UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Departamento de Electrónica e Computación

TESIS DOCTORAL

MODELADO ANALÍTICO DEL RENDIMIENTO DE APLICACIONES EN SISTEMAS PARALELOS

Presentada por:

Diego Rodríguez Martínez

Dirigida por:

Francisco Fernández Rivera

José Carlos Cabaleiro Domínguez

Vicente Blanco Pérez

Santiago de Compostela, marzo 2011

ISBN 978-84-9887-761-8 (Edición digital PDF)

Francisco Fernández Rivera, Profesor Catedrático de Universidad del Área de Arquitectura de Computadores de la Universidad de Santiago de Compostela

José Carlos Cabaleiro Domínguez, Profesor Titular de Universidad del Área de Arquitectura de Computadores de la Universidad de Santiago de Compostela

Vicente Blanco Pérez, Profesor Titular de Universidad del Área de Arquitectura de Computadores de la Universidad de La Laguna

HACEN CONSTAR:

Que la memoria titulada **MODELADO ANALÍTICO DEL RENDIMIENTO DE APLICACIONES EN SISTEMAS PARALELOS** ha sido realizada por D. **Diego Rodríguez Martínez** bajo nuestra dirección en el Departamento de Electrónica e Computación de la Universidad de Santiago de Compostela, y constituye la Tesis que presenta para obtar al grado de Doctor en Física.

Santiago de Compostela, 29 de marzo de 2011

Francisco Fernández Rivera
Codirector de la tesis
Director del Departamento de Electrónica y Computación

José Carlos Cabaleiro Domínguez
Codirector de la tesis

Vicente Blanco Pérez
Codirector de la tesis

Diego Rodríguez Martínez
Autor de la tesis

A Raquel

Trying is the first step to failure.

Homer J. Simpson

You tried your best and failed miserably.
The lesson is never to try.

Homer J. Simpson

Eh, everybody makes mistakes.
That's why they put erasers on pencils.

Lenford L. Leonard, "Lenny"

Hey! Hey, hey, stop it! Stop it! Why are you guys jumping to such ridiculous conclusions? Haven't you ever heard of Occam's Razor? "The simplest explanation is probably the correct one".

Elizabeth Marie "Lisa" Simpson

AGRADECEMENTOS

En primeiro lugar, quero expresar o meu máis sincero agradecemento ós meus directores de tese, os profesores Francisco F. Rivera, José C. Cabaleiro e Vicente Blanco, pola súa confianza no meu traballo. Agradecemento que me gustaría extender ó profesor Tomás F. Pena, xa que considero que tamén é, aínda que de xeito non oficial, director desta tese. Estou profundamente agradecido ós catro porque son os responsables do meu crecemento profesional —e tamén persoal— durante os anos empregados na elaboración desta tese.

Quero expresar tamén a miña gratitude ó Departamento de Electrónica e Computación, da Universidade de Santiago de Compostela, e ó Departamento de Estadística, Investigación Operativa y Computación, da Universidad de La Laguna, por poñer ó meu alcance a infraestructura adecuada para o desenvolvemento desta tese. Así mesmo, quero agradecer tamén a financiación que me permitiu a súa realización: a beca Predoctoral de Formación de Personal Investigador asociada ó proxecto de investigación TIN2004-07797-C02-01 do Ministerio de Ciencia y Tecnología, os proxectos TIN2005-09037-C02-01 e TIN2007-67537-C03-01 do Ministerio de Educación y Ciencia, os proxectos TIN2008-06570-C04-03 e TIN2010-17541 do Ministerio de Ciencia e Innovación de España, a rede europea *High Performance and Embedded Architecture and Compilation* (HiPEAC-2), a rede española de Computación de Altas Prestaciones sobre Arquitecturas Paralelas Heterogéneas (CAPAP-H) e a rede Galega de Computación de Altas Prestacións (GHPC).

Gracias tamén a tódolos meus compañeiros do Departamento de Electrónica e Computación e do Grupo de Arquitectura de Computadores, que xa son moitas as experiencias compartidas. En especial, os compañeiros dos «clubes» ós que pertenzo: a PFF (a pesar das poucas alegrías, síntome moi orgulloso de formar parte da CdE), o CC, AULUSC e o equipo de Física (¡Que grande esa tempada 2005-2006, cando aínda eramos DELCO, e nos sacaron a foto de «fútbol base»!). En particular, gustaríame facer unha mención especial a Dani, Roberto,

Xulio, Óscar e Pichel, os meus compañeiros de despacho, café, viños, copas, tertulias e penas (¡sobre todo esto último!).

Moitas graciñas a Linus Torvalds por Linux e a Leslie Lamport por LaTeX.

Finalmente, quero darlle as gracias a meus pais, porque considero que a súa inversión na miña educación tivo un éxito rotundo, e tamén ós meus amigos e primos —¡que non amigos!— polos momentos de alegrías que me permitiron sobrevivir estos anos (mención especial para Molina, por ensinarme a xogar ó mus). Tamén á familia Blanco Ríos por coidarme mentres estiven en La Laguna. En especial, quero darlle as gracias a dúas persoas. A Pablo polo café de pola mañá —e xa van alá moitos anos—, que é un pequeno pracer ó que non tiña pensado renunciar. E a Raquel, por moitas cousas... pero sobre todo pola túa paciencia e cariño durante estos anos, ¡e os que che quedan!

Santiago de Compostela, 29 de marzo de 2011

Índice general

INTRODUCCIÓN

El desarrollo de nuevas arquitecturas en sistemas paralelos ha sido siempre un área de investigación muy activa debido, en gran parte, a la incesante demanda por parte de la comunidad científica de un mayor poder computacional. En las últimas décadas, esta progresión se ha visto acelerada por las nuevas tendencias en el desarrollo de microprocesadores, por lo que los sistemas actuales disponen de un número cada vez mayor de núcleos computacionales, dispuestos en una estructura intrínsecamente jerárquica. Sin embargo, esta mayor capacidad computacional conlleva también una mayor dificultad para el desarrollo de aplicaciones que aprovechen eficientemente los recursos computacionales, por lo que es necesario disponer de mecanismos apropiados para analizar, comprender y predecir el comportamiento de las aplicaciones en los sistemas paralelos. En este sentido, las herramientas de análisis del rendimiento son un instrumento fundamental para explorar y explotar el potencial de estos sistemas. Además, los resultados obtenidos por estas herramientas también proporcionan un enfoque empírico de las posibilidades de mejora, indicando posibles direcciones para el desarrollo de futuros sistemas.

El hecho de que exista un desarrollo continuado de nuevas arquitecturas condiciona la investigación asociada al análisis del rendimiento en el ámbito de la computación de altas prestaciones. Las técnicas, los métodos y las herramientas deben ser capaces de adaptarse a las nuevas exigencias impuestas por la evolución de los sistemas paralelos. En cualquier caso, el desarrollo de una metodología universal de análisis del rendimiento es un objetivo prácticamente inalcanzable debido al amplio, y cada vez mayor, espectro de arquitecturas y configuraciones. Para poder abordar el análisis del rendimiento de sistemas paralelos han surgido diferentes aproximaciones que se centran en aspectos concretos del análisis del rendimiento. La aproximación más inmediata para realizar un análisis del rendimiento son los programas de prueba o *benchmarks*. La ejecución de un *benchmark* en un sistema concreto

permite disponer de una estimación real del comportamiento de otras aplicaciones en ese sistema. La validez de la estimación dependerá del grado de similitud entre las aplicaciones y el código del *benchmark*. En cualquier caso, esta aproximación es un mecanismo adecuado para establecer una clasificación de los sistemas paralelos como, por ejemplo, la lista TOP500. Las herramientas de diagnóstico también son muy populares, ya que permiten realizar, desde un punto de vista fundamentalmente descriptivo, un análisis detallado de la ejecución de aplicaciones paralelas. En general, este tipo de herramientas caracterizan una aplicación determinada en función de la información de las métricas de rendimiento monitorizadas durante su ejecución en un determinado sistema. En este caso, la experiencia y la pericia del analista son fundamentales para evaluar los resultados obtenidos y extrapolar conclusiones del comportamiento de otras aplicaciones o sistemas paralelos.

Mediante las técnicas de modelado, se puede construir una representación abstracta del comportamiento de las aplicaciones y los sistemas paralelos. De esta forma, los detalles de implementación se ocultan en el propio proceso de modelado. Esta aproximación de análisis del rendimiento proporciona el nivel de abstracción necesario para poder comparar y contrastar eficazmente diferentes algoritmos, implementaciones y sistemas, así como para predecir su comportamiento. En el ámbito de la computación de altas prestaciones, las herramientas de modelado se pueden catalogar en dos grandes grupos: las basadas en simulación y las que se establecen con modelos analíticos. Los simuladores definen un modelo de arquitectura sobre el que se reproduce el comportamiento de las aplicaciones. De esta forma, es posible recrear cualquier sistema que encaje dentro de los parámetros de diseño del simulador, incluyendo arquitecturas y sistemas experimentales. Las técnicas actuales de simulación de arquitecturas paralelas son muy precisas, pero la evaluación de los resultados conlleva un elevado coste temporal, por lo que esta aproximación no es adecuada para tareas en las que el tiempo de evaluación de los modelos sea un factor crítico como, por ejemplo, en la planificación de trabajos en un sistema de computación paralelo. Por el contrario, los modelos analíticos describen el comportamiento de las aplicaciones y sistemas paralelos mediante expresiones matemáticas y, por lo tanto, requieren un tiempo de evaluación muy pequeño. Sin embargo, para poder describir sistemas reales mediante expresiones analíticas, es necesario realizar suposiciones y simplificaciones que, generalmente, merman la precisión de este tipo de aproximación.

El objetivo principal de esta tesis es el desarrollo de un entorno de análisis del rendimiento que permita obtener, de un modo sencillo, modelos analíticos muy precisos de aplicaciones en sistemas paralelos. El modelado de una aplicación es una tarea compleja porque es necesario establecer las relaciones entre los diferentes componentes implicados en la ejecución de la

aplicación. La inspección manual del código fuente es una aproximación muy utilizada, pero es un procedimiento muy exigente que implica un elevado tiempo de desarrollo y un conocimiento en profundidad tanto del código fuente como del sistema paralelo subyacente. Existen diversas alternativas que permiten agilizar y automatizar el proceso pero que requieren, en cualquier caso, un conocimiento experto de la plataforma de ejecución y de la aplicación.

La metodología que se propone en esta tesis proporciona un mecanismo automático de generación de modelos analíticos a partir de la información de rendimiento que se puede obtener tras la ejecución de las aplicaciones en un sistema concreto. En particular, esta metodología ha sido diseñada para que el usuario concentre sus esfuerzos en el diseño experimental y en la valoración de los resultados, de tal forma que no sean necesarios conocimientos detallados del algoritmo de la aplicación o de la arquitectura subyacente. En general, las diferentes aproximaciones de modelado del rendimiento suelen exigir un importante esfuerzo por parte del analista, descuidando otros aspectos importantes en el proceso de análisis como, por ejemplo, la adecuada selección de las métricas y parámetros para el correcto estudio del rendimiento. En este sentido, la metodología propuesta pretende redistribuir la carga de trabajo del proceso de análisis, de tal manera que el usuario se concentre en el diseño experimental, seleccionando un conjunto adecuado de métricas y parámetros que alimenten los mecanismos automáticos de instrumentación y modelado, definidos en la propia metodología.

En particular, se ha definido una metodología de modelado que consta de dos fases interrelacionadas. La primera consiste en la instrumentación del código para obtener información de rendimiento acerca del comportamiento de la aplicación durante su ejecución en múltiples situaciones experimentales. Esta fase se ha desarrollado a partir de los mecanismos de instrumentación de la herramienta `call`. Esta herramienta permite introducir directivas de compilación con las órdenes necesarias para recopilar, con una mínima perturbación en el comportamiento de la aplicación, el valor de las métricas de rendimiento seleccionadas al ejecutarse el código instrumentado. En la segunda fase, estos valores de rendimiento son analizados mediante técnicas estadísticas y se construye automáticamente, a partir de las métricas y parámetros de rendimiento seleccionados por el usuario, un modelo analítico preciso del comportamiento de la aplicación. Esta fase ha sido implementada mediante una librería en el entorno estadístico R. En particular, el procedimiento de modelado ha sido desarrollado mediante técnicas de selección de modelos basadas en el criterio de información de Akaike, una herramienta objetiva que permite cuantificar la idoneidad de un modelo particular respecto de un conjunto finito de modelos. Este criterio es utilizado ampliamente en el campo de la biología y la econometría, en situaciones en las que es difícil realizar experimentos controlados

y únicamente se dispone de datos empíricos con un elevado número de variables independientes. El diseño global del entorno presenta una estructura flexible y escalable, ya que la interfaz entre ambas fases se realiza a través de ficheros XML. Esta característica permite, por ejemplo, sustituir el mecanismo de instrumentación sin que sea necesario modificar la fase de análisis o utilizar los datos de rendimiento en otro tipo de análisis o herramienta. Además, el diseño de ambas fases es muy modular, lo que facilita la ampliación y sustitución de las funcionalidades del entorno.

Organización de la tesis

- En el capítulo 1 se muestra una breve revisión del estado del arte en el análisis del rendimiento. En particular, se discute con detalle aspectos referentes a los sistemas paralelos y se hace un breve repaso de las herramientas de análisis del rendimiento actuales.

- El entorno de análisis del rendimiento desarrollado en esta tesis se describe detalladamente en el capítulo 2. En primer lugar, se discuten las motivaciones que han propiciado el desarrollo de este entorno y, a continuación, se describen detalladamente la estructura y las características del entorno, así como de los diferentes componentes que lo conforman.

- En el capítulo 3 se presenta el modelo LoOgGP, un nuevo modelo de la familia LogP que puede ser interpretado como una generalización de los modelos LogP y LogGP. En este capítulo también se describe un método de caracterización del comportamiento real de las comunicaciones en un sistema paralelo. Este método de caracterización, basado en este nuevo modelo, se integra en el entorno de análisis descrito en el capítulo 2.

- En el capítulo 4 se describe un mecanismo de caracterización de aplicaciones paralelas basado en la selección de modelos utilizando el criterio de información de Akaike. En primer lugar, se presentan las motivaciones y fundamentos que han propiciado este mecanismo de modelado. A continuación, se describe detalladamente la implementación de este método en el entorno de análisis y, finalmente, se realiza una comparación de los resultados obtenidos mediante esta técnica, para diferentes códigos, con modelos teóricos basados en análisis algorítmicos.

- En el capítulo 5 se analizan los resultados obtenidos mediante el mecanismo de modelado basado en la selección de modelos, para diferentes casos de estudio. En particular,

se presentan los resultados obtenidos para diferentes códigos del NPB (*NAS Parallel Benchmark*) y del *benchmark* HPL (*High Performance LINPACK*). En este último caso, se utiliza la simulación de la planificación de trabajos en un cluster para comparar la predicción del modelo obtenido mediante la selección de modelos y un modelo teórico del benchmark HPL. Por último, se presenta un caso de estudio en el que se describe un método de caracterización de los fallos cache utilizando modelos basados en la selección de modelos.

– Finalmente se concluye con una exposición de conclusiones y posibles líneas futuras de trabajo.

Capítulo 1

Análisis del rendimiento de sistemas paralelos

Uno de los mayores retos de la computación paralela es aprovechar el máximo potencial que ofrece un sistema computacional con más de un procesador, lo que permitirá resolver problemas de mayor dimensionalidad en un menor tiempo. Además, el elevado coste (tanto en desarrollo como en mantenimiento) de estos sistemas exige que la utilización de los recursos disponibles sea óptima y, por lo tanto, las aplicaciones paralelas deben explotar eficientemente los recursos disponibles. Actualmente, y a diferencia de la programación secuencial, el desarrollo de software eficiente para un sistema paralelo exige un conocimiento detallado del comportamiento de la arquitectura del sistema paralelo, independientemente de las abstracciones que ofrecen los diferentes lenguajes de programación paralelos. El análisis de rendimiento es la herramienta que permite al programador evaluar, analizar e incluso predecir el rendimiento de una aplicación en un determinado sistema paralelo.

En este capítulo se mostrará un breve repaso del estado del arte en el análisis de rendimiento de sistemas paralelos. En primer lugar, se hará una breve revisión de las arquitecturas paralelas distribuidas más habituales en sistemas de computación de alto rendimiento. A continuación se discutirán aspectos generales referentes al análisis de rendimiento, con especial énfasis en aquellos relacionados con los sistemas paralelos y, en particular, se profundizará en el modelado analítico de las comunicaciones en sistemas paralelos. Finalmente, se hará una breve revisión de las herramientas de análisis de rendimiento de aplicaciones y sistemas paralelos.

1.1 Sistemas paralelos en entornos HPC

El auge de las sistemas paralelos ha sufrido un importante impulso en los últimos años debido, entre otros factores, a las restricciones de consumo energético de los microprocesadores y que han derivado en la integración de múltiples núcleos en un único chip [3, 17, 44, 51, 63, 97, 146]. Sin embargo, el concepto de arquitectura paralela ha sido ampliamente explotado desde hace varias décadas, especialmente en los entornos de Computación de Altas Prestaciones (*High Performance Computing*, HPC). En cualquier caso, en este contexto, un sistema paralelo puede definirse como una colección de elementos de procesamiento independientes que cooperan y se comunican entre sí, y que son capaces de resolver de forma conjunta un único problema. Obviamente, esta definición es poco precisa y abarca desde los recientes sistemas multinúcleo hasta los entornos Grid [59] o Cloud [12].

El amplio espectro de posibles aproximaciones para aplicar un procesamiento paralelo, junto con el continuo desarrollo de nuevas configuraciones, dificulta la unificación de los criterios de clasificación de las arquitecturas paralelas. La primera clasificación ampliamente difundida de arquitecturas de computadores basada en el flujo de instrucciones y de datos es conocida como la Taxonomía de Flynn [56]. Esta clasificación distingue cuatro tipos de sistemas computacionales. A pesar de que actualmente, la mayoría de los sistemas no encajan exactamente en una de sus categorías, la taxonomía de Flynn es ampliamente utilizada porque es muy sencilla y proporciona una primera aproximación académicamente correcta. Sin embargo, esta taxonomía no es adecuada para clasificar los sistemas paralelos actuales porque, a pesar de las diferencias que presentan, casi todos ellos coincidirían dentro de la misma categoría.

La principal clasificación de los sistemas paralelos actuales en entornos HPC se basa en el paradigma de programación de estos sistemas [39]. Esta clasificación distingue básicamente entre tres grandes familias: memoria compartida, procesamiento paralelo de datos y arquitecturas distribuidas (o de paso de mensajes). Esta clasificación es también muy genérica y existen otras aproximaciones diferentes, pero es generalmente considerada como una buena aproximación para la clasificación de la mayoría de los sistemas utilizados en entornos HPC.

1.1.1 Sistemas de memoria compartida

En los sistemas de memoria compartida, también conocidos como multiprocesadores, todos los procesadores pueden acceder a cualquier dirección de memoria del sistema utilizando

las instrucciones convencionales de acceso a memoria. El mecanismo más habitual para programar estos sistemas es mediante OpenMP [134] que, a través de directivas de compilación, proporciona una interfaz sencilla para generar código paralelo.

Según su taxonomía, los sistemas de memoria compartida pueden clasificarse en procesadores simétricos (*Symmetric MultiProcessor*, SMP) —también denominados UMA (*Uniform Memory Access*)— y sistemas de acceso a memoria no uniforme (*Non-Uniform Memory Access*, NUMA) [73]. La arquitectura SMP utiliza un bus de comunicaciones para conectar los procesadores con la memoria, garantizando que el coste de acceso a memoria es el mismo para todos los procesadores, independientemente de la posición de memoria solicitada. Los problemas de contención en el bus del sistema condicionan, en gran medida, la escalabilidad de estos sistemas. Por el contrario, en los sistemas NUMA la distribución de la memoria no es simétrica, por lo que el coste de acceso a la memoria dependerá de la distancia entre el procesador y la posición de memoria solicitada. El esquema más sencillo de los sistemas NUMA asocia una sección de memoria a cada procesador, de tal manera que el acceso de cada procesador a su memoria asociada presenta una baja latencia mientras que el acceso a posiciones de memoria que estén asociadas a otro procesador implica una mayor penalización temporal. Esta estructura jerárquica puede tener más niveles en la arquitectura.

1.1.2 Procesamiento paralelo de datos - *Stream Processing*

La característica principal de estos sistemas es que una operación se ejecuta en paralelo en cada elemento de una estructura regular, actuando sobre diferentes datos. Este tipo de sistemas encaja dentro de la categoría *Single Instruction Multiple Data* [56] y es la base de los procesadores vectoriales.

En los entornos HPC actuales, los sistemas basados únicamente en este paradigma de programación han desaparecido —se puede observar en la evolución de la lista TOP500 [160]—, pero este tipo de técnicas siguen vigentes en otros sectores como, por ejemplo, en el procesamiento de imágenes mediante tarjetas gráficas (*Graphics Processing Units*, GPUs) o los *systolic arrays*. En los últimos años, ha cobrado especial relevancia la programación de propósito general de GPUs (*General-Purpose computation on Graphics Processing Units*, GPGPU), en la que se utiliza la GPU como un coprocesador matemático para realizar operaciones sobre grandes estructuras de datos [109]. En ciertas aplicaciones, este tipo de técnica obtiene un rendimiento muy superior al obtenido con microprocesadores de propósito gene-

ral [151, 174, 35]. De hecho, los sistemas híbridos que integran GPUs se han situado en los primeros puestos de la última lista TOP500 [158].

1.1.3 Arquitecturas paralelas distribuidas

Los sistemas paralelos distribuidos están formados por elementos computacionales independientes (nodos) unidos mediante una red de interconexión. La principal característica de estos sistemas es que los accesos a memorias no locales —es decir, a datos almacenados en la memoria local de otros nodos— exigen una comunicación explícita entre los nodos implicados. El paradigma de programación de paso de mensajes es el mecanismo de programación habitual en este tipo de sistemas. Aunque existen otras alternativas, la librería MPI [124, 128] se ha convertido en el estándar de facto del paradigma de paso de mensajes en arquitecturas paralelas distribuidas.

Los sistemas paralelos distribuidos son altamente escalables y, además, permiten introducir nuevos niveles arquitecturales dentro de cada nodo computacional. Estas características de escalabilidad y versatilidad han favorecido su desarrollo en los entornos HPC. En particular, utilizando la lista TOP500 como referencia, las arquitecturas más comunes en HPC de los últimos años han sido arquitecturas paralelas distribuidas: clusters, constelaciones y computadores masivamente paralelos [160].

Clusters y constelaciones

Aunque existen diferentes interpretaciones [21, 47], un cluster puede definirse —en un sentido amplio— como un sistema paralelo formado por nodos independientes conectados a través de una red de interconexión dedicada. En general, los clusters son sistemas poco acoplados, lo que permite que sean muy robustos y, al mismo tiempo, facilita la administración y desarrollo de los propios sistemas. Además de su flexibilidad y versatilidad, el éxito de estos sistemas en los entornos de HPC se debe a la excelente relación precio-rendimiento. De hecho, la mayoría de las configuraciones pueden catalogarse como *commodity hardware*, es decir, sus componentes (los nodos y la red) pueden encontrarse en el mercado, ya que no han sido desarrollados específicamente para un sistema particular. El impacto de los cluster en los entornos HPC se refleja claramente en la lista del TOP500 de noviembre de 2010, en la que más del 80 % de los sistemas de la lista presentan una configuración que podría encajar en la definición de cluster.

La configuración más sencilla de cluster, conocida como Beowulf, consiste en múltiples PCs conectados mediante una red de tipo Ethernet, generalmente utilizando un sistema operativo GNU/Linux. Sin embargo, actualmente los sistemas más habituales en HPC están formados por procesadores multinúcleo (como las familias Intel® Xeon® y AMD Opteron™, por ejemplo) interconectados por redes InfiniBand™ o Gigabit Ethernet —en este grupo podríamos incluir un gran número de los clusters de la lista TOP500—. La aparición de nodos formados por múltiples núcleos produce, inherentemente, dos niveles de comunicación diferentes: internodo e intranodo. Debido a las diferentes características que presentan estos dos niveles de comunicación, la adaptación de los programas a esta distribución jerárquica de los elementos computacionales es fundamental para aprovechar eficazmente estos sistemas [142].

Las constelaciones se consideran una subclase de cluster en la que el número de procesadores por nodo es superior al número de nodos del sistema completo [47]. Es decir, el nivel de paralelismo dominante en las constelaciones es el nivel intranodo. En general, los nodos de estos sistemas están formados por multiprocesadores (SMP o sistemas NUMA), con un gran número de núcleos. Para obtener alto rendimiento de este tipo de sistemas es necesario utilizar un paradigma de programación híbrido, que utilice paso de mensajes en las comunicaciones internodo, mientras dentro de cada nodo se pueden ejecutar diferentes hilos bajo el paradigma SMP.

Computadores masivamente paralelos

Al contrario que los clusters, los sistemas masivamente paralelos (*Massively Parallel Processing*, MPP) son sistemas distribuidos con un nivel de acoplamiento muy alto. En los sistemas MPP los nodos son independientes (es decir, contienen su propia memoria y una copia del sistema operativo) y están conectados a través de una red de alta velocidad, pero se gestionan como un único sistema, de manera similar a un multiprocesador. En general, el rendimiento de estos sistemas es mayor que el de los clusters, ya que algunos componentes —especialmente la red de interconexión— suelen diseñarse específicamente para estos sistemas. De hecho, esta arquitectura suele ocupar los primeros puestos de la lista TOP500.

Redes de interconexión

La escala de los actuales sistemas paralelos distribuidos convierte a las redes de interconexión en elementos fundamentales en el rendimiento de estos sistemas. Además de las soluciones propietarias y los diseños específicos, en la última lista TOP500 podemos encon-

trar fundamentalmente dos familias comerciales de redes de interconexión: Gigabit Ethernet e InfiniBand™. Estas dos familias son las opciones más utilizadas por los *commodity* clusters.

- Gigabit Ethernet es una ampliación del estándar Ethernet (concretamente el estándar 802.3 de IEEE) que permite la transmisión de tramas Ethernet a una velocidad de 1 Gbps, utilizando una extensión del protocolo CSMA/CD de sus predecesoras (Ethernet y Fast Ethernet) [105, 136]. En su configuración más habitual, utiliza redes conmutadas *full-duplex* y, además de cables de cobre, permite el uso de fibra óptica. En un sentido más amplio, también se consideran parte de la familia Gigabit Ethernet las extensiones del estándar Ethernet con velocidades de transmisión superiores, como el estándar 10 Gigabit Ethernet, con una velocidad nominal de 10 Gbps, basado fundamentalmente en el uso de fibra óptica y conexiones *full-duplex*.

- InfiniBand™ es una especificación que define una arquitectura de comunicación específicamente diseñada para ser escalable y que presenta baja latencia —menor que en las redes Gigabit Ethernet [79]—, alto ancho de banda, tolerancia a fallos y calidad de servicio (*Quality of Service*, QoS). Sus especificaciones son desarrolladas y mantenidas por la InfiniBand™ Trade Association [86], formada por empresas importantes en el sector de las tecnologías de la información, lideradas por IBM, Intel, Mellanox, Oracle, QLogic, System Fabric Works y Voltaire.

1.1.4 Computación Grid y computación Cloud

La computación Grid es una tecnología que permite utilizar de forma coordinada recursos que no están sujetos a un control centralizado [59, 58]. En este sentido, es una nueva forma de computación distribuida, en la cual los recursos pueden ser heterogéneos y se encuentran en dominios administrativos diferentes. La computación Grid establece los mecanismos necesarios para conseguir la cooperación y la unión de recursos entre diferentes instituciones, proporcionando la capacidad suficiente que permita resolver problemas cuya resolución, en un tiempo razonable, requiere una gran cantidad de recursos [19, 43].

La computación Cloud es un nuevo modelo de prestación de servicios de negocio y tecnología [12], que permite al usuario acceder a un catálogo de servicios estandarizados y responder a las necesidades de su negocio, de forma flexible y adaptativa. Los Clouds pueden interpretarse como una reserva de recursos virtualizados fácilmente utilizables y accesibles (hardware, plataformas de desarrollo y/o servicios). Estos recursos pueden ser reconfigurados

dinámicamente para adaptarse a una carga variable, consiguiendo una utilización óptima de los recursos [52, 141].

1.2 Análisis del rendimiento

El análisis del rendimiento es la herramienta que permite evaluar y predecir el coste computacional de la ejecución de las aplicaciones. Desde un punto de vista teórico, el objetivo principal del análisis del rendimiento es determinar *la forma correcta de funcionamiento* de la aplicación en un sistema concreto y que nos pueda conducir a una mayor productividad. Por lo tanto, el análisis de rendimiento debe proporcionar al usuario la información necesaria para diagnosticar el comportamiento de una aplicación al ejecutarla en un determinado sistema. En la práctica, la evaluación del rendimiento se consigue mediante la caracterización del rendimiento, lo que, a su vez, permite predecir el comportamiento de las aplicaciones en futuras ejecuciones. Los resultados del análisis de rendimiento sirven también como punto de referencia para el desarrollo de nuevos entornos de ejecución, ya que la caracterización del rendimiento permite identificar las carencias o debilidades de los sistemas al ejecutar determinadas aplicaciones. Debido a la inherente complejidad de los sistemas paralelos, el análisis de rendimiento se ha convertido en una herramienta fundamental en este tipo de sistemas [46, 110].

Actualmente, no existe una metodología universal para realizar un análisis exhaustivo del comportamiento de una aplicación en un sistema paralelo, por lo que este análisis ha sido abordado desde diferentes puntos de vista y utilizando diferentes técnicas que, generalmente, se centran en aspectos concretos del rendimiento. Por otro lado, la presentación de datos de rendimiento de sistemas paralelos requiere un tratamiento especial, ya que el volumen de información obtenido puede llegar a ser muy elevado. Las técnicas de visualización desarrolladas en este contexto proporcionan un método eficaz para resumir y destacar aspectos macroscópicos (en general, no triviales) de la ejecución de un programa paralelo.

1.2.1 Modelos

Los modelos son una herramienta fundamental del análisis de rendimiento. Un modelo es una abstracción que permite obtener una descripción concisa de un sistema. En general, es prácticamente imposible una caracterización exacta de un sistema real, por lo que los modelos suelen centrarse en los aspectos más relevantes. En cualquier caso, es evidente que el proceso

de generación de modelos tiene un componente subjetivo muy importante, ya que se debe identificar cuáles son los aspectos más relevantes del sistema y cómo es la interacción entre sus diferentes componentes.

Aunque no existe una clasificación generalizada, los modelos de rendimiento de sistemas paralelos pueden dividirse en modelos arquitecturales y modelos de aplicaciones [171, 23]. Los modelos arquitecturales describen de forma genérica el comportamiento de un sistema. Es decir, estos modelos definen una máquina abstracta que implementa los procesos de interacción entre los componentes del sistema. Los modelos arquitecturales más comunes utilizan parámetros escalares para definir y categorizar las interacciones descritas en la máquina abstracta (como, por ejemplo, los modelos LogP [37] y BSP [162]). Los modelos de aplicaciones describen el comportamiento de las aplicaciones paralelas al ejecutarse en un determinado sistema. Puesto que el rendimiento de las aplicaciones está inherentemente ligado al sistema sobre el que se ejecuta, esta clase de modelos deben contener un modelo arquitectural subyacente. Por ejemplo, el comportamiento de una aplicación puede describirse en función de los parámetros de un modelo arquitectural; en este caso particular, el modelo de aplicación será válido para cualquier sistema cuyas características encajen en la descripción de la máquina abstracta. En cualquier caso, aunque no es necesario que exista una declaración explícita del sistema subyacente, el binomio aplicación-sistema es indivisible en este tipo de modelos. En otras palabras, existe una dependencia inherente entre el comportamiento de la aplicación y el sistema paralelo de ejecución.

1.2.2 Métricas y parámetros

En el contexto de análisis de rendimiento, el término «métrica» se utiliza para referirse a magnitudes o funciones que pueden ser utilizados como criterio para evaluar algún aspecto del rendimiento, mientras que el término «parámetro» se refiere a cualquier magnitud o característica que afecte al rendimiento [91]. El porcentaje de uso de CPU y el número de fallos cache, son dos ejemplos de métricas. El número de procesos en un sistema paralelo o las variables de la aplicación (número de iteraciones del algoritmo, dimensión de los vectores o matrices, etc.) son ejemplos de parámetros. La métrica más importante en análisis de rendimiento es el tiempo de ejecución o de respuesta —el tiempo entre el inicio y el final de un determinado evento— porque, generalmente, la minimización del tiempo de ejecución de una aplicación es el objetivo fundamental del propio análisis de rendimiento.

En general, no es necesario conocer el valor de las métricas o parámetros de rendimiento, porque el resultado final de la ejecución de una aplicación es indiferente de los valores de rendimiento. En otras palabras, los valores de rendimiento son factores independientes que no afectan a la correcta ejecución de una aplicación. En algunos casos, es necesario introducir mecanismos de medida especializados que permitan la lectura de estos valores. Por ejemplo, para acceder a la información almacenada en los contadores hardware de rendimiento implementados en la mayoría de los microprocesadores actuales es necesario utilizar una librería especializada [135, 28].

En los sistemas paralelos se utilizan parámetros que caracterizan el tamaño de la ejecución paralela, en función del número de procesos, el número de procesadores o, en general, el número de unidades computacionales. Los niveles de paralelismo de la arquitectura subyacente determinarán, en última instancia, el número de parámetros asociados a la ejecución de una aplicación en el sistema paralelo. Por ejemplo, en un cluster homogéneo con procesadores multinúcleo se diferencian, al menos, dos parámetros asociados al carácter paralelo del sistema: el número de nodos y el número de núcleos.

Se denominan métricas derivadas a aquellas métricas que se obtiene mediante una función cuyas variables son magnitudes medibles. El número de operaciones en punto flotante por segundo (*FLoating point OPerations per Second*, FLOPS) y el número de instrucciones por segundo (*Million Instructions Per Second*, MIPS) son ejemplos de métricas derivadas [39]. En sistemas paralelos es habitual utilizar métricas derivadas, como la aceleración o la eficiencia, para caracterizar la escalabilidad de las aplicaciones en un sistema paralelo [39, 171].

1.2.3 Técnicas de medida del rendimiento

Puesto que el valor de las métricas y parámetros de rendimiento es, en general, indiferente para la correcta ejecución de una aplicación, es preciso introducir explícitamente algún mecanismo que gestione el acceso y el almacenamiento de los valores de rendimiento durante la ejecución de una aplicación. Aunque no existe una clasificación claramente establecida, en los entornos HPC se distinguen dos alternativas para clasificar las técnicas de medida del rendimiento. En función del instante en el que se realizan las medidas, los mecanismos de medida del rendimiento se dividen en dos grupos: métodos de muestreo o métodos accionados por evento [150]. En función del formato de almacenamiento de los datos recogidos, podemos diferenciar también dos grandes grupos: los perfiles (*profiles*) y las trazas (*traces*) [46].

Los métodos de muestreo utilizan un reloj o contador externo que indica el instante en el que se registran los valores de las métricas y de los parámetros de rendimiento. En cada uno de los instantes de muestreo, se detiene la ejecución y se registra el contexto de la aplicación en ese instante. El resultado final es un histograma de los diferentes contextos registrados durante la ejecución del programa. Esta aproximación es válida para aplicaciones con un tiempo de ejecución muy grande porque es necesario que el número de medidas sea suficientemente grande para obtener una distribución representativa de la ejecución. La principal ventaja del muestreo es que, al utilizar un mecanismo de temporización externo, no es necesario modificar el ejecutable y, por lo tanto, su intrusismo es mínimo. Sin embargo, esta técnica no proporciona información acerca de las acciones que tienen lugar entre dos medidas consecutivas.

Los métodos accionados por eventos realizan las medidas únicamente en cuando se activan determinados eventos —por ejemplo, en las llamadas a funciones y el retorno de las mismas—, lo que permite obtener un registro con las distintas ocurrencias de un suceso específico. Por lo tanto, en este caso es necesario indicar en qué puntos se deben realizar las correspondientes medidas. El mecanismo más común es mediante la instrumentación de la aplicación, que consiste en la inserción de instrucciones específicas para acceder a las métricas o parámetros de rendimiento en los puntos del código apropiados. Es evidente que la instrumentación conlleva un determinado intrusismo en el comportamiento de la aplicación, y que dependerá del método de instrumentación empleado, por lo que la minimización de este intrusismo es fundamental para obtener unos datos representativos [150]. Entre las técnicas de instrumentación más comunes podemos destacar las siguientes:

– *Código fuente.* Esta técnica inserta código de alto nivel en el código fuente de la aplicación de forma manual o automática [107]. Esta aproximación permite una mayor flexibilidad en la manipulación de eventos, así como en la selección de métricas y parámetros de rendimiento, pero tiene un elevado nivel de intrusismo porque es necesario compilar el código instrumentado.

– *Traductores código a código.* Similar a la técnica de código fuente, pero utilizan directivas (`pragmas`) que se traducen en las correspondientes instrucciones de código fuente durante la ejecución del precompilador [125, 22]. Esta aproximación permite una mejor gestión de la instrumentación y permite la coexistencia de ambas versiones (instrumentada y no instrumentada) en el mismo código fuente, pero también es necesario compilar nuevamente el código instrumentado.

– *Interposición de librería.* Esta técnica permite instrumentar, sin modificar el código fuente, las llamadas que se realizan a una librería, mediante la sustitución de esta por una versión preinstrumentada. El estándar MPI define un mecanismo similar para instrumentar las funciones de esta librería (*Profiling MPI*, PMPI) [128].

– *Dinámicamente.* En este caso, la aplicación se ejecuta directamente en un entorno de ejecución en el que se introduce dinámicamente la instrumentación en los eventos detectados por el entorno de ejecución [29]. Aunque el intrusismo de esta aproximación es mínimo, al no disponer de información de alto nivel, mediante esta técnica es difícil discriminar los eventos que no aportan información relevante.

El perfil de una ejecución consiste en un resumen estadístico del comportamiento, durante la propia ejecución, de las diferentes métricas y parámetros de rendimiento consideradas. Los valores registrados en cada instante de medida son procesados junto con todos los valores registrados con anterioridad para obtener el resumen estadístico parcial correspondiente. El volumen de información almacenado con esta técnica es muy reducido, pero a costa de perder la dimensión temporal en la evolución de cada métrica. Esta técnica es especialmente útil para obtener una visión general del rendimiento y para extraer información de las métricas locales, como las que se derivan de los contadores hardware de rendimiento, por ejemplo.

Las trazas registran la secuencia temporal de los valores de rendimiento en cada instante de medida. De esta forma es sencillo reproducir o visualizar el comportamiento de cada métrica durante la ejecución de la aplicación. Sin embargo, el volumen de los datos almacenados suele ser muy grande —especialmente en sistemas paralelos en los que se registra una traza diferente por cada proceso— por lo que es necesario introducir un mecanismo de filtrado o compresión que haga manejable este volumen de información [61]. En cualquier caso, la visualización de las trazas permite una interpretación visual del comportamiento de la ejecución en un sistema.

Aunque el análisis de los perfiles o las trazas suele realizarse *post mórtem* —es decir, una vez que ha finalizado la ejecución del código instrumentado—, los datos de rendimiento pueden ser evaluados dinámicamente, durante la propia ejecución, usando técnicas de monitorización en tiempo real [46, 144, 130, 33]. Esta aproximación es especialmente útil en aplicaciones con un tiempo de ejecución muy grande porque proporciona la posibilidad de realizar un ajuste dinámico de los parámetros de ejecución en función del análisis dinámico de los datos monitorizados.

1.3 Métodos de análisis de rendimiento

Los métodos de análisis de rendimiento suelen clasificarse en tres categorías diferentes [171, 72]: medida directa, simulación y modelos analíticos. Esta clasificación se basa principalmente en el mecanismo de caracterización del rendimiento. Sin embargo, es habitual encontrar en la bibliografía soluciones híbridas que combinan aspectos asociados a diferentes categorías [13, 68, 5].

1.3.1 Medida directa y *benchmarking*

La ejecución real de una aplicación en un sistema es la mejor referencia para determinar el comportamiento de esa aplicación en ese sistema particular. De hecho, esta aproximación es ampliamente utilizada, durante el desarrollo de aplicaciones, para identificar y corregir los potenciales problemas de rendimiento. El método de medida directa es un componente habitual en herramientas de diagnóstico y suele complementarse con una potente herramienta visualización, generalmente interactiva, que facilite la evaluación subjetiva de los datos [149, 64, 129].

Estrictamente, la aplicabilidad de este método se limita exclusivamente a las condiciones de ejecución utilizadas, ya que no se proporciona ninguna referencia explícita acerca del comportamiento de esa aplicación en otro sistema. Es decir, los datos empíricos únicamente caracterizan el comportamiento presente de un sistema determinado. Sin embargo, desde un punto de vista puramente subjetivo basado en la experiencia y habilidad del analista, es posible extrapolar los datos obtenidos a otros entornos de ejecución.

Un *benchmark* está compuesto por diversas aplicaciones, reales o sintéticas, formando un conjunto representativo de las aplicaciones típicas en un entorno determinado. Los tiempos de ejecución de un *benchmark* ofrecen una caracterización realista del potencial del sistema, proporcionando un mecanismo efectivo de comparación —como, por ejemplo, la lista TOP500 [160]—. Aunque proporcionan una estimación de la eficiencia real de los sistemas, los *benchmarks* no son referencias válidas para predecir el comportamiento de una aplicación determinada, a menos que esté contenida en el propio *benchmark*. Entre los ejemplos más conocidos en entornos paralelos destacan el *benchmark High Performance Linpack* (HPL) [45] —el *benchmark* que se utiliza como referencia para generar la lista TOP500— y la colección *NAS Parallel Benchmarks* (NPB) [15, 165].

1.3.2 Simulación

Un simulador, construido como una aplicación software, es un entorno completo que emula el comportamiento de los diferentes elementos que caracterizan un sistema real. Por lo tanto, las técnicas de simulación proporcionan un entorno controlado del sistema para realizar experimentos sin perturbar el sistema real. Esta técnica está especialmente indicada para desarrollar nuevas arquitecturas, ya que permite estimar el comportamiento de aplicaciones en sistemas que no han sido implementados físicamente.

Por un lado, es preciso que el sistema sea suficientemente fiel al sistema original para que los resultados obtenidos sean realistas. Sin embargo, el entorno de simulación debe ser, a su vez, suficientemente sencillo para evitar un gran consumo de recursos y un elevado coste de desarrollo, así como para obtener tiempos de simulación relativamente pequeños. De hecho, una de las principales desventajas de esta aproximación reside en su elevado coste de evaluación, que hace prohibitivo su uso en ciertas situaciones.

Desde el punto de vista de los datos de entrada, se distinguen dos tipos de simuladores: de código o de trazas [91]. Los simuladores de código (*execution-driven*) utilizan el propio código de la aplicación para gestionar el comportamiento de la simulación, mientras que los simuladores de traza (*trace-driven*) utilizan una traza obtenida a partir de una ejecución real en un determinado sistema. Aunque han demostrado ser muy precisos, el coste computacional de los simuladores de código es inabordable cuando las simulaciones son muy largas y el sistema es complejo. Por lo tanto, los simuladores basados en trazas son más comunes en el contexto de sistemas paralelos [42, 176, 13, 68].

1.3.3 Modelos analíticos

La idea básica de los modelos analíticos consiste en modelar tanto la arquitectura paralela como el algoritmo mediante expresiones analíticas [2, 139, 25]. De esta forma, el programa paralelo puede analizarse y evaluarse, tanto cuantitativa como cualitativamente, de forma independiente a la arquitectura en la que será ejecutado, lo que permite incluso realizar estudios de futuras arquitecturas o sistemas. Gracias a la rápida evaluación de las expresiones analíticas, los modelos analíticos son una herramienta eficaz y flexible, en comparación con otras técnicas de análisis. Sin embargo, para poder describir sistemas reales mediante expresiones analíticas es necesario realizar suposiciones y simplificaciones que, generalmente, merman la precisión de este tipo de aproximación.

1.4 Modelos analíticos de sistemas paralelos

Este tipo de modelos es una herramienta fundamental en el análisis, diseño y evaluación de algoritmos paralelos eficientes porque, desde el punto de vista del desarrollador, los detalles de implementación de los sistemas reales se ocultan mediante el propio modelo. De esta manera, se puede establecer un criterio eficaz para comparar diferentes implementaciones, así como desarrollar algoritmos paralelos genéricos, independientemente de los componentes particulares de los sistemas.

La precisión de los modelos analíticos de sistemas paralelos dependerá de nivel de abstracción que consigamos, es decir, de la complejidad del modelo. El objetivo ideal es un modelo suficientemente detallado para reflejar aspectos realistas de los sistemas (especialmente aquellos que influyan significativamente en el rendimiento) pero, al mismo tiempo, suficientemente abstracto para ser independiente de la máquina y sencillo para que el análisis y la evaluación del modelo sean prácticos.

A continuación se describen brevemente algunos de los modelos analíticos de sistemas paralelos más utilizados.

1.4.1 Modelos PRAM

En su forma más simple, una *máquina paralela de acceso aleatorio* (*Parallel Random Access Machine*, PRAM) consiste en un conjunto de P procesadores con una memoria global compartida que ejecutan el mismo programa de forma síncrona [57]. Puede haber algunas diferencias en cuanto a la definición de un sistema PRAM, pero básicamente consiste en un sistema MIMD (*Multiple Instruction Multiple Data* [56]) donde cada procesador accede a cualquier posición de memoria en una unidad de tiempo, independientemente de donde se encuentre localizada esta. La principal diferencia entre los diferentes tipos de PRAM que podemos encontrar en la bibliografía se vincula a los problemas de contención en los accesos concurrentes de lectura y escritura en la memoria.

A pesar de las grandes diferencias que hay en los computadores paralelos actuales, el modelo PRAM puede ofrecer información útil para el diseño de programas paralelos ya que, aunque este modelo ignora los costes de la red de interconexión, puede proporcionar una estimación del máximo grado de paralelismo que podemos alcanzar en nuestros códigos. Sin embargo, el hecho de ignorar el coste del uso de estos recursos puede dar lugar a una mala utilización de los mismos por lo que el modelo PRAM puede distorsionar los resultados.

1.4.2 Modelo de Hockney

El modelo de Hockney [74] es un modelo muy sencillo que asume que el tiempo para enviar un mensaje de tamaño m entre dos nodos se comporta linealmente, es decir:

$$T = \alpha + \beta m \tag{1.1}$$

En esta ecuación, α, normalmente conocido como latencia, se corresponde con el intervalo de tiempo que transcurre desde que se inicia la comunicación hasta que el proceso receptor recibe el primer byte del mensaje. El parámetro β representa el tiempo de transferencia por byte, que es equivalente al recíproco del ancho de banda del canal de comunicación.

Este modelo define únicamente el tiempo de envío de un mensaje, pero no establece ninguna premisa acerca de las posibles relaciones entre distintos mensajes como, por ejemplo el solapamiento entre mensajes o la congestión de la red.

1.4.3 Modelo BSP

La arquitectura que define el modelo BSP (*Bulk-Synchronous Parallel*) [162] consiste en un conjunto de procesadores con su memoria local que ejecutan una serie de procesos virtuales y una red de interconexión que permite el intercambio de paquetes de tamaño fijo entre ellos. Las computaciones se dividen en *supersteps*, y en cada uno de ellos el procesador realiza operaciones sobre datos locales y envía/recibe paquetes. Un paquete enviado en un determinado *superstep* llegará a su destino en el siguiente *superstep*, y cada uno de estos *supersteps* están separados por una sincronización global de todos los procesadores.

El tiempo de comunicaciones de un algoritmo en el modelo BSP se rige por una función de coste simple, basada en dos parámetros que modelan la máquina paralela:

- el *gap g*, denominado también *permeabilidad*, que refleja el ancho de banda de la red de interconexión, y
- la latencia L, que se define como la duración mínima de un *superstep* y que refleja el coste mínimo para el envío de un paquete a través de la red, así como el sobrecoste de realizar una sincronización global.

Una característica del modelo BSP es que requiere una comunicación global para enviar un simple mensaje de un procesador a otro. A pesar de que parece una restricción fuerte, es necesaria para preservar las optimizaciones globales que se hagan en el código.

1.4.4 Familia de modelos LogP

El modelo LogP [37] ha sido diseñado para crear una abstracción realista, pero al mismo tiempo sencilla, del comportamiento de los sistemas homogéneos formados por máquinas completas conectadas a través de una red de interconexión. Este modelo surge como una propuesta alternativa a los modelos teóricos que se basan en factores irreales como, por ejemplo, un tiempo de retardo nulo o un ancho de banda infinito en la red de interconexión.

Para caracterizar un sistema paralelo, el modelo LogP define cuatro parámetros:

- *Latencia de comunicación* (**L**). Límite superior del retardo de comunicación para transmitir un mensaje de una sola palabra entre dos nodos, en ausencia de conflictos en la red.
- *Overhead* (**o**). Tiempo que un procesador se dedica a tareas de comunicación para enviar o recibir un mensaje, y durante el cual no puede realizar tareas de cómputo.
- *Gap* (**g**). Intervalo de tiempo mínimo entre la transmisión y recepción de dos mensajes consecutivos por el mismo procesador. Su inverso determina el ancho de banda efectivo.
- Número de procesadores (**P**). Número de procesadores que componen el sistema.

Los primeros tres parámetros se expresan en unidades de tiempo y caracterizan el rendimiento de las comunicaciones punto a punto del sistema paralelo. El último parámetro es una burda descripción del poder computacional del sistema.

Esta máquina es asíncrona, es decir, las tareas realizadas entre los diferentes nodos son independientes y no hay ningún tipo de sincronización entre ellas. Además, la red de interconexión de esta máquina tiene un ancho de banda finito, que define el número máximo de mensajes por unidad de tiempo, y sólo contempla el envío de mensajes de tamaño mínimo.

El hecho de que los parámetros del modelo LogP sean independientes del sistema favorece el desarrollo y análisis de algoritmos portables y, al mismo tiempo, evita la necesidad de conocer los detalles específicos de las diferentes máquinas consideradas. Además, el reducido número de parámetros de esta familia de modelos ha sido fundamental para su uso en diversas situaciones [153, 95, 10]. La simplicidad y la precisión del modelo LogP han sido elementos fundamentales para la derivación de otros modelos que cubriesen las carencias detectadas de este modelo [96].

El modelo LogP ha sido diseñado para tamaños de mensaje fijos, de tamaño mínimo, y no contempla tamaños de mensaje variables. En la bibliografía se pueden encontrar diferentes

modelos que extienden el modelo LogP en este sentido. Una de las aproximaciones más inmediatas consiste en sustituir los parámetros de red (**L**, **o** y **g**) por funciones que dependen del tamaño del mensaje [10, 50]. Sin embargo, este tipo de generalización del modelo LogP aumenta considerablemente la complejidad, al mismo tiempo que dificulta su manejabilidad, por lo que la tendencia general ha sido utilizar otras aproximaciones más sencillas y accesibles, como el modelo *Parameterized* LogP (P-LogP) [100] o el modelo LogGP [8].

La contención en la red y la sincronización son otros aspectos, relacionados con el rendimiento del sistema paralelo, que han propiciado el desarrollo de otras extensiones del modelo LogP. Los modelos LoPC [60] y LogGPC [127] son extensiones de LogP y LogGP, respectivamente, que añaden el parámetro C para caracterizar la congestión de la red de interconexión. El modelo LogGPS [87] considera el coste que supone la sincronización que introducen las librerías de comunicación de alto nivel para enviar mensajes de gran tamaño.

En la bibliografía se pueden encontrar otras muchas variaciones del modelo LogP como, por ejemplo, el modelo $\text{Log}_\text{n}\text{P}$ [31], una generalización del modelo LogP que introduce el coste asociado a las diferentes capas existentes en los mecanismos de comunicación punto a punto. También se han desarrollado modelos que consideran sistemas paralelos particulares, como el modelo HLogGP [25], que considera el caso de sistemas heterogéneos, o el modelo LogfP, para mensajes cortos en redes InfiniBand$^{\text{TM}}$ [78].

Modelo LogGP

El modelo LogGP [8] es una extensión del modelo LogP que permite extender el modelo para tamaños de mensaje variables. Este modelo añade un nuevo parámetro (**G**) que caracteriza el coste temporal que supone el envío de un byte para tamaños de mensaje grandes. El inverso de este parámetro cuantifica el ancho de banda asintótico para tamaños de mensaje muy grandes. Por tanto, este modelo se puede interpretar como una generalización del modelo LogP, en la que latencia y *overhead* son constantes mientras que el comportamiento del *gap* está caracterizado por una función lineal:

$$\text{LogGP=}\begin{cases} \text{latencia}(s) = \mathbf{L} \\ \text{overhead}(s) = \mathbf{o} \\ \text{gap}(s) = \mathbf{g} + \mathbf{G} \times \mathit{size}(\mathit{bytes}) \end{cases}$$

La arquitectura que propone el modelo LogGP se ajusta adecuadamente a la mayoría de las redes de interconexión actuales [20]. Asimismo, este modelo también proporciona la

precisión necesaria para realizar un análisis de rendimiento de funciones colectivas de librerías de comunicación de paso de mensajes [85, 139, 79].

1.5 Herramientas de análisis de rendimiento

En esta sección se describen algunas de las herramientas de análisis de rendimiento más representativas en el contexto de sistemas paralelos HPC, clasificadas en función del enfoque de sus diseños.

1.5.1 Herramientas de diagnóstico

El objetivo principal de este tipo de herramientas es realizar un diagnóstico del comportamiento y detectar los posibles problemas de rendimiento —así como las posibilidades de mejorarlo—, a partir de la información obtenida durante la ejecución de las aplicaciones. En este sentido, la visualización de la información es muy importante para que el usuario de las herramientas pueda valorar correctamente los resultados obtenidos.

TAU

El sistema TAU (*Tuning and Analysis Utilities*) es un entorno de análisis de rendimiento de sistemas y aplicaciones paralelos [156, 149]. TAU es un proyecto común desarrollado por el Performance Research Lab (University of Oregon), el LANL Advanced Computing Laboratory y el Research Centre Jülich. Este sistema está formado por un conjunto integrado de herramientas de instrumentación, medida y análisis, que permite obtener perfiles y trazas para realizar un análisis de rendimiento de programas paralelos escritos en Fortran, C, C++, Java o Python.

TAU utiliza la técnica de instrumentación para generar perfiles y trazas, y dispone de diferentes mecanismos de instrumentación. Por ejemplo, la instrumentación puede realizarse a nivel de código fuente —con la herramienta de instrumentación automática PDT (*Program Database Toolkit*) [107] o manualmente, mediante una API de instrumentación— o dinámicamente a nivel de código máquina —utilizando DyninstAPI [29]—. La instrumentación de las funciones MPI se puede realizar mediante el mecanismo estándar PMPI [128], mientras que las directivas OpenMP se instrumentan con el traductor Opari [125]. Por otro lado, el mecanismo de medida de TAU permite seleccionar entre diferentes tipos de métricas temporales e integra interfaces de acceso a los contadores hardware de los microprocesadores [28].

La visualización de los perfiles se realiza con *ParaProf*, una herramienta de visualización integrada en TAU. A partir de los datos de perfil de una aplicación, esta herramienta es capaz de generar gráficos y tablas habituales —como, por ejemplo, perfiles planos (*flat profiles*) o árboles de llamadas (*callpaths*)— pero también es capaz de generar representaciones interactivas en 3D que permiten, por ejemplo, comparar simultáneamente la distribución de los valores de dos métricas en las diferentes funciones y procesos/threads de la ejecución. En cualquier caso, en las gráficas generadas por *ParaProf* es posible acceder directamente a los valores cuantitativos de las métricas consideradas. Por otro lado, TAU proporciona herramientas para transformar las trazas de eventos generadas en los formatos SLOG-2 [34], OTF [102] o EPILOG [126], lo que permite utilizar herramientas de visualización independientes como Paraver [106] o JumpShot [175].

El entorno de TAU incluye la herramienta PerfDMF para la gestión de los perfiles obtenidos por múltiples experimentos [83]. El entorno también integra una herramienta de minería de datos (*PerfExplorer*) que permite realizar un análisis transversal sobre múltiples instancias de experimentos gestionados con PerfDMF, utilizando técnicas de *clustering* y análisis de correlación, entre otros [84].

SCALASCA

SCALASCA [147, 64] es un entorno de código abierto especialmente diseñado para el análisis de rendimiento de sistemas paralelos masivos, aunque también es aplicable a sistemas de menor escala. Este entorno se centra principalmente en aplicaciones paralelas de carácter científico o de ingeniería basadas en las interfaces MPI y OpenMP —incluyendo aproximaciones híbridas—, y escritas en C/C++ o Fortran. Es un proyecto desarrollado por el Jülich Supercomputing Centre del Forschungszentrum Jülich y el Laboratory for Parallel Programming del German Research School for Simulation Sciences.

SCALASCA utiliza la técnica de instrumentación para obtener perfiles o trazas. Además, dispone de un mecanismo de análisis post mortem de las trazas obtenidas. Este análisis permite la identificación de potenciales cuellos de botella del rendimiento —en particular, aquellos relacionados con las comunicaciones y las sincronizaciones—, proporcionando la información necesaria para explorar sus causas. Alternativamente, es posible fusionar las trazas obtenidas y realizar un análisis automático con KOJAK [168]. La visualización de los perfiles y el resultado de análisis de las trazas puede visualizarse en una herramienta interactiva inte-

grada en el entorno. Además, SCALASCA incluye conversores para exportar las trazas a otros formatos compatibles con herramientas de visualización como Paraver [106] o Vampir [129].

SCALASCA y TAU son herramientas muy similares. De hecho, ambos entornos pueden incluso interactuar en varios aspectos. Por ejemplo, SCALASCA es capaz de interpretar la API de instrumentación de TAU. Asimismo, los resultados obtenidos con SCALASCA pueden gestionarse y visualizarse con las herramientas que proporciona TAU.

HPCTOOLKIT

HPCTOOLKIT es una colección de herramientas para medir y analizar el rendimiento de programas en sistemas HPC, desarrollada por la Rice University [81, 1]. El análisis que realiza HPCTOOLKIT se basa en la correlación de las métricas obtenidas dinámicamente durante la ejecución del programa y la estructura del código fuente. Para obtener un análisis independiente del lenguaje, la estructura lógica del programa se obtiene directamente a partir del código binario.

La estructura de HPCTOOLKIT ha sido diseñada para que tanto la medida como el análisis sean escalables. En concreto, este entorno está formado por cuatro componentes principales:

- `hpcrun`. Es el entorno de ejecución de HPCTOOLKIT. Las medidas se realizan dinámicamente en el código binario, y es posible utilizar técnicas de muestreo o de eventos.
- `hpcstruct`. Analiza la estructura del código fuente a partir de la estructura del código binario.
- `hpcprof`. Combina la información obtenida por las herramientas `hpcrun` y `hpstruct` para obtener información detallada del rendimiento de la aplicación.
- `hpcviewer`. Es una herramienta de visualización interactiva para explorar los resultados obtenidos por `hpcprof`. Está diseñada para proporcionar la información de manera jerárquica, dando prioridad a los potenciales problemas de rendimiento.

El análisis que realiza HPCTOOLKIT, correlacionando las métricas de rendimiento y la estructura lógica de la aplicación, permite la creación de una descripción independiente del hardware para la predicción del comportamiento de la aplicación en otras arquitecturas [111].

Vampir — Intel® Trace Analyzer and Collector (ITAC)

Vampir (*Visualization and Analysis of MPI Resources*) es una herramienta comercial ampliamente utilizada, desarrollada por el Center for Applied Mathematics of Research Center Jülich y el Center for High Performance Computing de la Technische Universität Dresden [163, 129]. Durante la ejecución de un programa, la herramienta VampirTrace genera una traza en el formato OTF que se visualiza interactivamente con Vampir. VampirTrace instrumenta automáticamente las funciones MPI mediante la técnica de interposición de librería, pero también permite añadir eventos definidos por el usuario utilizando una API para instrumentación de código fuente.

Intel® Trace Analyzer and Collector es un entorno para la instrumentación y el análisis de códigos paralelos MPI [89]. Este entorno está compuesto por una herramienta de instrumentación (Intel® Trace Collector, ITC) y una herramienta de visualización interactiva (Intel® Trace Analyzer, ITA). Ambas herramientas son el producto de la colaboración de Intel® con los desarrolladores de la herramienta comercial Vampir [164].

Paraver

Paraver es una herramienta flexible para el análisis y visualización del rendimiento de aplicaciones paralelas [106]. Esta herramienta forma parte del entorno CEPBA-Tools, desarrollado en el Barcelona Supercomputing Center - Centro Nacional de Supercomputación [18].

Paraver utiliza un formato de traza flexible, que permite realizar diferentes tipos de análisis. Estas trazas pueden obtenerse a partir del código fuente de aplicaciones paralelas escritas en OpenMP o MPI —e incluso de códigos híbridos—, pero también pueden obtenerse a partir del simulador Dimemas [106].

Además de la representación visual personalizable de los eventos registrados, Paraver ofrece otras posibilidades como, por ejemplo, realizar un análisis cuantitativo de las diferentes métricas consideradas en la traza, la generación de nuevas métricas derivadas o el análisis concurrente de varias trazas. Paraver también proporciona una salida en texto con información muy detallada.

1.5.2 Herramientas de predicción

El objetivo principal de este tipo de herramientas es predecir el comportamiento de una aplicación al ejecutarla en un sistema determinado. Generalmente, estas herramientas utilizan

una caracterización del comportamiento de las aplicaciones paralelas, obtenida a partir de una ejecución real o mediante una inspección manual del código, para simular o evaluar su comportamiento en un determinado sistema paralelo.

Dimemas

Dimemas es un simulador dirigido por eventos para predecir el comportamiento de programas paralelos que utilicen el paradigma de paso de mensajes [106]. Al igual que Paraver, esta herramienta forma parte del entorno CEPBA-Tools, desarrollado en el Barcelona Supercomputing Center - Centro Nacional de Supercomputación [18].

El simulador Dimemas es capaz de reconstruir el comportamiento temporal de la aplicación paralela a partir de un fichero de traza que contiene datos de rendimiento de una ejecución real. Estas trazas contienen tres tipos de elementos:

- Comunicación: información acerca de las características de los mensajes.
- Eventos: cualquier tipo de evento relevante para el rendimiento de la aplicación (por ejemplo, el valor de los contadores hardware) o que informe acerca de su estructura (por ejemplo, inicio/fin de una función).
- Consumo: tiempo entre dos registros consecutivos.

Los eventos registrados en la traza son reproducidos en una máquina virtual, descrita por un conjunto de parámetros arquitecturales y de rendimiento. En particular, la arquitectura de Dimemas se corresponde con una red de multiprocesadores SMP. La red de interconexión está formada por B buses, que determinan el número máximo de mensajes simultáneos, y L enlaces con cada nodo SMP. Las diferentes configuraciones de parámetros permiten la simulación de diferentes tipo de arquitecturas como, por ejemplo, redes de estaciones de trabajo, sistemas SMP, computadores paralelos de memoria distribuida, e incluso sistemas heterogéneos. El coste de la comunicación, tanto de mensajes punto a punto como de comunicaciones colectivas, se calcula mediante modelos lineales, aunque se contemplan efectos no lineales como conflictos de red. El resultado de la simulación es una nueva traza que describe el comportamiento de la aplicación sobre la plataforma simulada, y que puede ser visualizada y analizada en las herramientas Paraver [106] o Vampir [163].

FASE

FASE (*Fast and Accurate Simulation Environment*) es un entorno de simulación diseñado en la University of Florida, para obtener una predicción precisa del tiempo de ejecución sin dilatar excesivamente el tiempo de simulación [68].

Este entorno se basa en la separación del problema en dos dominios diferentes: simulación y aplicación. En el dominio de la simulación, el sistema virtual se construye mediante la conexión de modelos abstractos de los componentes individuales. En el dominio de la aplicación, la información proporcionada por perfiles y trazas se utiliza conjuntamente para obtener una caracterización precisa de la aplicación. Esta información es procesada, creando una traza de eventos más compacta, y se genera un estímulo adecuado para realizar la simulación en la máquina virtual definida. Durante la simulación, se monitorizan diferentes aspectos del rendimiento para generar un perfil y una traza —visualizable con Jumpshot [175]— de la ejecución en la máquina virtual. El usuario puede controlar el tiempo de simulación ajustando el nivel de precisión de la monitorización, ya que el coste temporal de la simulación está directamente relacionado con la intensidad de la monitorización.

WARPP

El entorno WARPP (*WARwick Performance Prediction*), desarrollado en la University of Warwick, está diseñado para analizar el rendimiento de aplicaciones MPI en sistemas MPP mediante la simulación de eventos discretos [167, 71, 69]. Este entorno ha sido desarrollado a partir del entorno PACE (*Performance Analysis and Characterization Environment*) [132, 94].

WARPP implementa un proceso automático de caracterización de las aplicaciones que genera una descripción de su comportamiento en función de los eventos interpretables por el simulador. Por otro lado, el entorno también implementa un proceso automático, que se basa en el análisis de la ejecución de *benchmarks* paralelos, para obtener una caracterización del coste de cada evento en el sistema paralelo sobre el que se realizará la simulación. El simulador de eventos de WARPP, escrito en Java, reproduce el flujo de eventos de la aplicación, evaluando el coste de la sucesión de eventos en la arquitectura considerada.

Prophesy

Prophesy es un entorno de análisis que permite obtener automáticamente un modelo analítico de la ejecución de aplicaciones a partir de la información de rendimiento obtenida de

ejecuciones reales [159, 157]. Este entorno está compuesto por tres módulos: instrumentación (*data collection*), bases de datos (*databases*) y análisis (*data analysis*). Prophesy utiliza un portal web para acceder a los datos de rendimiento, a la instrumentación automática y al mecanismo de modelado [173].

El módulo *data collection* se encarga de la instrumentación, a nivel de código fuente, de la aplicación. La instrumentación automática se realiza sobre los principales lazos y funciones del código, pero también es posible insertar directivas manualmente. Los resultados obtenidos tras la ejecución del código instrumentado, así como información de la estructura del código, se almacenan en una base de datos del módulo *databases*. El módulo *data analysis* construye automáticamente un modelo analítico de la aplicación a partir de los datos almacenados en las bases de datos del módulo *databases*. Además de los datos de rendimiento, se utiliza una base de datos de plantillas de modelos y una base de datos de sistemas.

Prophesy proporciona diferentes métodos de modelado, como el ajuste por mínimos cuadrados y la parametrización mediante una inspección manual. Además, incorpora un nuevo mecanismo de modelado basado en el acoplamiento de *kernels* computacionales, de tal forma que el modelo se construye como una combinación de *kernels* independientes [65, 172].

Capítulo 2

Entorno para el análisis y la predicción del rendimiento

En este capítulo se describe el entorno TIA (*Tools for Instrumentation and Analysis*), un entorno de análisis diseñado para obtener de forma sencilla modelos analíticos precisos del rendimiento de aplicaciones paralelas. En primer lugar, se discutirán las motivaciones que han propiciado el desarrollo de este entorno y se introducirá el sistema CALL [22], el entorno de análisis del rendimiento a partir del cual se ha desarrollado TIA. A continuación, se describirá detalladamente la estructura del entorno TIA y los diferentes componentes que lo conforman.

2.1 Motivaciones

Actualmente existe un gran desarrollo de las tecnologías y metodologías de análisis del rendimiento de aplicaciones en sistemas paralelos, motivado especialmente por la necesidad de explotar eficazmente los costosos sistemas paralelos de los actuales entornos HPC. En estos entornos, el elevado número de factores que pueden afectar al rendimiento dificulta especialmente el proceso de análisis. En función del enfoque del problema, se distinguen dos tipos de análisis del rendimiento complementarios: diagnóstico y predicción.

Por un lado, es necesario caracterizar la ejecución de las aplicaciones en los sistemas actuales para comprender su comportamiento y explorar las posibilidades de mejora, tanto desde el punto de vista del código como de los componentes del sistema. En este sentido, existe un gran número de herramientas de diagnóstico para obtener la información de rendimiento a partir de ejecuciones reales [28, 107, 29, 125], así como herramientas que permiten

procesar y visualizar esta información de un modo eficaz [106, 129, 175]. La previsión de crecimiento de los sistemas HPC [48] ha propiciado el desarrollo de entornos de análisis del rendimiento que integren todos los mecanismos necesarios para obtener estas caracterizaciones [149, 64, 1], incluyendo métodos automáticos para la detección de circunstancias excepcionales que limitan el rendimiento de las aplicaciones como, por ejemplo, cuellos de botella o desbalanceos [84, 168, 154]. Sin embargo, esta aproximación únicamente proporciona la caracterización de una ejecución particular en un sistema concreto, dejando en manos del analista experto su adecuada valoración e interpretación, así como la extrapolación de los resultados obtenidos a otros sistemas o aplicaciones.

La predicción del rendimiento de una aplicación pretende una caracterización de las aplicaciones sobre un hardware subyacente, es decir, se construye un modelo de la aplicación. De este modo, es posible reproducir o evaluar su comportamiento para diferentes situaciones, por lo que estos modelos de rendimiento se utilizan como componentes en otras tareas como, por ejemplo, la optimización del código [70], la distribución eficiente de tareas [131, 92] o la verificación del rendimiento de los sistemas paralelos [99]. Además, gracias al nivel de abstracción necesario para su obtención, los modelos de predicción son una herramienta adecuada para desarrollar y explorar nuevas arquitecturas o entornos de ejecución. Fundamentalmente, las dos aproximaciones más utilizadas para obtener modelos de rendimiento son la simulación y los modelos analíticos.

Actualmente, la mayoría de las herramientas se centran en una predicción basada en la simulación, ya que las actuales técnicas de simulación son muy precisas [69, 68, 106]. Sin embargo, la evaluación de estos modelos tiene asociado un tiempo de evaluación muy elevado, ya que es necesario reproducir el comportamiento de la aplicación en el sistema objetivo. Esta característica limita su aplicabilidad en aquellas situaciones en las que el tiempo de evaluación sea un factor crítico o en entornos de ejecución complejos. La firma (*signature*) de aplicaciones paralelas es una aproximación que permite extraer únicamente aquellos aspectos más relevantes del comportamiento de las aplicaciones, obteniendo una caracterización más compacta y favoreciendo una evaluación más ágil [41, 170].

Los modelos analíticos presentan un coste de evaluación muy reducido —en general, el tiempo de evaluación de una expresión matemática—. Sin embargo, el desarrollo de los modelos analíticos en sistemas paralelos es un proceso muy costoso porque no sólo requiere un conocimiento en detalle de la implementación del código, sino que también es necesario tener presente el sistema de ejecución. En otras palabras, es necesario conocer los detalles del mecanismo de mapeo del código en el hardware subyacente. La inspección manual del

algoritmo en que se basa la aplicación y de su implementación es el mecanismo más inmediato para construir un modelo analítico de aplicaciones paralelas [32, 98]. Esta aproximación suele combinarse con un modelo del sistema paralelo como Hockney [9], BSP [67] o LogP [143, 2, 139, 25]. En general, debido al coste excesivo asociado a una caracterización completa, suelen utilizarse aproximaciones —tanto a nivel de código como a nivel de sistema— que simplifiquen el proceso de modelado, con la consecuente pérdida de precisión del modelo. Esta técnica es propensa a errores debido a su carácter manual y, por otro lado, el empleo del mecanismo de inspección manual está restringido a usuarios expertos, capaces de relacionar las características del código fuente con las del sistema paralelo, por lo que los modelos de predicción analíticos están relegados a círculos especializados.

Para poder generalizar el uso de los modelos de predicción es necesario disponer de un mecanismo automático de modelado. En la bibliografía se pueden encontrar diferentes aproximaciones para obtener una metodología de modelado automática, por ejemplo:

– La herramienta Prophesy proporciona un mecanismo para caracterizar automáticamente el comportamiento de una aplicación [157]. Esta herramienta realiza una inspección automática del código fuente que inserta las correspondientes rutinas de instrumentación para obtener información de rendimiento durante la ejecución de la aplicación bajo diferentes condiciones experimentales. A partir de estos datos, se realiza un proceso automático que permite obtener una expresión analítica mediante diferentes técnicas. En cualquier caso, la automatización del mecanismo sólo es aplicable a programas con un patrón de comportamiento sencillo, por lo que en la mayoría de las situaciones es necesaria la intervención del usuario.

– S. Verboven et ál. [166] obtienen modelos de experimentos paramétricos en función del comportamiento estadístico de los parámetros de la aplicación, a partir de observaciones pasadas.

– S. R. Alam y J. F. Vetter [5] proponen un mecanismo de anotación del código (*model assertions*). Mediante estas anotaciones el usuario asocia un segmento del código con una expresión analítica del rendimiento, utilizando únicamente parámetros de ejecución del código. Por lo tanto, los modelos generados con esta técnica reflejarán la estructura del código de la aplicación. La ejecución del código anotado genera una traza que permite un análisis post mortem para validar y analizar el modelo. Esta aproximación

exige al usuario una propuesta de modelo (en general, no inmediata) que relacione los parámetros del código con el rendimiento de los segmentos anotados.

– El sistema CALL [22], desarrollado en la Universidad de La Laguna, utiliza un mecanismo manual de anotaciones, en las que el usuario introduce una expresión analítica como propuesta de modelo de una sección determinada del código fuente. A partir de estas anotaciones, se genera automáticamente un código instrumentado que permite obtener los valores de los parámetros implicados en la expresión del modelo durante la ejecución de la aplicación. Los datos obtenidos a partir de esta instrumentación se utilizan para obtener los coeficientes del modelo a través de un estudio estadístico.

En esta tesis se presenta el entorno TIA (*Tools for Instrumentation and Analysis*), un entorno de análisis que define una metodología de modelado de aplicaciones en sistemas paralelos [115, 116, 118]. Esta metodología realiza una instrumentación manual del código fuente que permite obtener los valores de determinadas métricas y parámetros de rendimiento durante la ejecución de la aplicación en sistemas reales. A partir de estos datos de rendimiento, se realiza un análisis post mortem y se construye un modelo analítico utilizando técnicas estadísticas y de selección de modelos. Por lo tanto, el entorno TIA está diseñado para obtener un modelo analítico preciso del comportamiento de una aplicación en un determinado sistema, de un modo prácticamente automático y sin que sea necesario conocer los detalles de implementación del código fuente, de los compiladores o de los algoritmos de comunicación. Además, la estructura de este entorno es suficientemente flexible como para realizar otro tipo de análisis estadístico de datos de rendimiento como, por ejemplo, la obtención de los coeficientes de los modelos de la familia LogP.

CALL ofrece ciertas funcionalidades útiles para el entorno TIA, por lo que se ha utilizado como punto de partida en su desarrollo. Sin embargo, existen diferencias fundamentales entre las metodologías definidas por ambos entornos. Por un lado, en TIA no es necesario indicar una propuesta de modelo y únicamente se le exige al usuario que determine cuáles son las métricas y los parámetros que, a priori, deberían ser incluidos en la expresión del modelo. Como consecuencia, el procesamiento de los datos obtenidos tras la ejecución del código es completamente diferente en ambos entorno, ya que en el entorno TIA son necesarios mecanismos de obtención de modelos analíticos que, a partir de los valores medidos durante la ejecución del código instrumentado, proporcionan un modelo adecuado para la predicción del rendimiento.

2.1.1 El sistema CALL

CALL es un entorno de predicción del rendimiento de aplicaciones basado en la integración de modelos analíticos dentro del propio *profiling* de una aplicación [22]. Este entorno está compuesto por diversas herramientas de análisis del rendimiento que permiten una adecuada gestión de los datos de rendimiento. En concreto, el sistema CALL está formado por un traductor código a código (`call`), un conjunto de funciones de instrumentación y una librería de análisis estadístico (`llac`). De este modo, CALL es una herramienta de *profiling* y, al mismo tiempo, también es una herramienta de modelado que permite al analista incorporar expresiones analíticas durante el proceso de instrumentación. Por lo tanto, comprende todas las tareas necesarias para la obtención de una predicción adecuada del comportamiento de una aplicación. CALL puede utilizarse tanto en sistemas secuenciales como para el análisis de diferentes lenguajes de programación paralelos como, por ejemplo, OpenMP o MPI.

El proceso de instrumentación de CALL utiliza una aproximación de *profiling* mediante el uso de *pragmas* en el código fuente de la aplicación. En realidad, los *pragmas* son una interfaz sencilla y directa con las indicaciones necesarias para que el traductor de código `call` inserte, en los puntos indicados por el usuario, las funciones de instrumentación adecuadas. Por lo tanto, además de delimitar la sección del código sobre la que se desea hacer el *profile*, el usuario debe indicar explícitamente una propuesta de modelo analítico del comportamiento de un observable en la sección considerada. Esta expresión deberá estar formada por variables y parámetros monitorizables durante la ejecución de la aplicación. La integración del modelo analítico durante la instrumentación permite automatizar ciertos aspectos en la posterior gestión del proceso de análisis estadístico. También simplifica el proceso de instrumentación, ya que `call` inserta automáticamente las instrucciones adecuadas para medir, durante la ejecución del código, los valores de los parámetros y métricas de rendimiento implicados en la expresión analítica. Aunque el observable más habitual suele ser el tiempo de ejecución, CALL permite utilizar cualquier otra magnitud que sea medible como, por ejemplo, número de fallos cache, accesos a memoria u operaciones en punto flotante.

En principio, es posible utilizar cualquier parámetro o variable en el modelo analítico, siempre que sea una magnitud medible y exista el mecanismo de instrumentación (*driver*) adecuado en CALL. Los *drivers* de CALL pueden dividirse en dos grandes grupos: de observables y de comunicaciones. Los *drivers* de observables son los responsables de acceder a los valores de las variables o parámetros durante la ejecución del código instrumentado. Los parámetros algorítmicos del programa, que se correspondan con variables del código, se

pueden utilizar en la descripción del modelo sin ningún tipo de restricción, ya que existe un *driver* intrínseco en CALL que permite la medida de cualquier variable del código. La utilización de parámetros dependientes de la arquitectura depende de la capacidad de CALL para acceder, a través de un *driver*, al valor de ese parámetro durante la ejecución de la aplicación. En general, el acceso a magnitudes dependientes de la arquitectura se realiza a través de librerías portables de medida del rendimiento como, por ejemplo, PAPI [28]. Por otra parte, los *drivers* de comunicaciones gestionan los datos obtenidos en entornos paralelos por los *drivers* de observables, garantizando la correcta escritura de los datos en los correspondientes ficheros de *profile*. Además, este tipo de *drivers* proporcionan algunos parámetros específicos de estos entornos como, por ejemplo, el número de procesos y el identificador del proceso en un entorno MPI.

La librería `llac` ofrece una interfaz sencilla especialmente diseñada para manejar los datos generados por `call` y realizar el correspondiente análisis. Esta librería está formada por una colección de funciones y estructuras desarrolladas dentro del entorno estadístico R [140]. Sus principales funciones permiten importar a un formato específico los datos de rendimiento y calcular los coeficientes del modelo analítico en función de los datos experimentales recogidos durante la ejecución del código instrumentado. Además, también se contemplan mecanismos para evaluar la validez y calidad del modelo. Adicionalmente se incorporan diversas funciones que automatizan la generación de gráficos específicos con los resultados del análisis.

La figura 2.1 representa un esquema del funcionamiento del sistema CALL [66]. El proceso de instrumentación del código se inicia mediante la inclusión de los correspondientes *pragmas* de CALL al código fuente de la aplicación (src.code, en la figura 2.1). Los *pragmas* de CALL son ignorados en una compilación normal, permitiendo que la versión instrumentada esté contenida en el propio código fuente original. A partir del código fuente que contiene los *pragmas* de CALL, la herramienta `call` produce un nuevo código fuente que contiene las funciones de instrumentación integradas (src.cll.code en la figura), así como las estructuras de datos necesarias para la instrumentación (src.cll.h). Al compilar el código fuente instrumentado, se obtiene un nuevo ejecutable (src.cll.exe) que genera, además de la salida habitual (src.output), un fichero con los datos de rendimiento obtenidos durante su ejecución. Para garantizar un tratamiento estadístico adecuado, es necesario disponer de múltiples ejecuciones del código instrumentado bajo diferentes condiciones experimentales y/o modificando parámetros de la aplicación. Los datos generados en cada ejecución pueden combinarse en un

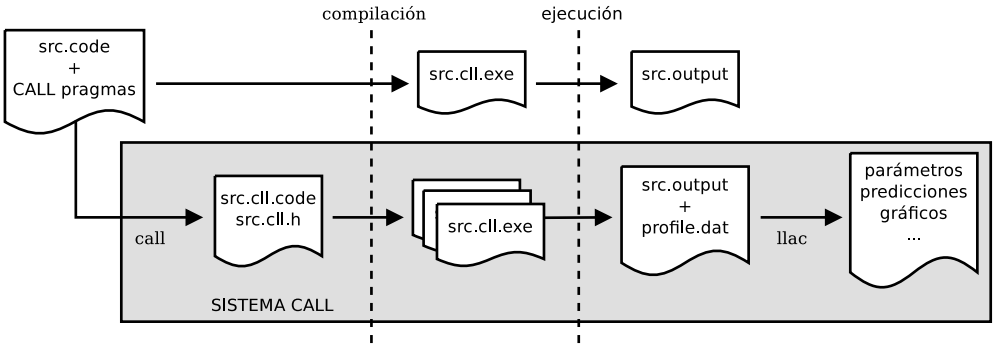


Figura 2.1: Flujo de compilación y ejecución del entorno CALL y comparación con el flujo de compilación y ejecución normal

único fichero (profile.dat) para ser analizados mediante las correspondientes funciones de la librería `llac`.

2.2 El entorno TIA

El entorno TIA (*Tools for Instrumentation and Analysis*) es un sistema completo de análisis del rendimiento, centrado en la predicción del rendimiento de aplicaciones MPI [115, 116, 118]. Este entorno integra las distintas etapas y funcionalidades necesarias para obtener un modelo analítico partiendo del código fuente. TIA es un entorno altamente flexible y escalable debido, en gran medida, a las herramientas sobre las que se ha desarrollado el entorno. Por un lado, `call` [22] es una herramienta de instrumentación que permite la incorporación de nuevas capacidades y funcionalidades de un modo sencillo gracias a su diseño modular (véase la sección 2.1.1). Por otro lado, el entorno estadístico R [140] es un conocido lenguaje estadístico altamente flexible, que permite la gestión de nuevas librerías y funciones de un modo sencillo. Además, los diferentes componentes de TIA están perfectamente definidos y desacoplados, por lo que es posible sustituir la implementación de cualquiera de ellos sin afectar el flujo de modelado definido por el entorno.

Aunque el desarrollo de este entorno está basado en el sistema CALL, existen dos diferencias fundamentales entre ambos entornos. En primer lugar, mientras que en el sistema CALL el usuario debe proporcionar un modelo analítico durante el proceso de instrumentación, en el entorno TIA la instrumentación del código fuente y el análisis de los datos de

profile están completamente desacoplados. Esta característica dota al entorno TIA de una mayor flexibilidad durante el proceso de análisis. Además, el propio entorno es más versátil ya que los procesos de instrumentación y análisis pueden utilizarse de manera independiente en otros contextos. La interfaz entre ambos procesos se implementa mediante ficheros XML. La segunda diferencia entre los dos entornos reside en el proceso de análisis de los datos de rendimiento. En el entorno TIA, y como consecuencia del desacoplamiento de los procesos, ha sido necesario desarrollar una nueva librería de análisis. Además de adaptar los mecanismos existentes en la librería `llac`, se han introducido nuevas funcionalidades que permiten un análisis más potente. En este sentido, la principal diferencia entre ambos enfoques reside en la capacidad del entorno TIA para realizar, mediante técnicas de selección de modelos, un análisis simultáneo de varios modelos, mientras que en el sistema CALL el análisis de los datos de *profile* está limitado a un único modelo.

2.2.1 Estructura

El entorno TIA está formado por dos fases que se corresponden con los procesos de instrumentación del código fuente y de análisis de los datos de *profile*. La figura 2.2 muestra un esquema simplificado del entorno. En la primera fase (fase de instrumentación) se realiza la instrumentación del código fuente de la aplicación y se almacenan en ficheros XML los datos generados, obtenidos tras múltiples ejecuciones del código instrumentado. Esta información se utiliza en la segunda fase (fase de análisis) para obtener un modelo analítico de la aplicación mediante un análisis estadístico. La herramienta `call` es el eje fundamental de la primera fase, mientras que la fase de análisis está integrada dentro del entorno estadístico R.

El diseño de este entorno está pensado para que el usuario que lo desee no sea un mero observador e interaccione con los procesos de instrumentación y modelado, proporcionando su conocimiento y experiencia. En particular, es responsabilidad del usuario instrumentar adecuadamente el código fuente y proporcionar la información necesaria para iniciar el proceso de análisis. En este sentido, TIA proporciona un entorno sencillo, potente y flexible, en el que un usuario experto puede controlar la precisión de la instrumentación así como la calidad del análisis. Sin embargo, las responsabilidades del usuario están perfectamente delimitadas, lo que permite que el resto de las acciones del flujo de modelado puedan ser completamente automáticas y transparentes.

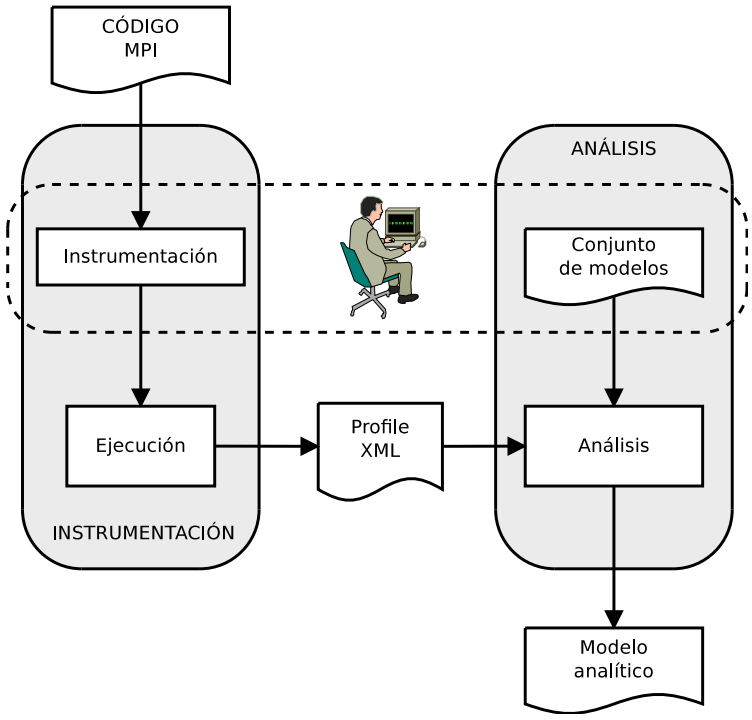


Figura 2.2: Esquema del entorno TIA

2.3 Fase de instrumentación

La fase de instrumentación integra todos los procesos necesarios para generar datos de *profile* adecuados para obtener modelos analíticos de las aplicaciones objeto de estudio. En concreto, esta fase comprende los procesos de instrumentación del código y el almacenamiento de los datos de *profile* recogidos durante la ejecución en los correspondientes ficheros. La implementación de esta fase se basa en el sistema de instrumentación y entorno de ejecución del sistema CALL, en concreto, la herramienta `call` y el conjunto de funciones de instrumentación. Estas herramientas han sido modificadas para manejar las nuevas características del entorno TIA. De este modo, al igual que en el sistema CALL, se proporciona al usuario un método sencillo y directo para obtener datos de diferentes valores de rendimiento durante la ejecución de una aplicación.

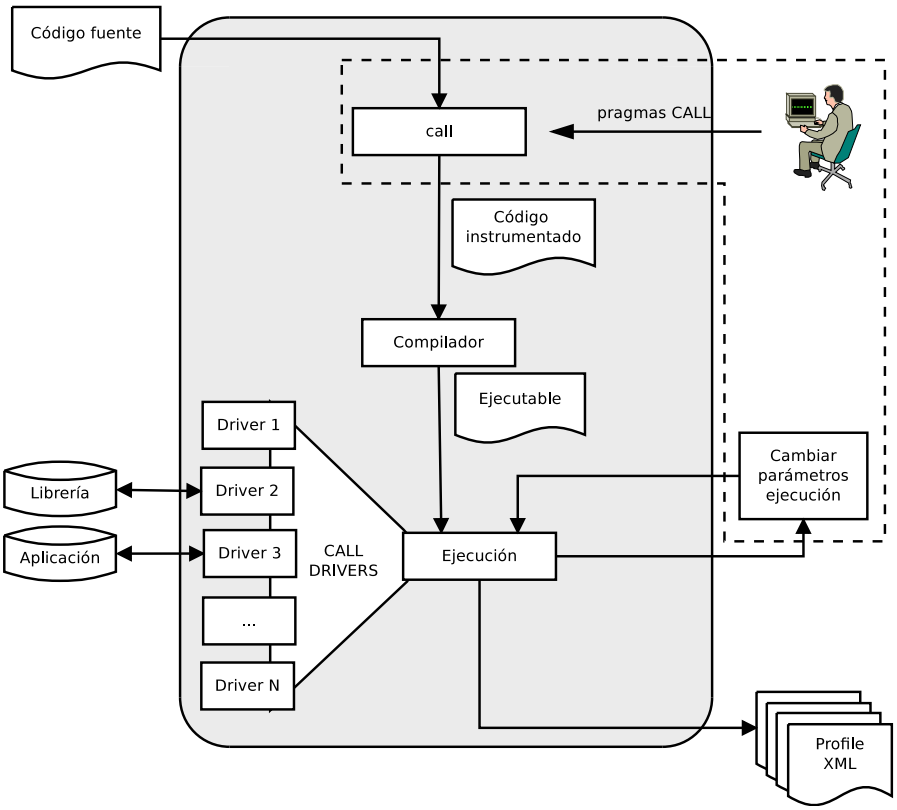


Figura 2.3: Esquema de la fase de instrumentación del entorno TIA

El mecanismo de medida de cada variable depende de su propia naturaleza y, en algunos casos, puede implicar el acceso a librerías y herramientas de medida o monitorización externas. Por lo tanto, la instrumentación debe comportarse de manera diferente en función del tipo de métrica o parámetro: variables del código, contadores hardware, parámetros del entorno de ejecución,... En concreto, las funciones de instrumentación de la herramienta `call` están organizadas en módulos, denominados *drivers*, con una funcionalidad perfectamente delimitada: cada *driver* se encarga de las medidas relacionadas con un tipo de métrica determinada o de la gestión de estos datos en un entorno de ejecución paralelo.

La figura 2.3 muestra un esquema con los distintos componentes de esta fase. El primer proceso de la fase de instrumentación consiste en el marcado del código a través de directi-

vas, es decir, la inserción de los correspondientes *pragmas* de `call` en el código fuente de la aplicación. En particular, es necesario establecer los límites del experimento CALL, es decir, el segmento de código fuente que se desea estudiar. Los límites de un experimento CALL se establecen con las directivas `cll` y `cll end`. A diferencia de lo que sucede en el sistema CALL, en el que es necesario indicar explícitamente la expresión analítica del modelo, la directiva `cll` únicamente necesita como argumento aquellas variables, del código o del entorno de ejecución, que deban ser consideradas en el posterior análisis de los datos de *profile*. Por otro lado, en caso de utilizar un *driver* de comunicación distinto del secuencial, es necesario indicarlo explícitamente mediante la directiva `parallel`. La directiva `report` también es necesaria para indicar el momento de la ejecución en el que se guardarán, en los ficheros XML, los datos de *profile* recogidos.

La figura 2.4 muestra un esquema del marcado básico de un experimento CALL en un código MPI. La línea 3 indica el uso del correspondiente *driver* de comunicación. Las líneas 10 y 13 delimitan el experimento CALL, denominado `exp_name`. En particular, en la línea 10 se indica que, a priori, el observable `CLOCK` es la magnitud elegida para caracterizar el rendimiento. Esta variable es un *comodín* de `call` que se corresponde con el tiempo real de ejecución del experimento, siendo el método de medida dependiente del sistema —por defecto, en un entorno Unix, su valor se obtiene a través de la función `gettimeofday`—. En esta línea también se indica que los parámetros `var1` y `var2` formarán parte, a priori, de la expresión analítica de nuestro modelo y, por lo tanto, su valor deberá registrarse en los correspondientes ficheros XML. Esta expresión no refleja en absoluto la estructura del modelo de rendimiento, sino que simplemente indica, además del inicio del experimento CALL, cuales son las métricas y los parámetros involucrados en la instrumentación. La escritura de los valores medidos en los ficheros XML se realizará en el punto indicado por la directiva de la línea 15.

La instrumentación manual del código permite una mayor flexibilidad al usuario. Por un lado, los experimentos CALL no tienen por qué coincidir estrictamente con una función o un bloque básico, como sucede en las herramientas de instrumentación automáticas; la única restricción es que cualquier función o bloque básico que se desee considerar tiene que estar completamente contenido dentro del experimento. Además, la instrumentación manual permite la coexistencia, dentro de un mismo código, de diferentes experimentos con diferentes métricas y parámetros asociados, permitiendo una instrumentación personalizada e independiente de diferentes aspectos o componentes de una aplicación.

```
#include "mpi.h"
...
#pragma cll parallel MPI
...
int main (int argc, char *argv[])
{
  ...
  MPI_Init(argc, argv);
  ...
#pragma cll exp_name CLOCK=exp_name[0]*var1*var2
  /* Experimento CALL */
  ...
#pragma cll end exp_name
  ...
#pragma report exp_name
  ...
  MPI_Finalize();
  ...
}
```

Figura 2.4: Esquema del marcado básico de un experimento CALL en un código MPI

La herramienta `call` convierte los *pragmas* en las instrucciones y funciones adecuadas para realizar una correcta monitorización durante la ejecución de la aplicación acorde con las indicaciones del usuario, dando lugar al código instrumentado (figura 2.3). En particular, `call` genera dos nuevos ficheros a partir del código instrumentado. Uno de los ficheros contiene el código fuente original con las funciones de monitorización insertadas en los lugares indicados y el otro contiene las estructuras de datos que necesitan estas funciones de monitorización.

La compilación de este nuevo código genera un ejecutable de la aplicación con la instrumentación integrada para obtener, durante su ejecución, los datos de *profile* solicitados por el usuario, en los puntos del código indicados, y almacenarlos en los correspondientes ficheros XML. En general, será necesario realizar múltiples ejecuciones de la aplicación, bajo diferentes condiciones experimentales, para obtener un conjunto de valores que permitan realizar un adecuado análisis estadístico. Cada ejecución producirá un nuevo fichero XML con los datos de *profile* correspondientes. El tratamiento de los valores obtenidos en las múltiples ejecuciones se realizará posteriormente en la fase de análisis, que se tratará en la sección 2.5.

2.3.1 Drivers del entorno TIA

Una de las principales características de la herramienta `call` es la estructura modular de los *drivers*. Este diseño permite abordar de manera independiente la medida y gestión de diferentes parámetros de ejecución, aunque sean de distinta naturaleza, proporcionando al usuario una interfaz única. También facilita la integración de herramientas de monitorización externas, ya que los *drivers* se pueden diseñar como una interfaz que interaccione con la API o la librería correspondiente.

Al igual que en el sistema CALL, existen dos tipos fundamentales de *drivers*: de observables y de comunicaciones. Cada *driver* de observable está asociado a un tipo de métrica determinado y contiene las correspondientes funciones de medida. Los denominados *drivers* de comunicación se encargan de la gestión de los valores medidos (por cualquier *driver* observable) para su correcta escritura en los ficheros de *profile*. Además, los *drivers* de comunicación también pueden aportar funciones de medida de parámetros asociados al entorno de ejecución correspondiente.

El usuario debe indicar explícitamente, mediante la directiva `uses`, la utilización de los diferentes *drivers*. En general, no existe ningún tipo de restricción respecto al número de *drivers* que se pueden utilizar en cada ejecución del código instrumentado. En cualquier caso, debe tenerse en cuenta la influencia de la sobrecarga que introduce cada *driver* utilizado en el tiempo de ejecución de la aplicación, especialmente cuando se utilizan herramientas de monitorización externas.

Además de la adaptación de los *drivers* disponibles en el sistema CALL, se han desarrollado nuevos *drivers* en el entorno TIA, especialmente diseñados para sistemas paralelos en entornos HPC. Los *drivers* Ganglia y NWS son *drivers* de observables que proporcionan un acceso sencillo, desde el propio código de la aplicación, a los parámetros monitorizados por estas herramientas externas en entornos Grid. A continuación se detallan estos *drivers*.

Drivers del sistema CALL

Al estar basados en la misma herramienta de instrumentación, todos los *drivers* desarrollados en el sistema CALL puede utilizarse en el entorno TIA. A continuación se describe un conjunto de *drivers* de CALL que son útiles para TIA.

Aunque el usuario debe indicar explícitamente el uso de un *driver* particular, existe un *driver* intrínseco que no necesita una declaración explícita. Este *driver* permite, al inicio del experimento, acceder al valor de cualquiera de las variables del código fuente. Este *driver*

también es el responsable de establecer la variable `CLOCK` y de su adecuada implementación en los diferentes sistemas.

El *driver* PAPI es un *driver* de observable desarrollado en el sistema CALL que permite obtener información de los contadores hardware de los actuales microprocesadores, a través de la librería portable PAPI [28]. Estos contadores son registros hardware que almacenan el número de veces que se produce un determinado *evento* —suceso de una señal específica relacionada con alguna función del procesador, como los fallos cache, el número de instrucciones en punto flotante o el número de saltos condicionales tomados, entre otros—. Además, este *driver* proporciona un nuevo observable, `PAPI_REAL_USEC`, que proporciona el tiempo de ejecución en microsegundos del experimento CALL, obtenido a partir de los ciclos de reloj del procesador consumidos durante el experimento.

El sistema CALL dispone de diferentes *drivers* de comunicación que permiten instrumentar diferentes entornos paralelos. El *driver* SEQ se emplea en la instrumentación de aplicaciones secuenciales y es el *driver* que se utiliza por defecto, es decir, cuando no se explicita el uso de ningún otro *driver* de comunicaciones. El *driver* PUB_BSP ha sido diseñado para la instrumentación de códigos que utilicen el paradigma de computación BSP [162] mediante la librería Paderborn University BSP [24]. El *driver* de comunicación OpenMP ha sido desarrollado para la instrumentación de aplicaciones que utilicen esta implementación del paradigma de memoria compartida. El *driver* de comunicación MPI ha sido diseñado para proporcionar información del entorno de ejecución MPI —en particular, el número de procesos y el identificador de cada proceso—, así como para la adecuada gestión, utilizando el propio paradigma de paso de mensajes, de los valores obtenidos por otros *drivers* de observables en los diferentes procesos.

Driver Ganglia

Ganglia [123] es una herramienta distribuida y escalable que permite monitorizar sistemas de computación de altas prestaciones tales como clusters o Grids. Utiliza estructuras de datos y algoritmos cuidadosamente diseñados para que la carga en los diferentes nodos sea mínima y para obtener una alta concurrencia. Ganglia es una herramienta robusta que puede utilizarse en un amplio conjunto de sistemas operativos y arquitecturas.

Hemos desarrollado el *driver* Ganglia para acceder al sistema de monitorización de Ganglia desde una aplicación MPI. De esta forma, este *driver* proporciona el mecanismo adecuado para obtener el valor de cualquier parámetro monitorizado por Ganglia durante la ejecución

```
1  #include "mpi.h"
2  ...
3  #pragma cll parallel MPI
4  #pragma cll uses Ganglia
5  ...
6  int main (int argc, char *argv[])
7  {
8    ...
9    MPI_Init(argc, argv);
10 #pragma cll gan_exp GANGLIA_MPI=gan_exp[0]
11   ...
12 #pragma cll end gan_exp
13   MPI_Finalize();
14   ...
15 }
```

Figura 2.5: Esquema de uso del *driver* Ganglia de CALL.

del código instrumentado, de manera transparente al usuario. La figura 2.5 muestra un esquema de uso del *driver* Ganglia en un código MPI. En este caso, el experimento Ganglia comienza y termina en las líneas 10 y 12, respectivamente. Este *driver* debe ser utilizado dentro de un entorno MPI, ya que internamente implementa llamadas a funciones MPI.

El *driver* Ganglia realiza una consulta al sistema Ganglia utilizando el comando `telnet`, en el puerto asignado a Ganglia, al finalizar el experimento CALL. La respuesta de Ganglia es un fichero XML en el que están codificados todos los valores monitorizados por la propia herramienta en ese instante y que el *driver* almacena en un fichero con extensión `.ganglia.xml`. Para cada nodo, la información que proporciona Ganglia incluye diversos parámetros estáticos (velocidad del procesador, número de CPUs del cluster al que pertenece, etc.) y métricas dinámicas (la carga del sistema durante los últimos cinco minutos, el número de bytes entrantes y salientes a través de la red, el porcentaje de uso de CPU utilizada por el usuario, etc.). El formato XML del fichero generado por Ganglia permite que la propia fase de análisis haga la lectura de los valores directamente, evitando una transformación entre formatos XML. De esta manera, no es necesario seleccionar ningún parámetro de Ganglia en las opciones de la orden `cll` porque el filtrado de los datos que se deseen utilizar se realiza en la propia fase de análisis y, al mismo tiempo, se evita una sobrecarga innecesaria durante la ejecución del código instrumentado.

```
1 nameserver serv01
2 memoryserver serv01
3 sensors 3
4 sensor node01
5 sensor node02
6 sensor node03
```

Figura 2.6: Ejemplo de fichero de configuración del *driver* NWS

Driver NWS

La herramienta NWS (*Network Weather Service*) es un sistema distribuido que monitoriza y predice de forma dinámica el rendimiento de recursos computacionales y de red en un intervalo de tiempo determinado [169]. NWS recopila periódicamente información acerca del estado instantáneo del sistema monitorizado a través de un conjunto de sensores de rendimiento. Por ejemplo, es posible monitorizar la latencia y ancho de banda efectivos entre los nodos en los que se está ejecutando una aplicación MPI. La latencia efectiva se define como el tiempo real empleado en transmitir un mensaje TCP de longitud mínima entre dos nodos, mientras que el ancho de banda efectivo es la velocidad real a la cual se transmite un mensaje TCP. Ambos parámetros son dinámicos y dependientes de la carga de la red en cada instante, especialmente en sistemas débilmente acoplados.

El *driver* NWS que hemos desarrollado en TIA [116, 118] proporciona un acceso sencillo a un sistema de monitorización NWS desde aplicaciones MPI. El *driver* supone que existe un sistema NWS independiente que monitoriza la red del sistema en el que se ejecutará el código instrumentado. Un sistema NWS consta de tres elementos diferentes: un servidor de nombres, un servidor de memoria y un determinado número de sensores. Mediante un fichero de configuración, el usuario proporciona la información necesaria para acceder al sistema NWS, indicando el nombre de cada uno de los diferentes componentes del sistema de monitorización. La figura 2.6 muestra un ejemplo del fichero de configuración del *driver* NWS. En este ejemplo, tanto el servidor de nombres como el servidor de memoria están situados en la máquina `serv01` y se monitoriza el estado de la red entre tres máquinas: `node01`, `node02` y `node03`.

El *driver* inicia una actividad NWS, utilizando el protocolo TCP, al comienzo del experimento de CALL, que monitoriza la latencia y ancho de banda efectivo de la red durante el experimento. Durante esta actividad, el sistema NWS registra los valores medidos de laten-

```
1  #include "mpi.h"
2  ...
3  #pragma cll parallel MPI
4  #pragma cll uses NWS
5  ...
6  int main (int argc, char *argv[])
7  {
8    ...
9    MPI_Init(argc,argv);
10 #pragma cll nws_exp NWS_LBW=nws_exp[0]
11   ...
12 #pragma cll end nws_exp
13   MPI_Finalize();
14   ...
15 }
```

Figura 2.7: Esquema de uso del *driver* NWS de CALL

cia y ancho de banda efectivos con una determinada frecuencia. Al finalizar el experimento, el *driver* NWS detiene la actividad y almacena los valores registrados en los ficheros XML correspondientes (con extensión `.nws.xml`). El procesamiento adecuado de estos valores se realiza en la posterior fase de análisis.

Cada experimento CALL que utilice el *driver* NWS realiza dos comunicaciones con el sistema NWS externo, al principio y al final del experimento, para iniciar y detener, respectivamente, la actividad NWS correspondiente. Debido al elevado coste asociado a estas llamadas, el *driver* NWS no debe ser utilizado en el interior de otros experimentos CALL, para evitar que estas llamadas influyan de forma significativa en las medidas de otros *drivers*. Al igual que el *driver* Ganglia, el *driver* NWS debe utilizarse en el interior de un entorno MPI, ya que implementa internamente llamadas MPI. La figura 2.7 muestra un esquema de uso del *driver* NWS en un código MPI. En este caso, el experimento CALL que monitoriza la red (experimento `nws_exp`) comienza y termina en las líneas 10 y 12, respectivamente, por lo que abarca todo el ámbito MPI en este caso particular. La inclusión de otros experimentos CALL debería realizarse en el interior del experimento `nws_exp` (línea 11, en la figura 2.7).

2.4 Interfaz entre fases

Una de las principales diferencias entre el sistema CALL y el entorno TIA reside en el nivel de acoplamiento entre fases. Mientras que en CALL los procesos de instrumentación

y análisis están altamente acoplados, el entorno TIA desacopla totalmente ambos procesos. Esta característica permite una mayor flexibilidad, ya que facilita la modificación de procedimientos en ambas fases —e incluso la utilización de herramientas diferentes— de manera independiente.

La interfaz entre ambas fases se implementa a través de la estructura XML de los ficheros de *profile* generados tras la ejecución del código instrumentado. Estos ficheros —a excepción de los generados por el *driver* Ganglia, que utilizan su propio formato XML— verifican un DTD (*Document Type Definition*) que define un tipo de datos (`cll_data`) específicamente diseñado para datos de *profile* obtenidos durante la ejecución del código instrumentado con CALL (figura 2.8). Cada elemento `cll_data` está formado por diferentes estructuras `cll_experiment` que contienen datos asociados a un experimento CALL concreto (es posible definir más de un experimento CALL dentro del mismo ejecutable). A su vez, cada experimento está formado por tres elementos: máquinas (`machines`), cabeceras (`headers`) y muestras (`sample`). El elemento `machines` enumera los diferentes nodos (`node`) en los cuales se ejecuta cada proceso. El elemento `headers` enumera los diferentes nombres (`h`) de los parámetros o variables medidas. El elemento `sample` contiene una estructura matricial en la que cada fila (`row`) se corresponde con una instancia concreta del experimento CALL correspondiente y el valor de cada columna (`c`) se identifica con los valores de las métricas y los parámetros considerados en esa instancia del experimento CALL. Los atributos de las estructuras `cll_data` y `cll_experiment` han sido definidos para que los ficheros sean compatibles con el sistema CALL.

2.5 Fase de análisis

La fase de análisis [115] realiza el procesamiento de los datos almacenados en los ficheros de *profile* generados durante la fase de instrumentación y ha sido desarrollada en R, un lenguaje y entorno de programación para análisis estadístico y gráfico [140]. En esta fase, los datos obtenidos tras la ejecución del código instrumentado, que han sido almacenados en los correspondientes ficheros XML, son importados dentro de un entorno R, generándose automáticamente las estructuras de datos adecuadas para su posterior análisis.

Se han diseñado una serie de funciones especiales, que han sido recogidas en una librería de R (`cll`) para facilitar la importación y análisis estadístico de los datos de *profile* generados por la ejecución de una aplicación instrumentada con `call`. El objetivo principal de esta librería es la obtención de un modelo analítico del rendimiento de un experimento CALL en

```
<!DOCTYPE cll_data [
  <!ELEMENT cll_data (cll_experiment+)>
    <!ATTLIST cll_data CALL_VERSION CDATA #REQUIRED>
    <!ATTLIST cll_data PROGRAM      CDATA #REQUIRED>
    <!ATTLIST cll_data NODE_NAME    CDATA #IMPLIED>
    <!ATTLIST cll_data SYSNAME      CDATA #REQUIRED>
    <!ATTLIST cll_data RELEASE      CDATA #REQUIRED>
    <!ATTLIST cll_data VERSION      CDATA #REQUIRED>
    <!ATTLIST cll_data PARALLEL_ID  CDATA #REQUIRED>
  <!ELEMENT cll_experiment (machines*,headers,sample)>
    <!ATTLIST cll_experiment EXPERIMENT   CDATA #REQUIRED>
    <!ATTLIST cll_experiment BEGIN_LINE           CDATA #IMPLIED>
    <!ATTLIST cll_experiment END_LINE             CDATA #IMPLIED>
    <!ATTLIST cll_experiment FORMULA              CDATA #IMPLIED>
    <!ATTLIST cll_experiment INFORMULA            CDATA #IMPLIED>
    <!ATTLIST cll_experiment MAXTESTS             CDATA #IMPLIED>
    <!ATTLIST cll_experiment DIMENSION            CDATA #IMPLIED>
    <!ATTLIST cll_experiment PARAMETERS           CDATA #IMPLIED>
    <!ATTLIST cll_experiment NUMIDENTS            CDATA #IMPLIED>
    <!ATTLIST cll_experiment IDENTS               CDATA #IMPLIED>
    <!ATTLIST cll_experiment OBSERVABLES          CDATA #IMPLIED>
    <!ATTLIST cll_experiment COMPONENTS           CDATA #IMPLIED>
    <!ATTLIST cll_experiment POSTFIX_COMPONENT_0  CDATA #IMPLIED>
    <!ATTLIST cll_experiment POSTFIX_COMPONENT_1  CDATA #IMPLIED>
    <!ATTLIST cll_experiment NUMTESTS             CDATA #IMPLIED>
  <!ELEMENT machines (node)+>
  <!ELEMENT node (#PCDATA)>
  <!ELEMENT headers (h)+>
  <!ELEMENT h (#PCDATA)>
  <!ELEMENT sample (row)+>
  <!ELEMENT row (c)+>
  <!ELEMENT c (#PCDATA)>
]>
```

Figura 2.8: DTD del fichero de profile generado por la instrumentación de TIA

función de las métricas y parámetros de rendimiento, obtenidas en la fase de instrumentación. Se ha desarrollado un mecanismo de descripción que permite definir los modelos considerados en esta fase de análisis. Además, en esta librería también se han contemplado otras tareas complementarias como, por ejemplo, la caracterización de la red de interconexión de un sistema paralelo en una ejecución MPI, mediante modelos basados en LogP. En cualquier caso, gracias a la versatilidad del entorno R, es posible realizar cualquier tipo de análisis estadístico con los datos de instrumentación, así como representar gráficamente los resultados obtenidos.

2.5.1 Descripción de modelos

El proceso de obtención del modelo analítico implementado en esta fase necesita que el usuario proporcione un determinado conjunto inicial de modelos. La complejidad del conjunto de modelos dependerá del tipo de análisis, de la experiencia del analista y del nivel de precisión deseado, así como de la complejidad intrínseca del experimento CALL considerado. El tamaño del conjunto de modelos puede ser muy elevado, o puede ser el mínimo —un único modelo—. Por tanto, es necesario un mecanismo de representación que permita al usuario sintetizar en una estructura flexible y escalable un conjunto de modelos.

Se ha diseñado un mecanismo de descripción para representar un conjunto de modelos mediante una estructura de dos niveles. Esta descripción define un modelo global y considera como elementos del conjunto de modelos todos los posibles modelos anidados derivados del modelo global. Los elementos constitutivos del nivel más interno del anidamiento son las diferentes métricas o parámetros que a priori tienen influencia en el comportamiento del experimento CALL. Estas variables, que llamaremos primarias, pueden ser directas —procedentes de la ejecución del experimento CALL instrumentado— o derivadas —cualquier función de una o varias de las variables directas—. El nivel más externo del anidamiento está formado por conjuntos disjuntos de las diferentes variables primarias, denominado `LI` (Lista Inicial). El agrupamiento de las variables primarias permite definir, de manera implícita, una nueva lista de variables derivadas formada por todos los posibles productos de variables primarias pertenecientes a diferentes conjuntos. En esta lista también se considera que cada grupo contiene implícitamente el *elemento identidad*. Llamaremos *términos* a los elementos de esta nueva lista, denominada `LT` (Lista de Términos). El modelo global que define el conjunto de modelos está formado por la suma de todos los términos, con sus correspondientes coeficientes. Para una lista de N términos, el modelo global tendrá la siguiente estructura, que no representa más

que la combinación lineal de todos los términos considerados:

$$M_{\text{global}} = \sum_i C_i T_i \,, \qquad i = 1, 2, \ldots, N$$

donde T_i representa el término i-ésimo y C_i su correspondiente coeficiente de ponderación. En cualquier caso, una vez definidos los diferentes conjuntos de variables primarias, la construcción del modelo global, incluyendo la generación de términos, es automática. En este proceso el usuario únicamente necesita determinar los conjuntos de variables primarias. En un entorno R, esta estructura de conjuntos se puede implementar mediante una lista de listas, en la que cada lista interior contiene las variables correspondientes a cada conjunto.

Para ilustrar el funcionamiento de este mecanismo, mostramos a continuación un ejemplo simple detallado. Supongamos un experimento CALL para el producto matriz-vector paralelo, siendo la matriz cuadrada, en el que la matriz se distribuye por bloques y cada proceso contiene una copia del vector. Durante la fase de instrumentación se determinó que las dos variables primarias relevantes en el rendimiento de este caso particular (y, por tanto, los valores de rendimiento disponibles en la fase de análisis) son el tamaño del vector (n) y el número de procesadores (p). Nótese que este programa no necesita realizar ninguna operación de comunicación. Teniendo en cuenta la distribución de la carga entre los diferentes procesos y obviando el tiempo de reparto de la carga, una posible lista inicial (`LI`) sería:

$$\texttt{LI} = \left\{ n, n^2 \right\}, \left\{ \frac{1}{p} \right\}$$

El agrupamiento de estas variables se ha hecho en dos grupos. El primer grupo caracteriza el volumen de operaciones realizadas, cuyos elementos se corresponden con el número de elementos del vector (n) y de la matriz (n^2). El segundo grupo caracteriza la distribución de la carga computacional, suponiendo un reparto equitativo de la matriz en el que a cada proceso le corresponde calcular la p-ésima parte del vector resultado. La lista de *términos* (`LT`) que determina `LI` es:

$$\texttt{LT} = \left\{ 1, n, n^2, \frac{1}{p}, \frac{n}{p}, \frac{n^2}{p} \right\}$$

Nótese que `LT` está formado por todos los productos posibles entre conjuntos de `LI`. Por lo tanto, el modelo global asociado a esta lista de *términos* es el siguiente:

$$M_{\text{global}} = C_0 + C_1 n + C_2 n^2 + C_3 \frac{1}{p} + C_4 \frac{n}{p} + C_5 \frac{n^2}{p}$$

Este modelo caracteriza el conjunto de 63 ($2^6 - 1$) posibles modelos que se enumeran en la figura 2.9.

1 : $t = C_0$
2 : $t = C_0 n$
3 : $t = C_0 n^2$
4 : $t = C_0 p$
5 : $t = C_0 n/p$
6 : $t = C_0 n^2/p$
7 : $t = C_0 + C_1 n$
8 : $t = C_0 + C_1 n^2$
9 : $t = C_0 + C_1 p$
10 : $t = C_0 + C_1 n/p$
11 : $t = C_0 + C_1 n^2/p$
12 : $t = C_0 n + C_1 n^2$
13 : $t = C_0 n + C_1 p$
14 : $t = C_0 n + C_1 n/p$
15 : $t = C_0 n + C_1 n^2/p$
16 : $t = C_0 n^2 + C_1 p$
17 : $t = C_0 n^2 + C_1 n/p$
18 : $t = C_0 n^2 + C_1 n^2/p$
19 : $t = C_0 p + C_1 n/p$
20 : $t = C_0 p + C_1 n^2/p$
21 : $t = C_0 n/p + C_1 n^2/p$
22 : $t = C_0 + C_1 n + C_2 n^2$
23 : $t = C_0 + C_1 n + C_2 p$
24 : $t = C_0 + C_1 n + C_2 n/p$
25 : $t = C_0 + C_1 n + C_2 n^2/p$
26 : $t = C_0 + C_1 n^2 + C_2 p$
27 : $t = C_0 + C_1 n^2 + C_2 n/p$
28 : $t = C_0 + C_1 n^2 + C_2 n^2/p$
29 : $t = C_0 + C_1 p + C_2 n/p$
30 : $t = C_0 + C_1 p + C_2 n^2/p$
31 : $t = C_0 + C_1 n/p + C_2 n^2/p$
32 : $t = C_0 n + C_1 n^2 + C_2 p$
33 : $t = C_0 n + C_1 n^2 + C_2 n/p$
34 : $t = C_0 n + C_1 n^2 + C_2 n^2/p$
35 : $t = C_0 n + C_1 p + C_2 n/p$
36 : $t = C_0 n + C_1 p + C_2 n^2/p$
37 : $t = C_0 n + C_1 n/p + C_2 n^2/p$
38 : $t = C_0 n^2 + C_1 p + C_2 n/p$
39 : $t = C_0 n^2 + C_1 p + C_2 n^2/p$
40 : $t = C_0 n^2 + C_1 n/p + C_2 n^2/p$
41 : $t = C_0 p + C_1 n/p + C_2 n^2/p$
42 : $t = C_0 + C_1 n + C_2 n^2 + C_3 p$
43 : $t = C_0 + C_1 n + C_2 n^2 + C_3 n/p$
44 : $t = C_0 + C_1 n + C_2 n^2 + C_3 n^2/p$
45 : $t = C_0 + C_1 n + C_2 p + C_3 n/p$
46 : $t = C_0 + C_1 n + C_2 p + C_3 n^2/p$
47 : $t = C_0 + C_1 n + C_2 n/p + C_3 n^2/p$
48 : $t = C_0 + C_1 n^2 + C_2 p + C_3 n/p$
49 : $t = C_0 + C_1 n^2 + C_2 p + C_3 n^2/p$
50 : $t = C_0 + C_1 n^2 + C_2 n/p + C_3 n^2/p$
51 : $t = C_0 + C_1 p + C_2 n/p + C_3 n^2/p$
52 : $t = C_0 n + C_1 n^2 + C_2 p + C_3 n/p$
53 : $t = C_0 n + C_1 n^2 + C_2 p + C_3 n^2/p$
54 : $t = C_0 n + C_1 n^2 + C_2 n/p + C_3 n^2/p$
55 : $t = C_0 n + C_1 p + C_2 n/p + C_3 n^2/p$
56 : $t = C_0 n^2 + C_1 p + C_2 n/p + C_3 n^2/p$
57 : $t = C_0 + C_1 n + C_2 n^2 + C_3 p + C_4 n/p$
58 : $t = C_0 + C_1 n + C_2 n^2 + C_3 p + C_4 n^2/p$
59 : $t = C_0 + C_1 n + C_2 n^2 + C_3 n/p + C_4 n^2/p$
60 : $t = C_0 + C_1 n + C_2 p + C_3 n/p + C_4 n^2/p$
61 : $t = C_0 + C_1 n^2 + C_2 p + C_3 n/p + C_4 n^2/p$
62 : $t = C_0 n + C_1 n^2 + C_2 p + C_3 n/p + C_4 n^2/p$
63 : $t = C_0 + C_1 n + C_2 n^2 + C_3/p + C_4 n/p + C_5 n^2/p$

Figura 2.9: Conjunto de modelos asociado a la lista inicial $\{n, n^2\}, \{\frac{1}{p}\}$

Para complementar este mecanismo, se ha considerado una extensión que permite al usuario contemplar otros *términos* aparte de los generados automáticamente a través de los grupos de variables primarias. Por ejemplo, y siguiendo con el supuesto anterior, supongamos que el experimento se ejecuta en un sistema distribuido y que la distribución de la matriz es externa al experimento (es decir, todos los procesos disponen de su bloque de matriz antes de que comience el experimento), pero el reparto del vector se realiza dentro del propio experimento CALL, mediante una comunicación de tipo *broadcast*. Suponiendo que el mecanismo de *broadcast* utiliza una estructura lineal, una posible lista inicial sería:

$$\texttt{LI} = \{n, n^2\}, \{\frac{1}{p}\}, \{np\}^*$$

donde el símbolo * indica que las variables primarias contenidas en ese grupo deben ser consideradas directamente como *términos* y no se combinan con las demás. Por lo tanto, el modelo global que caracteriza el conjunto de modelos será:

$$M_{\text{global}} = C_0 + C_1 n + C_2 n^2 + C_3 \frac{1}{p} + C_4 \frac{n}{p} + C_5 \frac{n^2}{p} + C_6 np$$

En este caso, el conjunto de modelos estará compuesto por 127 ($2^7 - 1$) modelos.

2.5.2 Estructura de la fase de análisis

La fase de análisis también se ha diseñado de forma modular. En particular, las diferentes funciones se han organizado en cuatro módulos, agrupados a su vez en dos bloques principales denominados «Procesamiento» y «Ajuste». En la figura 2.10 se muestra un esquema de los diferentes módulos que conforman esta fase.

El bloque de Procesamiento comprende las tareas de importación de los datos de *profile* de los ficheros XML correspondientes a un experimento CALL, así como su adecuado procesamiento para obtener una estructura de datos adecuada y posterior tratamiento estadístico. Los módulos asociados a este bloque se denominan IMPORT, SIEVE y JOIN. El bloque de Ajuste tiene como objetivo obtener un modelo analítico del experimento CALL a partir de la estructura de datos generada por el bloque de Procesamiento y el conjunto de modelos proporcionado por el usuario. Este bloque comprende únicamente el módulo FIT.

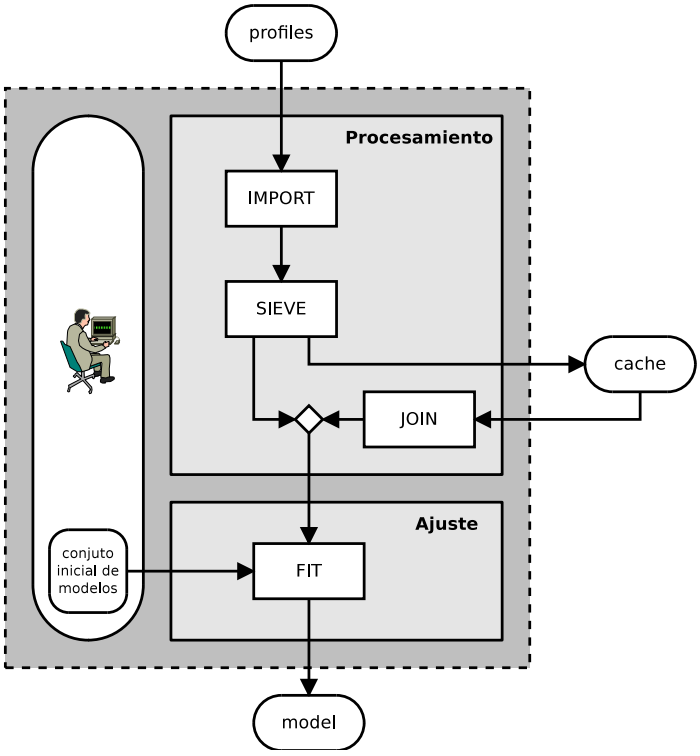


Figura 2.10: Esquema de la fase de análisis del entorno TIA.

Módulo IMPORT

El módulo IMPORT contiene todas las funciones necesarias para la lectura de los ficheros XML con el tipo de datos `cll_data` (véase la sección 2.4) y el almacenamiento de estos valores en estructuras de datos de R.

La función principal de este módulo es `cll.import`. Esta función devuelve una estructura `cll.xml.data`, que contiene todos los datos recogidos en los diferentes ficheros de *profile* considerados. Algunos de los parámetros más relevantes de esta función son:

`program`: es el único argumento obligatorio de esta función y contiene el nombre del ejecutable instrumentado.

path: contiene una lista con los diferentes directorios en los que están almacenados los ficheros XML del ejecutable considerado; por defecto, se considera únicamente el directorio de trabajo actual.

NWS: parámetro booleano que indica si se deben considerar los ficheros generados por el *driver* NWS.

GANGLIA: parámetro booleano que indica si se deben considerar los ficheros generados por el *driver* Ganglia.

gan.metrics: lista con las métricas y los parámetros de Ganglia que serán incluidas en la estructura de datos final.

new.columns: parámetro que permite incluir explícitamente, a partir de las métricas o parámetros disponibles, nuevas variables en la estructura de datos resultante.

Módulo SIEVE

Las funciones del módulo SIEVE realizan un *filtro* que permiten seleccionar los datos más relevantes para realizar el análisis. Por ejemplo, seleccionar únicamente las ejecuciones en las que el número de procesos sea una potencia entera de 2. Además, también integra funciones para eliminar casos espurios (*outliers*). El resultado final de este módulo es una estructura de datos (`cll.data`) que contiene los valores seleccionados para realizar el posterior ajuste de los datos.

La función principal de este módulo es `cll.sieve`. Esta función devuelve una estructura `cll.data`, con los datos seleccionados de la estructura `cll.xml.data` considerada. Algunos de los parámetros más relevantes de esta función se indican a continuación:

data: es el único argumento obligatorio de esta función y contiene el nombre de la estructura `cll.xml.data` con los valores recogidos de los ficheros de *profile*.

clocks: parámetro que indica el nombre del observable del experimento CALL; el resto de métricas y parámetros se considerarán *variables* del experimento; si no se indica ningún nombre explícitamente, la función intentará utilizar por defecto las variables `PAPI_REAL_USEC` y `CLOCK`, en este orden.

select: expresión lógica que establece las condiciones que deben cumplir las variables de cada instancia del experimento CALL para ser incluidas en el análisis posterior.

`outliers:` parámetro lógico que indica si se deben buscar *outliers*; la búsqueda de *outliers* sólo tiene sentido cuando se realizan múltiples medidas del observable para una misma configuración de los factores del experimento.

`factors:` lista con los nombres de aquellas variables que son factores, es decir, parámetros configurables del experimento; facilita la tarea de detección de *outliers*, ya que permite identificar las diferentes situaciones experimentales.

`file:` indica el nombre del fichero en el que se debe almacenar la estructura resultante; el valor por defecto de esta opción es `null`, que indica que la estructura resultante no se guardará en un fichero.

Módulo JOIN

El módulo JOIN contiene funciones que permiten la integración de diferentes instancias de un mismo experimento CALL, y que previamente hayan sido almacenados en una *cache* externa con el formato `data.model`.

La función principal de este módulo es `cll.join`, que únicamente necesita como argumento la lista con los ficheros que se desean combinar en una única estructura de datos. Únicamente se tendrán en consideración las variables comunes a todos los ficheros.

Módulo FIT

El objetivo principal de este módulo es obtener una expresión analítica del experimento CALL. Esta expresión describirá el comportamiento de un observable en función de los parámetros o variables monitorizados durante la ejecución del experimento. Al igual que en el sistema CALL, aunque el observable más utilizado suele ser el tiempo, es posible caracterizar cualquier observable como, por ejemplo, el número de operaciones en punto flotante o el número de fallos de los diferentes niveles de la jerarquía de memoria.

La función principal de este módulo es `cll.fit`. Esta función devuelve una estructura `data.model` que contiene, por un lado, la estructura `cll.data` con los datos utilizados en el proceso de obtención del modelo y, por otro lado, el modelo obtenido en ese proceso en un formato propio de R. Algunos de los parámetros más relevantes de esta función son:

`data.model:` es un parámetro obligatorio que indica el nombre de la estructura `cll.data` que se debe emplear.

`var.list:` es también un parámetro obligatorio que contiene la lista de *términos* que describen el modelo o conjunto de modelos que deben ser considerados.

`type:` indica el modo de procesamiento que debe ser empleado; por defecto utiliza el modo `PROBE`, que realiza un ajuste simple de los datos con el modelo global descrito por el parámetro `var.list`; el otro modo `FULL` realiza, a partir del conjunto de modelos proporcionado, un proceso de selección de modelos basado en el criterio de información de Akaike (este método se describe en detalle en el capítulo 4).

`options:` es una lista que contiene los parámetros necesarios para los diferentes tipos de procesamiento.

2.6 Contribuciones

La contribución más relevante de este capítulo es el diseño e implementación del entorno de análisis del rendimiento para el modelado de aplicaciones paralelas. En concreto, se han diseñado e implementado los *drivers* NWS y Ganglia, la interfaz entre las diferentes fases del entorno, y todas las funciones de la librería de la fase análisis. El trabajo desarrollado en este capítulo ha derivado en la siguiente serie de publicaciones:

- *Modelización del Rendimiento de Aplicaciones MPI en Entornos Grid*, XVII Jornadas de Paralelismo 2006 [118].
- *Analytical Performance Models of Parallel Programs in Clusters*, Parallel Computing: Architectures, Algorithms and Applications, ParCo 2007 [115].
- *Software Tools for Performance Modeling of Parallel Programs*, IEEE International Parallel and Distributed Processing Symposium, IPDPS 2007 [116].

Capítulo 3

Caracterización precisa de las comunicaciones en entornos MPI

En este capítulo se presenta el modelo LoOgGP, una extensión de los modelos LogP y LogGP que permite una caracterización muy precisa del comportamiento real de las comunicaciones en un sistema paralelo. También se presenta un método de caracterización, basado en este nuevo modelo, que se integra completamente dentro del entorno de análisis TIA. En la sección 3.1 se presenta el estado del arte en el cálculo de parámetros de los modelos LogP y LogGP. La propuesta de un nuevo modelo denominado LoOgGP se describe en la sección 3.2, así como un mecanismo para obtener los valores numéricos de los parámetros del modelo. La sección 3.3 describe la implementación en el entorno TIA de la caracterización LoOgGP en entornos paralelos MPI. Además de calcular el valor de los correspondientes parámetros, esta implementación determina automáticamente los diferentes rangos de comportamiento de los actuales entornos MPI, debidos a los diferentes protocolos utilizados en función de la longitud del mensaje.

3.1 Cálculo de los parámetros LogP/LogGP

Una determinación precisa de los parámetros de red de los modelos LogP y LogGP es fundamental para obtener predicciones precisas, permitiendo realizar un adecuado análisis y evaluación, acerca del comportamiento de algoritmos paralelos [49, 75, 139, 80]. En la bibliografía se pueden encontrar diferentes alternativas para calcular los parámetros de los diferentes modelos de la familia LogP. El tiempo de ida y vuelta (*Round Trip Time*, RTT) es un

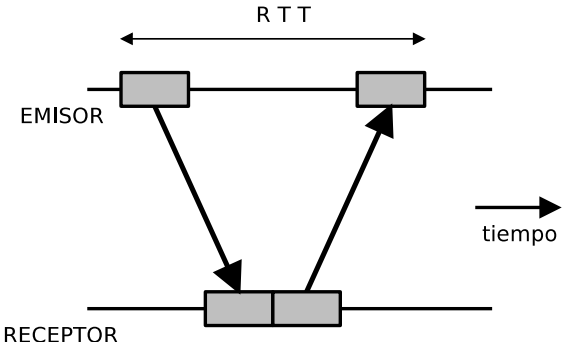


Figura 3.1: Esquema del *microbenchmark* clásico *Ping-Pong*

elemento fundamental común a todas ellas. En una comunicación entre dos procesos, emisor y receptor, el RTT se define como el tiempo que transcurre desde que un mensaje es enviado hasta que ese mismo mensaje regresa al proceso emisor, tras un procesamiento mínimo en el receptor. De esta forma, se soluciona el problema de precisión de muchos sistemas paralelos que deriva de la inexistencia de un reloj que sincronice los nodos con la suficiente precisión para medir el tiempo de envío de mensajes (del orden de microsegundos). El mecanismo más habitual para medir el RTT es el *microbenchmark* clásico *Ping-Pong*. La figura 3.1 muestra un esquema de este *microbenchmark* ; las flechas indican el envío de mensajes a través de la red de interconexión, mientras que las zonas sombreadas simbolizan el tiempo empleado por el correspondiente proceso durante el envío/recepción del mensaje.

Culler et ál. [38] han propuesto un *microbenchmark* que determina los parámetros LogP, diferenciando entre el *overhead* del emisor ($\mathbf{o}_s$) y el *overhead* del receptor ($\mathbf{o}_r$). En este caso, únicamente los parámetros $\mathbf{o}_s$ y $\mathbf{g}$ se calculan directamente. El resto de parámetros dependen de valores obtenidos previamente, con la consecuente propagación de errores.

Utilizando técnicas similares, la propuesta de Kielmann et ál. [101] calcula los parámetros P-LogP. Los valores de los modelos LogP y LogGP se pueden obtener a partir del modelo P-LogP mediante una tabla de equivalencia. Pješivac-Grbovíc et ál. [139] utilizan este método para determinar directamente los parámetros de los modelos P-LogP, LogP y LogGP.

Bell et ál. [20] han desarrollado su propio conjunto de *benchmarks* para calcular los valores de los parámetros correspondientes a una ligera variación del modelo LogGP. Esta variación está basada en el hecho de que el parámetro $\mathbf{L}$ del modelo LogP original no se ajusta al comportamiento real de los sistemas actuales porque existe un solapamiento entre latencia y

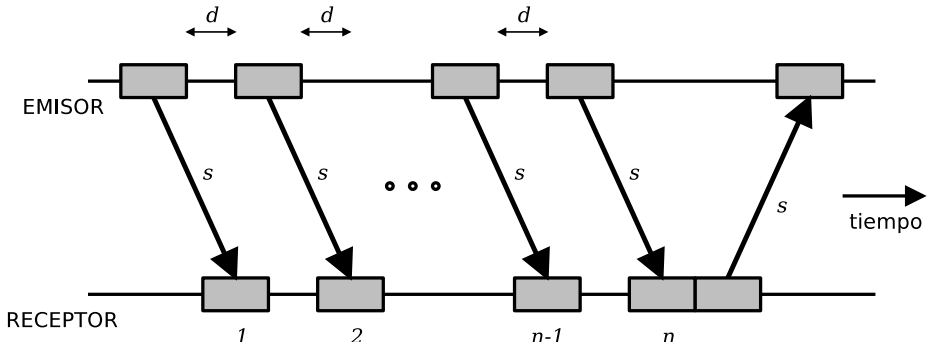


Figura 3.2: Esquema del *microbenchmark* PRTT

overhead. Para solventar este inconveniente, la latencia es sustituida por un nuevo parámetro denominado EEL (*End-to-End Latency*), que se corresponde con el RTT de un mensaje de tamaño mínimo. Al igual que en los modelos anteriores, sólo algunos de los parámetros se obtienen de manera directa.

3.1.1 Método PRTT

Hoefler et ál. [76, 79] han diseñado un método para determinar el valor de cada parámetro LogGP utilizando medidas directas, sin dependencias con otros parámetros. De este modo, se elimina la propagación de errores de primer orden que aparece en las propuestas de Culler et ál. [38], Kielmann et ál. [101] y Bell et ál. [20]. Por otro lado, también se proporciona un mecanismo de detección automática de los cambios de protocolo que los sistemas de comunicación actuales realizan de forma transparente para optimizar la comunicación. Este método de medida ha sido implementado en la herramienta Netgauge [77].

El método PRTT está basado en el *microbenchmark Parameterized Round Trip Time*, puede considerarse una generalización del *Ping-Pong*. Este *microbenchmark* refleja el tiempo necesario para completar el envío de n mensajes consecutivos de tamaño s, con un retardo d entre cada envío, y recibir una réplica del último mensaje. La figura 3.2 muestra un esquema de este *microbenchmark* . El valor correspondiente a una configuración concreta $\{n,d,s\}$ se denota $PRTT(n,d,s)$. Por ejemplo, el *microbenchmark Ping-Pong* para un mensaje de tamaño mínimo se corresponde con la configuración $PRTT(1,0,1)$.

Utilizando diferentes configuraciones del *microbenchmark* PRTT, es posible obtener los parámetros de *gap* y *overhead* del modelo LogGP. Este cálculo se basa en las siguientes expresiones [79]:

$$T_o(s) = \frac{PRTT(n,d(s),s) - PRTT(1,0,s)}{n-1} - d(s) \tag{3.1}$$

$$T_g(s) = \frac{PRTT(n,0,s) - PRTT(1,0,s)}{n-1} \tag{3.2}$$

donde $d(s)$ debe ser mayor o igual que dos veces el valor de $PRTT(1,0,s)$. El parámetro n debe ser suficientemente grande para que se minimicen los efectos debidos al establecimiento de la comunicación. Las variables $T_o(s)$ y $T_g(s)$ son las correspondientes generalizaciones, para un tamaño s arbitrario, de los conceptos *overhead* y *gap* del modelo LogP. En particular, $T_o(s)$ indica el tiempo que un proceso necesita para enviar un mensaje de tamaño s, mientras que $T_g(s)$ se corresponde con el tiempo que el canal de comunicación del emisor está ocupado al enviar un mensaje de tamaño s. Según estas expresiones, el comportamiento de los conceptos *gap* y *overhead* presenta una dependencia con el tamaño de mensaje.

Teniendo en cuenta el significado de las variables $T_o(s)$ y $T_g(s)$, su relación con los parámetros de *overhead* y *gap* del modelo LogGP (es decir, los parámetros **o**, **g** y **G**) es inmediata:

$$T_o(1) = \mathbf{o} \tag{3.3}$$

$$T_g(s) = \mathbf{g} + \mathbf{G} \times (s-1) \tag{3.4}$$

El cálculo de la latencia, tal y como se define en el modelo LogGP, no es inmediato. En consonancia con Bell et ál. [20], Hoefler et ál. consideran la mitad del RTT de un mensaje de tamaño mínimo como aproximación de la latencia. Utilizando el *microbenchmark* PRTT, esta expresión sería:

$$\mathbf{L} = \frac{PRTT(1,0,1)}{2} \tag{3.5}$$

Por lo tanto, para calcular los parámetros LogGP mediante este método, es necesario realizar medidas de las diferentes configuraciones $PRTT(n,d,s)$ que aparecen en las ecuaciones (3.1), (3.2) y (3.5). En particular, para el cálculo de los parámetros **g** y **G** es necesario resolver un sistema lineal, por lo que a priori sólo serían necesarios dos valores experimentales de $T_g(s)$. Sin embargo, Hoefler et ál. obtienen los valores de estos parámetros mediante el ajuste a una función lineal de diferentes valores de $T_g(s)$ porque, de esta manera, es posible detectar, de manera simultánea a la medida de los *microbenchmarks* PRTT, los tamaños de

Tabla 3.1: Entornos de comunicación MPI considerados

Referencia	Red	Implementación MPI
`GB+MPICH`	Gigabit Ethernet	MPICH
`GB+OpenMPI`	Gigabit Ethernet	Open MPI (TCP BTL)
`IB+OpenMPI`	InfiniBand™	Open MPI (OpenIB BTL)
`IB+MVAPICH`	InfiniBand™	MVAPICH

mensaje en los cuales aparece un cambio de protocolo [79]. En particular, la heurística para la detección de intervalos de comportamiento desarrollada por estos autores recorre el intervalo de tamaños de mensaje considerado de manera creciente, comenzando con 1 byte. En cada tamaño de mensaje (s_j) la pendiente del *gap* se recalcula utilizando los tiempos de los *microbenchmarks* PRTT correspondientes al intervalo $[1, s_j]$ y se comparan con la pendiente del intervalo $[1, s_{j-2}]$. Una diferencia cuantitativamente sustancial en ambas pendientes se interpreta como un cambio de comportamiento. Sin embargo, los propios autores advierten que esta aproximación depende fuertemente de la sensibilidad considerada.

3.2 El modelo LoOgGP

El comportamiento de las variables $T_g(s)$ y $T_o(s)$ ha sido medido, utilizando el *microbenchmark* PRTT, en diferentes situaciones experimentales en el cluster `tegasaste`, un cluster homogéneo de 10 nodos Intel® Xeon® a 2,6 GHz y sistema operativo Linux 2.6.16. El cluster `tegasaste` dispone de dos redes de interconexión diferentes (Gigabit Ethernet e InfiniBand™) y tres implementaciones diferentes del estándar MPI (Open MPI, MVAPICH y MPICH). La Tabla 3.1 resume las cuatro situaciones experimentales consideradas, que se diferencian por el tipo de red y la implementación MPI —en la implementación Open MPI se han considerado diferentes componentes BTL (*Byte Transfer Layer*) en función del tipo de red—. Las figuras 3.3, 3.4, 3.6 y 3.5 muestran los valores experimentales de ambas variables en las diferentes situaciones experimentales consideradas. En las figuras se puede identificar un comportamiento lineal por tramos. Aunque en algunos tramos el comportamiento de ambas variables es muy similar, en general, sus pendientes no mantienen relación. En cualquier caso, las pendientes de los diferentes tramos son positivas, en coherencia con el comportamiento esperable, ya que siempre es más costoso enviar un mensaje de mayor tamaño.

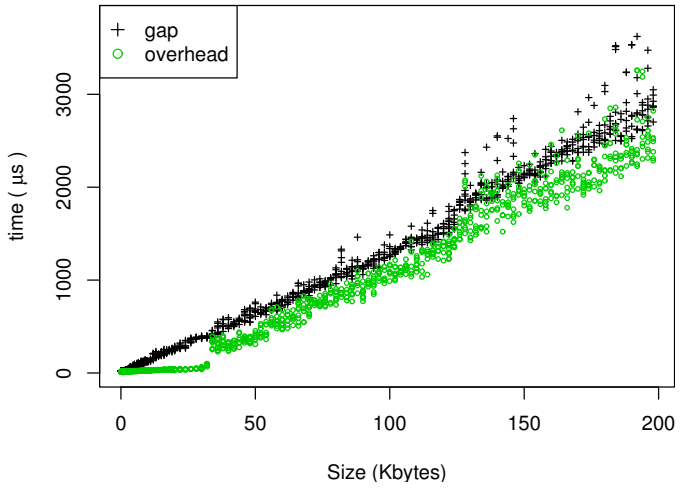


Figura 3.3: Valores medidos de *gap* y *overhead* en el entorno `GB+MPICH`

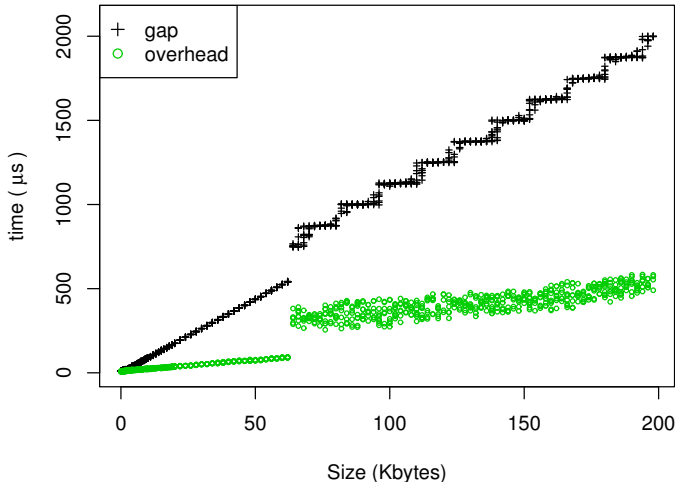


Figura 3.4: Valores medidos de *gap* y *overhead* en el entorno `GB+OpenMPI`

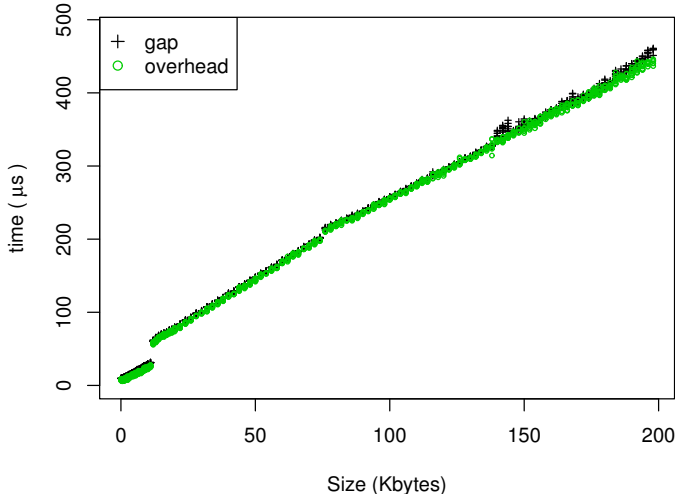


Figura 3.5: Valores medidos de *gap* y *overhead* en el entorno `IB+OpenMPI`

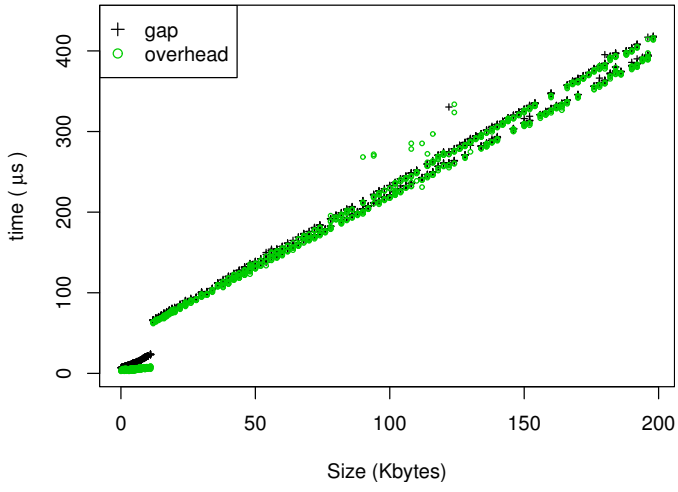


Figura 3.6: Valores medidos de *gap* y *overhead* en el entorno `IB+MVAPICH`

Los modelos LogP y LogGP no contradicen el hecho de que el *overhead* y el *gap* tengan una dependencia lineal con el tamaño de mensaje. En realidad, estos modelos seleccionan las características más relevantes necesarias para una descripción suficientemente precisa del comportamiento del mecanismo de comunicación. Por ejemplo, el modelo original LogP, diseñado específicamente para tamaños de mensaje mínimos, simplemente ignora esta dependencia. Por otra parte, el modelo LogGP considera que efectivamente existe una dependencia con el tamaño del mensaje y que, en el caso del *gap*, una descripción lineal es consistente con el comportamiento real. Pero este modelo no considera que la dependencia del *overhead* con el tamaño de mensaje aporte información relevante para una descripción precisa del mecanismo de comunicación. Sin embargo, en los valores obtenidos experimentalmente se puede observar que la variable $T_o(s)$ también se comporta de manera lineal en diferentes entornos MPI, independientemente del tipo de red, para tamaños de mensaje menores que 200 KB.

Proponemos el modelo LoOgGP para considerar el comportamiento lineal observado experimentalmente tanto en el *gap* como en el *overhead* [117, 119, 122]. De manera análoga al desarrollo del modelo LogGP, este comportamiento lineal se puede describir como una extensión del modelo LogGP, disgregando el *overhead* en dos componentes: una componente intrínseca (**o**) y un *overhead* por byte (**O**), que defina el tiempo necesario para preparar cada byte en tamaños de mensaje grandes. En este sentido, el modelo LoOgGP se puede entender como una generalización del modelo LogP, para un tamaño de mensaje variable, que considera que tanto el *gap* como el *overhead* presentan una dependencia lineal con el tamaño de mensaje.

Por lo tanto, este nuevo modelo interpreta que hay 4 características que describen el comportamiento de los mensajes: el *overhead*, el *gap*, la latencia y el número de procesadores. Dos de estas características (*overhead* y *gap*) exhiben un comportamiento lineal respecto al tamaño del mensaje enviado, mientras que las otras dos (latencia y número de procesadores) son independientes de la cantidad de bytes de la comunicación.

Los modelos LogP y LogGP pueden obtenerse como simplificaciones del modelo LoOgGP. En concreto, el modelo LogP se puede obtener directamente a partir del modelo LoOgGP simplemente eliminando los parámetros que consideran tamaños de mensaje variables (**O** y **G**). Por otro lado, el modelo LogGP se corresponde con el modelo LoOgGP si se elimina únicamente el parámetro **O**. En cualquier caso, el modelo LoOgGP es un modelo puramente descriptivo, que permite caracterizar de manera muy precisa el comportamiento de los mensajes en un sistema concreto en función de los conceptos definidos por el modelo LogP.

3.2.1 Cálculo de los parámetros LoOgGP

El modelo LoOgGP encaja de manera natural en la metodología de medida PRTT, por lo que el cálculo de los parámetros de este nuevo modelo puede realizarse de manera análoga. Si consideramos el comportamiento lineal del *overhead*, la expresión (3.3) se transformará en:

$$T_o(s) = \mathbf{o} + \mathbf{O} \times (s-1) \tag{3.6}$$

Por lo tanto, el cálculo de los parámetros **g**, **G**, **o** y **O** del modelo LoOgGP se podrá realizar mediante el ajuste lineal de los valores experimentales de las variables $T_g(s)$ y $T_o(s)$, de acuerdo con las expresiones (3.4) y (3.6).

Debido a su propia naturaleza, el parámetro **L** no depende del tamaño del mensaje, por lo que debe ser tratado de manera independiente. Este parámetro se puede calcular de cualquiera de las formas utilizadas por los diferentes métodos de medida de la familia LogP. Sin embargo, las medidas habituales de este parámetro no son aplicables en los sistemas actuales debido al solapamiento inherente entre la latencia y el *overhead* [20]. Siguiendo la misma aproximación que Hoefler et ál. [79], una aproximación adecuada del valor de la latencia se puede obtener a partir de la expresión (3.5), es decir, la mitad del tiempo del *microbenchmark Ping-Pong* para mensajes de tamaño mínimo.

3.3 Caracterización de las comunicaciones en TIA

El sistema operativo, el *middleware* de comunicación y el hardware de red pueden influir de manera directa en el valor de los parámetros del modelo. Por lo tanto, es necesario un método eficaz de caracterización del comportamiento de la red en un sistema paralelo concreto, que proporcione el valor numérico de los parámetros del modelo y los diferentes rangos de comportamiento.

El procedimiento de cálculo de los parámetros del modelo LoOgGP se puede implementar dentro del entorno TIA, ya que la metodología PRTT se puede dividir en dos tareas independientes —medida y procesamiento de datos—, que se identifican claramente con las dos fases del entorno TIA, instrumentación y análisis [117]. En particular, se han diseñado un *driver* de `call` y una función en R para implementar estas dos tareas. El *driver* permite realizar todas las medidas PRTT necesarias, mientras que la función calcula automáticamente, durante la fase de análisis del entorno TIA, los valores de los parámetros y los tamaños de mensaje en los cuales aparece un cambio de comportamiento. A continuación se describen estos dos componentes.

```
#include "mpi.h"
...
#pragma cll parallel MPI
#pragma cll uses NGMPI
...
int main (int argc, char *argv[])
{
  ...
  MPI_Init(argc, argv);

#pragma cll ng NGMPI_LOGP=ng[0]

  ...

#pragma cll end ng

  ...

  MPI_Finalize();
  ...
}
```

Figura 3.7: Esquema de uso del *driver* NGMPI de CALL.

3.3.1 *Driver* NGMPI

NGMPI es un *driver* de `call` que permite obtener los datos necesarios para calcular los parámetros del modelo LoOgGP, en un rango de tamaño de mensajes determinado, utilizando el método descrito anteriormente. En concreto, este *driver* mide el tiempo de ejecución de las diferentes configuraciones $PRTT(n,d,s)$ necesarias para calcular $T_g(s)$ y $T_o(s)$, de acuerdo con las expresiones (3.1) y (3.2). Además, para realizar el cálculo de la latencia, este *driver* mide el tiempo de ejecución del *microbenchmark Ping-Pong* para mensajes de tamaño mínimo.

La figura 3.7 muestra un esquema del uso del *driver* NGMPI. Al igual que los *drivers* Ganglia y NWS (véase la sección 2.3.1), este *driver* utiliza internamente funciones de comunicación MPI, por lo que es necesario que se ejecute dentro de un entorno MPI. El observable `NGMPI_LOGP` proporciona los valores medidos de las diferentes configuraciones del *microbenchmark* PRTT necesarias para obtener los parámetros LoOgGP, concretamente, $PRTT(1,0,s)$, $PRTT(n,0,s)$, $PRTT(n,d(s),s)$ y $PRTT(1,0,1)$ —el valor de $d(s)$ se ha establecido exactamente al doble del valor de $PRTT(1,0,s)$—. Llamaremos experimento PRTT

a la ejecución de este conjunto de *microbenchmarks* para una configuración $\{n,s\}$ particular. Los diferentes experimentos PRTT se ejecutan en el inicio del experimento CALL, por lo que en realidad se está caracterizando el estado particular de la red en el instante de iniciar el experimento (línea 11, en la figura 3.7).

Aunque los experimentos PRTT se realizan dentro del propio *driver*, la naturaleza de este observable es similar a otros observables que recogen información de aplicaciones externas —como, por ejemplo, los observables `GANGLIA_MPI` y `NWS_LBW`, en los *drivers* Ganglia y NWS, respectivamente—. Por lo tanto, y para evitar posibles interferencias en los valores de otros *drivers*, el *driver* NGMPI debe ser utilizado en un experimento CALL independiente, cuyo único observable sea `NGMPI_LOGP`. Asimismo, y de manera similar a los *drivers* Ganglia y NWS, los valores generados por el *driver* NGMPI se almacenan en un nuevo fichero XML, con extensión `.ngmpi.xml`, ya que el procesamiento de estos datos es totalmente independiente respecto al de los valores generados por otros experimentos CALL.

El experimento PRTT se ejecuta para diferentes tamaños de mensaje, desde el tamaño mínimo (1 byte) hasta un tamaño máximo determinado. Las distintas ejecuciones del *microbenchmark* PRTT utilizan únicamente dos procesadores —nótese que los modelos LogP, LogGP y LoOgGP no contemplan la contención de la red—, por lo que existen diferentes alternativas para ejecutar los *microbenchmarks* en un entorno MPI de más de dos procesos. En concreto, el *driver* NGMPI dispone de 4 modos de funcionamiento:

- *Sampling*. Este modo selecciona aleatoriamente dos procesos en cada ejecución del experimento PRTT.
- *Subgroup*. Este modo se comporta igual que el modo *sampling*, pero utilizando únicamente un subgrupo de los procesos.
- *Pairs*. En este modo los procesos MPI se agrupan por parejas que realizan los distintos experimentos PRTT de manera independiente.
- *Full*. Este último modo ejecuta de manera independiente los distintos experimentos PRTT con todas las posibles parejas de procesos que se puedan formar.

Las diferentes opciones del *driver* NGMPI se pueden modificar a través del fichero auxiliar `cll_ngmpi_map`. La figura 3.8 muestra un ejemplo con las diferentes opciones de este *driver*. La opción `CLL_NGMPI_N_PARAMETER` determina el parámetro *n* del *microbenchmark* PRTT. El número de repeticiones de cada experimento PRTT se determina a tra-

```
1  CLL_NGMPI_N_PARAMETER 32
2  CLL_NGMPI_TIMES 1
3  CLL_NGMPI_SIZE_MAX 100000
4  CLL_NGMPI_NUM_STEPS 2
5  CLL_NGMPI_STEP 1024 10000
6  CLL_NGMPI_STEP 10240 100000
7  CLL_NGMPI_NUM_PROC 6
8  CLL_NGMPI_MODE 1
9  CLL_NGMPI_SUBSIZE 3
10 CLL_NGMPI_SUBNODES 1 3 5
```

Figura 3.8: Ejemplo de fichero de configuración del *driver* NGMPI

vés de la opción `CLL_NGMPI_TIMES`, es decir, para cada tamaño de mensaje se obtendrán `CLL_NGMPI_TIMES` medidas experimentales. La opción `CLL_NGMPI_SIZE_MAX` determina el tamaño máximo en bytes y, por ende, el rango de tamaños de mensaje considerado —el tamaño de mensaje mínimo es siempre 1 byte—. El muestreo del espacio de tamaños de mensaje se configura a través de los parámetros `CLL_NGMPI_NUM_STEPS` y `CLL_NGMPI_STEP`. El primero de ellos indica el número de segmentos en los que se divide el espacio de muestreo, y que presentarán diferentes frecuencias de muestreo. Para cada uno de estos segmentos debe proporcionarse una línea `CLL_NGMPI_STEP` que indique la distancia en bytes entre tamaños de mensaje consecutivos y el tamaño máximo del segmento. Cada segmento comienza en el tamaño máximo del anterior, siendo 1 byte el valor mínimo del primero y `CLL_NGMPI_SIZE_MAX` el valor máximo del último segmento. Este mecanismo de muestreo permite reducir el número de medidas experimentales, evitando un muestreo excesivo en zonas de poco interés, pero también permite una mayor resolución en otras zonas de mayor interés. El número de procesos MPI involucrados en la ejecución del experimento NGMPI se indica a través del parámetro `CLL_NGMPI_NUM_PROC`. Este valor debe ser igual al número de procesos del entorno MPI. Los 4 modos de funcionamiento del *driver* NGMPI se seleccionan a través del valor del parámetro `CLL_NGMPI_MODE`. Los valores 0, 1, 2 y 3 seleccionan los modos *sampling*, *subgroup*, *pairs* y *full*, respectivamente. La opción `CLL_NGMPI_SUBSIZE` determina el tamaño del subgrupo en el modo *subgroup*. Los procesos considerados en este modo se indican a través del parámetro `CLL_NGMPI_SUBNODES`.

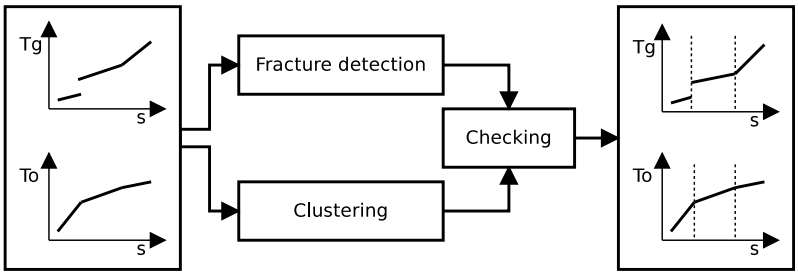


Figura 3.9: Esquema del proceso de detección de los diferentes intervalos de comportamiento de las funciones $T_o(s)$ y $T_g(s)$

3.3.2 Detección de intervalos y cálculo de los parámetros

En la fase de análisis del entorno TIA, se han desarrollado un conjunto de procedimientos que implementan los cálculos necesarios para obtener los parámetros del modelo LoOgGP, de acuerdo con las ecuaciones (3.4), (3.5) y (3.6), en un intervalo de tamaño de mensajes determinado, a partir de los datos proporcionados por el *driver* NGMPI. Sin embargo, es preciso establecer previamente los intervalos en los cuales las variables $T_o(s)$ y $T_g(s)$ se pueden aproximar linealmente.

La figura 3.9 muestra un esquema del proceso de detección automática de intervalos de comportamiento lineal. En primer lugar, es necesario calcular los valores experimentales de $T_o(s)$ y $T_g(s)$ utilizando las ecuaciones (3.1) y (3.2), a partir de los valores medidos en los diferentes experimentos PRTT. El dominio de las funciones $T_o(s)$ y $T_g(s)$ es un conjunto discreto, $\{S\}$, que contiene los tamaños de mensaje considerados durante la ejecución del *driver* NGMPI —es decir, se dispone de un número limitado de medidas experimentales—, cuyos valores mínimo y máximo son $s_{\text{mín}}$ y $s_{\text{máx}}$, respectivamente. Por lo tanto, el problema se convierte en la distribución de un conjunto discreto de valores en función del comportamiento de las funciones $T_o : S \rightarrow \mathbb{R}$ y $T_g : S \rightarrow \mathbb{R}$. El hecho de que el conjunto $\{S\}$ sea discreto, unido a la inevitable dispersión de las medidas experimentales, dificulta especialmente la discriminación de pequeñas distorsiones que no impliquen un cambio significativo en el comportamiento principal de la función. Para realizar una detección eficaz de los intervalos de comportamiento en este contexto particular, se han desarrollado dos tareas que implementan métodos independientes para detectar los elementos de S en los cuales existe una alteración del comportamiento lineal, mediante un análisis de los valores calculados de $T_o(s)$ y $T_g(s)$. En particular, la primera tarea (*fracture detection*) busca discontinuidades en estas funciones,

mientras que la segunda tarea (*clustering*) caracteriza cada punto experimental en función de la estimación local de algunos parámetros del modelo LoOgGP. El resultado de cada tarea es un conjunto de tamaños de mensaje que son potenciales *candidatos* a ser frontera entre dos intervalos de comportamiento diferentes. Finalmente, los candidatos generados por ambas tareas se combinan adecuadamente en un solo conjunto para determinar la distribución definitiva de los intervalos de comportamiento detectados (*checking*). En este último proceso, también deben eliminarse aquellos intervalos de comportamiento con un rango de tamaños de mensaje relativamente pequeño en comparación con el intervalo $[s_{\text{mín}}, s_{\text{máx}}]$.

Este método de detección de intervalos presenta diversas ventajas frente a la aproximación de Hoefler et ál. [79]. En primer lugar, utiliza el modelo LoOgGP, un modelo más complejo que el modelo LogGP y que permite una mejor caracterización del comportamiento experimental. Asimismo, este método permite la aplicación de técnicas de detección más potentes y eficaces ya que, siguiendo la filosofía del entorno TIA, el análisis se realiza una vez que todos los valores PRTT han sido medidos —de esta forma, se puede obtener una visión completa del comportamiento global de las variables T_o y T_g en el intervalo de tamaños considerado—. Además, la detección automática de los intervalos de comportamiento también presenta ventajas frente a una inspección manual de los valores experimentales de $T_g(s)$ y $T_o(s)$. En particular, la automatización del proceso proporciona un mecanismo que permite agilizar la toma de decisiones en procesos críticos —como, por ejemplo, la planificación de mensajes o los algoritmos de enrutamiento adaptables— sin la intervención de un operador. Por otro lado, esta caracterización precisa del comportamiento real de las comunicaciones puede integrarse fácilmente en herramientas de análisis que simulen entornos paralelos basados en modelos analíticos de comunicación como, por ejemplo, Dimemas [14].

Tarea *fracture detection*

El objetivo de esta tarea es la detección de aquellos puntos de $T_o(s)$ y $T_g(s)$ en los que exista una discontinuidad sustancial en el comportamiento lineal de estas funciones. Esta tarea recorre, para cada función, los diferentes tamaños de mensaje considerados dentro del rango $[s_{\text{mín}}, s_{\text{máx}}]$, buscando saltos cualitativamente importantes en los valores de la función, y que no puedan ser atribuidos al comportamiento lineal o a errores experimentales.

Este proceso de detección se basa en el hecho de que, tanto para $T_o(s)$ como para $T_g(s)$, los distintos tramos presentan comportamientos lineales con pendientes positivas. Además, las discontinuidades en el comportamiento lineal implican, en cualquier caso, un salto temporal

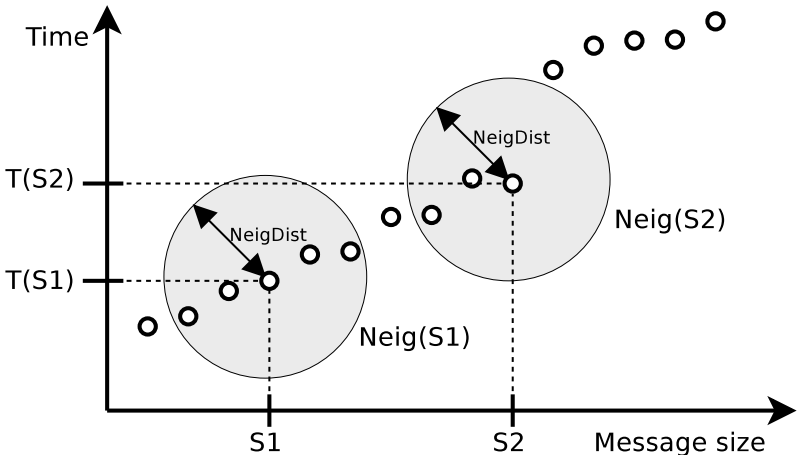


Figura 3.10: Esquema del funcionamiento de la tarea *fracture detection*

positivo. Estas dos consideraciones son una consecuencia directa de que, en mayor o menor grado, siempre es más costoso enviar un mensaje de mayor tamaño. Teniendo en cuenta que las funciones $T_o(s)$ y $T_g(s)$ son en realidad un conjunto discreto de puntos experimentales, es necesario que el método sea suficientemente robusto para discriminar la dispersión experimental.

La figura 3.10 muestra el fundamento de este método de detección. En la figura se representan 15 puntos experimentales de una función genérica $T(s)$ que presenta el mismo tipo de discontinuidades que las funciones $T_o(s)$ y $T_g(s)$. Consideremos el tamaño de mensaje s_k y definamos un *conjunto de vecindad* con los valores de $T(s)$ más próximos a $T(s_k)$. Aunque existen diversas alternativas para definir este entorno, en este ejemplo particular se ha utilizado la distancia `NeigDist` en el espacio euclídeo formado por el dominio y la imagen de la función. Los valores experimentales cuya distancia a $T(s_k)$ sea menor que `NeigDist` se consideran pertenecientes a la vecindad de $T(s_k)$. En el ejemplo de la figura, la vecindad de S1 —`Neig(S1)`— contiene cinco puntos experimentales (incluyendo su propio valor), mientras que la vecindad de S2 —`Neig(S2)`— sólo contiene tres elementos. En general, un tamaño de mensaje s_k no se corresponde con el mayor tamaño de mensaje asociado a su vecindad, excepto cuando el punto está situado en el borde de una discontinuidad (por ejemplo, el punto S2 en la figura).

```
1  input:
2
3     {S} = {s1, s2, s3...sN} /* Set of message sizes */
4     To: {S} → ℝ /* Experimental data: overhead */
5     Tg: {S} → ℝ /* Experimental data: gap */
6     NEIGH /* Neighborhood distance */
7
8  begin
9
10    {fractures} = ∅  /* Set of candidate message sizes */
11
12    for ( i in 1 to N ) do
13
14       {Neio} = GetNeighborhood( To(si), NEIGH )
15       {Neig} = GetNeighborhood( Tg(si), NEIGH )
16
17       if ( si == máx({Neio}) OR si == máx({Neig}) )
18          {fractures} = {fractures} ∪ {si}
19       end if
20
21    end for
22
23    return {fractures}
24
25 end
```

Figura 3.11: Pseudocódigo de la tarea *fracture detection*

La figura 3.11 muestra el pseudocódigo de esta tarea. Para cada tamaño de mensaje (s_i) se determina la vecindad correspondiente a ese tamaño de mensaje, en cada una de las funciones (líneas 14 y 15). El parámetro `NEIGH` determina el tamaño de la vecindad. Si el valor de s_i se corresponde con el valor máximo de tamaño de mensaje en al menos una de las dos vecindades (línea 17), el tamaño de mensaje s_i es considerado como una *fractura* en el comportamiento lineal de las funciones. El resultado final de la tarea es un conjunto de tamaños de mensaje (`fractures`) en los que se han detectado discontinuidades en las funciones $T_o(s)$ y $T_g(s)$.

Tarea *clustering*

La segunda tarea utiliza técnicas de *clustering* [90] para clasificar cada tamaño de mensaje en función de estimaciones locales de algunos parámetros del modelo LoOgGP. Esta tarea también recorre los diferentes tamaños de mensaje en el intervalo $[s_{\text{mín}}, s_{\text{máx}}]$ pero, en este caso, se estiman los valores de los parámetros **o**, **O**, **g** y **G** del modelo LoOgGP en la vecin-

dad de cada tamaño de mensaje. El conjunto de tamaños de mensaje es clasificado según la afinidad de estas estimaciones y de los propios valores de las funciones $T_o(s)$ y $T_g(s)$.

La figura 3.12 muestra el pseudocódigo de esta tarea. Para cada tamaño de mensaje considerado, se determina la vecindad de cada función (líneas 16 y 17) y se calculan los valores de los parámetros del modelo LoOgGP en las correspondientes vecindades (líneas 19 a 22). Para un tamaño de mensaje s_k, las estimaciones locales de los dos parámetros de *overhead* se corresponden con el punto de corte —$\sigma_o(s_k)$— y la pendiente —$\omega_o(s_k)$—, que se obtienen del ajuste lineal de los puntos experimentales de $T_o(s)$ en la vecindad de $T_o(s_k)$. Análogamente, las estimaciones de los parámetros $\mathbf{g}$ y $\mathbf{G}$ para el tamaño de mensaje s_k —$\sigma_g(s_k)$ y $\omega_o(s_k)$, respectivamente— se obtienen del ajuste lineal de $T_g(s)$ en la vecindad de $T_g(s_k)$.

Para aplicar la técnica de *clustering*, es necesario considerar un conjunto que describa las características de cada uno de los elementos (denominados patrones) que se desean clasificar. En este caso, cada patrón se corresponde con una determinada medida experimental. Se deben considerar aquellas características que aporten información relevante respecto al comportamiento lineal de $T_o(s)$ y $T_g(s)$. En particular, se han considerado las cuatro estimaciones calculadas en los ajustes —$\sigma_o(s)$, $\omega_o(s)$, $\sigma_g(s)$ y $\omega_g(s)$—, así como las variables $T_o(s)$ y $T_g(s)$. Por lo tanto, el conjunto de patrones se define como una matriz $N \times 6$, donde N es el número de medidas experimentales consideradas. Las columnas de esta matriz están formadas por cada una de las seis variables consideradas. Por otro lado, cada fila se corresponde con un punto en un espacio hexadimensional, identificando de manera singular las diferentes características de cada medida experimental.

Para garantizar una contribución equitativa de las diferentes variables, éstas se deben normalizar (líneas 26 y 27). En particular, las variables $T_o(s)$ y $T_g(s)$ son normalizadas linealmente pero en el resto de variables se utiliza una función *sigmoid* (una función lineal con límites de saturación). Esta consideración especial para las variables que provienen de los ajustes es necesaria para evitar que la normalización esté dominada por valores puntuales, alejados de la media. Aunque pueden surgir debido a valores espurios, este tipo de valores singulares suelen estar relacionados con los cambios de comportamiento —y, especialmente, con las discontinuidades— de $T_o(s)$ y $T_g(s)$.

Los diferentes elementos del conjunto de patrones se agrupan en función de sus características mediante una técnica de *clustering* (línea 29). En este caso, el número final de posibles grupos o *clusters* no es conocido a priori, por lo que se utiliza una técnica de agrupamiento jerárquico (*agglomerative hierarchical technique* [90]). Inicialmente, esta técnica considera que existen tantos *clusters* como patrones, de tal manera que cada *cluster* está formado por un

```
1  input:
2
3    {S} = {s1, s2, s3...sN} /* Set of message sizes */
4    To: {S} → ℝ /* Experimental data: overhead */
5    Tg: {S} → ℝ /* Experimental data: gap */
6    NEIGH /* Neighborhood distance */
7    THRES /* Maximum distance allowed between cluster centroids */
8
9  begin
10
11   σo, ωo: {S} → ℝ
12   σg, ωg: {S} → ℝ
13
14   for ( i in 1 to N ) do
15
16     {Neio} = GetNeighborhood( To(si), NEIGH )
17     {Neig} = GetNeighborhood( Tg(si), NEIGH )
18
19     σo(si) = GetLinearFitSlope( {Neio} )
20     ωo(si) = GetLinearFitOrigin( {Neio} )
21     σg(si) = GetLinearFitSlope( {Neig} )
22     ωg(si) = GetLinearFitOrigin( {Neig} )
23
24   end for
25
26   LinearNormalization( To, Tg )
27   SigmoidNormalization( σo, ωo, σg, ωg )
28   matrix = SetPatterns(To, σo, ωo, Tg, σg, ωg)
29   Dendrogram = AgglomerativeClustering( matrix )
30
31   nclus = 1
32   repeat
33     nclus = nclus+1
34     {clusters} = GetClusters( Dendrogram, nclus )
35     {distances} = GetCentroidsDistance( {clusters} )
36   until ( mín( {distances} ) < THRES )
37   {clusters} = GetClusters( Dendrogram, nclus-1 )
38
39   {clusters} = CheckOverlap( {clusters} )
40   {borders} = GetBorders( {clusters} )
41
42   return {borders}
43
44 end
```

Figura 3.12: Pseudocódigo de la tarea *fracture detection*

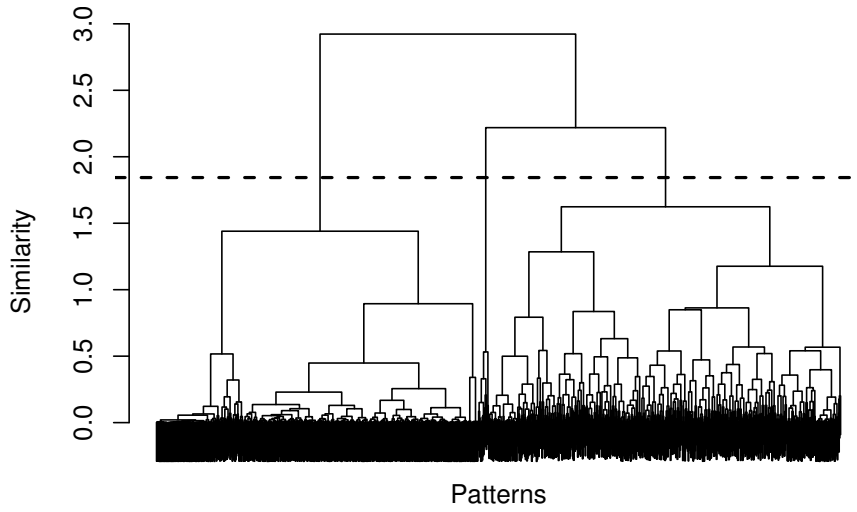


Figura 3.13: Ejemplo de dendograma

único elemento del conjunto de patrones. En un proceso iterativo, los *clusters* se fusionan por parejas hasta que todos los patrones estén contenidos en un único *cluster*. En cada iteración, cada *cluster* se fusiona con el *cluster* más afín según el criterio de similitud establecido. El resultado se puede mostrar mediante un dendograma que represente la estructura jerárquica del proceso y el nivel de similitud entre cada *cluster*. La figura 3.13 muestra un ejemplo de dendrograma, definido para todos los puntos en el espacio 6-dimensional, obtenido mediante agrupamiento jerárquico sobre valores $T_o(s)$ y $T_g(s)$.

Sin embargo, el agrupamiento jerárquico no proporciona un número concreto de posibles *clusters* o grupos, por lo que es necesario establecer un criterio para determinar la agrupación definitiva de los diferentes patrones considerados. En la tarea *fracture detection* se ha desarrollado un proceso iterativo para determinar el número óptimo de *clusters* (líneas 31 a 36). Este proceso comienza considerando el *cluster* final al que pertenecen todos los patrones. A partir de la información obtenida durante el proceso de *aglomeración*, se determina la distribución de los patrones (el conjunto `clusters` en el pseudocódigo) si se aumentase en una unidad el número de *clusters* (línea 34). La distancia entre los centroides de esta nueva distribución, en el espacio hexadimensional definido por el conjunto de patrones, es calculada en esta nueva distribución (línea 35). Este proceso se repite, incrementando en cada iteración el número de *clusters*, hasta que la distancia entre los centroides más próximos en la distribu-

ción considerada sea menor que un determinado umbral (`THRES`). Este umbral especifica la distancia mínima para considerar que dos *clusters* pueden ser claramente diferenciados. En la figura 3.13, la línea horizontal punteada indica el nivel de similitud alcanzado mediante este proceso y, en el caso particular de la figura, el número óptimo de *clusters* es tres.

El último proceso de esta tarea debe verificar la coherencia de los *clusters* detectados, resolviendo los posibles solapamientos (línea 39). Es decir, si suponemos dos *clusters* diferentes, α y β, los tamaños de mensaje del conjunto deben verificar que, si el mínimo de α es menor que el mínimo de β, entonces el máximo de α debe ser también menor que el mínimo de β.

El resultado final de la tarea es un conjunto de tamaños de mensaje (`borders`) que identifican los límites de los diferentes intervalos de comportamiento lineal, es decir, los valores máximo y mínimo de tamaño de mensaje asociados a cada *cluster*.

Tarea *checking*

Los candidatos propuestos por las tareas *fracture detection* y *clustering* definen los tamaños de mensaje en los cuales existe una alteración en el comportamiento lineal de las funciones $T_o(s)$ y $T_g(s)$, es decir, estas tareas determinan los puntos frontera entre los diferentes rangos de comportamiento. Idealmente, los resultados de la tarea *fracture detection* deberían estar incluidos en el conjunto de candidatos de la tarea *clustering*. Sin embargo, debido a la naturaleza experimental de los valores obtenidos, es posible que no haya una coincidencia exacta o, incluso, que algún valor sólo sea detectado por una de las tareas. La tarea *checking* es la responsable de combinar adecuadamente los resultados de ambas tareas.

La combinación de los resultados de ambas tareas es inmediata, ya que simplemente es la unión de dos conjuntos. Sin embargo, es necesario comprobar que esta combinación proporcione rangos de comportamiento adecuados, comprobando que el tamaño de los diferentes rangos de tamaño de mensaje resultantes supere un tamaño mínimo predeterminado. Por ejemplo, la aparición de dos puntos frontera muy próximos —es decir, un rango de comportamiento muy pequeño— es, probablemente, una distorsión en los datos experimentales y ambos puntos identifican el mismo cambio de comportamiento. La tarea *checking* es, por lo tanto, la responsable de resolver estas situaciones y determinar a qué rango corresponden los tamaños de mensaje comprendidos entre estos dos valores.

Implementación del proceso de detección de intervalos y cálculo de parámetros

La detección de los diferentes intervalos de comportamiento de las funciones $T_o(s)$ y $T_g(s)$ y el cálculo de los correspondientes parámetros LoOgGP, que se resume en la figura 3.9, han sido implementados en la fase de análisis del entorno TIA. En concreto, se ha diseñado una función integrada en la función `cll.import` del módulo IMPORT (véase la sección 2.5), lo que permite la incorporación de los parámetros LoOgGP como posibles variables en otros experimentos CALL.

La implementación del proceso de detección de los rangos de comportamiento se puede configurar mediante el parámetro `ngmpi.options`, de la función `cll.import`. Este parámetro es una lista que está formada por los siguientes elementos:

`window.size`. Establece el tamaño de la vecindad de cada experimento PRTT en función del tamaño de la muestra. Más concretamente, este parámetro indica un porcentaje del número total de valores experimentales considerados. Por lo tanto, su valor debe estar comprendido entre 0 y 1.

`clus.method`. Indica el método de agrupamiento jerárquico utilizado en la tarea *clustering*. Las alternativas implementadas son `complete` y `single`, que represetan, respectivamente, los métodos *complete link* y *single link* [90].

`clus.metric`. Indica el tipo de métrica utilizada en el método de agrupamiento jerárquico. Las diferentes alternativas implementadas actualmente son la métrica euclídea (`euclidean`) y la métrica Manhattan (`manhattan`).

`clus.threshold`. Determina el umbral durante el proceso de optimización del número de *clusters*, es decir, el límite de separación entre los centroides de los diferentes *clusters*. Este parámetro indica un porcentaje de la distancia máxima teórica dentro del espacio que determina el conjunto de patrones, por lo que su valor debe estar comprendido entre 0 y 1.

En cualquier caso, en esta implementación se proponen valores por defecto, basados en la experiencia: un tamaño de vecindad del 10 % (`0.1`), el método `complete link`, la métrica `manhattan` y un umbral del 10 % (`0.1`).

En primer lugar, esta función lee los valores PRTT almacenados en los archivos XML. Para cada experimento PRTT —es decir, para cada medida experimental proporcionada por

el *driver* NGMPI— los valores de las funciones $T_o(s)$ y $T_g(s)$ se calculan según las ecuaciones (3.1) y (3.2). Antes de continuar con las distintas tareas del proceso de caracterización de $T_o(s)$ y $T_g(s)$, se aplica un filtro para eliminar posibles valores espurios, o *outliers*, en estas funciones. Este filtro elimina aquellos experimentos PRTT que generen valores negativos en $T_o(s)$ y $T_g(s)$, ya que estos valores son incongruentes con el significado físico de las propias funciones. Asimismo, para cada función, también se eliminan aquellos experimentos PRTT cuyo valor está alejado de la media obtenida para las diferentes instancias que comparten el mismo tamaño de mensaje; el límite práctico se establece en dos veces la desviación estándar. Por otro lado, para cada función, se determina la vecindad de cada instancia y se eliminan aquellos puntos cuyo valor de tamaño de mensaje esté cercano al valor medio de la vecindad, pero para los que su valor temporal asociado esté situado cerca de los valores extremos de la vecindad. La vecindad de un experimento PRTT, con un tamaño de mensaje asociado s_i, está formada por las N instancias con las que se obtiene una menor diferencia $|s_i - s|$, siendo N el número de instancias determinado por el parámetro `window.size`. Esta aproximación prioriza la resolución del muestreo sobre la magnitud de los valores experimentales.

Una vez filtrados los valores experimentales de $T_o(s)$ y $T_g(s)$, se realiza la detección de discontinuidades (tarea *fracture detection*). De manera análoga a la detección de *outliers*, en esta tarea se determina, para cada función, la vecindad de cada experimento PRTT mediante el parámetro `window.size`. Las potenciales discontinuidades serán los tamaños de mensaje asociados a las instancias cuyo tamaño de mensaje se corresponda con el mayor tamaño de mensaje de su vecindad.

La tarea *clustering* se ejecuta a continuación. En este caso, las correspondientes estimaciones locales de los parámetros LoOgGP se calculan en la vecindad de cada experimento PRTT. Estas estimaciones se normalizan y se construye el conjunto de patrones. La técnica de *clustering* se aplica sobre este conjunto de patrones mediante la función `agnes`, del paquete `cluster` de R, utilizando el método y la métrica que definen `clus.method` y `clus.metric`, respectivamente. A continuación, se determina el número óptimo de *clusters*, utilizando como umbral el parámetro `clus.threshold`. Como existe la posibilidad de que varias instancias de experimentos PRTT tengan el mismo tamaño de mensaje asociado, es necesario verificar que los intervalos detectados sean conjuntos disjuntos respecto del tamaño de mensaje, en otras palabras, dos instancias que tengan asociado el mismo tamaño de mensaje deben pertenecer al mismo *cluster*. Finalmente, se resuelven los posibles solapes entre los *clusters* detectados. En particular, los segmentos del rango de tamaños de mensaje que se solapan entre dos o más *clusters* se consideran un nuevo *cluster*.

Los diferentes intervalos obtenidos en ambas tareas se combinan en la tarea *checking* y se comprueba que el tamaño final de los *clusters* tenga un tamaño mínimo. En concreto, se utiliza como límite el número de valores experimentales definido por el parámetro `window.size`, por lo que el tamaño mínimo de los intervalos está determinado, en realidad, por la resolución del muestreo.

El valor de los parámetros **o**, **O**, **g** y **G**, para cada uno de los intervalos de comportamiento automáticamente detectados, se calcula mediante un ajuste lineal de los valores experimentales asociados al intervalo considerado, de acuerdo con las expresiones (3.4) y (3.6). El parámetro de latencia, **L**, que debido a su propia naturaleza es independiente de los diferentes protocolos de comunicación, es considerado de manera independiente al proceso de detección de intervalos y se calcula directamente a partir de la ecuación (3.5) utilizando los valores $PRTT(1,0,1)$ de los experimentos PRTT, una vez eliminadas las instancias asociadas con *outliers*.

3.4 Resultados experimentales

La implementación en TIA de la metodología de caracterización de los parámetros del modelo LoOgGP en entornos MPI ha sido evaluada en el cluster `tegasaste`, descrito en la sección 3.2. Las diferentes configuraciones experimentales consideradas se corresponden con los entornos mostrados en la tabla 3.1. El driver NGMPI han sido ejecutado utilizando únicamente dos nodos del cluster. A continuación, se presentan por separado la caracterización para tamaños de mensaje grandes (hasta 200 KB) y tamaños de mensaje pequeños (menores que 4 KB).

3.4.1 Tamaños de mensaje grandes

En esta sección se presenta la caracterización LoOgGP obtenida para tamaños de mensaje comprendidos entre 1 byte y 200 Kbytes. En concreto, el driver NGMPI ha sido configurado para utilizar el modo *sampling*, siendo $n = 32$ el parámetro del *microbenchmark* PRTT y realizando 10 repeticiones de cada experimento PRTT. Además, se han considerado diferentes incrementos entre tamaños de mensaje consecutivos. En particular, se ha utilizado un incremento de 256 bytes en los tamaños menores que 2 KB, 512 bytes en el intervalo 2-10 KB, 1024 bytes en el intervalo 10-100 KB y 2 KB para tamaños mayores que 100 KB.

Tabla 3.2: Caracterización de los parámetros LoOgGP para tamaños de mensaje grandes

	L (μs)	**o** (μs)	**O** (μs/KB)	**g** (μs)	**G** (μs/KB)	**Interval** (bytes)
GB+MPICH	73,1	12,65	1,39	10,09	12,34	1 → 32769
	73,1	-314,38	14,25	-146,65	14,84	34817 → 161793
	73,1	-427,67	14,77	-427,99	16,74	163841 → 202753
GB+OpenMPI	124,1	9,97	1,30	1,86	8,49	1 → 63489
	124,1	216,41	1,47	208,80	8,68	65537 → 202753
IB+OpenMPI	6,1	5,98	1,75	8,72	1,96	1 → 11265
	6,1	41,66	2,01	43,28	2,03	12289 → 202753
IB+MVAPICH	5,6	3,82	0,29	6,17	1,44	1 → 11265
	5,6	41,30	1,79	44,23	1,79	12289 → 55297
	5,6	40,68	1,79	48,84	1,72	57345 → 94209
	5,6	49,60	1,75	46,10	1,78	96257 → 202753

El método de detección de intervalos ha sido configurado con el método *complete link* con la métrica Manhattan. El tamaño de la vecindad ha sido establecido en un 10 % de los puntos experimentales. Asimismo, el umbral de la tarea *clustering* ha sido establecido en un 10 % de la distancia máxima teórica del espacio definido por el conjunto de patrones.

La tabla 3.2 muestra, para los diferentes entornos considerados en la tabla 3.1, los parámetros obtenidos en los diferentes intervalos detectados automáticamente. Las figuras 3.14, 3.15, 3.16 y 3.17 muestran gráficamente estos resultados, junto con los datos experimentales. En el entorno GB+MPICH el comportamiento experimental de $T_g(s)$ es prácticamente homogéneo y sin grandes variaciones, mientras que el comportamiento experimental de $T_o(s)$ presenta una clara discontinuidad en 32 KB. En concreto, el método implementado en TIA detecta tres intervalos de comportamiento cuyos límites se sitúan en 32 KB y 158 KB. El primer cambio de comportamiento se identifica con el tamaño del *socket buffer* de MPICH, es decir, el límite entre los protocolos *eager* y *rendezvous*. Sin embargo, en este caso particular, el segundo cambio de comportamiento detectado no se identifica claramente con ningún parámetro del sistema y se debe a la alta dispersión de los datos experimentales para tamaños mayores que 100 KB. En los dos casos en los que se utiliza Open MPI se detectan únicamente dos intervalos de com-

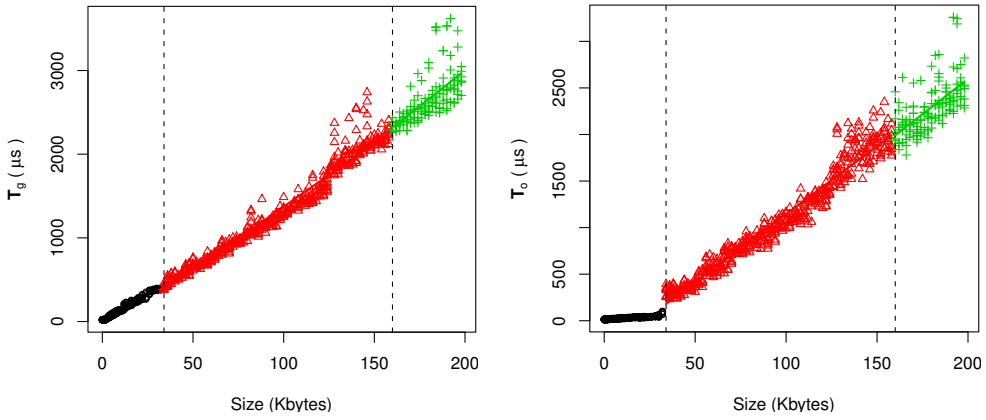


Figura 3.14: Resultados de la detección automática de intervalos en el entorno `GB+MPICH` para tamaños de mensaje grandes

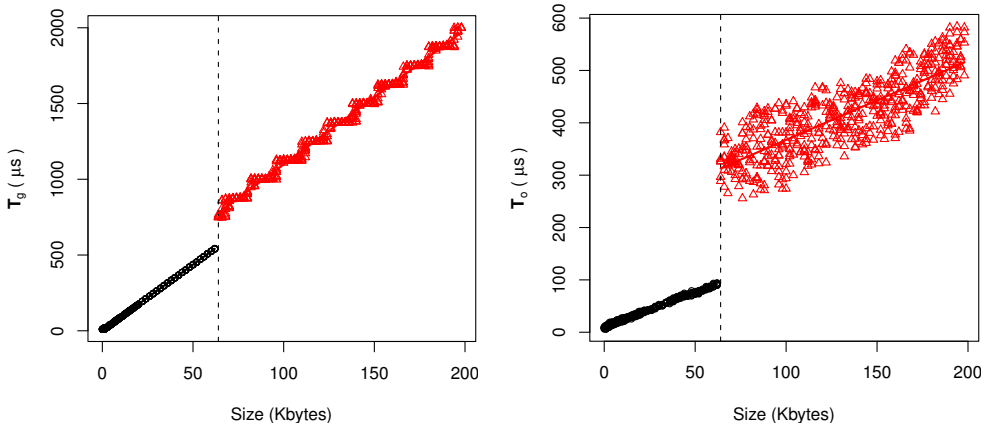


Figura 3.15: Resultados de la detección automática de intervalos en el entorno `GB+OpenMPI` para tamaños de mensaje grandes

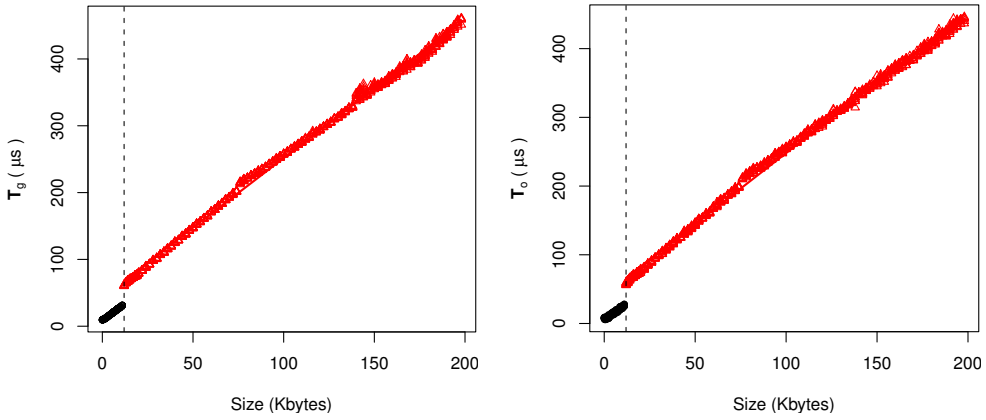


Figura 3.16: Resultados de la detección automática de intervalos en el entorno `IB+OpenMPI` para tamaños de mensaje grandes

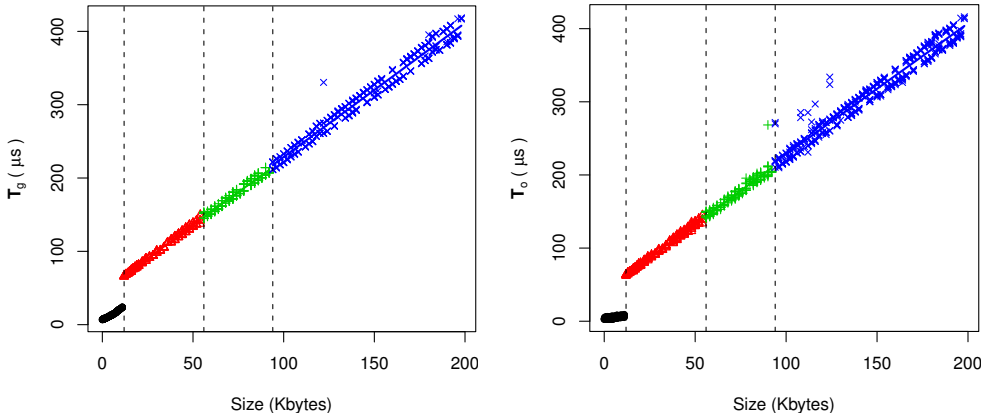


Figura 3.17: Resultados de la detección automática de intervalos en el entorno `IB+MVAPICH` para tamaños de mensaje grandes

portamiento. En el entorno `GB+OpenMPI` el cambio de comportamiento se sitúa en 64 KB y en el entorno `IB+OpenMPI` en 12 KB. En ambos casos, los valores detectados se identifica claramente con el tamaño límite entre los protocolos *eager* y *rendezvous* establecido en la implementación Open MPI del cluster `tegasaste`. Por otro lado, en el entorno `IB+MVAPICH` se detectan varios intervalos. Al igual que en el otro entorno con red InfiniBand™, uno de los valores detectados se corresponde con el límite entre los protocolos *eager* y *rendezvous* de esta configuración del cluster. Los otros dos cambios de comportamiento han sido detectados por la tarea *clustering* y están asociados a una mayor dispersión de los datos experimentales conforme se incrementa el tamaño del mensaje.

El hecho de que en el entorno `GB+MPICH` se detecten valores negativos en los parámetros **o** y **g** no supone una contradicción en la interpretación del modelo LoOgGP, ya que la aplicación de los modelos basados en LogP para la caracterización de las comunicaciones punto a punto proporciona, para cualquier rango de comportamiento mostrado en la tabla 3.2, valores de tiempo positivos. Es decir, para el rango de tamaños de mensaje de cada intervalo de comportamiento detectado, siendo s el tamaño de mensaje, las expresiones

$$\mathbf{o} + \mathbf{O} \times s$$
$$\mathbf{g} + \mathbf{G} \times s$$

siempre proporcionan valores de tiempo positivos. Por otro lado, estos valores negativos están asociados a valores elevados de **O** y **G**, lo que refleja el bajo rendimiento de las comunicaciones en estos intervalos de comportamientos. Desde otro punto de vista, se podría decir que la influencia de la componente intrínseca en el coste total de la comunicación es prácticamente residual, y que las componentes **O** y **G** son los factores dominantes en estos intervalos. Además, en estos intervalos de comportamiento también existe una alta dispersión experimental, que disminuye el rendimiento efectivo de las comunicaciones y que puede estar asociada a factores no contemplados en el modelo —como, por ejemplo, la contención en la red—.

3.4.2 Tamaños de mensaje pequeños

En este caso, el rango de mensajes considerado ha sido entre 1 y 4096 bytes. Al igual que en el caso de tamaños de mensaje grandes, el driver NGMPI ha sido configurado en el modo *sampling*, con $n = 32$ y 10 repeticiones. También se han considerado diferentes incrementos entre tamaños de mensaje consecutivos. En particular, se han utilizado incrementos de 10 bytes para tamaños menores que 1 KB y de 30 bytes para tamaños mayores que 1 KB.

Tabla 3.3: Caracterización de los parámetros LoOgGP para tamaños de mensaje pequeños

	L (μs)	**o** (μs)	**O** (μs/KB)	**g** (μs)	**G** (μs/KB)	**Interval** (bytes)
GB+MPICH	70,8	12,22	0,85	20,08	0,57	1 → 1271
	70,8	13,73	1,21	3,39	13,43	1301 → 4091
GB+OpenMPI	124,0	8,26	1,04	11,24	1,10	1 → 1301
	124,0	9,89	1,29	3,91	7,44	1331 → 4091
IB+OpenMPI	5,9	6,11	1,62	9,06	1,73	1 → 4091
IB+MVAPICH	7,8	3,90	0,42	7,05	0,96	1 → 1241
	7,8	4,32	0,19	6,38	1,44	1271 → 4091

Como en el caso anterior, en el método de detección de intervalos se ha utilizado el método *complete link* con la métrica Manhattan, siendo el tamaño de la vecindad el 10 % del número total de medidas experimentales y el umbral en la tarea *clustering* el 10 % de la distancia teórica máxima del espacio que define el conjunto de patrones.

La tabla 3.3 muestra, para los diferentes entornos considerados en la tabla 3.1, los parámetros obtenidos en los diferentes intervalos detectados. Las figuras 3.18, 3.19, 3.20 y 3.21 muestran gráficamente estos resultados, junto con los datos experimentales. En estos casos, todos los intervalos de comportamiento han sido detectados por la tarea *clustering*. De hecho, se puede apreciar en las figuras que no existe ninguna discontinuidad en los valores medidos de T_o y T_g. Por un lado, en los entornos que utilizan la red Gigabit Ethernet (`GB+OpenMPI` y `GB+MPICH`) se han detectado dos intervalos de comportamiento diferentes y, en ambos casos, el límite entre los intervalos se sitúa entre 1 KB y 1,5 KB. Este cambio de comportamiento es coherente con el comportamiento esperado si tenemos en cuenta que el valor de la MTU (*Maximum Transmission Unit*) de la red Gigabit Ethernet del cluster `tegasaste` es 1 500 bytes. Por otro lado, los entornos que utilizan InfiniBand™ presentan un valor de latencia bajo, como era de esperar por las características de esta tecnología. En cualquier caso, las pendientes, tanto de $T_o(s)$ como de $T_g(s)$, en el primer intervalo detectado de los cuatro entornos es siempre pequeña. Por lo tanto, podemos afirmar que el comportamiento detectado según el modelo LoOgGP es consecuente con la aproximación de un *overhead* y un *gap* constantes para tamaños pequeños, tal y como era esperable según el modelo LogP.

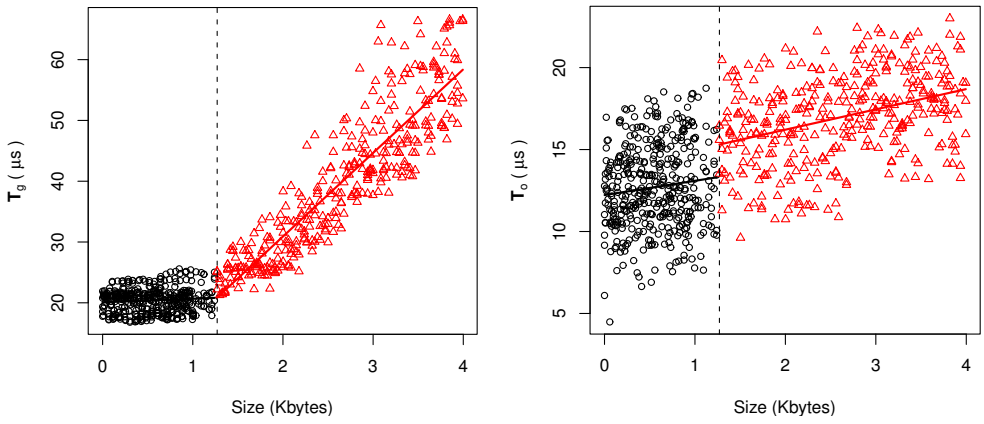


Figura 3.18: Resultados de la detección automática de intervalos en el entorno `GB+MPICH` para tamaños de mensaje pequeños

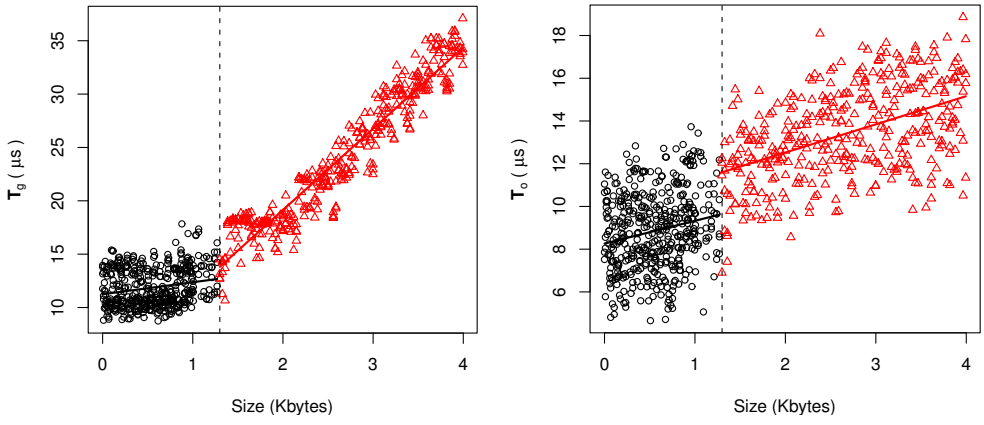


Figura 3.19: Resultados de la detección automática de intervalos en el entorno `GB+OpenMPI` para tamaños de mensaje pequeños

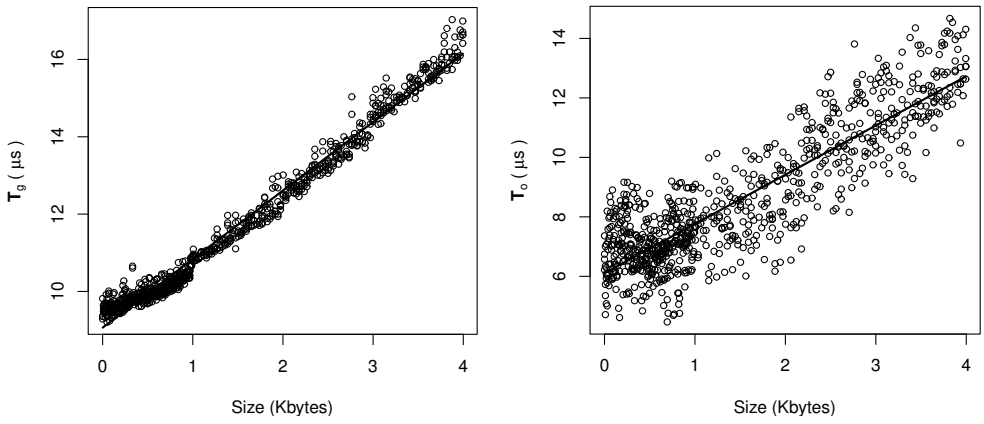


Figura 3.20: Resultados de la detección automática de intervalos en el entorno `IB+OpenMPI` para tamaños de mensaje pequeños

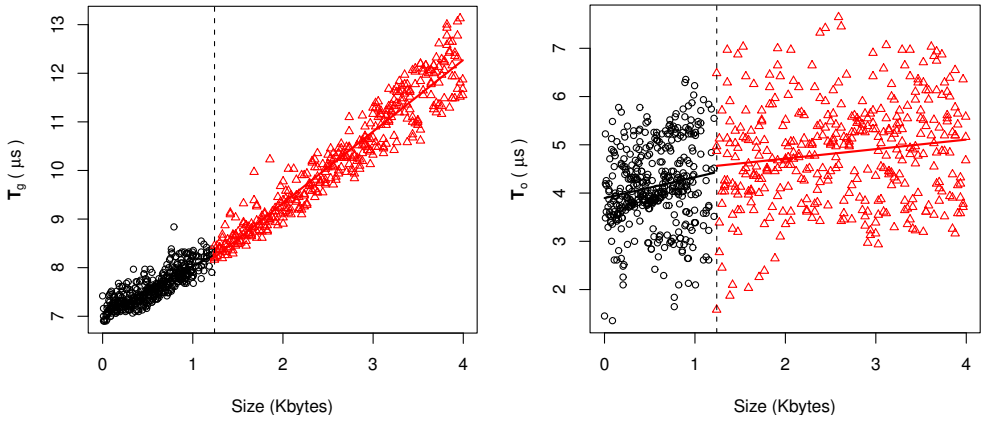


Figura 3.21: Resultados de la detección automática de intervalos en el entorno `IB+MVAPICH` para tamaños de mensaje pequeños

3.5 Contribuciones

Las contribuciones más relevantes de este capítulo son la definición del nuevo modelo LoOgGP y el diseño de la detección de intervalos, así como su implementación en TIA. Esta implementación ha sido utilizada para obtener los parámetros del modelo LoOgGP en diferentes sistemas paralelos. El trabajo desarrollado en este capítulo ha derivado en las siguientes publicaciones:

- *Obtención de modelos estadísticos de rendimiento de las comunicaciones en aplicaciones MPI*, XIX Jornadas de Paralelismo 2008 [122].
- *Accurate Analytical Performance Model of Communications in MPI Applications*, 8th International Workshop on Performance Modeling, Evaluation, and Optimization of Ubiquitous Computing and Networked Systems, PMEO UCNS'2009 [119].
- *Automatic Parameter Assessment of LogP-based Communication Models in MPI Environments*, International Conference on Computational Science, ICCS 2010 [117].

Capítulo 4

Selección de modelos con AIC

La selección de modelos es una herramienta estadística que permite cuantificar objetivamente la capacidad de un determinado número de modelos para caracterizar un comportamiento experimental. En este capítulo se presentan las motivaciones y fundamentos de la selección de modelos y, en particular, el método de selección de modelos basado en el criterio de información de Akaike. A continuación, se describe la implementación de este método en el entorno TIA y, finalmente, se realiza una comparación de los resultados obtenidos mediante esta técnica para diferentes casos de estudio con modelos teóricos basados en análisis algorítmicos.

4.1 Selección de modelos

Conceptualmente, los datos experimentales obtenidos mediante un experimento u observación contienen información acerca del proceso que se está estudiando. El objetivo de los modelos es caracterizar esa información y expresarla de una manera compacta y comprensible. En general, es imposible caracterizar completamente el proceso real mediante un modelo en base a la experimentación, simplemente por el hecho de que las muestras experimentales tienen un tamaño finito y, por lo tanto, solamente contienen una cantidad limitada de información del proceso real (*modelo verdadero*) que ha generado los datos. Esta circunstancia imposibilita la caracterización del comportamiento real de un proceso con absoluta certeza a partir de datos experimentales. El hecho de que todos los modelos se deben interpretar como aproximaciones se refleja claramente en la cita que formuló George Box: *Todos los modelos son falsos, pero algunos son útiles* [26].

En los procesos reales, el *modelo verdadero* es extremadamente complejo y, probablemente, tenga millones de variables. Sin embargo, cada variable afecta con una intensidad diferente al comportamiento real del proceso. Es habitual que haya un número relativamente pequeño de variables que contengan los efectos principales del proceso, muchas variables que representen los efectos secundarios y una miríada de variables que representen efectos prácticamente inapreciables. Desde un punto de vista pragmático, los modelos deben caracterizar únicamente la información importante del proceso (los efectos principales), a través de una representación concisa y comprensible, tratando como *ruido* el resto de efectos secundarios. Desde el punto de vista de la información, el objetivo real es obtener el modelo que pierda la menor cantidad de información posible.

La *selección de modelos* tiene como objetivo determinar el modelo más adecuado para representar la información contenida en una muestra experimental. En 1922, Ronald Fisher estableció que para obtener inferencias estadísticas válidas se deben considerar tres aspectos en el análisis de la muestra experimental [55]: la especificación del modelo, la estimación de los parámetros del modelo y la estimación de la precisión. Los métodos de estimación, tanto de parámetros como de precisión, han sido ampliamente estudiados en el campo de la estadística. Sin embargo, durante mucho tiempo, la especificación del modelo era una tarea que no correspondía a la estadística y que se basaba principalmente en los modelos existentes en la bibliografía y en la pericia del científico para interpretar los datos experimentales.

Actualmente, la selección de modelos es un campo de investigación activo que proporciona métodos objetivos para seleccionar, entre un conjunto de modelos candidatos, cuál de ellos representa con mayor fidelidad la información contenida en la muestra experimental, siendo el más adecuado para poder hacer inferencias estadísticas sobre toda la población. Estos métodos comparten la esencia del *principio de parsimonia*, también conocido como *navaja de Ockham*: «En igualdad de condiciones, la solución más sencilla es posiblemente la correcta». Según Box y Jenkis, este principio nos indica que los modelos deben tener el menor número posible de parámetros para la adecuada representación de los datos experimentales [27]. Dicho de otro modo, dentro del contexto de la selección de modelos, el modelo óptimo debe ser un equilibrio entre modelos complejos (con un gran número de parámetros) y modelos simples (con un reducido número de parámetros). En general, un modelo con un mayor número de parámetros se ajustará mejor a los datos experimentales, caracterizando con mayor precisión el comportamiento experimental, pero también es más probable que se ajuste de manera particular a efectos espurios presentes en la muestra experimental. Por otro lado, los modelos más simples tienden a caracterizar únicamente los efectos principales del proceso de

estudio —previniendo que un sobreajuste del modelo produzca una caracterización errónea de la población— pero pueden ser insuficientes para una correcta caracterización de los efectos principales del *modelo verdadero*. En cualquier caso, el tamaño óptimo del modelo dependerá de la complejidad intrínseca del problema.

En algunas situaciones, puede ocurrir que los métodos de selección no sean capaces de proporcionar un único resultado, es decir, proporcionen varios modelos dentro del conjunto de modelos candidatos que presentan las características necesarias para ser seleccionado como la mejor aproximación. Este hecho sugiere dos interpretaciones. Por un lado, puede significar que el conjunto de modelos candidatos es inadecuado y no contiene buenos modelos. Sin embargo, por otro lado, el comportamiento del proceso real es completamente desconocido y puede que sea extremadamente complicado, por lo que obtener una representación *sencilla* mediante un modelo puede ser absolutamente inviable. La inferencia multimodelo es un mecanismo que permite abordar este tipo de situaciones y que se basa en la utilización simultánea de un conjunto de modelos, en vez de uno solo, para caracterizar un proceso y garantizar una inferencia adecuada sobre la población.

En general, los resultados obtenidos por los métodos de selección presentan una gran dependencia con la calidad del conjunto de modelos candidatos. Por lo tanto, la constitución de este conjunto de modelos es un aspecto fundamental para garantizar unos resultados concluyentes. En particular, se debe establecer cuidadosamente cuál será la estructura de los modelos (polinómica, exponencial, etc.) así como las variables explicativas que los forman.

4.1.1 Métodos de selección de modelos

Uno de los métodos de selección más simples y populares es el chequeo secuencial, basado en la comparación directa de un modelo con el resultado de añadir o eliminar una variable al modelo. El número de pasos de este método dependerá, por lo tanto, del número de variables consideradas. La implementación de este método es sencilla pero, en general, no da lugar a buenos resultados. Además, esta técnica sólo es adecuada para la comparación de modelos anidados, es decir, cuando un modelo es una simplificación de otro más general.

Los parámetros estadísticos que caracterizan la calidad del ajuste también han sido ampliamente utilizados en la selección de modelos. Sin embargo, no existe una teoría que garantice que los modelos seleccionados bajo estos criterios tengan buenas propiedades para la inferencia. Un ejemplo de utilización de estos criterios es la selección del modelo que presente un mayor valor del coeficiente de correlación ajustado (R^2 ajustado).

El método de *cross-validation* [11] ha sido utilizado como base para diversos métodos de selección de modelos, que se basan en la división de la muestra en dos conjuntos: un conjunto se utiliza para el ajuste del modelo, mientras que el otro es utilizado para su validación. Este proceso de segregación se repite, utilizando diferentes particiones, en un proceso iterativo. El principal inconveniente de estos métodos es que son computacionalmente intensos y resultan impracticables cuando la muestra o el número de modelos candidatos es grande.

Actualmente, las dos aproximaciones más extendidas de la selección de modelos se basan, respectivamente, en la teoría de la información y en la estadística bayesiana, cuyos casos más representativos son el criterio de información de Akaike [4] y el criterio de información bayesiano [148]. En ambos casos, el criterio de selección es una cantidad numérica que permite ordenar y categorizar todos los modelos candidatos, seleccionando aquel que presente el mejor valor [108]. No existe un claro indicador para establecer qué aproximación es la más adecuada en cada caso, ya que el modelo seleccionado puede diferir según el método utilizado. Sin embargo, el nivel técnico que requieren los métodos bayesianos, así como el hecho de que son computationalmente más intensos que los basados en la teoría de la información, son serias desventajas que limitan su utilización, especialmente cuando se consideran modelos con un gran número de parámetros.

4.2 El criterio de Información de Akaike (AIC)

El criterio de información de Akaike (*An Information Criterion*, AIC) proporciona un método sencillo, eficaz y objetivo de selección de modelos [4]. Este criterio establece una relación formal entre la distancia de Kullback–Leibler (paradigma de la teoría de la información) y la máxima verosimilitud (paradigma dominante en la estadística). La distancia de Kullback–Leibler puede interpretarse como la información perdida cuando se utiliza un modelo para aproximar la realidad [104]. La máxima verosimilitud permite determinar los valores de los parámetros libres de un modelo estadístico [7]. Hirotsugu Akaike encontró que se podía obtener una estimación relativa de la distancia de Kullback–Leibler utilizando la máxima verosimilitud.

El criterio de información de Akaike se define como:

$$\mathrm{AIC} = -2\log(\mathscr{L}(\widehat{\theta})) + 2K \tag{4.1}$$

donde $\log(\mathscr{L}(\widehat{\theta}))$ es el logaritmo de la máxima verosimilitud y K es el número de parámetros libres del modelo. Esta expresión proporciona una estimación relativa de la distancia entre el

modelo y el mecanismo real que genera los datos observados. Como la estimación se hace en función de los datos experimentales, esta distancia es siempre dependiente del conjunto de datos experimentales. Por lo tanto, un valor individual de AIC no es interpretable por sí solo, es decir, los valores AIC sólo tienen sentido cuando se realizan comparaciones entre modelos utilizando los mismos datos experimentales.

Aunque no justifica las bases teóricas en las que se fundamenta, es posible hacer una interpretación heurística de la expresión (4.1). Esta interpretación considera el primer término de esta ecuación, $-2\log(\mathscr{L}(\widehat{\theta}))$, como una medida de la calidad con la que el modelo se ajusta a los datos experimentales, mientras que el segundo, $+2K$, sería una *penalización* que se incrementa con la complejidad del modelo. Por ejemplo, supongamos que, a partir de un conjunto de datos experimentales, se calculan los valores AIC obtenidos para dos modelos diferentes. El menor valor de AIC indica que o bien el modelo se ajusta mejor a los datos experimentales o que es menos complejo, y en realidad una combinación de ambos factores. Por lo tanto, este criterio ofrece un valor objetivo que, de manera relativa, cuantifica simultáneamente la precisión y sencillez del modelo.

Cuando el número de parámetros (K) es muy elevado en relación con el tamaño de la muestra (n), los resultados que proporciona AIC pueden no ser satisfactorios. En estos casos se utiliza una aproximación de segundo orden:

$$\mathrm{AIC}_c = \mathrm{AIC} + \frac{2K(K+1)}{n-K-1} \tag{4.2}$$

Cuando el cociente n/K es suficientemente grande, ambos valores (AIC y AIC_c) son muy similares.

4.2.1 Selección de modelos con AIC

El criterio de información de Akaike permite asociar a cada modelo un valor numérico (AIC) que cuantifica la distancia relativa del modelo con la *realidad*. Por lo tanto, un conjunto de modelos puede ordenarse en función de los valores AIC. Utilizando el criterio de información de Akaike, el problema de selección de modelos se convierte en la búsqueda del modelo candidato que presente un menor valor AIC [30].

Por otro lado, el hecho de que los valores AIC caractericen la distancia relativa entre modelo y *realidad* permite cuantificar, para un conjunto de modelos finito, la importancia relativa de cada modelo respecto del resto de modelos. Los pesos de Akaike proporcionan el mecanismo adecuado para esta caracterización. Estos pesos se basan en las diferencias AIC.

Considerando un conjunto de modelos M, compuesto por R modelos, la diferencia AIC del modelo i se define como:

$$\Delta_i^M = \mathrm{AIC}_i - \min_{j=1,2,\ldots,R} \mathrm{AIC}_j \tag{4.3}$$

Obviamente, el *mejor* modelo cumplirá que $\Delta_i^{\mathrm{M}} = 0$. El peso de Akaike del modelo i, dentro del mismo conjunto de modelos M, es:

$$\omega_i^M = \frac{\exp\left(-\frac{1}{2}\Delta_i^M\right)}{\sum_{r=1}^{R} \exp\left(-\frac{1}{2}\Delta_r^M\right)} \tag{4.4}$$

Este valor puede ser interpretado como la probabilidad de que el modelo i sea realmente el mejor modelo en el conjunto escogido. Es evidente que estos valores están intrínsecamente relacionados con un determinado conjunto de modelos y, por lo tanto, los pesos calculados considerando un conjunto de modelos candidatos M no son comparables con los pesos calculados para un conjunto de modelos diferente.

En cualquier caso, los pesos de Akaike deben ser evaluados cuidadosamente, teniendo presente que únicamente aportan información dentro del conjunto de modelos candidatos considerado. Por lo tanto, la elección adecuada del conjunto de modelos candidatos es fundamental para que el proceso de selección proporcione un resultado significativo. Si todos los modelos considerados tienen una *calidad* pobre (es decir, los modelos no caracterizan adecuadamente el comportamiento real del sistema), el resultado final será también pobre en términos absolutos, independientemente de la información proporcionada por los pesos de Akaike.

Por otro lado, los pesos de Akaike pueden proporcionar resultados inadecuados cuando existen muchos modelos con valores AIC cercanos al mínimo. Este hecho indica que, con los datos experimentales utilizados para obtener los valores AIC, el conjunto de modelos escogido no aporta información concluyente para un adecuado proceso de selección. En otras palabras, los datos experimentales no son suficientes para caracterizar ciertos aspectos reflejados en los modelos considerados: los parámetros, los factores o la propia estructura del modelo.

Los pesos de Akaike tienen otras aplicaciones que permiten valorar la calidad del conjunto de modelos escogido mediante un análisis multimodelo. Una de sus posibles aplicaciones consiste en el cálculo de la importancia relativa de cada una de las variables predictoras —es decir, las variables independientes— del conjunto de modelos candidatos. La importancia relativa de una variable predictora j, $\omega_+^M(j)$, será la suma de los pesos de Akaike de todos los modelos del conjunto M que contienen a j como variable predictora.

4.3 Implementación del criterio de selección de modelos en TIA

En la fase de análisis del entorno TIA se ha implementado un proceso para la obtención de modelos estadísticos de aplicaciones basado en la selección de modelos mediante AIC [112, 121]. Este proceso calcula el valor AIC de cada uno de los modelos de un conjunto finito de modelos candidatos. El modelo que presente el valor AIC más bajo es propuesto como el *mejor candidato*, dentro del conjunto considerado. Además, se realiza un análisis global de todos los valores AIC del conjunto de modelos, que permite al usuario valorar la calidad del modelo seleccionado.

El conjunto de modelos candidatos (`MC`) es el elemento fundamental para garantizar la calidad del proceso de selección de modelos. A partir de la lista anidada de variables `LI`, proporcionada por el usuario, se construye la lista `LT`, que contiene los diferentes *términos* (productos de variables) con los que se construirán todos los modelos del conjunto de candidatos `MC` —la descripción del conjunto de modelos se describe detalladamente en la sección 2.5.1—.

Es responsabilidad del usuario construir un conjunto de modelos adecuado. En cualquier caso, el análisis global de los valores AIC del conjunto de modelos permite integrar este proceso de selección dentro de un proceso iterativo de refinamiento, en el que en cada iteración el usuario modifica el conjunto de modelos en base a la información proporcionada en la iteración anterior.

Una vez determinado el conjunto de modelos candidatos, se inicia el proceso de selección. En primer lugar, cada modelo del conjunto se ajusta a los datos experimentales para obtener el valor AIC_c de cada modelo. Teniendo en cuenta el mecanismo de construcción del conjunto de modelos candidatos, la mayoría de los modelos considerados son no lineales —de hecho, sólo bajo ciertas condiciones particulares se obtendrá un conjunto de modelos en el que no haya ningún modelo no lineal—. Los métodos de ajuste de modelos no lineales son más complejos y computacionalmente costosos que en el caso lineal y, en general, hacen uso de métodos numéricos iterativos para aproximar los parámetros de la función a partir de una aproximación inicial (por ejemplo, el método Gauss-Newton). En el contexto de la herramienta TIA, la utilización de métodos de resolución no lineales limitaría el número de elementos del conjunto de candidatos para obtener resultados en un tiempo razonable. Además, sería necesario que el usuario proporcionase explícitamente, para cada uno de los modelos, una estimación inicial de los parámetros que garantice la convergencia de los algoritmos. Por otro lado, como consecuencia del mecanismo de construcción, cualquier modelo candidato se puede aproximar a un modelo lineal, si se considera que cada uno de los *términos* que componen el modelo global

se corresponde con una única variable explicativa. Aunque la interpretación de los diferentes estadísticos que determinan la calidad del ajuste no sería válida en este supuesto (como, por ejemplo, R^2), la utilización de métodos de resolución lineales proporciona los mismos valores numéricos para los parámetros de los modelos, con un coste mucho menor. Sin embargo, como en la selección de modelos basada en el criterio de información de Akaike el parámetro de *calidad* de los modelos es el valor AIC, la aplicación de las técnicas de regresión lineal múltiple para obtener el ajuste a los datos experimentales proporciona unos resultados válidos en este caso.

Siguiendo la aproximación lineal, los modelos candidatos son ajustados a los datos experimentales utilizando la función `lm()` de R, una función contenida en el paquete estándar `stats`. Debido al carácter no lineal de los modelos considerados, en general, los residuos del ajuste lineal tendrán un comportamiento heterocedástico —es decir, la varianza de los residuos no será constante—. Para evitar que únicamente los valores de mayor valor absoluto determinen el comportamiento del ajuste se utiliza la aproximación *relative weighting*, que minimiza la distancia relativa entre los valores medidos y los estimados por el modelo. Asimismo, se utilizará la desviación estándar del ajuste para caracterizar el error relativo del modelo. El resultado del ajuste se utiliza para calcular el valor AIC_c mediante la función `AIC()`, que también está en el paquete `stats`. Este procedimiento se aplica a todos los modelos candidatos considerados. El modelo que presente un menor valor de AIC_c se propone como *mejor* modelo. Una vez conocidos los valores AIC_c de todos los modelos candidatos, se calculan los pesos de Akaike (ω^{MC}) y la importancia relativa de cada uno de los *términos* considerados en el proceso (ω_+^{MC}). Por otra parte, también se ofrece al usuario información acerca de los modelos con menor AIC en función de la dimensión de los modelos, siendo la dimensión el número de sus *términos*. Con esta información, el usuario tiene la capacidad de valorar si el modelo propuesto es adecuado para caracterizar el comportamiento de la aplicación bajo estudio. Además, en caso de obtener resultados insatisfactorios, esta información proporciona al usuario indicaciones para mejorar el conjunto de modelos como, por ejemplo, eliminar del análisis aquellos términos que presenten una menor importancia relativa o seleccionar un modelo cuyo comportamiento encaje dentro de un modelo teórico.

La figura 4.1 muestra el pseudocódigo del proceso de selección de modelos implementado en la fase de análisis del entorno TIA. En primer lugar, dentro de un lazo (líneas 10-17) se analiza individualmente cada modelo del conjunto $\{Candidates\}$, que contiene todos los modelos candidatos considerados. Para cada modelo se realiza el ajuste a los datos experimentales y se calcula su valor AIC asociado (líneas 11 y 12, respectivamente). El valor AIC_c

```
1  input:
2    {Candidates} = {m1, m2, m3...mM}        /* Set of candidate models */
3    {Terms} = {t1, t2, t3...tR}             /* Set of terms */
4    {data} = {d1, d2, d3...dN}              /* Experimental data */
5
6  begin
7
8    {AICc} = ∅
9
10   for ( i in 1 to M ) do
11     fit = lm( mi , {data} )
12     aic = AIC( mi , {data} , fit )
13     aicci = getAICc( mi , N , aic )
14     if ( aicci = min( {aicc} ) then
15       best.model = { mi , fit }
16     end if
17   end for
18
19   {W} = getAkaikeWeights( {AICc} )
20
21   {Wplus} = {Wp1 = 0, Wp2 = 0, Wp3 = 0...WpR = 0}
22   for ( i in 1 to M ) do
23     for ( j in 1 to T ) do
24       if ( tj ⊂ mi ) then
25         Wpj += Wi
26       end if
27     end for
28   end for
29
30   return { best.model , {W} , {Wplus} }
31 end
```

Figura 4.1: Pseudocódigo del proceso de selección de modelos en el entorno TIA

se calcula a continuación (línea 13) utilizando la función `getAICc()`, que implementa la expresión (4.2). En cada iteración se comprueba si el último valor AIC_c calculado se corresponde con el mínimo del conjunto de valores AIC_c obtenidos hasta el momento (líneas 14-16). En caso afirmativo, el último modelo se convierte, temporalmente, en el mejor candidato (*best.model*). Una vez obtenidos todos los valores AIC_c del conjunto de candidatos, se calculan los pesos de Akaike (línea 19) y la importancia relativa de cada término (líneas 21-28). El peso de Akaike de cada modelo se calcula de acuerdo con la expresión (4.4). La importancia relativa de los diferentes términos se inicializa a cero y, a continuación, se examinan los términos presentes en los diferentes modelos del conjunto de candidatos. Si el modelo *i* contiene

al término j entonces el peso de Akaike del modelo i es añadido a la importancia relativa asociada al término j. El resultado final de este proceso está compuesto por tres elementos (línea 30): el modelo seleccionado, los pesos de Akaike de los diferentes modelos candidatos y la importancia relativa de cada *término*.

El coste computacional de este proceso de selección depende del número de modelos del conjunto de candidatos y, en menor medida, del tamaño de los datos experimentales. En concreto, la mayor parte del tiempo total de ejecución del proceso se consume en el ajuste de todos los modelos. Por ejemplo, en un ordenador de sobremesa con un procesador Intel® Pentium® 4 y 3 GB de memoria principal, son necesarios 2 minutos, aproximadamente, para analizar un conjunto de 4 095 modelos (es decir, 12 términos) con datos experimentales correspondientes a 297 ejecuciones de la aplicación. El coste temporal del análisis supera las 48 horas si se considera un conjunto de candidatos con 16 777 215 modelos (es decir, 24 términos) y unos datos experimentales con 80 ejecuciones de la aplicación a modelar.

Si el tamaño del conjunto de candidatos es suficientemente grande, la memoria física del sistema puede ser insuficiente para almacenar el conjunto de candidatos y sus valores AIC_c asociados, lo que implicaría el aumento de fallos de página y accesos a la unidad de disco, con el consecuente agravamiento en el rendimiento. Para prevenir estos problemas relacionados con la memoria, se han añadido al proceso dos nuevas características que limitan el consumo de memoria. Por un lado, en cada instancia del ajuste a los datos experimentales (línea 11, en la figura 4.1), el modelo correspondiente se construye dinámicamente a partir de la lista de términos. La construcción dinámica de los modelos necesita que se defina previamente un procedimiento que asocie unívocamente el índice del lazo con un único modelo candidato. En particular, esta relación biunívoca se ha establecido asociando a cada término una potencia entera de 2, de tal manera que un modelo se construye utilizando los términos asociados a las posiciones en las que aparece un 1 en la codificación binaria del índice del lazo. Por otro lado, solo se almacenan en memoria un subconjunto de candidatos, con aquellos modelos que presenten los menores valores AIC. Si el tamaño del subconjunto es suficientemente grande, esta limitación no afecta al cálculo de los pesos de Akaike —y, por ende, de la importancia relativa—, ya que la diferencia de Akaike de los modelos que se desestiman es tan grande que el peso de Akaike asociado es prácticamente nulo. En particular, se ha observado que limitando el tamaño de este subconjunto a 10 000 modelos se garantiza el suficiente volumen de modelos para un adecuado análisis de los pesos de Akaike, sin incurrir en un consumo excesivo de memoria.

En la implementación final del método se ha introducido una opción para desestimar aquellos modelos que presenten un ajuste muy pobre en términos absolutos. En particular, esta opción permite al usuario poner un límite en el error relativo permitido, de tal manera que los modelos cuyo error sea mayor que dicho límite serán excluidos del conjunto de candidatos en el análisis final.

4.4 Primer ejemplo: *benchmark* secuencial

En este caso de estudio se muestra un ejemplo sencillo del mecanismo de modelado basado en la selección de modelos. En concreto, se utiliza el *benchmark* sintético *Whetstone*, un pequeño código secuencial que caracteriza el rendimiento de las operaciones en punto flotante de un procesador [40]. Este *benchmark* tiene un único parámetro configurable (*it*) que indica el número de repeticiones de las funciones internas del programa.

El *benchmark whetstone*, adecuadamente instrumentado, ha sido ejecutado en un procesador Intel® Xeon® Quad-Core 2,33 GHz, con 8 GB de memoria principal y sistema operativo Linux 2.6.30, utilizando diferentes valores del parámetro *it*. En concreto, se han utilizado valores de *it* desde 100 hasta 1000, con incrementos de 20.

El proceso de selección de modelos ha sido aplicado a los datos obtenidos en las distintas ejecuciones. Como, a priori, no se conoce el comportamiento del código en función del parámetro *it*, para iniciar el proceso de selección se propone la siguiente lista inicial:

$$\mathtt{LI}_{\mathrm{ws}} = \{it\}, \{it^2\}, \left\{it^4, it^5\right\}^*$$

Esta lista genera un conjunto de 63 ($2^6 - 1$) modelos candidatos. La lista completa de los candidatos considerados se muestra en la figura 4.2. En realidad, este conjunto de modelos representa a todas las posibles funciones polinómicas de quinto grado (incluyendo coeficientes nulos). El proceso de selección de modelos proporciona, como mejor modelo dentro del conjunto de candidatos, el siguiente:

$$T_{\mathrm{ws}}(\mu s) = 30it + 11{,}11it^2$$

Este modelo presenta un error de 0,4 % y su peso de Akaike es $0,2020$. La tabla 4.1 muestra la importancia relativa de cada término considerado. El término it^2 es, con diferencia, el término con mayor importancia relativa. La figura 4.3 muestra gráficamente el resultado obtenido. En esta figura se aprecia claramente que el modelo se ajusta con alta precisión a los datos experimentales.

$1: t = C_0$
$2: t = C_0 it$
$3: t = C_0 it^2$
$4: t = C_0 it^3$
$5: t = C_0 it^4$
$6: t = C_0 it^5$
$7: t = C_0 + C_1 it$
$8: t = C_0 + C_1 it^2$
$9: t = C_0 + C_1 it^3$
$10: t = C_0 + C_1 it^4$
$11: t = C_0 + C_1 it^5$
$12: t = C_0 it + C_1 it^2$
$13: t = C_0 it + C_1 it^3$
$14: t = C_0 it + C_1 it^4$
$15: t = C_0 it + C_1 it^5$
$16: t = C_0 it^2 + C_1 it^3$
$17: t = C_0 it^2 + C_1 it^4$
$18: t = C_0 it^2 + C_1 it^5$
$19: t = C_0 it^3 + C_1 it^4$
$20: t = C_0 it^3 + C_1 it^5$
$21: t = C_0 it^4 + C_1 it^5$
$22: t = C_0 + C_1 it + C_2 it^2$
$23: t = C_0 + C_1 it + C_2 it^3$
$24: t = C_0 + C_1 it + C_2 it^4$
$25: t = C_0 + C_1 it + C_2 it^5$
$26: t = C_0 + C_1 it^2 + C_2 it^3$
$27: t = C_0 + C_1 it^2 + C_2 it^4$
$28: t = C_0 + C_1 it^2 + C_2 it^5$
$29: t = C_0 + C_1 it^3 + C_2 it^4$
$30: t = C_0 + C_1 it^3 + C_2 it^5$
$31: t = C_0 + C_1 it^4 + C_2 it^5$
$32: t = C_0 it + C_1 it^2 + C_2 it^3$
$33: t = C_0 it + C_1 it^2 + C_2 it^4$
$34: t = C_0 it + C_1 it^2 + C_2 it^5$
$35: t = C_0 it + C_1 it^3 + C_2 it^4$
$36: t = C_0 it + C_1 it^3 + C_2 it^5$
$37: t = C_0 it + C_1 it^4 + C_2 it^5$
$38: t = C_0 it^2 + C_1 it^3 + C_2 it^4$
$39: t = C_0 it^2 + C_1 it^3 + C_2 it^5$
$40: t = C_0 it^2 + C_1 it^4 + C_2 it^5$
$41: t = C_0 it^3 + C_1 it^4 + C_2 it^5$
$42: t = C_0 + C_1 it + C_2 it^2 + C_3 it^3$
$43: t = C_0 + C_1 it + C_2 it^2 + C_3 it^4$
$44: t = C_0 + C_1 it + C_2 it^2 + C_3 it^5$
$45: t = C_0 + C_1 it + C_2 it^3 + C_3 it^4$
$46: t = C_0 + C_1 it + C_2 it^3 + C_3 it^5$
$47: t = C_0 + C_1 it + C_2 it^4 + C_3 it^5$
$48: t = C_0 + C_1 it^2 + C_2 it^3 + C_3 it^4$
$49: t = C_0 + C_1 it^2 + C_2 it^3 + C_3 it^5$
$50: t = C_0 + C_1 it^2 + C_2 it^4 + C_3 it^5$
$51: t = C_0 + C_1 it^3 + C_2 it^4 + C_3 it^5$
$52: t = C_0 it + C_1 it^2 + C_2 it^3 + C_3 it^4$
$53: t = C_0 it + C_1 it^2 + C_2 it^3 + C_3 it^5$
$54: t = C_0 it + C_1 it^2 + C_2 it^4 + C_3 it^5$
$55: t = C_0 it + C_1 it^3 + C_2 it^4 + C_3 it^5$
$56: t = C_0 it^2 + C_1 it^3 + C_2 it^4 + C_3 it^5$
$57: t = C_0 + C_1 it + C_2 it^2 + C_3 it^3 + C_4 it^4$
$58: t = C_0 + C_1 it + C_2 it^2 + C_3 it^3 + C_4 it^5$
$59: t = C_0 + C_1 it + C_2 it^2 + C_3 it^4 + C_4 it^5$
$60: t = C_0 + C_1 it + C_2 it^3 + C_3 it^4 + C_4 it^5$
$61: t = C_0 + C_1 it^2 + C_2 it^3 + C_3 it^4 + C_4 it^5$
$62: t = C_0 it + C_1 it^2 + C_2 it^3 + C_3 it^4 + C_4 it^5$
$63: t = C_0 + C_1 it + C_2 it^2 + C_3/p + C_4 it^4 + C_5 it^5$

Figura 4.2: Conjunto de modelos asociado a la lista $\texttt{LI}_{\text{ws}}$

Tabla 4.1: Importancia relativa de cada término para el modelo T_{ws}

	ω_+^{MC}
CTE	0.450259
it	0.4801369
it^2	0.9134028
it^3	0.2419385
it^4	0.2656774
it^5	0.2945005

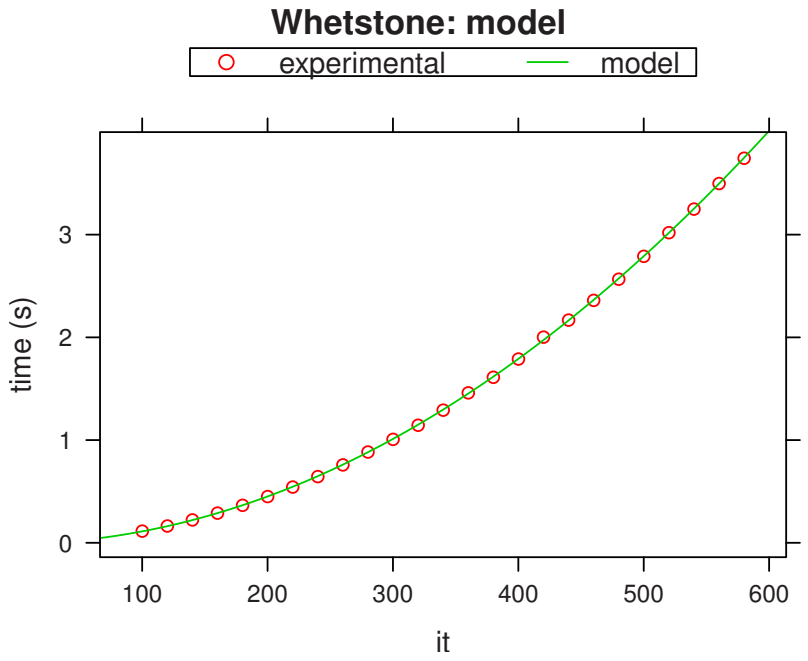


Figura 4.3: Datos experimentales y modelo del *benchmark whetstone*

En cualquier caso, el mejor ajuste a los datos experimentales se obtiene con una función polinómica de quinto orden, con cinco términos. Sin embargo, en este caso particular, es probable que un modelo con tantos términos se ajuste de manera particular a efectos espurios presentes en la muestra experimental. En cambio, el modelo proporcionado por el proceso de selección de modelos es suficientemente preciso, pero mucho más sencillo y su término principal (it^2) es el término con mayor importancia relativa —un valor próximo a la unidad, lo que indica que este modelo está presente en los modelos con mayor valor AIC—. Por lo tanto, el modelo proporcionado por el proceso de selección de modelos es más apropiado, dentro del conjunto considerado, para realizar predicciones del tiempo de ejecución, ya que únicamente caracteriza los principales factores de comportamiento de la muestra experimental.

4.5 Segundo ejemplo: comunicaciones *broadcast* en Open MPI

Utilizando el proceso de obtención de modelos estadísticos descrito en la sección 4.3, se han obtenido modelos estadísticos de rendimiento para diferentes algoritmos de comunicación del tipo *broadcast* en un sistema distribuido particular [112, 121]. En concreto, se han seleccionado cuatro algoritmos diferentes, implementados en Open MPI [62]: lineal, segmentado o *pipeline*, binario y binomial. Los resultados obtenidos han sido comparados con los desarrollos teóricos propuestos por Pjěsivac-Grbovíc et ál. [139].

4.5.1 Metodología de medida

La metodología de medida desarrollada por Nupairoj y Ni permite una mayor precisión en las medidas experimentales de comunicaciones colectivas del tipo *One-to-Many* [133]. Al igual que en otros métodos de medida —como, por ejemplo, el implementado en la colección Intel® MPI Benchmarks [88]—, el tiempo de ejecución de la función de comunicación se obtiene a partir del tiempo de ejecución de un lazo cuyo cuerpo contenga únicamente la función de comunicación. Si el número de iteraciones es suficientemente grande, se garantiza que el error del valor medido sea despreciable frente a la precisión de la medida y, al mismo tiempo, se reduce el efecto de las posibles perturbaciones debidas al establecimiento de la comunicación. Sin embargo, si no existe una sincronización entre cada iteración del lazo, este mecanismo de medida puede generar un efecto *pipeline* y problemas de contención, que introducen una componente indeterminada en la medida. Además, es necesario que la propia sincronización no adolezca de estos mismos problemas, por lo que una sincronización

global entre todos los procesos no es adecuada en muchos casos como, por ejemplo, en sistemas distribuidos. Nupairoj y Ni proponen una sincronización que involucre únicamente el último proceso del camino crítico del algoritmo y el proceso raíz (el proceso que inicia la comunicación), ya que, en este tipo de comunicaciones, el proceso raíz bloquea el inicio de la comunicación en el resto de los procesos. De este modo, se consigue una sincronización muy ligera e independiente del número de procesos, que resuelve simultáneamente los problemas de efecto *pipeline* y contención.

La metodología de Nupairoj y Ni consta de dos fases. La primera sirve para determinar el camino crítico del algoritmo de la función colectiva objeto de estudio. En la segunda fase, se realiza la medida del tiempo de ejecución de un número determinado de repeticiones de la función colectiva, utilizando la sincronización descrita anteriormente entre el proceso crítico y el proceso raíz.

La figura 4.4 muestra el pseudocódigo de la primera fase de esta metodología, responsable de buscar el proceso crítico del algoritmo de comunicación. El procedimiento de búsqueda considera que cualquier proceso puede ser el proceso crítico, por lo que el mecanismo de medida se repite para todos estos supuestos (línea 4). Antes de comenzar con el proceso de medida en cada supuesto, se ejecuta una comunicación del tipo *Many-to-One* (línea 6) que garantiza que todos los procesadores estén esperando por el proceso raíz. A continuación, se repite `N` veces la ejecución de la comunicación colectiva (línea 12). Después de cada ejecución, se realiza una sincronización (líneas 15 y 19) que consiste en el envío de un mensaje de tamaño mínimo entre el supuesto proceso crítico (`rsp`) y el proceso raíz (`root`). Cuando `rsp` no es el proceso crítico, el solapamiento de la comunicación colectiva con la sincronización puede provocar una congestión de mensajes en el proceso `root`, que adultera impredeciblemente el tiempo medido. Esta congestión se puede evitar garantizando que todos los procesos hayan terminado la comunicación colectiva antes de que se envíe el mensaje de sincronización. Prácticamente, este efecto se consigue haciendo que el proceso `rsp` espere un determinado tiempo, suficientemente largo, antes de iniciar la sincronización (línea 18). Aunque este mecanismo también distorsiona el tiempo de medida de la función de colectiva, si el tiempo de espera es constante e independiente del proceso considerado, el resultado de la búsqueda no se verá adulterado. Finalmente, se calcula el tiempo medio de ejecución de la función de comunicación en cada supuesto (línea 26), siendo el valor `Tack` el tiempo de ejecución de la sincronización. El proceso que obtenga el mayor tiempo de ejecución será el proceso crítico (línea 29), ya que cualquier proceso que no sea crítico podrá solapar el tiempo de espera con la ejecución del algoritmo en el proceso crítico.

```
1  id=MPI_Comm_rank();
2  num_proc=MPI_Comm_size();
3
4  for (rsp=0; rsp<num_proc; rsp++) {
5
6    Many-to-one();              // root finishes the last
7
8    if  (id==root) T1=get_time();
9
10   for (i=0; i<N; i++) {
11
12     One-To-Many();           // Collective execution
13
14     if (id==root) {
15       MPI_Recv(rsp);         // Wait ACK from responder
16     }
17     else if (id==rsp) {
18       dummy_loop();          // Wait and [...]
19       MPI_Send(root);        // [...] send ACK to root
20     }
21
22   }
23
24   if (id==root) T2=get_time();
25
26   Tm[rsp]=(T2-T1)/N - Tack; //  Collective completion time
27 }
28
29 CR=which(Tm==max(Tm));      // Set the Critical Responder
```

Figura 4.4: Pseudocódigo de la primera fase de la metodología de medida

La figura 4.5 muestra el pseudocódigo de la segunda fase de esta metodología, que presenta una estructura similar a la fase anterior pero, en este caso, el proceso crítico es conocido (`CR`). La función colectiva se ejecuta dentro de un lazo de `N` iteraciones, implementando la correspondiente sincronización entre el proceso crítico (línea 16) y el que inicia la comunicación (línea 13) al finalizar cada iteración. Como consecuencia, el tiempo de la comunicación colectiva (línea 23) se corresponde con el tiempo medio de una iteración del lazo menos el tiempo asociado a una sincronización (`Tack`).

```
1  id=MPI_Comm_rank();
2  num_proc=MPI_Comm_size();
3
4  Many-to-one();                  // root finishes the last
5
6  if (id==root) T1=get_time();
7
8  for (i=0; i<N; i++) {
9
10   One-To-Many();                // Collective execution
11
12   if (id==root) {
13     MPI_Recv(CR);               // Wait ACK from Critical Reponder
14   }
15   else if (id==CR) {
16     MPI_Send(root);             // Send ACK to root
17   }
18
19 }
20
21 if (id==root) T2=get_time();
22
23 Tc=(T2-T1)/N - Tack;            //  Collective completion time
```

Figura 4.5: Pseudocódigo de la segunda fase de la metodología de medida

4.5.2 Resultados experimentales

Los tiempos de ejecución de cada uno de los cuatro algoritmos considerados han sido medidos en el cluster `tegasaste`, que, como se describió en el capítulo 3, es un cluster homogéneo de 10 nodos Intel® Xeon® a 2,6 GHz y con sistema operativo Linux 2.6.16. En este caso, se ha utilizado la red de interconexión Gigabit Ethernet y la implementación Open MPI (con la componente TCP BTL) del estándar MPI. Se han realizado medidas para diferentes tamaños de mensaje y número de procesadores —no se permite más de un proceso por procesador—, con un tamaño del lazo de `N`=1 000 iteraciones. Como la implementación del estándar MPI permite la segmentación interna de los datos, este factor también ha sido considerado. La tabla 4.2 muestra, para cada parámetro experimental, los diferentes valores utilizados durante el experimento. El tiempo de ejecución de cada algoritmo ha sido medido para cada posible configuración experimental, pero manteniendo siempre que el tamaño del

Tabla 4.2: Valores numéricos considerados en los tres parámetros experimentales

Parámetro	Valores
Número de procesos (P)	2, 3, 4, 5, 6, 7, 8, 9 y 10
Tamaño de mensaje (M)	2, 4, 6, 8, 10, 12, 14 y 16 KB
Tamaño de segmento (m_s)	sin segmentar y 1, 2, 4 y 8 KB

mensaje sea un múltiplo entero del tamaño del segmento. Se ha supuesto que las condiciones de la red se mantienen constantes e iguales en cada una de las configuraciones experimentales.

En el análisis de todos los algoritmos se ha utilizado la misma lista inicial de variables:

$$\texttt{LI} = \{n_s\}, \{m_s\}, \{P, \lceil \log_2 P \rceil, \lfloor \log_2 P \rfloor\}$$

donde n_s representa el número de segmentos en los que se ha dividido el mensaje, m_s el tamaño en KB de cada segmento y P el número de procesos. Estos parámetros han sido seleccionados porque se corresponden con las variables que aparecen en los modelos teóricos basados en LogGP [139, 138], cuando los parámetros de la red (latencia, *overhead* y *gap*) se consideran constantes. Por lo tanto, de acuerdo con el mecanismo descrito en la sección 2.5.1, la lista con los diferentes términos que definen el conjunto de modelos candidatos será:

$$\begin{aligned}\texttt{LT} = \{ & \text{CTE}, \\ & n_s, m_s, P, \lceil \log_2 P \rceil, \lfloor \log_2 P \rfloor, \\ & n_s m_s, n_s P, n_s \lceil \log_2 P \rceil, n_s \lfloor \log_2 P \rfloor, m_s P, m_s \lceil \log_2 P \rceil, m_s \lfloor \log_2 P \rfloor, \\ & n_s m_s P, n_s m_s \lceil \log_2 P \rceil, n_s m_s \lfloor \log_2 P \rfloor \}\end{aligned}$$

donde el término CTE se refiere al término independiente. De este modo, todos los modelos de las expresiones analíticas de la función *broadcast* desarrolladas en [139] formarán un subconjunto dentro del conjunto de candidatos que define la lista LT. Teniendo en cuenta el número de elementos de esta lista, el conjunto de modelos (MC) tendrá un tamaño de 65 535 ($2^{16} - 1$) modelos.

La tabla 4.3 muestra los modelos obtenidos para los diferentes algoritmos utilizando el mecanismo de selección de modelos basado en AIC (que denominaremos modelos AIC), junto con su error relativo y su correspondiente peso de Akaike ($\omega^{\texttt{MC}}$). La tabla 4.4 muestra la importancia relativa de cada término, proporcionada por el proceso de selección de modelos, para los diferentes algoritmos considerados. Para una mejor interpretación de los resultados,

Tabla 4.3: Modelos obtenidos automáticamente, mediante la selección de modelos basada en AIC, para los diferentes algoritmos de *broadcast*

	Modelo AIC (selección automática)	% Error	$\omega^{\mathtt{MC}}$
Lineal	$T_{\mathrm{AIC}}(\mu s) = 229 + 9P - 18\lceil \log_2 P \rceil - 7{,}1 n_s m_s + 8{,}7 m_s n_s P$	4,9	0,011
Pipeline	$T_{\mathrm{AIC}}(\mu s) = -60 - 10 n_s - 7 m_s + 102P + 30\lfloor \log_2 P \rfloor + 50\lceil \log_2 P \rceil + + 12{,}9 n_s P + 9{,}6 m_s P - 9 m_s \lceil \log_2 P \rceil + 6{,}5 m_s n_s \lceil \log_2 P \rceil$	4,0	0,094
Binario	$T_{\mathrm{AIC}}(\mu s) = 40 - 10P + 170\lfloor \log_2 P \rfloor + 18\lceil \log_2 P \rceil + 6 n_s m_s - 7 m_s \lfloor \log_2 P \rfloor + + 2{,}7 m_s P + 3 m_s \lceil \log_2 P \rceil + 13{,}0 n_s \lfloor \log_2 P \rfloor + 3{,}3 m_s n_s \lceil \log_2 P \rceil$	5,8	0,011
Binomial	$T_{\mathrm{AIC}}(\mu s) = 50 + 9 n_s + 8 m_s + 140\lfloor \log_2 P \rfloor + 29\lceil \log_2 P \rceil + 0{,}9 m_s P - - 5 m_s \lceil \log_2 P \rceil + 14 n_s \lfloor \log_2 P \rfloor - 8 n_s \lceil \log_2 P \rceil + 7{,}7 m_s n_s \lceil \log_2 P \rceil$	6,2	0,015

Tabla 4.4: Importancia relativa de cada término, $\omega_{+}^{\mathtt{MC}}$ (sólo se muestran los valores superiores a $0{,}5$)

	Algoritmo			
Término	Lineal	Segmentado	Binario	Binomial
CTE	0,90	0,90	0,90	0,90
n_s	-	0,90	-	-
m_s	-	0,90	-	-
P	0,71	0,90	-	-
$\lceil \log_2 P \rceil$	0,65	0,90	0,51	0,63
$\lfloor \log_2 P \rfloor$	-	0,74	0,90	0,90
$n_s m_s$	0,90	-	0,90	0,69
$n_s P$	-	0,90	-	-
$n_s \lceil \log_2 P \rceil$	-	-	-	0,65
$n_s \lfloor \log_2 P \rfloor$	-	-	0,90	0,90
$m_s P$	-	0,90	0,69	-
$m_s \lceil \log_2 P \rceil$	-	0,84	0,60	-
$m_s \lfloor \log_2 P \rfloor$	-	-	0,57	-
$m_s n_s P$	0,90	-	-	-
$m_s n_s \lceil \log_2 P \rceil$	-	0,81	0,88	0,89
$m_s n_s \lfloor \log_2 P \rfloor$	-	-	-	-

únicamente se muestran los valores cuya importancia relativa es mayor de $0,5$. Además, el entorno TIA proporciona la evolución de los mejores modelos del conjunto a medida que aumenta el número de términos —es decir, con la dimensión del modelo—. Las tablas 4.5, 4.6, 4.7 y 4.8 muestran esta evolución para los algoritmos lineal, segmentado, binario y binomial, respectivamente. En estas tablas sólo se muestran los modelos con una dimensión menor que la del modelo seleccionado. Esta información permite que el usuario pueda elegir el modelo más adecuado para sus propósitos, en función del compromiso entre complejidad y error relativo. Las figuras 4.6, 4.7, 4.8 y 4.9 muestran los datos experimentales medidos (puntos) y los modelos AIC obtenidos automáticamente (líneas) en función del tamaño de mensaje (M), el tamaño del segmento (m_s) y el número de procesos (P), para los cuatro algoritmos considerados. En estas figuras se puede apreciar que los modelos obtenidos se ajustan a los datos experimentales.

Consideremos, en primer lugar, el algoritmo lineal. El valor tan bajo del peso de Akaike ($\omega^{\mathtt{MC}} = 0,011$) indica a priori que el modelo AIC obtenido tiene poca *calidad* respecto del conjunto de candidatos. Esto significa que el conjunto de candidatos no contiene buenos modelos o que muchos modelos presentan una *calidad* semejante —dicho de otro modo, se están sobreajustando modelos con términos irrelevantes—. En realidad, en este caso sucede lo segundo, ya que el mejor modelo de dimensión tres está formado por los tres términos con mayor importancia relativa (CTE, $n_s m_s$ y $n_s m_s P$), siendo el error relativo y los coeficientes muy similares, como se puede observar en la tabla 4.5. Además, en esta tabla también se puede apreciar cómo los coeficientes de estos tres términos presentan valores muy similares en los modelos correspondientes a las últimas dimensiones consideradas.

Al contrario que en el algoritmo lineal, el peso de Akaike del algoritmo segmentado es relativamente alto ($\omega^{\mathtt{MC}} = 0,094$) y, además, los diez términos que conforman el modelo seleccionado se corresponden con los diez términos con mayor importancia relativa. Ambas evidencias son indicadores de que este modelo es la mejor aproximación, dentro del conjunto seleccionado, para caracterizar el comportamiento real del sistema. Por otro lado, de la evolución de los mejores modelos del conjunto en función de la dimensión no se puede inferir ningún patrón de comportamiento que indique términos claramente dominantes. Este hecho, junto con la elevada dimensión del modelo obtenido, sugiere que el conjunto de modelos considerado no contempla adecuadamente los mecanismos del sistema real, o que este comportamiento es demasiado complejo y no puede aproximarse adecuadamente mediante modelos lineales.

Tabla 4.5: Modelos con mejor valor AIC en función de la dimensión (δ) del modelo – algoritmo lineal

δ	Modelo (μs)	Error (%)
1	$T_{\text{AIC}} = 11,7 m_s n_s P$	26,5
2	$T_{\text{AIC}} = 199 + 7,96 m_s n_s P$	7,5
3	$T_{\text{AIC}} = 229 - 8,0 m_s n_s + 8,9 m_s n_s P$	5,0
4	$T_{\text{AIC}} = 229 - 7,0 m_s n_s + 9,6 m_s n_s P - 2 m_s n_s \lceil \log_2 P \rceil$	4,9
5	$T_{\text{AIC}} = 229 + 9P - 18 \lceil \log_2 P \rceil - 7,1 n_s m_s + 8,7 m_s n_s P$	4,9

Tabla 4.6: Modelos con mejor valor AIC en función de la dimensión (δ) del modelo – algoritmo segmentado

δ	Modelo (μs)	Error (%)
1	$T_{\text{AIC}} = 202P$	21,7
2	$T_{\text{AIC}} = 139P + 7,6 m_s n_s P$	13,7
3	$T_{\text{AIC}} = 131P + 7,3 n_s P + 5,6 m_s n_s P$	8,6
4	$T_{\text{AIC}} = -105 + 155P + 14,5 n_s \lceil \log_2 P \rceil + 5,6 m_s n_s P$	6,0
5	$T_{\text{AIC}} = -107 + 148P + 3,8 m_s P + 10,9 n_s P + 6,1 m_s n_s \lceil \log_2 P \rceil$	4,8
6	$T_{\text{AIC}} = -12,9 m_s - 13,6 n_s + 123P + 6,7 m_s P + 13,9 n_s P + 6,4 m_s n_s \lceil \log_2 P \rceil$	4,3
7	$T_{\text{AIC}} = -34 - 10 m_s - 10 n_s + 131P + 5,9 m_s P + 13,0 n_s P + 6,4 m_s n_s \lceil \log_2 P \rceil$	4,2
8	$T_{\text{AIC}} = -33 - 8 m_s - 11 n_s + 131P + 5,4 m_s P + 7,0 n_s P + 13 n_s \lceil \log_2 P \rceil + 3,2 m_s n_s P$	4,1
9	$T_{\text{AIC}} = -47 - 7 m_s - 10 n_s + 114P + 40 \lceil \log_2 P \rceil + 9,6 m_s P - 9 m_s \lceil \log_2 P \rceil + + 12,9 n_s P + 6,5 m_s n_s \lceil \log_2 P \rceil$	4,0
10	$T_{\text{AIC}} = -60 - 10 n_s - 7 m_s + 102P + 30 \lfloor \log_2 P \rfloor + 50 \lceil \log_2 P \rceil + 12,9 n_s P + + 9,6 m_s P - 9 m_s \lceil \log_2 P \rceil + 6,5 m_s n_s \lceil \log_2 P \rceil$	4,0

Tabla 4.7: Modelos con mejor valor AIC en función de la dimensión (δ) del modelo – algoritmo binario

δ	Modelo (μs)	Error (%)
1	$T_{\text{AIC}} = 317\lfloor \log_2 P \rfloor$	23,4
2	$T_{\text{AIC}} = 212\lfloor \log_2 P \rfloor + 10 m_s n_s \lceil \log_2 P \rceil$	12,6
3	$T_{\text{AIC}} = 16,3 n_s + 202\lfloor \log_2 P \rfloor + 7,9 m_s n_s \lceil \log_2 P \rceil$	8,0
4	$T_{\text{AIC}} = 90 + 155\lfloor \log_2 P \rfloor + 9,0 n_s \lfloor \log_2 P \rfloor + 7,7 m_s n_s \lceil \log_2 P \rceil$	6,9
5	$T_{\text{AIC}} = 90 + 145\lfloor \log_2 P \rfloor + 3,7 m_s \lceil \log_2 P \rceil + 13,3 n_s \lfloor \log_2 P \rfloor + 5,5 m_s n_s \lceil \log_2 P \rceil$	6,1
6	$T_{\text{AIC}} = 66 + 3,3 m_s n_s + 155\lfloor \log_2 P \rfloor + 3,5 m_s \lceil \log_2 P \rceil + 13,0 n_s \lfloor \log_2 P \rfloor + 4,5 m_s n_s \lceil \log_2 P \rceil$	6,0
7	$T_{\text{AIC}} = 68 - 4m_s + 5 m_s n_s + 155\lfloor \log_2 P \rfloor + 5,2 m_s \lceil \log_2 P \rceil +$ $+ 12,9 n_s \lfloor \log_2 P \rfloor + 3,8 m_s n_s \lceil \log_2 P \rceil$	5,9
8	$T_{\text{AIC}} = 55 + 5 m_s n_s + 163\lfloor \log_2 P \rfloor + 1,6 m_s P - 6 m_s \lfloor \log_2 P \rfloor +$ $+ 4 m_s \lceil \log_2 P \rceil + 12,9 n_s \lfloor \log_2 P \rfloor + 3,7 m_s n_s \lceil \log_2 P \rceil$	5,8
9	$T_{\text{AIC}} = 57 - 2m_s + 6 m_s n_s + 162\lfloor \log_2 P \rfloor + 1,2 m_s P - 5 m_s \lfloor \log_2 P \rfloor +$ $+ 5 m_s \lceil \log_2 P \rceil + 12,8 n_s \lfloor \log_2 P \rfloor + 3,4 m_s n_s \lceil \log_2 P \rceil$	5,8
10	$T_{\text{AIC}} = 40 - 10P + 170\lfloor \log_2 P \rfloor + 18\lceil \log_2 P \rceil + 6 n_s m_s + 2,7 m_s P - 7 m_s \lfloor \log_2 P \rfloor +$ $+ 3 m_s \lceil \log_2 P \rceil + 13,0 n_s \lfloor \log_2 P \rfloor + 3,3 m_s n_s \lceil \log_2 P \rceil$	5,8

Tabla 4.8: Modelos con mejor valor AIC en función de la dimensión (δ) del modelo – algoritmo binomial

δ	Modelo (μs)	Error (%)
1	$T_{\text{AIC}} = 313\lfloor\log_2 P\rfloor$	23,1
2	$T_{\text{AIC}} = 213\lfloor\log_2 P\rfloor + 9,3m_s n_s\lceil\log_2 P\rceil$	13,4
3	$T_{\text{AIC}} = 17,7n_s + 203\lfloor\log_2 P\rfloor + 7,3m_s n_s\lceil\log_2 P\rceil$	8,0
4	$T_{\text{AIC}} = 100 + 151\lfloor\log_2 P\rfloor + 9,4n_s\lfloor\log_2 P\rfloor + 7,2m_s n_s\lceil\log_2 P\rceil$	6,7
5	$T_{\text{AIC}} = 62 + 5,2m_s n_s + 168\lfloor\log_2 P\rfloor + 9,2n_s\lfloor\log_2 P\rfloor + 5,5m_s n_s\lceil\log_2 P\rceil$	6,3
6	$T_{\text{AIC}} = 70 + 5m_s n_s + 164\lfloor\log_2 P\rfloor + 11n_s\lfloor\log_2 P\rfloor - 2n_s\lceil\log_2 P\rceil + 5,8m_s n_s\lceil\log_2 P\rceil$	6,3
7	$T_{\text{AIC}} = 60 + 6m_s n_s + 146\lfloor\log_2 P\rfloor + 19\lceil\log_2 P\rceil +$ $+ 14n_s\lfloor\log_2 P\rfloor - 4n_s\lceil\log_2 P\rceil + 5,2m_s n_s\lceil\log_2 P\rceil$	6,3
8	$T_{\text{AIC}} = 50 + 6m_s n_s + 140\lfloor\log_2 P\rfloor + 23\lceil\log_2 P\rceil - 1,9n_s P +$ $+ 15n_s\lfloor\log_2 P\rfloor + 1,0m_s n_s P + 3m_s n_s\lceil\log_2 P\rceil$	6,3
9	$T_{\text{AIC}} = 40 + 7m_s + 9n_s + 148\lfloor\log_2 P\rfloor + 23\lceil\log_2 P\rceil - 3m_s\lceil\log_2 P\rceil +$ $+ 14n_s\lfloor\log_2 P\rfloor - 7n_s\lceil\log_2 P\rceil + 7,7m_s n_s\lceil\log_2 P\rceil$	6,2
10	$T_{\text{AIC}} = 50 + 9n_s + 8m_s + 140\lfloor\log_2 P\rfloor + 29\lceil\log_2 P\rceil + 0,9m_s P - 5m_s\lceil\log_2 P\rceil +$ $+ 14n_s\lfloor\log_2 P\rfloor - 8n_s\lceil\log_2 P\rceil + 7,7m_s n_s\lceil\log_2 P\rceil$	6,2

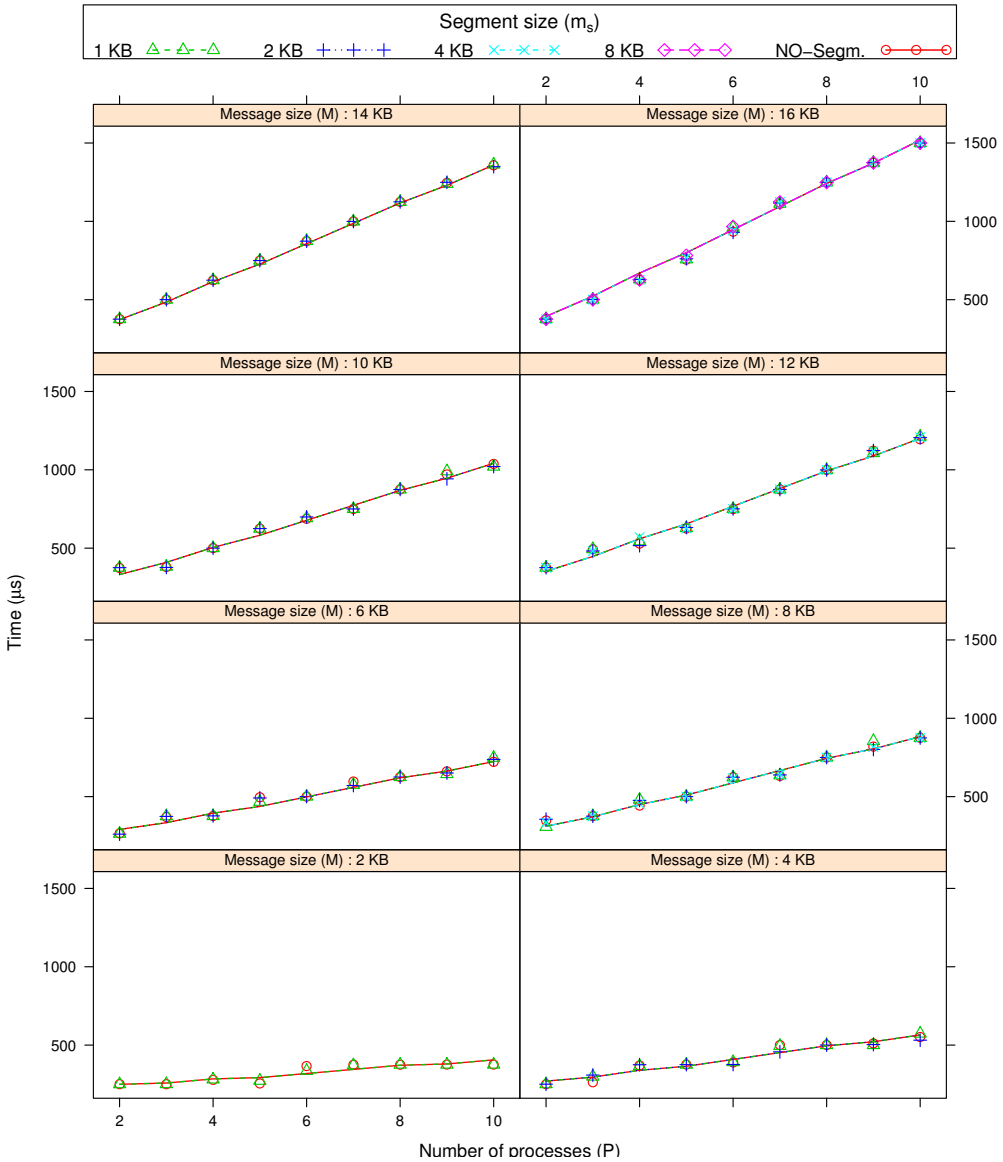


Figura 4.6: Datos experimentales (puntos) y modelo AIC obtenido automáticamente (líneas) del algoritmo lineal de *broadcast*

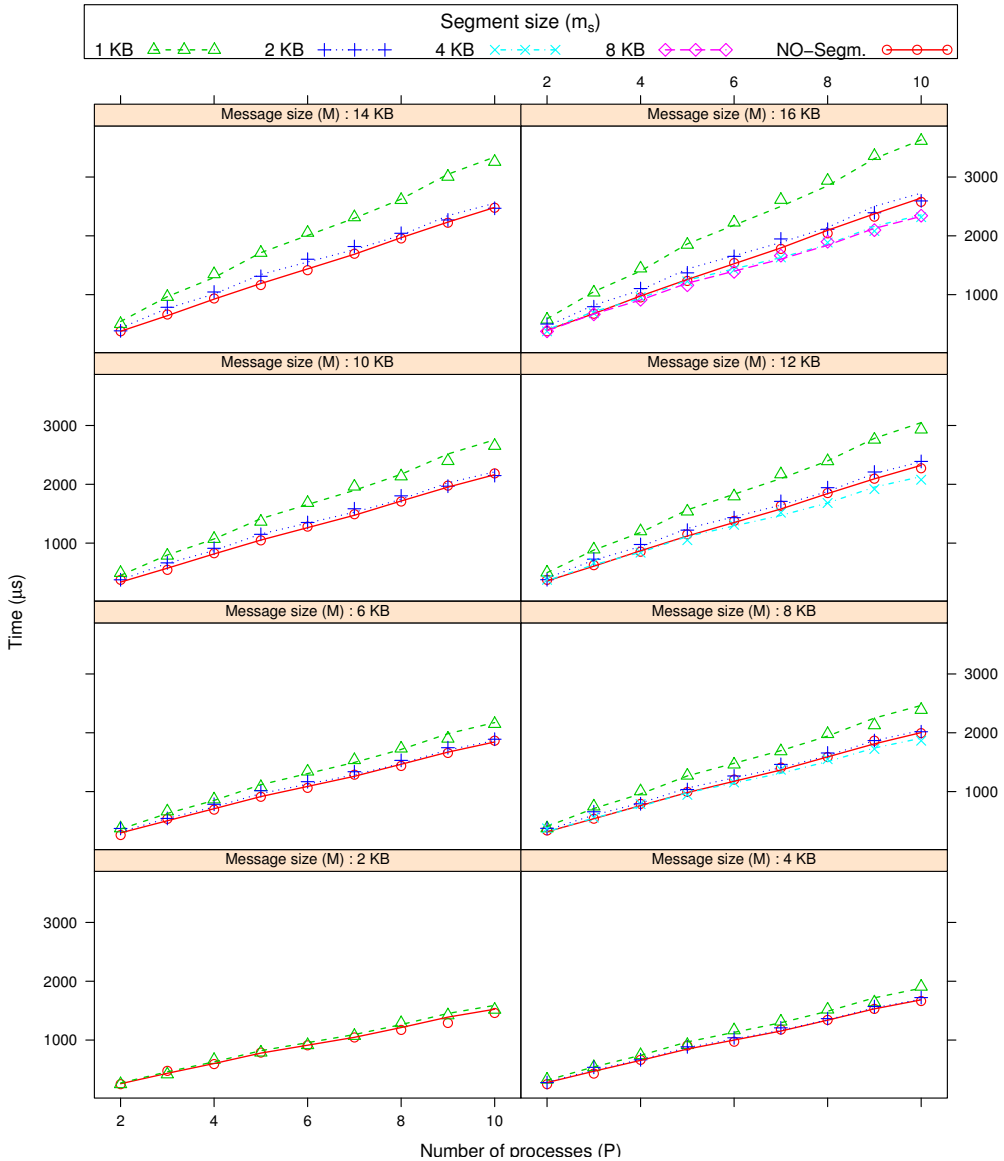


Figura 4.7: Datos experimentales (puntos) y modelo AIC obtenido automáticamente (líneas) del algoritmo segmentado de *broadcast*

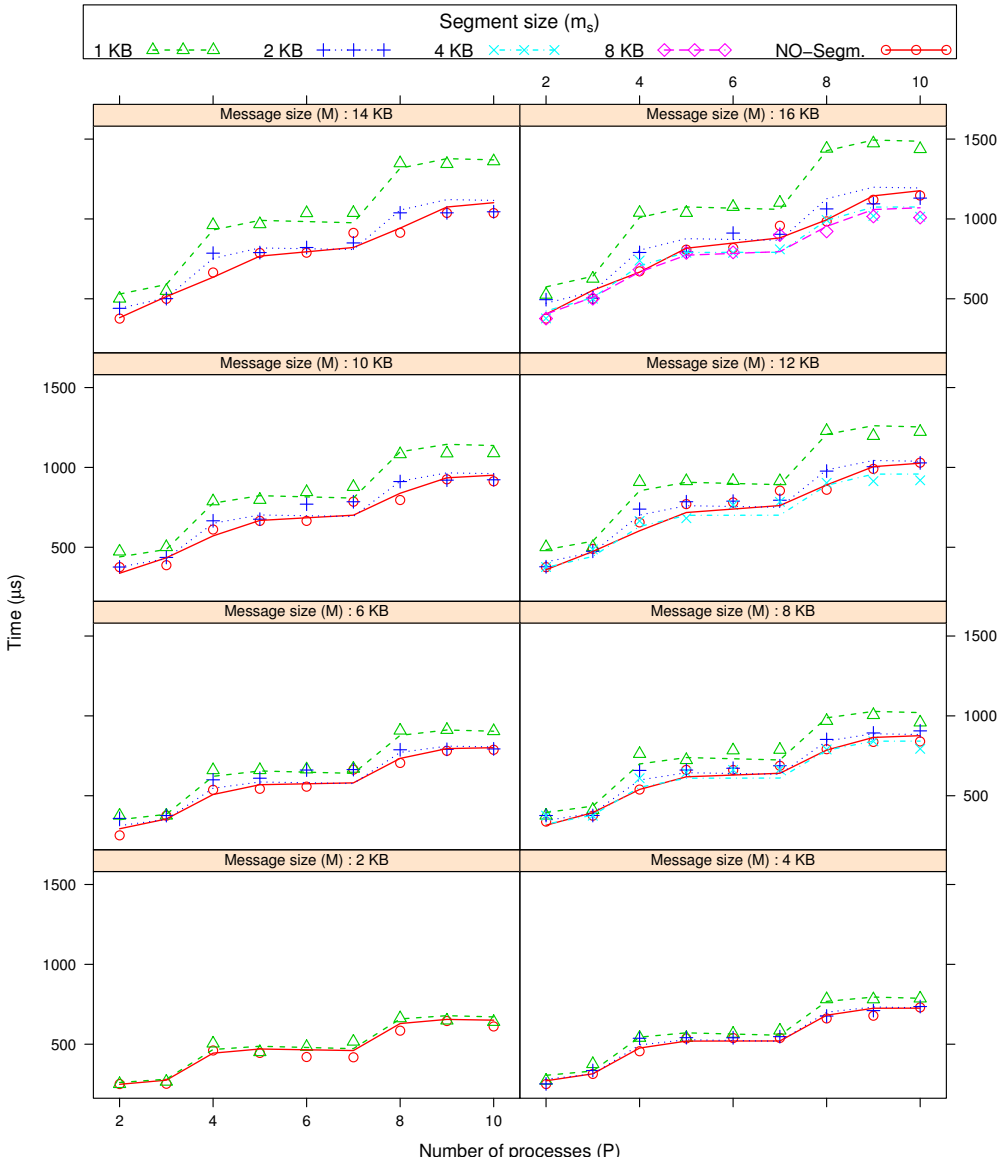


Figura 4.8: Datos experimentales (puntos) y modelo AIC obtenido automáticamente (líneas) del algoritmo binario de *broadcast*

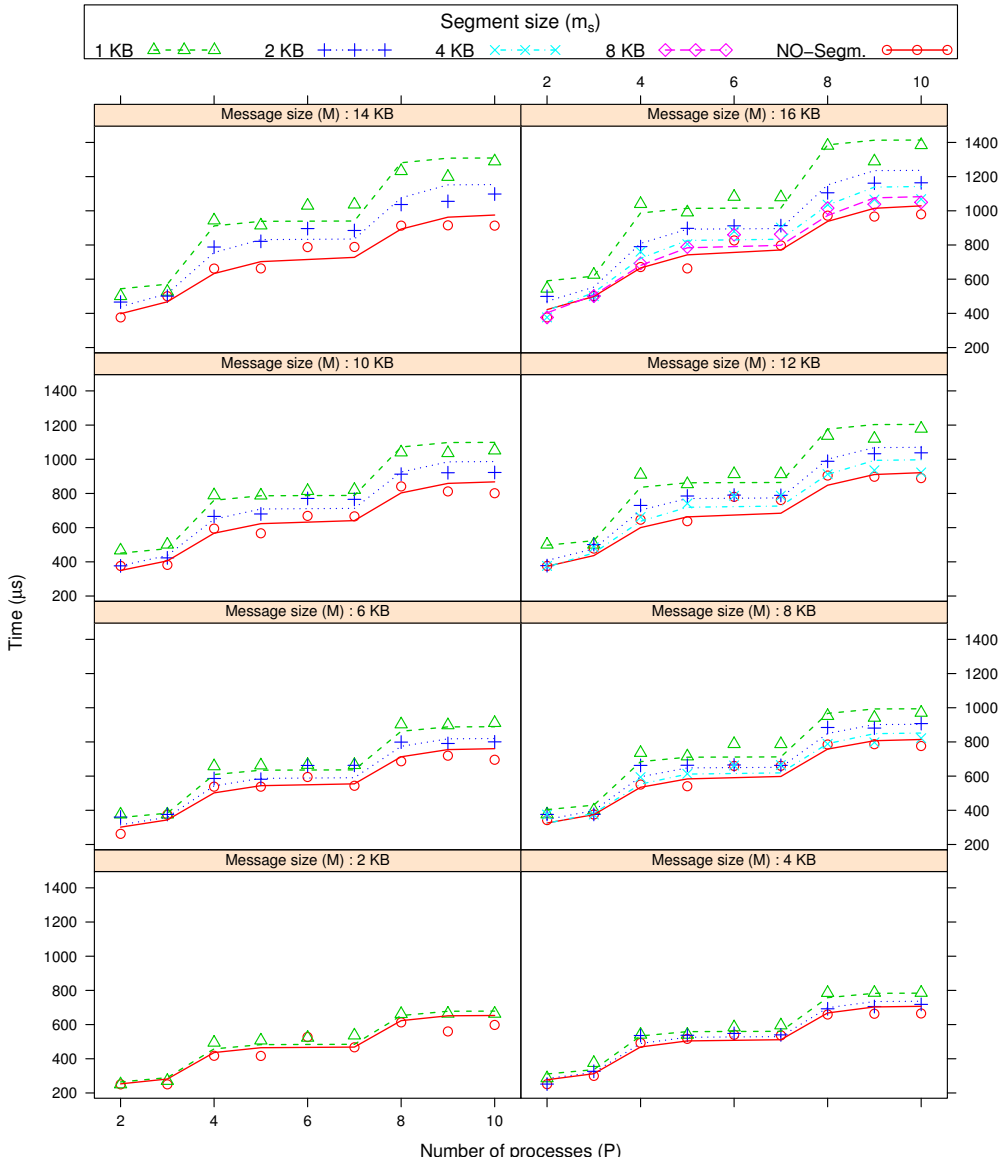


Figura 4.9: Datos experimentales (puntos) y modelo AIC obtenido automáticamente (líneas) del algoritmo binomial de *broadcast*

Tabla 4.9: Modelos AIC, obtenidos tras el análisis de los resultados del proceso de selección, para los diferentes algoritmos de *broadcast*

	Modelo AIC (selección experta)	Error (%)
Lineal	$T_{\mathrm{AIC}}(\mu s) = 229 - 8,0 n_s m_s + 8,9 m_s n_s P$	5,0
Pipeline	$T_{\mathrm{AIC}}(\mu s) = -60 - 10 n_s - 7 m_s + 102 P + 30 \lfloor \log_2 P \rfloor + 50 \lceil \log_2 P \rceil + + 12,9 n_s P + 9,6 m_s P - 9 m_s \lceil \log_2 P \rceil + 6,5 m_s n_s \lceil \log_2 P \rceil$	4,0
Binario	$T_{\mathrm{AIC}}(\mu s) = 66 + 3,3 m_s n_s + 155 \lfloor \log_2 P \rfloor + 3,5 m_s \lceil \log_2 P \rceil + 13,0 n_s \lfloor \log_2 P \rfloor + + 4,5 m_s n_s \lceil \log_2 P \rceil$	6,1
Binomial	$T_{\mathrm{AIC}}(\mu s) = 100 + 151 \lfloor \log_2 P \rfloor + 9,4 n_s \lfloor \log_2 P \rfloor + 7,2 m_s n_s \lceil \log_2 P \rceil$	6,7

Los modelos obtenidos para los algoritmos binario y binomial presentan características muy similares, ya que ambos modelos tienen una dimensión elevada (10 términos) y el peso de Akaike de ambos es relativamente bajo. Sin embargo, en ambos casos, un análisis conjunto de la importancia relativa de los distintos términos y de la evolución del mejor modelo en función del número de términos permite reducir la dimensión de los modelos. Por ejemplo, en el algoritmo binario, los cinco términos con una importancia relativa próxima a 0,9 forman parte del mejor modelo de dimensión seis. Un análisis equivalente puede realizarse para algoritmo binomial, ya que los cuatro términos con mayor importancia relativa conforman el modelo AIC de dimensión cuatro. En ambos casos, la diferencia con el error relativo de los modelos seleccionados sería mínima.

Teniendo en cuenta estas valoraciones, los modelos AIC obtenidos tras un análisis de los resultados de la selección de modelos se muestran en la tabla 4.9. Las figuras 4.10, 4.11, 4.12 y 4.13 muestran el ajuste de estos modelos a los datos experimentales. Como se puede apreciar en estas figuras, la precisión de estos modelos seleccionados tras el análisis (tabla 4.9) es similar a la de los modelos AIC obtenidos automáticamente (tabla 4.3).

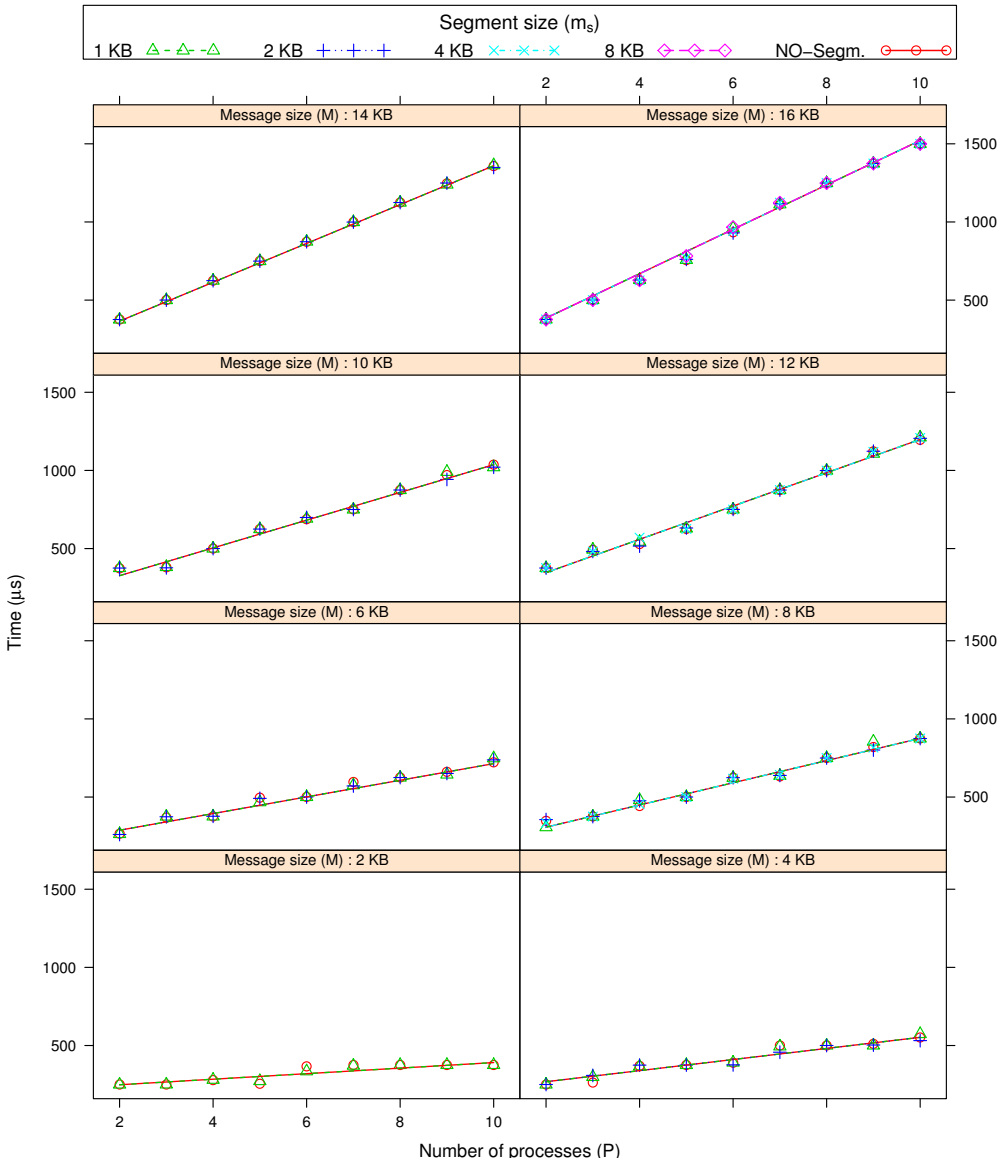


Figura 4.10: Datos experimentales (puntos) y modelo AIC (líneas) del algoritmo lineal de *broadcast*

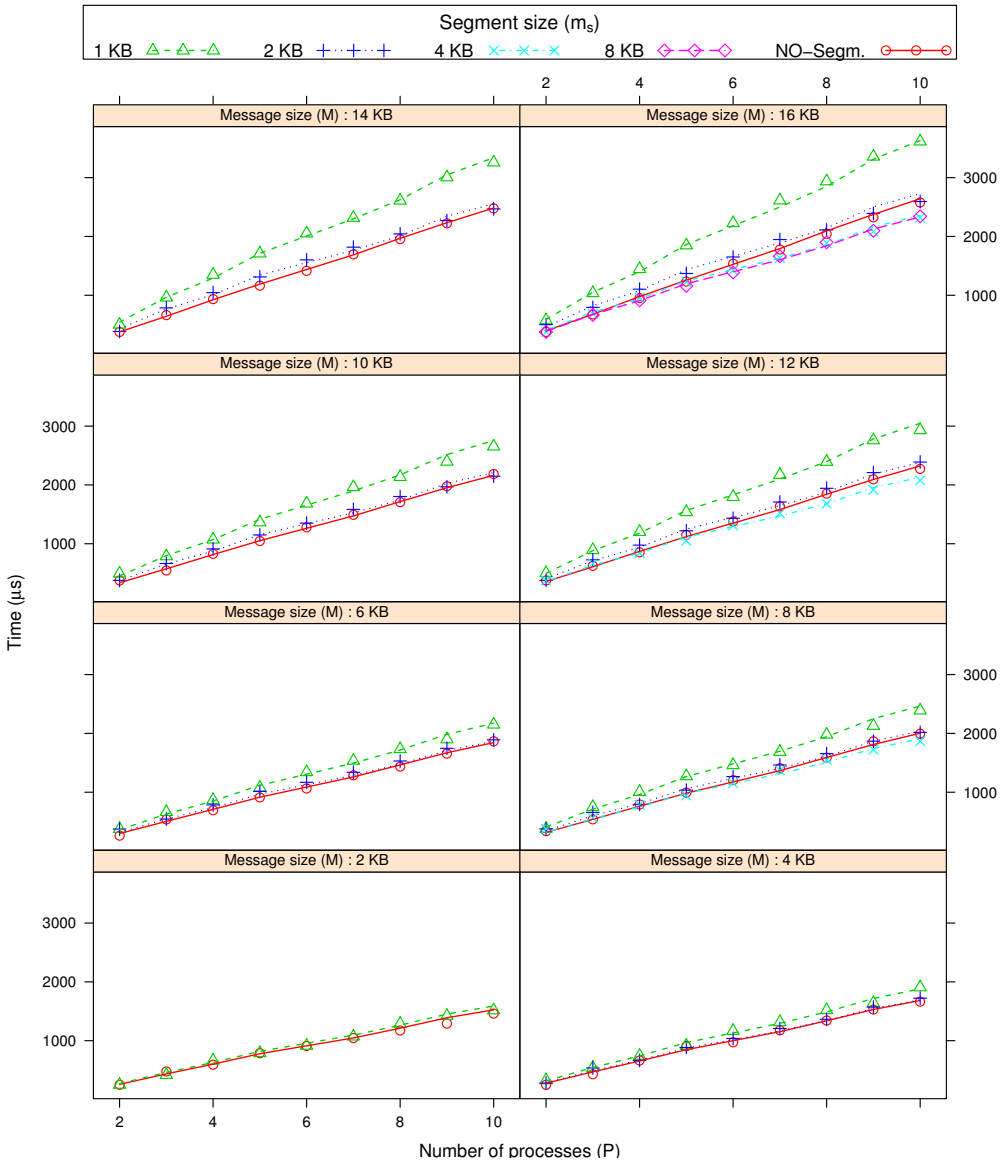


Figura 4.11: Datos experimentales (puntos) y modelo AIC (líneas) del algoritmo segmentado de *broadcast*

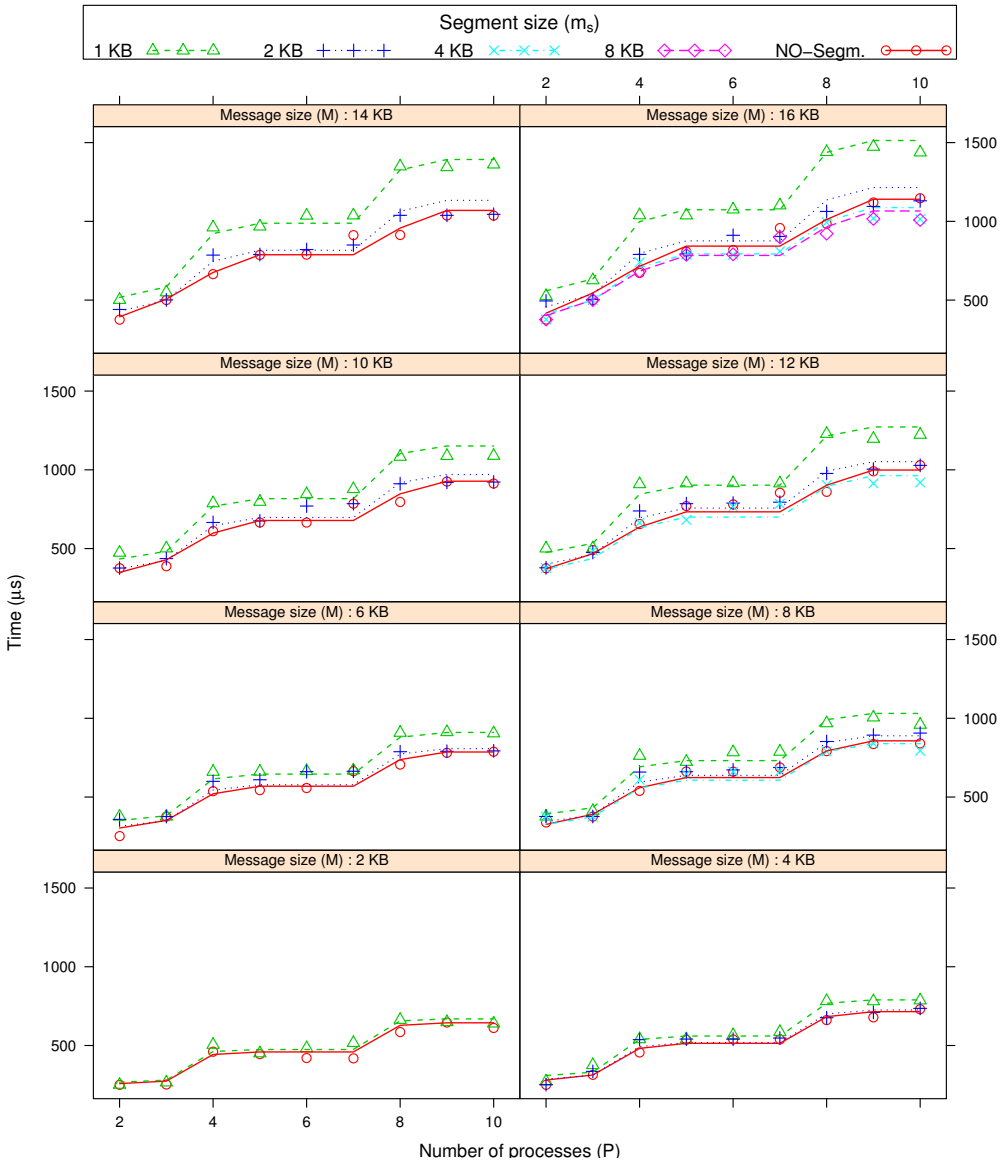


Figura 4.12: Datos experimentales (puntos) y modelo AIC (líneas) del algoritmo binario de *broadcast*

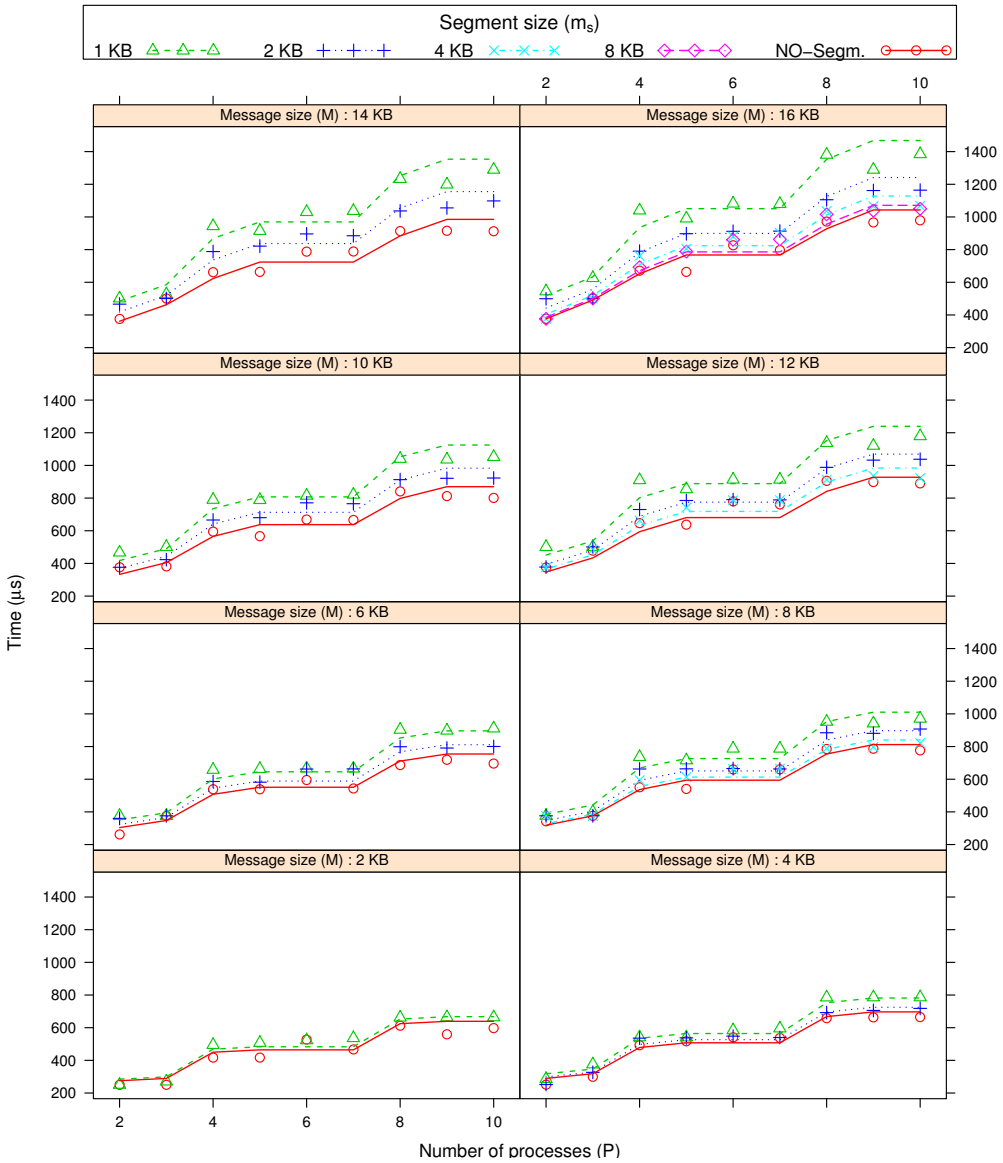


Figura 4.13: Datos experimentales (puntos) y modelo AIC (líneas) del algoritmo binomial de *broadcast*

4.5.3 Validación

Las expresiones de los modelos teóricos desarrollados por Pjěsivac-Grbovíc et ál., correspondientes a los cuatro algoritmos de *broadcast* considerados, se resumen en la tabla 4.10. Estos modelos suponen que el comportamiento de la red está caracterizado por el modelo LogGP y, por lo tanto, es esperable que no sean extremadamente precisos, ya que el modelo LogGP es, en realidad, una aproximación del mecanismo real que rige las comunicaciones. Además, en el desarrollo de las expresiones teóricas no se han considerado posibles factores relacionados con la implementación particular de los algoritmos como, por ejemplo, el posible solape de los diferentes envíos cuando se segmenta un mensaje, la congestión de mensajes en un único receptor o la gestión de los datos para el correcto enrutamiento de los mensajes.

Los modelos teóricos de comportamiento de los algoritmos de *broadcast* en el cluster `tegasaste` se obtienen sustituyendo los valores de los parámetros LogGP en las expresiones teóricas. Teniendo en cuenta que, si se ignora el parámetro **O**, los modelos LogGP y LoOgGP son análogos, las expresiones teóricas particulares pueden obtenerse a partir de los valores de los parámetros LoOgGP. En la tabla 4.11 se muestan los valores de los parámetros LoOgGP de la configuración utilizada en el cluster `tegasaste`, medidos con el driver NGMPI de TIA (sección 3.3.1). Las expresiones particulares de los modelos teóricos en este cluster, para los diferentes algoritmos de *broadcast*, se muestran en la tabla 4.12. En esta tabla también se muestra el error relativo de estas expresiones respecto de los datos experimentales.

Las figuras 4.14, 4.15, 4.16 y 4.17 muestran los valores de los modelos teóricos (líneas) frente a los datos experimentales (puntos), para los algoritmos de *broadcast* considerados. En estas figuras se puede observar cómo los modelos teóricos subestiman el tiempo de ejecución en la práctica totalidad de los casos. Además, la predicción de los modelos teóricos acerca del comportamiento de los algoritmos en función del número de segmentos utilizados no se corresponde con el comportamiento de los datos experimentales. Por ejemplo, los distintos modelos estiman que si aumenta el número de segmentos —y, por ende, disminuye el tamaño del segmento— el tiempo de ejecución debería disminuir, independientemente del algoritmo. Sin embargo, en los resultados experimentales, el menor tamaño de segmento utilizado (1 KB) tiene asociado el mayor tiempo de ejecución en los algoritmos segmentado, binario y binomial, independientemente del tamaño global del mensaje.

Si se comparan las expresiones teóricas (tabla 4.12) con los modelos obtenidos mediante selección de modelos (tabla 4.9), se observa que la precisión de los modelos AIC es mucho mayor (en cualquier caso, menor del 10 %). Además, como se puede apreciar en las figuras,

Tabla 4.10: Modelos teóricos basados en LogGP para los diferentes algoritmos de *broadcast*

	Modelo teórico LogGP
Lineal	$T_{\text{teó}} = (\mathbf{L}+2\mathbf{o}-\mathbf{g})+(\mathbf{G}-\mathbf{g})n_s-\mathbf{G}n_sm_s+(\mathbf{g}-\mathbf{G})n_sP+\mathbf{G}n_sm_sP$
Pipeline	$T_{\text{teó}} = (-\mathbf{L}-3\mathbf{o}+\mathbf{G}-\mathbf{g})+(\mathbf{g}-\mathbf{G}+\mathbf{o})n_s-2\mathbf{G}m_s+(\mathbf{L}+2\mathbf{o}-\mathbf{G})P+$ $+\mathbf{G}n_sm_s+\mathbf{G}m_sP$
Binario	$T_{\text{teó}} = (2\mathbf{G}-\mathbf{o}-2\mathbf{g})+(\mathbf{o}+2\mathbf{g}-2\mathbf{G})n_s-2\mathbf{G}m_s+2\mathbf{G}n_sm_s+$ $+(\mathbf{L}+\mathbf{g}+2\mathbf{o}-2\mathbf{G})\lfloor\log_2 P\rfloor+2\mathbf{G}m_s\lfloor\log_2 P\rfloor$
Binomial	$T_{\text{teó}} = (\mathbf{L}+2\mathbf{o}-\mathbf{g})\lceil\log_2 P\rceil+(\mathbf{g}-\mathbf{G})n_s\lceil\log_2 P\rceil+\mathbf{G}n_sm_s\lceil\log_2 P\rceil$

Tabla 4.11: Valores de los parámetros LoOgGP en el cluster `tegasaste`, medidos con el driver NGMPI

L (μs)	**o** (μs)	**O** (μs/KB)	**g** (μs)	**G** (μs/KB)
124,1	9,97	1,30	1,86	8,49

Tabla 4.12: Modelos teóricos basados en LogGP para los diferentes algoritmos de *broadcast* en el cluster `tegasaste`

	Modelo teórico LogGP (`tegasaste`)	Error (%)
Lineal	$T_{\text{teó}}(\mu s) = 142{,}18+6{,}63n_s-8{,}49n_sm_s-6{,}63n_sP+8{,}49n_sm_sP$	16,2
Pipeline	$T_{\text{teó}}(\mu s) = -147{,}38+3{,}34n_s-16{,}98m_s+135{,}55P+8{,}49n_sm_s+8{,}49m_sP$	14,6
Binario	$T_{\text{teó}}(\mu s) = 3{,}29-3{,}29n_s-16{,}98m_s+16{,}98n_sm_s+128{,}92\lfloor\log_2 P\rfloor+16{,}98m_s\lfloor\log_2 P\rfloor$	16,8
Binomial	$T_{\text{teó}}(\mu s) = 142{,}18\lceil\log_2 P\rceil-6{,}63n_s\lceil\log_2 P\rceil+8{,}49n_sm_s\lceil\log_2 P\rceil$	20,7

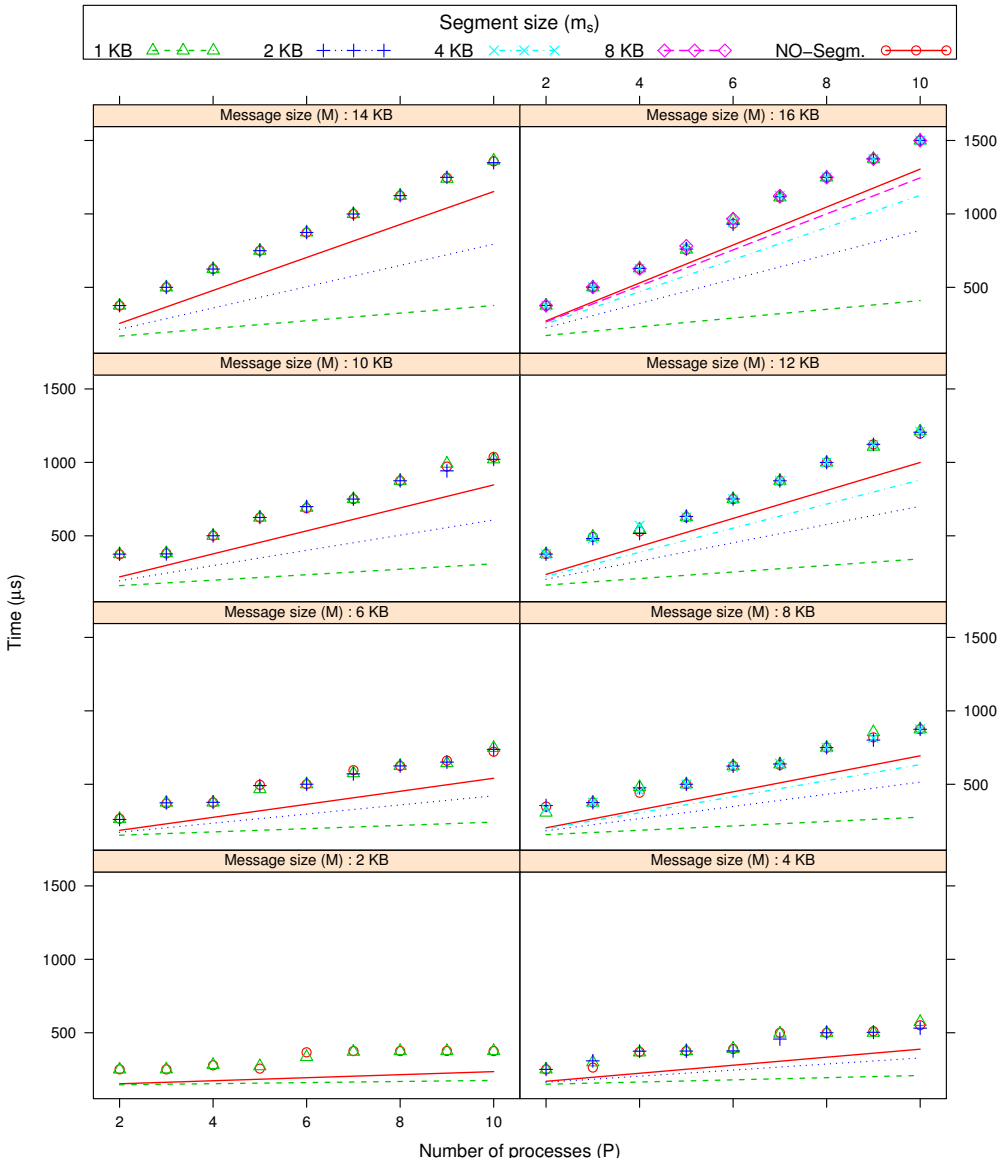


Figura 4.14: Datos experimentales (puntos) y modelo teórico (líneas) del algoritmo lineal de *broadcast*

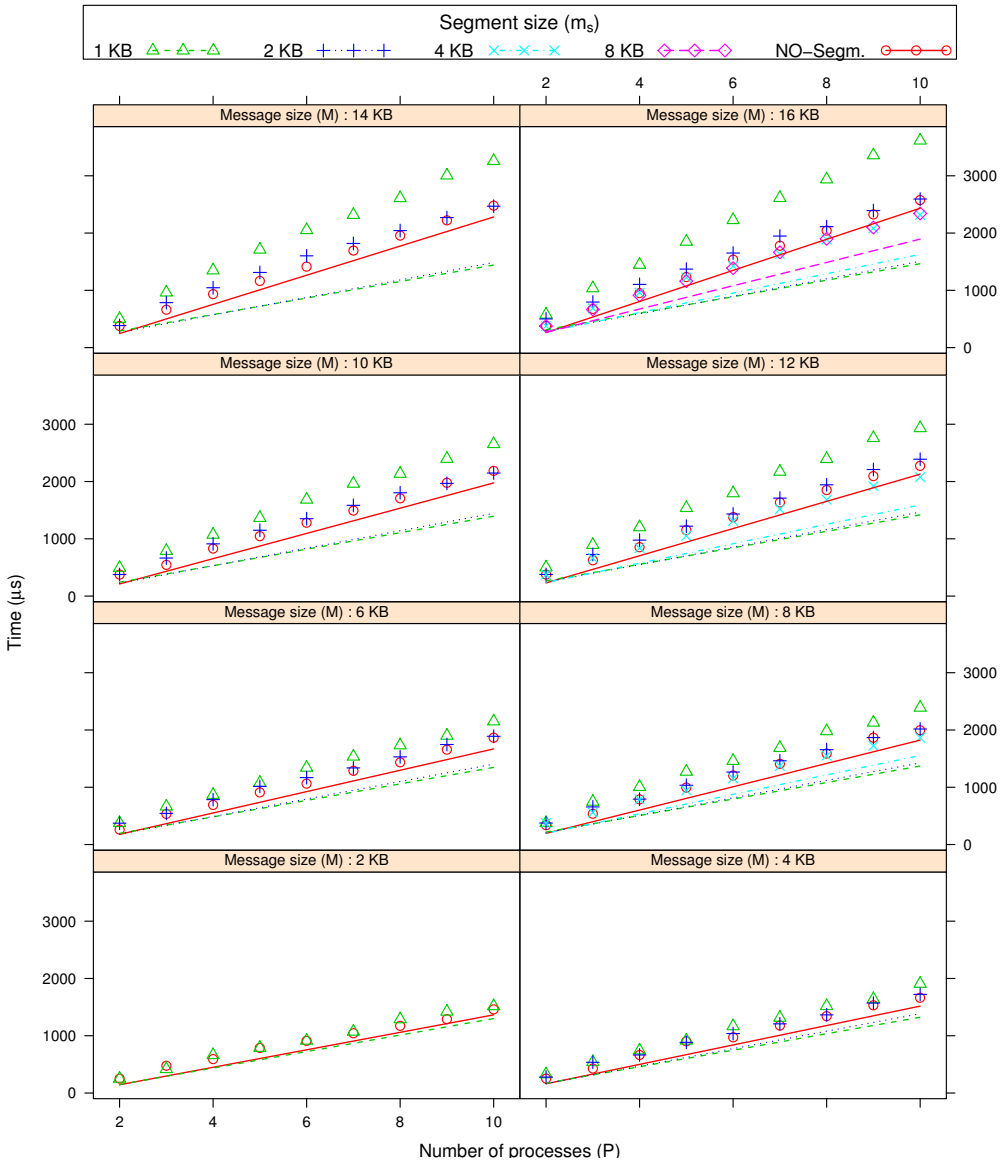


Figura 4.15: Datos experimentales (puntos) y modelo teórico (líneas) del algoritmo segmentado de *broadcast*

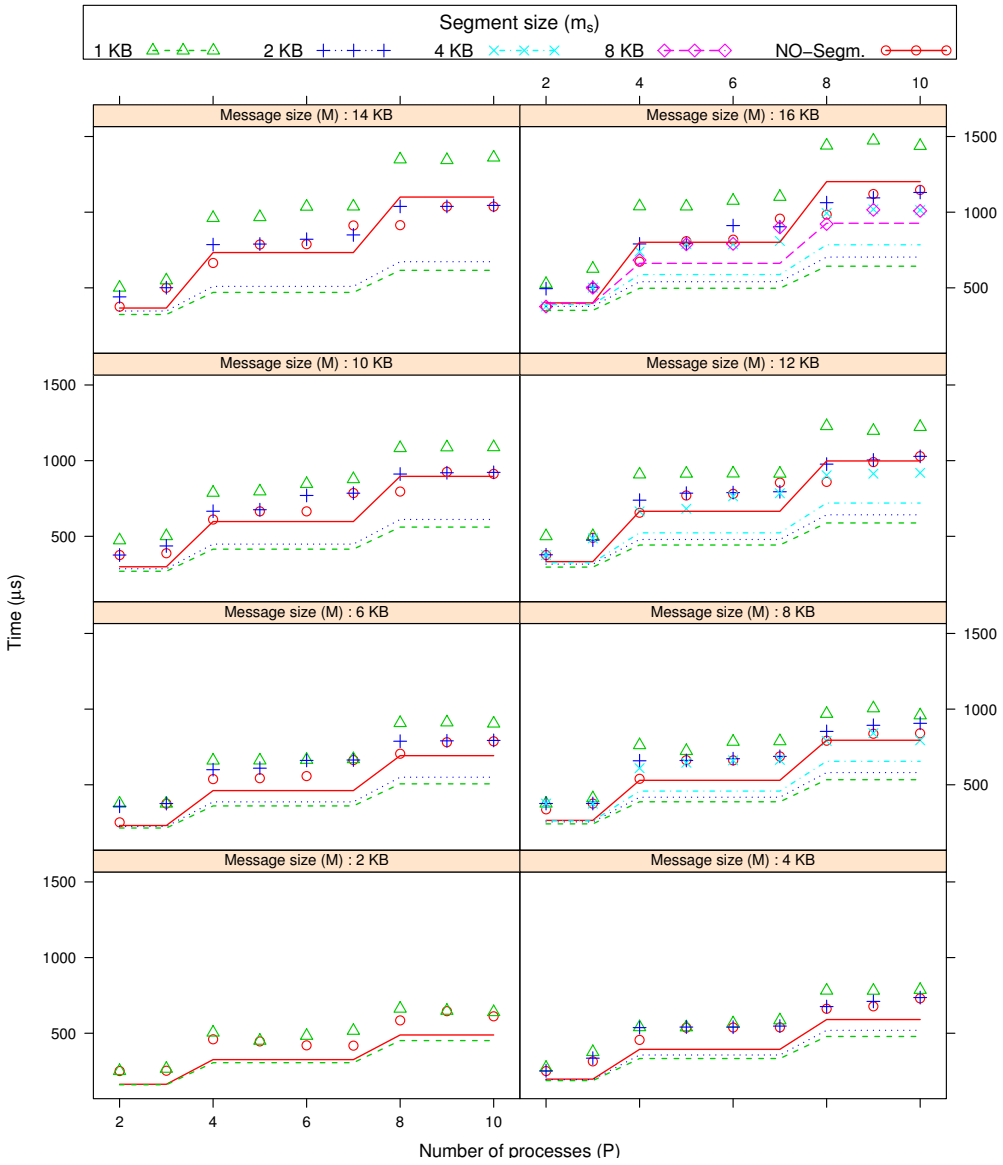


Figura 4.16: Datos experimentales (puntos) y modelo teórico (líneas) del algoritmo binario de *broadcast*

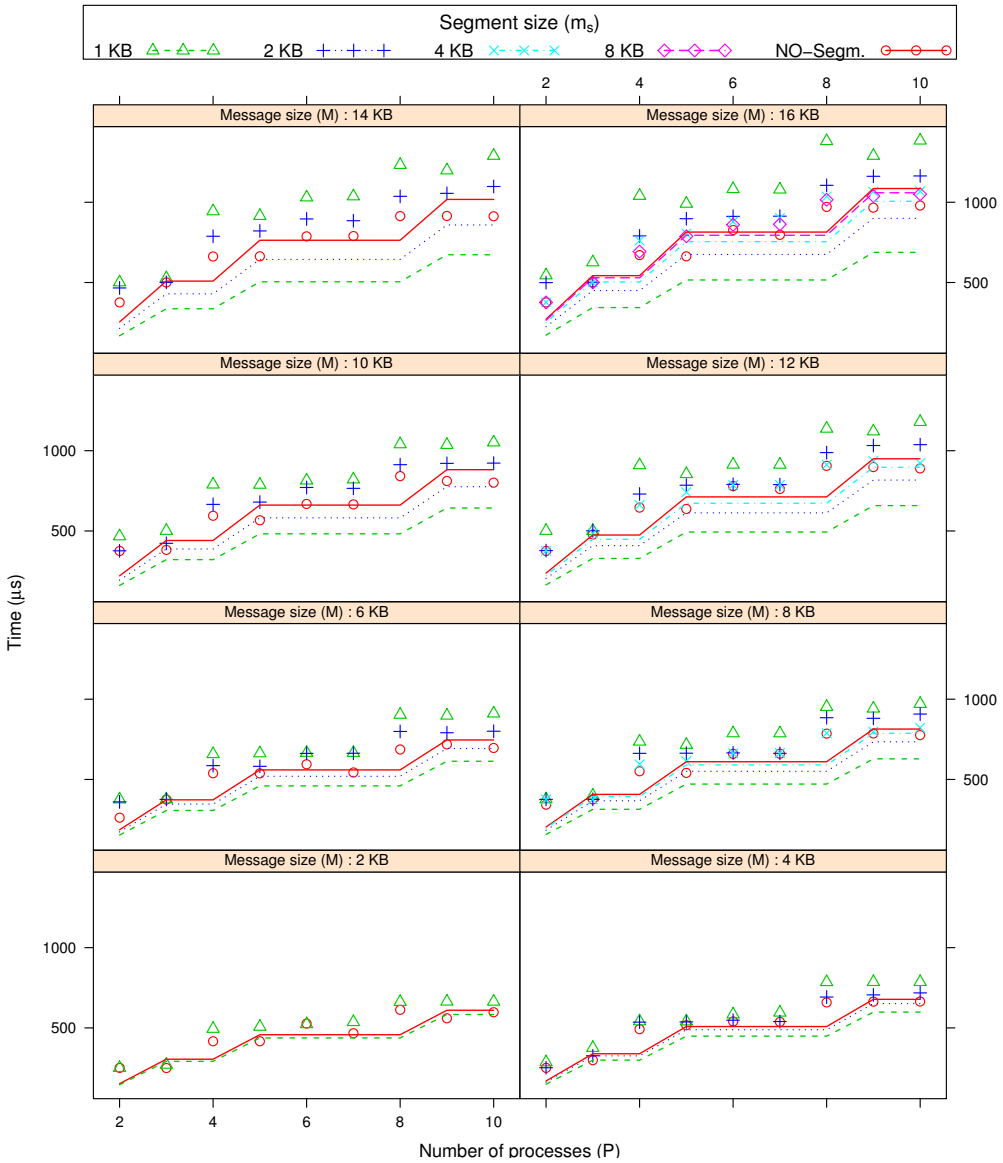


Figura 4.17: Datos experimentales (puntos) y modelo teórico (líneas) del algoritmo binomial de *broadcast*

el error de los modelos teóricos está sesgado, ya que los modelos subestiman el tiempo de ejecución en todos los casos. En este análisis se debe tener en cuenta que el propio modelo LogGP presenta un error asociado al caracterizar comunicaciones colectivas. De hecho, Alexandrov et ál. [8] obtienen un error en torno al 20 % para la caracterización de una comunicación del tipo *all-to-all*, cuya predicción también subestima el comportamiento real. Alexandrov et ál. asocian estos errores a la contención de la red y a la variación del parámetro **G** al aumentar el número de mensajes en la red.

Sin embargo, en algunos algoritmos se pueden apreciar ciertas similitudes. Por ejemplo, los modelos correspondientes al algoritmo lineal son muy semejantes en estructura y valores numéricos, ya que los coeficientes de $n_s m_s$ y $n_s m_s P$ son muy similares, tanto en signo como en magnitud. Por lo tanto, teniendo en cuenta que $n_s m_s$ se corresponde con el tamaño global del mensaje, en este caso particular se puede afirmar que la segmentación de los mensajes no influye en el comportamiento del mecanismo de comunicación, tal y como se aprecia claramente en los resultados experimentales. Por otro lado, el modelo teórico del caso binario pronostica un efecto dominante de los términos $\lceil \log_2 P \rceil$ y, sin embargo, en los datos experimentales existe una clara dependencia del factor $\lfloor \log_2 P \rfloor$, como se puede apreciar en las figuras —nótese el error sistemático del modelo teórico para estimar el tiempo de ejecución cuando se utilizan 2, 4 y 8 procesos—. De hecho, existe una clara similitud entre el comportamiento experimental del algoritmo binomial y el algoritmo binario.

En definitiva, los modelos AIC permiten hacer predicciones del tiempo de ejecución con mayor precisión que los modelos teóricos. Además, los modelos AIC son capaces de caracterizar comportamientos que, a priori, no hayan sido considerados como, por ejemplo, los debidos a la implementación particular de los algoritmos en el código de la librería Open MPI.

4.6 Contribuciones

El método de selección de modelos basado en AIC para obtener modelos analíticos de rendimiento de aplicaciones paralelas es la contribución más relevante de este capítulo. En concreto, este método se ha implementado en una función de la librería de análisis del entorno TIA y se ha probado para diferentes códigos. El mecanismo de modelado ha sido utilizado para modelar el tiempo de ejecución de diversos algoritmos de comunicación de tipo *broadcast*, y los resultados obtenidos han sido contrastados con modelos teóricos obtenidos mediante un análisis algorítmico. Las siguientes publicaciones son el fruto de las principales contribuciones del trabajo desarrollado en este capítulo:

- *El Criterio de Información de Akaike en la Obtención de Modelos Estadísticos de Rendimiento*, XX Jornadas de Paralelismo 2009 [112].
- *Performance Modeling of MPI Applications Using Model Selection*, 18$^{\text{th}}$ Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP 2010 [121].

CAPÍTULO 5

CASOS DE ESTUDIO

En este capítulo se presentan tres casos de estudio de modelos analíticos de rendimiento obtenidos mediante el mecanismo de selección de modelos descrito en el capítulo 4. El primer caso de estudio trata sobre algunos códigos contenidos en el *benchmark* paralelo NAS, para los que se obtienen los modelos analíticos y se comparan con los datos experimentales. En el segundo caso de estudio, se analiza la utilización del modelo basado en AIC del *benchmark* HPL en las políticas de planificación de trabajos en un cluster. En el tercer caso de estudio se obtiene un modelo analítico de los fallos cache. Este último caso de estudio ilustra cómo el entorno TIA permite la obtención de modelos de cualquier observable, no sólo del tiempo de ejecución.

5.1 NPB - NAS *paralell benchmarks*

El *benchmark* paralelo NAS (*NAS Parallel Benchmark*, NPB) [15, 165] está constituido por diversos códigos relacionados con la dinámica de fluidos computacional (*Computational Fluid Dynamics*, CFD). Este *benchmark* ha sido diseñado para comparar el rendimiento de sistemas de computación paralelos y ha sido reconocido como un indicador estándar de su rendimiento. En su versión 2.4, el NPB consta de tres pseudoaplicaciones de CFD (LU, BT y SP) y cinco *kernels* (EP, MG, CG, FT e IS) [165]. Todos los códigos están escritos en Fortran, a excepción del *kernel* IS que lo está en C, y utilizan funciones MPI para la comunicación entre los diferentes procesos.

Las pseudoaplicaciones son una emulación de los métodos computacionales habituales en las aplicaciones CFD reales. Los tres códigos considerados en el *benchmark* resuelven una

versión discretizada de las ecuaciones de Navier-Stokes en un espacio tridimensional, pero utilizando diferentes algoritmos:

- *Lower-Upper* (LU). Emplea un método de sobrerelajación y divide la matriz en dos mitades, una triangular inferior y otra triangular superior.
- *Scalar Pentadiagonal* (SP). Utiliza el algoritmo de factorización *Beam and Warming*.
- *Block Triagonal* (BT). Calcula la solución factorizando la matriz como producto de tres operandos, uno por cada dimensión.

Los cinco *kernels* son una muestra representativa de los métodos numéricos empleados habitualmente en este tipo de aplicaciones:

- *Embarrassingly Parallel* (EP). Método típico de las aplicaciones Monte Carlo.
- *Multigrid* (MG). Este *kernel* resuelve una versión simplificada de una ecuación diferencial parcial 3D de Poisson.
- *Conjugate Gradient* (CG). Este *kernel* calcula una aproximación del mayor autovalor de una matriz dispersa, simétrica y definida positiva.
- *Fourier Transform* (FT). Este *kernel* implementa la solución de un sistema de ecuaciones diferenciales parciales utilizando FFTs (*Fast Fourier Transforms*).
- *Integer Sort* (IS). Ordena una lista de números enteros utilizando el algoritmo *bucket sort*.

Los ocho códigos del *benchmark* proporcionan una estimación de diferentes niveles de rendimiento, tanto en comunicación como en cómputo, de los métodos numéricos típicos de aplicaciones CFD [16, 54, 145]. Las tres pseudoaplicaciones utilizan fundamentalmente comunicaciones punto a punto, pero presentan diferentes características de comunicación. En particular, la aplicación LU presenta un mayor número de comunicaciones, pero el volumen de datos enviados es menor, el código SP envía un mayor volumen de datos y la aplicación BT realiza un menor número de envíos. El *kernel* EP apenas tiene comunicaciones, por lo que representa una buena estimación de rendimiento de las operaciones en punto flotante del sistema. El *kernel* MG es un buen estimador de las comunicaciones estructuradas, mientras que el *kernel* CG utiliza un patrón de comunicaciones irregular. El rendimiento del *kernel* FT depende en gran medida de la librería utilizada para el cálculo de las transformadas de Fourier. Por

último, el *kernel* IS presenta una elevada proporción comunicaciones-computaciones, pero no utiliza operaciones en punto flotante, por lo que también es un buen estimador de la capacidad de procesamiento de números enteros.

El *benchmark* tiene definidos varios tamaños de problema, conocidos como clases, para cada uno de los códigos considerados. La clase S es para realizar pruebas y la clase W para sistemas con poca memoria. Las clases A, B y C proporcionan, en orden creciente, diferentes tamaños de problema. La clase D ha sido incluida para sistemas con grandes recursos.

Utilizando el entorno TIA, se han obtenido los modelos AIC de algunos de los códigos incluidos en el *benchmark* paralelo NAS [121]. En particular, se han seleccionado dos *kernels* (IS y CG) y una aplicación de simulación (SP), porque ofrecen un espectro significativo de las diferentes estructuras y relaciones comunicación-computación del *benchmark*. El tiempo de ejecución de los códigos SP, CG y IS —adecuadamente instrumentados con las directivas de `call`— ha sido medido en el cluster `burdeos`, un cluster con siete nodos interconectados con una red *Gigabit Ethernet*. Cada nodo del cluster tiene dos procesadores Intel® Xeon® QuadCore 2,33 GHz, con 8 GB de memoria principal y sistema operativo Linux 2.6.30. Los códigos han sido compilados utilizando la librería Open MPI 1.2.9.

5.1.1 *Integer Sort* - IS

Este *kernel* se basa en una implementación paralela del algoritmo *bucket sort* para ordenar una lista de tk números enteros. El número de procesos de este *kernel* debe ser una potencia entera de dos. En este caso particular, se considera la clase C. Este *kernel* utiliza funciones MPI de comunicación colectivas. El entorno de ejecución se ha configurado para utilizar un algoritmo lineal en todas las funciones de comunicación colectivas.

La figura 5.1 muestra los datos experimentales obtenidos para diferentes valores de procesos (P) y nodos del cluster (N). Para cada condición experimental, se ha ejecutado el código diez veces. En esta gráfica se puede observar que el tiempo de ejecución del *benchmark* para 32 procesos presenta una elevada dispersión. Además, los valores medidos se distribuyen en dos grupos claramente diferenciados. Dada la naturaleza de este *benchmark*, en el que existen un gran número de comunicaciones [16, 54], estas situaciones son habituales debido a los conflictos en la red de interconexión. Los valores del conjunto con menor tiempo de ejecución se corresponderían con ejecuciones en las que apenas existen conflictos de red, mientras que los valores con un mayor tiempo de ejecución se corresponderían con situaciones en las que se producen interacciones en la red de interconexión del sistema.

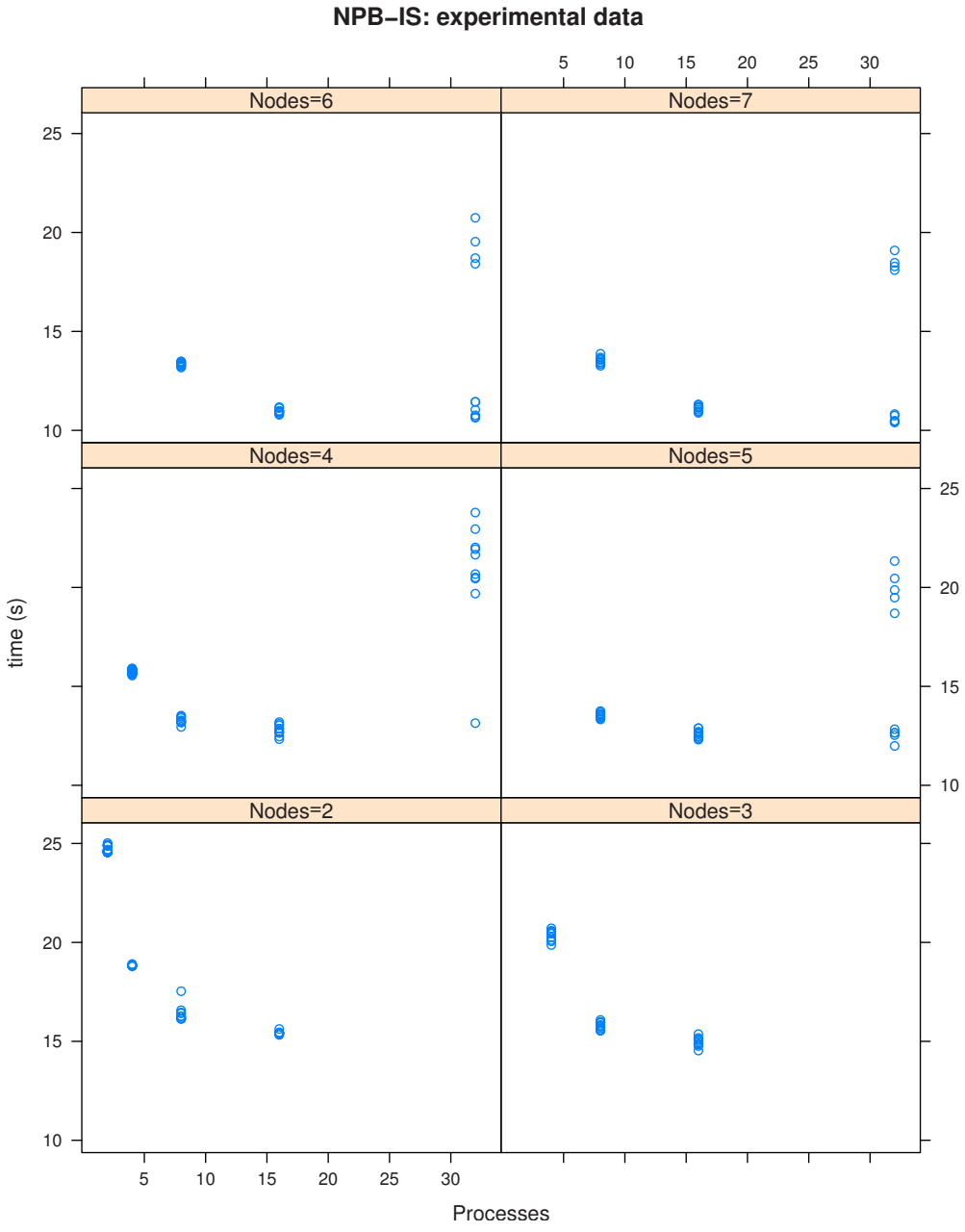


Figura 5.1: Tiempos de ejecución del *benchmark* IS

En esta situación el modelo que se obtendría de la aplicación directa del proceso de selección de modelos sobre los datos experimentales no representaría una situación realista porque, en el caso de 32 procesos, se estaría favoreciendo un valor promedio de todos los valores, que probablemente conduciría a una estimación situada justo en el *gap* entre ambos grupos y que no sería representativa de ninguna situación real. Del mismo modo, al existir una clara clasificación de los valores en dos grupos, tampoco sería apropiado realizar un proceso estadístico de centralización —como, por ejemplo, la media o la mediana— a las repeticiones del experimento. En este caso, partiendo del supuesto de que este dimorfismo es consecuencia de la contención de la red, se han considerado dos casos independientes. Por un lado, se ha considerado el caso más «favorable» —con un menor número de conflictos de red y, por lo tanto, un menor tiempo de ejecución asociado— y, por otro lado, se ha considerado también el caso más «desfavorable» —con un mayor número de conflictos—. Estos dos casos se corresponden con los valores extremos de las dos situaciones experimentales observadas. En concreto, estos casos han sido caracterizados utilizando los valores mínimo y máximo, respectivamente, de los tiempos de ejecución medidos en cada situación experimental. El método de selección de modelos ha sido aplicado de manera independiente a estas dos situaciones y, dado que las condiciones experimentales son las mismas, se ha utilizado la misma lista inicial (`LI`) para ambas:

$$\texttt{LI}_{\text{IS}} = \left\{ \frac{1}{P}, P \right\}, \left\{ N, \frac{1}{N}, \left\lceil \frac{P}{N} \right\rceil \right\}$$

El primer elemento de esta lista contiene los términos correspondientes a la distribución de la carga computacional entre los procesos: por un lado, suponemos un reparto equitativo ($\frac{1}{P}$) pero, por el otro lado, también una componente de sobrecarga al aumentar el número de procesos (P). En el segundo elemento se reflejan aquellos parámetros que dependen de la configuración del cluster. En concreto, se ha considerado el número de nodos (N, $\frac{1}{N}$) y el número máximo de procesos por nodo en cada ejecución ($\left\lceil \frac{P}{N} \right\rceil$). El número de modelos del conjunto de candidatos resultante es 4095 ($2^{12}-1$).

Los modelos obtenidos con el proceso de selección de modelos de TIA son:

$$T_{\min}(s) = 6{,}3 + 24{,}4\frac{1}{P} + 0{,}17P + 19\frac{1}{P}\left\lceil \frac{P}{N} \right\rceil - 7\frac{1}{N} - 0{,}019PN$$

$$T_{\text{Max}}(s) = 9 + 7\frac{1}{P} + 30\frac{1}{P}\left\lceil \frac{P}{N} \right\rceil + 0{,}119P\left\lceil \frac{P}{N} \right\rceil - 2{,}9\left\lceil \frac{P}{N} \right\rceil$$

El modelo $T_{\min}$ presenta un error de $1{,}8\,\%$ y su peso de Akaike dentro del conjunto de modelos es 0,1158. En el caso de T_{Max}, el error y el peso de Akaike son, respectivamente, $3{,}9\,\%$

Tabla 5.1: Importancia relativa de cada término para la lista $\mathtt{LI}_{\mathtt{IS}}$ en los dos casos considerados (sólo se muestran los valores superiores a 0,5)

	$\omega_{+}^{\mathtt{MC}}$	
	$T_{\min}$	T_{Max}
CTE	0.9025615	0.7869771
$\frac{1}{P}$	0.7070166	-
P	-	-
N	0.6333485	-
$\frac{1}{N}$	0.9025615	-
$\left\lceil \frac{P}{N} \right\rceil$	-	0.9016280
$\frac{N}{P}$	-	-
$\frac{1}{PN}$	-	-
$\frac{1}{P}\left\lceil \frac{P}{N} \right\rceil$	0.9025615	0.9016280
PN	-	-
$\frac{P}{N}$	-	-
$P\left\lceil \frac{P}{N} \right\rceil$	-	0.9016280

y 0,1156. La tabla 5.1 muestra la importancia relativa de cada término considerado. En la figura 5.2 se muestran los modelos $T_{\min}$ y T_{Max}, junto con los datos experimentales. Como se puede observar en esta figura, los modelos representan adecuadamente el comportamiento experimental.

Las diferencias entre ambos modelos son consecuentes con el escenario de conflictos de red supuesto inicialmente. Por un lado, el coeficiente del término $\frac{1}{P}$ es menor en el modelo T_{Max}, que denota la menor importancia del balanceo de la carga en esta situación. Por otro lado, la importancia del número de procesos por nodo es mayor en este modelo. Además, los términos $\frac{1}{P}\left\lceil \frac{P}{N} \right\rceil$ y $P\left\lceil \frac{P}{N} \right\rceil$ tienen coeficientes positivos, mientras que el coeficiente del término $\left\lceil \frac{P}{N} \right\rceil$ es negativo. Estos valores indican que, independientemente del número de nodos, el número de potenciales conflictos aumenta con el número de procesos por nodo, mientras que el coste temporal de las comunicaciones intranodo es menor en comparación con las comunicaciones internodo, ya que entre procesos dentro del mismo nodo no se crea, en ningún caso, un conflicto de red.

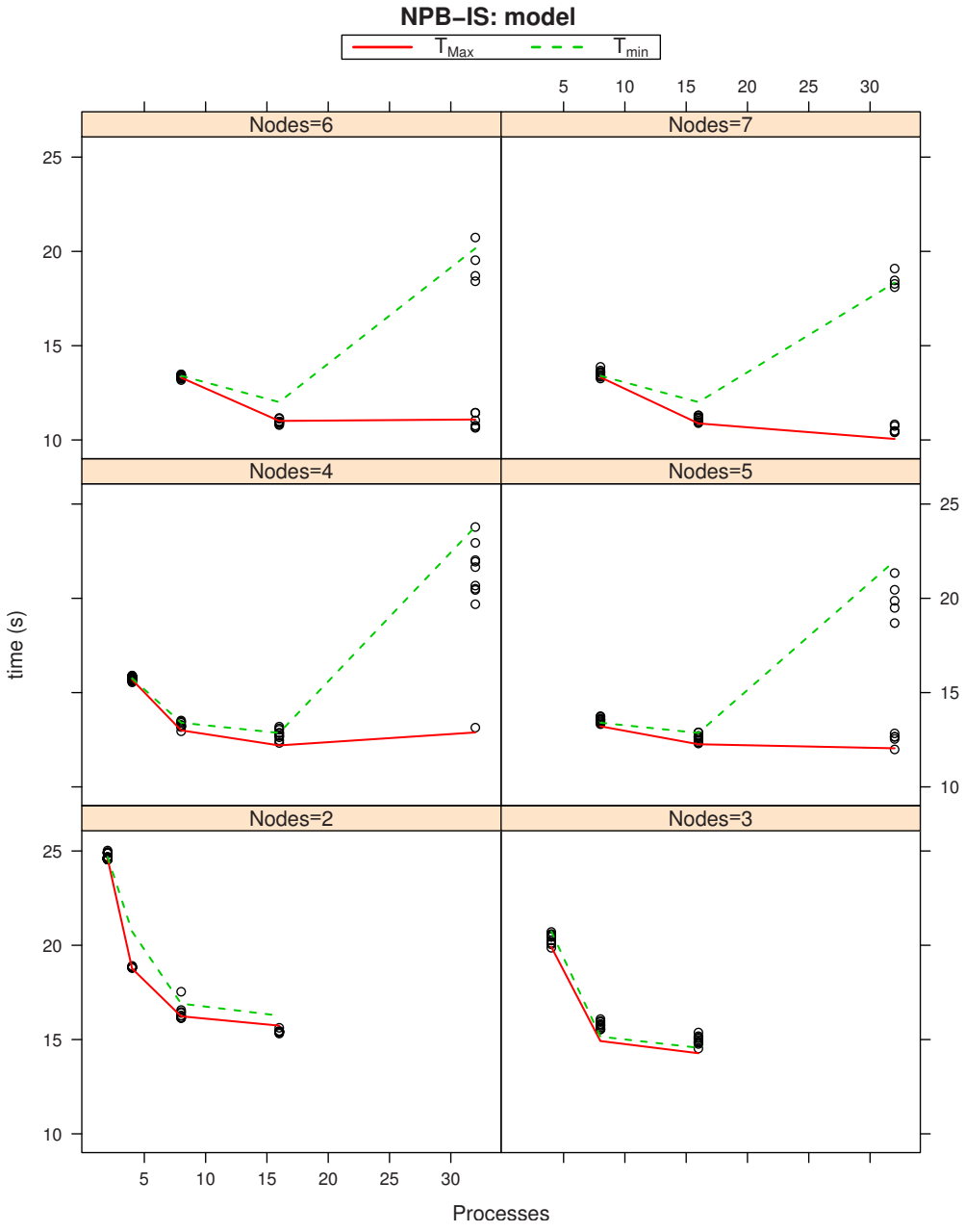


Figura 5.2: Datos experimentales y modelos $T_{\min}$ y T_{Max} del *benchmark* IS

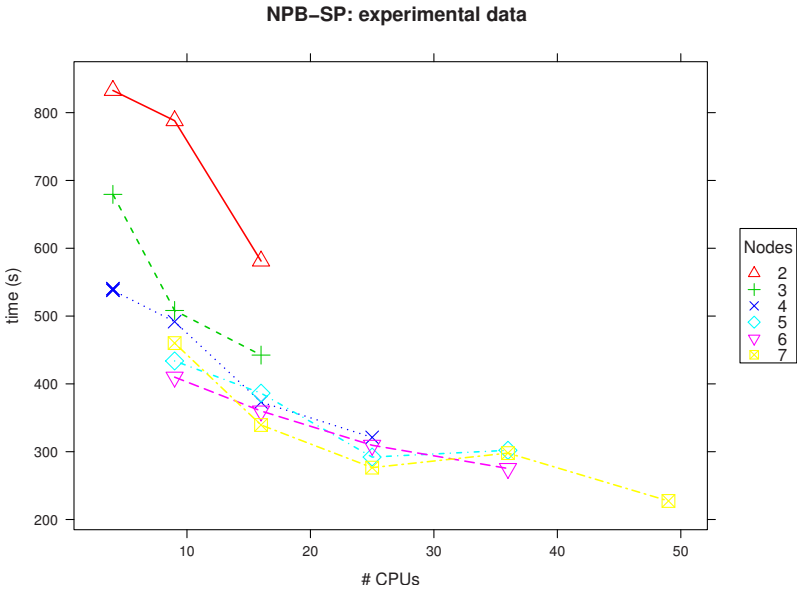


Figura 5.3: Datos experimentales del *benchmark* SP

5.1.2 *Scalar Pentadiagonal* - SP

Esta simulación resuelve un sistema de ecuaciones en un espacio tridimensional. El algoritmo paralelo distribuye por bloques el espacio simulado entre los diferentes procesos. Iterativamente, cada proceso resuelve su porción de espacio de manera independiente y los resultados de los puntos frontera entre bloques contiguos se intercambian en cada iteración del código. El número de procesos debe ser un cuadrado perfecto. En este caso particular, se considera la clase C.

La figura 5.3 muestra los resultados experimentales de la ejecución del código SP para diferentes valores de número de procesos (P) y nodos (N). Para cada situación experimental se ha obtenido un único valor experimental. El conjunto de parámetros y la configuración del experimento es muy similar al caso del *kernel* IS, analizado anteriormente, por lo que se ha considerado la misma lista inicial:

$$\mathtt{LI}_{\mathrm{SP}} = \left\{P, \frac{1}{P}\right\}, \left\{N, \frac{1}{N}, \left\lceil \frac{P}{N} \right\rceil\right\}$$

Tabla 5.2: Importancia relativa de cada término para la lista $\mathtt{LI}_{\mathtt{SP}}$ (sólo se muestran los valores superiores a $0,5$)

	$\omega_{+}^{\mathtt{MC}}$
$\frac{1}{N}$	0.8855806
$\frac{P}{N}$	0.8693725
N	0.7004454
$P\left\lceil\frac{P}{N}\right\rceil$	0.6727668
PN	0.5857881
$\frac{1}{P}\left\lceil\frac{P}{N}\right\rceil$	0.5092569

El modelo obtenido mediante TIA es el siguiente:

$$T_{\mathrm{SP}}(s) = 40N + 1830\frac{1}{N} - 0{,}7PN - 68\frac{P}{N} + 12P\left\lceil\frac{P}{N}\right\rceil$$

Este modelo presenta un error de $5,1\,\%$ y su peso de Akaike es $0,1292$. La tabla 5.2 muestra la importancia relativa de cada término considerado. La figura 5.4 muestra gráficamente el resultado obtenido.

En el modelo de T_{SP} se puede observar que el número de nodos tiene más influencia que el número de procesos en el rendimiento del código SP. De hecho, los términos $\frac{1}{N}$ y N están entre los términos con mayor peso de Akaike (tabla 5.2). La relación entre el número de procesos y el número de nodos de la ejecución se refleja en los tres últimos términos del modelo. Por un lado, el coeficiente del término $P\left\lceil\frac{P}{N}\right\rceil$ es un indicador del sobrecoste de incrementar el número de procesos sin reducir el número de procesos por nodo. El valor negativo de los coeficientes de los otros dos términos es una evidencia de que el tiempo de ejecución tiende a disminuir al aumentar el número de procesos.

5.1.3 *Conjugate Gradient* - CG

El *kernel* CG utiliza el método de iteración inversa para encontrar una estimación del mayor autovalor de una matriz dispersa simétrica, definida positiva y con un patrón aleatorio de elementos distintos de cero. En cada iteración de este método, se estima la solución a un sistema lineal de ecuaciones utilizando el método del gradiente conjugado. Al igual que en el *kernel* IS, el número de procesos debe ser una potencia entera de dos. En este caso, obtendremos un modelo del tiempo de ejecución del kernel CG en función del tamaño de problema y del número de procesos, para un número concreto de nodos computacionales.

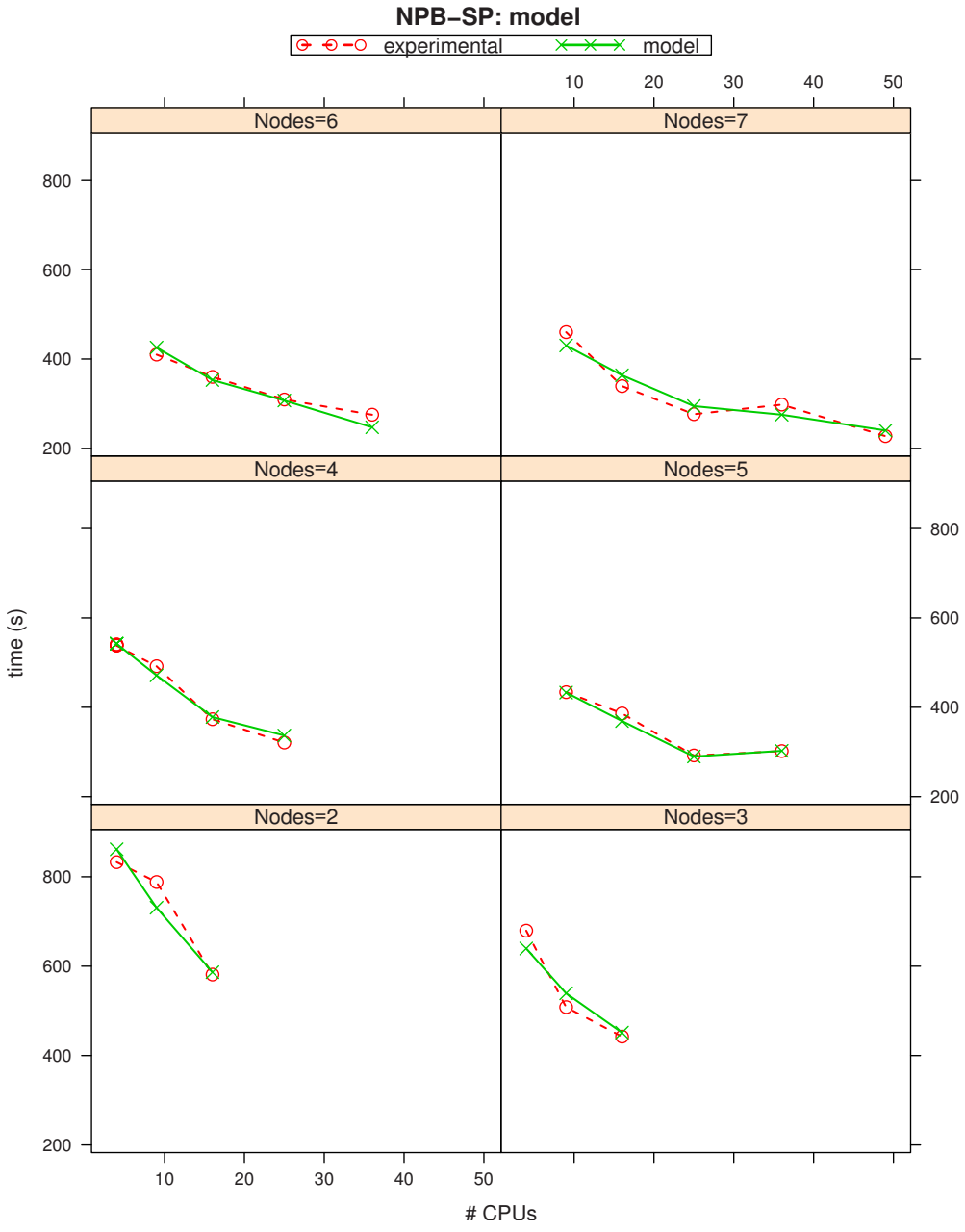


Figura 5.4: Datos experimentales y modelo del *benchmark* SP

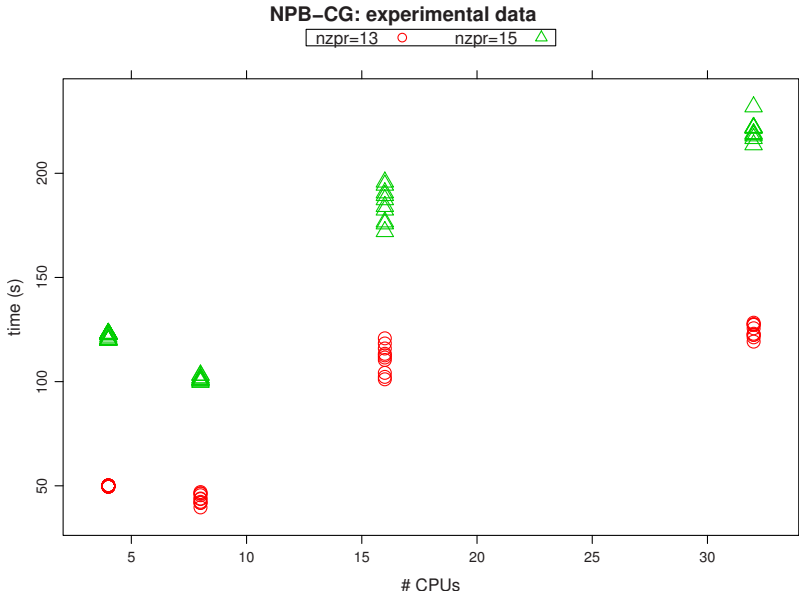


Figura 5.5: Datos experimentales del *benchmark* CG

La figura 5.5 muestra los resultados experimentales de la ejecución del kernel CG para diferentes valores del número de procesos (P) y para distintos tamaños de problema (clases B y C), cuando se utilizan únicamente 4 nodos del cluster. Para cada condición experimental, se ha ejecutado diez veces el código. El único parámetro en el que se diferencian los dos tamaños de problema considerados es el número de ceros por fila ($nzpr$).

Teniendo en cuenta las condiciones experimentales, en este caso se ha considerado la siguiente lista inicial:

$$\mathtt{LI}_{\mathrm{CG}} = \{nzpr\}, \left\{P, \frac{1}{P}\right\}$$

El primer elemento de la lista refleja el tamaño de problema de las distintas clases del *kernel*. El segundo elemento se corresponde con las listas $\mathtt{LI}_{\mathrm{IS}}$ y $\mathtt{LI}_{\mathrm{SP}}$, para el caso particular en el que el número de nodos (N) se mantiene constante y, simultáneamente, el número de procesos (P) es un múltiplo entero de N. Esta lista genera un conjunto de modelos candidato con 63 ($2^6 - 1$) elementos.

Tabla 5.3: Importancia relativa de cada término para la lista $\mathtt{LI}_{\mathrm{CG}}$

	$\omega_{+}^{\mathtt{MC}}$
$nzprP$	0.9468941
$\frac{nzpr}{P}$	0.8449970
$nzpr$	0.7929585
CTE	0.7870517
$\frac{1}{P}$	0.6363694
P	0.5536112

Utilizando la lista $\mathtt{LI}_{\mathrm{CG}}$, el proceso de selección de modelos de TIA proporciona el siguiente modelo:

$$T_{\mathrm{CG}}(s) = -210 + 16nzpr - 700\frac{1}{P} - 8P + 60\frac{nzpr}{P} + 0{,}9nzprP$$

El error del modelo 12,4 % y su peso de Akaike es 0,24814. La tabla 5.3 muestra la importancia relativa de cada término considerado y la figura 5.6 muestra gráficamente el resultado obtenido.

Teniendo en cuenta los parámetros de calidad del mecanismo de selección de modelos —es decir, el peso de Akaike y el peso relativo de los términos—, el modelo seleccionado es un buen modelo dentro del conjunto de candidatos seleccionado. Sin embargo, T_{CG} se corresponde con el modelo de mayor dimensión del conjunto de candidatos y, además, la distribución del error no es uniforme respecto del número de procesos. Esta situación indica que el conjunto de modelos escogido no contiene «buenos» modelos, es decir, hay comportamientos del *kernel* que no se reflejan en los modelos candidatos.

En este caso particular, la distribución de los procesos está restringida, por la propia configuración del código, a potencias enteras de dos. Por lo tanto, el analista puede suponer que sea posible que las comunicaciones se realicen mediante algoritmos de complejidad logarítmica. La incorporación de este potencial comportamiento al proceso de selección de modelos se puede realizar mediante la siguiente lista inicial:

$$\mathtt{LI}_{\mathrm{CG}}^{\log} = \{nzpr\}, \left\{P, \frac{1}{P}\right\}, \{\log_2 P\}$$

Esta lista genera un conjunto de modelos candidatos con 4 095 $(2^{12}-1)$ elementos, incluyendo todos los modelos generados por la lista $\mathtt{LI}_{\mathrm{CG}}$. Utilizando esta nueva lista, el entorno TIA

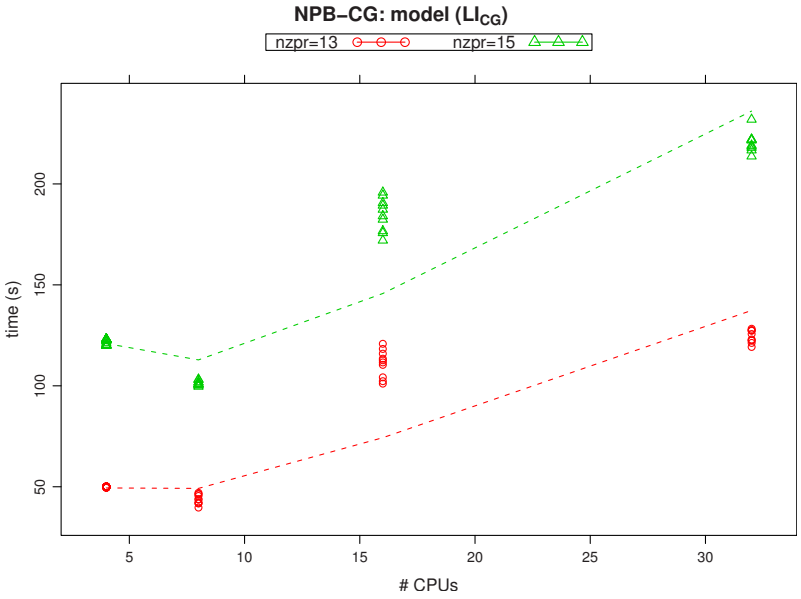


Figura 5.6: Datos experimentales y modelo T_{CG} del *benchmark* CG

proporciona el siguiente modelo:

$$T_{\mathrm{CG}}^{\log}(s) = -1260\frac{1}{P} - 9,1P + 232\frac{nzpr}{P} - 43\log_2 P + 10,9nzpr\log_2 P - 88\frac{nzpr}{P}\log_2 P$$

Este modelo presenta un error relativo de 2,7 % y su peso de Akaike es 0,013518. La tabla 5.4 muestra la importancia relativa de cada término considerado y la figura 5.7 muestra gráficamente el resultado obtenido. Como se puede observar en la figura 5.7, el modelo $T_{\mathrm{CG}}^{\log}$ se adapta adecuadamente al comportamiento experimental y, además, presenta una distribución uniforme de los residuos del ajuste. Por otro lado, los dos términos con coeficientes positivos de esta expresión se pueden identificar claramente con el reparto de tareas ($\frac{nzpr}{P}$) y con la comunicación entre procesos ($nzpr\log_2 P$).

La comparación directa entre los modelos T_{CG} y $T_{\mathrm{CG}}^{\log}$, en términos de los parámetros de calidad del proceso de selección, no es posible porque el conjunto de modelos es diferente en los dos casos. En términos absolutos, el modelo $T_{\mathrm{CG}}^{\log}$ es claramente superior en precisión, ya que el error es mucho menor, a pesar de que ambos modelos tienen la misma dimensión. Teniendo en cuenta los datos de la tabla 5.4 y la estructura de las listas iniciales, se puede

Tabla 5.4: Importancia relativa de cada término para la lista $\mathtt{LI}_{\mathrm{CG}}^{\log}$ (sólo se muestran los valores superiores a $0,5$)

	$\omega_{+}^{\mathtt{MC}}$
$\frac{1}{P}$	0.7328793
$\frac{nzpr}{P}$	0.6800037
$\frac{nzpr}{P}\log_2 P$	0.6474020
$\log_2 P$	0.5838214
$nzpr\log_2 P$	0.5738641
P	0.5689844
$nzpr$	0.5541470
$P\log_2 P$	0.5029771

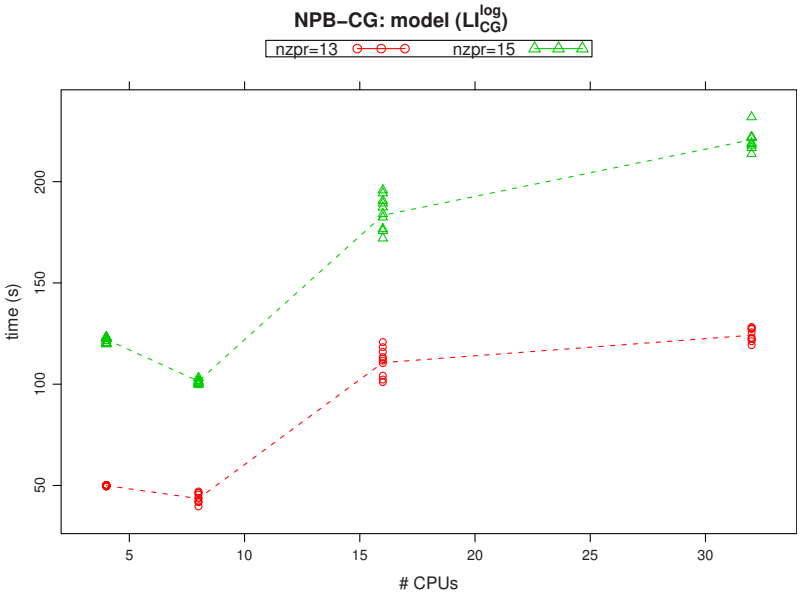


Figura 5.7: Datos experimentales y modelo $T_{\mathrm{CG}}^{\log}$ del *benchmark* CG

concluir que el factor que diferencia ambos conjuntos ($\log_2 P$) es fundamental para obtener modelos que se ajusten con mayor precisión a los datos experimentales.

Las limitaciones del diseño experimental también influyen en los valores de los pesos de Akaike y de la importancia relativa de los términos, que se obtienen con la lista $\mathtt{LI}_{\mathrm{CG}}^{\log}$. En este caso particular, el número de configuraciones experimentales diferentes es del mismo orden que el número de parámetros que potencialmente pueden influir en el comportamiento de la aplicación. Por lo tanto, es posible que modelos de elevada dimensión se ajusten coyunturalmente a los datos experimentales. Esta situación particular proporciona valores relativamente bajos en los pesos de Akaike y, por ende, en la importancia relativa de los términos, ya que en el conjunto de modelos habrá muchos cercanos al valor AIC mínimo. Este tipo de situación es habitual en sistemas paralelos, ya que no existe un espacio continuo e infinito en muchas de las variables que influyen directamente. Por ejemplo, el número de procesadores está limitado al tamaño máximo del sistema o el número de procesos debe seguir un patrón determinado (como en el caso del *kernel* CG) o el tamaño de problema está parametrizado. En cualquier caso, incluso en estas situaciones, el mecanismo de selección de modelos, implementado en TIA, es capaz de proporcionar no sólo un modelo preciso sino también la información adicional necesaria para realizar una valoración adecuada del resultado.

5.2 Planificación de trabajos en un cluster

Los modelos basados en AIC que genera el entorno TIA, permiten determinar el tiempo de ejecución de una aplicación de manera muy precisa. La idea de este caso de estudio es utilizar esta característica para mejorar la eficiencia de las estrategias de *backfilling* en la planificación de trabajos en un cluster [113, 114].

En primer lugar, utilizando el entorno TIA, en esta sección se obtendrá el modelo analítico del tiempo de ejecución del *benchmark* HPL en un cluster particular. A continuación, mediante el simulador GridSim toolkit, se simulará el comportamiento de diversas políticas o técnicas de distribución de tareas en un cluster, utilizando los modelos TIA para obtener la predicción del tiempo de ejecución necesaria en la políticas de *backfilling*. Estas simulaciones se compararán con las equivalentes utilizando el modelo teórico, proporcionado por los desarrolladores del *benchmark*, para predecir el tiempo de ejecución del *benchmark*. Este estudio nos mostrará un posible uso práctico de los modelos ofrecidos por TIA.

5.2.1 HPL - High Performance LINPACK

HPL (*High Performance Linpack*) es una extensión del *benchmark* LINPACK para sistemas paralelos con memoria distribuida [82, 45]. En concreto, este *benchmark* resuelve sistemas lineales densos de ecuaciones de orden N:

$$Ax = b; \qquad A \in \mathbb{R}^{N \times N}; \quad x, b \in \mathbb{R}^{N} \tag{5.1}$$

Está escrito en C y utiliza doble precisión en las operaciones en punto flotante. HPL utiliza la librería BLAS o VSIPL para las operaciones de álgebra lineal y las comunicaciones entre procesos se realizan con funciones MPI. Los detalles de implementación de HPL pueden encontrarse en [45]. HPL es un *benchmark* muy extendido en los entornos HPC porque es la referencia utilizada para evaluar el rendimiento de los sistemas en la lista TOP500 [160].

HPL ha sido diseñado para ser un programa altamente escalable. En este sentido, la distribución de la matriz de coeficientes entre los procesadores garantiza la escalabilidad del programa, así como un buen balanceo de la carga. La matriz, de tamaño $N \times (N+1)$, se distribuye en bloques de tamaño $NB \times NB$ de forma cíclica, en ambas dimensiones, en una malla bidimensional de $P \times Q$ procesadores.

El código de HPL permite que, a través de un fichero de configuración (por defecto `HPL.dat`), el usuario pueda ejecutar diferentes configuraciones del *benchmark*. Estos parámetros son de naturaleza muy diversa y gestionan, por ejemplo, las diferentes posibilidades implementadas en HPL para la descomposición LU o para el mecanismo de comunicación colectiva *broadcast*, entre otros. Esta versatilidad del código permite que el usuario pueda seleccionar la configuración óptima para obtener el mejor rendimiento. Sin embargo, los parámetros N, NB, P y Q, que determinan el tamaño de la matriz de coeficientes y el esquema de distribución de datos, son los más relevantes para el rendimiento del *benchmark* [155] y, en principio, pueden tomar cualquier valor entero.

5.2.2 Obtención de los modelos analíticos de HPL

El número de posibles configuraciones de ejecución del *benchmark* HPL es muy elevado, ya que este *benchmark* dispone de 18 parámetros: diez de ellos tienen un número arbitrario de posibles valores, mientras que los ocho restantes únicamente pueden tomar un conjunto discreto de valores predefinidos. La obtención de un modelo analítico que contemple todos estos parámetros sería un proceso muy costoso. En consecuencia, los modelos analíticos del comportamiento de este *benchmark* suelen considerar una configuración concreta de ejecución y

caracterizan el rendimiento en función de un conjunto reducido de parámetros [155, 36]. De hecho, el modelo analítico propuesto por los propios desarrolladores de HPL ha sido obtenido para una configuración particular del *benchmark* [45]. Este modelo, desarrollado teóricamente gracias a un conocimiento en detalle de la implementación del *benchmark*, caracteriza el tiempo de ejecución de HPL en función de los cuatro parámetros que afectan en mayor medida al rendimiento: N, N_B, P y Q [155]. Este modelo teórico se define de la siguiente forma:

$$T_{\text{teó}} = \gamma_3 \frac{2N^3}{3PQ} + \beta \frac{N^2(3P+Q)}{2PQ} + \alpha \frac{N((N_B+1)\log P + P)}{N_B} \tag{5.2}$$

donde γ_3 es la tasa de operaciones en punto flotante (*floating point operation*, FLOP) de las operaciones matriz-matriz, mientras que α y β se corresponden con la latencia y el ancho de banda de la red de comunicación, respectivamente. Una asunción importante en el desarrollo de esta ecuación es que se considera la opción *modified increasing ring* para el *broadcast*, que reduce a uno el número de comunicaciones que se deben considerar en cada iteración del algoritmo. En cualquier caso, este modelo obvia diversos aspectos que afectan al rendimiento del procesador (como los fallos cache o los fallos de TLB) y asume que la memoria física disponible es suficiente para evitar *swapping* durante la ejecución del código [45].

El entorno TIA proporciona un mecanismo sencillo para obtener un modelo estadístico del comportamiento real del *benchmark* HPL. Utilizando la misma aproximación que los desarrolladores de HPL, se ha establecido una configuración de ejecución en la que las únicas variables del *benchmark* sean los mismo cuatro parámetros del modelo teórico: N, N_B, P y Q. Para establecer esta configuración, se han considerado las sugerencias de [82] y [155]. En particular, se ha escogido la configuración codificada como `WR03L2L1`, utilizando la notación del propio *benchmark*. En contraposición al desarrollo teórico, esta configuración utiliza la opción *modified increasing two-ring* de *broadcast*. Aunque, según los desarrolladores del código, la opción *modified increasing ring* es la más adecuada para la mayoría de los sistemas, no existe ningún indicador que determine el mejor algoritmo de *broadcast* y, de hecho, los propios desarrolladores indican que la versión *two-ring* es su «más serio competidor». Sin embargo, si utilizamos la opción *two-ring*, la expresión (5.2) no reflejaría correctamente el comportamiento real y debería corregirse, lo que a priori significaría un procedimiento tedioso, aún conociendo los detalles de la implementación. En este sentido, este caso de estudio muestra cómo el entorno TIA facilita el proceso de obtención de un modelo analítico sin que sea preciso conocer en detalle la implementación ni las características del sistema.

Tabla 5.5: Valores de los parámetros HPL considerados en el experimento

Parámetro	Valores
N	6144, 12288, 18432, 24576, 30720
NB	8, 16, 32, 64, 128, 256
P	1, 2, 3, 4, 5, 6, 7
Q	1, 2, 3, 4, 5, 6, 7, 8

Se ha construido un modelo analítico de la función `HPL_pdgesv`, que es la función que caracteriza la expresión (5.2). Esta función ha sido instrumentada adecuadamente para obtener el tiempo de ejecución inclusivo de la función y los parámetros de ejecución del *benchmark*. El código instrumentado ha sido ejecutado en el cluster `burdeos`. Como se ha descrito anteriormente, este cluster dispone de 7 nodos conectados con una red Gigabit Ethernet y en el que cada nodo contiene un biprocesador Intel® Xeon® Quadcore con 2,33 GHz y 8 Gigabytes de memoria. Aunque potencialmente se disponga de 56 núcleos, la gestión del cluster no permite que los nodos del cluster tengan asignadas dos o más aplicaciones MPI simultáneamente —independientemente del número de núcleos solicitados—, es decir, cada nodo se dedica en exclusiva a la instancia HPL asignada. La Tabla 5.5 muestra los diferentes valores considerados para cada parámetro. El código instrumentado ha sido ejecutado 1680 veces, una ejecución por cada posible configuración de valores.

El conocimiento básico que un usuario debe tener sobre la ejecución del *benchmark* es suficiente para proporcionar una lista adecuada para iniciar el proceso de selección implementado en TIA. Por un lado, cada matriz tiene $N \times N$ elementos, pero el algoritmo secuencial tiene una complejidad $O(N^3)$. Por otra parte, los otros tres parámetros son responsables de la distribución de la carga computacional en el sistema, que a priori se realizará de forma equitativa. La siguiente lista genera un modelo global que refleja todos estos aspectos de una forma simple:

$$\texttt{LI} = \left\{N^3, N^2\right\}, \left\{\frac{1}{NB}\right\}, \left\{\frac{1}{Q}\right\}, \left\{\frac{1}{P}\right\} \tag{5.3}$$

Esta lista describe un conjunto con más de 16 millones de modelos candidatos. Utilizando los valores experimentales obtenidos con la ejecución del código instrumentado, se ha utilizado el proceso de selección de modelos, y el siguiente modelo es el seleccionado como mejor

modelo del conjunto:

$$\begin{aligned} T_{\text{AIC}}(s) =& 1{,}812 - 2{,}749 \cdot 10^{-12} N^3 + 1{,}974 \cdot 10^{-7} N^2 - 29{,}65 \frac{1}{NB} - 3{,}239 \frac{1}{Q} \\ &+ 48{,}49 \frac{1}{NBQ} + 7{,}099 \frac{1}{QP} - 7{,}903 \cdot 10^{-7} \frac{N^2}{NB} + 8{,}228 \cdot 10^{-8} \frac{N^2}{Q} \\ &+ 4{,}059 \cdot 10^{-12} \frac{N^3}{P} - 2{,}706 \cdot 10^{-7} \frac{N^2}{P} + 7{,}940 \cdot 10^{-10} \frac{N^3}{NBP} \\ &- 5{,}817 \cdot 10^{-7} \frac{N^2}{NBP} + 1{,}613 \cdot 10^{-10} \frac{N^3}{QP} - 1{,}666 \cdot 10^{-10} \frac{N^3}{NBQP} \end{aligned} \tag{5.4}$$

La desviación estándar del ajuste (σ) de este modelo es 13,6 %.

Utilizando el mismo mecanismo proporcionado por TIA, se puede realizar el ajuste del modelo teórico a los datos experimentales definiendo una lista inicial que genere un conjunto de modelos constituido por un único elemento que se corresponda con la expresión (5.2). En este caso, la lista inicial será:

$$\texttt{LI}_{\text{HPL}} = \left\{ \frac{2N^3}{3PQ}, \frac{N^2(3P+Q)}{2PQ}, \frac{N((N_B+1)\log P + P)}{N_B} \right\} \tag{5.5}$$

El resultado obtenido tras el ajuste es:

$$\begin{aligned} T_{\text{teó}}(s) =& 3{,}195 \cdot 10^{-10} \frac{2N^3}{3PQ} + 1{,}015 \cdot 10^{-7} \frac{N^2(3P+Q)}{2PQ} \\ &+ 2{,}823 \cdot 10^{-4} \frac{N((N_B+1)\log P + P)}{N_B} \end{aligned} \tag{5.6}$$

En este caso, la desviación estándar del ajuste es 29,1 %.

Las figuras 5.8-5.15 muestran el valor de tiempo de ejecución medido experimentalmente y los valores que predicen los modelos T_{AIC} y $T_{\text{teó}}$, en el caso particular en que $P = 4$ —el comportamiento de las predicciones es muy similar para otros valores de P—. Las figuras 5.16-5.19 muestran una comparación de la distribución de los residuos del ajuste de T_{AIC} y $T_{\text{teó}}$, en función de los diferentes parámetros del *benchmark*. Como se puede observar en estas figuras, T_{AIC} se ajusta con mayor precisión a los datos experimentales que $T_{\text{teó}}$. Además, se puede apreciar que la dispersión de los residuos del modelo teórico no es independiente para todas las variables consideradas, un síntoma de que en este modelo no se están considerando ciertas características del *benchmark* que, bajo estas condiciones experimentales, afectan sustancialmente al comportamiento del mismo.

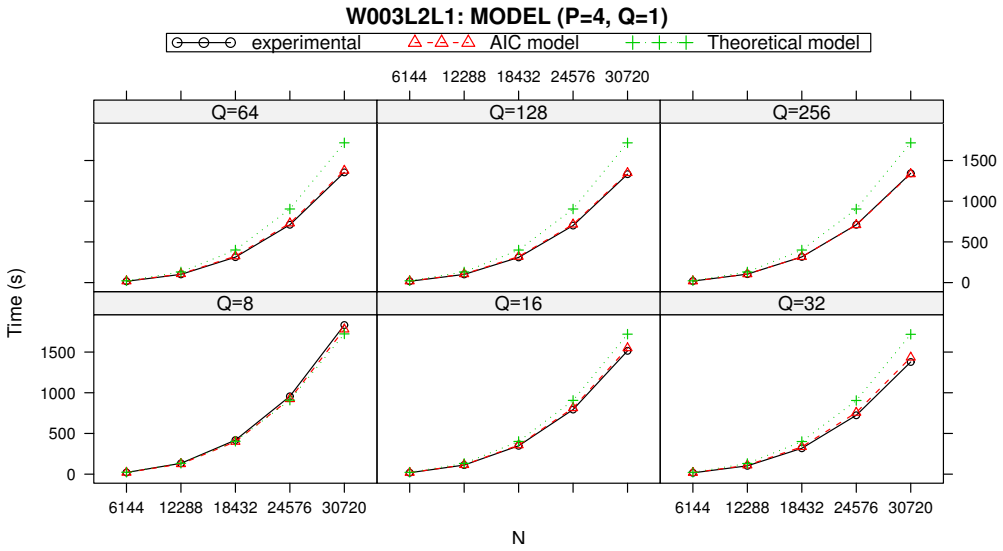


Figura 5.8: Modelos T_{AIC} y $T_{\text{teó}}$ frente a los datos experimentales ($P = 4$ y $Q = 1$)

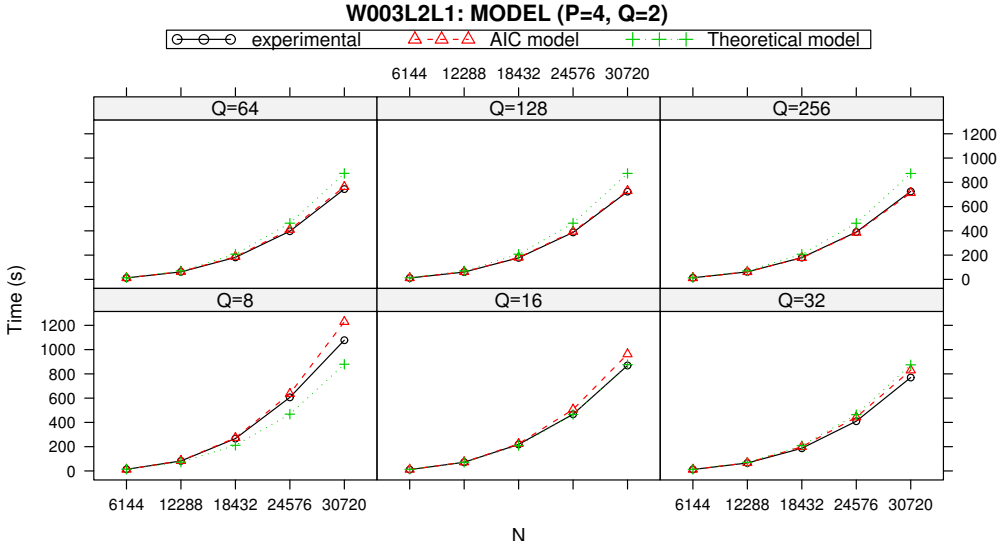


Figura 5.9: Modelos T_{AIC} y $T_{\text{teó}}$ frente a los datos experimentales ($P = 4$ y $Q = 2$)

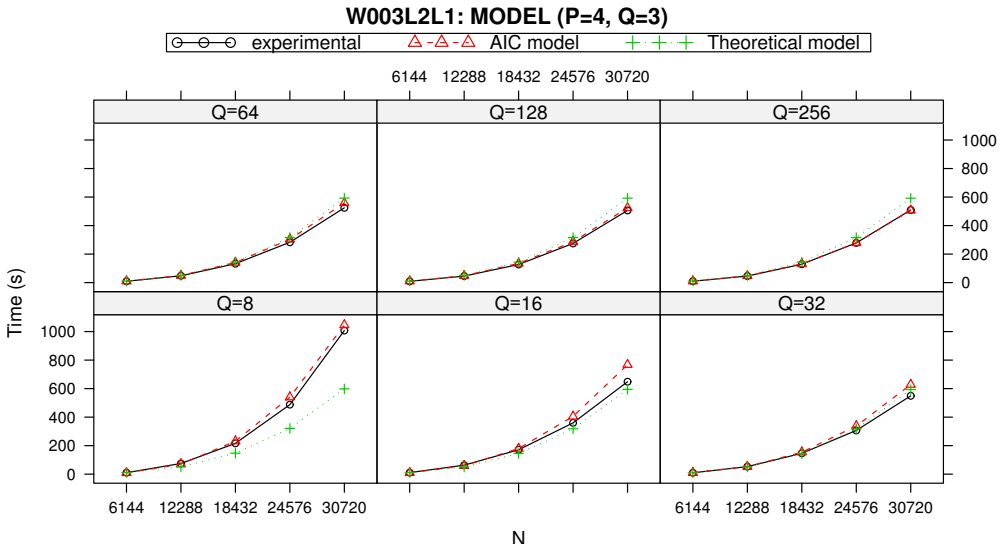


Figura 5.10: Modelos T_{AIC} y $T_{\text{teó}}$ frente a los datos experimentales ($P = 4$ y $Q = 3$)

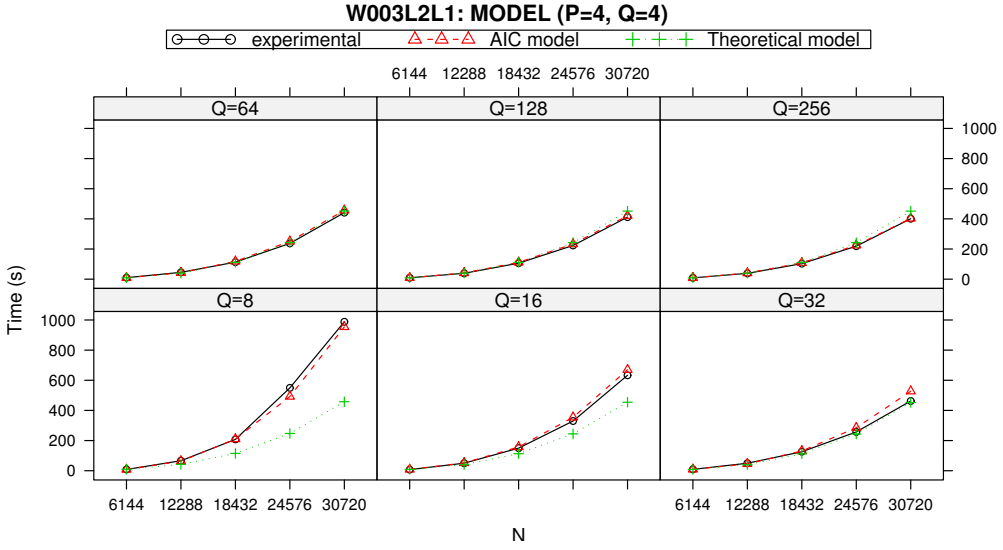


Figura 5.11: Modelos T_{AIC} y $T_{\text{teó}}$ frente a los datos experimentales ($P = 4$ y $Q = 4$)

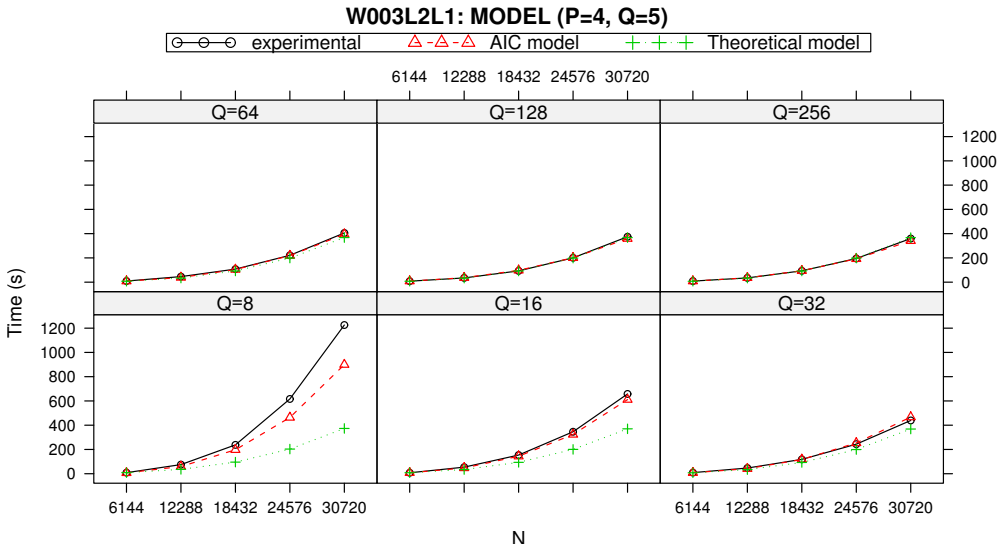


Figura 5.12: Modelos T_{AIC} y $T_{\text{teó}}$ frente a los datos experimentales ($P = 4$ y $Q = 5$)

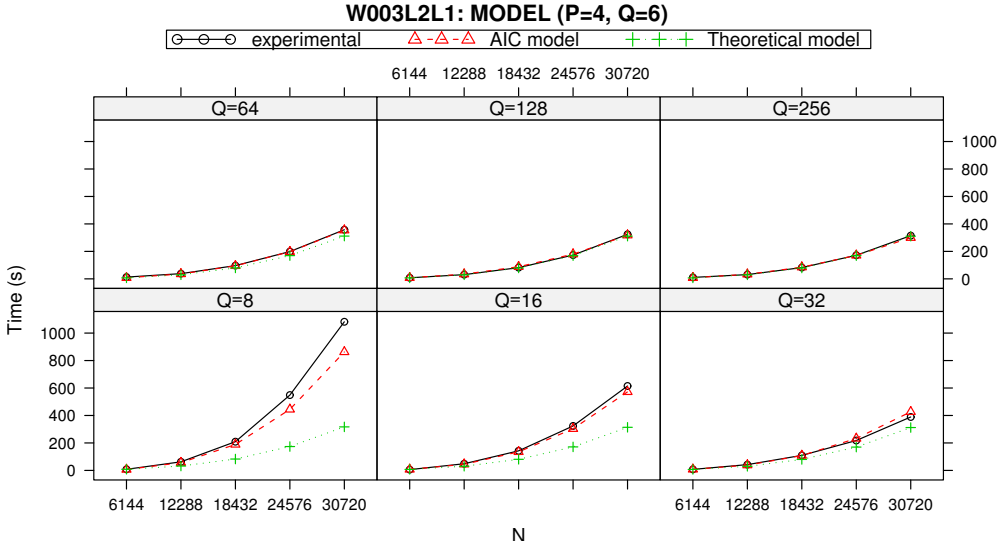


Figura 5.13: Modelos T_{AIC} y $T_{\text{teó}}$ frente a los datos experimentales ($P = 4$ y $Q = 6$)

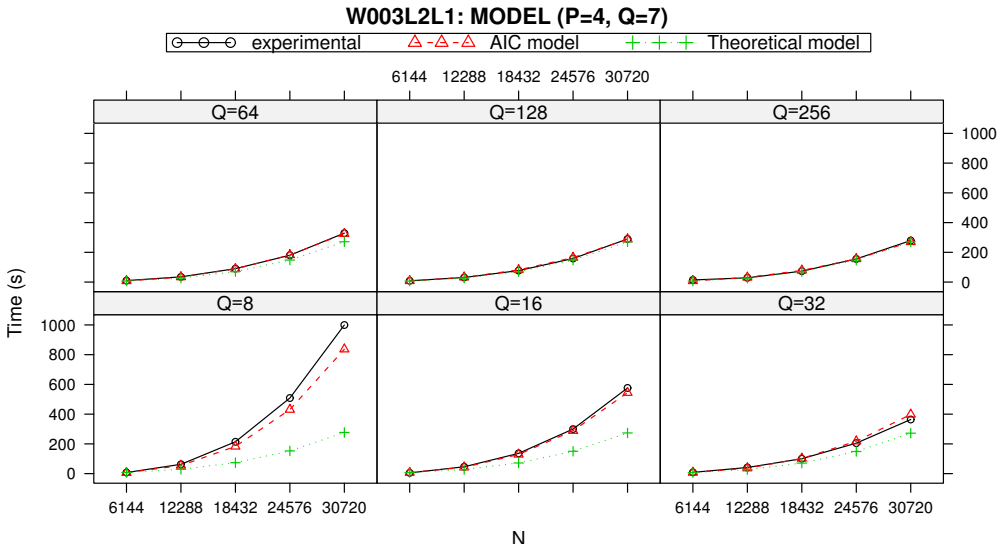


Figura 5.14: Modelos T_{AIC} y $T_{\text{teó}}$ frente a los datos experimentales ($P = 4$ y $Q = 7$)

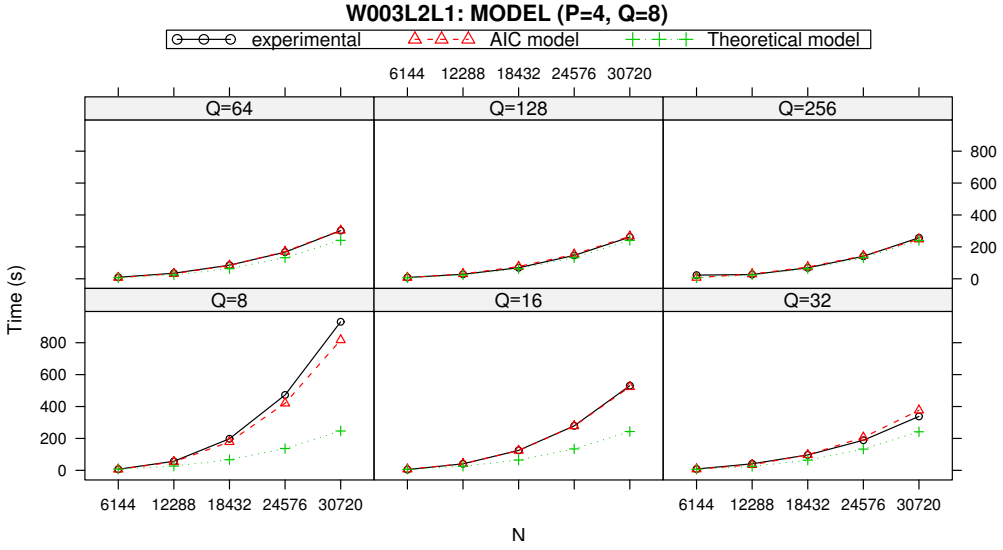


Figura 5.15: Modelos T_{AIC} y $T_{\text{teó}}$ frente a los datos experimentales ($P = 4$ y $Q = 8$)

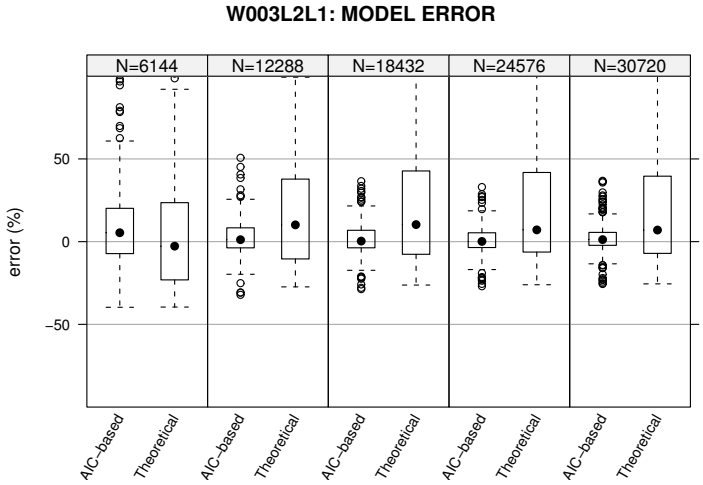


Figura 5.16: Distribución de los residuos del ajuste de T_{AIC} y $T_{\text{teó}}$ en función del parámetro N

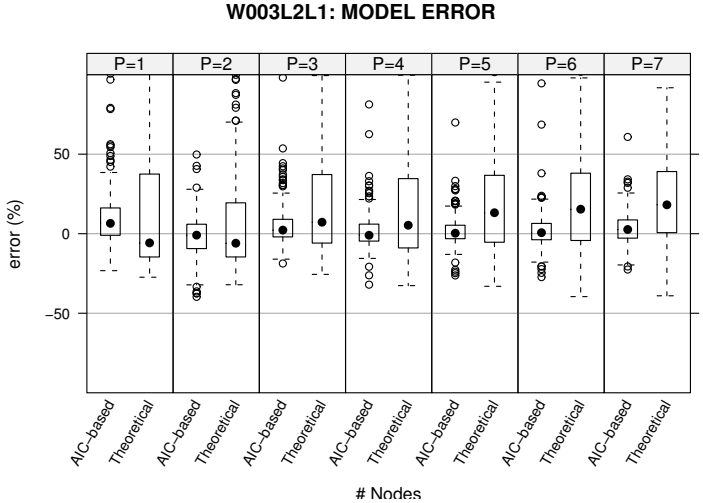


Figura 5.17: Distribución de los residuos del ajuste de T_{AIC} y $T_{\text{teó}}$ en función del parámetro P

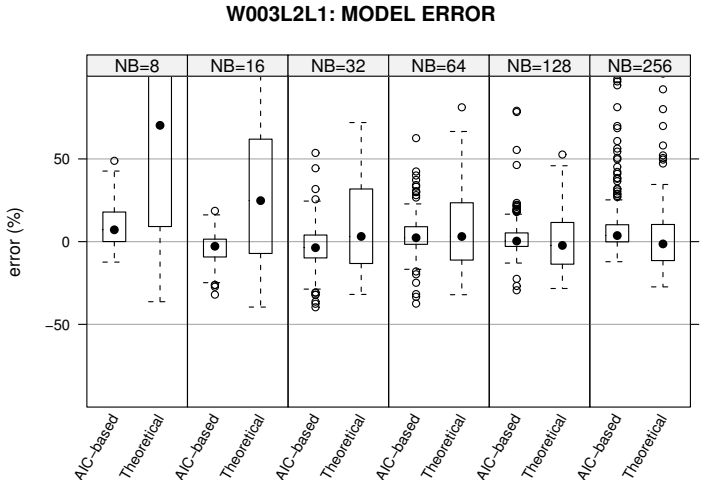


Figura 5.18: Distribución de los residuos del ajuste de T_{AIC} y $T_{\text{teó}}$ en función del parámetro *NB*

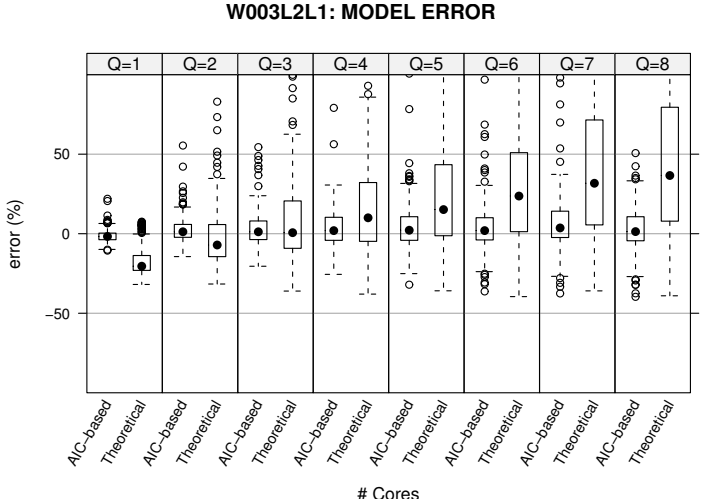


Figura 5.19: Distribución de los residuos del ajuste de T_{AIC} y $T_{\text{teó}}$ en función del parámetro Q

La principal diferencia entre estas dos aproximaciones de modelado reside en la naturaleza de las simplificaciones consideradas para obtener los modelos analíticos. En cualquier caso, estas simplificaciones son necesarias porque el modelo *real* tiene un elevado número de variables y es necesario transformarlo en un modelo manejable. Por un lado, $T_{\text{teó}}$ surge de un desarrollo puramente teórico, basado en la implementación del código, realizando asunciones acerca del comportamiento del sistema (por ejemplo, que no haya fallos cache ni de TLB o que el tiempo de comunicaciones se comporte según el modelo de Hockney, que no haya solapamiento entre comunicaciones y computaciones, etc. [45]). En cambio, el modelo obtenido por selección de modelos (T_{AIC}) simplemente restringe la forma en la que cada parámetro actúa sobre el comportamiento del binomio aplicación-sistema, según se le haya indicado en la lista inicial ($\texttt{LI}_{\text{HPL}}$). En ambos casos, estas simplificaciones introducen una incertidumbre en el resultado final, cuya magnitud será tanto mayor cuanto más distantes estén de la realidad las asunciones consideradas. Sin embargo, en la selección de modelos es el propio método el que, de una forma objetiva, determina la *calidad* del modelo, mientras que en un desarrollo teórico la responsabilidad recae exclusivamente en el analista, que debe decidir en función de su conocimiento del código y del sistema, así como de su experiencia.

El hecho de que T_{AIC} sea más complejo que $T_{\text{teó}}$ también está directamente relacionado con la naturaleza de las asunciones realizadas en el entorno TIA. Al proponer una lista inicial, se establece un número limitado de potenciales comportamientos de las variables consideradas, presuponiendo que el comportamiento de la aplicación —o al menos sus efectos principales— puede ser descrito mediante una combinación determinada de los elementos de la lista, de acuerdo con el mecanismo de descripción de modelos del entorno TIA (véase la sección 2.5.1). Cuanto mayor sea la lista inicial, mayor capacidad habrá para caracterizar el comportamiento real de la aplicación, pero a costa de obtener expresiones más complejas. Sin embargo, el método de selección de modelos buscará, de todas formas, el modelo que mejor se ajuste al comportamiento experimental la aplicación y que, simultáneamente, sea más adecuado para futuras inferencias. En consecuencia, el resultado del proceso de selección contendrá la mejor combinación (arbitrariamente compleja) de elementos de la lista inicial que mejor se adapte al comportamiento real. En cualquier caso, T_{AIC} no debe interpretarse como un «modelo absoluto», sino como la mejor aproximación, dentro del conjunto de modelos considerado, para caracterizar el comportamiento real del *benchmark* en este cluster concreto.

5.2.3 Simulación de la distribución de trabajos HPL en un cluster

La distribución de trabajos en un cluster (planificación o *scheduling*) no influye en la ejecución ni en el resultado de los propios trabajos, pero sí tiene una influencia significativa en la eficiencia del sistema. En esta sección se estudiará, mediante la simulación de un entorno de ejecución real, cómo las predicciones basadas en modelos analíticos del *benchmark* HPL pueden ser usados para mejorar la distribución de tareas de un cluster.

Las simulaciones han sido realizadas en GridSim [152], un *toolkit* de simulación, desarrollado en Java, que ha sido diseñado para estudiar el comportamiento de los algoritmos de *scheduling* en un entorno controlado y reproducible. Este simulador, basado en eventos discretos, permite simular y modelar recursos, usuarios y aplicaciones. Algunos de los algoritmos de *scheduling* locales más utilizados [103] han sido implementados dentro del entorno GridSim [6]. Estos algoritmos son los siguientes:

- *First Come-First Served* (FCFS). Es una política de distribución, ampliamente utilizada en entornos de producción, en la que todos los trabajos son ejecutados en el mismo orden en que son recibidos, sin ningún tipo de preferencia o privilegio.

- *First Fit*. Es una política muy agresiva en la que se ejecuta el primer trabajo (según el orden de llegada) que sea adecuado según los recursos disponibles en ese momento.

- *Backfilling*. Al igual que FCFS, esta estrategia sigue el orden de llegada de los trabajos. Sin embargo, en este caso se promocionan aquellas tareas que encajen dentro de los recursos ociosos, cuando el siguiente trabajo de la cola no pueda ser ejecutado. Esta promoción no debe retrasar la ejecución del resto de los trabajos encolados. Se han implementado dos aproximaciones diferentes para la promoción de los trabajos:

  - *First Fit*: se promociona el primer trabajo adecuado dentro del propio orden de la cola.

  - *Predictive*: se promociona el trabajo que sea adecuado y no retrase la mejor posibilidad de ejecución de trabajos previamente encolados.

- *EASYBackfilling*. Es una técnica agresiva que promociona los trabajos siempre que no se retrase la ejecución del primer trabajo de la cola.

Las políticas de *backfilling* son habituales en sistemas en los que se ejecutan trabajos paralelos porque reducen los casos de *starvation* —como, por ejemplo, que haya trabajos que no puedan acceder a recursos ociosos porque están bloqueados por otros trabajos con mayor prioridad— sin un cambio significativo en las prioridades de los trabajos [161]. Para utilizar de forma eficiente todos los recursos disponibles, estos algoritmos necesitan una estimación precisa de los tiempos de ejecución de cada proceso. Aunque es evidente que la precisión de las predicciones influye de manera directa en la eficiencia del proceso, el tiempo de evaluación del modelo también debe ser considerado en los procesos de *scheduling*, lo que invalida el uso de modelos basados en simulación. Los modelos analíticos generados por el entorno TIA son adecuados en este tipo de situaciones.

Configuración experimental

En este caso de estudio se simula el comportamiento del cluster `burdeos` al ejecutar trabajos HPL, utilizando los modelos analíticos T_{AIC} y $T_{\text{teó}}$ para predecir el tiempo de ejecución del *benchmark*. En concreto, se simulará el comportamiento del cluster para las diferentes políticas de *scheduling* implementadas en GridSim, usando la correspondiente estimación del tiempo de ejecución en las técnicas de *backfilling*. Al igual que en el cluster real, no se permitirá que un nodo tenga simultáneamente más de una aplicación MPI asignada. Los resultados obtenidos serán analizados desde el punto de vista del usuario y del sistema.

La simulación consta de dos procesos asíncronos. Por un lado, se emula la recepción de los trabajos enviados por los usuarios, que serán añadidos a la cola de tareas. Por otro, se simula el funcionamiento de los nodos y el sistema decide, utilizando la estrategia de *scheduling* correspondiente, el trabajo de la cola que debe ser enviado a los nodos para su ejecución cuando existan recursos disponibles. Para evitar problemas de interpretación, siguiendo el punto de vista del sistema, utilizaremos el término «recepción» para referirnos al proceso por el cual los usuarios envían trabajos a la cola del sistema y el término «envío» para el proceso por el cual el sistema envía trabajos desde la cola a los nodos del cluster para su ejecución.

La cancelación de trabajos supone una utilización improductiva de los recursos del sistema y, por lo tanto, tiene un impacto dramático en el rendimiento del sistema. En cualquier caso, las cancelaciones debidas a una predicción errónea deben ser evitadas ya que, debido a la naturaleza de las técnicas de *backfilling*, una predicción errónea en un trabajo puede afectar al envío de otros trabajos. Este aspecto debe contemplarse en el planificador para realizar una correcta estimación del tiempo de ejecución del *benchmark* HPL. Teniendo en cuenta el modo

en que se han obtenido los coeficientes de los modelos T_{AIC} y $T_{\text{teó}}$, en este caso de estudio, las predicciones de tiempo de ejecución de los modelos se han sobrestimado en una cantidad proporcional a la desviación estándar (σ) del ajuste de los modelos.

La carga de trabajo de estas simulaciones no debería estar formada únicamente por ejecuciones del *benchmark* HPL, porque esta situación sería completamente artificial y poco representativa. Para emular un entorno más realista, la carga de trabajo de las simulaciones se construyó a partir del modelo empírico propuesto por Jann [93], basado en la carga real de un cluster en producción en 1997, que determina la distribución del tiempo entre recepciones consecutivas, así como la distribución del número de nodos solicitados y del tiempo de ejecución de cada trabajo. En particular, las cargas de trabajo de las diferentes simulaciones fueron construidas intercalando un determinado número de trabajos HPL dentro de una lista de trabajos generada siguiendo las distribuciones del modelo de Jann. Por lo tanto, la carga de trabajo final consistirá en dos tipos diferentes de trabajos. Un determinado porcentaje de trabajos serán trabajos HPL, mientras que el resto serán trabajos generados artificialmente siguiendo las distribuciones impuestas en el modelo de Jann (denotaremos como «no-HPL» a este tipo de trabajos).

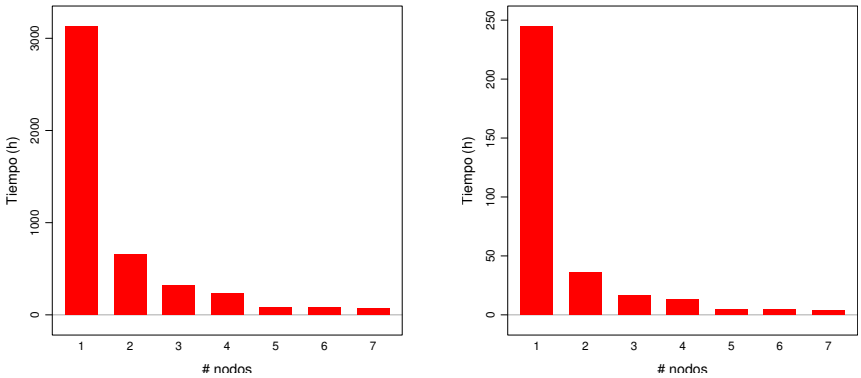
Los datos de tiempo de ejecución y número de procesadores de los trabajos HPL se han obtenido de ejecuciones reales en el cluster `burdeos` —ejecuciones distintas a las utilizadas para obtener los modelos—, usando diferentes configuraciones de los parámetros de ejecución N, N_B y $P \times Q$. Asimismo, se han seleccionado un determinado número de estas ejecuciones para que los trabajos HPL también sigan la distribución de número de nodos del modelo de Jann. De esta forma, en la carga de trabajo resultante, todos los trabajos siguen las distribuciones de tiempo entre sucesivas recepciones y de número de nodos del modelo de Jann. Sin embargo, la distribución del tiempo de ejecución de la carga resultante no es homogénea ya que los trabajos de HPL, al estar basados en ejecuciones reales, son independientes del modelo de Jann. En los procesos no-HPL, la estimación del tiempo de ejecución se supondrá perfecta, es decir, la predicción coincidirá exactamente con el tiempo de ejecución.

En primer lugar, se construyó una carga de trabajo con 1 674 trabajos, de los cuales 521 son trabajos HPL (aproximadamente, el 30 %). Sin embargo, al utilizar los coeficientes descritos del modelo original de Jann [93] para generar los trabajos artificiales, surge un fuerte desbalanceo en el tiempo de ejecución de la carga de trabajo: los trabajos HPL consumen únicamente un 2,2 % de los recursos del cluster. En esta situación, los trabajos HPL estarían compitiendo con un porcentaje considerable de trabajos cuyo tiempo de ejecución es muy superior. Realmente, estamos comparando tiempos de ejecución en un cluster actual con

tiempos de ejecución generados a partir de datos obtenidos en un cluster hace más de 10 años. Esta asimetría en los tiempos de ejecución podría darse en sistema reales, pero representa una situación muy particular. Este escenario de simulación lo denominaremos LOW.

Los coeficientes del trabajo original de Jann pueden interpretarse como una instancia particular de un cluster que siga el modelo de Jann. Bajo este punto de vista, la modificación de los valores de los coeficientes, sin alterar la estructura de las distribuciones establecidas en el propio modelo, puede interpretarse como una nueva instancia de un cluster diferente. Por ejemplo, disponer de procesadores con el doble de capacidad de cómputo implicaría, a priori, una reducción del tiempo de ejecución de los trabajos. Bajo esta aproximación, es posible escalar los coeficientes del modelo para generar una carga de trabajo en la que el 30 % de los trabajos sean trabajos HPL y que éstos representen un 30 % del consumo de los recursos. En concreto, para obtener estos porcentajes en la carga de trabajo resultante, se han escalado los valores de los coeficientes de la distribución de tiempo de ejecución, pero también, y en la misma proporción, la distribución de tiempo entre recepciones consecutivas, conservando la relación entre ambas distribuciones. En esta nueva situación, los trabajos HPL estarían compitiendo por los recursos con otros trabajos de similares características. Este nuevo escenario de simulación lo denominaremos HIGH.

Por lo tanto, se han considerado dos escenarios diferentes, LOW y HIGH, caracterizados por cargas de trabajo diferentes. La figura 5.20 muestra la distribución del tiempo de ejecución de la carga de trabajo, en función del número de procesos, de los dos escenarios considerados (incluyendo trabajos HPL y no-HPL). Ambos escenarios presentan una distribución muy similar —en los dos casos, las cargas de trabajo siguen la distribución del número de nodos del modelo de Jann—, pero la suma del tiempo de ejecución de todos los trabajos es diferente. En el escenario LOW, la suma del tiempo de ejecución de todos los trabajos de la carga es superior a 3 500 horas, mientras que en el escenario HIGH apenas llega a las 300 horas. Esta diferencia es consecuencia directa del escalado de los tiempos de ejecución de los trabajos no-HPL. De este modo, el peso de los trabajos HPL tendrá una influencia diferente en cada escenario. En el escenario LOW los trabajos HPL representan en torno a un 2 % del uso del cluster, mientras que en el escenario HIGH este porcentaje aumenta hasta un 30 %. Por otro lado, los escenarios LOW y HIGH también presentan diferencias en el tiempo entre recepciones consecutivas. Aunque las cargas de trabajo de los dos escenarios siguen la correspondiente distribución del modelo Jann, los tiempos entre recepciones consecutivas del entorno HIGH han sido reducidos en el mismo factor que el tiempo de ejecución de los trabajos no-HPL.



(a) Escenario LOW (b) Escenario HIGH

Figura 5.20: Distribución del tiempo de ejecución en función del número de procesos

Análisis de los resultados

La tabla 5.6 muestra, para cada escenario, los tiempos de simulación de las diferentes políticas de *scheduling* del entorno GridSim. En esta tabla se aprecia la diferencia entre los tiempos de simulación de los escenarios LOW y HIGH, debida a las diferentes cargas de trabajo consideradas. En las políticas de *backfilling* se muestran los tiempos de simulación en función de la sobrestimación de la predicción. En cada escenario, estas políticas presentan unos tiempos de simulación similares entre sí y, teniendo presente que los trabajos no-HPL tienen una predicción perfecta —es decir, el tiempo de ejecución coincide exactamente con el tiempo predicho—, las únicas diferencias entre las diferentes estrategias de *backfilling* son debidas al diferente tratamiento de los trabajos HPL. Por otro lado, sus tiempos de simulación son sustancialmente mejores que los de la estrategia FCFS y, además, son comparables con los resultados de *First Fit* —esta estrategia se puede interpretar como una referencia del nivel de aprovechamiento de los recursos del cluster, ya que *First Fit* maximiza la utilización de los nodos—.

Independientemente del escenario, en los resultados de la tabla 5.6 no se aprecia una diferencia significativa entre las simulaciones que utilizan en modelo T_{AIC} en sus predicciones frente a las que utilizan el modelo $T_{\mathrm{teó}}$. Al contrario, la mayoría de las simulaciones que utilizan el modelo $T_{\mathrm{teó}}$ presentan una ligera mejora en el tiempo de simulación y, por lo tanto, de la utilización de los recursos del cluster. Sin embargo, esta percepción es falsa desde el punto de vista del rendimiento obtenido por el usuario, ya que las cancelaciones de trabajos debidas

Tabla 5.6: Tiempo de simulación, en horas, para diferentes configuraciones de simulación

Escenario	Modelo	Política	Sobrestimación				
			No-Pred.	1σ	2σ	3σ	4σ
LOW	-	FCFS	1859,2	-	-	-	-
	-	First Fit	1211,7	-	-	-	-
	T_{AIC}	B. predictive	-	1241,8	1241,7	1243,1	1245,6
		B. first fit	-	1236,9	1236,9	1237,5	1237,5
		EASYBackfilling	-	1245,6	1242,2	1242,2	1242,2
	$T_{\text{teó}}$	B. predictive	-	1245,6	1242,5	1243,2	1242,8
		B. first fit	-	1235,3	1231,9	1232,6	1232,6
		EASYBackfilling	-	1241,3	1244,4	1244,2	1244,2
HIGH	-	FCFS	115,8	-	-	-	-
	-	First Fit	76,8	-	-	-	-
	T_{AIC}	B. predictive	-	75,9	76,4	76,8	76,4
		B. first fit	-	74,9	75,6	76,0	76,4
		EASYBackfilling	-	75,7	75,7	75,7	75,5
	$T_{\text{teó}}$	B. predictive	-	73,7	74,8	75,3	76,2
		B. first fit	-	73,1	74,2	74,9	75,1
		EASYBackfilling	-	72,9	74,4	74,9	75,3

a predicciones erróneas no están siendo consideradas. La ejecución de trabajos que finalmente son cancelados reduce el rendimiento efectivo del sistema, independientemente del nivel de utilización de los recursos del cluster, ya que el uso de los recursos realizado por los trabajos cancelados es absolutamente improductivo. La tabla 5.7 muestra, para las diferentes sobrestimaciones consideradas, el número de trabajos cancelados debidos a estimaciones erróneas —el número de cancelaciones es independiente del escenario considerado, ya que la estimación del tiempo de ejecución de los trabajos HPL es igual en ambos escenarios—. También se muestra el tiempo de ejecución total asociado a los trabajos HPL cancelados. En esta tabla se aprecia como, independientemente de la sobrestimación, las predicciones obtenidas con el modelo $T_{\text{teó}}$ presentan un número muy superior de cancelaciones que, obviamente, implican un uso improductivo de los nodos muy significativo.

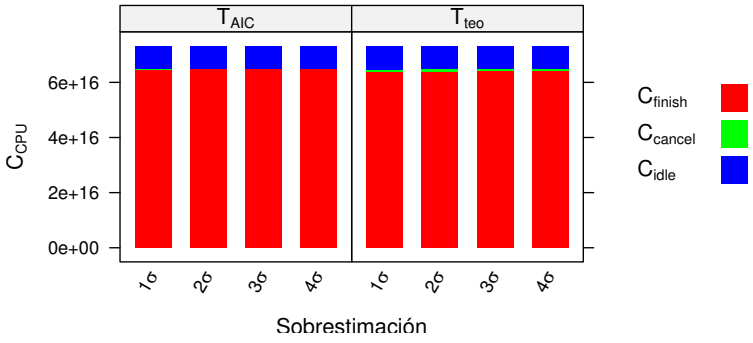
La figura 5.21 muestra el uso del cluster, desde el punto de vista del sistema, durante las simulaciones del escenario LOW para diferentes políticas de *backfilling* y utilizando di-

Tabla 5.7: Número de trabajos cancelados debido a predicciones erróneas y tiempo de ejecución asociado (en minutos)

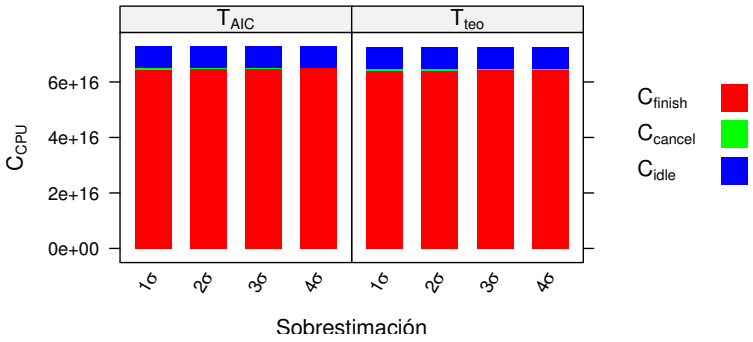
Sobrestimación	T_{AIC}		$T_{\text{teó}}$	
	Cancelaciones	Tiempo (m)	Cancelaciones	Tiempo (m)
1σ	110	1837	340	2651
2σ	21	402	237	2430
3σ	1	1	178	2104
4σ	0	0	131	1881

ferentes sobrestimaciones de los modelos T_{AIC} y $T_{\text{teó}}$. La figura 5.22 muestra los resultados análogos para el escenario HIGH. En estas figuras C_{CPU} representa los ciclos de procesador gestionados durante la simulación. Teniendo en cuenta que un nodo no puede ejecutar más de un proceso simultáneamente, el máximo valor de C_{CPU} se corresponde con la suma de todos los ciclos gastados por un procesador durante la simulación multiplicado por el número de nodos del sistema. Los ciclos de cada simulación se clasifican según su productividad: C_{finish} indica los ciclos consumidos por procesos que finalizaron su ejecución correctamente, mientras que C_{cancel} se refiere a los ciclos utilizados por procesos que han sido cancelados y que han supuesto un gasto improductivo de los nodos correspondientes. Además, C_{idle} caracteriza aquellos ciclos que no han sido usados por ningún proceso, es decir, que no han podido ser asignados a ningún trabajo. Como en estas simulaciones las cancelaciones sólo afectan a los trabajos HPL, el efecto de las cancelaciones tiene una mayor relevancia en el escenario HIGH. Recuerdese que en el escenario LOW los trabajos HPL representan únicamente un $2,2\,\%$ de la carga de trabajo global. Por otro lado, en estas figuras también se aprecia como, a pesar del aumento de la sobrestimación en la predicción, las cancelaciones debidas a una predicción errónea del modelo $T_{\text{teó}}$ tienen una influencia significativa en el rendimiento del sistema. Sin embargo, el modelo T_{AIC} alcanza, gracias a la mayor precisión de este modelo, una simulación libre de cancelaciones con una sobrestimación de 4σ.

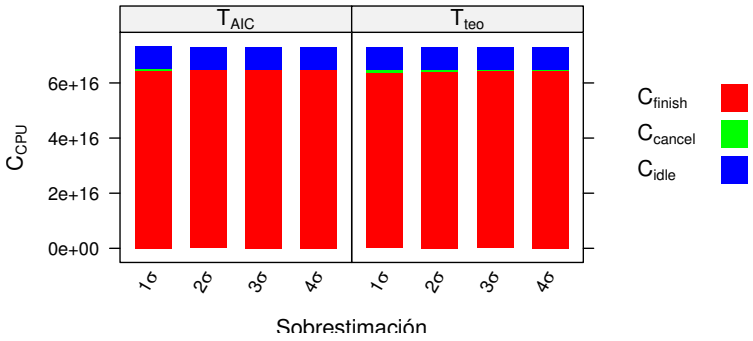
El impacto de las cancelaciones en el rendimiento global del sistema es difícil de cuantificar y depende en gran medida de las políticas de gestión del cluster. Por ejemplo, una cancelación puede suponer un esfuerzo inútil de los recursos del sistema, devolviendo el trabajo al usuario sin ningún resultado significativo. En general, es imposible establecer, en términos de rendimiento, una comparación directa entre un trabajo cuya ejecución finalice correctamente y otro cancelado, ya que este último realmente no produce ningún resultado efectivo.



(a) B. predictive

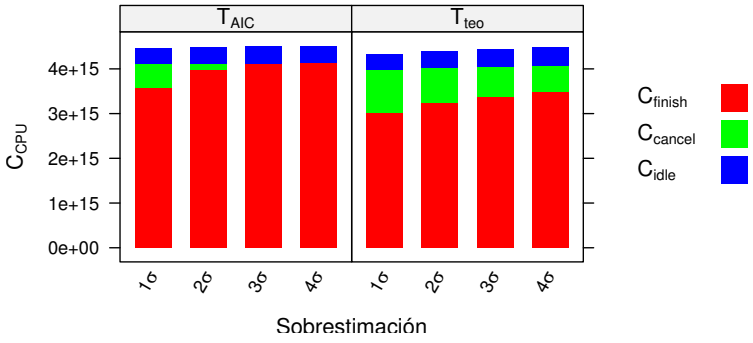


(b) B. First Fit

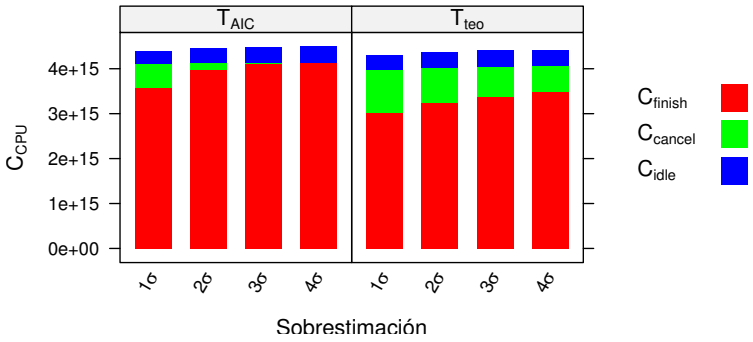


(c) EASYBackfilling

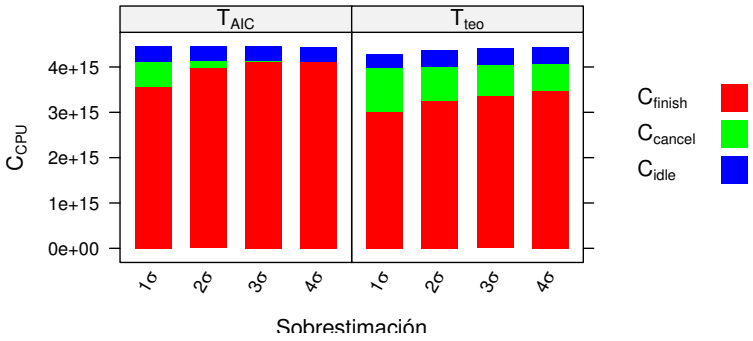
Figura 5.21: Uso del cluster de las estrategias de *backfilling* en el escenario LOW



(a) B. predictive



(b) B. First Fit



(c) EASYBackfilling

Figura 5.22: Uso del cluster de las estrategias de *backfilling* en el escenario HIGH

Tabla 5.8: Resultados de las simulaciones para los diferentes escenarios y políticas de *backfilling*

		Simulación (horas)				Beneficios AIC		
Escenario	Algoritmo	T_{sim}^{AIC}	$\sum T_{wait}^{AIC}$	$T_{sim}^{teó}$	$\sum T_{wait}^{teó}$	% jobs	T_{wait}(%)	T_{idle}(%)
LOW	B. predictive	1245,6	8490	1256,5	10259	40,07	17,24	0,77
	B. first fit	1237,5	9400	1243,3	11333	50,48	17,06	0,42
	EASYB.	1242,2	9070	1247,4	10806	39,69	16,07	0,37
HIGH	B. predictive	76,4	3141	77,9	3888	66,47	19,21	1,68
	B. first fit	76,4	3106	76,7	3668	61,66	15,37	0,36
	EASYB.	75,5	2951	76,7	3719	68,98	20,65	1,39

Por lo tanto, para poder realizar una evaluación adecuada de los modelos T_{AIC} y $T_{\text{teó}}$ como estimadores del tiempo de ejecución en las políticas de *backfilling*, es necesario que las predicciones consideradas no produzcan ninguna cancelación. En concreto, siguiendo el criterio de predicción establecido, se han considerado las sobrestimaciones mínimas en las que no haya ninguna cancelación. Estas sobrestimaciones se corresponden con 4σ y 12σ para los modelos T_{AIC} y $T_{\text{teó}}$, respectivamente.

La tabla 5.8 resume los resultados de las simulaciones para las diferentes políticas de *backfilling*, en los dos escenarios considerados, utilizando predicciones que no producen cancelaciones —es decir, sobrestimando 4σ la predicción del modelo T_{AIC} y 12σ la predicción del modelo $T_{\text{teó}}$—. En la tabla se muestra el tiempo de simulación total (T_{sim}) y la suma de los tiempos de espera en cola de los trabajos HPL (denotado como $\sum T_{wait}$) para las estimaciones en T_{AIC} y $T_{\text{teó}}$ (denotadas con los superíndices «AIC» y «teo», respectivamente). Las últimas tres columnas muestran el beneficio de la estimación basada en el modelo T_{AIC} frente a la estimación basada en el modelo teórico. Las dos primeras columnas son el porcentaje de trabajos HPL que tienen un menor tiempo de espera en la cola (% jobs) y el porcentaje de mejora del tiempo de espera en los trabajos HPL (T_{wait}). La última columna (T_{idle}) se corresponde con el porcentaje de mejora del tiempo en la simulación total, incluyendo los trabajos no-HPL.

En el escenario LOW, aunque el tiempo total simulado es similar y el porcentaje de trabajos HPL en los que la estimación AIC presenta mejores resultados es próximo al 50 %, el tiempo total de espera se reduce significativamente en las tres simulaciones. Este hecho significa que la precisión de las estimaciones del tiempo de ejecución basadas en el modelo T_{AIC} permiten que el planificador maneje más cuidadosamente los trabajos HPL, especialmente aquellos con un tiempo de ejecución elevado. En consecuencia, se obtiene una mejor utiliza-

ción de los recursos, con una ligera reducción en el porcentaje de utilización de los recursos del cluster, a pesar de que el margen de mejora es pequeño.

En el caso del escenario HIGH, los resultados también muestran que las estimaciones basadas en el modelo T_{AIC} proporcionan un menor tiempo de simulación. Desde el punto de vista del usuario, en todos los casos, la utilización del modelo T_{AIC} reduce significativamente el tiempo total de espera de los trabajos HPL y, además, en un mayor porcentaje de los trabajos (en torno al 60 %). Desde el punto de vista del sistema, la mejora en la utilización de los recursos (T_{idle}) es mejor que el escenario LOW para las estrategias *backfilling predictive* y *EASYbackfilling*. Sin embargo, el porcentaje de utilización del algoritmo *backfilling first fit* es peor que en el escenario LOW debido a que esta estrategia no evalúa completamente la cola (simplemente localiza el primer trabajo que se adapte a los recursos disponibles, sin hacer ninguna previsión para futuros envíos). Por otra parte, el algoritmo *backfilling predictive* proporciona una mejor utilización de los recursos del cluster en ambos escenarios. Este algoritmo evalúa completamente la cola del sistema y, de este modo, gracias a una predicción más precisa del tiempo de ejecución, se obtiene un mejor aprovechamiento de los recursos del sistema y un mayor rendimiento desde el punto de vista del usuario.

5.3 Influencia de los fallos cache

Aunque la métrica más habitual, en términos de análisis del rendimiento, es el tiempo de ejecución, el entorno TIA proporciona al usuario la capacidad de obtener modelos estadísticos de rendimiento de cualquier métrica disponible a través de los *drivers* de CALL. En concreto, el *driver* PAPI proporciona el mecanismo apropiado para medir diferentes aspectos del rendimiento de la memoria cache en los microprocesadores actuales como, por ejemplo, el número de fallos cache de lectura/escritura en los diferentes niveles [28].

Los fallos cache no se pueden utilizar para predecir el tiempo de ejecución, ya que esta métrica es desconocida antes de ejecutar la aplicación. Sin embargo, utilizando el entorno TIA, es posible obtener un modelo de los fallos cache que pueda utilizarse en tareas de predicción. Por otro lado, si se identifica la contribución de los fallos cache al modelo del tiempo de ejecución, se puede determinar la influencia de estos en el rendimiento de la aplicación. Utilizando el entorno TIA, se ha diseñado una metodología para modelar la influencia de los fallos cache en el tiempo de ejecución de una aplicación paralela [120]. En cualquier caso, esta metodología no está restringida únicamente a la caracterización de los fallos cache, sino

que podría aplicarse a cualquier otro parámetro arquitectural cuyo valor no se conozca antes de la ejecución de una aplicación.

En primer lugar, el tiempo de ejecución y el número de fallos cache son medidos a través de los drivers de TIA, así como otros parámetros de ejecución de la aplicación paralela como, por ejemplo, el número de procesos o el tamaño de problema. Una vez conocidos los valores experimentales de las diferentes métricas y parámetros de rendimiento, se obtienen los modelos estadísticos del tiempo de ejecución y de los fallos cache mediante el método de selección de modelos implementado en la fase de análisis del entorno TIA. Por un lado, el modelo de los fallos cache se obtiene como una función de los diferentes parámetros de ejecución medidos. Por otro lado, se construye el modelo del tiempo de ejecución como una función de los parámetros de ejecución y de los fallos cache —es decir, los fallos cache se consideran como un nuevo parámetro de ejecución independiente—. La variable que caracteriza los fallos cache en la expresión del modelo del tiempo de ejecución podrá ser sustituida por el correspondiente modelo. Por lo tanto, la expresión final del tiempo de ejecución de la aplicación sólo dependerá de los parámetros de ejecución. Por lo tanto, estas expresiones pueden evaluarse antes de ejecutar la aplicación, proporcionando un mecanismo adecuado para la predicción del rendimiento de la aplicación.

5.3.1 Producto paralelo de matrices

La estimación de la influencia de los fallos cache en la ejecución de aplicaciones paralelas ha sido evaluada en dos versiones diferentes del producto paralelo de matrices. En particular, estas dos versiones utilizan números en punto flotante de simple precisión y las comunicaciones entre los diferentes procesos se realiza a través de funciones MPI. La figura 5.23 muestra el pseudo-código de estas dos versiones, donde el tamaño de las matrices es $N \times N$, siendo P el número de procesos. En ambos casos, se ha supuesto que la distribución de las matrices `X` e `Y` se ha realizado en un paso previo y que el reparto de la matriz `X` ha sido realizado por bloques, de forma equitativa, entre los diferentes procesos implicados. Por un lado, el caso A —representado en la figura 5.23(a)— muestra una situación en la que todo el trabajo computacional es realizado intensivamente antes de iniciar el envío de los resultados parciales, mediante una única comunicación colectiva. Por otro lado, el caso B —representado en la figura 5.23(b)— muestra una situación en la que el número de comunicaciones es muy elevado, ya que cada fila de la matriz `Z` es enviada al proceso raíz inmediatamente después de calcularse en el proceso correspondiente. Por lo tanto, las diferencias entre los dos casos

```
1  /* Parallel loop */
2  for (i in 1 to N/P) do
3
4    for (j in 1 to N) do
5      Z_{i,j}=0
6      for (k in 1 to N) do
7        Z_{i,j}+=X_{i,k}*Y_{k,j}
8      end for
9    end for
10
11 end for
12
13 MPI_Barrier
14 MPI_Gather(Z)
15 MPI_Barrier
```

(a) Caso A, computación intensiva

```
1  /* Parallel loop */
2  for (i in 1 to N/P) do
3
4    for (j in 1 to N) do
5      Z_{i,j}=0
6      for (k in 1 to N) do
7        Z_{i,j}+=X_{i,k}*Y_{k,j}
8      end for
9    end for
10
11   MPI_Barrier
12   MPI_Gather(Y_{i,*})
13   MPI_Barrier
14
15 end for
```

(b) Caso B, comunicación intensiva

Figura 5.23: Pseudo-código de las dos versiones del producto paralelo de matrices ($Z = X \times Y$), donde el tamaño de matrices es $N \times N$, y P es el número de procesos

están en el número y tamaño de las funciones globales de comunicación (`MPI_Gather`), que envían los resultados parciales de la matriz resultado al proceso raíz. El caso A realiza una única comunicación global en la que envía la matriz completa, mientras que en el caso B, se realizan `N/P` comunicaciones globales y en cada una de ellas se envían `P` filas de la matriz resultado. Se han incluido las funciones `MPI_Barrier`, antes y después de cada función de comunicación, para reducir los posibles solapamientos entre la comunicación y la computación.

Ambos códigos han sido instrumentados con los *pragmas* de la herramienta `call` utilizando los *drivers* PAPI, MPI y NWS. El *driver* PAPI ha sido utilizado para obtener los fallos cache registrados durante las distintas ejecuciones. Asimismo, para obtener una mayor precisión, el tiempo de ejecución ha sido medido utilizando el observable `PAPI_REAL_USEC` del *driver* PAPI. El *driver* MPI es necesario para la correcta gestión de los observables obtenidos con PAPI. El *driver* NWS se ha utilizado para obtener el estado de la red de comunicación, proporcionando el ancho de banda y la latencia efectiva de la red justo antes de la ejecución de los códigos.

Los códigos instrumentados han sido ejecutados en un cluster homogéneo de seis nodos conectados a través de una red Gigabit Ethernet. El controlador de la interfaz de red de los nodos permite al usuario modificar la velocidad de la red, por lo que se han considerado

Tabla 5.9: Valores numéricos considerados en los tres parámetros experimentales

Parámetro	Valores
Número de procesadores	2, 3, 4, 5, y 6
Dimensión de la matrices	300, 400 y 500
Velocidad de la red	10, 100 y 1000 Mbps

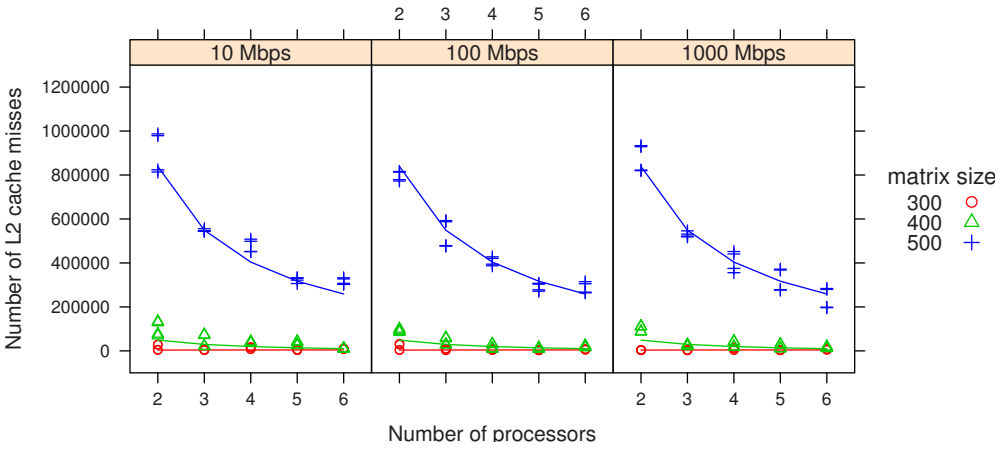
tres situaciones de red diferentes, utilizando diferentes parámetros en la configuración de este controlador. Los nodos del cluster están equipados con procesadores Intel® Pentium® 4. Este procesador dispone de 2 niveles de cache: L1 de 32 KB (16 KB de datos) y L2 de 1 MB. La latencia asociada a un fallo en el nivel L2 es, aproximadamente, un orden de magnitud mayor que la latencia del nivel L1 [137], por lo que sólo consideraremos los fallos de nivel L2 en este estudio. Cada aplicación ha sido ejecutada con tres tamaños de matriz diferentes para cada configuración de velocidad de la red de interconexión y número de procesos. La tabla 5.9 muestra, para cada parámetro experimental, los valores utilizados en este estudio. Para cada configuración experimental se han realizado cuatro ejecuciones independientes de la aplicación instrumentada.

Modelo del número de fallos cache L2

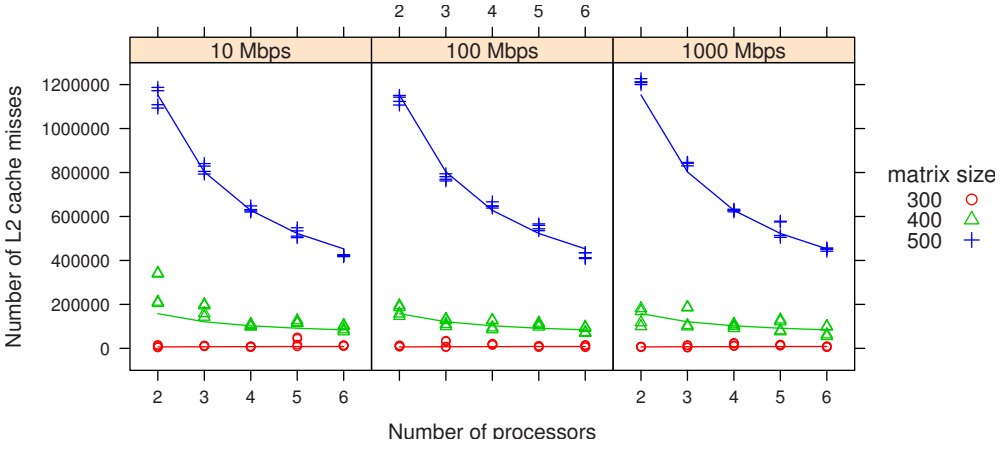
Teniendo en cuenta que la principal causa de fallos cache L2 es el volumen de operaciones MAC (multiplicar y acumular), los límites de los diferentes lazos deberían ser los factores fundamentales a tener en cuenta para la construcción de la lista inicial (`LI`) del método de selección de modelos del entorno TIA. Como la estructura de lazos es similar en los dos casos, la lista inicial propuesta para ambos casos es:

$$\texttt{LI} = \left\{ \frac{N}{P} \right\}, \{N^2, N\}$$

El primer elemento de la lista, $\{\frac{N}{P}\}$, se corresponde con el tamaño de lazo más externo. El segundo elemento, $\{N^2, N\}$, representa el número de instrucciones MAC que se ejecutan en cada instancia de los dos lazos internos. El modelo del número de fallos cache de nivel L2 —que denominamos Φ—, obtenido automáticamente mediante la selección de modelos implementada en TIA, se muestra en la tabla 5.10 para los dos casos considerados. Las figuras 5.24(a) y 5.24(b) muestran los datos experimentales (puntos) y el modelo obtenido automáticamente (líneas) para los casos A y B, respectivamente.



(a) Caso 1



(b) Caso 2

Figura 5.24: Valores experimentales (puntos) y modelo del número de fallos cache de nivel L2 (líneas)

Tabla 5.10: Modelo del número de fallos cache de nivel L2, obtenido automáticamente mediante el método de selección de modelos implementado en TIA

	Modelo	% Error	$\omega^{\mathtt{CM}}$
Caso A	$\Phi_{\mathrm{A}} = 0{,}144\frac{N^3}{P} - 98\frac{N^2}{P} - 0{,}39N^2 + 16400\frac{N}{P} + 130N$	37.4	0.999
Caso B	$\Phi_{\mathrm{B}} = 0{,}154\frac{N^3}{P} - 102\frac{N^2}{P} + 0{,}9N^2 + 17000\frac{N}{P} - 230N$	26.7	0.981

Los elevados pesos de Akaike indican que, dentro del conjunto de modelos candidatos seleccionado, estos modelos son una buena aproximación del comportamiento real de los fallos cache de nivel L2. En cualquier caso, las figuras muestran que, tanto en el caso A como en el B, como cabía esperar, el comportamiento de los fallos cache es prácticamente independiente de la configuración del controlador de la red. Además, existe una notable similitud entre las expresiones de los dos modelos ya que los coeficientes de los términos $\frac{N^3}{P}$, $\frac{N^2}{P}$ y $\frac{N}{P}$ —los términos dominantes en las expresiones— son similares. Ambos factores son indicadores de que la suposición inicial al respecto de que el número de fallos cache de nivel L2 es independiente de las comunicaciones, es correcta.

Modelo del tiempo de ejecución

Para construir la lista inicial (`LI`) del proceso de selección, en ambos casos, se han analizado de manera independiente las secciones de cómputo y comunicación, porque el solape entre ambas es prácticamente nulo por el uso de las funciones `MPI_Barrier`.

Por un lado, al análisis de los factores de cómputo en ambos casos es similar, ya que ambas aplicaciones ejecutan el mismo número ($\frac{N^3}{P}$) de operaciones MAC. Además, podemos considerar que el coste de la gestión de los lazos es despreciable. La influencia de los fallos cache también deber ser tenida en cuenta en el tiempo de ejecución en esta sección de cómputo, ya que, de acuerdo con la suposición considerada en la obtención de los modelos Φ, los fallos cache están correlacionados con la ejecución de las instrucciones MAC. Se supondrá que la influencia de los fallos cache de nivel L2 es directamente proporcional al número total de fallos cache, que es un dato experimental proporcionado por el *driver* PAPI (variable `PAPI_L2_TCM`). En este análisis se utilizará el símbolo ϕ para referirse al número total de fallos cache de nivel L2 medidos a través de este driver.

Por otro lado, la contribución de las comunicaciones al rendimiento de los distintos códigos merece un análisis detallado, porque las principales diferencias entre ambos códigos reside precisamente en el número y tamaño de las comunicaciones. En concreto, los parámetros relacionados con el número de llamadas a las funciones de comunicación y los parámetros relacionados con el coste de cada comunicación han sido consideradas de manera independiente. El número total de comunicaciones colectivas (`MPI_Gather`) es igual al producto de los límites de los lazos externos a la propia función. En el caso A no hay ningún lazo exterior, mientras que en el caso B la función colectiva está dentro de un lazo cuyo límite es $\frac{N}{P}$.

El análisis del coste asociado a cada función `MPI_Gather` no es inmediato, ya que el rendimiento real depende de las características de la red de interconexión y de algoritmo de comunicación, es decir, de la topología de las comunicaciones punto a punto implementadas internamente en la función colectiva. El rendimiento de las comunicaciones punto a punto puede modelarse mediante los parámetros de latencia (α) y ancho de banda ($1/\beta$), de acuerdo con el modelo de Hockney [74]. Sin embargo, el algoritmo de comunicación colectiva es, a priori, desconocido e incluso podría cambiar dinámicamente según el tamaño de mensaje [53]. En este caso, para cubrir varias posibilidades, se consideran los caminos críticos de los algoritmos lineal y binomial ($\lceil \log_2 P \rceil$ y P, respectivamente), que determinan el tiempo teórico máximo de la función de comunicación. La conexión entre el modelo de Hockney y los distintos algoritmos se realiza siguiendo la misma aproximación que Pjěsivac-Grbovíc et ál. [138]: la latencia multiplica el camino crítico del algoritmo y el ancho de banda divide el tamaño de mensaje global de la función colectiva —esto es, el tamaño del vector en el proceso raíz—. El tamaño global de cada comunicación colectiva es N^2 en el caso A y N en el caso B. Además, tanto los caminos críticos de los dos algoritmos considerados como el tamaño global de la función colectiva han sido considerados como variables individuales y, de este modo, se tiene en cuenta la posibilidad de que los parámetros del modelo de Hockney no influyan significativamente en el rendimiento de los códigos.

Teniendo en cuenta las consideraciones previas, la lista inicial utilizada en el caso A es:

$$\mathtt{LI}_{\mathrm{A}} = \left\{ N^2, \frac{N^2}{\beta}, \lceil \log_2 P \rceil, \alpha \lceil \log_2 P \rceil, P, \alpha P \right\}, \left\{ \frac{N^3}{P}, \phi \right\}^*$$

mientras que para el caso B sería:

$$\mathtt{LI}_{\mathrm{B}} = \left\{ \frac{N}{P} \right\}, \left\{ N, \frac{N}{\beta}, \lceil \log_2 P \rceil, \alpha \lceil \log_2 P \rceil, P, \alpha P \right\}, \left\{ \frac{N^3}{P}, \phi \right\}^*$$

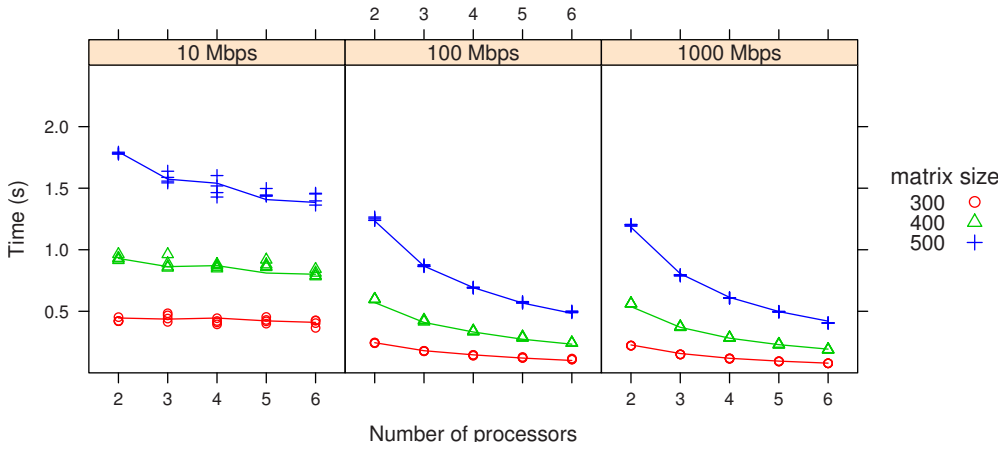
Tabla 5.11: Modelos del tiempo de ejecución, obtenidos automáticamente mediante el método de selección de modelos implementado en TIA

	Modelo	% Error	ω^{CM}
Caso A	$T_{\mathrm{A}}(\mu s) = 18{,}7\frac{N^2}{\beta} + 0{,}18N^2 - 4000\lceil\log_2 P\rceil + 40000\alpha\lceil\log_2 P\rceil - 27000\alpha P + 0{,}0154\frac{N^3}{P} + 0{,}19\phi$	4.4	0.418
Caso B	$T_{\mathrm{B}}(\mu s) = 21\frac{N^2}{P\beta} + 970\frac{N}{\beta} + 14000\lceil\log_2 P\rceil - 130000\alpha\lceil\log_2 P\rceil + 100\frac{N}{P}\lceil\log_2 P\rceil + 800\frac{N}{P}\alpha\lceil\log_2 P\rceil + 0{,}0142\frac{N^3}{P} + 0{,}13\phi$	3.5	0.0246

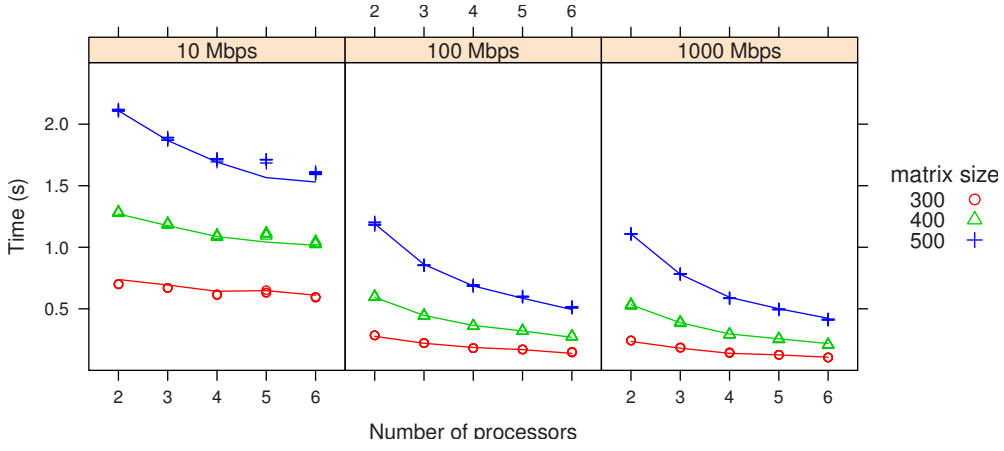
La tabla 5.11 muestra los modelos seleccionados entre el conjunto de modelos candidatos correspondiente, obtenidos automáticamente mediante el método de selección de modelos de TIA. El modelo T_{A} se corresponde con el modelo obtenido en el caso A y el modelo T_{B} con el obtenido en el caso B. La figura 5.25 muestra, para ambos casos, los valores experimentales de tiempo de ejecución (puntos) y el modelo obtenido con TIA (líneas).

Los términos asociados al número de operaciones MAC y al número de fallos cache son muy similares en ambos casos, aunque ligeramente menores en T_{B} respecto de T_{A}. Esta pequeña diferencia es consecuencia directa de la menor influencia de estos factores en el tiempo de ejecución del caso B. El coste de las comunicaciones colectivas es mayor que el coste asociado al cálculo de una fila de la matriz resultado y, por lo tanto, la función colectiva es el factor dominante en cada iteración del lazo más externo.

A diferencia de los modelos Φ_{A} y Φ_{B}, estos modelos presentan unos pesos de Akaike relativamente bajos que indican, a priori, una calidad pobre de los resultados. En cualquier caso, la importancia relativa de los diferentes términos considerados, proporcionada por el entorno TIA, permite que el usuario analice la calidad de estas expresiones y las modifique consecuentemente. La tabla 5.12 muestra la importancia relativa de los diferentes términos considerados en la obtención de los modelos del tiempo de ejecución en los casos A y B. Únicamente se muestran los términos con una importancia relativa mayor que 0,5. En el caso A, seis de los siete términos que aparecen en el modelo comparten el valor máximo de importancia relativa, lo que evidencia que el modelo T_{A} es realmente una aproximación adecuada del comportamiento real de la aplicación. En el caso B, aunque el modelo es muy preciso para caracterizar los datos experimentales, el peso de Akaike es muy pequeño en este caso



(a) Caso 1



(b) Caso 2

Figura 5.25: Valores experimentales (puntos) y modelo del tiempo de ejecución (líneas)

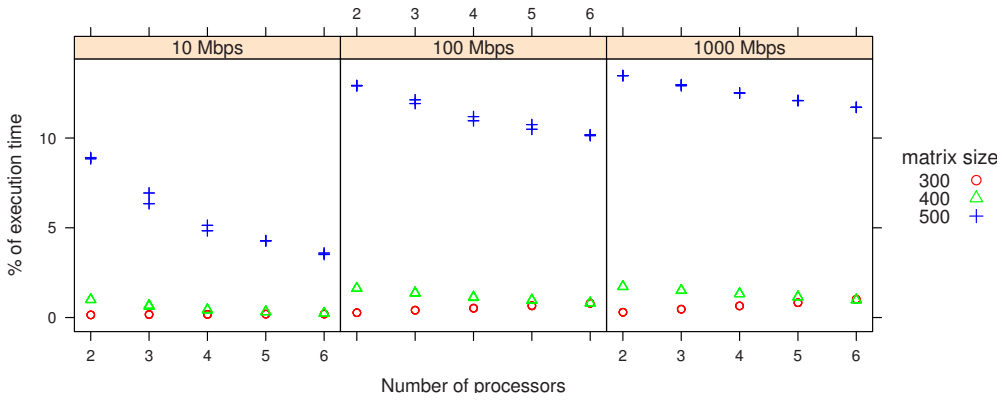
Tabla 5.12: Importancia relativa de los términos en los modelos del tiempo de ejecución (sólo se muestran los valores superiores a 0,5)

Caso A		Caso B	
ω_+^M	término	ω_+^M	término
0.9187411	$\frac{N^2}{\beta}$	0.9002893	$\frac{N}{\beta}$
0.9187411	$\frac{N^3}{P}$	0.9002893	$\frac{N^2}{P\beta}$
0.9187411	ϕ	0.8800227	ϕ
0.9187411	N^2	0.8776102	$\frac{N^3}{P}$
0.9187411	$\alpha\lceil\log_2 P\rceil$	0.8666854	$\alpha\lceil\log_2 P\rceil$
0.9187411	αP	0.8246401	$\frac{N}{P}\alpha\lceil\log_2 P\rceil$
0.5615440	$\lceil\log_2 P\rceil$	0.7727747	$\lceil\log_2 P\rceil$
0.5003511	P	0.5424446	$\frac{N}{P}\lceil\log_2 P\rceil$

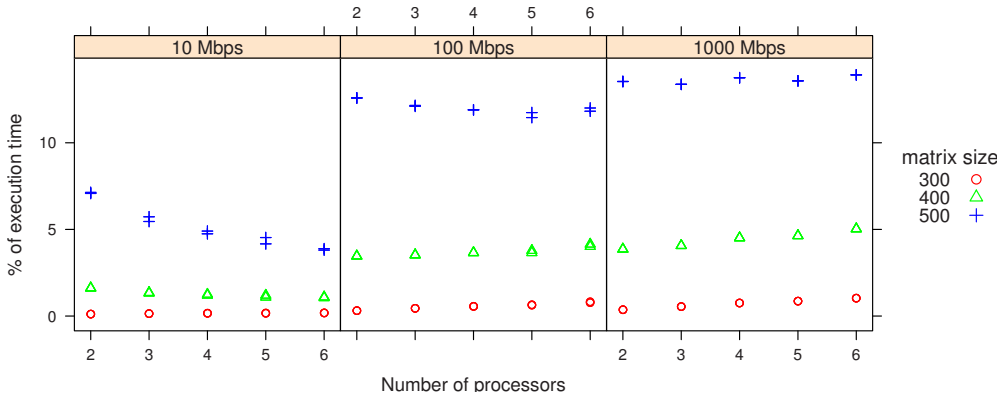
—se podría interpretar que el modelo T_B tiene únicamente un 2,46 % de posibilidades de ser el mejor modelo del conjunto de modelos candidatos considerado—. Es posible que, en este caso, los modelos considerados en el conjunto de candidatos no reflejen adecuadamente el comportamiento del caso B, aunque la mejor aproximación sea el modelo T_B. Por otro lado, los términos asociados al algoritmo lineal de la comunicación colectiva (P y αP) no están entre los términos con mayor importancia relativa. Este hecho indica que el algoritmo binomial es una mejor aproximación para caracterizar el mecanismo de comunicación.

Influencia de los fallos cache en el tiempo de ejecución

En los casos considerados, la influencia de los fallos cache L2 en el tiempo de ejecución se puede estimar combinando los modelos obtenidos para los fallos cache (tabla 5.10) y los modelos del tiempo de ejecución (tabla 5.11). Tanto en el caso A como en el caso B, la contribución de los fallos cache L2 en el tiempo de ejecución total de la aplicación se identifica en el sumando correspondiente al término ϕ. Por lo tanto, el coeficiente de estos términos proporciona el coste efectivo por cada fallo en la cache de nivel L2: 0,19 μs/fallo y 0,13 μs/fallo en los casos A y B, respectivamente. La figura 5.26 muestra la estimación del porcentaje de tiempo de ejecución asociado a los fallos cache L2, en ambos casos. Estos datos han sido obtenidos utilizando los modelos T y Φ para caracterizar, respectivamente, el tiempo de ejecución y los fallos cache L2. Por lo tanto, estas estimaciones han sido calculadas a partir de los parámetros de ejecución N, P, α y β. En el caso A, la influencia de los fallos cache L2 es



(a) Caso A



(b) Caso B

Figura 5.26: Estimación del porcentaje de tiempo de ejecución asociado a los fallos cache L2

prácticamente insignificante cuando la dimensión de la matriz es 300 o 400, pero alcanza un 10 % del tiempo de ejecución cuando la dimensión de la matriz es 500. En el caso B se puede observar un comportamiento similar, aunque la contribución es mayor cuando la dimensión de la matriz es 400. Aunque la influencia de los fallos cache disminuye con el número de procesos, el ritmo de decrecimiento es menor al aumentar el ancho de banda de la red. Además, en ambos casos, la importancia de los fallos cache es prácticamente constante cuando la velocidad de la red es 1000 Mbps. Ambos efectos están asociados al hecho de que el coste individual de cada comunicación disminuye al aumentar el ancho de banda de la red y, por lo tanto, la importancia de las comunicaciones en el coste total de la aplicación se reduce consecuentemente.

5.4 Contribuciones

El método de selección de modelos basado en AIC, descrito en el capítulo 4, ha sido utilizado para obtener modelos de rendimiento en diversos casos de estudio. En particular, se han obtenido modelos analíticos de diversos códigos del *benchmark* paralelo NAS y del *benchmark* HPL. En este último caso, se ha realizado una simulación de planificación de trabajos HPL en un cluster, utilizando el modelo obtenido para estimar el tiempo de ejecución, necesario en las políticas de *backfilling*. Estos resultado han sido comparados con simulaciones análogas utilizando una estimación del tiempo de ejecución basada en un modelo teórico del *benchmark*. El método de selección de modelos también ha sido utilizado para estimar la influencia de los fallos cache en dos versiones del producto paralelo matriz-vector. Las contribuciones de este capítulo han derivado en las siguientes publicaciones:

- *Improving the scheduling of parallel applications using accurate AIC-based performance models*, International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2010 [113].
- *Estimating the Effect of Cache Misses on the Performance of Parallel Applications Using Analytical Models*, 9th ACS/IEEE International Conference on Computing Systems and Applications, AICCSA 2011 [120].
- *Using accurate AIC-based performance models to improve the scheduling of parallel applications*, *Journal of Supercomputing* 2011 [114].

Conclusiones y principales aportaciones

En los últimos años, el desarrollo de nuevas arquitecturas paralelas ha permitido mantener el incremento sostenido del poder computacional de los sistemas HPC. Sin embargo, la complejidad intrínseca de las nuevas arquitecturas paralelas dificulta el desarrollo de aplicaciones que aprovechen eficientemente los recursos computacionales. Este nuevo contexto que se presenta en los entornos HPC ha constatado la necesidad de nuevas herramientas y metodologías de análisis, que permitan comprender y predecir el comportamiento de las aplicaciones en los actuales sistemas paralelos. El principal resultado de esta tesis es el entorno TIA (*Tools for Instrumentation and Analysis*), una nueva herramienta de análisis del rendimiento en sistemas paralelos, que define una metodología basada en el análisis estadístico de los datos generados mediante una instrumentación ligera del código fuente.

El diseño de TIA es modular y flexible, lo que permite su rápida adaptación a nuevas funcionalidades o requerimientos. Por un lado, el proceso de instrumentación y el proceso de análisis están desacoplados, y su conexión se realiza mediante ficheros XML, lo que proporciona el nivel de independencia suficiente para sustituir o modificar cualquiera de las partes sin que el proceso global se vea alterado. Por otro lado, las herramientas que se han utilizado para implementar los procesos de instrumentación y análisis son también modulares y escalables.

La instrumentación del código fuente ha sido desarrollada en torno a la herramienta `call`, un traductor de código que, mediante directivas de compilación, permite introducir funciones de instrumentación con una interferencia mínima en el comportamiento de la aplicación. Además, `call` está construida de manera modular y permite la incorporación de nuevas funcionalidades mediante el desarrollo de nuevos *drivers*. En particular, en este trabajo se han

desarrollado *drivers* para monitorizar métricas y parámetros de rendimiento relacionados con el estado de la red de interconexión y de los nodos computacionales.

El proceso de análisis ha sido desarrollado como una librería en R, un conocido lenguaje y entorno estadístico. En concreto, se han diseñado diversas funciones enfocadas a la obtención, de un modo sencillo y amigable, de modelos analíticos de rendimiento a partir de los datos generados durante la ejecución del código instrumentado, en diferentes situaciones experimentales. Las funciones de esta librería permiten la correcta gestión de los datos obtenidos mediante la instrumentación del código, y la construcción automática de un modelo analítico en función de las variables y los parámetros de ejecución. Además, la gran variedad de librerías disponibles en R proporciona un entorno adecuado para desarrollar cualquier otro tipo de análisis estadístico de los datos obtenidos de la ejecución del código instrumentado.

El proceso de modelado del entorno TIA ha sido desarrollado utilizando un mecanismo de selección de modelos basado en el criterio de información de Akaike. Este criterio permite asociar a cada modelo un valor numérico (AIC) que refleja objetivamente, en base al principio de parsimonia, la idoneidad del modelo para representar un conjunto de muestras experimentales. En particular, el mecanismo de selección implementado en TIA utiliza este valor para clasificar un conjunto finito de modelos candidatos y seleccionar el que presenta el mejor valor AIC. Para definir el conjunto de modelos candidatos —necesario para iniciar el proceso de modelado— se ha diseñado un mecanismo de descripción de modelos que construye este conjunto a partir de las diferentes métricas y parámetros monitorizados durante la ejecución del código instrumentado. Utilizando estos datos, se calcula el valor AIC de cada uno de los modelos de este conjunto y se selecciona el modelo con mejores características para representar a los datos experimentales. Además, este mecanismo permite realizar una valoración global, a través de los pesos de Akaike y de la importancia relativa de cada parámetro considerado, de la calidad del modelo seleccionado respecto del resto de modelos del conjunto de candidatos.

La metodología que define el entorno TIA proporciona un mecanismo de modelado automático que permite enfocar los esfuerzos del usuario en el diseño experimental y en el análisis de los resultados. En particular, esta metodología requiere que el usuario elija adecuadamente el conjunto de métricas y parámetros de rendimiento que deben ser considerados en la instrumentación y en el proceso de modelado. Tanto la instrumentación adecuada del código como la obtención del modelo de rendimiento son procesos automáticos, y el resultado final es un modelo de rendimiento en función de las métricas y de los parámetros considerados inicialmente. Además, la metodología propuesta encaja dentro de una aproximación iterativa que

permite refinar el resultado obtenido en función de la información estadística generada por el propio proceso de modelado.

El uso del mecanismo de selección de modelos ha sido aplicado en diferentes casos de estudio. En concreto, los modelos de comportamiento de diferentes algoritmos de comunicación de tipo *broadcast* obtenidos mediante TIA han sido comparados con las expresiones obtenidas a partir de un análisis puramente teórico basado en el modelo LogGP. El mecanismo de selección de modelos también ha sido aplicado a diferentes códigos del *benchmark* NPB, obteniendo modelos analíticos precisos en función de los parámetros de ejecución, sin realizar una inspección manual del código fuente. El modelo de rendimiento del *benchmark* HPL también se ha obtenido mediante el entorno TIA. Utilizando el simulador GridSim, se ha estudiado el comportamiento de diferentes estrategias de *backfilling* para la planificación de trabajos HPL en un cluster, utilizando el modelo obtenido para realizar una estimación del tiempo de ejecución. Estos resultados han sido comparados con los tiempos de simulación obtenidos utilizando el modelo teórico, proporcionado por los propios desarrolladores del *benchmark*, para estimar el tiempo de ejecución de los trabajos HPL. La mayor precisión del modelo obtenido con el entorno TIA permite un aprovechamiento más eficiente de los recursos, así como un menor tiempo de espera de los trabajos. Por otro lado, el diseño del mecanismo de selección de modelos implementado en TIA permite realizar modelos analíticos de cualquier métrica monitorizable durante la ejecución de los códigos instrumentados como, por ejemplo, los fallos cache. En particular, utilizando los modelos generados con TIA, se ha diseñado una metodología que permite realizar una estimación de la influencia de los fallos cache en el tiempo de ejecución de una aplicación.

La metodología definida es suficientemente flexible para analizar, a partir de datos monitorizables durante la ejecución de las aplicaciones, otras características de rendimiento. En particular, en esta tesis se ha desarrollado un mecanismo de caracterización de la red de interconexión de un sistema paralelo utilizando el modelo LoOgGP. Este nuevo modelo es una generalización de los modelos LogP y LogGP para tamaños de mensaje grandes, basado en el comportamiento lineal, observado experimentalmente, de las magnitudes *gap* y *overhead* en diferentes clusters. Este mecanismo ha sido completamente integrado en el entorno TIA, incluyendo un *driver* apropiado de `call` y una función en R.

Futuras líneas de investigación

En esta tesis, el modelo seleccionado en base al criterio de Akaike ha sido obtenido mediante fuerza bruta, calculando el correspondiente peso de Akaike de cada modelo candidato. Sin embargo, este procedimiento no es escalable porque el número de potenciales modelos crece rápidamente con el número de métricas y parámetros de rendimiento considerados. Por un lado, sería necesario desarrollar un mecanismo de descripción de modelos más sofisticado que permita incorporar o eliminar, de un modo más flexible, interacciones entre las diferentes magnitudes de rendimiento consideradas. Por otro lado, el número de potenciales factores que afectan al rendimiento de aplicaciones paralelas es muy elevado, por lo que es necesario explorar mecanismos de búsqueda más eficientes, que garanticen la selección de un modelo analítico adecuado dentro de un conjunto con un gran número de modelos candidatos. En este sentido, las metaheurísticas —como, por ejemplo, *simulated annealing*, algoritmos genéticos o búsqueda tabú— proporcionan un marco adecuado para resolver este tipo de situaciones. Los parámetros de calidad del proceso de selección de modelos podrían ser utilizados como indicadores de calidad de los modelos evaluados durante el proceso de búsqueda.

La inferencia multimodelo puede ser una alternativa adecuada para obtener predicciones precisas del comportamiento de aplicaciones paralelas. En situaciones reales, la ejecución de una aplicación paralela depende de un número elevado de factores, por lo que su descripción mediante un único modelo, con un número limitado de variables explicativas, no proporciona el adecuado nivel de precisión. La inferencia multimodelo, por el contrario, utiliza varios modelos simultáneamente —cada uno de ellos con una determinada probabilidad— para describir el comportamiento de un sistema.

El entorno TIA permite obtener modelos basados en los *factores* de ejecución, es decir, en los parámetros de rendimiento modificables por el usuario. Por lo tanto, es posible extrapolar el análisis de rendimiento implementado en TIA para obtener modelos en cualquier sistema que disponga de factores de ejecución, y en el que existan métricas claramente definidas que sean accesibles a través de *drivers* de `call`. En particular, pueden obtenerse modelos de consumo energético de dispositivos integrados en chip (*system on chip*, SoC) si se dispone de un *driver* que permita el acceso a las características energéticas del sistema, como por ejemplo la corriente consumida o la temperatura del dispositivo. Las aplicaciones en entornos heterogéneos, como los sistemas híbridos CPU+GPU, también son susceptibles de ser modelados con TIA en función de los factores de la propia aplicación, utilizando métricas que caractericen adecuadamente la estructura física de este tipo de sistemas.

Bibliografía

[1] Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John Mellor-Crummey y Nathan R. Tallent: *HPCToolkit: Tools for performance analysis of optimized parallel programs*. Concurrency and Computation: Practice and Experience, 22(6):685–701, 2010.

[2] Vikram S. Adve, Rajive Bagrodia, James C. Browne, Ewa Deelman, Aditya Dube, Elias Houstis, John Rice, Rizos Sakellariou, David Sundaram-stukel, Patricia J. Teller y Mary K. Vernon: *POEMS: End-to-End Performance Design of Large Parallel Adaptive Computational Systems*. IEEE Transactions on Software Engineering, 26:1027–1048, 2001.

[3] A. Agarwal y M. Levy: *The kill rule for multicore*. En *Proceedings of the 44th Design Automation Conference*, 2007.

[4] H. Akaike: *A new look at the statistical model identification*. IEEE Transactions on Automatic Control, 19:716–723, 1974.

[5] S.R. Alam y J.S. Vetter: *A framework to develop symbolic performance models of parallel applications*. En *Proc. 20th International Parallel and Distributed Processing Symposium*, IPDPS, 2006.

[6] J. L. Albín, J. A. Lorenzo, J. C. Cabaleiro, T. F. Pena y F. F. Rivera: *Simulation of Parallel Applications in GridSim*. En *1st Iberian Grid Infrastructure Conference Proceedings*, 2007.

[7] John Aldrich: *R. A. Fisher and the making of maximum likelihood 1912–1922*. Statistical Science, 12(3):162–176, 1997.

[8] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser y Chris Scheiman: *LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation*. Journal of Parallel and Distributed Computing, 44:71–79, 1997.

[9] F. Almeida, J. A. Gomez y J. M. Badia: *Performance analysis for clusters of symmetric multiprocessors*. En *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, PDP'07, 2007.

[10] Wolfram Amme, Peter Braun, Welf Löwe y Eberhard Zehendner: *LogP modelling of list algorithms*. En *11th Symposium on Computer Architecture and High Performance Computing*, 1999.

[11] Sylvain Arlot y Alain Celisse: *A survey of cross-validation procedures for model selection*. Statistics Surveys, 4:40–79, 2010.

[12] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica y Matei Zaharia: *A view of cloud computing*. Communications of the ACM, 53(4):50–58, 2010.

[13] R. Badia, J. Labarta, J. Giménez y F. Escalé: *DIMEMAS: Predicting MPI applications behavior in Grid environments*. En *Workshop on Grid Applications and Programming Tools*, GGF8, 2003.

[14] Rosa M. Badia, Germán Rodríguez y Jesús Labarta: *Deriving analytical models from a limited number of runs*. En *Parallel Computing*, ParCo, 2003.

[15] D. Bailey, E. Barscz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan y S. Weeratunga: *The NAS parallel benchmarks, RNR-94-007*. Informe técnico, NASA Ames Research Center, 1994.

[16] D. H. Bailey, E. Barszcz, L. Dagum y H. D. Simon: *NAS parallel benchmark results*. IEEE Parallel Distributed Technology: Systems Applications, 1(1):43–51, 1993.

[17] Saisanthosh Balakrishnan, Ravi Rajwar, Michael Upton y Konrad K. Lai: *The Impact of Performance Asymmetry in Emerging Multicore Architectures.* En *32st International Symposium on Computer Architecture*, 2005.

[18] Barcelona Supercomputing Center - Centro Nacional de Supercomputación. http://www.bsc.es, 2011.

[19] Ingrid Barcena, José Antonio Becerra, Joan Cambras, Richard Duro, Carlos Fernández, Javier Fontán, Andrés Gómez, Ignacio López, Caterina Parals, José Carlos Pérez y Juan Villasuso: *A Grid Supercomputing Enviroment for High Demand Computational Applications*. Informe técnico, Centre de Supercomputació de Catalunya (CESCA), Autonomous Systems Group (GSA), University of A Coruña (UDC), Centro de Supercomputación de Galicia (CESGA), 2000. http://www.cesga.es/pdf/Grid_CESGA_CESCA.PDF.

[20] C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Hargrove, P. Husbands, C. Iancu, M. Welcome y K. Yelick: *An evaluation of current high-performance networks*. En *Proc. International Parallel and Distributed Processing Symposium*, 2003.

[21] Gordon Bell y Jim Gray: *What's next in high-performance computing?* Communications of the ACM, 45(2):91–95, 2002.

[22] V. Blanco, J. A. González, C. León, C. Rodríguez, G. Rodríguez y M. Printista: *Predicting the performance of parallel programs*. Parallel Computing, 30:337–356, 2004.

[23] Vicente Blanco: *Análisis, predicción y visualización del rendimiento de métodos iterativos en HPF y MPI*. Tesis de Doctorado, Universidad de Santiago de Compostela, 2002.

[24] Olaf Bonorden, Ben Juulink, Ingo von Otto y Ingo Rieping: *The Paderborn University BSP (PUB) library - design, implementation and performance*. En *13th International Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing*, 1999.

[25] J.L. Bosque y L.P. Perez: *HLogGP: a new parallel computational model for heterogeneous clusters*. En *Proc. IEEE International Symposium on Cluster Computing and the Grid*, CCGrid, 2004.

[26] G. E. P. Box y N. R. Draper: *Empirical model-building and response surfaces*. John Wiley & Sons, Inc., 1987.

[27] G. E. P. Box y G. M. Jenkins: *Time series analysis: Forecasting and control*. Holden-Day, San Francisco, 1970.

[28] S. Browne, J. Dongarra, N. Garner, G. Ho y P. Mucci: *A portable programming interface for performance evaluation on modern processors*. International Journal of High-Performance Computing Applications, 14(3):189–204, 2000.

[29] B. Buck y J. Hollingsworth: *An API for runtime code patching*. International Journal of High Performance Computing Applications, 14(4):317–329, 2000.

[30] Kenneth P. Burnham y David R. Anderson: *Model Selection and Multimodel Inference. A Practical Information-Theoretic Approach*. Spring Science + Bussiness Media, LLC, 2002.

[31] K.W. Cameron, R. Ge y X. H. Sun: *log_nP and log_3P: Accurate Analytical Models of Point-to-Point Communication in Distributed Systems*. IEEE Transactions on Computers, 56(3):314–327, 2007.

[32] M. Casas, R. M. Badia y J. Labarta: *Prediction of behavior of MPI applications*. En *Proc. IEEE International Conference on Cluster Computing*, 2008.

[33] P. Caymes-Scutari, A. Morajko, T. Margalef y E. Luque: *Scalable dynamic Monitoring, Analysis and Tuning Environment for parallel applications*. Journal of Parallel and Distributed Computing, 70(4):330–337, 2010.

[34] Anthony Chan, William Gropp y Ewing Lusk: *An Efficient Format for Nearly Constant-Time Access to Arbitrary Time Intervals in Large Trace Files*. Scientific Programming, 16:155–165, 2008.

[35] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer y Kevin Skadron: *A performance study of general-purpose applications on graphics processors using CUDA*. Journal of Parallel and Distributed Computing, 68(10):1370–1380, 2008.

[36] Chau Yi Chou, Hsi Ya Chang, Shuen Tai Wang y Chang Hsing Wu: *A Semi-Empirical Model for Maximal LINPACK Performance Predictions*. En *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, 2006.

[37] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian y T. von Eicken: *LogP: Towards a Realistic Model of Parallel Computation*. En *Principles Practice of Parallel Programming*, 1993.

[38] D. E. Culler, Lok Tin Liu, R. P. Martin y C. O. Yoshikawa: *Assessing fast network interfaces*. IEEE MICRO, 16(1):35–43, Feb. 1996.

[39] D. E. Culler, J. P. Singh y A. Gupta: *Parallel Computer Architecture: A Hardware/Software Approach*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 1998.

[40] H.J. Curnow y B.A. Wichman: *A Synthetic Benchmark*. Computer Journal, 19(1):43–49, 1976.

[41] A. Snavely D. H. Bailey: *Performance Modeling: Understanding the Present and Predicting the Future*. En *EuroPar*, 2005.

[42] Wolfgang E. Denzel, Jian Li, Peter Walker y Yuho Jin: *A framework for end-to-end simulation of high-performance computing systems*. En *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008.

[43] S. Dong, G.E. Karniadakes y N.T. Karonis: *Cross-site computations on the TeraGrid*. Computing in Science Engineering, 7(5):14–23, 2005.

[44] Jack Dongarra: *The Impact of Multicore on Math Software and Exploiting Single Precision Computing to Obtain Double Precision Results.* En *4th International Symposium Parallel and Distributed Processing and Applications*, 2006.

[45] Jack Dongarra, Piotr Luszczek y Antoine Petitet: *The LINPACK Benchmark: Past, Present and Future*. Concurrency and Computation: Practice and Experience, 15(9):803–820, 2003.

[46] Jack Dongarra, Allen D. Malony, Shirley Moore, Philip Mucci y Sameer Shende: *Performance instrumentation and measurement for terascale systems*. En *Proceedings of the 2003 international conference on Computational science*, ICCS, 2003.

[47] Jack Dongarra, Thomas Sterling, Horst Simon y Erich Strohmaier: *High-Performance Computing: Clusters, Constellations, MPPs, and Future Directions*. Computing in Science and Engineering, 7:51–59, 2005.

[48] Jack Dongarra et ál.: *The International Exascale Software Project roadmap*. International Journal of High Performance Computing Applications, 25:3–60, 2011.

[49] Andrea C. Dusseau, David E. Culler, Klaus Erik Schauser y Richard P. Martin: *Fast Parallel Sorting Under LogP: Experience with the CM-5*. IEEE Transactions on Parallel and Distributed Systems, 7:791–805, 1996.

[50] Joern Eisenbiegler, Welf Loewe y Andreas Wehrenpfennig: *On the Optimization by Redundancy Using an Extended LogP Model*. En *APDC '97: Proceedings of the 1997 Advances in Parallel and Distributed Computing Conference*, 1997.

[51] Rolf Ernst: *Networks, multicore, and systems evolution—facing the timing beast*. En *Conference on Emerging Technologies and Factory Automation*, 2008.

[52] C. Evangelinos y C. N. Hill: *Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2*. En *Proc. Cloud Computing and Its Applications*, 2008.

[53] G. Fagg, J. Pjěsivac-Grbovíc, G. Bosilca, T. Angskun y J. Dongarra: *Flexible Collective Communication Tuning Architecture applied to Open MPI*. En *2006 Euro PVM/MPI*, 2006.

[54] Ahmad Faraj y Xin Yuan: *Communication Characteristics in the NAS Parallel Benchmarks*. En *Fourteenth IASTED International Conference on Parallel and Distributed Computing and Systems*, PDCS, 2002.

[55] R. A. Fisher: *On the mathematical foundations of theoretical statistics*. Philosophical Transactions of the Royal Society of London, 222:309–368, 1922.

[56] M. Flynn: *Some Computer Organizations and Their Effectiveness*. IEEE Transactions on Computers, C-21:948+, 1972.

[57] Steven Fortune y James Wyllie: *Parallelism in random access machines*. En *Tenth Annual ACM Symp. Theory of Computing*, 1978.

[58] I. Foster, C. Kesselman y S. Tuecke: *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. The International Journal of High Performance Computing Applications, 15(3):200–222, 2001.

[59] Ian Foster y Carl Kesselman: *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 2003.

[60] Matthew I. Frank, Anant Agarwal y Mary K. Vernon: *LoPC: Modeling Contention in Parallel Algorithms*. En *Proc. of the SIXTH ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1997.

[61] Felix Freitag, Jordi Caubet y Jesus Labarta: *On the Scalability of Tracing Mechanisms*. En *8th International Euro-Par Conference*, 2002.

[62] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham y Timothy S. Woodall: *Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation*. En *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.

[63] David Geer: *Industry Trends: Chip Makers Turn to Multicore Processors.* IEEE Computer, 38(5):11–13, 2005.

[64] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker y Bernd Mohr: *The* SCALASCA *performance toolset architecture*. Concurrency and Computation: Practice and Experience, 22(6):702–719, 2010.

[65] Jonathan Geisler, Valerie Taylor, Xingfu Wu y Rick Stevens: *Using Kernel Coupling to Improve the Performance of Multithreaded Applications*. En *16th International Conference on Parallel and Distributed Computing Systems*, PDCS, 2003.

[66] J. A. González, C. León, J. L. Roda, C. Rodríguez, J. M. Rodríguez, F. Sande y M. Printista: *Model Oriented Profiling of Parallel Programs*. En *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, EUROMICRO-PDP, 2002.

[67] J. A. González, Casiano Rodríguez, José L. Roda, Daniel González-Morales, F. Sande, Francisco Almeida y C. Léon: *Performance and Predictability of MPI and BSP Programs on the CRAY T3E*. En *Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 1999.

[68] Eric Grobelny, David Bueno, Ian Troxel, Alan D. George y Jeffrey S. Vetter: *FASE: A Framework for Scalable Performance Prediction of HPC Systems and Applications*. Simulation, 83(10):721–745, 2007.

[69] S.D. Hammond, G.R. Mudalige, J.A. Smith, S.A. Jarvis, J.A. Herdman y A. Vadgama: *WARPP: A Toolkit for Simulating High Performance Parallel Scientific Codes*. En *SIMUTools09*, 2009.

[70] S.D. Hammond, G.R. Mudalige, J.A. Smith, A.B. Mills, S.A. Jarvis, J. Holt, I. Miller, J.A. Herdman y A. Vadgama: *Performance Prediction and Procurement in Practise: Assessing the Suitability of Commodity Cluster Components for Wavefront Codes*. IET Software, 3(6):509–521, 2009.

[71] S.D. Hammond, J.A. Smith, G.R. Mudalige y S.A. Jarvis: *Predictive Simulation of HPC Applications*. En *The IEEE 23rd International Conference on Advanced Information Networking and Applications*, AINA, 2009.

[72] P. Heidelberger y S.S. Lavenberg: *Computer Performance Evaluation Methodology*. IEEE Transactions on Computers, 33:1195–1220, 1984.

[73] John L. Hennessy y David A. Patterson: *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2006.

[74] R. Hockney: *The communication challenge for MPP: Intel Paragon and Meiko CS-2*. Parallel Computing, 20:389–398, 1994.

[75] T. Hoefler, L. Cerquetti, T. Mehlan, F. Mietke y W. Rehm: *A practical approach to the rating of barrier algorithms using the LogP model and Open-MPI*. En *Proceedings of the 2005 International Conference on Parallel Processing Workshops*, 2005.

[76] T. Hoefler, A. Lichei y W. Rehm: *Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks*. En *21st IEEE International Parallel & Distributed Processing Symposium*, IPDPS, 2007.

[77] T. Hoefler, T. Mehlan, A. Lumsdaine y W. Rehm: *Netgauge: A Network Performance Measurement Framework*. En *High Performance Computing and Communications, Third International Conference*, HPCC, 2007.

[78] T. Hoefler, T. Mehlan, F. Mietke y W. Rehm: *LogfP - a model for small messages in InfiniBand*. En *20th IEEE International Parallel and Distributed Processing Symposium*, IPDPS, 2006.

[79] T. Hoefler, T. Schneider y A. Lumsdaine: *LogGP in theory and practice—An in-depth analysis of modern interconnection networks and benchmarking methods for collective operations*. Simulation Modelling Practice and Theory, 17(9):1511– –1521, 2009. Advances in System Performance Modelling, Analysis and Enhancement.

[80] Torsten Hoefler, Timo Schneider y Andrew Lumsdaine: *LogGOPSim: simulating large-scale applications in the LogGOPS model*. En *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC, 2010.

[81] HPCToolkit. http://www.hpctoolkit.org, 2011.

[82] HPL—A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. http://www.netlib.org/benchmark/hpl/, 2010.

[83] K. A. Huck, A. D. Malony, R. Bell y A. Morris: *Design and Implementation of a Parallel Performance Data Management Framework*. En *Proceedings of the International Conference on Parallel Processing*, 2005.

[84] K. A. Huck, A. D. Malony, S. Shende y A. Morris: *Knowledge support and automation for performance analysiswith PerfExplorer 2.0*. Scientific Programming special issue on Large-Scale Programming Tools and Environments, 16(2-3):123–134, 2008.

[85] Giulio Iannello: *Efficient Algorithms for the Reduce-Scatter Operation in LogGP*. IEEE Transactions on Parallel and Distributed Systems, 8(9):970–982, 1997.

[86] Infiniband Trade Association. http://www.infinibandta.org, 2010.

[87] Fumihiko Ino, Noriyuki Fujimoto y Kenichi Hagihara: *LogGPS: a parallel computational model for synchronization analysis*. En *PPoPP '01: Proceedings of the*

8th ACM SIGPLAN symposium on Principles and Practices of Parallel Programming, 2001.

[88] Intel® MPI Benchmarks. User Guide and Methodology Description. http://www.intel.com/software/imb, 2011.

[89] Intel® Trace Analyzer and Collector. http://software.intel.com/en-us/articles/intel-trace-analyzer, 2011.

[90] A. K. Jain, M. N. Murty y P. J. Flynn: *Data Clustering: A Review*. ACM Computing Surveys, 31(3):264–323, 1999.

[91] R. K. Jain: *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, Inc., 1991.

[92] Seung hye Jang, Valerie Taylor, Xingfu Wu y Mieke Prajugo: *Performance Prediction-based versus Load-based Site Selection: Quantifying the Difference*. En *Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems*, 2005.

[93] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira y J. Riodan: *Modeling of workload in MPPs*. En *IPPS'97: Proceedings of the Job Scheduling Strategies for Parallel Procesing*, 1997.

[94] Stephen A. Jarvis, Daniel P. Spooner, Helene N. Lim Choi Keung, Junwei Cao, Subhash Saini y Graham R. Nudd: *Performance prediction and its use in parallel and distributed computing systems*. Future Generation Computer Systems, 22(7):745–754, 2006.

[95] Tomasz Kalinowski, Iskander Kort y Denis Trystram: *List scheduling of general task graphs under LogP*. Parallel Computing, 26926:1109–1128, 2000.

[96] Kimberly K. Keeton, Thomas E. Anderson y David A. Patterson: *LogP Quantified: The Case for Low-Overhead Local Area Networks*. En *Hot Interconnects III: A Symposium on High Performance Interconnects*, 1995.

[97] Ken Kennedy: *Software Challenges for Multicore Computing.* En Yves Robert, Manish Parashar, Ramamurthy Badrinath y Viktor K. Prasanna (editores):

13th International Conference High Performance Computing, volumen 4297 de *Lecture Notes in Computer Science*, página 4. Springer, 2006.

[98] D. J. Kerbyson, Al H. J., A. Hoisie, F. Petrini, H. J. Wasserman y M. Gittings: *Predictive performance and scalability modeling of a large-scale application*. En *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, 2001.

[99] Darren J. Kerbyson, Adolfy Hoisie y Harvey J. Wasserman: *High Performance Scientific and Engineering Computing, Hardware/Software Support*, capítulo Verifying Large-Scale System Performance During Installation using Modeling, páginas 143–156. Kluwer Academic Publishers, 2003.

[100] Thilo Kielmann, Henri E. Bal y Sergei Gorlatch: *Bandwidth-efficient Collective Communication for Clustered Wide Area Systems*. En *IEEE International Parallel and Distributed Processing Symposium*, IPDPS, 2000.

[101] Thilo Kielmann, Henri E. Bal y Kees Verstoep: *Fast Measurement of LogP Parameters for Message Passing Platforms*. En *IEEE International Parallel and Distributed Processing Symposium*, IPDPS, 2000.

[102] A. Knüpfer, R. Brendel, H. Brunst, H. Mix y W. E. Nagel: *Introducing the Open Trace Format (OTF)*. En *Proceedings of the 6th International Conference on Computational Science*, ICCS, 2006.

[103] Jochen Krallmann, Uwe Schwiegelshohn y Ramin Yahyapour: *On the Design and Evaluation of Job Scheduling Algorithms*. En *IPPS/SPDP '99/JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing*, 1999.

[104] S. Kullback y R. A. Leibler: *On information and sufficiency*. Annals of Mathematical Statistics, 22:79–86, 1951.

[105] James F. Kurose y Keith W. Ross: *Computer Networking: A Top-Down Approach*. Addison Wesley, 2009.

[106] Jesús Labarta, Sergi Girona, Vincent Pillet, Toni Cortés y Luis Gregoris: *DiP: A Parallel Program Development Environment*. En *2nd International EuroPar Conference*, EuroPar, 1996.

[107] Kathleen A. Lindlan, Janice Cuny, Allen D. Malony, Sameer Shende, Reid Rivenburgh, Craig Rasmussen y Bernd Mohr: *A tool framework for static and dynamic analysis of object-oriented software with templates*. En *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, 2000.

[108] José-Juan López-Espín y Domingo Giménez: *Obtaining Simultaneous Equation Models from a set of variables through Genetic Algorithms*. En *10^{th} International Conference on Computational Science*, 2010.

[109] David Luebke, Mark Harris, Jens Krüger, Tim Purcell, Naga Govindaraju, Ian Buck, Cliff Woolley y Aaron Lefohn: *GPGPU: general purpose computation on graphics hardware*. En *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, 2004.

[110] Allen D. Malony, Sameer Shende, Alan Morris, Scott Biersdorff, Wyatt Spear, Kevin Huck y Aroon Nataraj: *Evolution of a Parallel Performance System*. En *2^{nd} International Workshop on Parallel Tools for High Performance Computing*, 2008.

[111] Gabriel Marin y John Mellor-Crummey: *Cross-architecture performance predictions for scientific applications using parameterized models*. ACM SIGMETRICS Performance Evaluation Review, 32(1):2–13, 2004.

[112] Diego R. Martínez, Julio L. Albín, José Carlos Cabaleiro, Tomás F. Pena, Francisco F. Rivera y Vicente Blanco: *El Criterio de Información de Akaike en la Obtención de Modelos Estadísticos de Rendimiento*. En *XX Jornadas de Paralelismo*, 2009.

[113] Diego R. Martínez, Julio L. Albín, Tomás F. Pena, José Carlos Cabaleiro, Francisco F. Rivera y V. Blanco: *Improving the scheduling of parallel applications using accurate AIC-based performance models*. En *2010 International Conference on Computational and Mathematical Methods in Science and Engineering*, CMMSE, 2010.

[114] Diego R. Martínez, Julio L. Albín, Tomás F. Pena, José Carlos Cabaleiro, Francisco F. Rivera y Vicente Blanco: *Using accurate AIC-based performance models to improve the scheduling of parallel applications*. Journal of Supercomputing, -:–, 2011. (disponible online).

[115] Diego R. Martínez, Vicente Blanco, Marcos Boullón, José Carlos Cabaleiro y Tomás F. Pena: *Analytical Performance Models of Parallel Programs in Clusters*. En *Parallel Computing: Architectures, Algorithms and Applications*, ParCo, 2007.

[116] Diego R. Martínez, Vicente Blanco, Marcos Boullón, José Carlos Cabaleiro, Casiano Rodríguez y Francisco F. Rivera: *Software Tools for Performance Modeling of Parallel Programs*. En *IEEE International Parallel and Distributed Processing Symposium*, IPDPS, 2007.

[117] Diego R. Martínez, Vicente Blanco, José Carlos Cabaleiro, Tomás F. Pena y Francisco F. Rivera: *Automatic Parameter Assessment of LogP-based Communication Models in MPI Environments*. En *International Conference on Computational Science*, ICCS, 2010.

[118] Diego R. Martínez, Vicente Blanco, José Carlos Cabaleiro, Tomás F. Pena y Francisco F. Rivera: *Modelización del Rendimiento de Aplicaciones MPI en Entornos Grid*. En *XVII Jornadas de Paralelismo*, 2006.

[119] Diego R. Martínez, José Carlos Cabaleiro, Tomás F. Pena, Francisco F. Rivera y V. Blanco: *Accurate Analytical Performance Model of Communications in MPI Applications*. En *8th International Workshop on Performance Modeling, Evaluation, and Optimization of Ubiquitous Computing and Networked Systems*, PMEO UCNS'09, 2009.

[120] Diego R. Martínez, José Carlos Cabaleiro, Tomás F. Pena, Francisco F. Rivera y V. Blanco: *Estimating the Effect of Cache Misses on the Performance of Parallel Applications Using Analytical Models*. En *9th ACS/IEEE International Conference on Computing Systems and Applications*, AICCSA, 2011. (Enviado).

[121] Diego R. Martínez, Tomás F. Pena, José Carlos Cabaleiro, Francisco F. Rivera y V. Blanco: *Performance Modeling of MPI Applications Using Model Selection*. En *18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010.

[122] Diego R. Martínez, Tomás F. Pena, José Carlos Cabaleiro, Francisco F. Rivera y Vicente Blanco: *Obtención de modelos estadísticos de rendimiento de las comunicaciones en aplicaciones MPI*. En *XIX Jornadas de Paralelismo*, 2008.

[123] Matthew L. Massie, Brent N. Chun y David E. Culler: *The Ganglia distributed monitoring system: desing, implementation, and experience*. Parallel Computing, 30:817–840, 2004.

[124] Message Passing Interface Forum. http://www.mpi-forum.org, 2010.

[125] Bernd Mohr, Allen D. Malony, Sameer Shende y Felix Wolf: *Towards a performance tool interface for OpenMP: an approach based on directive rewriting*. En *Proceedings of the Third European Workshop on OpenMP*, 2001.

[126] Bernd Mohr y Felix Wolf: *KOJAK - A tool set for automatic performance analysis of parallel applications*. En *Proceedings of the European Conference on Parallel Computing*, EuroPar, 2003.

[127] C. A. Moritz y M. I. Frank: *LoGPC: Modeling Network Contention in Message-Passing Programs*. IEEE Transactions on Parallel and Distributed Systems, 12(4):404–415, 2001.

[128] MPI Forum: *MPI: A Message-Passing Interface Standard*. The International Journal of Supercomputer Applications and High Performance Computing, 8:159–416, 1994.

[129] W. E. Nagel, A. Arnold, M. Weber, H. Ch. Hoppe y K. Solchenbach: *VAMPIR: Visualization and Analysis of MPI Resources*. Supercomputer, 12:69–80, 1996.

[130] Aroon Nataraj, Allen D. Malony, Alan Morris, Dorian C. Arnold y Barton P. Miller: *A framework for scalable, parallel performance monitoring*. Concurrency and Computation: Practice and Experience, 22(6):720–735, 2010.

[131] G. R. Nudd y S. A. Jarvis: *Performance-based middleware for Grid computing*. Concurrency and Computation: Practice and Experience, 17:215–235, 2005.

[132] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper y D. V. Wilcox: *PACE–A Toolset for the Performance Prediction of Parallel and Distributed Systems*. The International Journal of High Performance Computing Applications, 14(3):228–251, 2000.

[133] Natawut Nupairoj y Lionel M. Ni: *Performance Metrics and Measurement Techniques of Collective Communication Services*. En *CANPC '97: Proceedings of the First International Workshop on Communication and Architectural Support for Network-Based Parallel Computing*, 1997.

[134] OpenMP API. http://www.openmp.org, 2010.

[135] Performance Application Programming Interface. http://icl.cs.utk.edu/papi, 2011.

[136] Larry L. Peterson y Bruce S. Davie: *Computer Networks: A Systems Approach*. Morgan Kaufmann, 2011.

[137] J. C. Pichel, D. B. Heras, J. C. Cabaleiro y F. F. Rivera: *Increasing data reuse of sparse algebra codes on simultaneous multithreading architectures*. Concurrency and Computation: Practice and Experience, 21(15):1838–1856, 2009.

[138] J. Pjěsivac-Grbovíc: *Towards Automatic and Adaptive Optimizations of MPI Collective Operations*. Tesis de Doctorado, University of Tennessee, 2007.

[139] J. Pjěsivac-Grbovíc, T. Angskun, G. Bosilca, G.E. Fagg, E. Gabriel y J.J. Dongarra: *Performance analysis of MPI collective operations*. En *Proc. 19th IEEE International Parallel and Distributed Processing Symposium*, 2005.

[140] R Development Core Team: *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. `http://www.R-project.org`.

[141] John J. Rehr, Fernando D. Vila, Jeffrey P. Gardner, Lucas Svec y Micah Prange: *Scientific Computing in the Cloud*. Computing in Science and Engineering, 12:34–43, 2010.

[142] Ruymán Reyes, Antonio Dorta, Francisco Almeida y Francisco de Sande: *Automatic Hybrid MPI+OpenMP code generation with llc*. En *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Lecture Notes in Computer Science, 2009.

[143] German Rodríguez, Rosa M. Badia y Jesús Labarta: *Generation of simple analytical models for message passing applications*. En *Proceedings of the Euro-Par Conference*, 2004.

[144] Philip C. Roth y Barton P. Miller: *On-line automated performance diagnosis on thousands of processes*. En *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP, 2006.

[145] S. Saini, J. Chang, R. Hood y H. Jin: *A Scalability Study of Columbia using the NAS Parallel Benchmarks, NAS-06-011*. Informe técnico, NASA Ames Research Center, 2006.

[146] Lauren Sarno, Wen mei W. Hwu, Craig Lund, Markus Levy, James R. Larus, James Reinders, Gordon Cameron, Chris Lennard y Takashi Yoshimori: *Corezilla: Build and Tame the Multicore Beast?* En *Design Automation Conference*, 2007.

[147] Scalasca. http://www.scalasca.org, 2011.

[148] G. Schwarz: *Estimating the dimension of a model*. Annals of Statistics, 6:461–464, 1978.

[149] S. Shende y A. D. Malony: *The TAU Parallel Performance System*. International Journal of High Performance Computing Applications, SAGE Publications, 20(2):287–331, 2006.

[150] Sameer Shende, Allen D. Malony y Alan Morris: *Optimization of instrumentation in parallel performance evaluation tools*. En *Proceedings of the 8^{th} international conference on Applied parallel computing: state of the art in scientific computing*, 2007.

[151] George Stantchev, Derek Juba, William Dorland y Amitabh Varshney: *Using Graphics Processors for High-Performance Computation and Visualization of Plasma Turbulence*. Computing in Science and Engineering, 11:52–59, 2009.

[152] Anthony Sulistio, Uros Cibej, Srikumar Venugopal, Borut Robic y Rajkumar Buyya: *A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim*. Concurrency and Computation: Practice and Experience (CCPE), 20(13):1591–1609, 2008.

[153] David Sundaram-Stukel y Mary K. Vernon: *Predictive Analysis of a Wavefront Application Using LogGP*. En *Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1999.

[154] Nathan R. Tallent, John M. Mellor-Crummey, Laksono Adhianto, Michael W. Fagan y Mark Krentel: *Diagnosing performance bottlenecks in emerging petascale applications*. En *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009.

[155] Tuan Zea Tan, R.S.M. Goh, V. March y S. See: *Data mining analysis to validate performance tuning practices for HPL.* En *IEEE International Conference on Cluster Computing and Workshops*, CLUSTER '09, 2009.

[156] TAU - Tuning and Analysis Utilities. http://www.cs.uoregon.edu/research/tau, 2011.

[157] Valerie Taylor, Xingfu Wu y Rick Stevens: *Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications*. ACM SIGMETRICS Performance Evaluation Review, 30(4):13–18, 2003.

[158] The 36th TOP500 List. http://www.top500.org/lists/2010/11, Noviembre 2010.

[159] The Prophesy System Website. http://prophesy.cs.tamu.edu, 2011.

[160] Top 500 Supercomputer Sites. http://www.top500.org, 2010.

[161] Dan Tsafrir, Yoav Etsion y Dror G. Feitelson: *Backfilling Using System-Generated Predictions Rather than User Runtime Estimates*. IEEE Transactions on Parallel and Distributed Systems, 18:789–803, 2007.

[162] L. G. Valiant: *A bridging model for parallel computation*. Communications of the ACM, 33(8):103–111, 1990.

[163] Vampir. http://www.vampir.eu, 2011.

[164] Vampir: product history. http://www.vampir.eu/Historie.html, 2011.

[165] R. F. Van Der Wijngaart: *NAS parallel benchmarks, version 2.4, NAS-02-007*. Informe técnico, NASA Ames Research Center, Moffett Field (USA), 2002.

[166] Sam Verboven, Peter Hellinckx, Frans Arickx y Jan Broeckhove: *Runtime Prediction Based Grid Scheduling of Parameter Sweep Jobs*. En *Proceedings of the IEEE Asia-Pacific Services Computing Conference*, 2008.

[167] WARwick Performance Prediction Toolkit (WARPP). http://www2.warwick.ac.uk/fac/sci/dcs/people/research/csrcbc/research/wppt, 2011.

[168] Felix Wolf y Bernd Mohr: *Automatic performance analysis of hybrid MPI/OpenMP applications*. Journal of Systems Architecture, 49(10-11):421–439, 2003.

[169] Rich Wolski: *Experiences with Predicting Resource Performance On-line in Computational Grid Settings*. ACM SIGMETRICS Performance Evaluation Review, 30(4):41–49, 2003.

[170] A. Wong, D. Rexachs y E. Luque: *Extraction of Parallel Application Signatures for Performance Prediction*. En *12th IEEE International Conference on High Performance Computing and Communications*, HPCC, 2010.

[171] Xingfu Wu: *Performance, evaluation, prediction and visualization of parallel systems*. Kluwer Academic Publishers, 1999.

[172] Xingfu Wu, Valerie Taylor, Jonathan Geisler y Rick Stevens: *Isocoupling: Reusing Coupling Values to Predict Parallel Application Performance*. En *18th International Parallel and Distributed Processing Symposium*, IPDPS, 2004.

[173] Xingfu Wu, Valerie Taylor y Joseph Paris: *A Web-based Prophesy Automated Performance Modeling System*. En *International Conference on Web Technologies, Applications and Services*, WTAS, 2006.

[174] Juekuan Yang, Yujuan Wang y Yunfei Chen: *GPU accelerated molecular dynamics simulation of thermal conductivities*. Journal of Computational Physics, 221(2):799–804, 2007.

[175] O. Zaki, E. Lusk, W. Gropp y D. Swider: *Toward Scalable Performance Visualization with Jumpshot*. The International Journal of High Performance Computing Applications, 13(3):277–288, 1999.

[176] Gengbin Zheng, Gunavardhan Kakulapati y Laxmikant V. Kale: *BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines*. En *IEEE International Parallel and Distributed Processing Symposium*, 2004.

Índice de figuras

Índice de tablas