en round robin, se necesitarían tantos intervalos de actualización como cantidad de sps, sin tener en cuenta el tiempo que tardan en arribar las respuestas, para tener información sobre todos los nodos. Al terminar ese ciclo la información recibida al principio del mismo será $(N-1)\Delta t$[1] menos actual (N = cantidad de sps), esto empeora la calidad de la información, el valor del GIR.

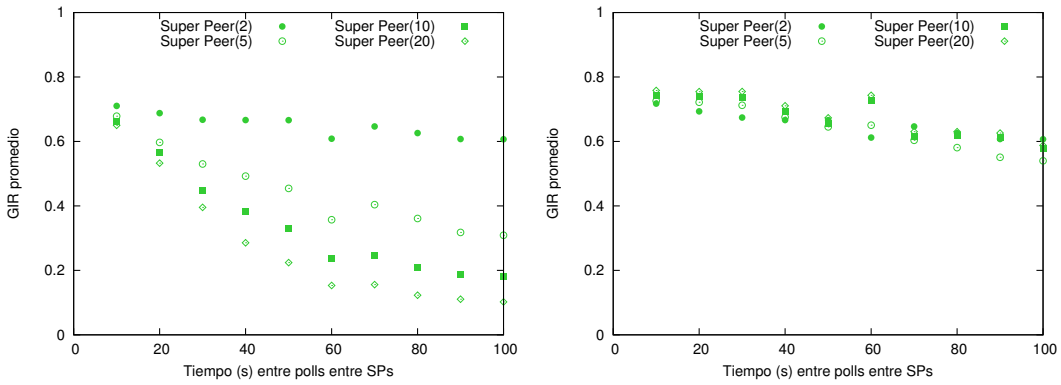
En la Sección 3.3 se analiza la influencia en la performance de modificar la cantidad de mensajes de actualización simultáneos, en la Figura 3.5(b) se encuentran los resultados del GIR promedio al modificar la política para que los super-peers envíen una solicitud a cada uno de los demás super-peers en cada intercambio, de modo que por cada poll realizado por un sp se obtenga la información de todos los demás clusters.

Vemos que con esta estrategia la influencia del incremento del período de poll es mucho menor, del mismo modo la cantidad de super-peers, que es casi mínima.

En Mastroianni et al. [15], donde podemos encontrar resultados relacionados, se sugiere una relación de entre 2 y 8 sps para sistemas de entre 10 y 10000 nodos, para el envío de consultas entre super-peers. Sería interesante profundizar el estudio de estos temas, analizando cuántos sp es conveniente tener según la cantidad total de nodos y en cada caso a cuántos de ellos es necesario contactar para optimizar la performance, y si esto es factible para la infraestructura de red dada (según cuántos mensajes extras tolere sin limitar su servicio). Se deja para trabajos futuros.

Para comparar los resultados se utilizará Super Peer con el 2 % y el 5 % de nodos como sps (*2-SP y 5-SP*). Con el mismo tiempo de `poll` y `push` para todos los nodos, incluyendo los `polls` entre sps. Cada sp realiza una sóla solicitud a otro sp por intervalo

[1] Tiempo entre `polls`.



(a) 100 nodos. Poll a 1 sp.

(b) 100 nodos. Poll a todos los sp.

Fig. 3.5: GIR promedio en redes de topología exponencial formadas por 100 nodos para distintos tamaños de cluster (distintos porcentajes de super-peers) al incrementar el tiempo en el período de intercambio entre super-peers. Cuando cada super-peer realiza un solo poll a otro super-peer, (a), y cuando realiza un poll a todos los demás super-peers, (b). El incrementar la cantidad de envíos simultáneos a otros super-peers realizando un poll a cada uno de todos los demás super-peers la performance mejora lo suficiente como para absorber el impacto negativo del incremento en la cantidad de super-peers.

de actualización.

3.3. Política Random

La política Random (**R**) es implementada por medio de la selección aleatoria del nodo al que se realizará la solicitud[2].

Al utilizar Random, los nodos sólo envían un `poll` a un nodo y un `push` a otro, es decir que sólo se actualiza la información en un nodo por `push` y en un nodo por respuesta a un `poll`.

Sea Δt el intervalo de tiempo en el que todos los nodos hayan realizado un `push`. Al cabo de ese tiempo la mayoría (sino todos) los nodos tendrán información sobre sí mismos y sobre algún otro nodo en la red. Si el sistema tuviera 100 nodos, se tendría información sobre el 2 % del total de nodos, tal vez un poco más teniendo en cuenta las respuestas agregadas que se reciban (información que se recibe indirectamente sobre la vecindad).

En este escenario, se necesitarían aproximadamente $50\Delta t$ para que cada nodo llegue a tener información sobre todo el sistema. Sin embargo, dada la dinámica de la información sobre recursos, la información queda desactualizada mucho antes.

Si bien la performance de la política Random será considerada en este trabajo el comportamiento de peor caso, es importante mencionar que su performance es semejante a la de las otras políticas en redes de pocos nodos y que algunas de sus características merecen un análisis más profundo.

En el caso de redes de gran tamaño, por ejemplo, no es posible afirmar que la performance de Random es peor que la de SP (*5-SP*). Utilizando una variación de esta política que llamaremos *N-Random*, consistente en realizar N solicitudes a destinos aleatorios por vez, se obtienen resultados como los de la Figura 3.6(a).

Para estos resultados, Figura 3.6(a), se fijó un tiempo de expiración y se obtuvieron los

[2] Es común encontrar en la literatura la denominación *query* para los mensajes de solicitud.

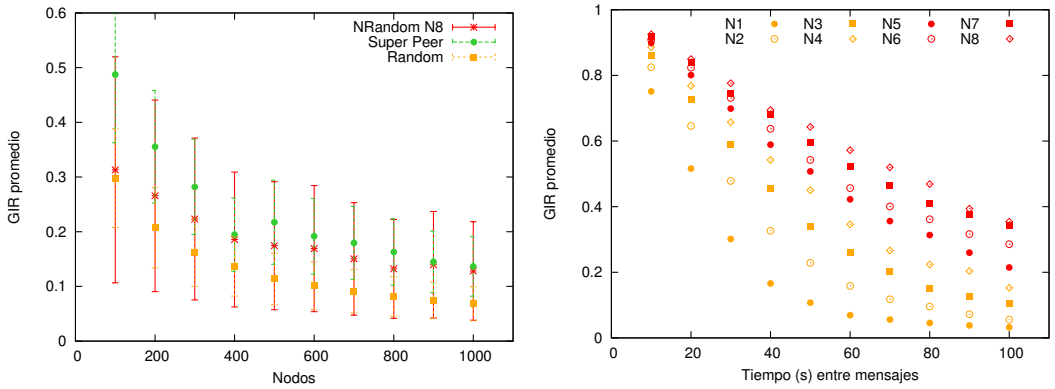
valores del GIR para Random y para 8-Random, manteniendo constante la cantidad de mensajes totales enviados en ese periodo (se varía el tiempo de refresco para 8-Random).

A medida que aumenta el tamaño de la red, la performance de 8-Random se mantiene mejor en promedio que la de 1-Random, a pesar de estar enviando la misma cantidad de mensajes, aunque con un desvío estándar mayor, con un comportamiento más cercano a 5-SP.

Si bien el envío de mensajes a más de un nodo incrementa el tráfico en ese período, resulta necesario un estudio más profundo a fin de determinar su impacto en el comportamiento del sistema. Por ejemplo, si se considera una ventana de tiempo de 10 segundos y se envía un mensaje por segundo (tiempo de refresco 1s), a los 10 segundos 10 nodos contarían con información 10 segundos desactualizada (como máximo), si no hubiera destinatarios de mensajes repetidos. Si, en cambio, se envían esos 10 mensajes *de una sola vez* (tiempo de refresco $10s$), también se estarían mandando 10 mensajes en 10 segundos, por lo que podría reducirse el tiempo de refresco en un orden de magnitud para obtener la misma performance.

Este análisis es preliminar, pero podemos ver en la Figura 3.6(b) que, por ejemplo, un valor de GIR alrededor de 0.5 se obtiene enviando 6 mensajes cada 60s u 8 mensajes cada 80s, la cantidad de mensajes en un período de 1000s será la misma en ambos casos, y sería posible aumentar el tiempo de refresco.

Frente a un mismo tiempo de expiración de la información, más importante que el tiempo entre mensajes de refresco es la cantidad de nodos que lo reciben. Incrementar la cantidad de mensajes, frente a un mismo período de intercambio, siempre produce una mejora, aunque en este caso debería tenerse en cuenta cuál es el límite en el incremento de mensajes para no saturar la red.



(a) 8-Random al aumentar la cantidad de nodos.

(b) 100 nodos.

Fig. 3.6: (a) GIR promedio para las políticas de distribución Super Peer (*5-SP*), Random (1-Random) y 8-Random, para redes de topología Exponencial con distintas cantidades de nodos (100 a 1000 nodos), y sus respectivos desvíos. (b) GIR promedio para N-Random, N entre 1 y 8, para una grid de topología exponencial de 100 nodos, variando el tiempo entre mensajes de refresco (push y poll) entre 10 y 100. Para cada uno de estos valores de tiempo de refresco se modifica la cantidad de solicitudes simultáneas obteniendo, por ejemplo, un valor de GIR alrededor de 0.5 al enviar seis mensajes cada 60s (N6) u ocho mensajes cada 80s (N8), la cantidad de mensajes en un período de 1000s será la misma en ambos casos. Con tiempo de expiración fijo para todos los casos en el mismo valor, para un mismo tiempo de refresco se obtiene mejor performance cuando se incrementa la cantidad de mensajes enviados simultáneamente.

En este trabajo se utilizará como criterio para comparar los resultados obtenidos por 1-Random, que para el resto del trabajo denominaremos simplemente, Random.

3.4. Política Best Neighbor

La implementación de la política Best Neighbor (**BN**) mantiene la cantidad de nodos que cada nodo *conoce* de la red (vecino o neighbor), es decir, en todo momento un nodo puede saber con qué porcentaje de los nodos de la red se ha comunicado alguna vez. Aquel nodo que va a efectuar una solicitud (llamado *nodo origen*) determina si va a seleccionar un *nodo destino* de manera aleatoria o si lo hará mediante la utilización de la información adquirida previamente seleccionándolo de la lista de vecinos. Esta selección se guía de acuerdo a una función denominada *función de scoring* que realiza una combinación lineal de distintos parámetros.

Esta implementación de Best Neighbor se basa en la estrategia *Learning Based + Best Neighbor* descripta en Iamnitchi et al.

Esta política de distribución de la información puede considerarse completamente distribuida (descentralizada, no-estructurada).

Como se describió en la introducción, Best Neighbor almacena información a partir de cada respuesta recibida y el siguiente nodo al que se le enviará una solicitud será elegido según la "calidad" de la misma (cantidad de información recibida y tiempo de respuesta), Figura 1.1(d).

Inicialmente, el nodo posee un desconocimiento total acerca de los demás nodos en la red y selecciona alguno de manera aleatoria. A medida que recibe respuestas, genera una lista de vecinos (neighbors) para luego seleccionar al que mejor satisface sus requerimientos. Cada nodo mantiene también una pequeña probabilidad de elegir de manera aleatoria a quien consultar, aún cuando ya tiene la información de todo el sistema, de manera de poder adecuarse a eventuales cambios en la topología de la red.

Siguiendo la política Best Neighbor, cada nodo atraviesa dos etapas bien diferenciadas:

- **Etapa de Aprendizaje**: el nodo no conoce a ningún otro nodo en la red y simplemente realiza solicitudes y publicaciones (`poll` y `push`) seleccionando aleatoriamente un nodo (procedimiento similar al que se realiza en la política Random). Durante esta etapa el nodo adquiere información acerca de la red que lo rodea.

- **Etapa de Aplicación**: una vez que tiene *suficiente* información sobre la red de la que forma parte, comienza a seleccionar los nodos de su lista de vecinos (neighbors), eligiendo el mejor de ellos según un ranking que cada nodo confecciona en base a la información adquirida en la etapa anterior. Este ranking se basa en una función que utiliza los parámetros que conoce de la red (i.e. Round Trip Time del enlace o `RTT`) y/o de los otros nodos (i.e. poder de cómputo). Esta función se denomina *función de scoring*. Una vez iniciada esta segunda etapa, se mantiene un bajo porcentaje de selecciones aleatorias para evitar máximos locales y también para absorver los efectos negativos de eventuales fallas en los nodos.

La implementación estándar de (**BN**) utiliza la siguiente función de scoring para determinar el ranking de vecinos [14]:

$$f_{\text{scoring}} = a \cdot \texttt{INFO\_RES} + b \cdot \texttt{RTT} + c \cdot \texttt{RESPONSE\_FAILED}$$

donde `INFO_RES` refiere a la cantidad de información sobre recursos que posee el host vecino, `RTT` refiere al Round Trip Time y `RESPONSE_FAILED` cuenta el número de mensajes perdidos. a, b, y c son parametros que determinan la relevancia de cada variable, y si su

influencia es positiva (a) o negativa (b,c).

`INFO_RES` representa la cantidad y actualidad de la información sobre recursos que los hosts proveen y `RTT` es el valor que representa el peso de la latencia entre cada par de nodos. En la implementación utilizada de Best Neighbor, el valor de este segundo parámetro se calcula, para cada host, teniendo en cuenta el promedio de las diferencias entre el envío de una solicitud y la recepción de la respuesta de todas solicitudes realizadas a ese host. Es decir no es estrictamente el valor del `RTT`.

Por otro lado el valor del `RTT` para Gridmatrix depende del valor de la latencia asignada a los enlaces en la configuración de la grid, al igual que para los recursos, sólo que en este caso debe considerarse la descripción del modelo de red utilizado por Gridmatrix, es decir el modelo de red de SimGrid [22] que es el framework sobre el que se desarrolló Gridmatrix [14, 18], del que se puede obtener una descripción detallada en Casanova et al. [23].

Es razonable esperar que la información global con la que cuenta cada nodo sea *mejor* que si se estuviera utilizando la política Random, dado que incluye mayor esfuerzo computacional en la generación del score para guiar la selección del nodo a consultar, y que esto se vea reflejado en una mejor performance.

Contrariamente a lo esperado, los resultados iniciales utilizando la política Best Neighbor en diversos escenarios mostraron un desempeño pobre (como puede verse en la Figura 2.5 para sistemas de topología Exponencial), similar a la política Random, por lo que se decidió buscar las razones causantes de este comportamiento.

Al analizar la implementación de la política Best Neighbor es posible reconocer tres aspectos fundamentales:

1. La cantidad de nodos que conoce cada nodo en relación a la cantidad de nodos totales en la red.

2. La manera en que un nodo obtiene información de los demás nodos y actualiza la información que posee sobre los nodos que considera vecinos.

3. La función que se utiliza para elegir el mejor vecino (función de scoring).

En el siguiente Capítulo se analiza cada uno y se proponen soluciones a los problemas encontrados.

4. RESULTADOS

En este Capítulo se estudian en detalle los aspectos fundamentales de Best Neighbor que fueron identificados en el Capítulo 3, y se proponen algunas soluciones a los problemas encontrados, corresponde a los aportes principales de la presente tesis.

En la Sección 4.1 se identifican las intervenciones aleatorias en la implementación de BN que podrían estar influyendo de manera negativa en la performance. Se proponen dos estrategias para reducir la Etapa de Aprendizaje y se analiza el impacto de las mismas en la performance.

En la Sección 4.2 se profundiza sobre la forma de selección de nodos destino para solicitudes en cuanto a la función de scoring utilizada. Se estudian las magnitudes que esta función observa, que son los componentes que intervienen en la generación del ranking, de qué manera influye cada una de ellas, y se propone la inclusión de un nuevo componente a la misma. Se observa la influencia de una implementación de esta propuesta en la performance en varias topologias en sistemas de distintos tamaños.

Finalmente en la Sección 4.3 se experimenta con diversos mecanismos para la publicación de recursos y se estudia cómo influyen en la performance del sistema. Las soluciones propuestas para algunos de los problemas encontrados constituyen dos nuevas políticas de distribución de la información sobre recursos completamente distribuidas basadas en BN y el principal aporte de este trabajo. Sobre la implementación de las mismas se estudia, en la Seccion 4.4, su performance para sistemas de distintos tamaños en diversas topologías.

4.1. Conociendo vecinos

En la implementación de la política Best Neighbor, se mantiene la cantidad de vecinos que cada uno de los nodos *conoce* de la red, es decir, un nodo puede saber con qué porción del sistema se ha comunicado alguna vez. A pesar de que en un escenario real el total de nodos del sistema podría no ser conocido, puede recurrirse en ese caso a alguna estimación del mismo, al máximo registrado o a alguna otra métrica que resulte adecuada, ya que lo importante será distinguir en qué momento un nodo conoce *suficiente* del sistema.

Aquel host que va a efectuar una solicitud (llamado *nodo origen*) determina si va a seleccionar un *nodo destino* de manera aleatoria o si lo hará mediante la utilización de la información adquirida previamente. La expresión utilizada para determinar el modo de selección en un nodo i es:

$$c_i = \frac{|\texttt{vecinos}(i)|}{\texttt{hostCount}}$$

donde $\texttt{vecinos}(i)$ es la lista de vecinos conocidos por el nodo i (incluye a aquellos nodos que respondieron una solicitud de recursos realizada por el nodo), en tanto que `hostCount` es la cantidad total de nodos presentes en el sistema.

El aumento de c_i permite el cambio de etapa, pasando de la *Etapa de Aprendizaje* a la *Etapa de Aplicación*, cuando el valor de c_i indica que el nodo conoce suficiente del sistema. Durante la primera etapa, cada nodo utiliza la política Random y en la segunda envía solicitudes prioritariamente a nodos que se encuentren en su lista de vecinos, realizando consultas aleatorias con muy baja probabilidad. En este último caso, la selección del nodo

a utilizar es guiada por la función de scoring. Es decir, deja de comportarse siguiendo la política Random para comenzar a funcionar como Best Neighbor propiamente dicho.

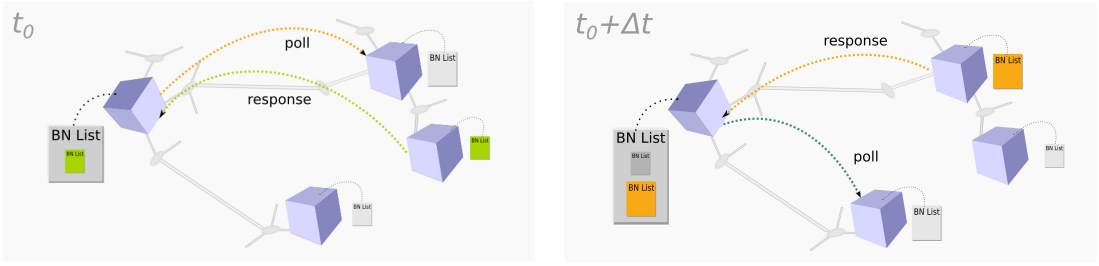
Se observó en sistemas de topología exponencial de hasta 1000 nodos que el valor de c_i aumenta muy lentamente. A modo de ejemplo, en una red de 400 nodos sólo se llega a conocer un 8 % del sistema luego de 2000 segundos de simulación realizando una solicitud por segundo.

Se considera que la lentitud con que crece este valor puede deberse a que en cada `poll` sólo se conoce, a lo sumo, un nuevo nodo (o *nuevo vecino*), de manera similar a lo observado para la política Random en la Sección 3.3 (página 27). Esto hace que en una red de 100 nodos, c_i se incremente a razón de 0,01 por `poll`, si no se repiten los destinos. Al aumentar la cantidad de nodos esta situación empeora. En una red de 1000 nodos, el incremento del indicador a 0,001 por `poll` puede tornarse en una situación crítica. El crecimiento tan suave de c_i prolonga demasiado la *Etapa de Aprendizaje*, haciendo que Best Neighbor se comporte durante un período muy prolongado del mismo modo que Random.

Analizando más profundamente este comportamiento, se verifica que la implementación base de la política Best Neighbor de la que parte este trabajo, utiliza eventos de `push` y de `poll`, pero para los eventos de tipo `push` se utiliza siempre la política Random independientemente de lo seleccionado por el nodo para el `poll` o en qué etapa se encuentre. De este modo, se decide deshabilitar los eventos de `push` y se aumenta de manera coherente la frecuencia de los `poll` para mantener la cantidad de mensajes por período de tiempo.

Queda aún el comportamiento Random durante la etapa de Aprendizaje. Una posibilidad para acortar esta etapa es modificar el valor que regula el cambio de etapa.

Otra estrategia alternativa para disminuir la duración de la Etapa de Aprendizaje consiste en realizar un *merge de listas*. En la Figura 4.1 se esquematiza el proceso. Al realizar un `poll`, cada nodo recibe, eventualmente, una respuesta del nodo destino de ese `poll`. En la respuesta, el nodo podría enviar también el conocimiento que ya posee del sistema, es decir, su propia lista de vecinos. Al recibir la respuesta, el nodo que originó el `poll` utiliza esta información para actualizar la propia, realizando un merge de las listas de vecinos.



(a) Se envía una solicitud en t_0

(b) Al obtener respuesta, al cabo de un Δt, se realiza el merge de listas

Fig. 4.1: Merge de listas. Al recibir respuestas (`response`) a las solicitudes realizadas (`poll`) los hosts actualizan la información de sus listas de vecinos con la información de la lista de vecinos del host que respondió a la solicitud. Este mecanismo permite que todos los nodos alcancen un conocimiento completo de los recursos en la red en menor tiempo.

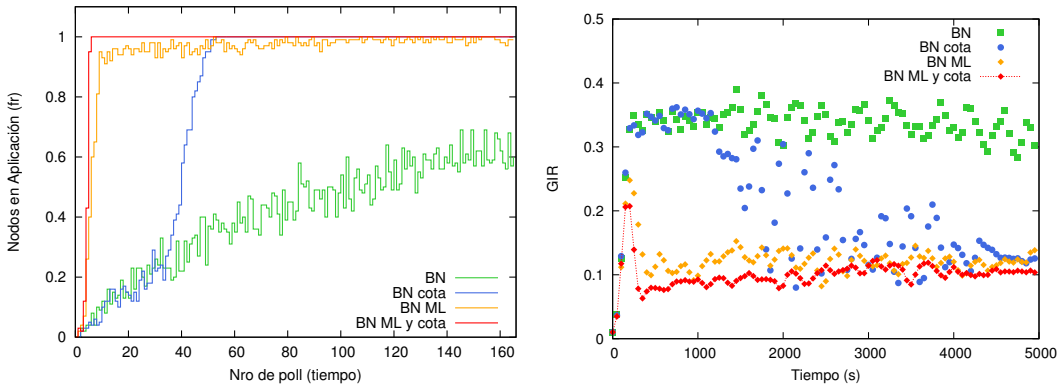
Para la primera estrategia definimos e como la proporción de nodos que se debe conocer

para pasar de la etapa de Aprendizaje a la de Aplicación.

Para confeccionar la Figura 4.2(a), en cada paso de la simulación se registra en qué etapa se encuentra cada nodo de un sistema de 100 nodos y 50 routers con topología exponencial, determinándose así cuál es la proporción de nodos que se encuentra en la segunda etapa de la política Best Neighbor del total de nodos que realizan un `poll`.

BN cota es el resultado de utilizar $e = 0{,}3$ como cota (en azul), en tanto que BN muestra el comportamiento sin cota (en verde).

Observamos que la utilización de este límite reduce bastante la etapa de Aprendizaje. Sin embargo, esto se traduce en una caída de la performance, como puede apreciarse en los resultados del GIR incluídos en la Figura 4.2(b), esta política presenta peores resultados comparados con la versión de la política sin modificar. Esto podría deberse a que los nodos pasan a la Etapa de Aplicación sin suficiente conocimento del sistema para la correcta selección del mejor vecino con el cual comunicarse.



(a) Nodos en etapa de Aplicación.

(b) GIR (100 nodos).

Fig. 4.2: 4.2(a) Frecuencia relativa de nodos en etapa de Aplicación. Se simularon 5000 pasos temporales en una red de topología exponencial de 50 routers y 100 nodos. Cada nodo realiza alrededor de 160 solicitudes en el período simulado. En cada **poll** los nodos pueden elegir un nodo utilizando la política Random (etapa de Aprendizaje) o Best Neighbor (etapa de Aplicación) si ya tiene suficiente conocimiento de la red y han pasado a la segunda etapa de la política. Se ve claramente que una mayor proporción de nodos pasan más tempranamente a la segunda etapa de la política cuando se utiliza una cota para esta etapa, y lo hace casi inmediatamente cuando se utiliza el *merge* de las listas de vecinos. 4.2(b) Evolución del GIR para cada una de las mejoras introducidas. Los valores del GIR para Best Neighbor al acortar la etapa de Aprendizaje disminuyen fuertemente.

En cuanto a la segunda estrategia descripta, al introducir el merge de listas (**ML**) modificando la política BN para que en cada respuesta a un `poll` se obtenga la información de los vecinos del nodo que respondió, agregando la lista del *nodo destino* a la propia lista de vecinos del *nodo origen*, siguiendo el mecanismo presentado en la Figura 4.1, observamos que los nodos pasan inmediatamente a la Etapa de Aplicación al utilizar BN con cota (BN ML y cota), pero aún sin la introducción de la cota el cambio de etapa en la mayoría de los nodos se realiza mucho antes (BN ML) que con la primera estrategia presentada o la versión original de BN, Figura 4.2(a).

Sin embargo, las modificaciones introducidas no producen mejoras sino que hacen disminuir el valor del GIR, como puede verse en la Figura 4.2(b).

Hasta aquí analizamos como afecta el conocimiento de los demás nodos en el cambio de etapa de Best Neighbor y la forma en que cada nodo obtiene información sobre los demás nodos en la grid, y encontramos una estrategia para reducir la etapa de Aprendizaje de

Best Neighbor, lo que nos permitió identificar el comportamiento de esta política sin la influencia de la selección aleatoria de nodos.

Esta estrategia disminuye fuertemente la performance observada hasta el momento para la política, aunque aún resta analizar los otros elementos que influyen el comportamiento de la política: la función de scoring utilizada y cómo cada nodo actualiza la información de sus vecinos.

4.2. fBN: Ajustando la función de scoring

El hecho de que la política Best Neighbor tenga un desempeño pobre aún en el caso en que tiene información de la mayoría de los nodos de la red, lleva a analizar la manera en que el host utiliza la información que recibe de sus vecinos o, más precisamente, cómo utiliza esa información para elegir el *mejor* entre ellos.

La función de scoring que evalúa cada nodo para seleccionar el mejor candidato de su lista de vecinos, podría ser cualquier función con dominio en los valores de los parámetros que caractericen los nodos o los servicios que estos proveen, y cuya imagen admita orden parcial.

Una metodología podría ser seleccionar el nodo que presente valores altos del GIR, intentando obtener una mayor cantidad de información del sistema en cada mensaje. Elegir el nodo con información de mayor calidad permitiría tener mayor seguridad sobre la información de los recursos con la que cuenta y, en consecuencia, aprovechar mejor los recursos.

Al cabo de cada intervalo de `poll`, un nodo n_i elegiría el mejor vecino usando:

$$f_{\text{scoring}}(n_i) = \max_{i \neq j \wedge n_j \in \text{vecinos}(n_i)} (\texttt{INFO\_RES}(n_j))$$

donde f_{scoring} es la función que se utiliza para seleccionar el *mejor* vecino de la lista de vecinos conocidos, n_i es cualquiera de los nodos del sistema, y vecinos(n_i) es la lista de vecinos del host n_i.

Podría ocurrir que más de un host provea información con igual calidad pero uno se encuentre más lejos que otro o a través de una ruta temporalmente congestionada, para estos casos sería importante tener en cuenta la información del tiempo de comunicación entre tales hosts, es decir el `RTT`, en el cálculo del mejor vecino, lo que se logra incluyendo esta magnitud como parámetro en la función.

Cuanto mayor sea el valor del `RTT` más negativo se lo considerará, por lo que la función de scoring quedará:

$$f_{\text{scoring}}(n_i) = a \cdot \texttt{INFO\_RES}(n_i) + b \cdot \texttt{RTT}(n_i), (b < 0)$$

en donde los parámetros a y b se utilizan para ajustar el peso relativo de cada una de las magnitudes en la función[1].

Al ser la latencia una magnitud del orden de los milisegundos, suponiendo una latencia en los enlaces de 0,1ms, en un sistema con mil nodos y topología de línea, se obtendría un `RTT` máximo de 0,2s. En los escenarios estudiados en este trabajo, este valor no resulta significativo si se lo compara con el total de recursos del sistema al suponer, por ejemplo,

[1] Agregando como parámetro la cantidad de respuestas perdidas se obtiene la función estándar mencionada anteriormente y usada en trabajos previos.

un recurso disponible por nodo. Esta diferencia entre las magnitudes involucradas produce la necesidad de introducir en las constantes a y b valores que no parecen tener una relación inmediata con el problema en estudio. Además, las diferencias entre sistemas con distintas distribución de recursos y/o infraestructura de red tienen como consecuencia que los ajustes hechos a las constantes para un escenario sean difíciles de utilizar bajo otras condiciones. Es decir, un host que provee x recursos de un total de $100 \cdot x$ en el sistema no puede ser pesado de la misma forma que un nodo que provee x recursos de un total de $1000 \cdot x$.

Concentrándonos sólo en la información sobre los recursos que el host puede proveer y la latencia, se decide normalizar la función de scoring de modo de obtener un valor entre 0 y 1, tanto para `INFO_RES` como para `RTT`.

Para normalizar el `RTT`, se utiliza el máximo `RTT` en un sistema con topología de línea con la misma cantidad de routers, lo que resulta en una cota para el máximo camino posible. Para `INFO_RES`, dado que este valor representa la cantidad de información sobre los recursos en el sistema y cuán actual esta es, se normalizará a partir de considerar que todos los hosts tienen la información de todos los recursos y todos los recursos se encuentran disponibles, es decir, la información es lo más actual que puede ser.

$$f_{\text{scoring}} = a_1 \cdot \frac{\texttt{INFO\_RES}}{\texttt{totalResCount} \cdot \texttt{hostCount}} + b_1 \cdot \frac{\texttt{RTT}}{\texttt{maxLineRTT(routersCount)} \cdot \beta}$$

Donde a_1 y b_1 son constantes y β se introduce como ajuste a los resultados relacionados con la infraestructura de red teniendo en cuenta el modelo utilizado por el simulador, como se mencionó en la Sección 2.2 (página 11).

Tomando $a_1 = a \cdot \texttt{totalResCount} \cdot \texttt{hostCount}$ y $b_1 = b \cdot \texttt{maxLineRTT(routersCount)} \cdot \beta$, con a y b las constantes utilizadas en los resultados previos, se recupera la función original.

Simplificando la notación:

$$f_{\text{scoring}} = a_1 \cdot \lambda_{IR} + b_1 \cdot \lambda_{RTT}$$

con $\lambda_{IR} = \frac{\texttt{INFO\_RES}}{\texttt{totalResCount} \cdot \texttt{hostCount}}$ y $\lambda_{RTT} = \frac{\texttt{RTT}}{\texttt{maxLineRTT(routersCount)} \cdot \beta}$, $(b_1 < 0)$

Para poder elegir valores a_1 y b_1 realizamos el siguiente análisis. `INFO_RES` es una función creciente que puede tomar valores muy grandes, mientras que `RTT` es casi constante, aunque una vez normalizada la función de scoring los valores de λ_{IR} resultan muy pequeños en relación a los valores de λ_{RTT}.

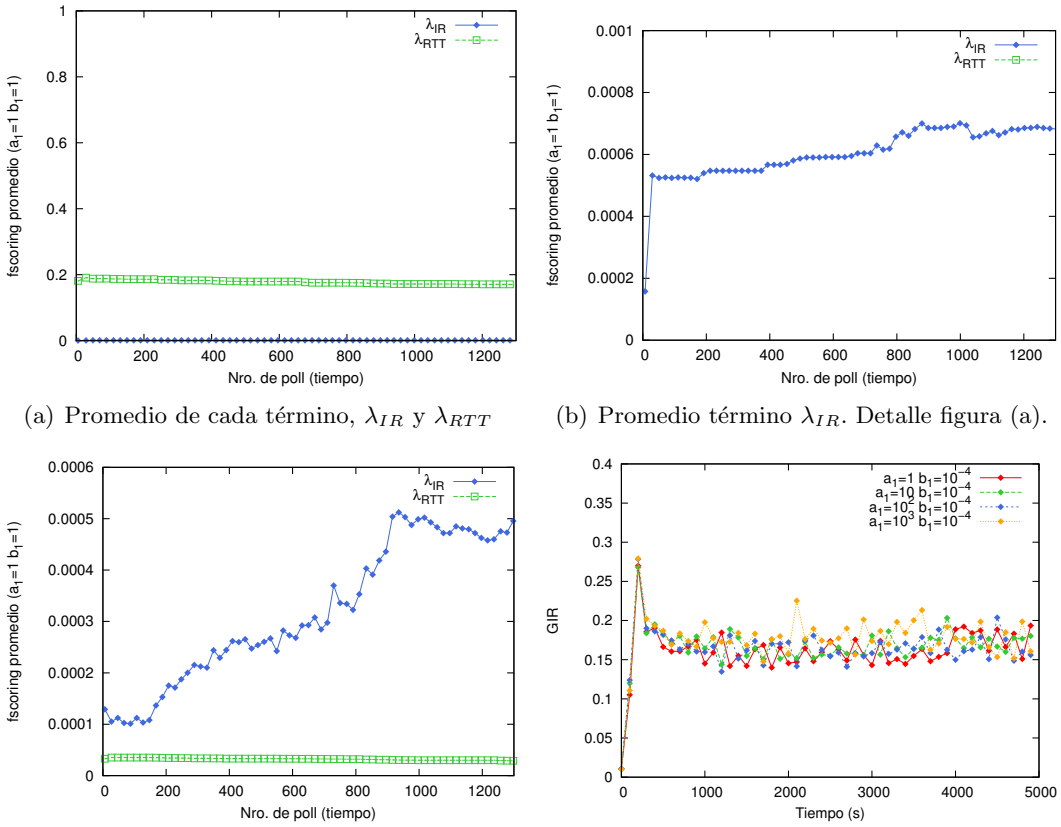
En la Figura 4.3(a) se encuentran los promedios en función del número de `poll` (tiempo), de λ_{IR} y λ_{RTT}, calculados a partir de los valores que toman para cada host conocido en un host determinado n_i, con valores para $a_1 = 1$ y $b_1 = 1$ (en la Figura $n_i = Host\_27$). Es decir:

$$\overline{f_{\text{scoring}}}\Big|_{n_i} = \frac{1}{N-1} \sum_{n_j \in \text{vecinos}(n_i)} \lambda_{IR}(n_j)$$

$$\overline{f_{\text{scoring}}}\Big|_{n_i} = \frac{1}{N-1} \sum_{n_j \in \text{vecinos}(n_i)} \lambda_{RTT}(n_j)$$

Al modificar la escala para poder observar mejor el comportamiento de λ_{IR}, Figura 4.3(b), se ve que que las prioridades planteadas están invertidas durante cierto tiempo y no indefinidamente, ya que al ser creciente y acotada en $[0, 1]$, en algún momento λ_{IR} comenzará a tener más peso que λ_{RTT}.

El momento en el cuál esto sucede es distinto para cada host y, posiblemente, para cada sistema. También depende de la performance del sistema, por lo que parece mejor fijar esa relación desde el inicio. Observamos en los resultados que los valores iniciales que toman λ_{IR} y λ_{RTT} pueden equipararse agregando al segundo término la constante 10^{-4}, Figura 4.3(c), quedando la constante $b_1 = 10^{-4} \cdot b'_1$.



(a) Promedio de cada término, λ_{IR} y λ_{RTT}

(b) Promedio término λ_{IR}. Detalle figura (a).

(c) Términos equiparados.

(d) GIR para distintos valores de a_1.

Fig. 4.3: Al normalizar, λ_{IR} sigue siendo una función creciente, pero que crece muy suavemente, mientras que λ_{RTT} es casi constante con valores varios órdenes de magnitud mayor. Al tener λ_{IR} una pendiente tan suave la mayor parte del tiempo quedan invertidas las prioridades, teniendo más peso `RTT` que la información de recursos, aunque al ser está última creciente y acotada entre 0 y 1, en algún momento comenzará a tomar valores mayores o iguales que `RTT`. Una forma de forzar esta condición es agregar una constante al segundo término, λ_{RTT}, que permita equiparar los valores desde el inicio de la simulación. En las Figuras (a) y (b) se pueden observar los resultados de cada una de las magnitudes que intervienen en la función de scoring en función del tiempo, tomando el promedio de todos los vecinos para los que se calcula la función en un nodo determinado. En la Figura (c) se ven los promedios una vez agregada la constante. En (d) se ve la evolución del GIR en un sistema de 100 nodos para distintos valores del parámetro a_1.

Con este valor para b_1 y tomando $b'_1 = 1$ se obtienen los resultados para el GIR de la Figura 4.3(d) en un sistema de 100 nodos, para distintos valores de a_1. Si bien los mejores resultados son los obtenidos cuando se utiliza $a_1 = 10^3$, la elección del valor de esta constante en la función normalizada muestra no tener mucha influencia en la performance. Para estas pruebas, sólo se realizan eventos de `poll` y se utiliza el merge de listas presentado en la sección anterior.

Otro punto a tener en cuenta es que, a pesar de tener un LIR alto, un nodo podría tener una baja cantidad de recursos propios que ofrecer o compartir, es decir, que si bien puede poseer información actualizada sobre sus vecinos, el nodo podría carecer de recursos

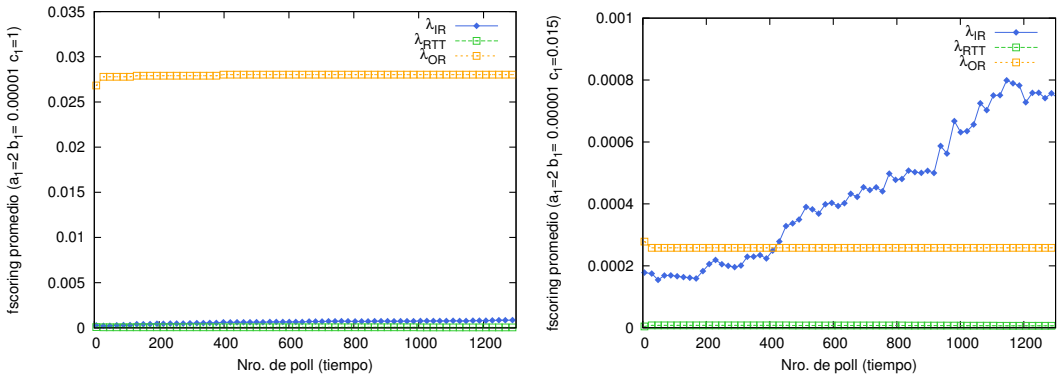
propios que informar, con lo que se estaría ante la posibilidad de sólo estar utilizando información de *segunda mano*.

Una forma de atacar esta situación es pesar en la función de scoring no sólo la cantidad de recursos informados por cada nodo, sino también la cantidad de recursos propios que posee, de manera tal de poder elegir aquel nodo que tiene buena información sobre los vecinos y, a su vez, la mayor cantidad de recursos propios. Los recursos de cada host (`OWN_RES_COUNT`) dependen de los valores asignados en los archivos deploy a los nodos, este valor puede ser normalizado para que en la función el término tome valores entre 0 y 1, teniendo en cuenta la cantidad de recursos totales en el sistema. Con este agregado, la función de scoring queda hasta aquí:

$$f_{\text{scoring}} = a_1 \cdot \lambda_{\texttt{IR}} + b_1 \cdot \lambda_{\texttt{RTT}} + c_1 \cdot \lambda_{\texttt{OR}}$$

dónde $\lambda_{\texttt{OR}} = \frac{\texttt{OWN\_RES\_COUNT}}{\texttt{totalResCount}}$ representa la proporción de recursos propios que posee el nodo en relación al total del sistema y $b_1 < 0$.

Corresponde analizar cuál es el valor más conveniente para la constante c_1. Repetimos las simulaciones relevando en este caso también el valor de $\lambda_{\texttt{OR}}$ según el transcurso del tiempo, como promedio de los valores de los vecinos, con los resultados se genera la Figura 4.4(a). Como ocurre con el término que representa la latencia, el valor de este término es constante y mucho mayor que el de $\lambda_{\texttt{IR}}$. Con el mismo criterio que se expuso anteriormente se elige un valor para c_1 que permita comparar los términos. En este caso, utilizando $c_1 = 0{,}015$ se obtiene una relación que permite priorizar los recursos propios cuando todavía los nodos no cuentan con suficiente información del resto de la red y luego priorizar `INFO_RES`. Los resultados que se incluyen en la Figura 4.4(b) se obtienen a partir de las mismas simulaciones anteriores, Figura 4.4(a), pero utilizando $c_1 = 0{,}015$.



(a) Promedio de cada término $c_1 = 1$ (b) Promedio de cada término $c_1 = 0{,}015$

Fig. 4.4: $\lambda_{\texttt{OR}}$ es constante, con valores varios órdenes de magnitud mayor que $\lambda_{\texttt{IR}}$. Como en el caso anterior se agrega una constante a $\lambda_{\texttt{OR}}$ (denominada c_1) para equiparar los valores desde el inicio de la simulación. Los resultados en la Figura (a) corresponden a cada una de las magnitudes que intervienen en la función de scoring y su evolución en función del tiempo, tomando el promedio de todos los vecinos para los que se calcula la función en un nodo determinado. La Figura (b) muestra los promedios una vez agregada la constante.

Una vez normalizada la función de scoring se analiza su desempeño en redes de distinto tamaño y distintas topologías.

La Figura 4.5 presenta resultados obtenidos para el GIR promedio en sistemas de entre 100 y 1000 nodos utilizando topología Exponencial. Estos promedios corresponden

a simulaciones cuya duración se prolonga hasta que los mismos no evidencian diferencias perceptibles con simulaciones de mayor duración. La duración de las simulaciones es de $3000s$ salvo en el caso de fBN, en que fue necesario prolongarlas hasta $5000s$. Puede decirse que, para sistemas de menor tamaño, el valor de c_1 puede influir notablemente en la performance, mientras que a medida que aumenta el tamaño del sistema esta influencia es menos apreciable.

En relación a las otras políticas de distribución de información sobre recursos observamos que con las modificaciones realizadas a Best Neighbor, la performance de fBN finalmente se aleja de la de Random, y que, para sistemas de tamaño mayor a 300 nodos, la performance es tan buena en promedio como la obtenida al utilizar *5-SP*, evidenciando una importante mejora respecto de los resultados de la 2.5.

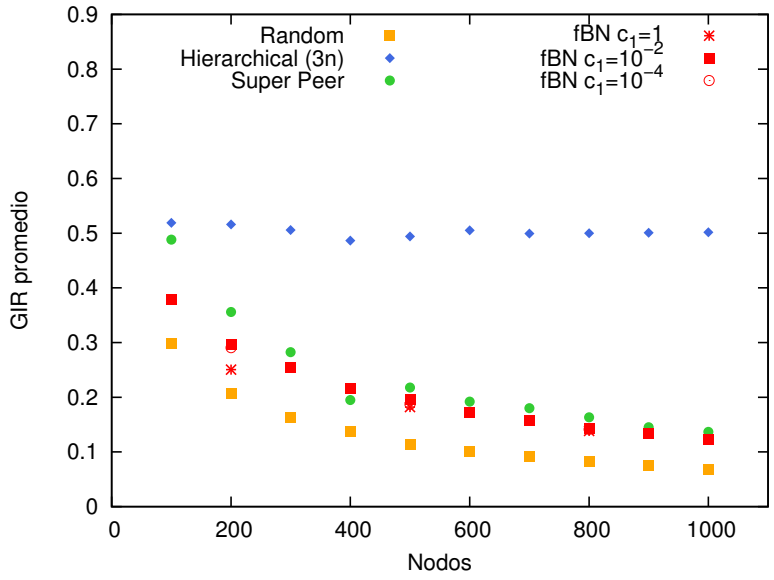
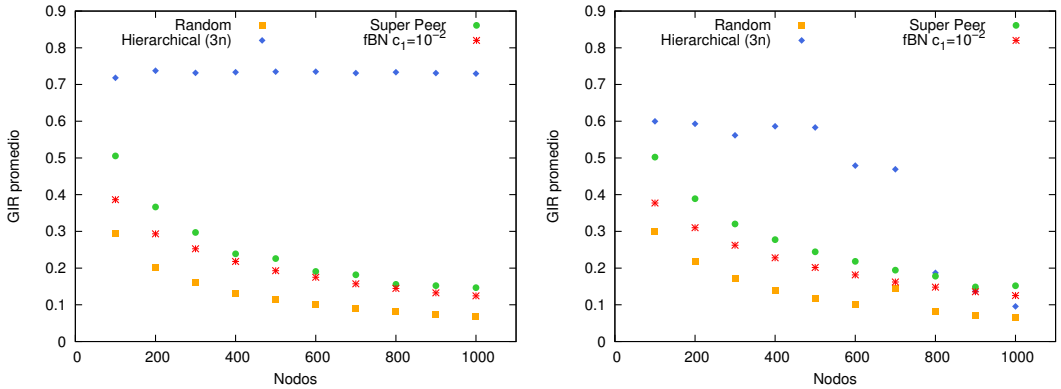


Fig. 4.5: GIR promedio para las políticas de distribución Jerárquica, Super Peer (*5-SP*), Random y fBN: $a_1 = 1, b_1 = 10^{-4} c_1 = 1, 10^{-2}, 10^{-4}$, para redes de topología Exponencial con distintas cantidades de nodos (100 a 1000 nodos).



(a) Clique (b) Anillo

Fig. 4.6: GIR promedio para las políticas de distribución Jerárquica, Super Peer (*5-SP*), Random y fBN: $a_1 = 1, b_1 = 10^{-4} c_1 = 10^{-2}$, para redes de topología Clique, Figura (a), y Anillo, Figura (b), con distintas cantidades de nodos (100 a 1000 nodos).

La Figura 4.6 introduce los resultados obtenidos al realizar las simulaciones en condiciones similares a los de la Figura 4.5, pero en este caso se utilizan topologías Clique y Anillo con distintas cantidades de nodos. En la Figura 4.6(b), se observa para sistemas entre 100 y 1000 nodos que las políticas se comportan de manera similar a como se comportan sobre una topología Exponencial. Jerárquica muestra mejor performance que 5-SP y ésta, a su vez, mejor performance que Random, aunque con algunas diferencias. En el caso de Jerárquica, los resultados para Clique son muy superiores que los obtenidos en Anillo e, incluso, en Exponencial. Además, en el caso de Anillo se observa que para sistemas de tamaño mayor a 500 nodos la performance se deteriora a medida que crece el tamaño del sistema, llegando a ser semejante a la performance de Random en sistemas de 1000 nodos.

Los motivos de este deterioro pueden buscarse en el hecho de que una jerarquía de tres niveles en una topología de Anillo significa que los nodos en el último nivel deben esperar que sus mensajes recorran entre la cuarta parte y la mitad del sistema antes de ser distribuidos por el *root* de la jerarquía. Adicionalmente, las respuestas a sus solicitudes llegan con una demora aún mayor. El aumento en el tamaño del Anillo impacta notablemente ya que el camino que deben recorrer los mensajes crece tanto como nodos se agreguen, sin olvidar el incremento de la cantidad de mensajes, lo que puede producir que la información contenida en los mensajes expiren mucho antes de llegar a destino o que lleguen con información muy desactualizada. Queda pendiente realizar un estudio en mayor profundidad de este tema, ya que queda fuera del alcance planteado para esta trabajo.

Por otro lado, 5-SP se ve comporta mejor en Anillo que en Clique o Exponencial, aunque con valores de GIR muy por debajo de los obtenidos por Jerárquica. La política fBN mantiene la misma performance en Clique y Anillo que la obtenida sobre la topología Exponencial, con resultados semejantes a los de 5-SP.

4.3. pBN y xBN: Agregando el push a fBN

A medida que el tiempo transcurre, la información que posee cada nodo sobre el resto del sistema va desactualizándose. La publicación de recursos (`push`), permite informar a otros nodos qué recursos propios están disponibles. Al llegar una publicación de recursos de algún host es posible agregarlo a la lista de vecinos, actualizar la información que se tiene en la propia lista de vecinos (si es que el host ya pertenecía a esta lista), o realizar el merge de listas conservando la información más actual sobre los nodos en las listas de vecinos de ambos.

En la sección anterior, los experimentos realizados excluyeron los eventos de `push`. En esta sección se agregan dichos eventos de manera de que cada nodo publica sus recursos (realiza un `push`) eligiendo un host al que enviar su información, inicialmente de manera aleatoria (`pushRandom`) y luego utilizando f_{scoring} (`pushBN`).

La Figura 4.7(a) presenta los resultados de la evolución del GIR para distintas variaciones de la política fBN, presentada en la Sección 4.2, pero con el agregado de los eventos de `push`:

- `push Random` : corresponde a elegir aleatoriamente un nodo al cual enviarle la información de recursos propios.
- `push Random y ML` : se elije aleatoriamente un nodo, se envía la información de recursos propios y el resto de la información recolectada sobre el sistema. El host

destino realiza el *merge* de listas.

- `push BN` : se elige el nodo destino utilizando la función de scoring (política best neighbor) y se envía sólo la información de recursos propios.
- `push BN y ML` : se elige el nodo destino utilizando la función de scoring (política best neighbor), se envía toda la información almacenada y el nodo destino realiza el *merge* de listas.
- `push BN, ML y LP20` : se agrega al punto anterior un control adicional que impide repetir el mismo host destino de la consulta más de una cierta cantidad de veces a partir de las 20 comunicaciones.

Agregar los eventos de `push` mediante la selección aleatoria del destino no muestra una mejora significativa ni aún mediante la incorporación del mecanismo de merge de listas.

Al enviar una publicación de recursos se intenta que los recursos propios queden disponibles a quien los necesite, realizar el `push` a un nodo elegido aleatoriamente no garantizaría que quien los necesite se entere a tiempo de la disponibilidad de estos recursos.

El host al cual el nodo n_i realizó el último `poll` (seleccionado por medio de $f_{BN}(n_i)$) es el nodo que ha sido identificado como con mejor información sobre el sistema. Esto hace que sea más probable que sea elegido como destino por otros nodos y, a su vez, surge como mejor candidato para recibir una publicación de recursos (destino de `push`). Sin embargo, la Figura 4.7(a) muestra que utilizar esta técnica para seleccionar el destino del `push` produce un leve decaimiento (en promedio) respecto a la selección aleatoria, en tanto no parece haber diferencias si se utiliza el *merge* de listas.

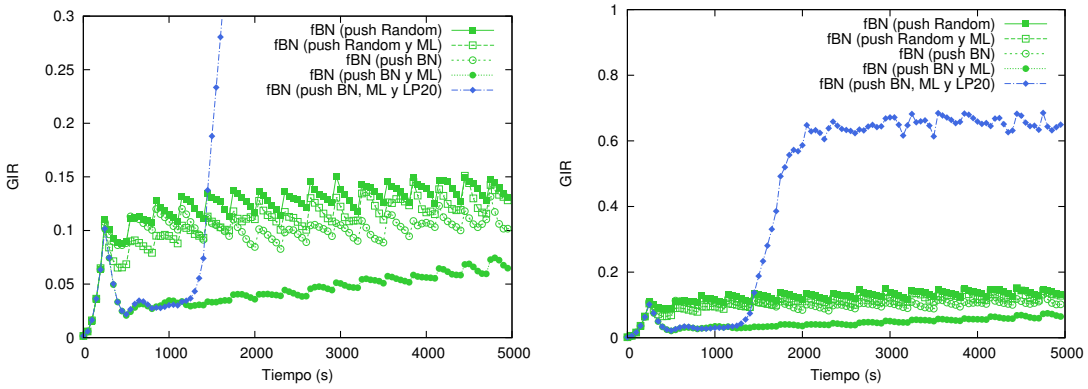
Al analizar las secuencias de destinatarios de `polls` de cada hosts, es decir la lista de best neighbors elegidos a lo largo del tiempo, se encuentra que el sistema realiza una clusterización de destinos. Luego de un periodo corto de tiempo, los nodos repiten la elección del destino. Estudiando los resultados de la función de scoring para esos casos, se observó que el resultado de f_{scoring} para el best neighbor da un valor muy superior que el del host que obtiene el segundo lugar. Adicionalmente, se verificó que esta diferencia se incrementa con el transcurrir de los mensajes.

Es decir, el nodo queda "imposibilitado" de elegir otro nodo distinto al que ya venía eligiendo. Se forman pequeños grupos de nodos que sólo obtienen información adicional sobre el sistema cuando realizan una solicitud aleatoria, y esto sucede con muy baja probabilidad.

Para alivianar esta situación, se agrega un mecanismo de control que no permite la repetición sistemática en la selección del mejor vecino: cada nodo mantiene un registro de los últimos hosts que fueron elegidos como mejor vecino, es decir, una lista de los destinos de los últimos `polls` (LP). Luego de que se han realizado una cierta cantidad de `polls` en la etapa de Aplicación, se busca evitar que un nodo no quede asociado a un mismo nodo forzando al segundo en su lista como destino.

Los resultados de este experimento comenzando la verificación cuando la lista alcanza los 20 `polls` ($|length(LP)| \geq 20$), Figura 4.7(b), son muy diferentes a los obtenidos hasta el momento. Al cabo de unos 1200s, se ve un crecimiento del GIR que se estabiliza aproximadamente a los 2000s en un valor de 0,6, indicando muy buena performance, estable, en el sistema a partir de ese momento.

Basándonos en las características de la información con la que cuenta el sistema sobre sí mismo, podemos distinguir dos etapas dentro de la Etapa de Aplicación al utilizar LP:



(a) 500 nodos (b) 500 nodos (escala ampliada)

Fig. 4.7: Evolución del GIR para un sistema de 500 nodos al agregar distintas estrategias para la selección del nodo destino en los eventos de `push`, con y sin ML (*Merge de Listas*). Al utilizar *pushRandom* cada nodo elige de manera aleatoria el nodo que recibirá la publicación de recursos, al utilizar pushBN el nodo al que se realizará el `push` es el nodo al que se le realizó el último `poll` (seleccionado por la función de scoring). Los resultados en el detalle de la Figura (a) no muestran mejoras evidentes en la performance al agregar los eventos de `push`, aunque si se observa una leve pérdida en la performance cuando se utiliza el pushBN en conjunto con ML. Evitar la clusterización temprana por medio del control de repeticiones de destinos de `poll` (LP (*Lista de Polls*) produce una mejora notable en la performance.

i) Etapa de Crecimiento, y ii) una Etapa de Estabilidad.

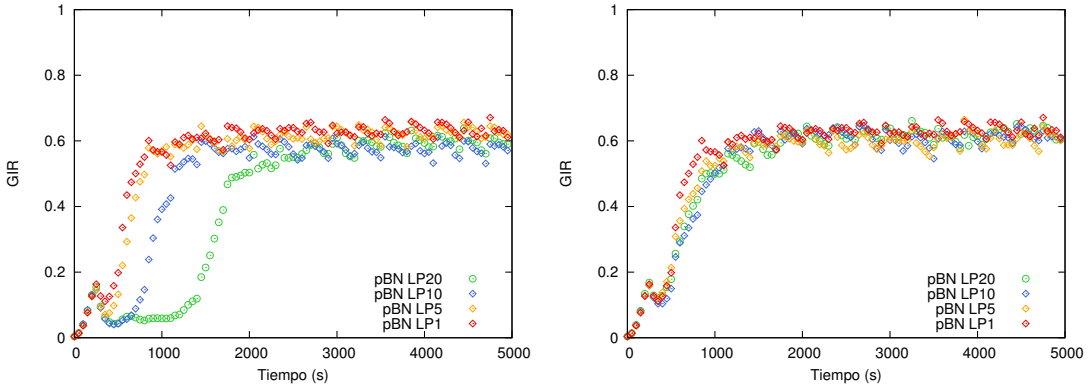
La Etapa de Crecimiento puede acortarse reduciendo el período de espera para comenzar el control de repeticiones. La Figura 4.8 presenta el comportamiento de un sistema de 200 nodos en el que se ha realizado una variación de las longitudes de LP. La Figura 4.8(a) se observa el GIR en función del tiempo para distintas longitudes de LP. Si bien los resultados obtenidos no presentan una variación sustancial en la Etapa de Estabilidad, sí se obtiene mejor performance al dismiuir la longitud de LP. El caso extremo de llevar esta longitud a *cero* corresponde con comenzar la verificación de no-repetición desde el inicio mismo de la Etapa de Aplicación.

La Figura 4.8(b) presenta el comportamiento de la política manteniendo la longitud del período de espera pero realizando durante el mismo un control simple para que no se efectúen dos `polls` seguidos al mismo host, en este caso, si la función de scoring elige dos veces seguidas el mismo, se enviará la solicitud al segundo mejor calificado (se hace un *swap*).

Los resultados obtenidos en los sistemas evaluados fueron similares a los incluidos en la Figura 4.8(a). Esto abre la posibilidad de utilizar cualquiera de las dos opciones para evitar la repetición sistemática en los envíos de solicitudes. Los dos mecanismos evaluados mostraron una muy buena performance y no resulta claro que uno sea superior al otro. De todas formas, cualquiera sea el mecanismo elegido, resulta clave evitar la clusterización temprana del sistema si se desea mantener una buena performance.

De aquí en adelante, denominaremos a la política obtenida luego de agregar pushBN y el control de repeticiones a fBN como **pBN**. En la siguiente sección, se profundiza el análisis del comportamiento en sistemas de distintos tamaños y topologías.

La implementación standard de BN mantiene una probabilidad baja de seleccionar un nodo aleatoriamente aún en la Etapa de Aplicación. Es decir, ignora la selección propuesta por el ranking confeccionado en base al historial de comunicaciones previas y elige un nodo cualquiera como destino de su mensaje. El objetivo de este mecanismo es poder permitir

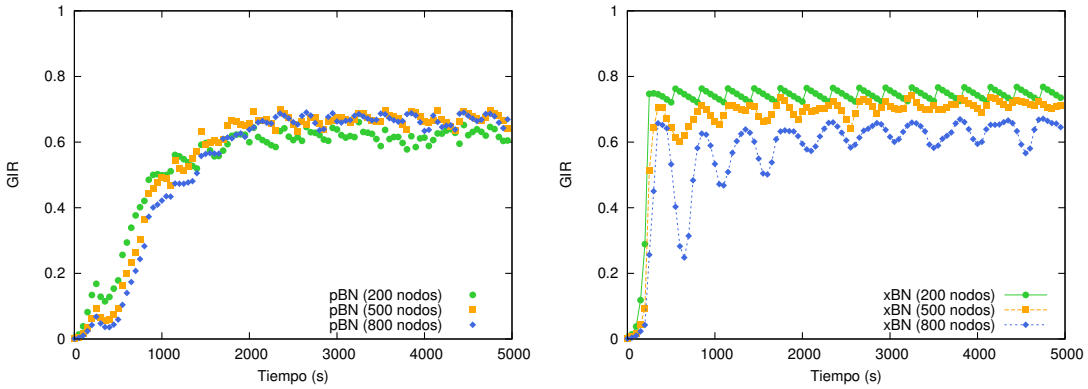


(a) Distintas longitudes de LP. (b) Distintas longitudes de LP + swap.

Fig. 4.8: La Figura (a) muestra la evolución del GIR en 200 nodos para distintas longitudes de LP (1, 5, 10 y 20). En todos los casos se observa que esta modificación no influye notoriamente en la performance alcanzada para la Etapa de Estabilidad, por lo que la Etapa de Crecimiento puede reducirse todo lo necesario para obtener buena performance lo más temprano posible. Los resultados para una solución más ajustada se presentan en la Figura (b). También corresponden al GIR para una grid de 200 nodos, pero a diferencia de los anteriores se agrega un control simple durante los `polls` de la Etapa de Aplicación en los que no se realiza el control, consistente en intercambiar el primer y segundo host mejor calificado por la función de scoring en caso de repetición para dos evíos consecutivos. De esta forma la performance no muestra casi ningún efecto para todas las longitudes de LP analizadas.

al nodo salir de una situación del estilo de mínimo local.

Si en lugar de realizar la selección aleatoria del host cada cierto tiempo se hace el envío a un nodo fijo cualquiera predeterminado (denominado x o *nodo fuente*), el mecanismo de LP se vuelve innecesario. Denominaremos a la inclusión de esta mejora, como **xBN**.



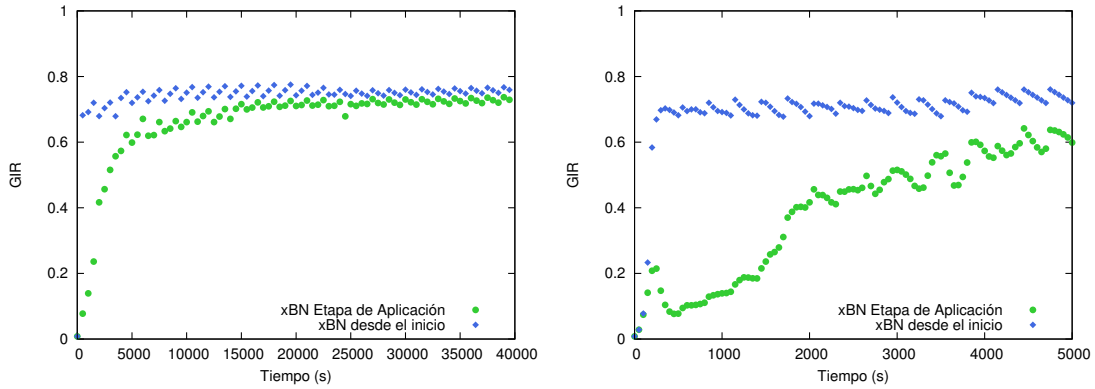
(a) GIR promedio pBN. (b) GIR promedio xBN.

Fig. 4.9: (a)Evolución del GIR para 200, 500 y 800 nodos de la política de distribución pBN con LP20sw, durante los primeros 20 `polls` (LP20) de la Etapa de Aprendizaje se realiza una verificación simple, realizando la solicitud al segundo host vecino mejor calificado en caso de repetición (swap). Luego de los primeros 20 `polls` se comienza con un control más estricto, no permitiendo más de tres repeticiones en los últimos 10 `polls`. (b)Evolución del GIR para 200, 500 y 800 nodos de la política de distribución xBN utilizando el *Host_3* como nodo fuente desde el inicio. En ambos casos la performance es muy buena y no se ve muy afectada por el incremento en el tamaño del sistema.

Los resultados en la Figura 4.9 corresponden a la evolución del GIR promedio con ventana de 10 puntos. Se comparan las dos últimas políticas introducidas como variación de fBN. En la Figura 4.9(a) se incluye a la política de distribución pBN con LP20sw y

en la 4.9(b) se encuentra xBN utilizando el nodo fuente desde el inicio (esto último se discute más adelante). Para los resultados de pBN durante los primeros 20 `polls` (LP20) de la Etapa de Aprendizaje se realiza una verificación simple, realizando la solicitud al segundo vecino mejor calificado en caso de repetición (swap). Luego de los primeros 20 `polls` se comienza con un control más estricto, no permitiendo más de tres repeticiones en los últimos 10 `polls`. En tanto que para los resultados de xBN, se utiliza el $Host\_3$ como nodo fuente. En ambos casos la performance es muy buena y no se ve muy afectada por el incremento en el tamaño del sistema, sin embargo pBN presenta una mejor capacidad de escalar que xBN.

Queda aún pendiente determinar si conviene que xBN esté activa desde el inicio o que lo haga a partir del comienzo de la Etapa de Aplicación. La Figura 4.10 muestra la evolución del GIR en un sistema de 100 nodos con topología exponencial en el que el mecanismo de selección aleatoria se ha reemplazado por el envío cada cierta cantidad de tiempo aleatorio a un nodo fijo predeterminado. La Figura 4.10(a) muestra que utilizar xBN desde el inicio o en la Etapa de Aplicación no muestra mayores diferencias una vez que el sistema ha evolucionado. Sin embargo, la Figura 4.10(b) revela que durante los primeros 5000s hay una notoria diferencia entre las dos opciones y muestra que utilizar xBN desde el inicio disminuye notablemente la duración de la Etapa de Aprendizaje.



(a) 40000s. (b) 5000s. Detalle de la Etapa de Crecimiento.

Fig. 4.10: (a) Evolución del GIR en 100 nodos durante 40000s al utilizar fBN con pushBN y *solicitudes a intervalos aleatorios de tiempo a un nodo fuente x*, xBN desde el inicio, y solo a partir de la Etapa de Aplicación. Promedio con ventana de 10 graficando cada 100 puntos. (b) Detalle de la Etapa de Crecimiento para cada caso. Se alcanza la performance de la Etapa de Estabilidad inmediatamente al utilizar xBN desde el inicio, mientras que pasados los primeros 5000s el sistema todavía se encuentra en la Etapa de Crecimiento si solo se utiliza xBN a partir de la Etapa de Aprendizaje. Promedio con ventana de 10 graficando cada 10 puntos. En ambos casos se utilizó como nodo fuente el Host_3.

La aplicación de estas dos mejoras, obtenidas a partir de la inclusión de pushBN a fBN, puede considerarse como la definición de sendas políticas de distribución de la información sobre recursos basadas en BN, que llamamos pBN y xBN, y las analizaremos según su escalabilidad en las distintas topologías en la siguiente sección.

4.4. Escalabilidad y otras topologías

4.4.1. pBN

Esta política de distribución de la información sobre recursos es completamente distribuida, al igual que Best Neighbor. Durante la etapa de Aprendizaje utiliza el Merge de Listas para obtener rápidamente información sobre toda la red. Una vez terminada esta etapa, los nodos dejan de realizar solicitudes aleatorias a otros nodos en el sistema y empiezan a seleccionar el nodo al que se realizará el siguiente poll utilizando la función de scoring según se definió para fBN en la Sección 4.2 (página 34). Para realizar los `push` durante la Etapa de Aplicación, se utiliza el pushBN, descripto en la Sección 4.3 (página 39). Se utilizanrá para evitar el particionado temprano del sistema el control de repeticiones de destino en `polls` consecutivos.

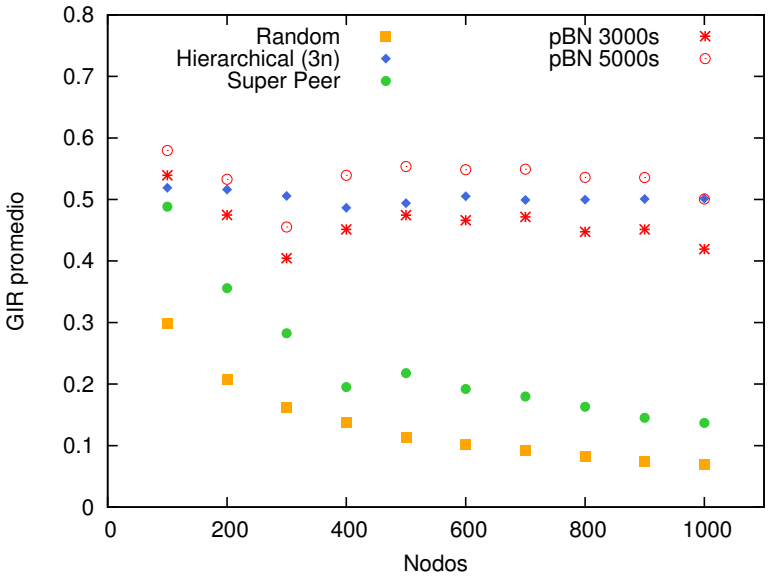
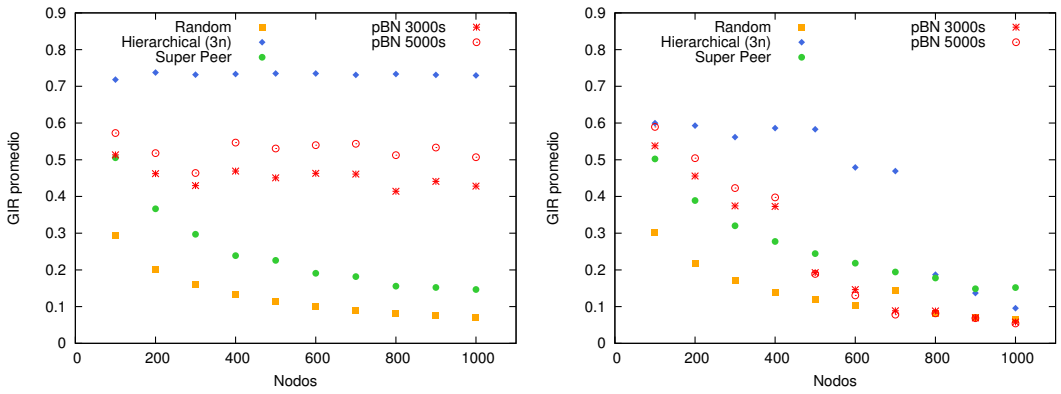


Fig. 4.11: GIR promedio en 3000s, para las políticas de distribución Jerárquica, Super Peer (5-SP), Random (1-Random) y pBN (3000s y 5000s, LP20sw), para redes de topología Exponencial con distintas cantidades de nodos.

Los resultados de la Figura 4.11 fueron obtenidos para para sistemas de topología exponencial de distintos tamaños con pBN acortando la Etapa de Crecimiento a 20 `polls` y utilizando durante ese periodo un control simple consistente en intercambiar el primer y segundo mejores vecinos en caso de repetición. En el caso de la Figura 4.11 el GIR promedio de 3000s para todas las políticas y 3000s y 5000s para pBN resulta igual o mejor que la obtenida al utilizar la política Jerárquica, salvo para el caso de 300 nodos, sin embargo, aún los resultados son mucho mejores que los obtenidos hasta el momento.

El mismo criterio se utilizó para los resultados de la Figura 4.12 pero para otras topologías: 4.12(a) Clique y 4.12(b) Anillo. Para la topología de Clique se obtuvieron valores de GIR promedio muy semejantes a los obtenidos para la topología Exponencial, superiores a 0,5 para un promedio de 5000s en todos los tamaños de sistema, salvo para 300 nodos. A diferencia de lo que ocurre en la topología Exponencial, en la topología Clique es indistinto qué host se utilice para generar la jerarquía en la política Jerárquica, como se explicó en la Sección 3.1, con lo que esta política obtiene su mejor performance, con promedios alrededor de 0,7, muy por encima incluso de pBN. Para la topología Anillo,

vemos para todas las políticas los mismos resultados ya explicados al presentar los resultados de fBN en la Sección 4.2, en cuanto a pBN vemos que la performance se deteriora rápidamente al aumentar la cantidad de nodos, incluso para redes de menor tamaño que para las que Jerárquica presenta muy mala performance.



(a) Clique (b) Anillo

Fig. 4.12: GIR promedio en 3000s, para las políticas de distribución Jerárquica, Super Peer (5-SP), Random (1-Random) y pBN (3000s y 5000s, LP20sw), para sistemas con distintas cantidades de nodos.(a) Topología Clique. (b) Topología Anillo

Al mejorar la conectividad del sistema, la performance de pBN no se ve afectada y resulta independiente de la cantidad de nodos, mientras que minimizar la conectividad redunda en una notable pérdida de performance a partir de sistemas de tamaño superior a 400 hosts.

4.4.2. xBN

xBN, es una política de distribución de la información sobre recursos completamente distribuida basada en BN que, del mismo modo que pBN, utiliza el Merge de Listas y la función de scoring propuesta en la Sección 4.2. Al utilizar BN, durante la Etapa de Aplicación todos los nodos realizan solicitudes, con baja probabilidad, a algún nodo elegido al azar sin tener en cuenta el resultado del scoring, al utilizar xBN estas solicitudes se realizan a un nodo x determinado (*nodo fuente*). Para realizar los `push` durante la Etapa de Aplicación, se utiliza el pushBN, descripto en la Sección 4.3, aunque en este caso, a diferencia de pBN, no se realiza ningún tipo de control.

Los resultados presentados en la Figura 4.13 corresponden al GIR promedio para xBN (desde el inicio, para no prolongar innecesariamente las simulaciones). La performance de esta política resulta aún mejor en promedio que la de pBN, sobre todo para sistemas de hasta 500 nodos, siendo muy leve la influencia del tiempo, es decir la diferencia entre los resultados para 3000s y 5000s, gracias a la reducción de la Etapa de Crecimiento dentro de la Etapa de Aplicación.

Esta política se ve más afectada que pBN por el incremento en el tamaño del sistema, aunque en todos los casos observados se mantiene con un promedio superior o igual a los promedios obtenidos por Jerárquica.

Se repite el experimento para sistemas con topologías de Clique y Anillo. Los resultados obtenidos son, respectivamente, los presentados en las Figura 4.14(a) y 4.14(b)

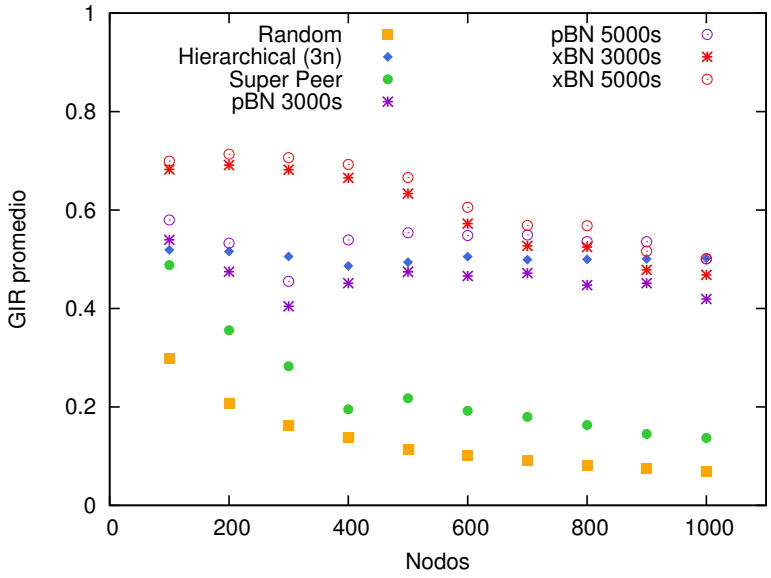
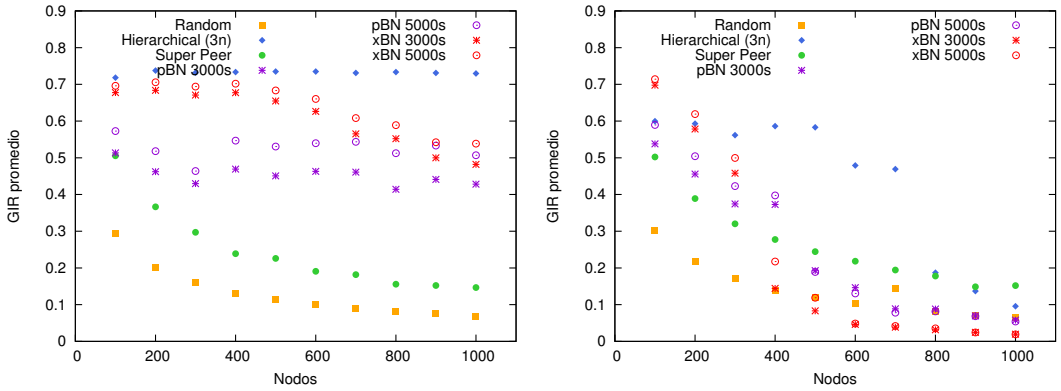


Fig. 4.13: GIR promedio en 3000s, para las políticas de distribución Jerárquica, Super Peer (5-SP), Random (1-Random), pBN (3000s y 5000s, LP20sw) y xBN desde el inicio (3000s y 5000s), para redes de topología Exponencial con distintas cantidades de nodos.



(a) Clique (b) Anillo

Fig. 4.14: GIR promedio en 3000s, para las políticas de distribución Jerárquica, Super Peer (5-SP), Random (1-Random), pBN (3000s y 5000s, LP20sw) y xBN desde el inicio (3000s y 5000s), para sistemas con distintas cantidades de nodos.(a) Topología Clique. (b) Topología Anillo

Los resultados para Jerárquica, Random y 5-SP en topologías Clique y Anillo ya fueron comentados anteriormente, al igual que los de pBN en la Sección 4.4.2. Como sucede para los resultados obtenidos para la topología Exponencial, en la topología de Clique el uso de xBN mantiene la performance del sistema por encima de la obtenida por las demás políticas distribuidas, incluso mejor que la obtenida por pBN, aunque en este caso los resultados son superados, en promedio, por los de Jerárquica. Para sistemas de hasta 500 nodos no se encuentra una diferencia importante entre la performance de Jerárquica y la xBN. Estos resultados son los presentados en la Figura 4.14(a), en los que también podemos distinguir mejor influencia del tiempo transcurrido, está diferencia se hace mayor a medida que aumenta el tamaño del sistema.

La performance xBN en la topología Anillo, según los resultados de la Figura 4.14(b), es

peor en promedio que los de todas las demás políticas, incluso peores que los de Random, cuando el tamaño del sistema supera los 400 nodos. Pero para sistemas de hasta 200 nodos la performance de xBN en esta topología es superior o igual, en promedio, que la que alcanza pBN, mejor incluso que la que alcanza Jerárquica.

Como en el caso de pBN, al mejorar la conectividad del sistema, la performance de pBN no se ve afectada independientemente del tamaño del mismo, aunque se distingue levemente la influencia del tiempo transcurrido. Minimizar la conectividad redunda en una notable pérdida de performance al aumentar el tamaño del sistema.

5. CONCLUSIONES Y TRABAJO FUTURO

5.1. Conclusiones

La computación de alto rendimiento (HPC, High Performance Computing) es un término que engloba un conjunto de procedimientos y estrategias computacionales para resolver *eficientemente* problemas complejos que demandan un gran poder de cómputo. Una de las tecnologías que ha emergido y se ha consolidado en los últimos años es *Grid Computing*, que permite el acceso a diversos recursos distribuidos geográficamente de manera remota.

Debido a su naturaleza altamente distribuida, uno de los puntos más importantes a considerar dentro de las Grids es saber dónde están los recursos y su disponibilidad de manera de poder coordinar su utilización, siendo ésta una problemática común a Cloud Computing y a sistemas distribuidos en general. Cómo se obtiene y distribuye la información sobre estos recursos es lo que se denomina *política de distribución de la información de recursos*.

Una clasificación posible de las políticas de distribución de la información es considerarlas centralizadas o descentralizadas, y a estas últimas a su vez como estructuradas o no-estructuradas. En esta tesis se estudiaron algunos aspectos de cuatro políticas de distribución de la información que resultan representativas dentro de esta clasificación: Jerárquica, Super Peer, Random y Best Neighbor. Las primeras fueron estudiadas en sus características principales en el Capítulo 3 y se dedicó a Best Neighbor un estudio más profundo en el Capítulo 4.

Sobre los análisis realizados y los resultados preliminares obtenidos para Jerárquica en la Sección 3.1 podemos decir que la selección del nodo que actuará como root de la jerarquía no tiene un efecto relevante en topologías Clique o Anillo, mientras que la selección del root para una grid de topología Exponencial puede impactar fuertemente en la performance. Observamos que la performance se maximiza cuando la cantidad de nodos en el segundo nivel de la jerarquía generada aumenta aunque para la mayoría de los casos se obtienen jerarquías con pocos nodos en el segundo nivel. Concluímos que seleccionar al centerum como root o nodo generador permite obtener un overlay representativo con el que poder comparar los resultados de las políticas distribuidas.

En cuanto a la política Super Peer encontramos que la pérdida de performance que evidencia esta política de distribución de la información sobre recursos al aumentar el particionado del sistema (aumentar la cantidad de sps) puede compensarse, en cierta medida, disminuyendo el período de refresco entre los super peers o incluso introduciendo un cambio a esta política para que el intercambio entre sps se realice a todos los sps y no a uno elegido aleatoriamente, como se explica en la Sección 3.2.

Más aún, al analizar Random, en la Sección 3.3, observamos que frente a un mismo tiempo de expiración de la información, más importante que el tiempo entre mensajes de refresco es la cantidad de nodos que lo reciben. Incrementar la cantidad de mensajes, frente a un mismo período de intercambio, siempre produce una mejora, aunque en este caso debería tenerse en cuenta cuál es el límite posible en el incremento de mensajes para no saturar la red. En redes de gran tamaño no es posible afirmar que la performance de Super Peer es mejor que la de Random, lo que se evidencia cuando se considera la

implemenrtación estándar de Super Peer y N-Random, variación de la política Random que realiza N solicitudes aleatorias simultáneas.

Al analizar la implementación de la política Best Neighbor fue posible reconocer tres aspectos fundamentales de la misma: la cantidad de nodos que conoce cada nodo en relación a la cantidad de nodos totales en la red, la manera en que un nodo obtiene información de los demás nodos y actualiza la información que posee sobre los nodos que considera vecinos y las características de la función que se utiliza para elegir el mejor vecino (función de scoring).

El estudio realizado sobre cada uno de estos tres aspectos de Best Neighbor, en el Capítulo 4, permitió encontrar soluciones a los problemas que se fueron identificando, cuyas implementaciones consiguen que la performance de políticas completamente distribuidas (basadas en BN) se comporte de manera semejante a la de la política Jerárquica y no a la de la política Random como se observó inicialmente.

En este proceso observamos que reducir el aporte de selecciones aleatorias no afecta de manera positiva en la performance. Introducimos el mecanismo de Merge de Listas en la Sección 4.1 como medio para reducir la duración de la Etapa de Aprendizaje, logrando al utilizarlo que todos los nodos en el sistema alcancen un conocimiento completo de los recursos en la red en menos tiempo, aunque forzar la reducción de la Etapa de Aprendizaje en BN no afecta de manera positiva en la performance de manera directa.

Respecto de la función de scoring, analizamos en la Sección 4.2 sus características y comparamos el comportamiento de las magnitudes que observa, lo que condujo a la normalización de la misma. Presentamos una forma de normalizar la función de scoring que permite que su evaluación no dependa del tamaõ del sistema. Observamos que no es trivial la elección de las magnitudes que intervendrán en la función y proponemos la inclusión de una nueva magnitud a tener en cuenta en la fórmula: los recursos propios que el host puede proveer. Al incluir el mecanismo de Merge de Listas y tener en cuenta los recursos propios de los host en la función de scoring se obtiene una mejora significativa de la performance de BN, cuya implementación llamamos fBN. Gracias a la normalización de la función de scoring, esta mejora no se ve afectada al aumentar el tamaño del sistema más que otras políticas distribuidas, ni tampoco al modificar la conectividad, performando del mismo modo en topologías Exponenciales, Cliques y Anillos.

En la Sección 4.3 se experimenta con diversos mecanismos para la publicación de recursos y se estudia cómo influyen en la performance del sistema. Reemplazar la selección aleatoria del nodo destino para efectuar una publicación de recursos eligiendo al mejor vecino como destino de la misma provoca clusterización temprana del sistema si no se controla la repetición de destinos, lo que afecta de manera negativa en la performance. Sin embargo, si al utilizar esta estrategia, que llamamos push BN, se realiza algún tipo de control para evitar la repetición en la selección de destinos se consigue muy buena performance. La elección de la estrategia para realizar las publicaciones de recursos en políticas basadas en BN no es trivial.

Presentamos pBN y xBN como políticas de distribución de la información sobre recursos completamente distribuidas. pBN, basada en fBN, utiliza el pushBN con un mecanismo de control de repeticiones que permite una mejora en la performance notable, similar a la obtenida por la política Jerárquica, al utilizarse en sistemas de entre 100 y 1000 nodos de topología Exponencial. pBN no se ve afectada por el aumento de la conectividad sin importar el tamaño del sistema, pero si por la disminución de la misma en sistemas de más de 400 nodos. xBN, también basada en fBN, que utiliza el pushBN pero sin control

de repeticiones y reemplaza las solicitudes esporádicas a destinos aleatorios por polls a un nodo fuente determinado, obtiene la mejor performance de entre las de las políticas estudiadas. Para sistemas de mediana escala la performance es superior o equivalente a la observada para la política Jerárquica al utilizarse en topologías Clique o Exponencial, mientras que minimizar la conectividad redunda en una notable pérdida de performance al aumentar el tamaño del sistema.

Como conclusión principal de esta tesis afirmamos que es posible obtener buena performance de manera escalable en los escenarios estudiados utilizando las políticas completamente distribuidas basadas en fBN, pBN y xBN.

5.2. Resultados ya publicados

"`Using distributed local information to improve global performance in Grids`".

P. Verghelet, D. Fernández Slezak, P. Turjanski, E. Mocskos.

CLEI ELECTRONIC JOURNAL, volume 15, number 3, Diciembre 2011.

"`Generating hierarchical overlays for exponential topologies`"

Paula Verghelet, Esteban Mocskos

High Performance Computing Day durante la 42º JAIIO (Poster premiado HPCLATAM 2013) - HPC 2013 - Jornadas Argentinas de Informática - FAMAF - Septiembre 2013, Córdoba.

5.3. Trabajo futuro

Durante el desarrollo de este trabajo surgieron algunos temas que pueden resultar de interés para trabajos futuros. Algunos de esos temas fueron delineados y necesitan una mayor profundización, algunos se desprenden de las características de la metodología utilizada y otros surgen como inquietudes propias de los resultados obtenidos. A continuación se listan algunos de ellos:

- Validación de los resultados en escenarios reales y/o testbeds para las nuevas propuestas de políticas distribuidas basadas en BN: fBN, pBN y xBN.
- Estudio de los beneficios de la utilización de fBN, pBN y xBN, para distintos superschedulers grids, hypervisors clouds y schedulers en general.
- Análisis de las relaciones entre las métricas utilizadas en este trabajo con otras métricas de uso común.
- Generación y análisis de resultados a partir de simulaciones con escenarios dinámicos.
- Estudio teórico sobre la relación entre la conectividad de la red subyacente y la performance del sistema.
- Análisis de la influencia del incremento en la cantidad de mensajes de intercambio, influencia del alcance de la diseminación en un paso de tiempo y su relación con los resultados sobre el índice R.

- Generación y análisis de resultados sobre la función de scoring a partir de utilizar distintas distribuciones de recursos en los nodos, su relación con distintos tipos de escenario y la influencia de las distintas magnitudes tenidas en cuenta en la función.
- Estudio de la utilización de Scoring Dinámico, utilización de más de una función según condiciones generales del sistema en determinado momento.
- Estudio de la escalabilidad del mecanismo de Merge de Listas.
- Otras aplicaciones del Merge de Listas (ej. recopilación de estado de sensores en WSN).
- Implementación de los algoritmos para MBTs u otros algoritmos para generación de jerarquías óptimas. Generación y análisis de resultados sobre su utilización en topologías Exponenciales o Power-Law.
- Estudio detallado y descriptivo sobre algoritmos para generar jerarquías.
- Estudio de la influencia de la política de distribución utilizada entre los super peers en la performance de la política de distribución Super Peer.
- Estudio de la influencia de la política de distribución utilizada entre los peers en la performance de la política de distribución Super Peer, al minimizar la cantidad de super peers en el sistema.
- Estudio de la la influencia de la utilización de topologías híbridas (ej. clique entre los super peers, power-law entre los peers) en la performance de la política de distribución Super Peer.
- Estudio del impacto del mecanismo utilizado para la selección de los super peers en la performance de la política Super Peer.
- Estudio del impacto de reducir el tiempo de refresco por medio del envío simultáneo de mensajes en la performance de las políticas de distribución de la información sobre recursos.
- Estudio en escenarios reales y/o testbeds de las nuevas propuestas para Random, Super Peer y Jerárquica.

Bibliografía

[1] H. Sutter, The free lunch is over: A fundamental turn toward concurrency in software, Dr. Dobb's journal 30 (3) (2005) 202–210.

[2] A. D. Stefano, G. Morana, D. Zito, A P2P strategy for QoS discovery and SLA negotiation in Grid environment, Future Generation Computer Systems 25 (8) (2009) 862–875.
URL http://www.sciencedirect.com/science/article/pii/S0167739X09000338

[3] R. Ranjan, A. Harwood, R. Buyya, Peer-to-peer-based resource discovery in global grids: a tutorial, IEEE Communications Surveys and Tutorials 10 (2) (2008) 6–33.

[4] N. J. Navimipour, A. M. Rahmani, A. H. Navin, M. Hosseinzadeh, Resource discovery mechanisms in grid systems: A survey, Journal of Network and Computer Applications (0) (2013) –.

[5] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, International Journal of High Performance Computing Applications 15 (3) (2001) 200–222.
URL http://portal.acm.org/citation.cfm?id=1080667

[6] I. Foster, C. Kesselman, J. Nick, T. S., The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Tech. rep., Open Grid Service Infrastructure WG, Global Grid Forum (2002).
URL http://www.globus.org/research/papers/ogsa.pdf

[7] A. Iamnitchi, I. Foster, D. Nurmi, A peer-to-peer approach to resource discovery in grid environments, in: Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 (HPDC' 02), IEEE, Edinbourgh, UK, 2002, p. 419.

[8] G. Pipan, Use of the TRIPOD overlay network for resource discovery, Future Generation Computer Systems 26 (8).
URL http://www.sciencedirect.com/science/article/pii/S0167739X1000018X

[9] R. Ranjan, L. Zhao, Peer-to-peer service provisioning in cloud computing environments, Journal of Supercomputing 65 (1) (2013) 154–184.

[10] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud Computing and Grid Computing 360-Degree Compared, in: Grid Computing Environments Workshop, 2008. GCE '08, 2008, pp. 1–10.

[11] D. Ergu, G. Kou, Y. Peng, Y. Shi, Y. Shi, The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment, Journal of Supercomputing 64 (3) (2013) 835–848.
URL http://dx.doi.org/10.1007/s11227-011-0625-1

[12] I. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems, in: H. Jin, D. Reed, W. Jiang (Eds.), IFIP International Conference on Network and Parallel Computing 2005, Vol. 3779 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2005, pp. 2–13.

[13] Ganglia: open source monitoring system, last visited on 02/11/2011.
URL http://ganglia.sourceforge.net

[14] E. E. Mocskos, P. Yabo, P. G. Turjanski, D. Fernandez Slezak, Grid Matrix: a Grid Simulation Tool to Focus on the Propagation of Resource and Monitoring Information, Simulation: Transactions of the Society for Modeling and Simulation International 88 (10) (2012) 1233–1246.

[15] C. Mastroianni, D. Talia, O. Verta, A super-peer model for resource discovery services in large-scale Grids, Future Generation Computer Systems 21 (8) (2005) 1235–1248.
URL http://www.sciencedirect.com/science/article/pii/S0167739X05000701

[16] P. Trunfio, D. Talia, C. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, S. Haridi, Peer-to-Peer resource discovery in Grids: Models and systems, Future Generation Computer Systems 23 (7) (2007) 864–878.

[17] B. Yang, H. Garcia-Molina, Designing a Super-Peer Network, in: Proceedings of 19th International Conference on Data Engineering (ICDE'03), IEEE Computer Society, Los Alamitos, CA, USA, 2003, p. 49.

[18] P. Yabo, Grid Matrix: una extensión a un simulador de Grid para enfocar en la propagación de información de recursos y monitoreo, Master's thesis, Facultad de Ciencias Exactas y Naturales, UBA (2008).

[19] D. G. Márquez, E. E. Mocskos, D. F. Slezak, P. G. Turjanski, Simulation of Resource Monitoring and Discovery in Grids., in: Proceedings of HPC 2010 High-Performance Computing Symposium, 2010, pp. 3258–3270.
URL http://www.39jaiio.org.ar/node/121

[20] D. G. Márquez, D. F. Slezak, P. G. Turjanski, E. E. Mocskos, Gaining insight in the analysis of performance for Resource Monitoring and Discovery in Grids, in: Proceedings of HPC 2011 High-Performance Computing Symposium, 2011.
URL http://www.40jaiio.org.ar/sites/default/files/T2011/HPC/1227.pdf

[21] B. Quétier, F. Cappello, A survey of Grid research tools: simulators, emulators and real life platforms, in: 17th IMACS World Congress (IMACS 2005), Paris, France, 2005.

[22] H. Casanova, A. Legrand, M. Quinson, SimGrid: A Generic Framework for Large-Scale Distributed Experiments, in: 10th IEEE International Conference on Computer Modeling and Simulation, IEEE Computer Society, Los Alamitos, CA, USA, 2008, pp. 126–131.

[23] H. Casanova, L. Marchal, A Network Model for Simulation of Grid Application, Rapport de recherche RR-4596, INRIA (2002).
URL http://hal.inria.fr/inria-00071989

[24] R. Albert, H. Jeong, A.-L. Barabási, Internet: Diameter of the World-Wide Web, Nature 401 (1999) 130–131.
URL http://adsabs.harvard.edu/abs/1999Natur.401..130A

[25] R. Albert, A.-L. Barabási, Statistical mechanics of complex networks, Reviews of Modern Physics 74 (2002) 47–97.
URL http://link.aps.org/doi/10.1103/RevModPhys.74.47

[26] A.-L. Barabási, R. Albert, H. Jeong, Mean-field theory for scale-free random networks, Physica A: Statistical Mechanics and its Applications 272 (1-2) (1999) 173–187.
URL http://www.sciencedirect.com/science/article/pii/S0378437199002915

[27] M. E. J. Newman, S. H. Strogatz, D. J. Watts, Random graphs with arbitrary degree distributions and their applications, Physical Review E 64.
URL http://link.aps.org/doi/10.1103/PhysRevE.64.026118

[28] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.

[29] A. Proskurowski, Minimum broadcast trees, Computers, IEEE Transactions on 100 (5) (1981) 363–366.

[30] A. M. Farley, A. Proskurowski, Broadcasting in trees with multiple originators, SIAM Journal on Algebraic Discrete Methods 2 (4) (1981) 381–386.

[31] A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs.

[32] Y.-H. Lin, W.-C. Chung, K.-C. Lai, K.-C. Li, Y.-C. Chung, SARIDS: A Self-Adaptive Resource Index and Discovery System, in: Proceedings of 10th Symposium on Parallel Architectures, Algorithms, and Networks, International, IEEE Computer Society, Los Alamitos, CA, USA, 2009, pp. 521–526.

Apéndice

A. DETALLES DE LA SIMULACIÓN

Como se mencionó en la Sección 2.2 es necesaria la generación de dos archivos para cada simulación: `deploy` y `platform`. En la Sección A.1 se describen los valores elegidos para los parámetros que conciernen a la infraestructura de red utilizada, es decir, los que se utilizaron para generar los `platform`s, y en la Sección A.2 se encuentra la descripción de los valores elegidos para la generación de los deploys utilizados en las simulaciones y una breve justificación de los mismos.

A.1. Valores utilizados para la generación de los platforms de las grids

En esta tesis se realizaron simulaciones para grids de topologías Exponencial (con exponente 2,5), Clique y Anillo de entre 100 y 1000 nodos. Gridmatrix permite la generación de platforms para estas y otras topologías definiendo algunos de sus parámetros, entre ellos el poder de cómputo en los nodos, el bandwith y la latencia en los enlaces, y el tipo de ruteo.

Para todos los escenarios se utilizaron los siguientes parámetros:

Cantidad de nodos: 100 a 1000.

Cantidad de routers: 50 a 500 (50 % de la cantidad de nodos).

Poder de cómputo por nodo: 10 $GFLOPS$.

Bandwith en cada enlace: 10 $Gbps$.

Latencia en cada enlace: $10^{-5}s$.

Algoritmo de ruteo entre hosts: Dijkstra.

Con esta configuración el uso de los mecanismos de control de congestión que provee TCP puede ser perjudicial, como se explica, por ejemplo, en el RFC 1323 [1] y en el trabajo de Hiroyuki Kamezawa et al.[2]. El modelo para la infraestructura de red es manejado por SimGrid y en este caso se trabajó con la configuración por defecto, la que utiliza el modelo CM02, lo que implica que se está utilizando MaxMin fairness y que el control de congestión es AIMD (Additive Increase Multiplicative Decrease). Queda fuera del alcance de este trabajo analizar los efectos de la selección del algoritmo para TCP más adecuado para la infraestructura utilizada en las simulaciones, aunque se tendrá en cuenta cuando se realicen pruebas en escenarios reales.

[1] TCP Extensions for High Performance http://tools.ietf.org/html/rfc1323 introduccion

[2] KAMEZAWA, Hiroyuki, et al. Inter-layer coordination for parallel TCP streams on Long Fat pipe Networks. En Proceedings of the 2004 ACM/IEEE conference on Supercomputing. IEEE Computer Society, 2004. p. 24.

A.2. Valores utilizados para la generación de los deploys de las políticas

Al generar los archivos de deploy para cada platform es necesario definir la duración de la simulación, el tiempo de refresco (intervalos de push y poll), el tiempo de expiración, la cantidad de recursos que provee cada host, qué nodos funcionarán como Super Peers para la política Super-Peer y la longitud del intervalo de intercambio entre los mismos, la cantidad de niveles de la jerarquía para la política Jerárquica, cuál host será el root y cómo se distribuyen los demás nodos definidos en el platform en los distintos niveles de la misma.

En *cuanto a la duración de las simulaciones*, se prolongan hasta que los resultados no evidencian diferencias perceptibles con simulaciones de mayor duración. La duración de las simulaciones para todos los resultados de Jerárquica, Super-Peer y Random es de $3000s$, mientras que para los resultados de BN, fBN, pBN y xBN se prolongaron hasta $5000s$ en general, salvo en los casos en que fue necesario más tiempo para evidenciar el fenómeno que se estaba estudiando (por ejemplo para los resultados de la Figura 4.10(a)).

Respecto de los *tiempos de refresco*, longitud del intervalo entre `polls` y entre `push`, y el *tiempo de expiración de la información* se tuvieron en cuenta los resultados obtenidos para el índice R por González Márquez et al. en [20], respecto de su relación con los valores de GIR. En consecuencia se eligieron valores para los intervalos de `push`, `poll` y `tiempo de expiración` de manera que la relación obtenida dé un valor para R menor que 10. Es necesario aclarar que si bien para todas las políticas se consideran eventos de `push` y `poll`, para algunas de las simulaciones de las distintas mejoras que se fueron realizando a Best Neighbor se eliminan los eventos de `push`, por ejemplo en las simulaciones correspondientes a la Sección 4.1 y la Sección 4.2 (fBN), dado que se buscaba minimizar los efectos de las intervenciones aleatorias que traían confusión al analizar resultados y los eventos de `push` se realizaban de ese modo. En general, al fijar el intervalo de `poll` en x y el intervalo de `push` en x, el tiempo de refresco de la información (utilizado para calcular el cociente R) es $x/2$ (debido a que hay dos eventos de comunicación en cada período, uno de `push` y otro de `poll`). Por este motivo, al eliminar los eventos de `push`, como se hace al generar los deploys para las simulaciones de fBN, fue necesario aumentar la frecuencia de los eventos de `poll` para que no se modifique el valor de R y que se mantengan las condiciones que permiten comparar los resultados entre un escenario y otro.

La *cantidad de recursos que provee cada host* es aleatoria con distribución *U(1,20)*, en todos los casos.

Cuando se generan los deploys para la política Super-Peer es necesario definir *qué nodos funcionarán como super peers* y *cuántos super peers habrá en el sistema*. Se realizaron simulaciones de esta política utilizando el 2 %, 5 %, 10 % y 20 % del total de los nodos como super peers. Para esto se particiona el sistema en tantas partes como sea necesario utilizando `metis` y se selecciona el nodo más céntrico en cada uno de estos clusters como super peer del mismo. La política de intercomunicación para los super peers es Random, en general, cada super peer elige otro super peer al que realizar la consulta de manera aleatoria, salvo en el caso de los resultados de la Figura 3.5(b) en la Sección 3.2 en el Capítulo 3, en los que se elige más de un destino por vez. Las consultas entre super peer se realizan cada cierto intervalo de tiempo fijo independiente del tiempo de refresco elegido para los peers (duración del intervalo de `push` y de `poll` definida en los deploys). Salvo para los resultados de la Sección 3.2 en el Capítulo Consideraciones Previas en el que se compararon resultados para distintas longitudes de dicho intervalo, se fijó el mismo

período de `poll` entre super peers en todas las simulaciones de esta política.

Para generar los deploys para las simulaciones de la política Jerárquica se fijó la *cantidad de niveles* en tres, salvo para los resultados de las figuras que se presentan en la Introducción donde se realizaron simulaciones también para jerarquías de dos niveles, es decir sistemas centralizados. Hasta la Sección 3.1 en la que se define el *centerum*, se utilizó como root de las jerarquías generadas el $Host\_0$, y luego, en los restantes resultados referidos a la política Jerárquica se utiliza como nodo raíz de la jerarquía al *centerum*. El *tiempo de refresco* es el mismo para todos los niveles, tanto para publicaciones como para solicitudes.

En los deploys utilizados para las simulaciones de la política Random se fijó la misma longitud para los intervalos entre polls y entre push, mientras que para los utilizados para N-Random se mantuvo fijo el tiempo de expiración (el mismo que para Random) y se ajustaron estos valores para que el R se mantuviera fijo, es decir, en los resultados de N-Random de la Figura 3.6(a) se envían la misma cantidad de mensajes por período de tiempo entre polls y pushs. Algo similar ocurre con los deploys utilizados para obtener los resultados de la Figura 3.6(b), para cada longitud del tiempo de refresco (entre 10 y 100) se realizaron simulaciones con distinta cantidad de solicitudes simultáneas.

Los siguientes son algunos de los valores numéricos utilizados para los parámetros al generar de los deploys como se mencionó previamente:

- R = 8
- Tiempo de expiración: 240
- Las longitudes de los intervalos entre consultas y publicaciones entre nodos utilizados para los casos generales son los indicados en la siguiente tabla:

Nombre de la política	Intervalo entre `polls`	Intervalo entre pushs	Intervalo entre `polls` SP
Best Neighbor	60	60	-
fBN	30	-	-
pBN	60	60	-
xBN	60	60	-
Random	60	60	-
Super Peer	60	60	40
Jerárquica	60	60	-

A.3. Otros datos de interés

Cada paso temporal en la simulación equivale a 1 segundo.

Otros valores que pueden resultar de interés corresponden directamente a las implementaciones de las políticas y no se incluyen en los deploys de los escenarios.

A.4. Scripts, platforms y deploys

El material con los scripts Python de cada una de las políticas de distribución de la información sobre recursos que fueron utilizados para las simulaciones con Gridmatrix, asi como los platforms y deploys generados con esta herramienta, a partir de los cuáles se obtuvieron los resultados de las figuras, pueden encontrarse en:

http://lsc.dc.uba.ar/hpc-grid/grid/grid-matrix-1