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

Paula Verghelet

Laboratorio de Sistemas Complejos, Departamento de Computación,
Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires
Buenos Aires (C1428EGA), Argentina.
pverghelet@dc.uba.ar

Resumen La computación de alto rendimiento (HPC, High Performance Computing) tiene como objetivo el estudio de los procedimientos y estrategias computacionales necesarios para resolver *eficientemente* problemas complejos que demandan gran poder de cómputo.
En sistemas distribuidos de gran escala como las Grids o las Federaciones de Clouds, resulta crucial saber dónde están los recursos y su disponibilidad para poder coordinar su utilización. 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 para políticas de distribución de la información es considerarlas centralizadas o descentralizadas. Estas últimas se dividen en estructuradas o no-estructuradas. Las políticas Jerárquica, Super-Peer, Random y Best-Neighbor resultan representativas dentro de esta clasificación.
Contar con una política de distribución que sea a la vez escalable, tolerante a fallas, que no demande mantenimiento en exceso ni consuma recursos de red y procesamiento del sistema en forma desmedida, resulta un desafío tecnológico remarcable.
El principal resultado de esta tesis son dos políticas basadas en Best Neighbor que evidenciaron muy buena performance siendo, a la vez, escalables, distribuidas y sin gran dependencia de administración manual.

Keywords: Cómputo Distribuido, Grid Computing, Información sobre Recursos, Políticas de Distribución de la Información, Best Neighbor.

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*. La computación de alto rendimiento (HPC, High Performance Computing) tiene por objetivo estudiar los procedimientos y estrategias computacionales necesarios 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 describen Ranjan et al. [3] y Navimipour et al. [4], pueden distinguirse distintas clases de Grids según su prioridad de uso: (i) *Cómputo*, (ii) *Datos*, (iii) *Inalámbrica* y (iv) *Multimedia*. En adelante nos referiremos a las Grids de Cómputo[1] simplemente como Grids.

Al utilizar Grid Computing es importante saber dónde están los recursos y su disponibilidad de manera de poder coordinar su utilización (task scheduling), siendo esta una problemática común a Cloud Computing y a sistemas distribuidos en general [5–7].

La optimización de la asignación de tareas y solicitudes de recursos de los usuarios requiere que la información sobre los mismos se mantenga tan actualizada como sea posible, tal como se menciona en Iamnitchi et al. [8] y en Pipan [9].

Cómo se obtiene y distribuye la información sobre estos recursos es lo que se denomina *política de distribución de la información de recursos*, lo que en el artículo de Iamnitchi, [8], se denomina *Resource Discovery Problem*. Una clasificación posible de las políticas de distribución de la información es considerarlas centralizadas o descentralizadas, y a éstas últimas a su vez, como estructuradas o no-estructuradas.

Las propuestas iniciales para los servicios de indexación y recolección de información de estado de los recursos incluían modelos centralizados y jerárquicos [3, 10]. Estos modelos tienen como principal problema la baja tolerancia a fallas (*single point of failure*), así como una probable congestión de la red.

Para sistemas de medianos a grandes, la dinámica de la información sobre recursos no puede ser capturada utilizando una jerarquía estática. El crecimiento en tamaño y recursos de los sistemas con el correr del tiempo trajo consigo la búsqueda de métodos más eficientes y robustos, despertándose un creciente interés en lo desarrollado en el paradigma P2P [2–4, 8, 11, 12].

A continuación se presenta una breve descripción de las políticas de distribución más usuales:

[1] Sistemas con gran capacidad de cómputo disponible para aplicaciones

- **Jerárquica (H):** Se establece con antelación una jerarquía entre los nodos. Los nodos intercambian información sólo con los del nivel inmediato inferior o inmediato superior de la jerarquía establecida, Figura 8(a). Esta política es la utilizada por MDS [10].
- **Random (R):** Cada nodo elige de manera aleatoria otro nodo de la red del cual obtener información, Figura 8(b). Suele utilizarse para comparar comportamiento de peor caso [10]. Completamente distribuida.
- **Super Peer (SP):** 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 [12], 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 8(c). Los nodos peers se comunican directamente con un único super-peer y a través de él con los demás nodos.
- **Best Neighbor (BN):** 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.), Figura 8(d). 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.

Del estudio de la performance obtenida por cada una de estas cuatro políticas, encontramos que Best Neighbor presenta algunos resultados inesperados, 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.

Los principales objetivos de este trabajo consisten en identificar las causas de la baja performance de la política Best Neighbor, estudiar la forma de mitigar los efectos negativos de estas causas y aplicar este conocimiento en mejorar la performance de esta política completamente distribuida.

2. Metodología

En esta Sección 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 previamente mencionadas: Jerárquica, Super Peer y Random.

Para el análisis y comparación de la performance de estas políticas de distribución se consideran los resultados 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, del siguiente conjunto de métricas presenta-

das en Mocskos et al. [10]. En el Apéndice B.1 se detalla la definición completa de las mismas, aquí las describiremos como:

LIR (Local Information Rate): 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. Para el host k, LIR_k es:

$$LIR_k = \frac{\sum_{h=1}^{N} f(age_h, expiration_h) \cdot resourceCount_h}{totalResourceCount}$$

GIR (Global Information Rate): Indica la cantidad de información con la que cuenta el sistema sobre el total de recursos disponibles. Se obtiene por medio del promedio de los LIR de cada nodo.

Utilizamos para las simulaciones `Gridmatrix` [10, 13], herramienta Open Source basada en `SimGrid` [14]. Este framework sobre el que se desarrolló, y que actualmente utiliza `Gridmatrix`, es un simulador para aplicaciones distribuidas en escenarios heterogéneos.

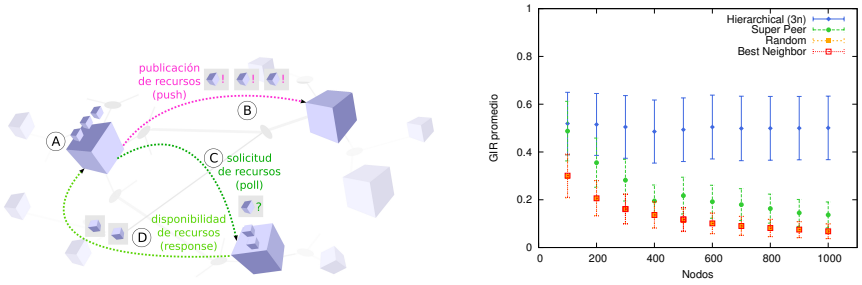
Se supondrá en este trabajo que la información que se obtiene en respuesta a una solicitud de recursos es siempre válida, es decir, si un nodo responde se supone que este puede satisfacer el pedido realizado en la solicitud. No se consideran requerimientos sobre QoS (Quality of Service), aunque en el caso de Best Neighbor se registra la cantidad de veces que no se obtuvo respuesta de un nodo o ésta fue tardía. La Figura 1(a) esquematizan los eventos de `push`[2] y `poll`[3] que utilizan los hosts para el intercambio de información en la aplicación.

Observar la *dinámica de la información sobre recursos* para cada una de las cuatro políticas, mediante el modelado y la simulación de este sistema complejo, requiere definir las variables del dominio que se tendrán en cuenta. Algunas de estas variables conciernen a la infraestructura mientras que otras son relativas a las políticas de distribución de la información sobre recursos. Fijamos algunos valores para las primeras, que se detallan en el Apéndice B, y profundizamos el estudio de algunas de las características que impactan sobre la performance para cada política que puede seguirse en el Apéndice C. A partir de dicho estudio se seleccionaron como referencia para este trabajo los valores del GIR obtenidos por Jerárquica (centerum), 5-SP y 1-Random.

Trabajamos con sistemas de entre 100 y 1000 nodos, sobre tres topologías distintas para la infraestructura de red: Clique, Anillo y Exponencial, en el Apéndice B se detallan las principales características de los escenarios.

Los resultados de la Figura 1(b) fueron obtenidos mediante el promedio de la evolución del GIR durante 3000s en sistemas entre 100 y 1000 nodos sobre una topología Exponencial. Puede observarse una coincidencia casi completa entre los resultados de Random y Best Neighbor.

[2] Publicación de recursos.
[3] Solicitud de recursos.



(a) Eventos de `push` y `poll` (b) Escalabilidad

Figura 1: (a) 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. (b) GIR promedio en 3000s de las políticas de distribución Jerárquica (Hierarchical), 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. El overlay jerárquico es de tres niveles para la política Jerárquica. En el caso de Super Peer se particiona de modo que el 5 % de los nodos actúen como super peer. En el Apéndice C se encuentra un análisis sobre algunas otras características de estas políticas.

3. Resultados

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

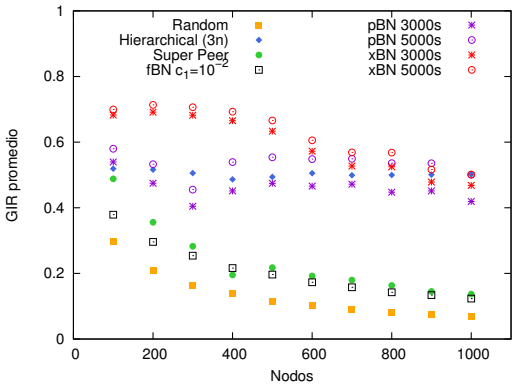
1. La cantidad de nodos que conoce cada nodo en relación a la cantidad de nodos totales en la red.
2. La función que se utiliza para elegir el mejor vecino (*función de scoring*).
3. La manera en que un nodo obtiene información de los demás nodos y actualiza la información que posee sobre los nodos que considera vecinos.

En las siguientes secciones se profundiza en el análisis de cada uno de ellos y se porponen alternativas a aquellas características que impactan de manera negativa en la performance.

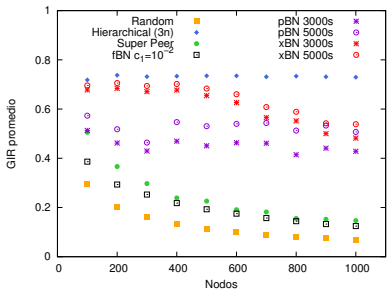
A partir de la implementación de estas alternativas obtuvimos *fBN*, *pBN* y *xBN*, políticas de distribución de la información sobre recursos completamente distribuidas basadas en Best Neighbor, que logran los resultados de la Figura 2 para sistemas de distinto tamaño en topologías Exponencial 2(a), Clique 2(b) y Anillo 2(c). Se pospone el análisis de los mismos para la Sección 3.3, sin embargo es posible reconocer una notable mejora en la performance al utilizar estas nuevas propuestas, llegando a superar los mejores resultados previos (Jerárquica).

3.1. Conociendo Vecinos

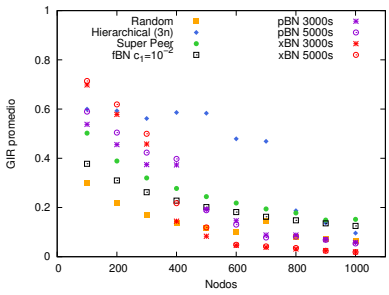
Siguiendo la política Best Neighbor, cada nodo en el sistema atraviesa dos etapas bien diferenciadas respecto de la manera en que son elegidos los hosts a los que solicitar recursos o información sobre recursos:



(a) Exponencial.



(b) Clique.



(c) Anillo.

Figura 2: 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.(b) Topología Clique, (b) Topología Exponencial, (c) Topología Anillo.

- **Etapa de Aprendizaje**: Los nodos se encuentran sin información sobre el sistema. El siguiente nodo se elige de manera aleatoria.
- **Etapa de Aplicación**: Se consulta al mejor vecino en el ranking generado por la *función de scoring*

Durante la Etapa de Aprendizaje el sistema funciona del mismo modo que si se utilizara la política Random. La duración de esta etapa en la que los nodos recopilan información sobre el sistema podría comprometer la performance si se extendiera demasiado. Es decir, el sistema se encontraría utilizando la política Random mientras se encuentre en esta etapa.

Al utilizar Best Neighbor 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:

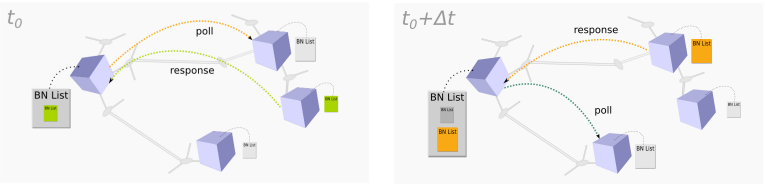
$$c_i = \frac{|\texttt{vecinos}(i)|}{\texttt{hostCount}}$$

donde `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.

Al estudiar la proporción de nodos que pasaban de una a otra etapa en el transcurso del tiempo observamos que la Etapa de Aprendizaje se prolongaba durante largo tiempo, como puede observarse, a modo de ejemplo, en los resultados para una sistema de 100 nodos de la figura 4(a)(BN).

Proponemos reducir la Etapa de Aprendizaje. Una posibilidad para acortar esta etapa es modificar el valor que regula el cambio de etapa, agregando una cota al incremento de c_i, forzando así el cambio de etapa para todos los nodos.

Otra estrategia alternativa para disminuir la duración de la Etapa de Aprendizaje consiste en realizar un *merge de listas*. En la Figura 3 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.

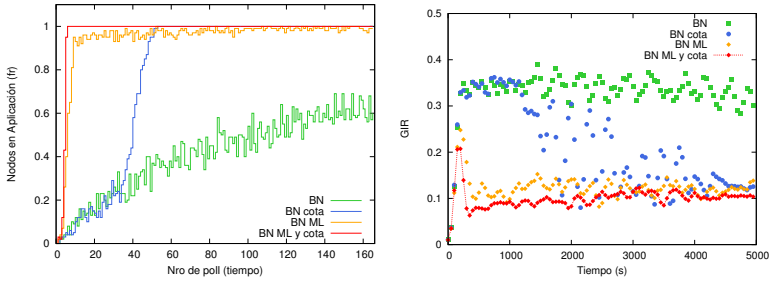
Figura 3: Merge de listas. Al recibir respuestas (`response`) a las solicitudes realizadas (`poll`) los hosts actualizan la información de sus listas de vecinos con la información de la lista de vecinos del host que respondió a la solicitud. Este mecanismo permite que todos los nodos alcancen un conocimiento completo de los recursos en la red en menor tiempo.

Si bien ambas estrategias reducen fuertemente la Etapa de Aprendizaje, como puede verse para la mayoría de los nodos en la figura 4(a)(BN cota, BN ML y BN ML y cota), esto no representa una mejora en la performance, si no que la afecta de manera negativa. En la figura 4(b) se incluyen los resultados de la evolución del GIR para el mismo sistema de 100 nodos al utilizar cada una.

3.2. 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.

Paula Verghelet



(a) Nodos en Etapa de Aplicación. (b) GIR (100 nodos).

Figura 4: 4(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(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.

La función de scoring utilizada durante la Etapa de Aplicación se encuentra definida del siguiente modo:

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

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. Los parámetros a, b y c se utilizan para ajustar el peso relativo de cada una de las magnitudes en la función.

En los escenarios estudiados en este trabajo, el valor de la latencia no resulta significativo si se lo compara con el total de recursos del sistema al suponer, por ejemplo, un recurso disponible por nodo. La importante 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[4].

Normalizar f_{scoring} permite independizarnos de las características del sistema. Denominaremos a f_{scoring} normalizada como f_{BN}, simplificando la notación:

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

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

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 menciona en el Apéndice B.2 (página).

[4] 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 $1000 \cdot x$.

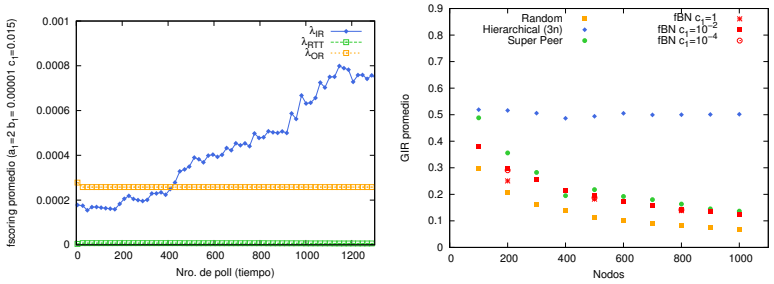
Otro punto a tener en cuenta es que, a pesar de tener un LIR alto, un nodo podría tener una baja cantidad de recursos propios que ofrecer o compartir, es decir, que si bien puede poseer información actualizada sobre sus vecinos, el nodo podría carecer de recursos propios que informar, con lo que se estaría 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. Con este agregado, la función de scoring queda hasta aquí:

$$f_{\text{BN}} = a_1 \cdot \lambda_{\texttt{IR}} + b_1 \cdot \lambda_{\texttt{RTT}} + c_1 \cdot \lambda_{\texttt{OR}} \tag{1}$$

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 $b_1 < 0$.

El análisis realizado para la normalización de f_{scoring} y la selección de los valores para las constantes puede seguirse en el Apéndice D, aquí sólo mencionaremos que cada uno de los términos de f_{BN} toma valores entre 0 y 1, y que los valores de las constantes se eligieron de modo que permitan comparar los términos. En este caso, utilizando $a_1 = 1$, $b_1 = 10^{-4}$ y $c_1 = 10^{-2}$ 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 5(a) se obtienen a partir de los promedios de cada término en un nodo determinado (se detalla el procedimiento en el Apéndice D).



(a) Valores de las constantes (b) Escalabilidad

Figura 5: $\lambda_{\texttt{OR}}$ 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. Se calculan para cada una de las magnitudes que intervienen en la función de scoring el promedio de todos los vecinos para los que se calcula la función en un nodo determinado y se observa su evolución en el tiempo. La Figura (a) muestra los promedios una vez agregada la constante. (b)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).

Llamamos *fBN* a esta alternativa para Best Neighbor que incluye el Merge de Listas descripto en la Sección 3.1 y la utilización de f_{BN}, la función de scoring definida en (**??**).

La Figura 5(b) 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 1(b).

3.3. pBN y xBN: Agregando el push a fBN

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. En el Apéndice E se incluyen los resultados obtenidos al utilizar distintas estrategias para realizar las publicaciones de recursos en políticas basadas en BN. Elegir la estrategia adecuada no resulta trivial.

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`). Denominamos a esta estrategia *push BN*.

Sin embargo, las simulaciones no evidenciaron una mejora en la performance al utilizar esta estrategia.

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, formándose 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 *clusterización temprana* del sistema, 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.

De aquí en adelante, denominaremos a la política obtenida luego de agregar *push BN con control de repeticiones* a fBN como *pBN*.

En la Figura 6(a) se observa el GIR en función del tiempo para distintas longitudes de LP en un sistema de 200 nodos. 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[5].

La Etapa de Crecimiento puede acortarse reduciendo el período de espera para comenzar el control de repeticiones. La Figura 6(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 6(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.

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[6].

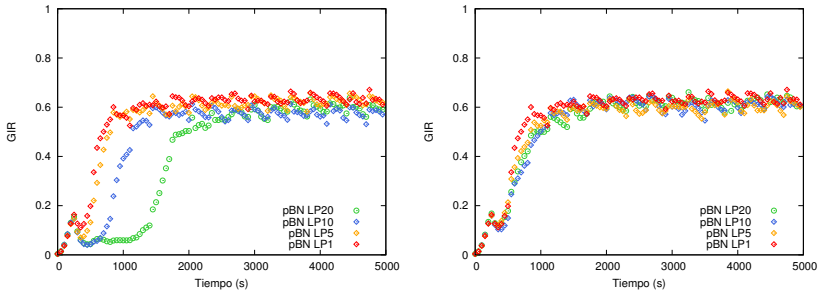
Si en lugar de realizar la selección aleatoria del host cada cierto tiempo se hace el envío a un nodo fijo cualquiera predeterminado (denominado x o *nodo fuente*), el mecanismo de LP se vuelve innecesario. Denominaremos a la inclusión de esta mejora, como *xBN*.

La Figura 7 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. En la Figura 7(a) observamos que utilizar xBN desde el inicio o en la Etapa de Aplicación no evidencia mayores diferencias una vez que el sistema ha evolucionado. Sin embargo, la Figura 7(b) revela que durante los primeros 5000s hay una notoria diferencia entre las dos opciones, utilizar xBN desde el inicio disminuye notablemente la duración de la Etapa de Aprendizaje.

La aplicación de estas dos mejoras, obtenidas a partir de la inclusión de *push BN* a fBN, puede considerarse como la definición de sendas políticas de

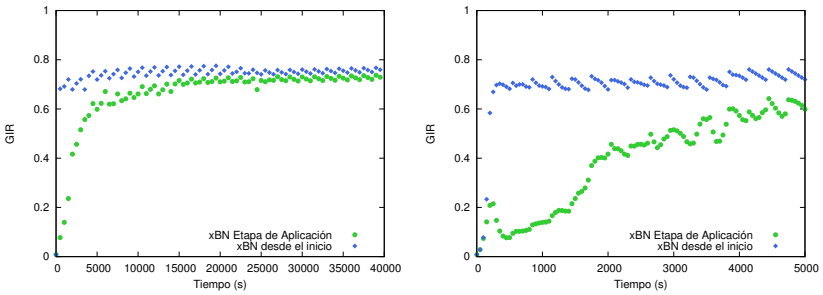
[5] Basándonos en las características de la información con la que cuenta el sistema sobre sí mismo, podemos distinguir dos etapas dentro de la Etapa de Aplicación al utilizar LP: i) Etapa de Crecimiento, y ii) Etapa de Estabilidad, esta segunda etapa inicia cuando la performance se estabiliza.

[6] El objetivo de este mecanismo es poder permitir al nodo salir de una situación del estilo de mínimo local.



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

Figura 6: 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.



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

Figura 7: (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.

distribución de la información sobre recursos basadas en BN, a las que llamamos pBN y xBN, y las analizaremos según su escalabilidad a continuación.

Los resultados de la Figura 2(a), que fueron mencionados en la introducción de esta Sección, fueron obtenidos 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, y xBN desde el inicio,

para no prolongar innecesariamente las simulaciones. Respecto de las condiciones para fBN, son las utilizadas en los resultados de la Sección 3.2. Se utilizó un overlay jerárquico de tres niveles generado a partir del *centerum*[7] para las simulaciones de la política Jerárquica y el 5 % de los nodos como super peers para las simulaciones de la política Super Peer. El mismo criterio se utilizó para los resultados de la Figura 2(b) y 2(c), pero para topologías Clique y Anillo respectivamente.

En el caso de la Figura 2(a) 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.

La performance de xBN 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.

Respecto de pBN, 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 explica en en el Apéndice C, con lo que esta política obtiene su mejor performance, con promedios alrededor de 0,7, muy por encima incluso de pBN. 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 de xBN.

En el caso de Anillo se observa que para sistemas de tamaño mayor a 500 nodos la performance de la política Jerárquica se deteriora a medida que crece el tamaño del sistema, llegando a ser semejante a la performance de Random en sistemas de 1000 nodos. El aumento en el tamaño del Anillo impacta notablemente, ya que el camino que deben recorrer los mensajes crece tanto como nodos se agreguen, lo que puede producir que la información contenida en los mensajes expire mucho antes de llegar a destino o que llegue muy desactualizada. Por otro lado, 5-SP se 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. En cuanto a pBN vemos que la performance se deteriora rápidamente al aumentar la cantidad de nodos. La performance de xBN es peor en promedio que la de todas las demás políticas, incluso peor que la de Random, cuando el tamaño del sistema

[7] Puede encontrarse la definición de *centerum* en el Apéndice C.

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.

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 al aumentar el tamaño del sistema.

4. Conclusiones y trabajos futuros

En este trabajo realizamos un estudio de la performance de Best Neighbor, política de distribución de la información sobre recursos utilizada en Grid Computing. A pesar de utilizar información histórica obtenida en base a consultas previas para guiar la política de distribución, BN presenta, de manera inesperada, resultados similares a los de la política Random.

Analizamos las posibles causas del mal desempeño de esta política, y propusimos alternativas a aquellas características de la misma que impactan de manera negativa en la performance.

Presentamos el mecanismo de *Merge de Listas* como una de las estrategias para reducir la Etapa de Aprendizaje en BN, aunque observamos en la Sección 3.1 que esta reducción no afecta de manera positiva en la performance. Conseguimos una mejora de la performance de *BN*, *escalable* e *independiente de la topología*, ajustando la *función de scoring normalizada*, política que denominamos *fBN* en la Sección 3.2. Agregamos el *push BN* a *fBN* y propusimos dos estrategias para evitar la *clusterización temprana* que mejoran notablemente la performance, obteniendo *pBN*. Reemplazamos las consultas aleatorias esporádicas por *consultas esporádicas a un nodo fijo x* al utilizar *fBN* junto con el *push BN*, y obtuvimos *xBN*.

El principal aporte de este trabajo se considera la presentación de pBN *y* xBN*, soluciones completamente distribuidas basadas en* fBN*, como políticas de distribución de la información sobre recursos con las que es posible obtener buena performance de manera escalable en los escenarios estudiados.*

Puede resultar de interés para trabajos futuros la validación de estos resultados en escenarios reales y/o testbeds, así como un estudio de su comportamiento en escenarios dinámicos y los posibles beneficios de su utilización para distintos superschedulers grids, hypervisors clouds y schedulers en general. El estudio de otras aplicaciones del Merge de Listas (ej. recopilación de estado de sensores en WSN), la utilización de Scoring Dinámico (utilización de más de una función según condiciones generales del sistema en determinado momento) o el 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 resultan direcciones por las que también se podría continuar este trabajo.

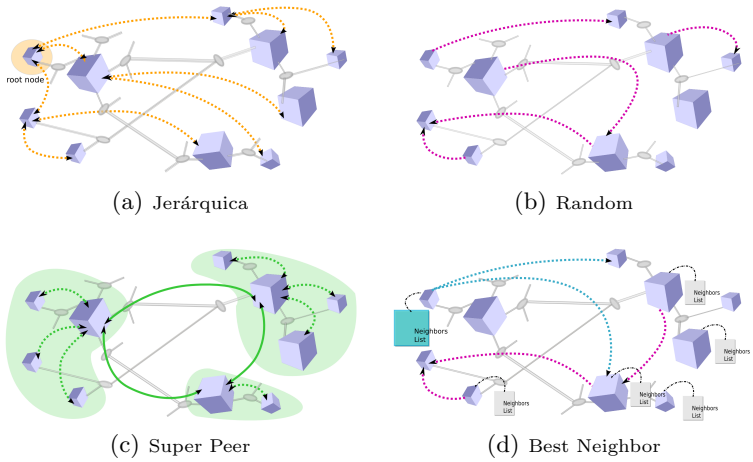
Referencias

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.
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. R. Ranjan, L. Zhao, Peer-to-peer service provisioning in cloud computing environments, Journal of Supercomputing 65 (1) (2013) 154–184.
6. 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.
7. 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.
8. 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.
9. G. Pipan, Use of the TRIPOD overlay network for resource discovery, Future Generation Computer Systems 26 (8).
10. 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.
11. 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.
12. 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.
13. 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).
14. 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.
15. 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.
16. 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

17. R. Albert, H. Jeong, A.-L. Barabási, Internet: Diameter of the World-Wide Web, Nature 401 (1999) 130–131.
18. R. Albert, A.-L. Barabási, Statistical mechanics of complex networks, Reviews of Modern Physics 74 (2002) 47–97.
19. 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.
20. M. E. J. Newman, S. H. Strogatz, D. J. Watts, Random graphs with arbitrary degree distributions and their applications, Physical Review E 64.
21. 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.
22. 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.
23. A. Proskurowski, Minimum broadcast trees, Computers, IEEE Transactions on 100 (5) (1981) 363–366.
24. A. M. Farley, A. Proskurowski, Broadcasting in trees with multiple originators, SIAM Journal on Algebraic Discrete Methods 2 (4) (1981) 381–386.

A. Esquemas del intercambio de mensajes para Jerárquica, Super Peer, Random y Best Neighbor

En los esquemas de la Figura 8 se ejemplifica el intercambio de mensajes para cada una de las cuatro políticas.



(a) Jerárquica (b) Random

(c) Super Peer (d) Best Neighbor

Figura 8: Esquemas del intercambio de mensajes de diferentes políticas de distribución de la información. (a) Los nodos intercambian información sólo con los del nivel inmediato inferior o inmediato superior de la jerarquía establecida. (b) Cada nodo elige de manera aleatoria otro nodo de la red del cual obtener información. (c) Los nodos peers se comunican directamente con un único super-peer, el de su propio cluster, y a través de él con los demás nodos en el sistema. Los nodos super peers reciben solicitudes de los peers en su cluster y solo se envían solicitudes entre sí. (d) Inicialmente cada nodo 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.)

B. Metodología

B.1. Métricas

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 [10]. En Trunfio et al. [12], 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.

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. [10] 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.

LIR (Local Information Rate): Cociente que toma valores entre 0 y 1 e 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:

$$LIR_k = \frac{\sum_{h=1}^{N} f(age_h, expiration_h) \cdot resourceCount_h}{totalResourceCount}$$

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**.

B.2. Simulador: Gridmatrix / SimGrid

Existe una gran variedad de herramientas de simulación que podrían utilizarse, como las mencionadas por Quétier et al. [15]. En particular, para el estudio de políticas de distribución de la información se cuenta con la herramienta de simulación `Gridmatrix` [10, 13], herramienta Open Source basada en `SimGrid`.

`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`). El modelo de red se describe en profundidad en el trabajo de Casanova et al. [16].

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 la Sección B.3 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 `platforms`, y en la Sección B.4 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.

B.3. 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 [8] y en el trabajo de Hiroyuki Kamezawa et al.[9]. El modelo para la infraestructura de red es manejado por SimGrid[10]. 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.

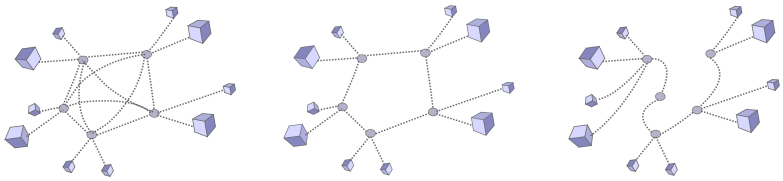
Tal como se ejemplifica en la Figura 9(a), en una topología de Clique cada nodo se encuentra conectado directamente con todos los demás, mientras que en una topología de Anillo cada nodo tiene grado 2, es decir, tiene conexión únicamente con otros dos nodos formando, como su nombre lo indica, una estructura con forma de anillo (tal cual se muestra en la Figura 9(b)). Finalmente, la Figura 9(c) 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. [17] o distintos tipos de redes colaborativas o redes sociales [18–20].

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.

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

[9] 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.

[10] Documentación de `SimGrid` en http://simgrid.gforge.inria.fr.



(a) Clique (b) Anillo (c) Exponencial

Figura 9: Topologías Clique, Anillo y Exponencial de 5 routers y 10 nodos.

B.4. 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 7(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 [21], 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 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` fue necesario aumentar la frecuencia de los eventos de `poll` para 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`[11] 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. Las consultas entre super peer se realizan cada cierto intervalo de tiempo fijo independiente del tiempo de refresco elegido para los peers, salvo para las simulaciones en las que se compararon resultados para distintas longitudes de dicho intervalo.

Para generar los deploys para las simulaciones de la política Jerárquica se fijó la *cantidad de niveles* en tres y se utiliza como nodo generador de la jerarquía al *centerum*[12]. 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 cada longitud del tiempo de refresco (entre 10 y 100).

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	-

Cada paso temporal en la simulación equivale a 1 segundo. 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.

C. Consideraciones Previas

Algunas características de las políticas de distribución de la información sobre recursos que pueden afectar la performance son, por ejemplo, la elección del

[11] METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

[12] Puede encontrarse la definición de *centerum* en el Apéndice C.

host que será la raíz (*root*) de la jerarquía en la política Jerárquica o la cantidad de super-peers, en el caso de Super Peer, según los resultados presentados previamente en [21]. Otros factores que pueden influir en el desempeño de la política Super Peer son la 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 [11].

Algunos de estos factores fueron analizados durante el desarrollo de la tesis y se explicarán a continuación de manera resumida aquí dado que se trata de resultados preliminares que se profundizarán en trabajos futuros.

La elección del nodo que actúa como root (o nodo generador) en Jerárquica impacta fuertemente en la performance en topologías Exponenciales, no así en Cliques, como puede observarse en las Figuras 10(a) y 10(b).

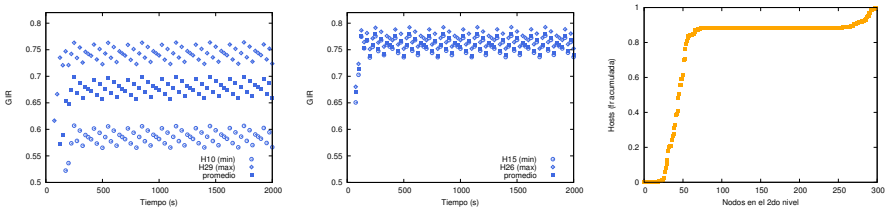
En este contexto, resulta fundamental encontrar un criterio con el cual elegirlo. Probar cada uno de los nodos del sistema no es una metodología escalable y no es sencillo hallar un criterio confiable para seleccionar el root de modo que la performance de la política jerárquica sea razonablemente buena.

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 nodos. El problema general de determinar el MBTime para un conjunto de nodos es NP completo [22]. Podría ser muy rico un análisis de las heurísticas para esta clase de problemas, como las que pueden seguirse en [23, 24], así como explorar los algoritmos utilizados para la generación de las jerarquías pero queda fuera del alcance de este trabajo.

Definimos al *centerum* como *aquel nodo con suma mínima de distancias a los demás nodos*. Utilizamos en este trabajo el *centerum* como nodo generador, dado que se observó que los overlays jerárquicos generados a partir del centerum resultan representativos de las posibles jerarquías generadas para una topología dada. En la figura 10(c) se presentan los resultados para un sistema de 300 nodos de la frecuencia relativa acumulada de hosts que generan jerarquías con la misma cantidad de nodos en el segundo nivel. La performance se maximiza cuando la cantidad de nodos en el segundo nivel de la jerarquía generada aumenta[13], pero la mayor parte de los nodos generan jeraquías similares a la generada por el *centerum*. Concluímos que seleccionar al centerum como root o nodo generador permite obtener un overlay representativo.

Respecto de los factores que afectan la performance de Super-Peer, encontramos que la pérdida de performance que evidencia la política de distribución Super Peer 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 la política de intercambio entre sps. En la Figura 11(a) incluímos los resultados del GIR para Super-Peer al seleccionar distintos porcentajes de nodos como super peers: 2 %, 5 %, 10 % y 20 % en un sistema de 100 nodos. Los resultados de la Figura 11(b) correspon-

[13] No se incluyen aquí los resultados.



(a) Exponencial (b) Clique (c) Centerum

Figura 10: (a) y (b) 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 Exponencial, la performance depende fuertemente de la jerarquía utilizada. (b) Topología Clique, la diferencia en la performance al utilizar distintas jerarquías es casi indistinguible. (c) Se grafica para un sistema de 300 nodos la frecuencia relativa acumulada de hosts que generan jerarquías con la misma cantidad de nodos en el segundo nivel. 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.

den al GIR, para los mismos particionados del mismo sistema, pero variando la frecuencia de intercambio entre super peers.

Al analizar la política Random, usualmente utilizada como peor caso, observamos 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 11(c) al variar los valores de N. Podemos ver en la Figura 11(c) que, por ejemplo, un valor de GIR alrededor de 0.5 se obtiene enviando 6 mensajes cada 60s u 8 mensajes cada 80s. 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.

En este trabajo se utilizará como criterio para comparar los resultados obtenidos 1-Random (o Random).

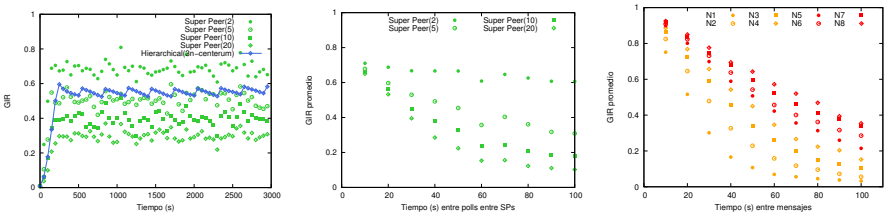
D. Normalización de $f_{scoring}$

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.

Simplificando la notación:

Paula Verghelet



(a) Cantidad de super peers (b) Frecuencias de poll sps. (c) Random

Figura 11: (a) Evolución del GIR para un sistema de 100 nodos de topología exponencial para distintos tamaños de cluster (distintos porcentajes de super peers). (b) 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. (c) 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). 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.

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

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

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 el Apéndice B.2[14].

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 12(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:

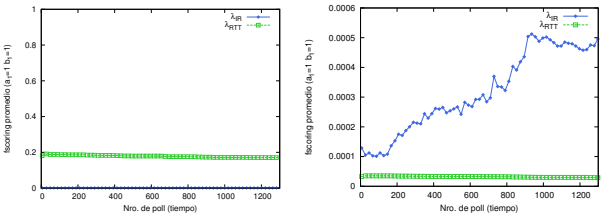
$$\overline{f_{\text{BN}}}\Big|_{n_i} = \frac{1}{N-1} \sum_{n_j \in \text{vecinos}(n_i)} \lambda_{IR}(n_j)$$

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

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

[14] 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.

distinto para cada host y, posiblemente, para cada sistema, incluso puede depender de la performance, por lo que parece mejor fijar esa relación desde el inicio seleccionando valores adecuados para a_1 y b_1.



(a) Promedio de λ_{IR} y λ_{RTT} (b) Términos equiparados.

Figura 12: 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) 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 (b) se ven los promedios una vez agregada la constante.

E. Estrategias para la selección de destinos de los eventos de push

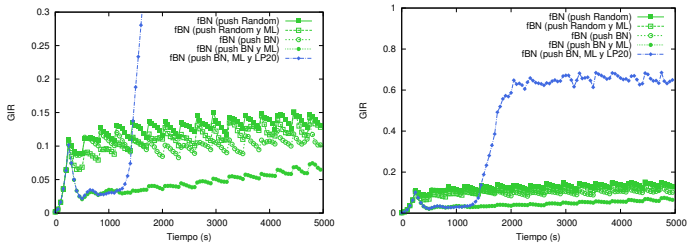
A medida que el tiempo transcurre, la información que posee cada nodo sobre el resto del sistema va desactualizándose. La publicación de recursos (`push`), permite informar a otros nodos qué recursos propios están disponibles. Al llegar una publicación de recursos de algún host es posible agregarlo a 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.

La Figura 13(a) presenta los resultados de la evolución del GIR para distintas variaciones de la política fBN, 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 f_{BN} y se envía sólo la información de recursos propios.
- `push BN y ML` : se elige el nodo destino utilizando f_{BN}, 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.



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

Figura 13: 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.

Como se dijo en la Sección 3.3, 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. Surge como mejor candidato para recibir una publicación de recursos (destino de `push`). Sin embargo, la Figura 13(a) muestra que utilizar esta técnica 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 comienzan a repetir la elección del destino. 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.

En la Sección 3.3 proponemos agregar un mecanismo de control de repeticiones, que evita esta *clusterización temprana*, logrando buena performance Figura 13(b).