Universitat Autònoma de Barcelona

Departament d'Arquitectura de Computadors i Sistemes Operatius

Màster en Computació d'Altes Prestacions

Análisis y Sintonización de Aplicaciones Paralelas/Distribuidas de Bioinformática: Caso de Estudio mpiBLAST

Memoria del trabajo de investigación relizado por Claudia Rosas Mendoza para optar al título de "Máster en Computación de Altas Prestaciones", bajo la dirección de la Dra. Anna Barbara Morajko.
Presentada en la Escuela Técnica Superior de Ingeniería (Departamento de Arquitectura de Computadores y Sistemas Operativos)

Bellaterra, Julio de 2009

Trabajo de investigación

Máster en Computación de Altas Prestaciones

Curso 2008-09

Análisis y Sintonización de Aplicaciones Paralelas/Distribuidas de Bioinformática: Caso de Estudio mpiBLAST

Autor

Claudia A. Rosas Mendoza

Directora

Anna Barbara Morajko

Departamento Arquitectura de Computadores y Sistemas Operativos

Escuela Técnica Superior de Ingeniería (ETSE)

Universitat Autònoma de Barcelona

Firma directora

Firma Autor

Resum

En termes de temps d'execució i ús de dades, les aplicacions paral·leles/distribuïdes poden tenir execucions variables, fins i tot quan sempra el mateix conjunt de dades déntrada. Existeixen certs aspectes de rendiment relacionats amb léntorn que poden afectar dinàmicament el comportament de l'aplicació, tals com: la capacitat de la memòria, latència de la xarxa, el nombre de nodes, lheterogeneïtat dels nodes, entre d'altres. És important considerar que l'aplicació pot executarse en diferents configuracions de maquinari i el desenvolupador d'aplicacions no pot garantir que els ajustaments de rendiment per a un sistema en particular continuïn essent vàlids per a altres configuracions. L'anàlisi dinàmica de les aplicacions ha demostrat ser el millor enfocament per a l'anàlisi del rendiment per dues raons principals. En primer lloc, ofereix una solució molt còmoda des del punt de vista dels desenvolupadors mentre que aquests dissenyen i evaluen les seves aplicacions paral·leles. En segon lloc, perquè s'adapta millor a l'aplicació durant l'execució. Aquest enfocament no requereix la intervenció de desenvolupadors o fins i tot l'accés al codi font de l'aplicació. S'analitza l'aplicació en temps real d'execució i es considera i analitza la recerca dels possibles colls d'ampolla i optimitzacions. Per a optimitzar l'execució de l'aplicació bioinformática mpiBLAST, vam analitzar el seu comportament per a identificar els paràmetres que intervenen en el rendiment d'ella, tals com: l'ús de la memòria, l'ús de la xarxa, patrons d'E/S, el sistema de fitxers emprat, l'arquitectura del processador, la grandària de la base de dades biològica, la grandària de la seqüència de consulta, la distribució de les seqüències dintre délles, el nombre de fragments de la base de dades i/o la granularitat dels treballs assignats a cada procés. El nostre objectiu és determinar quins d'aquests paràmetres tenen major impacte en el rendiment de les aplicacions i com ajustarlos dinàmicament per a millorar el rendiment de l'aplicació. Analitzant el rendiment de l'aplicació mpiBLAST hem trobat un conjunt de dades que identifiquen cert nivell de serial·lització dintre de l'execució. Reconeixent l'impacte de la caracterització de les seqüències dintre de les diferents bases de dades i una relació entre la capacitat dels workers i la granularitat de la càrrega de treball actual, aquestes podrien ser sintonitzades dinàmicament. Altres millores també inclouen optimitzacions relacionades amb el sistema de fitxers paral·lel i la possibilitat d'execució en múltiples multinucli. La grandària de gra de treball està influenciat per factors com el tipus de base de dades, la grandària de la base de dades, i la relació entre la grandària de la càrrega de treball i la capacitat dels treballadors.

Paraules claus: Anàlisi de Rendiment, Sintonización Dinàmica, mpiBLAST, Aplicacions Paral·leles Bioinfomáticas.

Resumen

En términos de tiempo de ejecución y uso de datos, las aplicaciones paralelas/distribuidas pueden tener ejecuciones variables, incluso cuando se emplea el mismo conjunto de datos de entrada. Existen ciertos aspectos de rendimiento relacionados con el entorno que pueden afectar dinámicamente el comportamiento de la aplicación, tales como: la capacidad de la memoria, latencia de la red, el número de nodos, la heterogeneidad de los nodos, entre otros. Es importante considerar que la aplicación puede ejecutarse en diferentes configuraciones de hardware y el desarrollador de aplicaciones no puede garantizar que los ajustes de rendimiento para un sistema en particular continúen siendo válidos para otras configuraciones. El análisis dinámico de las aplicaciones ha demostrado ser el mejor enfoque para el análisis del rendimiento por dos razones principales. En primer lugar, ofrece una solución muy cómoda para el punto de vista de los desarrolladores mientras que el diseña y evalua sus aplicaciones paralelas. En segundo lugar, porque se adapta mejor a la aplicación durante la ejecución. Este enfoque no requiere la intervención de desarrolladores o incluso el acceso al código fuente de la aplicación. Se analiza la aplicación en tiempo real de ejecución y se considera y analiza la búsqueda de los posibles cuellos de botella y optimizaciones. Para optimizar la ejecución de la aplicación bioinformática mpiBLAST, analizamos su comportamiento para identificar los parámetros que intervienen en el rendimiento de ella, tales como: el uso de la memoria, el uso de la red, patrones de E/S, el sistema de ficheros empleado, la arquitectura del procesador, el tamaño de la base de datos biológica, el tamaño de la secuencia de consulta, la distribución de las secuencias dentro de ellas, el número de fragmentos de la base de datos y/o la granularidad de

los trabajos asignados a cada proceso. Nuestro objetivo es determinar cuál de estos parámetros tienen mayor impacto en el rendimiento de las aplicaciones y cómo ajustarlos dinámicamente para mejorar el rendimiento de la aplicación. Analizando el rendimiento de la aplicación mpiBLAST hemos encontrado un conjunto de datos que identifican cierto nivel de serialización dentro de la ejecución. Reconociendo el impacto de la caracterización de las secuencias dentro de las diferentes bases de datos y una relación entre la capacidad de los workers y la granularidad de la carga de trabajo actual podrían ser sintonizadas dinámicamente. Otras mejoras también incluyen optimizaciones relacionadas con el sistema de ficheros paralelo y la posibilidad de ejecución en múltiples multinúcleo. El tamaño de grano de trabajo está influenciado por factores como el tipo de base de datos, el tamaño de la base de datos, y la relación entre el tamaño de la carga de trabajo y la capacidad de los trabajadores.

Palabras clave: Análisis de Rendimiento, Sintonización Dinámica, mpiBLAST, Aplicaciones Paralelas Bioinformáticas.

Abstract

In terms of execution time and data usage, parallel/distributed applications may have variable runtimes, even when using the same input data. There are certain performance aspects related to environment that may affect the dynamic behavior of the application, such as: memory capacity, network latency, number of nodes, node heterogeneity, among others. It is important to consider that the application can be executed on different hardware configurations. The application developer cannot guarantee that performance tuning for a particular system is still valid for other configurations. Dynamic analysis of applications has shown to be the best approach for performance analysis for two main reasons. First, it offers a very comfortable solution for developerspóint of view while designing and evaluation its parallel applications. Second, because it adapts better during the application execution. This approach does not require developer intervention or even access to the source code of the application. The current application runtime is considered and analyzed finding relevant bottlenecks and possible optimizations. To optimize the execution of mpiBLAST application, we analyze its behavior to identify the parameters involved in the application performance, such as: memory usage, network usage, I/O patterns, file system employed, processor architecture, biological database size, query sequence size, the sequence distribution inside them, number of database fragments and/or granularity of work assigned to each process. Our goal is to determine which of these parameters have higher impact in the application performance and how to tune them dynamically to improve the performance of the application. Analyzing the performance of mpiBLAST application we have found a data set that identifies certain level of serialization inside the execution. Recognize the impact of the characterization of the sequences inside the different databases and a relationship between the capacity of workers and the granularity of existing work that could be tuned dynamically. Other improvements also include optimizations related with the parallel file systems and the possibility of execution in a multithreaded multicore. The work grain size is influenced by factors as database type, database size, and the relationship between the size of workload and workers capacity.

Keywords: Performance Analysis, Dynamic Tuning, mpiBLAST, Parallel Biological Applications

A mis padres y mi hermana principalmente, y a toda mi familia en general
por la confianza y fé depositada en mi...

A Ania, Tomàs, Toni y Josep
por haberme recibido en su línea de investigación y estar a mi lado en cada paso...

A Emilio y Lola
por brindarme la oportunidad de pertenecer al departamento...

A Ronal, Caro y Juan Diego
por convertirse en mis amigos y familia acá...

A Genaro, Gonzalo, Yandi y Alvaro
por siempre tener una palabra de ayuda, consejo o aliento...

A todos los miembros de CAOS
por todo lo que he aprendido de cada uno de ustedes...

¡Muchísimas Gracias!

Índice general

Índice de figuras

Índice de tablas

Índice de ecuaciones

Índice de códigos fuente

Capítulo 1

Introducción

"Cuando estás buscando una aguja en un pajar, el más optimista usa guantes."

El pequeño libro de cosas para tener presentes.

En el siguiente capítulo se presenta la decripción general de la problemática que existe en cuanto a rendimiento en aplicaciones bioinformáticas del tipo mpiBLAST. Así como también se definen nuestros objetivos y restricciones asumidas, junto con la organización de este trabajo de investigación como tal.

1.1. Descripción general

El poder de cómputo de los ordenadores ha venido en aumento durante los últimos años [31] que junto con la introducción de conceptos como multiprocesamiento o procesamiento paralelo, facilitan el manejo de mayor cantidad de datos en menor tiempo; consiguiendo de esta manera alcanzar a resolver problemas cada vez más complejos con la ayuda de los computadores.

En campos científicos tan variados como el biológico, matemático, físico, médico, entre otros, se utilizan cada vez más aplicaciones que exigen grandes capacidades de procesamiento de datos, tales como: simulaciones en tiempo real, cálculos matemáticos complejos, búsqueda de genomas completos, etc. Aplicaciones que a su vez habrán de estar diseñadas para funcionar sobre sistemas de cómputo paralelos/distribuídos.

Para conseguir dar respuesta a esta solicitud de capacidad de procesamiento, se hace necesario comenzar a interconectar más de dos computadores para formar las conocidas granjas de computación (clústeres), que a su vez pudieran formar entornos multiclústers, y que actualmente, han crecido hasta formar los centros de supercomputación. Permitiendo así ejecutar más de una tarea simultáneamente y exigiendo de esta manera la introducción de términos y enfoques relacionados con programación paralela.

La misión ahora, de todos los desarrolladores, es diseñar aplicaciones no sólo eficientes sino que además sean capaces de sacar provecho de las capacidades de los sistemas de computación

de altas prestaciones (*HPC*, *High Performance Computing*); convirtiéndose esto en una de las situaciones más complejas de solventar cuando de rendimiento de aplicaciones se trata.

El diseño y análisis de rendimiento de las aplicaciones paralelas exige que los desarrolladores tengan un alto nivel de experticia y comprensión de la aplicación y el entorno sobre el cual va a ser ejecutada. El conocimiento del comportamiento de la aplicación es primordial para su paralelización ya que facilita su desarrollo y consigue que sea llevado a cabo adecuadamente, obteniendo el mejor rendimiento posible.

Las aplicaciones paralelas no son sólo difíciles de concebir sino también difíciles de analizar, comprender y depurar, es por ello que muchas veces los desarrolladores se valen de herramientas adicionales para observar el comportamiento que están presentando sus aplicaciones. Labor que no libera al desarrollador de tener que comprender los resultados gráficos y/o numéricos que éstas herramientas arrojan y tener que modificar de vez en cuando alguna línea de código, recompilar y reiniciar su aplicación para visualizar si el cambio que se ha hecho ha sido suficiente para solventar la situación de ineficiencia.

Cuando se trata de ineficiencias es necesario saber identificar el punto exacto donde se debe realizar la modificación, y conocer a fondo el comportamiento de la aplicación bajo uno u otro entorno. Jain [19] señala que una metodología de mejora de rendimiento típica involucra tres fases claramente identificables, que son: monitorización, análisis y modificación de código.

Varios trabajos han sido llevados a cabo [26],[37],[32] para facilitar ésta tarea a los desarrolladores y eximirlos de tener que entrar en código "desconocido". Herramientas que bajo diferentes enfoques de análisis y mejoras de rendimiento, han sido diseñadas con la finalidad de hacer más cómodo el proceso de evaluación de aplicaciones bajo entornos paralelos.

Inicialmente éstas se basaban en un enfoque estático y/o automático; es decir, una vez que la aplicación ha finalizado su ejecución era posible observar los cuellos de botella que se habían presentado durante esa única ejecución. Situación que tal como se ha señalado antes obliga al desarrollador a pausar la aplicación para realizar las modificaciones pertinentes y luego iniciar una nueva ejecución.

El proceso de análisis de rendimiento de las aplicaciones paralelas era llevado a cabo de forma estática, esto quiere decir que, una vez que la aplicación finalizaba se podia observar, basándose en datos o gráficos, la posible existencia de ineficiencias en esa única ejecución. Los datos sobre los que se tomaba alguna decisión eran producto de ejecuciones anteriores, en los que si la aplicación presentaba comportamientos irregulares aún con un mismo conjunto de datos, no era aconsejable emplearlos como base para tomar alguna decisión en particular.

Estos datos podían ser observados a través de trazas de eventos en los que se reflejan todas las funciones que ha ejecutado la aplicación en tiempo de ejecución. Éstas trazas eran obtenidas como producto de una instrumentación previa que se realizaba en funciones que componen la aplicación, generando un alto nivel de intrusión en las ejecuciones, que afectaba el tiempo de real de ejecución de la aplicación.

Éste tipo de análisis es conocido como análisis *post-mortem* por su característica de visualizar

el comportamiento de la aplicación una vez que la ejecución ha "muerto" (finalizado). De forma que se trabajaba en función de datos antig´uos, que no garantizaban que la modificación que se realizara posteriormente solventara la ineficiencia existente o en el peor de los casos generara alguna otra.

Hace algunos años y partiendo de éstos enfoques, surge la posibilidad de realizar todo el proceso de análisis de prestaciones de aplicaciones paralelas/distribuidas de forma dinámica; es decir, identificar los cuellos de botella, realizar las modificaciones adecuadas y sacar conclusiones de los resultados que se obtengan una vez que han sido efectuados los cambios, todo esto en tiempo de ejecución. El desarrollador no se vería en la necesidad de conocer el proceso interno de la aplicación ni habría necesidad de pausar las ejecuciones para tomar mediciones. Ejemplo de ello son las aproximaciones como la de Morajko [25] que, con su herramienta MATE (Monitoring Analysis and Tuning Environment, por sus siglas en inglés) crea un entorno que se encarga de la monitorización, análisis de rendimiento y sintonización de aplicaciones paralelas/distribuidas de forma dinámica.

Se desean acercar todos estos enfoques de análisis de rendimiento y sintonización dinámica a las aplicaciones científicas existentes hoy en día en el mercado. Incorporando conceptos de computación de altas prestaciones en el análisis de rendimiento de aplicaciones bioinformáticas como mpiBLAST [12].

El contexto bajo el que se desenvuelven las aplicaciones bioinformáticas como mpiBLAST está directamente relacionado con el estudio de los seres vivos en general. Un ser humano tiene en su organismo un cierto código genético formado por largas cadenas de caracteres denominadas ADN (ácido desoxirribonucleico), estas cadenas tienen forma de doble hélice en la que moléculas de azúcar denominados desoxirribosa y están ligados a compuestos como la Adenina, Timina, Guanina y Citosina, que a su vez están agrupados en pares; cada uno de estos caracteres es conocido como una *base* y nuestro código genético, así como el de la mosca de la fruta, el ratón o la levadura, está representado por cadenas de texto conformadas por A, C, T o G. Es decir, el ADN de una persona siguiendo el formato FASTA (descrito en el capítulo 4podría verse como:

>ADN_de_XXXXXXXX
ACGGGGGTTACTACGTCCCAAACTGACGTACCCGTAAACCCACGGGGGTTACT...

Por lo tanto una de las tareas más importantes de este tipo de aplicaciones de alineamiento de secuencias es encontrar rasgos en común, tanto estructurales como de procesos generales entre genomas, lo que se podría traducir a la capacidad que tiene el ser humano de heredar ciertos rasgos basándose en una característica propia de herencia existente en la mosca de la fruta *Drosophila melanogaster* [1]. Es por ello que lo importante de las herramientas de bioinformática es que son utilizadas alineamientos de secuencias que en lugar de experimentos de laboratorio se pueden estudiar las funciones de genes ya descubiertos buscando secuencias genómicas en bases de datos.

La biología logró obtener su primer conjunto de datos fundamental: las secuencias molecu-

[1] Drosophila="amante del rocío" melanogaster= "estómago negro"

lares. A principios de 1960, las primeras proteínas conocidas fueron almacenadas lentamente (cabe resaltar que las computadoras capaces de analizarlas no habían sido desarrolladas aún) las secuencias eran ensambladas, analizadas y comparadas (manualmente) escribiéndolas en hojas de papel, adhiriendo una al lado de la otra en las paredes de los laboratorios y/o moviéndolas alrededor hasta encontrar el alineamiento óptimo, conocido actualmente como patrones.

Tan pronto como las primeras computadoras estuvieron disponibles (grandes y rápidas con 8KB de RAM), los primeros biólogos computacionales comenzaron a insertar estos algoritmos manuales en los bancos de memoria. Surgiendo ésta nueva tendencia, ya que nadie antes de esto habia manipulado las secuencias biológicas como texto; la mayoría de los métodos tuvieron que ser inventados desde cero y a medida que avanzaba el proceso, una nueva área de investigación, el análisis de secuencias de proteínas usando computadoras, era creada. Este fue entonces el génesis de la bioinformática.

La primera técnica eficiente de secuenciación de ADN fue descubierta en 1977 y en 1995, la primera secuencia de un genoma entero (del microbio *Hemophilus influenzae*) fue determinada. Entre éstas dos fechas, las tecnologías de secuenciación de ADN fueron mejorando constantemente, pero dichas tecnologías aún tienden a concentrarse en la búsqueda de genes individuales para detallar información sobre ellos. Durante este período los biológos estuvieron secuenciando fragmentos de ADN que eran unos cuantos miles de nucleótidos de longitud, simplemente porque ellos estaban interesados en genes [2] específicos con los que habían empezado a trabajar años atrás. La mayoría de las herramientas bioinformáticas fueron creadas durante esta época, incluyendo: todos los programas básicos de alineamiento de secuencias, métodos de clasificación filogenéticos y varias herramientas de visualización adaptadas a pequeñas cadenas de proteínas de no más que unos miles de carateres de longitud.

Las bases de datos de secuencias son excelentes herramientas porque permiten aprender del pasado. Ellas permiten responder las interrogantes biológicas de hoy en día permitiendonos analizar secuencias que pueden haber sido determinadas hace más de 25 años atrás, cuando toda la tecnología había surgido. Haciendo esto, ellas conectan el pasado y el presente de la biología molecular. Las primeras bases de datos fueron creadas, de hecho, como una especie de museo de secuencias, donde las secuencias podían ser preservadas para toda la eternidad de una forma prístina, tal como fueron determinadas, interpretadas y publicadas por sus autores originales. Esta perspectiva histórica se mantiene en el banco genético (*GenBank*), el repositorio lider de secuencias de nucleótidos mantenida como un consorcio entre el National Center for Biotechnology Information (NCBI) de los Estados Unidos, el European Molecular Biology Laboratory (EMBL) y el DNA Data Bank de Japón (DDBJ).

Estos repositorios de bases de datos son herramientas muy útiles cuando se quiere obtener toda la información relacionada con una secuencia en particular, pero no proveen fácil acceso a la información de la secuencia cuando ésta abarca muchos tópicos relacionados a un gen o a

[2]Del griego genos = nacimiento, raza; del latín genus = raza, origen: segmentos específicos de ADN que controlan las estructuras y funciones celulares; la unidad funcional de la herencia. Secuencia de bases de ADN que usualmente codifican para una secuencia polipetídica de aminoácidos.

una función del gen más allá de su papel específico. Es por ello, que una segunda generación de bases de datos de nucleótidos ha adoptado una perspectiva más orientada al gen, donde toda la información relevante de un gen dado es accesible de una vez.

Actualmente, las secuencias de nucleótidos son determinadas de forma rutinaria a escalas que oscilan desde un genoma completo hasta el nivel de cromosomas. Ahora tenemos información no solo acerca de secuencias de genes, sino también de sus posiciones relativas a su orientación, y la presencia o ausencia de funcionas bioquímicas sobre un organismo completo. Para sacar provecho de esta información en forma global, los investigadores han tenido que diseñar un estado de arte centrado en el manejo de sistemas de información de secuenciación genómica que pueden conectar colecciones de secuencias especializadas con herramientas de búsqueda.

Una de las principales problemáticas que disciplinas como la bioinformática ha tenido que enfrentar ha sido el aumento considerable del número de datos con los que tienen que trabajar cada día; el tamaño de las bases de datos de ADN o proteínas han presentado un crecimiento exponencial durante los últimos años (ver figura 1.1), trayendo consigo la necesidad de disponer de sistemas computacionales que logren procesar este volumen de datos y generar resultados de la forma más rápida posible.

Tareas como el alineamiento local de secuencias, que versa en la comparación de dos secuencias para encontrar su homología, el cómo se pareces respecto a un ancerstro en común, es unas de las principales actividades llevadas a cabo más a menudo entre los biólogos.

Para agilizar este proceso se han actualizado algunos algoritmos de alineamiento existentes hasta la época, como en el caso del algoritmo de Smith-Waterman [33] donde identificar el máximo nivel de homología entre subsecuencias biológicas y conjuntos de largas secuencias era uno de los problemas más importantes en análisis molecular, y con la intención de conseguir un aumento en la velocidad de respuesta para cada consulta, intentos como [21] y la herramienta Basic Local Alignment Sequence Tool (BLAST)[1] descrito en 4.2.1 fueron las primeras aproximaciones a la automatización de dicho algoritmo, convirtiéndose ésta última en la herramienta base de referencia para la bioinformática en la última década.

Este tipo de aplicaciones ha dado pie al interés de diferentes grupos de investigación y desarrollo de construir una herramienta que permitiera que la filosofía de BLAST como tal, que se había convertido en parte fundamental de procesos de búsqueda de homología de secuencias, comulgara con el nuevo paradigma de paralelismo característico de la computación de altas prestaciones; naciendo de esta forma mpiBLAST, que sin duda alguna por ser una aplicación paralela requiere cierto nivel de análisis y evaluación para garantizar su máximo rendimiento en todas y cada una de sus ejecuciones según la máquina paralela en la que se esté ejecutando. Convergiendo de esta forma ambos, análisis de rendimiento de aplicaciones paralelas y aplicaciones bioinformáticas para dar lugar a nuestro trabajo de investigación.

La herramienta bioinformática paralela mpiBLAST, ha sido diseñada basada en el algoritmo básico de BLAST [1] (Basic Local Alignment Sequence Tool) que versa en un conjunto de pasos de programación dinámica planteados por Smith-Waterman [33] en el año 1981 donde se

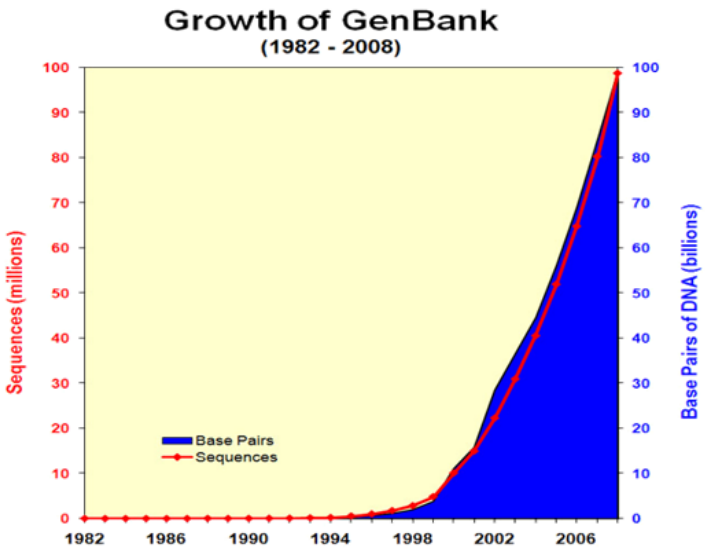

Figura 1.1: Crecimiento de las bases de datos biológicas en los últimos años.

considera cada cadena biológica como una cadena de texto, y luego se comparan uno a uno cada caracter de cada una de las dos cadenas que se estén secuenciando.

Inicialmente, siguiendo el algoritmo de BLAST, cada una de las secuencias se ubica una en el eje 'X' y otra en el eje 'Y' de una matriz de puntuación PAM o BLOSUM (la selección de la matriz depende de la naturaleza de las secuencias biológicas), y cada uno de los aciertos y desaciertos entre las secuencias es puntuado o penalizado respectivamente con un determinado valor.

Una vez que se ha construido todo el camino de alineamiento entre ambas secuencias se procede a seleccionar aquéllas con mayor grado de relevancia biológica, puntuando esta característica bajo un valor obtenido (*e-value*) a través de datos estadísticos de la base de datos y de las secuencias resultantes como tal, de esta forma se descartan aquéllos que podrían haber sido generados y acertados solamente por azar.

Tomando en consideración el crecimiento exponencial que vienen presentando las bases de datos biológicas en los últimos diez años, existen costos en los que es necesario comparar cadenas de miles de caracteres, lo que quizá por ser comparación de caracteres no consuma tanto poder de cómputo como una multiplicación de matrices sino que en cambio estamos hablando del manejo de un volumen superior a las capacidades de un servidor estándar.

Dado que el algoritmo en sí no presenta dependencia de datos entre una secuencia y otra, y que muchas de las bases de datos ya no caben en la memoria de un sólo ordenador, los creadores de mpiBLAST se han planteado separar la base de datos en fragmentos manejables de menor tamaño que puedan ser alojados en la memoria de los nodos de cómputo de las granjas de ordenadores, de esta forma llevando a cabo ejecuciones embarazosamente paralelas, se puede realizar la búsqueda de algunas secuencias en bases de datos biológicas cada vez mayores.

Para llevar a cabo una consulta en un entorno paralelo, es necesario preformatear las bases de datos en un conjunto de fragmentos que serán distribuídos a cada uno de los nodos de cómputo, entiéndase que es una aplicación paralela diseñada bajo un paradigma Master/Worker, en el que existe un nodo que figura como máster y coordina y distribuye el trabajo que va a realizar cada uno de los nodos de cómputo que figuran como workers, y que posteriormente recolectará los datos generados por cada uno de ellos para construir un fichero resultante final. La fase de formateo de la base de datos que acabamos de comentar básicamente fragmenta la base de datos en un número de segmentos igual o mayor al número de workers disponibles y para lograr esto han generado una estrategia en la que se leen todas las cadenas de la base de datos contra la que se va a realizar la búsqueda y se organizan de mayor a menor tamaño en un fichero temporal desde el cual se van a ir seleccionando de igual forma en orden descendente las cadenas para completar cada uno de los fragmentos, de forma que todos tengan el mismo tamaño, sin embargo no tendrán la misma cantidad de secuencias.

La herramienta mpiBLAST saca provecho del paralelismo de los nuevos centros de cómputo, enviando fragmentos de la base de datos (uno como mínimo) junto con la secuencia de consulta a cada worker; y todos de forma simultánea buscarán las secuencias similares dentro de su fragmento particular, de esta forma cada worker cubre una parte de la base de datos y se toma en cuenta como carga de trabajo la cantidad de datos existentes dentro de la misma. Fraccionando los datos de esta forma y repartiéndolos entre todos los workers se consigue abarcar conjuntos de datos masivos que son demasiado grandes para ser procesados en un sólo ordenador.

En estudios recientes se ha logrado llevar esta aplicación a entornos de supercomputación como el Blue Gene/P [23] en el que sacan provecho del paralelismo presente en la arquitectura del computador, y consiguen un 93 % de eficiencia en la ejecución de la aplicación, pero vale la pena recalcar que es un ordenador que está diseñado para ejecutar aplicaciones científicas que requieren un gran poder de cómputo y almacenamiento.

La importancia de esta investigación, es que analiza el rendimiento mpiBLAST como una aplicación bioinformática que emplea BLAST como algoritmo de alineamiento de secuencias, y estudiar y optimizar su rendimiento para grandes cantidades de datos.

1.2. Objetivos y Limitaciones

En los últimos años las aplicaciones científicas han tomado una posición significativa entre los diferentes grupos de investigación y han pasado a ser parte importante en el desarrollo de proyectos; mpiBLAST no es la excepción y precisa de ser analizada y evaluada en aras de alcanzar el mejor rendimiento posible en las consultas efectuadas. Las aplicaciones paralelas requieren un cuidado mayor durante su fase de diseño y desarrollo para que se adapten mejor a los entornos paralelos/distribuidos en los que va a ser ejecutada.

Cuando hablamos de *análisis de rendimiento* de aplicaciones paralelas, nos referimos al proceso de evaluación y búsqueda de ineficiencias. Ineficiencias, que a medida que se conoce el

comportamiento de la aplicación se podrán solventar con estrategias de sintonización.

El objetivo general de éste trabajo de investigación es *evaluar el rendimiento presentado por la aplicación paralela de análisis de secuencias biológicas mpiBLAST, con la finalidad de determinar aquéllos parámetros influyentes y sintonizables para aumentar el nivel de desempeño en términos de tiempo de ejecución bajo un determinado entorno paralelo.* El proceso de análisis de rendimiento está descrito en el capítulo 2.

En análisis de prestaciones es necesario realizar una monitorización previa de la aplicación; este proceso se realiza con el fin de identificar aquéllas funciones representativas durante la ejecución; es decir, las que están relacionadas con la aplicación y que consumen la mayor cantidad de tiempo, convirtiéndose en posibles cuellos de botella para el rendimiento de, en nuestro caso, mpiBLAST como tal.

Para las aplicaciones paralelas en general debemos conocer sus puntos de ineficiencia denominados cuellos de botella, donde el rendimiento de la aplicación se ve limitado a cierto conjunto de parámetros que le impiden alcanzar un mejor desempeño y conseguir sacar el mayor provecho posible del entorno paralelo/distribuido bajo el que están siendo ejecutadas.

Por tanto nuestra labor es, tener la habilidad para entender e identificar los factores contribuyentes en el rendimiento de un programa paralelo basándonos en el impacto que generan los diferentes parámetros del entorno en la generación de cuellos de botella dentro de mpiBLAST y proponer una estrategia que permita solventarlos con la finalidad de mejorar su rendimiento.

Para lograr este objetivo, es necesario:

1. Conocer y monitorizar la aplicación para identificar los procesos y funciones involucrados en la ejecución, y detectar cuáles consumen mayor cantidad de tiempo y recursos.

2. Entender e identificar los factores que contribuyen en el rendimiento basado en el impacto que ocasionan los diferentes parámetros relacionados con la aplicación y el clúster que generen cuellos de botella.

3. Identificar los problemas de rendimiento y efectuar los ajustes necesarios a los factores directamente relacionados con la ejecución de mpiBLAST que sean influyentes a fin de mejorar el nivel de desempeño.

La modificación y sintonización de código una vez finalizado el análisis, no deja de ser una tarea menos ardua; es por ello que una vez que hayan sido identificados los principales problemas de rendimiento de la aplicación se puede proceder a efectuar los ajustes respectivos en los factores influyentes con la finalidad de alcanzar los mejores niveles de desempeño posibles durante la ejecución. Es por ello que se espera conseguir durante la investigación la información suficiente que permita determinar los puntos en que mpiBLAST, como aplicación paralela/distribuida de análisis de secuencias biológicas, pueda mejorar.

El alcance de nuestra investigación se ve limitado al estudio del comportamiento de la aplicación bioinformática mpiBLAST sin entrar al módulo de alineamiento de secuencias como tal;

es decir, no se planea mejorar el algoritmo de BLAST sobre el que se basa mpiBLAST, sino determinar la mejor estrategia de configuración de los diferentes factores influyentes para obtener el mejor rendimiento en un clúster específico.

1.3. Organización del Trabajo

En este capítulo hemos introducido la situación general en la que se encuentra enmarcado nuestro proyecto de investigación, y el alcance que ésta tendrá. Los siguientes capítulos estarán dedicados a detallar los aspectos que están ligados a nuestro trabajo. Quedando de la siguiente manera:

Capítulo 2. Análisis de Rendimiento de Aplicaciones Paralelas. Introduce una descripción general del enfoque clásico de análisis de rendimiento de aplicaciones paralelas. En él se describe cuál es el objetivo que se persigue mientras se realiza esta clase de análisis y las características básicas de ineficiencia de una aplicación, conjuntamente con el enfoque inicial que se ha de tomar cuando se inicia un proceso de análisis de aplicaciones en cuanto a rendimiento. De igual forma, se presenta de forma descriptiva el estado de arte actual para lo que *Análisis de Rendimiento de Aplicaciones Paralelas* se refiere, tomando como base aportes previos y actuales proporcionados por los principales grupos de investigación en esta área a nivel mundial.

Capítulo 3. Modelos de Prestaciones y Sintonización. Describe detalladamente bajo qué situaciones se hace necesaria la participación de un proceso de *Sintonización Dinámica* y bajo cuales características es posible desarrollar un modelo de prestaciones que facilite la predicción del comportamiento de la aplicación.

Capítulo 4. Aplicaciones Bioinformáticas. Se centra en la descripción del conjunto de herramientas de las cuales se ha valido la bioinformática con el pasar de los años, sirviendo como base para la introducción general de la aplicación con la cual nos hemos planteado trabajar y que es nuestro objetivo aumentar su rendimiento.

Capítulo 5. Experimentación. Caso de Estudio: mpiBLAST En este capítulo se detallan todas y cada una de las diferentes evaluaciones y mediciones que le fueron hechas a mpiBLAST, junto con el modelado del comportamiento de la aplicación bajo determinadas condiciones de ejecución.

Capítulo 6. Conclusiones y Trabajos Futuros. Sumariza y concluye nuestro trabajo, se presentan los problemas que forman parte de nuestras líneas abiertas y se discuten direcciones para trabajos futuros.

Capítulo 2

Análisis de Rendimiento

"Mide lo que sea medible y haz medible lo que no lo sea."

Galileo Galilei (1564-1642).

Este capítulo está dedicado a introducir los conceptos de *análisis de rendimiento* necesarios para enmarcar el desarrollo del presente trabajo. Partiendo de la descripción del enfoque inicial de análisis basado en el uso de herramientas de visualización, y de los procesos de captura de información, tales como *profiling* y *tracing*, como puntos de referencia para determinar el rendimiento; seguidamente este enfoque de análisis de rendimiento fue sustituido por metodologías automáticas y dinámicas, que son la base de varias de las herramientas de análisis disponibles actualmente, las cuales serán comentadas brevemente al final del capítulo.

2.1. Introducción

Investigar el comportamiento de un sistema de cómputo o de una aplicación paralela, y encontrar limitantes en su rendimiento junto con los factores que las ocasionan, usualmente genera una modificación posterior. En el caso de una aplicación paralela esto sería cierta modificación del código fuente con la intención de mejorarla y/o adaptarla para evitar futuras pérdidas de rendimiento.

Este proceso se encuentra definido en el análisis de rendimiento en tres grandes etapas descrito por Jain [19], donde inicialmente se realiza:

- Una fase de monitorización y conocimiento de la aplicacón, bien sea observando resultados gráficos o numéricos obtenidos luego de cada ejecución o con la ayuda de herramientas automáticas o dinámicas, que proporcionará al analista un conjunto de datos e información.
- Una fase de análisis, en la que determinará la existencia de limitantes de rendimiento dentro de la aplicación.

- Y la fase de sintonización, en la que se corregirán las posibles ineficiencias que se estuvieran presentando.

Si miramos el análisis de rendimiento dentro del campo experimental de la computación de altas prestaciones se puede tomar como un proceso en el que se combina medición, interpretación y comunicación de cierta característica del objeto que está siendo evaluado, característica que puede ser su tiempo de ejecución, su capacidad para escalar, el número de tareas que es capaz de llevar a cabo en un determinado período de tiempo. Cuando se plantea la intención de realizar análisis de rendimiento, usualmente surge la interrogante en cuanto a la forma en que debe realizarse la medición o la interpretación de los datos; por ello se suele valerse de la inventiva y la imaginación de cada uno de los analistas para diseñar o proponer estrategias que se adecúen de mejor forma al objetivo que se persigue, realizando la menor perturbación posible dentro de la aplicación que se estudia.

Antes de comenzar el proceso de análisis de rendimiento, es necesario identificar no sólo los aspectos relacionados con la aplicación como tal sino también auéllas cosas útiles o interesantes para medir [22]. Usualmente suelen medirse características como:

- El número de veces que se repite un evento.
- La duración en tiempo o instrucciones de un proceso.
- El tamaño de algún parámetro.

Al final se convierten en nuestras *métricas de rendimiento*, que son diferentes valores medidos que se emplean para describir el rendimiento de la aplicación, por ejemplo: el número de veces que se entra a cierta rutina o sub-rutina, cuánto tiempo se está gastando en ella o la cantidad de datos que están siendo transmitidos en determinado momento.

Escoger una métrica apropiada depende de la situación específica para la que es requerida y la inversión necesaria para la recolección de los datos, así como también debe ser tomado en cuenta que la métrica para que realmente sea útil para nosotros como analistas.

- Linearidad: se suele esperar que la métrica seleccionada varíe en cierta proporción directamente proporcional con las variaciones observadas en el rendimiento del sistema.
- Confiabilidad: si existe una variación X en la aplicación, se espera que la métrica indique que ésa variación va a suceder.
- Repetibilidad: una métrica es repetible si cada vez que se realice un mismo experimento se obtienen valores similares, es decir, con una desviación estándar baja.
- Facilidad de medida: cuanto más difícil es determinar una métrica, mayor posibilidad existe de que el valor sea errado o impreciso; y aún más que dicha métrica no sea utilizada del todo.

- Consistencia: una métrica consistente es aquélla para la cual las unidades de medida y su definición exacta es la misma para diferentes sistemas o configuraciones de un mismo sistema.

- Independencia: la métrica no debe verse influenciada por factores externos.

Para nuestra investigación, métrica que se tomará en cuenta es el *tiempo de ejecución*, debido a que uno de los objetivos fundamentales de la HPC es brindar respuestas en un menor tiempo, y en el caso de la bioinformática que obliga a trabajar con datos de tamaño significativo, este tiempo de respuesta es un valor muy importante y muy fácil de comparar para tomar decisiones. Es por ello que es importante saber cómo medir el tiempo de ejecución de una aplicación, de una porción de la aplicación, y entender las limitaciones de la herramienta de medida; sin embargo estos conceptos serán ampliados en el capítulo 5.

2.1.1. Objetivos del Análisis de Rendimiento

En todo proceso de análisis de rendimiento, el objetivo es conocer el comportamiento de la aplicación para poder evaluarla y determinar cierto rendimiento deseado. Inicialmente, observando el proceso de análisis de rendimiento como una secuencia de pasos que deben llevarse a cabo se pueden definir como objetivos los siguientes [20]:

- Se persigue determinar si existen problemas de rendimiento, factores de ineficiencia y características que faciliten su identificación.

- Espera detallar el punto exacto donde se está sucediendo el problema de rendimiento y/o cuello de botella, qué lo ocasiona.

- Trazar una estrategia para evitar que se genere el problema de rendimiento o de ser posible, eliminarlo.

- Comprobar si una vez corregido el problema inicial de rendimiento, existe algún otro más y si es necesario iniciar el proceso de análisis de nuevo.

De acuerdo a lo señalado por Jorba [20], se podrían considerar modelos en los que pueda ser interesante el mínimo tiempo de ejecución de la aplicación, un cierto grado de eficiencia de uso de recursos(como el balanceo de trabajo entre los procesadores), una escalabilidad apropiada según el incremento de procesadores involucrados en el proceso.

2.2. Ineficiencia en las Aplicaciones Paralelas.

Cuando comenzamos a hablar de aplicaciones paralelas, es necesario realizar el análisis y evaluación de las mismas, que en un principio es el pilar fundamental de este trabajo de investigación siendo necesaria la descripción de algunos conceptos fundamentales, descritos a continuación.

2.2.1. Ecuaciones para Análisis de Tiempo de Ejecución Paralelo.

La primera interrogante que nos planteamos es cuán rápida la implementación paralela puede ser, para ello es necesario conocer el tiempo de ejecución en un único computador, t_s, contando los pasos computacionales del mejor algoritmo secuencial. Para un algoritmo paralelo, en aras de determinar el número de pasos computacionales, es necesario estimar el *overhead* de comunicación. En un sistema de paso de mensajes, el tiempo de envio de mensajes debe ser considerado en el tiempo total de ejecución de un problema. El tiempo de ejecución paralelo, t_p, está compuesto de dos partes, una parte computacional, llamada t_{comp} y una parte de comunicación, llamada t_{comm}; de forma que nos queda que:

$$t_p = t_{comp} + t_{comm} \tag{2.1}$$

Tiempo Computacional: viene dado al igual que con los algoritmos secuenciales por contar el número de pasos computacionales. Cuando mas de un proceso está siendo ejecutado simultáneamente, solamente podemos contar los pasos computacionales del proceso más complejo. A menudo, todos los procesos están llevando a cabo la misma operación, por lo que se puede contar los pasos computacionales de uno de los procesos. Sin embargo pueden presentarse situaciones en las que se obtenga el mayor número de pasos computacionales contando la concurrencia de algunos procesos. Muchas veces este tiempo se fragmenta en partes deteminadas por el paso de mensajes para luego determinar el tiempo computacional de cada parte [36]. Y viene dado por la función:

$$t_{comp} = f(n, p) \tag{2.2}$$

De donde n es el número de veces que el proceso p realiza cierto conjunto de pasos computacionales. Pudiendo determinar el tiempo total de cómputo con la suma de los diferentes tiempos de cómputo de cada proceso.

$$t_{comp} = t_{comp1} + t_{comp2} + t_{comp3} + \ldots \tag{2.3}$$

Usualmente cuando se analizan aplicaciones paralelas se asume que los tiempos de cómputo serán semejantes porque los procesadores son el mismo y el sistema operativo es el mismo. Ssin embargo, y tomando en cuenta que con el HPC todo el cómputo ha migrado a clústers es necesario recordar la posibilidad de que el entorno no sea homogéneo, para nuestro trabajo de investigación con la aplicación paralela máster/worker mpiBLAST, vamos a considerar que estamos trabqajando sobre una arquitectura paralela homogénea y específica.

Tiempo de Comunicación: el tiempo de comunicación va a depender del número de mensajes,

el tamaño de cada mensaje, la estructura de comunicación que exista por debajo y el modo de transferencia. Para una primera aproximación se puede decir que [36]:

$$t_{comm1} = t_{startup} + wt_{data} \tag{2.4}$$

Para el tiempo de comunicación del mensaje 1, donde $t_{startup}$, es el tiempo de inicio, algunas veces llamada *latencia del mensaje*. Este tiempo es esencialmente el tiempo necesitado para enviar un mensaje sin datos. Incluye el tiempo del empaquetado del dato en la fuente y desempaquetar el mensaje en el destino y es considerado constante. t_{data} es el tiempo de transmisión para enviar una palabra de datos, también asumido como constante, y hay w palabras de datos. La tasa de transmisión normalmente es medida en $bits/segundos$. El tiempo de comunicación final t_{comm}, será la sumatoria de los tiempos de comunicación de todos los mensajes secuenciales provenientes de los procesos. De esta forma:

$$t_{comp} = t_{comp1} + t_{comp2} + t_{comp3} + \ldots \tag{2.5}$$

Una vez que ha sido determinado el valor de tiempo secuencial t_s, el tiempo de cómputo t_{comp} y el tiempo de comunicación t_{comm}, podemos establecer el factor de *speedup*, que se refiere a:

$$S(p) = \frac{t_s}{t_p} \tag{2.6}$$

El máximo speedup posible es p con p procesadores, conocido como *speedup linear*. El speedup de p será adquirido cuando la computación puede ser dividida en pasos computacionales de igual tiempo de duración, con un proceso mapeado en un procesador y sin overhead adicional en la solución paralela.

Adicionalmente, es importante hacer mención sobre el concepto de *eficiencia*, y se refiere al cuán utilizados están los procesadores. La eficiencia E, está definida por:

$$E = \frac{t_s}{t_p * p} \tag{2.7}$$

Sin embargo, suelen aparecer factores dentro de las aplicaciones paralelas que limitan el speedup notablemente, como períodos en los que no todos los procesadores están llevando a cabo tareas, cómputo extra que no existe dentro de la versión secuencial, y el tiempo de comunicación entre los procesos. Era de esperar que algunas partes de la computación no puedan ser divididas en tareas concurrentes, y deban ser llevadas a cabo de forma secuencial. Asumiendo la posibilidad de que algunas partes sean realizadas por un sólo procesador, la situación ideal sería que todos

los procesadores disponibles trabajaran simultáneamente para los otros momentos. Si la fracción de cómputo que no puede ser dividida en partes concurrentes es f, y no se incurre en overhead alguno cuando el cómputo es dividido en partes concurrentes, el tiempo para llevar a cabo el cómputo con p procesadores viene dado por $ft_s+(1-f)t_s/p$. De esta forma Amdahl [3] identifica que el speedup sería:

$$S(p) = \frac{t_s}{ft_s + (1-f)t_s/p} = \frac{p}{1+(p-1)f} \tag{2.8}$$

Que señala que aún con infinitos procesadores, la parte secuencial va a limitar el rendimiento global de la aplicación.

2.2.2. Casos de ineficiencias.

Cuando se realiza un proceso de análisis de rendimiento, se pueden encontrar diferentes enfoques bajo los que ocurren problemas de eficiencia; existiendo ciertas características muy relacionadas al tipo de comunicación empleada en el paso de mensajes, bien sean punto a punto o colectivas o según su estructura [20].

Punto a Punto

Existen problemas relacionados con las tareas participantes que emplean operaciones de recepción bloqueante y no bloqueantes, siendo las primeras las principales causantes de los mayores tiempos de espera; sin embargo, las no bloqueantes pudieran llegar a presentar comportamientos similares dependiendo del esquema de comunicación que se esté empleando.

1. *Late Sender*: ésta se presenta cuando el envío del mensaje por parte del emisor se ha efectuado tiempo después de que el receptor (ver figura 2.1) haya iniciado el proceso de recepción, ocasionando al proceso receptor un tiempo inactivo que pudo haberse empleado en cómputo útil.

2. *Late Receiver*: en este caso el emisor ya ha generado la señal de envío del mensaje, pero el proceso receptor aún no ha llegado al punto de estar listo para recibirlo (figura 2.2); este bloqueo suele observarse cuando es una recepción síncrona y hasta que el receptor no esté apto para responder a la petición de recepción.

3. *Blocked Sender*: en este se ven involucradas al menos tres tareas (figura 2.3), donde una comunicación entre dos procesos se encuentra previamente bloqueada porque el emisor está bloqueado esperando la respuesta de un proceso anterior a él, suele presentarse cuando existe cierta dependencia entre los datos con los que se está trabajando.

4. *Wrong Order*: suele suceder cuando se está trabajando con operaciones de comunicación no bloqueantes donde la estrategia de comunicación ha fallado, se ven involucrados

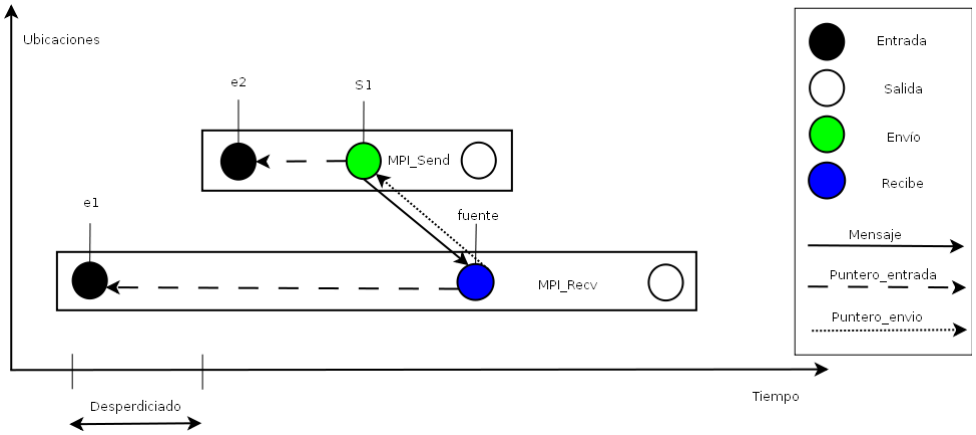

Figura 2.1: Late Sender

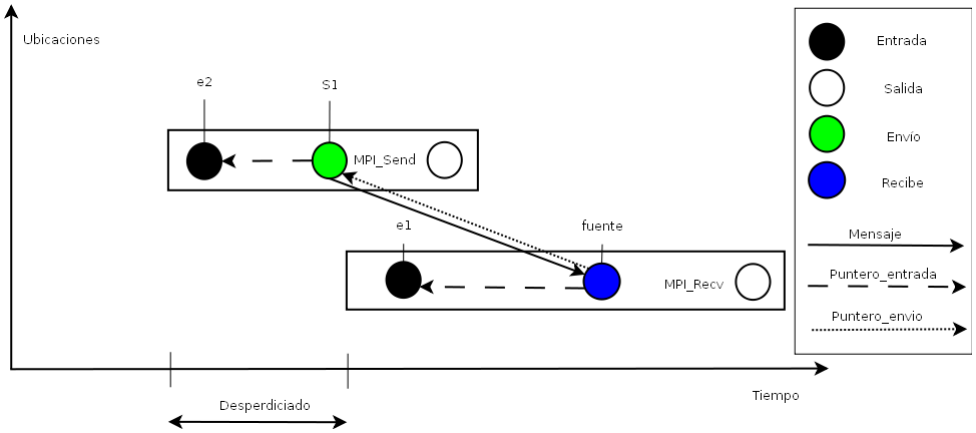

Figura 2.2: Late Receiver

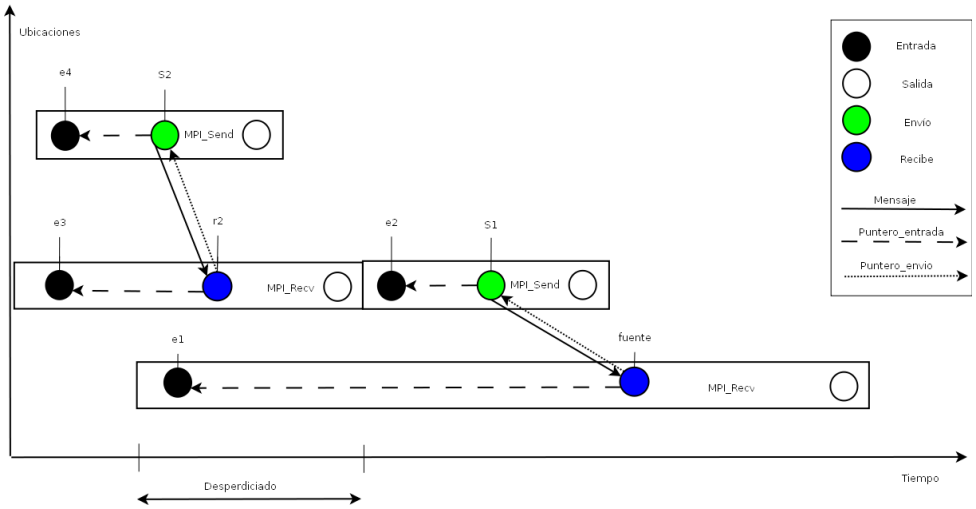

Figura 2.3: Blocked Sender

varios procesos con sus respectivos envíos y recepciones almacenados en el búffer de datos de forma diferente a como deberían ser entregados. En el peor de los casos, los mensajes pudieran llegar en orden contrario (ver figura 2.4) al que se iban a consumir. Si este caso se diera con envíos bloqueantes, nos encontraríamos frente a una situación de *deadlock* entre las tareas, ya que el receptor estaría esperando el mensaje y el emisor no lo enviaría hasta que el receptor realice un consumo de mensaje que no va a suceder.

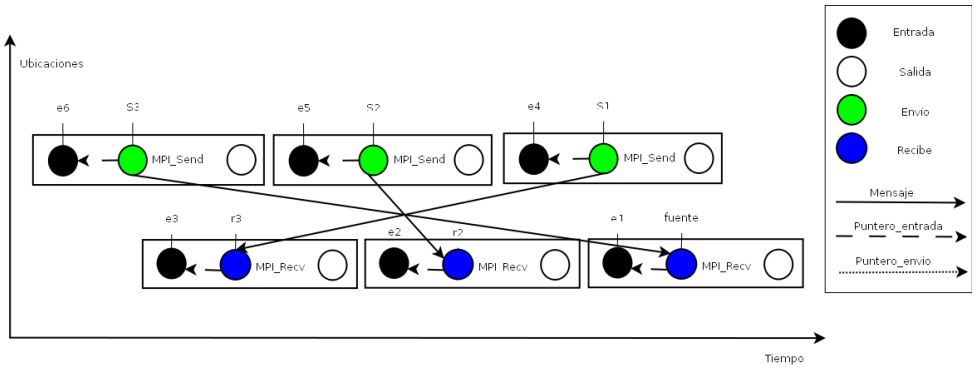

Figura 2.4: Wrong Order

5. *Multiple Output*: acá nos encontramos con un conjunto de tareas que son receptoras de secuencias de mensajes que se inician en serie desde un emisor común. El envío en serie provoca que las tareas se bloqueen hasta que les llegue el mensaje que les corresponda (ver figura 2.5), las tareas se encontrarían bloqueadas esperando un mensaje por parte del emisor, que aún no ha realizado el envío por estar realizándolo en ese momento a otro proceso específico.

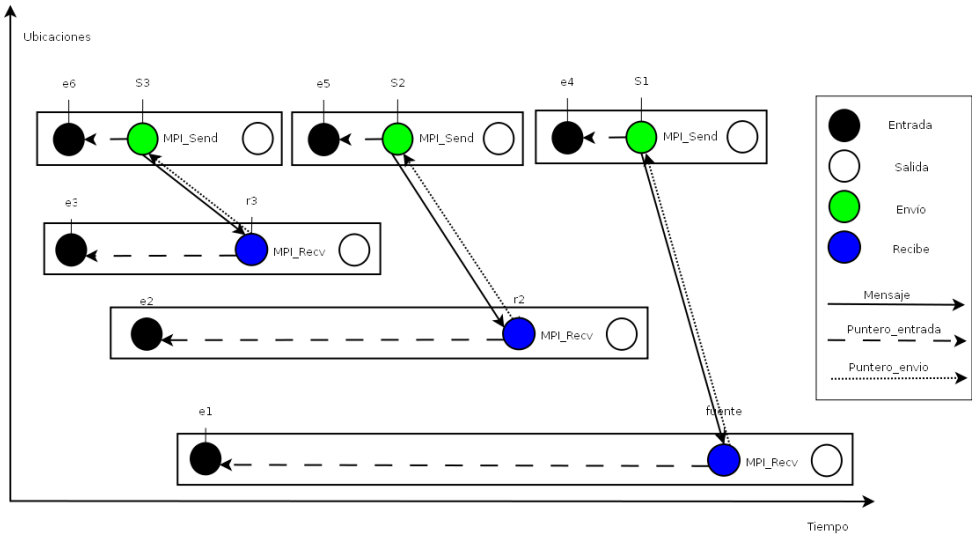

Figura 2.5: Multiple Output

Colectivas

Las operaciones colectivas, por su necesidad de realizar la operación de forma conjunta, suponen un cierto tiempo de bloqueo entre una o todas las tareas, ocasionando que la aplicación se detenga casi (o) en su totalidad, con la finalidad de estar sincronizadas para llevar a cabo la operación que se está esperando, convirtiéndose así en los problemas de prestaciones más frecuentes. Éstas primitivas están directamente ligadas con el número de procesos involucrados dentro de la aplicación y dependiendo del sentido que tengan pueden ser pérdidas de tiempo en los procesos que están esperando respuesta desde el proceso raíz ó, en caso contrario se generaría al momento de la salida de la aplicación donde el proceso raíz habrá de esperar que las demás tareas hayan finalizado.

1. *Blocked at Barrier*: en este caso la primera tarea en llegar se encuentra bloqueada hasta que hayan arribado todo el conjunto o grupo de tareas que deben acompañarla en una siguiente tarea (ver figura 2.6). Suele presentarse cuando todo el grupo de tareas habrán de iniciar una operación al mismo tiempo de forma que hasta que no estén todas en el mismo punto, el proceso no continuará, esta situación se ve directamente influenciada por desbalanceos entre la carga de trabajo, la existencia de entornos heterogéneos o problemas de eficiencia anteriores que afectaban solo algunas tareas.

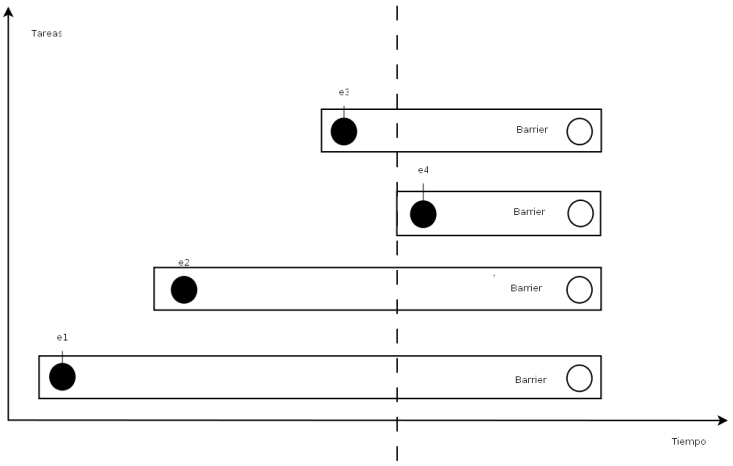

Figura 2.6: Bloqueado por Barrier

2. *Colectivas 1 a N*: se presenta cuando desde la tarea que ejerce el papel de raíz, ha enviado un mensaje al resto de tareas y se encontrarán bloqueadas aquéllas que hayan terminado antes y estén solo a la espera de la recepción de este mensaje (ver figura 2.7).

3. *Colectivas N a 1*: suele presentarse cuando el proceso que figura como raíz ha finalizado todas sus tareas antes de que le lleguen los mensajes que espera de las otras tareas (ver figura 2.8); es decir, al momento en que ha terminado su tarea anterior aún no está disponible más información para él.

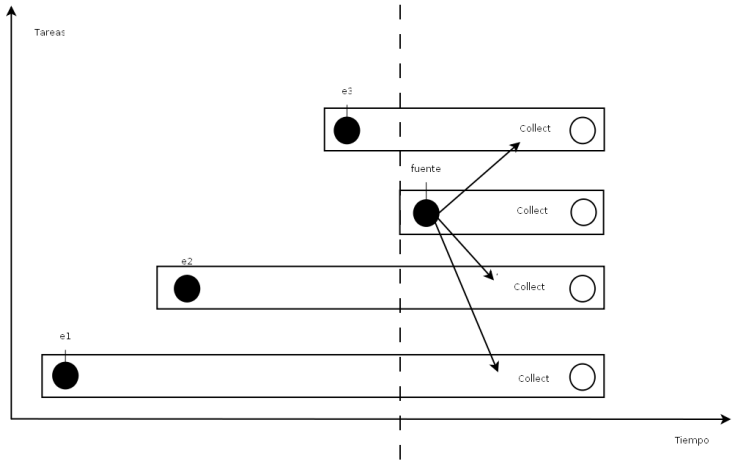

Figura 2.7: Colectivas 1 a N

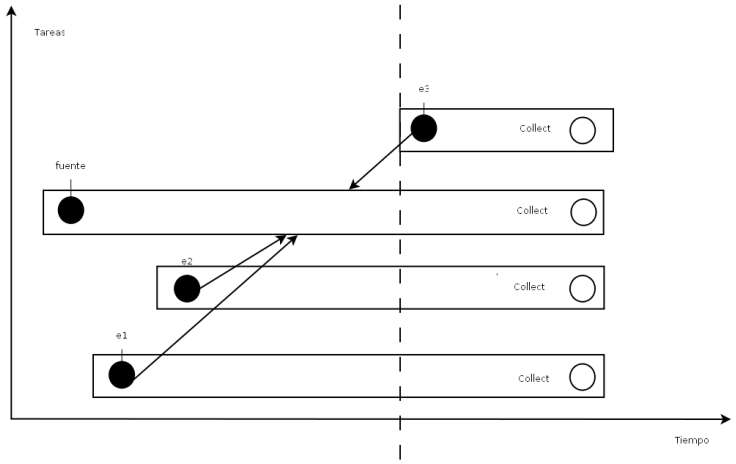

Figura 2.8: Colectivas N a 1

4. *Colectivas N a N*: este tipo de bloqueo se presenta cuando cada una de las tareas cumple a su vez tarea de recepción y envío de mensajes (ver figura 2.9). Suele estar directamente relacionada con las fases de repartición inicial, generación de los datos ó envío final a las tareas. Ocasionando que si alguna ha excedido el tiempo medio de recepción y envío, el tiempo global de la tarea se verá afectado y se bloquearán las demás tareas involucradas, y se podrá determinar como la suma total de los tiempos perdidos en cada tarea.

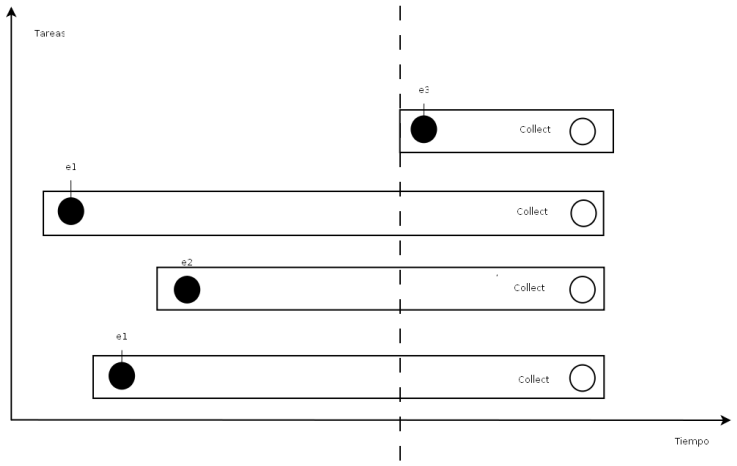

Figura 2.9: Colectivas N a N

Estructurales

Suele suceder que algunas etapas del desarrollo de la aplicación se comporten de forma similar a ciertos paradigmas de paralelismo, de esta forma se pueden prever posibles variaciones en el flujo de las tareas dentro del programa paralelo y predecir los problemas de ineficiencia presentes según un paradigma u otro. Dado que la aplicación paralela con la que estamos trabajando se encuentra diseñada bajo un paradigma máster/worker, se observan algunos problemas de ineficiencias ligados a la forma en que se comunican unas tareas con otras.

Máster/Worker: básicamente en una aplicación con este tipo de estructura, se realiza una fragmentación de datos por parte del máster, que son repartidos a los correspondientes trabajadores, éstos realizan el conjunto de tareas preestablecidas y luego devuelven los resultados al máster, pudiendo existir casos en los que éste proceso es llevado a cabo de forma iterativa. En el caso de éste tipo de paradigmas, suele tenerse en consideración los siguientes aspectos cuando de análisis de rendimiento se trata [20]:

1. Granularidad de las tareas: se refiere a la relación existente entre los bytes enviados y/o recibidos por una tarea y el cómputo realizado. Se puede realizar un estudio previo, con el modelo computacional y el ancho de banda disponible, para tener una idea de sobre la granularidad máxima posible.

2. Organización de las comunicaciones: está relacionado con la selección del número de mensajes de acuerdo a su longitud, ya que a mayor tamaño el tiempo de inicialización puede ser menor pero no garantiza que el tiempo global se reduzca, ya que al momento del manejo de la salida de los datos y procesamiento de los mensajes podría haber congestión en la red. De igual forma mensajes de un tamaño muy pequeño permiten solapar el cómputo con la comunicación; aunque normalmente el tamaño de los mensajes apropiado, y esta capacidad de solapamiento suelen estar estrechamente ligadas con la aplicación, convirtiéndose en factor de estudio obligado si se quiere optimizar.

3. Paradigma de Paralelismo: en modelos más complejos de máster/worker, el tipo de paralelismo empleado en la aplicación puede estar relacionado no solo con el particionado de los datos sino también con cierto paralelismo funcional. Si fuera este el caso las tareas de los workers podrían afectar directamente el rendimiento global de la aplicación según si realizan todos el mismo proceso o si cada uno varía en función de los datos que ha recibido. De igual forma si el máster se ve en la obligación de realizar algún otro proceso que no sea sólo la repartición de datos y posterior recolección, que frenaría algunos procesos de recepción de los trabajadores que ya están listos por estar finalizando alguna tarea previa.

Recordando lo comentado anteriormente, cuando se trata de entornos heterogéneos, además habrá de considerarse la configuración de las máquinas y la cantidad de carga que pudieran tener. A medida que se analiza una aplicación máster/worker visualizando los eventos se puede determinar la cantidad de carga generada, las comunicaciones existentes, y definir los posibles desbalanceos o retardos que existen, esto con la intención de determinar el número de workers apropiado para la carga de trabajo y reducir los posibles retrasos existentes durante la ejecución.

Una vez definidos el conjunto de factores que están directamente relacionados con éste tipo de estructura, se pueden considerar problemas de ineficiencia situaciones tales como:

- Desbalanceo de Carga, suele suceder cuando el trabajo no está uniformemente distribuído entre los workers desocupados disponibles. Problema que suele deberse a variaciones en las tareas (diferentes códigos o datos) con las que debe trabajar a cada worker.

- Número inapropiado de workers, si son muchos pueden generar una tasa de cómputo muy baja pero muy alta en cuanto a comunicación, o si son muy pocos generando largos períodos de inactividad por parte del máster.

- Retardos en las comunicaciones, debido a problemas previos de balanceo, que pudieran haberse presentado, o a la carencia de uniformidad en la repartición de las tareas. No dejando de lado causas como el ancho de banda, latencia, o contención de los mensajes.

- Tasa inapropiada de relación entre el tamaño de los mensajes y el número de veces que se envían.

2.3. Métodos de Análisis de Rendimiento

Con el paso de los años y gracias al desarrollo de nuevas herramientas y lenguajes para programación paralela, los enfoques de análisis de rendimiento han ido actualizándose y adaptándose más al comportamiento real de la aplicación.

Se ha pasado desde una visualización *post-mortem* de las ejecuciones, que solo permitía una vez finalizada la ejecución sacar conclusiones basadas en los llamados que se efectuaban de una función a otra, a la fase en que existen herramientas capaces de monitorizar la aplicación a medida que ésta se está ejecutando. Permitiendo introducir modificaciones que mejoren el rendimiento de la aplicación sin necesidad de detenerla, conocido como análisis de rendimiento dinámico.

Análisis de Rendimiento Clásico

La aproximación clásica de análisis de rendimiento, ha estado basado en la visualización de las ejecuciones con la ayuda de algunas herramientas. Una vez que las aplicaciones han sido diseñadas y depuradas, son ejecutadas con la ayuda de herramientas de monitorización que recolectan la información del comportamiento de la aplicación. Usualmente, una vez completada esta fase, la herramienta de visualización muestra la información que ha conseguido recolectar empleando vistas variadas (diagramas de Gantt, histogramas, diagramas de barras, entre otros...). Se cuenta con diferentes herramientas de visualización que difieren en interfaz de interacción con el usuario o el tipo de datos que muestran, pero sirven para dar una impresión rápida del rendimiento de la aplicación y sus posibles problemas principales. La forma más práctica de conocer si una aplicación no está sacando provecho de su diseño o del entorno paralelo sobre el que está siendo ejecutada, se puede monitorizar o instrumentar con la finalidad de obtener éstos datos. Sin embargo, éstas herramientas de monitorización pueden ser de diferentes tipos [20]:

- Temporales: aquélla que está relacionada con el tiempo de ejecución para identificar donde está invirtiendo más tiempo la aplicación paralela.
- Contadores: empleado para contar la cantidad de veces que se ejecuta una determinada función o evento.
- Muestreo: empleando ésta técnica, se obtienen ciertas mediciones periódicas del estado de la aplicación. Suele realizarse deteniendo la aplicación en puntos definidos para tomar la medida.
- Trazas: son secuencias de información asociadas a eventos determinados que se suceden dentro de las aplicaciones paralelas.

A partir de las medidas tomadas se puede obtener un resumen de la información asociada con el rendimiento obtenido en la ejecución de la aplicación paralela sobre un entorno de

cómputo paralelo, este puede estar presentado tanto textual como gráficamente. En la aproximación clásica de la visualización de dichos datos, se pasa por una fase de recolección de datos, transformación de los datos y posterior visualización. Los cuales se suelen insertar dentro de un proceso cíclico por parte del desarrollador en el que se persigue el mejor rendimiento posible para la aplicación. Con las herramientas de visualización se espera identificar los problemas de la aplicación relacionados con el rendimiento de las tareas y sus comunicaciones, la estructura de los algoritmos empleados y su implementación. En general una herramienta de visualización permite al desarrollador comparar fácilmente el patrón de ejecución observado en su aplicación con aquél que espera sea el ideal. Una visión global de la interacción existente entre las fases dentro del análisis de rendimiento clásico, se puede observar en la figura 2.10.

La hipótesis de rendimiento se enriquece y sustenta con los resultados obtenidos de cada uno de los experimentos de rendimiento realizados, permitiendo así generar una solución basada en la base de conocimientos de rendimientos para realizar los ajustes o las posibles soluciones para resolver las ineficiencias que se estuvieran presentando.

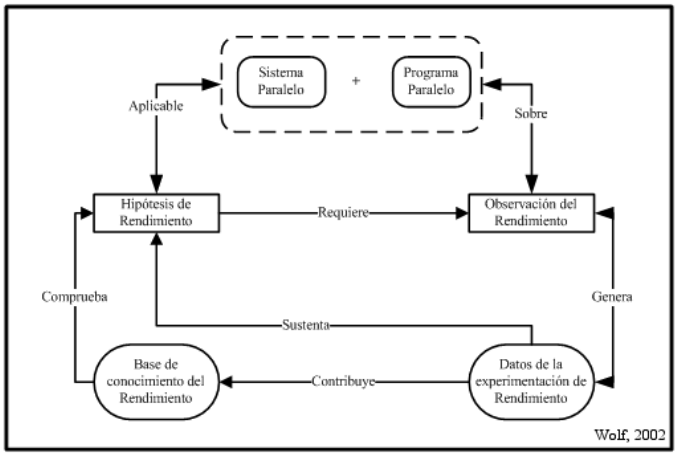

Figura 2.10: Análisis Cásico de Rendimiento

Entre los diagramas empleados para visualizar la información para post-procesamiento se utilizan los siguientes:

- *Grafos de comunicación.* También conocidos como diagramas de Gantt, permiten visualizar el estado de la ejecución, la interacción de las tareas y sus comunicaciones dentro de un intervalo de tiempo determinado. Estos estados y/o tareas son representados con el uso de colores.
- *Resúmenes estadísticos por tarea.* Mostrados con la ayuda de gráfico de torta (tarta),

en los que se representa el porcentaje de tiempo invertido en cada una de las tareas involucradas en la aplicación.

- *Resúmenes estdísticos por comunicación.* Suelen visualizarse como matrices de N x N tareas, donde cada celda representa la comunicación entre dos tareas a las que hace referencia, su unidad de medida acostumbra ser en bytes (Kbytes o Mbytes) enviados y recibidos. A veces, emplean colores para las celdas según los tamaños y con la intención de facilitar la visualización de las comunicaciones con gran volumen de datos. Dado que principalmente son de naturaleza acumulativa, no suelen ser demasiado útiles cuando hay eventos de naturaleza iterativa, porque no será fácil identificar los problemas o ineficiencias que hayan estado presentes.

- *Árboles de llamada.* Consiste en un diagrama de árbol que muestra las llamadas que han sido efectuadas entre las diferentes funciones dentro de la aplicación, puede representar desde objetos hasta subrutinas, o incluso llegar al nivel de bucles y condicionales. Es muy práctico para depurar el algoritmo al ver si la ejecución refleja el planteamiento inicial del desarrollador.

Análisis de Rendimiento Automático

Cuando se desea desligar a los desarrolladores de las tareas relacionadas con análisis de información gráfica o la defición de problemas de rendimiento, suele plantearse un análisis de rendimiento para la aplicación de forma automática. Este proceso requiere un modelo de rendimiento de los resultados esperados de ese proceso[38]. Es decir, "un proceso de examinación sistemática del rendimiento basándose en datos recogidos de una aplicación, con la finalidad de identificar propiedades de rendimiento relacionadas con regiones del código fuente de la aplicación".

Las herramientas empleadas para este tipo de análisis suelen estar basadas en dos principios. Primero, usan un conjunto de mediciones tomadas a partir de ejecuciones instrumentadas de la aplicación y provistas por herramientas de monitorización. Segundo, se basan en el conocimiento de los problemas de rendimiento almacenados en una base de conocimiento [20].

La naturaleza de los datos de rendimiento sin procesar viene dado por las técnicas de monitorización más comunes, las cuales usualmente entregan los datos en formas de perfiles o trazas de enventos. Éstas herramientas de análisis de rendimiento clásico soportan principalmente el proceso de búsqueda proveyendo vistas de bajo nivel de estos tipos de datos, vista que, usualmente incluyen visualización textual o gráfica (incluso a veces interactiva) como tablas o gráficos de barras de información de *profile*, lineas de tiempo, diagramas de trazas de eventos y análisis estadísticos.

Una vez que las mediciones han sido realizadas, el proceso de análisis automático puede ser llevado a cabo, donde la principal interrogante es cómo detectar los puntos de ineficiencia o *bottlenecks*. El trabajo con aplicaciones paralelas ha permitido identificar que muchas de ellas

tienen problemas de rendimiento bien definidos. Para buscar un problema de rendimiento, la herramienta debe estar soportada con la información acerca de cuales serían los posibles puntos de ineficiencia y cómo reconocerlos. Estos cuellos de botella se pueden hacer llegar a la aplicación como una base de conocimiento, conteniendo los modelos de rendimiento que proveen una via para entender los problemas de rendimiento. Usualmente se emplea un buen modelo analítico que facilitará la predicción de los cuellos de botella junto con sus causas y posibles optimizaciones, cabe destacar que cuando se desea crear un modelo de rendimiento, simplicidad y precisión no siempre van de la mano.

En este enfoque la búsqueda de cuellos de botella está aún basada en archivos de trazas generadas una vez que ha finalizado la ejecución, siendo conocido como análisis estático *post-mortem*. Las visualizaciones han sido reemplazadas por un análisis automático y la emisión de recomendaciones directas sobre los problemas detectados, es por ello que las herramientas basadas en este enfoque reducen significativamente la cantidad de tiempo invertida por los desarrolladores en análisis de rendimiento, ya que están soportados por la automatización, y recepción de datos más precisos, dentro del proceso global de sintonización. Las observaciones generadas por la herramienta suelen ser pistas muy útiles para conocer el comportamiento de la aplicación, sin embargo dado que son conclusiones obtenidas de una ejecución que ya ha finalizado, para una siguiente ejecución y dadas las características cambiantes de algunas aplicaciones puede que lo que ha funcionado para una ejecución anterior, no funcione para la siguiente, estableciéndose así que estas herramientas y este tipo de análisis es más adecuado para aplicaciones estables con datos de entrada similares.

Análisis de Rendimiento Dinámico

Aún cuando el desarrollador se ha visto beneficiado con la inserción de los procesos de análisis automáticos durante la ejecución sin sintonización, aún continuaba siendo necesario realizar manualmente los pasos de sintonización de la aplicación y viéndose involucrados muchos de los aspectos relacionados con el análisis clásico, tales como: la necesidad de instrumentar la aplicación, análisis basado en ficheros de trazas, realizar solo una ejecución a la vez en un entorno dado y un cierto comportamiento estable requerido [25]. Por ello, el análisis automático fue llevado de un punto de vista estático a uno más *dinámico*, donde el análisis de rendimiento es llevado a cabo durante la ejecución de la aplicación de forma totalmente automática para evitar la necesidad de realizar alguna instrumentación manual, sustituyendo un análisis *post-mortem* por uno en *tiempo real*; esto implica la necesidad de una monitorización constante, donde la principal ventaja es que ya no es necesario continuar usando ficheros de trazas para efectuar el análisis.

Este enfoque permite el control de la cantidad de instrumentación insertada en la aplicación con la ayuda de técnicas de instrumentación dinámica. La monitorización puede iniciar con una instrumentación muy sencilla y cuando ciertas condiciones particulares sean detectadas, sea instroducida instrumentación adicional. Cuando las condiciones desaparezcan, es posible

eliminar esa instrumentación extra. Bajo este enfoque el análisis debe ser realizado durante la ejecución de la aplicación, lo que conlleva a introducir cierto *overhead* dentro de ella. Por ello, el análisis debe ser relativamente simple para introducir la menor cantidad de overhead posible.

El análisis de rendimiento dinámico es el mejor acoplado a aquéllos programas que son iterativos y puede soportar aplicaciones con tiempos de ejecución muy altos y grandes volúmenes de datos. De igual forma, para resolver los problemas detectados, es necesario detener la aplicación, modificar, recompilar y ejecutarla de nuevo.

Cuando el análisis de rendimiento está basado en un enfoque dinámico no es necesaria la intervención del desarrollador ni el acceso al código fuente de la aplicación. La ejecución de la aplicación paralela es monitoreada, analizada y sintonizada automáticamente en tiempo de ejecución, sin la necesidad de recompilar y reiniciar. El comportamiento que esté presentando la aplicación será considerado y analizado buscando los cuellos de botella existentes y sus posibles optimizaciones.

Aún así, si la aplicación tiene un comportamiento diferente en diferentes ejecuciones, la sintonización dinámica lo adapta tomando en cuenta cambios en el comportamiento de la ejecución actual y cuando se ejecuta la aplicación bajo un entorno de análisis dinámico permitirá que el comportamiento de la aplicación se adapte también a las condiciones cambiantes dentro del entorno paralelo.

La sintonización dinámica debe proveer los siguientes servicios cooperando entre ellos en tiempo de ejecución [25]:

- Monitorización dinámico de la ejecución de la aplicación paralela. Este servicio permite la recolección de los datos a partir de la ejecución de la aplicación. Puede estar basado en cualquiera de las técnicas de monitorización que hemos venido mencionando, tales como: medición de tiempo, *profiling*, o trazas de eventos. Aún así, dado que el objetivo es reducir la intervención del usuario, la instrumentación debe ser llevada a cabo de forma automática por el sistema. Este servicio libera al desarrollador de la instrumentación manual dentro del código y lo exime de invocar todas las fases de recompilado y ejecución de nuevo una vez que se ha modificado el código de la aplicación.

- Análisis de rendimiento automático en tiempo de ejecución. Este servicio analiza las mediciones que se vienen generando, encontrando los cuellos de botellas y dando soluciones sobre como sobreponerlos. Para encontrar los puntos de ineficiencia y determinar como mejorar el rendimiento, el análisis debe tener incorporada una base de conocimiento de rendimiento acerca de los cuellos de botella que son representativos para la aplicación paralela. Para que sea útil debe incluir señalización de los problemas detectados y sugerencias para el usuario, a través de una interfaz gráfica o impresión en pantalla.

- Sintonización automática del programa en tiempo de ejecución. Permite usar las sugerencias generadas en el proceso de análisis para modificar automáticamente la aplica-

ción paralela en tiempo de ejecución. Evitando así por completo el acceso al código o la necesidad de recompilación y reinicio de la aplicación. La sintonización dinámica libera a desarrollador de tener que modificar el código dado que se modifica automáticamente a medida que se ejecuta.

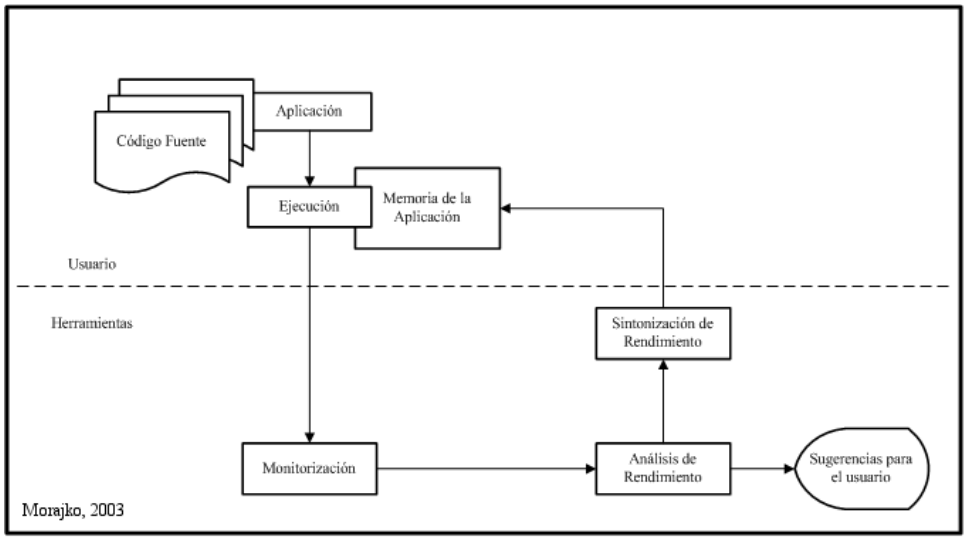

Figura 2.11: Análisis y Sintonización Dinámica

Lo que se observa gráficamente en la figura 2.11 es que todo el proceso de monitorización, análisis y sintonización está sucediendo fuera de la vista del usuario de forma que no sea necesaria su intervención para realizar las modificaciones necesarias en la aplicación.

2.4. Herramientas de análisis de rendimiento

Inicialmente las principales herramientas existentes para análisis de rendimiento se encuentran distribuídas entre los diferentes enfoques antes explicados en la sección 2.3 y emplean diferentes técnicas de monitorización, algunas de ellas como Jumpshot[8], PARAVER[30] y VAMPIR [26] emplean visualizaciín por trazas, en cambio otras como TAU [32] o Paradyn [24] están basados en *profiling*. Éstas y otras herramientas como MATE [25], GMATE [10], KOJAK [37] y SCALASCA[15] están disponibles actualmente para llevar a cabo el proceso de análisis de rendimiento y serán descritas a lo largo de esta sección.

Jumpshot: Jumpshot es una herramienta de visualización de análisis de rendimiento postmortem basada en Java. Empleando Java aumenta la portabilidad, mantenibilidad y las funcionalidades de las herramientas. La última versión disponible es Jumpshot-4, y presenta un rediseño total de la herramienta gráfica para SLOG-2. El nuevo formato de archivo de log es escalable y permite al visualizador detallar el archivo a cualquier nivel de acercamiento. Adicionalmente, posee nuevas funcionalidades de acercamiento y alejamiento,desplazamiento, expansión vertical de la línea de tiempo así como manipulación de la línea de tiempo disponible tanto en el módulo

de histograma como el de línea de tiempo. Posee una nueva tabla de leyendas que provee un control central para ambos módulos y permite que la manipulación de los gráficos sea mucho más fácil. El nuevo visualizador posee un convertidor de archivo de log integrado que permite convertir otros formatos de traza como CLOG, CLOG-2, RLOG y UTE.

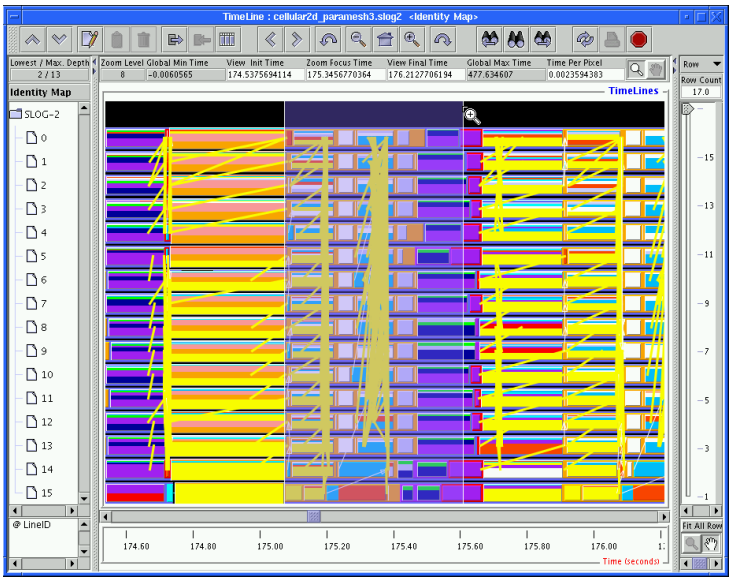

Figura 2.12: Visualización de ejecución con Jumpshot.

Paraver: se trata de un visualizador de trazas de eventos diseñado en la Universitat Politècnica de Catalunya (UPC) que soporta una gran gama de sistemas de programación distribuídos sobre plataforma UNIX. Visualiza trazas en líneas de tiempo, de eventos producidos en llamadas de sistema, junto con contadores de hardware y llamadas a funciones. Una característica particular de este sistema es la capacidad de definir las métricas que el usuario desea visualizar, a través de instrucciones u operaciones realizadas con la herramienta. Es empleada como visualizador de una herramienta denominada *Dimemas*, que es empleada para análisis de prestaciones mediante simulación de la aplicación.

VAMPIR (Visualization and Analysis of MPI Resources): es una herramienta que permite generar trazas de ejecución de aplicaciones paralelas con MPI, ha sido desarrollado por la empresa Pallas y se basa en la generación de trazas con la ayuda de contadores de tiempo, de subrutinas y/o bloques de código. Cuenta con múltiples vistas gráficas y generación de estadísticas sobre el tiempo empleado en cada función de la aplicación, referentes a MPI y otras; así como controlar el flujo y tamanño de los mensajes. Emplea la librería VAMPIRTrace para realizar la monitorización de todas las comunicaciones y permite la visualización de las métricas en gráficas de línea de tiempo, gráficas de barras, tarta e histogramas, tanto de forma global como también para cada uno de los procesos. Las salidas generadas por VAMPIR tienen un aspecto como el de la figura 2.13.

KOJAK: es un conjunto de herramientas diseñados para el análisis de rendimiento automático de aplicaciones paralelas en MPI, OpenMP, y aplicaciones híbridas desarrolladas en C, C++

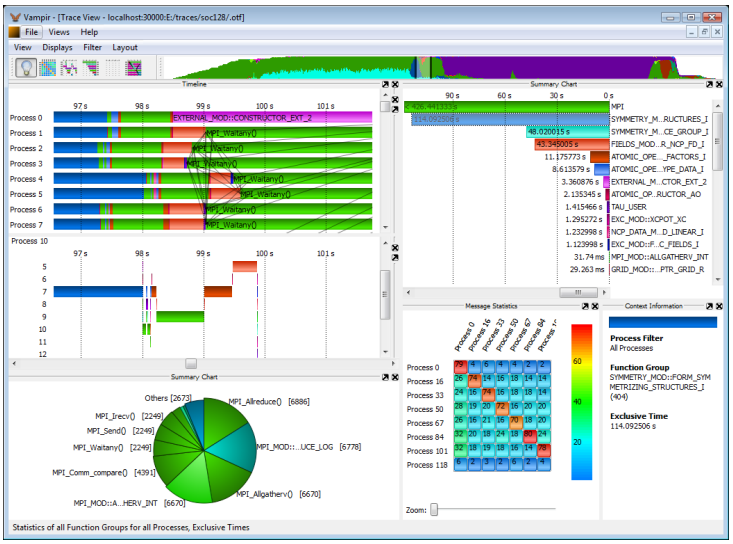

Figura 2.13: Salida de VAMPIR.

y FORTRAN; KOJAK genera trazas de eventos para aplicaciones en ejecución y luego determina las posibles ineficiencias existentes dentro de los patrones de ejecución. KOJAK ha sido un proyecto conjunto por Forschungszentrum Jülich en Alemania y la Universidad de Tenesse, Estados Unidos [37]. Cuenta con un módulo de generación de trazas (EPILOG) y un módulo de instrumentación automática empleando TAU [32]. Una vez que se ha completado la fase de postprocesado, la traza global es sujeta a un análisis en frío realizado por el componente EXPERT, el cual intenta identificar propiedades de rendimiento específicas. Internamente, EXPERT representa las propiedades de rendimento en la forma de patrones de ejecución que modelan un comportamiento ineficiente. Estos patrones son usados durante el proceso de análisis para reconocer, clasificar y cuantificar el comportamiento ineficiente en la aplicación. El proceso de análisis de forma automática transforma las trazas en rutas de llamadas compactas que incluyen las penalizaciones de tiempo ocasionadas por los diferentes patrones y que pueden ser visualizadas empleando herramientas como CUBE, tal como se muestra en la figura 2.14.

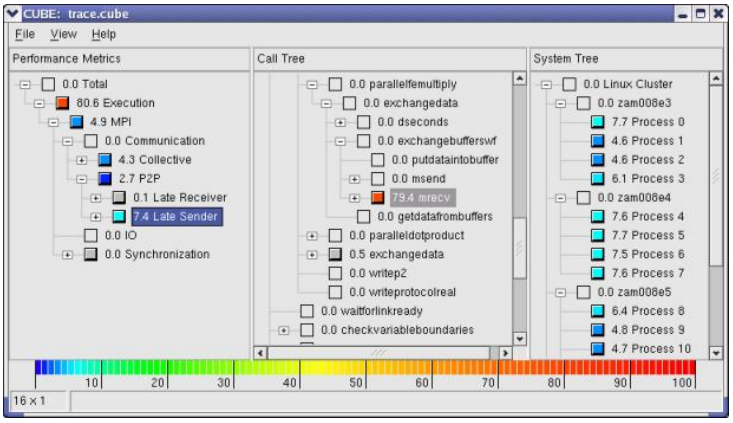

Figura 2.14: Visualización en Cube de un profile de KOJAK

DiMeMas es una herramienta de simulación de ejecuciones [4] que estuvo disponible comercialmente a través de PALLAS, y no tiene registro de desarrollo activo desde 2005, sin embargo es usada continuamente en el Barcelona Supercomputer Center (BSC). DiMeMas suaviza el uso de simulaciones basadas en instrucciones a través de la repetición de trazas obtenidas durante la ejecución de la aplicación. Soporta la evaluación bajo diferentes arquitecturas de máquinas paralelas a través de la regeneración de trazas. Este enfoque ha demostrado un uso exitoso en máquinas de más de 1000 elementos de procesamiento. El basarse en trazas evita tener que tocar el código y permite identificar los puntos necesarios para realizar la sintonización manual o generar los cambios necesarios en el modelo de rendimiento, dado que el código está contenido implícitamente dentro de la traza en vez de ser capturado explícitamente en un modelo abstracto editable por el usuario. Editar una estructura tan compleja no es trivial y la creación inicial requiere que el código sea escrito y ejecutado. Las trazas empleadas ocupan cierto espacio considerable en disco y memoria del sistema, generando límites severos en el tamaño máximo del modelo que puede ser procesado en una sóla máquina.

DiMeMas es una herramienta de simulación para el análisis paramétrico del comportamiento de aplicaciones MPI, desarrollada por el Centro Europeo de Paralelismo de Barcelona (CEPBA) de la Universitat Politècnica de Catalunya. Permite al usuario desarrollar y sintonizar aplicaciones paralelas, mientras genera una predicción bastante precisa de su comportamiento en una máquina paralela determinada. Habiendo leído los registros desde el archivo de trazas y especificado los parámetros de la arquitectura, Dimemas puede reconstruir el comportamiento de una aplicación paralela a través del tiempo. La herramienta simula la ejecución de la aplicación, escalando el tiempo empleado en cada bloque de acuerdo a la velocidad de la unidad central de proceso objetivo. Dimemas genera como salida archivos de trazas que son acoplables a diferentes herramientas como Paraver o Vampir. Los resultados incluyen información global de la aplicación: tiempo de ejecución y speedup. Adicionalmente, para cada proceso entrega la información sobre el tiempo de ejecución, el tiempo en que permanece bloqueado, el de cómputo, el número de mensajes enviados y recibidos, volumen de comunicación de datos. Aún más, Dimemas está diseñado para determinar la ruta crítica que devuelve el camino más largo de comunicación de la aplicación.

TAU (Tuning and Analysis Utilities): el sistema de análisis de rendimiento paraleo TAU, es el producto de 14 años de desarrollo para crear un framework robusto, flexible, portable e integrado y un conjunto de herramientas para instrumentación, medición, análisis y visualización de sistemas y aplicaciones paralelos de gran tamaño [32]. El éxito de este proyecto representa un trabajo conjunto entre los investigadores y colegas de la Universidad de Oregon y el Centro de Investigación de J*ü*lich junto con el Los Alamos National Laboratory. El sistema de rendimiento TAU considera los problemas de rendimiento desde tres niveles: instrumentación, medición, y análisis. TAU soporta la configuración e integración de estas tres capas para conseguir resolver problemas de rendimiento puntuales. De igual forma, la exploración efectiva de rendimiento requiere seleccionar prudentemente de un conjunto de metodos alternativos. TAU permite la

combinación de experimentos de rendimiento significativos que facilitan la obtención de propiedades de rendimiento relevantes. Para lograr esto, TAU ofrece soporte para realizar análisis de rendimiento de diferentes maneras, incluyendo la capacidad de realizar un proceso de instrumentación multinivel bastante robusto, modalidades de medición empleando trazado y profiling, análisis de rendimiento interactivo y gestión de datos de rendimiento, los datos obtenidos a través del profiling pueden visualizarse en diferentes gráficos estadísticos, algunos de ellos como el histograma mostrado en la figura 2.15.

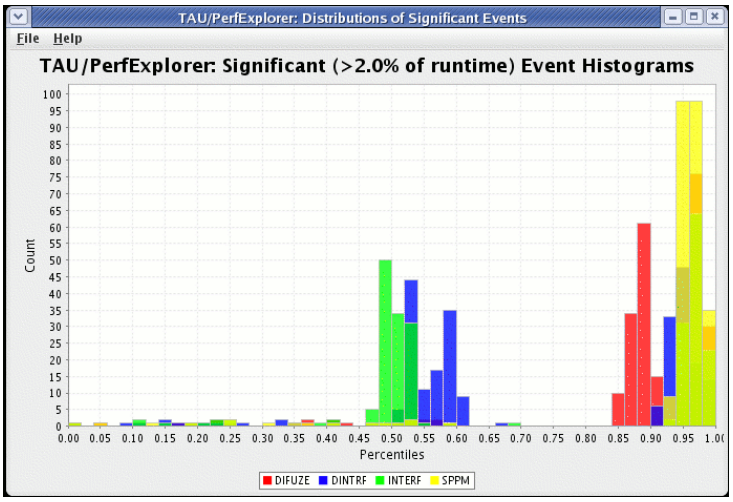

Figura 2.15: Histograma de eventos generado por TAU

Paradyn: es una herramienta desarrollada en la Universidad de Wisconsin y Maryland que monitoriza el rendimiento de aplicaciones paralelas[24]. Es capaz de monitorizar y realizar análisis automático en tiempo de ejecución. Emplea técnicas de profiling dinámicas a través de la instrumentación realizada con la ayuda de la librería DyninstAPI (http://www.dyninst.org/) que evita tener que modificar el código fuente de la aplicación, que inserta pequeños trozos de código *(snippets)* en la entrada y salida de las funciones. Paradyn facilita el análisis automático de las aplicaciones que están siendo ejecutada, procurando indentificar aquéllas partes dentro de la misma que consumen mayores períodos de tiempo y recursos, está basado en un modelo de identificación de cuello de botellas denominado *W3* (Why, Where, When; Por qué, donde y cuándo), con los que se espera determinar por qué la aplicación está presentando un rendimiento bajo, en que parte de los recursos está ocurriendo el cuello de botella (CPU, comunicaciones, E/S) y cuándo sucede.

El objetivo que esta herramienta persigue es aislar de la forma más rápida y precisa posible el problema sin tener que examinar demasiada información; además cuenta con un módulo denominado *consultor de rendimiento* que libera al desarrollador de tomar decisiones sobre el control de los datos dentro de la aplicación. Éste consultor busca cuales son los problemas, decide qué datos necesita almacenar y cuándo debe aplicar la sintonización, todo esto en tiempo de ejecución. Mientras se realiza el proceso se informa al desarrollador los cambios que han sido

llevados a cabo. Adicionalmente cuenta con herramientas de visualización llamadas *Performance Visualizations*, que informan el rendimiento de la aplicación y el trabajo realizado por Paradyn en tiempo de ejecución, visualizando los datos que se han estado obteniendo a medida que la ejecución del programa avanza.

SCALASCA: La versión actual de SCALASCA[15] soporta medición y análisis para aplicaciones de plataformas HPC que emplean MPI, OpenMP y programación híbrida, escritas tanto en C/C++ como Fortran. Antes de que cualquier dato de rendimiento sea recolectado, la aplicación que se va a estudiar debe ser instrumentada.

Cuando se ejecuta la aplicación sobre la máquina paralela, el usuario puede generar un reporte condensado (*profile)* con métricas agregadas para rutas de llamadas de funciones individuales (ver figura 2.16, y/o trazas de eventos que almacenan aquéllos que han sucedido en tiempo de ejecución, cuyo perfil o visualización de línea de tiempo será generada más adelante.

Dado que las trazas tienden a hacerse demasiado grandes en poco tiempo, puntuar el reporte condensado es más recomendable. Cuando el traceado está activo, cada proceso genera un archivo de traza que contiene los eventos para cada proceso local. Una vez que la aplicación ha finalizado, SCALASCA carga los archivos de trazas en memoria principal y las analiza en paralelo empleando tantas unidades de proceso (CPUs) como procesos hayan sido empleados dentro de la aplicación misma.

Durante el análisis, SCALASCA busca por patrones característicos que indiquen estados de espera o inactividad que estén relacionados con las propiedades de rendimiento, clasifica las instancias detectadas por categorías y cuantifica su significancia. El resultado es un reporte de análisis de patrón similar en estructura al reporte condensado pero enriquecido con metricas de comunicación y sincronización de alto nivel.

Tanto el reporte condensado como el de patrones contienen métricas de rendimiento para cada una de las rutas de llamadas de las funciones y de los procesos y/o threads, que pueden ser examinados de forma interactiva en el reporte de análisis provisto o con un navegador de profile como el ParaProf de TAU. Adicionalmente, al análisis de trazas escalabe, es también posible ejecutar el análisis secuencial de KOJAK después de unir los archivos de traza locales.

El análisis secuencial ofrece características que no están aún disponibles en esta versión paralela, incluyendo los análisis extendidos de MPI y OpenMP, y la habilidad de generar trazas de instancias de patrones (también conocidas como trazas de propiedades de rendimiento). Como una alternativa para la búsqueda automática de patrones, las trazas unidas pueden ser convertidas e investigadas empleando navegadores externos como Paraver o Vampir, sacando partido de sus completas funcionalidades de visualización en líneas de tiempo y estadísticas.

MATE (Monitoring, Analysis and Tuning Environment): es una herramienta de monitorización, análisis de rendimiento y sintonización de aplicaciones paralelas de forma dinámica, ha sido diseñada en la Universitat Autònoma de Barcelona [25], ha basado el monitoreo dinámico en trazas de eventos (entrada y salida de funciones). Inicialmente estaba pensada para sintonizar aplicaciones PVM paralelas/distribuidas desarrolladas en C/C++ ejecutándose en plataformas

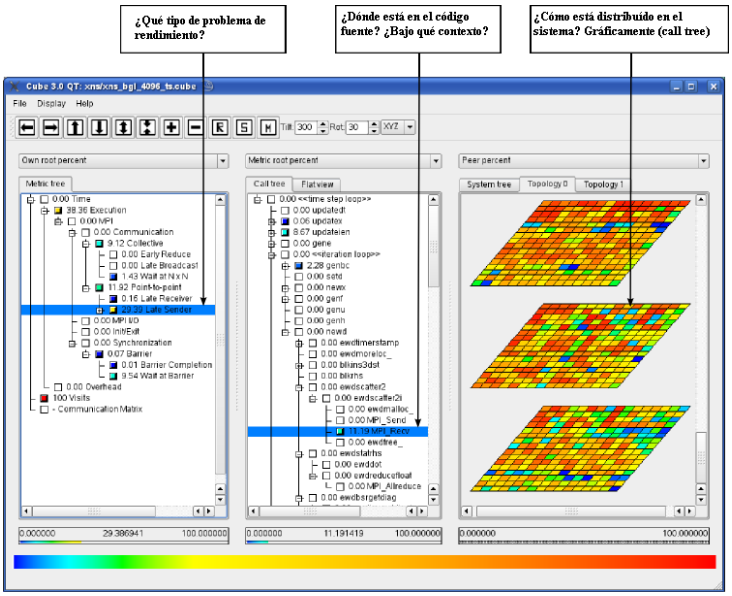

Figura 2.16: Análisis de Rendimiento con Scalasca

UNIX. MATE realiza la sintonización dinámica en tres fases básicas y contínuas: monitoreo, análisis de rendimiento y modificaciones. Este entorno dinámica y automáticamente instrumenta una aplicación en ejecución para reunir la información acerca del comportamiento de la aplicación. La fase de análisis recibe los eventos, busca e identifica los posibles cuellos de botella existentes, detecta que los ha causado y da soluciones sobre como corregirlos. Finalmente, la aplicación es sintonizada dinámicamente determinando, los puntos de medición, el modelo y las acciones a tomar, que permitirán la creación del *tunlet* específico.

MATE involucra ciertos componentes principales colaborando entre ellos para controlar y tratar de mejorar el rendimiento de la aplicación: *un controlador de la aplicación*, que vigila la ejecución de la aplicación en un huesped dado (gestión de tareas y máquinas), que provee la gestión de la tarea de instrumentación y modificación; *una librería de monitorización dinámica*, cargada por las tareas del controlador de la aplicación para facilitar la instrumentación y la recolección de los datos, esta librería contiene funciones que son responsables del registro de eventos con todos los atributos necesarios para ser analizados; y *un analizador*, que es un proceso que lleva a cabo el análisis de rendimeinto de la aplicación, automáticamente detecta los problemas de rendimiento en ejecución y solicita los cambios necesarios para mejorar el rendimiento de la aplicación.

GMATE (Grid Monitoring, Analysis and Tuning Environment): es una herramienta desarrollada en la Universitat Autònoma de Barcelona[10] basada en la herramienta MATE que ha sido descrita anteriormente, por lo que posee el enfoque de instrumentación, monitorización, análisis y sintonización de aplicaciones paralelas/distribuídas ejecutándose bajo entornos Grid; desde el punto de vista de la aplicación, la heterogeneidad es la característica que más influencia la ejecución y en entornos de Grid, la heterogeneidad existente entre las diferentes unidades de

cómputo y las comunicaciones involucradas es un hecho. GMATE consta de un *tunlet* desarrollado en Java que puede encapsular la lógica de lo que debe ser medido, como deben interpretarse los datos por el modelo de rendimento y que puede ser cambiado para obtener mejor tiempo de ejecución o mejor uso de recursos; cuenta con actuadores y sensores que están asociados a conjuntos de procesos o variables de sistema usadas como datos fuente.

En el caso de análisis de rendimiento de aplicaciones paralelas, la ordenación de los eventos y la sincronización son situaciones cruciales para la detección de los cuellos de botella. De modo que para proveer procesos de monitorización de eventos de sincronización dentro de la aplicación emplea dos enfoques, una herramienta de monitoreo y un sistema basados en sincronización del reloj.

Capítulo 3

Modelo de Prestaciones y Sintonización

"Investigar es ver lo que todo el mundo ha visto, y pensar lo que nadie más ha pensado"

Albert Szent-Györgi (1893-1986).

3.1. Introducción

En este capítulo se describe en términos generales la finalidad para la que se diseña un modelo analítico de prestaciones y las principales consideraciones que se deben tomar en cuenta. Realizar la medición para todos y cada uno de los posibles valores de entrada en un sistema o aplicación nos permite tener el conocimiento general de cómo será el comportamiento real de éste para un determinado conjunto de condiciones; sin embargo el consumo de tiempo y recursos que ésto significa es considerable (sino es que imposible) [22].

El rendimiento de aplicaciones paralelas en máquinas de *High Performance Computing* (HPC) se basa en factores como el algoritmo, implementación, compilador, sistema operativo, arquitectura del procesador y tecnologías de interconexión. Es por ello que, se puede concluir que los *modelos de rendimiento* para aplicaciones científicas en sistemas complejos debe estar acorde a todos los atributos de la aplicación y del sistema. Los métodos para evaluación de rendimiento pueden ser separados en dos áreas [19]: modelos estructurales y modelos funcionales analíticos.

Los modelos estructurales emplean descripciones de componentes individuales del sistema y sus interacciones, tales como los modelos detallados de simulación; mientras que los modelos analíticos y funcionales, separan los factores de rendimiento de un sistema para crear un modelo matemático.

Los simuladores detallados [19]son normalmente construídos por desarrolladores durante la fase de diseño de las arquitecturas para agregarlo al diseño. Para máquinas paralelas se suelen usar dos simuladores, uno para los procesadores y otros para la red de interconexión. Estos

simuladores tienen la ventaja de predecir el rendimiento de la aplicación de forma automática desde el punto de vista del usuario.

La desventaja es que la mayoria de estos simuladores suelen ser propietarios y no estar disponibles para usuarios y centros HPC. Además, dado que ellos capturan el comportamiento de la simulación de los procesadores pueden tomar hasta un millón de veces más tiempo que una ejecución real. Esto significa que para simular una hora de una aplicación, podría tomar aproximadamente 114 años de tiempo de CPU [19]. Los métodos de ejecución directos son empleados comúnmente para acelerar las simulaciones de arquitectura pero aún tendrían ralentización del tiempo de ejecución.

Para evitar estos inmensos costos computacionales, los simuladores de precisión por ciclos son usualmente sólo empleados para simular unos pocos segundos de una aplicación. Esto ocasiona un dilema de modelado, para la mayoría de las aplicaciones científicas el comportamiento completo de una ejecución no puede ser capturado en unos pocos segundos. Las aplicaciones raramente tardan todo su tiempo en una rutina y su comportamiento puede cambiar a medida que la aplicación evoluciona a lo largo de su simulación (en algunos casos el problema que se está resolviendo cambia físicamente). Estos simuladores de ciclo están limitados a sólo trabajar con el modelado del comportamiento del procesador para el cual han sido desarrollados, de forma que no son aplicables para otras arquitecturas.

En la segunda área de evaluación de rendimiento, los modelos analíticos y funcionales, el rendimiento de la aplicación en una máquina definida puede ser descrito en una ecuación matemática compleja.

Grupos como el de análisis de rendimiento de la Universitat Autónoma de Barcelona y del Jülich Supercomputer Centre han diseñado diferentes modelos para aplicaciones Máster/Worker [11], aplicaciones MPI [38], aplicaciones PVM [25] y ejecución de aplicaciones en entornos heterogéneos de grid [10].

La evaluación de rendimiento de sistemas y aplicaciones paralelas puede generar cantidades considerables de datos y hace necesario que el análisis de los resultados para tantos experimentos de rendimiento se convierta en uno de los problemas que están siendo investigados. Por ello la gestión de la información de rendimiento es el corazón de las herramientas de análisis de rendimiento.

Existe cierto contraste entre el rendimiento nominal de un sistema (el pico de desempeño posible) y el rendimiento actual de algunas aplicaciones paralelas; a medida que los sistemas paralelos crecen en tamaño y complejidad, este contraste se hace más y más importante, lo que justifica la búsqueda por herramientas y técnicas que permitan a los usuarios entender las fuentes donde se degrada el rendimiento, ya que una vez que se consigue comprender el rendimiento, es importante no sólo, aumentar la eficiencia de las aplicaciones, sino también plantear las modificaciones de acuerdo al sistema y al entorno de programación.

Existen dos estrategias básicas para llevar a cabo un análisis de rendimiento de aplicaciones paralelas, estos son: *medición* (y modificación) y *modelado*. El proceso de medición, se refiere a

pasos que se realizan de forma iterativa, en los que el usuario captura mediciones de rendimiento de una aplicación en tiempo de ejecución, identifica los cambios que son necesarios, modifica el programa y repite todo el proceso de nuevo. En cambio, el modelado, filtra ciertos parámetros del sistema paralelo en forma de abstracciones que son expresadas tanto como componentes del sistema como la interacción entre dichos componentes.

El enfoque de tomar mediciones y realizar modificaciones es insuficiente para entender la razón detras de los cuellos de botella de rendimiento porque puede ser un poco más difícil conectar el diseño con los problemas de rendimiento. Esta limitación es más evidente cuando los problemas de rendimiento son influenciados por factores externos tales como: las condiciones el entorno y los parámetros de ejecución, lo que genera que la conexión entre la implementación y los cuellos de botella existentes sea aún más complicado de detectar. Aún peor, cuando se realice alguna modificación en el programa para solucionar alguno de los problemas de rendimiento, puede generar nuevos problemas ocasionando un aún peor rendimiento.

Los modelos proveen una forma más estructurada de comprender los problemas de rendimiento, ya que toman en cuenta la influencia de factores externos y relacionan las fuentes de degradación del rendimiento con las características del sistema de rendimiento.

Adicionalmente, los modelos presentan una habilidad de predicción aún mayor, permitiendo al usuario estimar los efectos en el rendimiento ocasionados por diferentes paralelizaciones o variaciones en el entorno de ejecución. Esta habilidad es importante porque guía al usuario a encontrar un diseño "óptimo" del sistema, más allá de una búsqueda repetitiva de problemas entre grandes cantidades de datos.

La tarea de composición de modelos no es de las más sencillas que existen, dado que las abstracciones de los sistemas paralelos pueden ser difíciles de especificar debido a que no todas son lo suficientemente sencillas para ser manejadas, y lo suficientemente detalladas para que sean precisas.

3.2. Modelo de Prestaciones

El modelado de rendimiento, tal como se ha comentado anteriormente, permite observar en un nivel de abstracción mayor la interacción que existe entre los diferentes elementos que conforman un sistema o aplicación paralela, permitiendo hasta cierto punto generar una cierta predicción del comportamiento que esta presentará cuando se ha variado uno u otro parámetro. Sin duda alguna, el diseño de modelos analíticos no ocasiona intrusión alguna dentro del comportamiento de la aplicación como pudiera generarlo algún tipo de instrumentación; sin embargo, este diseño requiere especial atención en los parámetros involucrados y al comportamiento de la aplicación como tal.

Existen varios enfoques de modelado de rendimiento, donde uno de los primeros han sido el *bottom-up* y el *top-down*, que serán descritos a continuación:

- *Bottom-up*: los sistemas son descritos a través de conjuntos de parámetros o funciones

que abstraen el software, el hardware y las interacciones entre ellos. El significado y la importancia de éstos parámetros diferencia cada uno de ellos (HW de SW, por ejemplo). Las técnicas de modelado analítico y sus extensiones son ejemplos de modelado bottom-up.

- *Top-down*: El punto de inicio para el modelado top-down es la estructura de la aplicación. Los parámetros de los modelos se obtienen de las características de la aplicación, y de su interacción entre el hardware y el software derivadas de la aplicación. Los modelos basados en descripción y el análisis estático son ejemplos de éste tipo de modelado.

En general, el modelado bottom-up está mejor adaptado a la investigación de factores de arquitectura o del entorno, y el software se define a través de parámetros. El modelado Top-down se enfoca en la paralelización de la aplicación, el estilo de particionado de datos, y otros aspectos del software y por tanto está mejor ajustado al estudio de los problemas dentro de la aplicación.

3.2.1. Modelado Analítico

La técnica de modelado analítico abstrae las características de un sistema paralelo como un conjunto de parámetros o funciones parametrizadas con la intención de hacer la tarea de modelado más fácil. Esta técnica ha sido ampliamente utilizada en el desarrollo y modelado de sistemas paralelos. Como ventajas de emplear este tipo de modelado, se encuentra que es usualmente muy económico y provee una visión abstracta del hardware y del software. Aún así, los modelos son pocas veces lo suficientemente precisos cuando se compara con las ejecuciones reales debido a las simplificaciones en las que se incurre a medida que se realiza el modelado. Según la literatura existen tres enfoques que difieren en la forma en que los modelos son expresados, cada uno de los cuales serán descritos en la siguiente sección.

Cuando se trabaja con modelado analítico hay que tener en consideración algunos aspectos como el nivel de detalle del sistema que se está estudiando (o nivel de abstracción) que está relacionado con el modelo. Los parámetros por si mismos y su representación, determinan el nivel de abstracción. A medida que aumenta el número de parámetros disminuye el nivel de abstracción (más detalles en el modelo) y se incrementa el costo del modelo en tiempo de desarrollo. En cuanto a la representación, los parámetros escalares son más sencillos que las herramientas estadísticas y las funciones, pero menos flexibles. Las funciones requieren que se determine no solo sus coeficientes, sino también su forma. Las herramientas estadísticas requieren un conocimiento especializado que los programadores en promedio no poseen.

Otra consideración que hay que tomar en cuenta cuando se diseña el modelo analítico, es el grado de dificultad para determinar los valores de los parámetros, los coeficientes de las funciones o las especificaciones de las herramientas estadísticas (parámetros de distribución de variables, por ejemplo). Si los parámetros son demasiado específicos, pueden ser difíciles de capturar debido a la carencia de herramientas adecuadas o conocimiento de la aplicación. Si el parámetro es de muy alto nivel, puede requerir la representación de comportamientos característicos del sistema.

Los modelos analíticos, suelen tener bajo costo computacional, a excepción de algunas herramientas estadísticas que requieren esfuerzos significativos de simulación.

Modelado Analítico con Parámetros Escalares

Este enfoque emplea un conjunto de parámetros escalares (número de workers por ejemplo) para modelar el comportamiento de un sistema paralelo. Estos parámetros expresan el comportamiento promedio de un sistema paralelo bajo condiciones específicas; los parámetros suelen variar en número e información expresada según la finalidad con la que se diseña el modelo.

Modelado Analítico con Funciones

Con la finalidad de expresar los efectos de los parámetros de hardware y software en el tiempo de ejecución, muchos investigadores han adoptado las funciones matemáticas en vez de valores escalares en sus modelos. Ya que el empleo de funciones aumenta la flexibilidad y el nivel de expresividad del modelo, pero también puede aumentar la complejidad de los mismos, debido a la necesidad de determinar la forma y los coeficientes de las funciones. Los modelos basados en parámetros escalares son un caso simplificado del modelado con funciones, donde las funciones son constantes.

Modelado Analítico Estadístico

Otro enfoque para el modelado analítico de rendimiento para sistemas paralelos está caracterizado por el uso de herramientas de modelado estadístico como distribución de variables y modelos de Markov. Éstos difieren del enfoque de parámetros escalares en el sentido que los "parámetros" ahora son herramientas estadísticas.

3.2.2. Descomposición del Modelado Analítico

La descomposición del modelado analítico es una extensión del enfoque analítico. En éste enfoque los parámetros y funciones son usados para caracterizar el sistema paralelo. La idea detrás de la descomposición es simplificar el proceso de modelado. Permitiendo el diseño de modelos modulares, donde varias piezas pueden ser actualizadas o reemplazadas independientemente unas de otras. Los sistemas paralelos pueden ser descompuestos de varias formas dependiendo de la finalidad del modelado.

En general, la descomposición simplifica el proceso de modelado porque divide el sistema paralelo en partes que son mas amenas y modelables. La principal diferencia entre las estrategias de descomposición es la forma en que el sistema es dividido y las implicaciones de la división.

La descomposición vertical y horizontal están limitadas por la interfaz entre las piezas. En la descomposición horizontal, las interfaces limitan el rango de la aplicación que puede ser tratado por el modelo. Mientras que en la descomposición vertical, el modelo para cada pieza no considera la influencia de las piezas que la preceden, lo cual puede llevar a una imprecisión considerable.

La descomposición por overhead no tiene limitación por interfaz, porque usualmente asume que las categorías son mutuamente exclusivas.

Cabe destacar también que la descomposición no aborda los problemas o limitaciones de las técnicas empleadas para modelar las piezas.

Descomposición Horizontal

Bajo el enfoque de descomposición horizontal, un sistema paralelo es dividido en varias capas (por ejemplo se podría dividir en: aplicación, entorno de ejecución, y capas de arquitectura), cada uno de los cuales es modelado individualmente. La complejidad y el número de capas depende de las necesidades de modelado; la interfaz entre las capas es determinada por el modelo.

Descomposición Vertical

Cuando se trata de descomposición vertical, las operaciones de alto nivel como las rutinas de comunicación o las operaciones con vectores, son las unidades de modelado. Este enfoque suele llamarse así porque el objetivo principal está sobre las operaciones en el nivel de aplicación, y los modelos capturan las características de los componentes del sistema paralelo en múltiples niveles. Una de las principales características de la descomposición vertical es el uso de modelos basado en operaciones para predecir el rendimiento de la aplicación. Sin embargo, el nivel de imprecisión en las predicciones es proporcional a las variaciones de comportamiento en una operación. Tales variaciones pueden ser ocasionadas por diferentes distribuciones de datos al momento en que la operación es activada.

Descomposición por fuente de Overhead

En general, podemos definir el overhead como un retraso en la ejecución que aparece debido a la paralelización y previene la utilización del potencial de paralelismo del sistema. Este overhead suele aparecer implícitamente como parámetros de la arquitectura tales como: latencia, tiempo de pérdidas de cache; o como una consecuencia natural de la paralelización del software, como por ejemplo: un desbalanceo de carga o contención de los recursos. La descomposición por overhead divide estos overheads en categorías significativas que ayudan tanto al usuario como a la tarea de modelado.

El aspecto clave de la descomposición por overhead es la selección de las categorías. Más específicamente, éstas categorías pueden clasificarse en dos tipos: cualitativas y cuantitativas. El significado y el propósito (bajo el contexto de modelado) de cada categoría son ejemplos de aspectos cualitativos, junto con la habilidad de medir la categoría. Sin embargo, los aspectos cuantitativos, como el número de categorías dentro del modelo, afecta la complejidad y el costo de la tarea de modelado. Cabe acotar que la descomposición por overhead ayuda a entender el rendimiento del sistema pero no mejora las técnicas empleadas para modelar cada categoría.

3.2.3. Modelado Estructural

En el modelado estructural el punto de inicio en el proceso de modelado es la aplicación como tal. La información estructurada es obtenida tanto a través de descripciones del usuario como a través de análisis de compilador.

Modelado Basado en Descripción

Para modelos basados en descripción, el proceso de modelado está basado en información descriptiva provista por el usuario acerca de la estructura y comportamiento del sistema paralelo. Existen modelos que estructuran el cómputo en forma de árbol. Sus marcos de trabajo dividen los modelos en parámetros relacionados con el hardware y el sistema de software. Los parámetros relacionados con el hardware incluyen el tiempo de cómputo y de comunicación de la aplicación.

El tiempo de comunicación está dividido en comunicación que puede ser solapada con cómputo y comunicación que no puede ser solapada. La generación del modelo se inicia desde las hojas del gráfico de tareas y los tiempos son combinados de acuerdo a la semántica de los nodos. Con respecto a los parámetros de hardware, éstos son intuitivos pero difíciles de capturar (por ejemplo, el solapamiento entre cómputo y comunicación). Además, otra fuente común de degradación de rendimiento, como la contención, no es considerada. Adicionalmente, el modelo de programación se vuelve restrictivo dada la dependencia de los gráficos de tareas en forma de árbol. Eventos como sincronización y *barriers* no son soportados por el modelo.

Los modelos basados en descripción se caracterizan por una descripción del programa que consiste en un gráfico de tareas (o equivalente) describiendo la parelización de la aplicación y algunos parámetros que expresan las características del hardware. La mayor ventaja en emplear gráficos de tareas es que son más amenos para modelar. Sin embargo, algunos modelos de programación no pueden ser caracterizados precisamente a través de este tipo de gráficos, un hecho que limita la aplicabilidad de ésta técnica. En cuanto a los parámetros que caracterizan el hardware, los factores a considerar son similares a los enfoques analíticos basados en parámetros: el nivel de abstracción de los parámetros y la dificultad para obtenerlos.

Modelado con Análisis Estático

El análisis estático difiere del enfoque descriptivo en que requiere mucha menos información y esfuerzo por parte del usuario. La información de rendimiento es producida automáticamente desde el código fuente original. Esta información es empleada para la paralelización y la selección de la partición de datos o para ayudar a sintonizar la aplicación.

Las herramientas de análisis estático suelen ser parte de entornos de compilación, donde proveen información de rendimiento acerca de la paralelización y la partición de los datos dentro de la aplicación. Estas herramientas son costosas en términos de desarrollo e implementación, ya que involucran técnicas un poco más sofisticadas. De todas maneras, su uso es trivial y el costo computacional para producir predicciones es insignificante. Con respecto a la precisión, no son

precisas en términos generales, pero usualmente suelen ser apropiadas para clasificar diferentes paralelizaciones.

3.3. Metodología

Para llevar ésta investigación a buen término y una vez que se han definido los principales conceptos teóricos en cuanto a análisis de rendimiento de sistemas paralelos y la realización del modelado de la aplicación, es importante señalar que el conjunto de pasos a seguir para conseguir realizar el trabajo pueden estar basados en el enfoque dado por Jain [19], enfoque de análisis de rendimiento diseñado hace casi dos décadas atrás, que no deja de ser un fundamento apropiado para llevar a cabo el proceso de evaluación de aplicaciones paralelas y sistemas computacionales. Inicialmente está basado en mostrar el enfoque apropiado para realizar la evaluación evitando incurrir en errores típicos como el perder el objetivo por el que se está realizando todo el proceso. Entre las etapas más relevantes que señala el autor y que se deberían considerar a medida que se avanza están:

1. *Establecer los objetivos y definir el sistema*: el primer paso en cualquier proyecto de evaluación de rendimiento es establecer los objetivos del estudio y definir qué consituye el sistema, delineando los límites del sistema. Dado un sistema o una aplicación, la definición del sistema puede variar dependiendo de los objetivos del estudio. Dada una aplicación paralela el objetivo podría ser determinar cuan flexible es la aplicación para ser ejecutada en uno u otro entorno, o el punto en que ésta esté sacando provecho del paralelismo del entorno sobre el que está siendo ejecutada. La elección de las limitantes del sistema afecta las métricas de rendimiento así como las cargas de trabajo empleadas para comparar los sistemas.

   Aún así, entender éstas limitantes es importante; ya que la consideración clave en establecer las limitaciones del sistema es el objetivo del estudio. Otras consideraciones, como control administrativo de los auspiciadores del estudio, también podrían ser tomados en cuenta. Si los auspiciadores no tienen control sobre ciertos componentes, quizá quieran mantener esos componentes fuera de los límites del sistema.

2. *Listar los servicios y las salidas*: cada sistema provee un conjunto de servicios. Por ejemplo, una aplicación paralela permite realizar un trabajo que serialmente en un sólo ordenador es mucho más complejo, distribuyendo el trabajo entre dos o más unidades de cómputo. El siguiente paso cuando se analiza un sistema es listar esos servicios. Cuando un usuario solicita cualquiera de estos servicios, existe un número de posibles salidas. Algunas de estas salidas son deseables otras quizás no. La aplicación paralela con la que se estuviera trabajando, puede no entregar los resultados esperados, debido a algún error en el diseño o en el entorno paralelo sobre el que se ejecuta, o sencillamente no entregar resultado alguno debido a un deadlock o problemas similares. Una lista de

servicios y salidas posibles es útil más tarde cuando se desean seleccionar las métricas adecuadas y las cargas de trabajo.

3. *Seleccionar las métricas*: el siguiente paso es seleccionar un criterio para comparar el rendimiento. Estos criterios son denominados *métricas*. En general, las métricas están relacionadas con la velocidad, precisión y disponibilidad de servicios. El rendimiento de las aplicaciones paralelas puede estar, por ejemplo, relacionado con el nivel de escalabilidad que ésta puede alcanzar, la precisión de cálculo, la cantidad de datos que puede manejar, o la rapidez con que entrega los resultados al usuario.

   Para cada estudio de rendimiento, un conjunto de criterios de rendimiento o métricas deben ser seleccionadas. Una forma de preparar este conjunto es listando los servicios ofrecidos por el sistema. Para cada solicitud de servicio realizado por el sistema, existen varias salidas posibles. Generalmente éstas salidas pueden ser clasificadas en tres categorías; el sistema puede realizar el servicio (1) correctamente, (2) incorrectamente o (3) reusarse a realizar el servicio. Si el sistema realiza el servicio correctamente, su rendimiento es medido por el tiempo que le ha tomado realizar el servicio, la tasa a la que el servicio es llevado a cabo y la cantidad de recursos consumidos a medida que la tarea es llevada a cabo.

   Éstas tres métricas relacionadas con el tiempo-tasa-recursos para un rendimiento exitoso son también denominadas métricas de respuesta, productividad y utilización, respectivamente. El recurso con mayor retraso o utilización bajo este enfoque es denominado *cuello de botella*. La optimización del rendimiento ligado con este recurso es el que generará mayor beneficio. Determinando la utilización de varios recursos dentro del sistema es la parte importante de la evaluación de rendimiento.

   Si el sistema lleva a cabo el servicio incorrectamente, se dice que ha ocurrido un error. Es muy útili clasificar los errores y determinar la probabilidad de cada clase de errores. Si el sistema no lleva a cabo el servicio, se dice que está "caído", fallado o no disponible. Una vez más es útil clasificar los modos de falla y determinar las probabilidades de cada clase.

   Las métricas asociadas con las tres salidas, denominadas servicio exitoso, error y no disponibilidad, son también denominadas métricas de velocidad, confianza y disponibilidad (métricas de respuesta, productividad y utilización, respectivamente). Debería ser obvio que para cada servicio ofrecido por el sistema, se debería tener un número de métricas de velocidad, confianza y disponibilidad. La mayoría de los sistemas ofrecen más de un servicio y por ello el número de métricas crece proporcionalmemte.

   Para muchas métricas, el valor medio es lo único importante, pero sin embargo, nunca se debe subestimar el efecto de la variabilidad. Por ejempli un tiempo medio de respuesta muy alto en un sistema de tiempo compartido, así como una alta variabilidad del tiempo de respuesta puede degradar la productividad significativamente. En estos

casos, es necesario estudiar ambas métricas.

En sistemas computacionales compartidos por varios usuarios, se deben considerar dos tipos de métricas, las individuales y las globales. Las métricas individuales reflejan el uso de cada usuario, mientras que las métricas globales reflejan el uso completo del sistema. La utilización de recursos, confiabilidad y disponibilidad son métricas globales, mientras que el tiempo de respuesta y el *throughput* pueden ser medidos tanto individual como globalmente para el sistema.

Existen casos cuando la decisión que optimiza métricas individuales es diferente de aquella que optimiza una métrica del sistema. Dado un número de métricas, es recomendable emplear las siguientes consideraciones para seleccionar un subconjunto: poca variabilidad, que ayuda a reducir el número de repeticiones necesarias para obtener un nivel de confianza estadística; no redundacia, que los mismos datos no sean obtenidos en otra variable totalmente diferente; y exhaustividad, que se refiere a que los datos reflejen lo que en realidad se quiere mostrar.

Métricas de Rendimiento Comúmente Empleadas

- Tiempo de Respuesta, está definido como el intervalo entre la solicitud del usuario y la respuesta del sistema [19]. Esta definición sin embargo, es simplista debido a que las solicitudes así como las respeustas no son instantáneas. El tiempo de tipeo de la solicitud por parte del ususario y el tiempo que le toma al sistema generar la salida son omitidos. Por ello el tiempo de respuesta puede presentar dos posibles definiciones, puede ser definido tanto como el intervalo entre la finalización de una solicitur y el inicio de su respuesta correspondiente por parte del sistema, o como el intervalo entre la finalización de la solicitud y el fin de la respuesta correspondiente. Ambas definiciones son aceptables siempre y cuando estén claramente especificadas.
- Throughput, está definido como la tasa (solicitudes por unidad de tiempo) en que las solicitudes son atendidas por el sistema. Para sistemas interactivos (sistemas de video bajo demanda), este es medido en solicitudes por segundo; para CPU's, se mide en Millones de Instrucciones Por Segundo (MIPS) o Millones de Operaciones de Punto Flotante por Segundo (MFLOPS). Para redes es medido en paquetes por segundo (pps) o bits por segundo (bps). El throughput de un sistema generalmente aumenta a medida que la carga del sisteme aumenta. El throughput máximo alcanzable bajo unas condiciones de carga de trabajo ideal es denominado capacidad nominal del sistema. En algunos casos, es más interesante saber que el máximo throughput alcanzable sin exceder un tiempo límite de respuesta prestablecido, puede ser llamado como la capacidad usable del sistema.
- La utilización de un recurso es medida como la fracción de tiempo que el recurso es ocupado atendiendo solicitudes. Esta es la relación entre el tiempo ocupado

y el tiempo total transcurrido en un período dado. El período durante el cual el recurso no está siendo utilizado es denominado *tiempo de inactividad*. Los gestores de sistemas normalmente están interesados en balancear la carga de forma que un recurso no sea utilizado más que otros. Sin embargo, no siempre es posible lograrlo.

- La fiabilidad de un sistema es usualmente medida por la probabilidad de errores o por el tiempo medio entre errores. Éste último es conocido como segundos libres de errores.
- La disponibilidad de un sistema está definida por la fracción de tiempo que el sistema está disponible para servir las solicitudes de los usuarios. El tiempo durante el cual el sistema no está disponible es denominado *downtime*; y el tiempo durante el cual es sistema está dispuesto es llamado *uptime*.

Clasificación de Utilidad de las Métricas de Rendimiento

Dependiendo de la utilidad de una métrica de rendimiento, esta puede ser categorizada en tres clases:

- *Higher is Better* o HB, los usuarios del sistema y los gestores del sistema prefieren valores altos en éstas métricas. Por ejemplo: el troughput es una de ellas.
- *Lower is Better* o LB, los gestores y usuarios del sistema prefieren los valores más pequeños posibles en dichas métricas, como en el caso del tiempo de respuesta.
- *Nominal is Best* o NB. Tanto valores muy altos como muy bajos no son deseables. Mientras que un valor particular en el medio es considerado lo mejor. Por ejemplo, la utilización es considerada una característica NB. Ya que si existe una utilización muy alta es considerada mala por parte del usuario por tener un tiempo de respuesta muy alto. Pero una utilización muy baja es considerada mala por los administradores del sistema ya que los recursos del sistema no están siendo utilizados. Es por ello que se esperan valores entre un determinado rango para ser considerados los mejores.

4. *Listar los parámetros*, el siguiente paso en proyectos de rendimiento es realizar una lista de todos los parámetros que afectan el rendimiento. Esa lista puede ser dividida en parámetros del sistema y parámetros de carga de trabajo. Los parámetros de sistema incluyen tanto los de hardware como los de software, los cuales generalmente no varían a lo largo de varias instalaciones en el sistema.

   Los parámetros de carga de trabajo son características de las solicitudes de los usuarios, lo cuales varían de una instalación a la siguiente. La lista de parámetros puede que no esté completa; esto es, que luego de una primera revisión de análisis, se puede descubrir que existen parámetros adicionales que afectan el rendimiento.

Entonces pueden ser añadidos a la lista, pero todo el tiempo es recomendable conservar la lista lo más fácil de entender posible. Esto permite al analista y a aquéllos encargados de tomar decisiones de discutir el impacto de varios parámetros y determinar qué parámetros son necesarios para recolectar despueés o durante el análisis.

5. *Seleccionar los Factores para estudiar*, la lista de parámetros puede ser dividida en dos partes: aquellos que serán verificados durante la evaluación y aquéllos que no. Los parámetros que serán variados, son llamados factores y sus valores son llamados niveles. En general, la lista de factores, y sus posibles niveles, es mayor que la que los recursos disponibles podría permitir. De todas maneras, la lista continuaría creciendo hasta que se torne obvio que no hay suficientes recursos para estudiar el problema. Es mejor empezar con una lista corta de factores y un pequeño número de niveles por cada factor y luego extender la lista en la siguiente fase del proyecto, que es la selección de la técnica de evaluación si los recursos lo permiten.

   Los parámetros que se espera que tenga un impacto mayor en el rendimiento son los que se deben seleccionar preferiblemente como factores. Tal como sucede con las métricas, un error común cuando se seleccionan los factores es que los parámetros que son fáciles de variar y medir son usados como factores mientras que otros más influyentes son ignorados sencillamente por la dificultad que traen consigo para ser medidos. Cuando se seleccionan factores, es importante considerar las restricciones económicas y técnicas que existen así como incluir las limitaciones impuestas por las personas encargadas de la decisión final y el tiempo disponible para tomar la decisión. Esto aumenta las oportunidades de encontrar una solución que es aceptable e implementable.

6. *Seleccionar la técnica de evaluación.* Las tres técnicas para evaluación de rendimiento son el modelado analítico, la simulación y la medición sobre un sistema real. La selección de la técnica adecuada depende del tiempo y recursos disponibles para resolver el problema y el nivel de precisión deseado.

7. *Seleccionar la carga de trabajo.* La carga de trabajo consiste en una lista de solicitudes de servicios para el sistema. Por ejemplo la carga de trabajo para la comparación de varias bases de datos puede consistir en un conjunto de consultas. Dependiendo de la técnica de evaluación seleccionada, la carga de trabajo puede ser expresada de diferentes formas. Para el modelado analítico, la carga de trabajo puede ser expresada como la probabilidad de varias solicitudes. Para simulación, se puede usar una traza de solicitudes medidas en un sistema real. Y para las mediciones, la carga de trabajo puede consistir en scripts que serán ejecutados en el sistema. En todos los casos, es escencial que la carga de trabajo sea representativa del uso del sistema en la vida real. Para producir cargas de trabajo representativas, es necesario medir y caracterizar la carga de trabajo en sistemas existentes.

8. *Diseñar los experimentos.* Una vez que se tiene una lista de factores y sus niveles, es necesario decidir una secuencia de experimentos que ofrezcan la mayor cantidad de información con el menor esfuerzo. En la práctica, es muy útil realizar los experimentos en dos fases. En la primera fase, el número de factores puede ser grande pero el número de niveles lo más pequeño posible. El objetivo es determinar el efecto relativo de varios factores. Muchas veces, esto puede ser llevado a cabo con un diseño de experimentos en factorial. En la segunda fase, el número de factores es reducido y el número de niveles de esos factores que tienen impacto significativo incrementados.

9. *Analizar e interpretar los datos.* Es importante reconocer que las salidas de las mediciones y simulaciones son cantidades aleatorias y que la salida puede ser diferente cada vez que el experimento es repetido. Comparando las dos alternativas, es necesario tomar en cuenta la variabilidad de los resultados. Simplemente comparando las medias puede llevar a conclusiones imprecisas. Para ello habrá que hacer uso de técnicas estadísticas para comparar las dos alternativas.

   Interpretando los resultados de un análisis es parte clave cuando se trata de analizar el rendimiento. Se debe comprender que el análisis solo produce resultados y no conclusiones. Los resultados proveen las bases sobre las que los analistas pueden dibujar conclusiones. Cuando a un grupo de analistas les son dados los mismos conjuntos de resultados, la conclusión generada por cada uno de ellos puede ser diferente.

10. *Presentar los resultados.* El paso final de todo proyecto de análisis de rendimiento es comunicar los resultados a otros miembros del grupo de investigación o aquéllas personas relacionadas con la toma de decisión al final de la investigación. Es importante que los resultados sean presentados de forma que se puedan entender fácilmente. Normalmente esto requiere la presentación de forma gráfica y sin jerga estadística. Los gráficos deben estar diseñados con la escala apropiada.

    A menudo en este punto, todo el conocimiento generado una vez finalizado el estudio del proyecto, puede requerir que el desarrollador vuelva atrás y reconsiderar algunas de las decisiones tomadas en fases anteriores. Definir de nuevo algunas limitaciones del proyecto que podrían proporcionar mayor información para la decisión final con respecto a la aplicación o la idea final obtenida luego de todo el trabajo.

Todas estas etapas del enfoque de evaluación de rendimiento podrían ir acompañadas de enfoques de gestión de proyectos como *Plan, Do, Check, Act*, también conocido como el círculo de Demming (http://www.balancedscorecard.org/TheDemingCycle/tabid/112/Default.aspx), en el que cada una de estas fases es planificada con un alto nivel de atención, luego llevada a cabo con la intención de obtener determinados resultados que serán revisados y basándose en ellos se decide que acto habrá de ser llevado a cabo para continuar con la investigación. Entiéndase que este proceso iterativo va acompañando cada una de las fases que habrán de ser completadas en aras de realizar un análisis de rendimiento apropiado y preciso.

Existen diferentes técnicas de evaluación de rendimiento en un sistema o en una aplicación como tal, éstas van directamente relacionadas con la fase de desarrollo en que se encuentre el proyecto, la cantidad de tiempo que consumen, las herramientas o recursos necesarios para llevarlos a cabo, el nivel de precisón que brinda una u otra, el costo que genera, entre otras. Inicialmente se identifican tres técnicas diferentes para la observación del comportamiento de una aplicación: (1) Medición, (2) Simulación y (3) Modelado; principalmente la selección de una u otra técnica de medición viene ligada con el estado de desarrollo en que se encuentre el proyecto.

Según lo señala Jain [19] inicialmente la medición es más fácil de ser llevada a cabo en proyectos que ya están finalizados; es decir, que ya existe una versión del producto, pero si se tratase de una nueva idea o concepto que se quisiera evaluar, solamente el modelado y la simulación son capaces de brindar las respuestas necesarias, ya que son usados en situaciones donde no es posible realizar una medición; sin embargo, sería mucho más convincente para otros si el modelo analítico o la simulación están basados en mediciones previas.

Otro factor determinante al momento de seleccionar una técnica de evaluación es el tiempo disponible para realizar todo el proceso, cabe recordar que en el campo científico la mayoría de las veces los resultados se necesitan para ayer; si este fuera el único factor determinante sería necesario considerar el modelado analítico, ya que las simulaciones consumen una gran cantidad de tiempo, y aunque las mediciones toman mucho menos tiempo que las simulaciones, tienden a tomar un poco más que con el modelado, pero como cuando "nada puede ir mal seguramente irá", el tiempo para las mediciones es el más variable de las tres técnicas.

Una consideración que debe ser tomada en cuenta es la disponibilidad de herramientas y el nivel de habilidad del analista, ésta vendrá ligada directamente con el campo que mejor domine. Si es hábil con lenguajes de simulación probablemente se mantendrá alejado de técnicas de medición o modelado; pero si se le da muy bien el manejo de modelos, apostará por esta técnica o en su defecto se limitará a medición y simulación. Por lo que la selección de la técnica de evaluación termina convirtiéndose en algo muy relativo.

Asimismo, factores como el grado de precisión que se persigue en los resultados, se obtendrán valores *reales* muy variables provenientes de las mediciones tomadas sobre el objeto de estudio, sin embargo si el modelado y la simulación fueran llevados a cabo, el grado de precisión sería muy bajo y moderado respectivamente, ya que está directamente ligado con el nivel de similitud existente entre el modelo o el simulador con el entorno en la realidad. Un simulador es la aproximación más real que se puede generar para observar el comportamiento de un entorno en el que no es fácil realizar medidas.

Si el objetivo del proceso de evaluación es determinar el nivel en que diferentes factores afectan el desempeño de cierto sistema, es mucho más útil y viable realizar un modelo que permita observar la influencia que genera cada uno de los diferentes factores sobre el comportamiento global; aunque, si se desea comparar diferentes alternativas y determinar un valor exacto de un factor que es capaz de brindar el mejor rendimiento, técnicas como la de medición son las

más recomendables. Con la simulación es posible obtener un conjunto de datos que representan el punto más alto de rendimiento en el sistema, sin embargo no siempre es fácil identificar la influencia de cada uno de los factores.

En cuanto al costo, la generación de un modelo analítico, en contraposición con la técnica de medición, se convierte en la opción más económica, esto debido a que consume pocos recursos, muchas veces solo con lápiz y papel es posible generarlo, en cambio cuando hablamos de tomar medidas, es necesario tener el sistema a evaluar funcionando, invertir en herramientas de medición y en tiempo. Aún así suele ser recomendable acompañar cada una de las técnicas de evaluación con alguna de las otras dos simultánea o secuencialmente, de forma que pueda validarse cualquier evaluación realizada anteriormente.

En un principio el proceso de evaluación y obtención de resultados requiere cierto fundamento estadístico que permita darle validez a los datos obtenidos y posteriormente realizar el análisis necesario para sacar las conclusiones pertinentes que se persiguen durante el proceso de análisis de rendimiento; para ello es necesario seguir un cierto número de pasos que nos orientan como analistas durante el proceso de investigación y análisis de la aplicación.

Inicialmente es necesario haber definido concretamente cual es el problema y su alcance, con ello se consigue tener una idea general del enfoque que habrá que darle al proceso de experimentación de forma que genere los resultados necesarios para obtener las conclusiones pertinentes de la forma más práctica y que consuma la menor cantidad de tiempo y recursos posibles. Una vez que el problema ha sido enmarcado y se ha definido la salida que se espera obtener cuando los experimentos concluyan, se logra definir *variable de respuesta*; con la que se procede posteriormente a la selección de las variables que afectan su comportamiento y de los cuales se pueden tener varias alternativas, identificadas como *factores*. Factores que según los posibles valores que pudieran tomar tendrán ciertos *niveles* que habrán de ser considerados al momento de diseñar el grupo de experimentos que se van a realizar.

De igual forma los factores antes mencionados, podrán ser *factores primarios* o *factores secundarios*, estos se seleccionan de acuerdo a aquéllos que influyen en el comportamiento de los experimentos y que habrán de ser cuantificados o no, respectivamente. Suelen existir un gran número de factores que impactan en los resultados obtenidos y que algunas veces será necesario tener registros de los efectos que éstos generen, si este fuera es caso se trataría de factores primarios; mientras que si no se está interesado en cuantificar el impacto que éstos generen, éstos terminan clasificados como factores secundarios. Asimismo, habrá que fijar el número de veces que se plantea realizar un determinado experimento para obtener los resultados necesarios, este valor es conocido como *replicación*.

Por otro lado, es importante señalar que como *unidad experimental* se conoce a cualquier entidad que es usada para el experimento, generalmente solo aquéllas unidades experimentales que son consideradas como uno de los factores en el estudio son las que interesan; el objetivo del diseño experimental es minimizar el impacto de variación entre unidades experimentales, por ejemplo: en el caso de una aplicación paralela, el entorno sobre el que ésta está siendo probada o

donde están siendo llevado a cabo los experimentos. Asimismo, no se debe dejar de lado el nivel de *interacción* existente entre dos o más factores; se dice que dos factores interactúan uno con el otro cuando el efecto de uno depende del nivel del otro.

El desarrollo de la metodología se verá reflejado en el capítulo 5, donde se detalla cada una de las fases y los datos que se han tomado en consideración para realizar el análisis de rendimiento de mpiBLAST como aplicación paralela de bioinformática, y del cual se espera generar una conclusión sobre sus restricciones, limitantes y ventajas.

Capítulo 4

Aplicaciones Bioinformáticas

La genética... "Será una segunda pirámide de Keops: un coloso que está ahí y que nadie puede usar"

Erwin Chargaff (1905-2002).

4.1. Introducción

En los últimos años se ha observado un inmenso número de organismos que han sido secuenciados en proyectos de genómica e incluidos en bases de datos genómicas cada vez más grandes, con ello el desarrollo de la biología molecular ha ido tomando más y más fuerza y forma a medida que pasa el tiempo. Las bases de datos genómicas fueron diseñadas con la idea de tener un "respaldo" de la información biológica de los organismos para que puedan ser analizadas luego.

Observando estudios realizados por el GenBank [5] que es una de las principales bibliotecas de bases de datos biológicas, el crecimiento de estas en los últimos años ha sido de forma exponencial, aumentando en hasta varios órdenes de magnitud; adicionalmente, éstas bases de datos son consultadas de forma intensiva diariamente. Los biólogos han tenido que enfrentarse al problema de lidiar con bases de datos inmensas durante la búsqueda de similitudes significativas entre secuencias biológicas. Para alcanzar este objetivo, ha sido necesario recurrir a un gran poder de cómputo y grandes espacios de almacenamiento. Aún más todavía, han tenido que recurrir al empleo de algoritmos sofisticados para modelar las relaciones realísticas entre los organismos.

Los bancos de datos genómicos son escrutados rutinaria y diariamente por miles de investigadores. Una de las tareas más comunes en biología molecular es tratar de asignar una función [1] a un gen o proteína desconocido. Cuando se escanean las bases de datos, los biólogos lidian con un gran dilema: velocidad o calidad. Los datos genómicos crecen exponencialmente (se duplican cada 12-15 meses) tan o más rápido que el poder de cómputo (que se duplica cada 18 meses). En un ordenador estándar, una búsqueda de alta calidad puede tomar horas mientras que

[1] Una función es la respuesta a la pregunta de por qué algunos elementos o procesos ocurren en un sistema que ha evolucionado a través del proceso de selección.

una búsqeda apenas aproximada pudiera tomar solo unas cuantas décimas de segundos. Claro está que las búsquedas de alta calidad están basadas en métodos de programación dinámica que consumen gran cantidad de tiempo. Mientras que los aproximados como en el caso de BLAST, están basados en heurísticas.

Este proceso de búsqueda es conocido como *Alineamiento de Secuencias* [2] con la cual se puede realizar la medición del grado de similitud [3] existente entre dos secuencias. Para ello se superponen las secuencias de forma que se consiga determinar la cantidad de letras o símbolos que coinciden. Obteniéndose una matriz, en la que las filas son las secuencias y las columnas, son las posiciones de las letras.

BLAST (Basic Local Alignment Search Tool) [1] es la herramienta heurística más usada, que busca las similitudes entre secuencias biológicas. A diferencia de métodos exactos como el de Needleman [27] y el de Smith-Waterman [33] que presentan complejidad cuadrática de tiempo y espacio, BLAST es a menudo ejecutado rápidamente para comparación de pares de secuencias en pequeños espacios de memoria. Aún así, los biólogos usualmente no comparan sólo dos secuencias sino un conjunto de secuencias descubiertas con toda una secuencia genómica. En este caso, los tiempos de ejecución de BLAST pueden ser muy altos, y enfoques alternativos como cómputo paralelo y distribuído, debería producir resultados en un menor tiempo, ya que facilitan el manejo de un mayor volúmen de datos y permiten la posibilidad de ejecutar varias tareas simultáneamente. La ejecución serial de BLAST versa en que sólo un nódo de cómputo es el que se encarga de realizar la búsqueda en la totalidad de la extensión de la base de datos biológica, realizando los alineamientos y obteniendo aquéllas secuencias homólogas. Sin embargo, cuando se plantea el paradigma de paralelismo, este trabajo puede ser distribuído entre diferentes máquinas, de forma que 2 o más nodos de cómputo estarán realizando los alineamientos contra la base de datos.

Los ficheros de datos genómicos con los que las secuencias de consulta o patrones son comparados, son compilados en grandes bases de datos. GenBank, una colección de información de ADN y proteínas, mantiene más de 44 billones de pares bases en más de 40 millones de secuencias. De esta forma, el tamaño de cada caracter que representa una base en una cadena es solamente un byte, que considerados en conjunto estaríamos hablando de una considerable cantidad de datos. Los archivos de las bases de datos son a menudo mayores a un gigabyte cada uno. Por lo que la cantidad de cómputo requerido para completar una búsqueda es proporcional a la longitud de la secuencia de consulta y el tamaño de la base de datos. La rápidez con que un ordenador pueda finalizar el trabajo de búsqueda está determinada por la capacidad de solapar procesamiento y gestión de los datos de forma eficiente, de manera que se obtenga un mejor

[2] Las moléculas ADN, ARN y proteína se pueden considerar cadenas de las moléculas componentes, lo que en el lenguaje de la bioinformática se traduce por secuencias. Cada una de las moléculas componentes se abrevia con una letra, de manera que trataremos con secuencias de letras.

[3] Generalmente, una alta similitud entre dos secuencias se debe a que estas dos secuencias son homólogas. Esto significa que las dos secuencias tienen un ancestro común, es decir, que derivan de una misma molécula. La acumulación de mutaciones en el ADN a lo largo del tiempo es la causa de que las dos secuencias ya no sean idénticas, sino sólo similares. Cuando tratamos con un conjunto de proteínas homólogas, se utiliza el término familia de proteínas.[34]

rendimiento. Ordenadores diseñados a medida con muchos procesadores son usualmente muy costosos y exceden el tope financiero de laboratorios pequeños o individuales. Es por ello que búsquedas en huéspedes basado en web son sujeto de una afluencia intensiva de usuarios y el tiempo para obtener una respuesta aumenta durante los picos de uso.

Dado que BLAST es una herramienta bioinformática base en situaciones de alineamientos de secuencias se han planteado estrategias de ejecución fuera de las usuales en cústers de ordenadores dedicados y se ha considerado el uso de grids, trabajo que puede ser observado en [39]. De forma que se pueda contar con mayores recursos de cómputo para realizar la tarea. Sin embargo siempre se puede carecer el control total de los parámetros relacionados con el entorno de ejecución que para nuestra investigación es relevante.

El dicho "muchas manos aligeran el trabajo", se aplica bastante bien al cómputo. Muchos nodos de cómputo colaborando en una manera centralmente coordinada permite realizar mucho más trabajo en una forma mucho más rápida que si se hiciera en solo un ordenador. Esto debido a la necesidad de contrastar ficheros tan grandes como pudieran ser el genoma humano contrastado contra el genoma de otro mamífero como los ratones. Estaríamos hablando de la necesidad de repartir entre cada "mano" una gran parte del trabajo de forma que se consiga agilizar este tipo de alineamientos que son cómputointensivos y requieren cada vez mayor cantidad de recursos para poder gestionar la E/S de los datos involucrados. Acudir al paradigma de paralelismo en el caso de alineamientos masivos representa la principal opción para resolver este tipo de tareas.

El speedup ideal de una aplicación deberia ser dado por un factor n, donde n es el número de nodos sobre los que la aplicación es distribuida. En realidad este speedup teórico no es alcanzable. Tal como lo señaló Amdahl [3], el máximo speedup viene determinado por la parte de la aplicación que no puede ser distribuida. La parte de cómputo más intensiva de los programas de BLAST es la parte en la que se realiza el alineamiento de secuencia, que además es la porción que puede ser distribuída entre multiples nodos. Por lo tanto, aplicaciones que distribuyen la búsqueda de BLAST son las más eficientes.

En contrapartida, las particiones de las bases de datos deben ser distribuídas a las memorias locales, lo que requiere que el envío de datos a procesar sea más rápidos que la entrada y salida y se puedan solapar con el cómputo.

4.2. Alineamiento de Secuencias

La comparación de secuencias biológicas es una operación fundamental en la bioinformática, ya que facilita el relacionamiento entre ellas y permite la descripción de las funciones principales de cada una de ellas. Es de hecho un problema de correspondencia de patrones, que consiste en encontrar qué partes de las secuencias son homólogas o comparten una función. El alineamiento de secuencias es un proceso de comparación de dos (o más) secuencias de forma que se consiga, que un conjunto de caracteres individuales o patrones de caracteres se encuentren en el mismo orden en un "alineamiento" vertical.

Dado que una secuencia biológica contiene toda la información referente a un genoma o proteína determinado representado en caracteres, estamos hablando del manejo de la "esencia" de la vida como cadenas de texto; toda la información genética de una especie como tal se encuentra resumida en líneas de compuestos biológicos, que a su vez han ido evolucionando y/o mutando a través del proceso natural de selección dejando herencias unas cadenas en otra.

Para comprender la realidad de este tipo de comparaciones biológicas partimos de la idea de que un ser humano tiene una cadena biológica característica que lo identifica como ser humano; esta cadena biológica es el ADN (Ácido Desoxirribonucléico). El ADN está representado como una cadena de doble hélice compuesta por pares de dos de cuatro elementos constituyentes denominados nucleótidos (un nucleótido está conformado por un grupo de fosfatos enlazados con algunos azúcares denominado *desoxirribosa*, que a su vez están ligados a uno de cuatro tipos de bases orgánicas de nitrógeno de nombre Adenina, Timina, Guanina y Citosina). Simplificando así nuestra esencia como organismo vivo en todo su esplendor está reducido a cadenas de texto compuesta de los caracteres A, C, T y G.

Las proteínas están construídas de pequeños bloques denominados aminoácidos, que son moléculas que por si sólas ya son bastante complejas, compuestas de carbono, hidrógeno, oxígeno, nitrógeno y átomos de azufre.

Estos aminoácidos se encontraban ligados entre ellos como una cadena, y la esencia de la proteína estaba determinada no sólo por la cantidad de aminoácidos que contenían sino por el orden exacto en que éstos estaban distribuídos. La primera secuencia de aminoácidos de una proteína, fue determinada en 1951 y se trataba de la insulina, la cual se derivaba de la siguiente cadena:

Insulina = MALWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFF
YTPKTRREAEDLQVGQVELGGGPGAGSLQPLALEGSLQKRGIVEQCCTSICSLYQLEN
YCN.

Una vez que tenemos en consideración el tipo de cadenas biológicas (también denominadas secuencia) con las que estamos tratando podemos introducir el concepto de lo que un alineamiento representa en sí. Básicamente, hay dos tipos de comparaciones, las globales y las locales, y la selección de una u otra depende del interés que tenga el biólogo de descifrar alguna función basándose en la totalidad de la secuencia o sólo parcialmente [7].

Los alineamientos globales se refieren a un alineamiento total de la secuencia, donde todas las bases (nitrogenáda o aminoácido) son alineadas con otras bases o con un gap ("-"). Los algoritmos de alineamientos globales comienzal al principio de las dos secuencias (cadenas biológicas) y adicionan los gaps en la medida que construyen el alineamiento.

En los alineamientos locales, se alinean partes de la secuencia. No existe necesidad de alinear todas las bases y los algoritmos de este tipo de alineamiento buscan las regiones con mayor similitud y comienzan el alineamiento a partir de allí.

Para comparar dos secuencias, es necesario conseguir el mejor alineamiento entre ellas, el cual es colocar una secuencia sobre la otra, realizando la correspondencia entre caracteres similares.

En un alineamiento, los espacios pueden ser insertados en localidades arbitrarias a lo largo de las secuencias de forma que terminen con el mismo tamaño. Habiendo obtenido un alineamiento entre dos secuencias biológicas x e y, un puntaje puede ser asignado de la siguiente manera. Para cada columna, se asigna por ejemplo, por cada base, 0 si los dos caracteres son iguales, -1 si los caracteres son diferentes y -2 si uno de ellos es un espacio[35]. Estos puntajes se determinan de acuerdo al nivel de relevancia que se desea que tenga el alineamiento. El puntaje es la suma de los valores computados para cada columna y el mayor puntaje es la similitud entre las dos secuencias.

El algoritmo exacto de Needleman-Wunsh (NW)[27] está basado en programación dinámica, genera el mejor alineamiento global entre dos secuencias. Para calcular alineamientos de secuencias locales, Smith y Waterman[33], propusieron un algoritmo (SW), también basado en programación dinámica (ver figura 4.1), con complejidad cuadrática de tiempo y espacio.

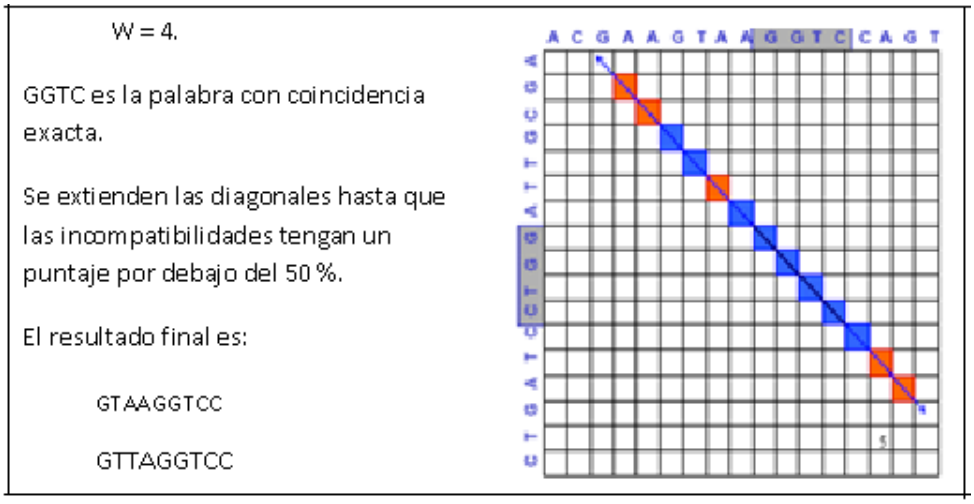

Figura 4.1: Alineamiento de secuencias basado en programación dinámica

Hirschberg [18] propuso un algoritmo exacto que calcula el alineamiento entre dos secuencias x e y en tiempo cuadrático pero espacio lineal. Este enfoque separa la secuencia x por la mitad, generando subsecuencias x_1 y x_2, y calcula el lugar correspondiente para partir la secuencia y, generando las subsecuencias y_1 e y_2, de esta forma el problema de alineamiento puede ser resuelto de forma recursiva en un *divide and conquer*.

Usualmente, una secuencia biológica dada es comparada contra miles o incluso millones de secuencias que componen las bases de datos genéticas. Uno de los más importantes repositorios es aquél que es parte de una colaboración que involucra al GenBank del National Center for Biotechnology Information (NCBI), el European Molecular Biology Laboratoy (EMBL) y el Banco de Datos de ADN de JAPON (DDBJ, DNA Data Bank of Japan). Estas organizaciones intercambian datos diariamente y una versión nueva es generada cada dos meses.

Bajo este escenario, el uso de métos exactos como el de Needleman-Wunsh y Smith-Waterman es prohibitivo. Por esta razón, métodos heurísticos más rápidos son propuestos, sin embargo,

estos no garantizan que el mejor alineamiento será producido. Normalmente, estos métodos heurísticos son evaluados empleando los conceptos de sensibilidad y selectividad. *Sensibilidad* es la habilidad para reconocer tantos alineamientos significativos como sean posibles, incluyendo secuencias distantemente relacionadas. Vendría dada por:

$$Sensibilidad = \frac{Número\ de\ coincidencias\ encontradas}{Número\ de\ coincidencias\ significativas\ de\ la\ Base\ de\ datos} \tag{4.1}$$

Mientras que la *Selectividad* es la habilidad de acercar la búsqueda con la intención de descartar los falsos positivos. Típicamente, existe una cierta separación entre sensibilidad y selectividad, ya que un algoritmo muy sensible es aquél que va a perseguir encontrar sólo aquéllos valores que coincidan exactamente unos con otros, ej. el algoritmo de SmithWaterman, mientras que un algoritmo como BLAST que es selectivo busca aquéllos alineamientos similares lo más rápido posible; es por ello que se acota que si se requieren resultados muy precisos y detallados habrá de usarse un método muy sensible, en cambio si se desean aproximaciones rápidas se emplearán aquéllos con un alto nivel de selectividad.

Los métodos heurísticos acostumbran usar matrices de puntuación para calcular las penalizaciones de no coincidir entre dos proteínas diferentes; algunas diferencias son más posibles que sucedan que otras y pueden indicar aspectos evolutivos. Es por esto que, los métodos de alineamiento de proteínas usan matrices con valores que penalizan la existencia de diferencias entre las secuencias, con la finalidad de reflejar posteriormente el nivel de similitud existente cuando han habido cambios entre ellas.

Las matrices de puntuación mayormente usadas son las matrices PAM (Percent Accepted Mutations)[9, 13] y las BLOSUM (Blocks Substitution Matrix) [17]. Las matrices PAM son el resultado de un trabajo extensivo que analiza la frecuencia en que un aminoácido determinado es reemplazado por otro aminoácido durante la evolución. Las matrices de puntuación BLOSUM son generadas considerando las tasas de evolución de una región de proteínas (un bloque) más que la proteína completa.

Conjuntamente con BLAST [1] que es una de las heurísticas más usadas para comparación de secuencias, existe FASTA [29], que además emplean tanto matrices PAM como las BLOSUM.

FASTA es el nombre de un programa de alineamiento de secuencias y búsqueda en bases de datos biológicas creado por W.R Pearson y D.J Lipman en 1988. Las secuencias usadas por FASTA deben estar descritas con el siguiente formato:

>Nombre_de_la_secuencia
ARCGTCRGCKINTANDRGCKINTANDCKINTANDARCGTCRGCKINTANDRGCKINTAND

La línea que inicia con >, es llamada la *línea de definición*, contiene un identificador único seguido de una descripción corta opcional. Las lineas que la suceden contienen la secuencia de ADN o la proteína, hasta que el siguiente símbolo > indique que ha iniciado una nueva secuencia.

Dado que FASTA es fácil de analizar, este formato se ha vuelto sumamente popular y es

actualmente el formato predeterminado de entrada en muchos software de análisis de secuencia, incluyendo BLAST, por ejemplo una secuencia de consulta puede tener la forma que se presenta en la figura 4.2 donde se puede percibir fácilmente donde comienza un nucleótido, donde termina y donde comienza la siguiente.

Las bases de datos biológicas están construídas como conjuntos de secuencias (proteínas o ADN) almacenadas en ficheros texto bajo formatos como el FASTA descrito anteriormente, y se encuentran almacenadas en repositorios de datos genéticos como el GenBank [4] donde se pueden visualizar nucleótidos como el de la figura 4.3

4.2.1. BLAST Basic Local Alignment Sequence Tool

BLAST fue diseñado por Altschul [1] en 1990. Está basado en un algoritmo heurístico que fue diseñado para ejecutarse rápidamente manteniendo un alto nivel de sensibilidad. La primer versión de BLAST buscaba alineamientos locales sin considerar *gaps* [5] y su motivación fue mejorar el rendimiento de los algoritmos FASTA [29], esto fue conseguido integrando el uso de matrices PAM en el primer paso del algoritmo. En 1996 y 1997, versiones mejoradas de BLAST que soportaban la inserción de *gaps* fueron propuestas, como en el caso de NCBI-BLAST (http://blast.ncbi.nlm.nih.gov/Blast.cgi) y Gapped BLAST [2].

BLAST provee programas para comparar diferentes combinaciones de tipos de secuencias y secuencias de bases de datos, traduciendo secuencias a medida que se ejecuta. Los diferentes programas que forman parte de la familia de BLAST se encuentran representados en la figura 4.4, donde dependiendo de la naturaleza de la secuencia de consulta o de la base de datos se emplea uno u otro.

Nombre de búsqueda	Tipo de Secuencia de Consulta	Tipo de Base de Datos	Traduce
blastn	Nucleótido	Nucleótido	Ninguna
tblastn	Proteína	Nucleótido	Base de Datos
blastx	Nucleótido	Proteína	Secuencia de consulta
blastp	Proteína	Proteína	Ninguna
tblastx	Nucleótido	Nucleótido	Secuencia de consulta y la Base de Datos

Tabla 4.1: Programas de la familia de BLAST

El algoritmo de BLAST está dividido en tres fases definidas:

1. *Seeding*, BLAST compara las secuencias de consulta contra todas las secuencias dentro de una base de datos. BLAST emplea el concepto de palabras que están conformadas

[4](http://www.ncbi.nlm.nih.gov/Genbank/)

[5]Un gap es un espacio en blanco representado con un guión () que se introduce en una secuencia con la finalidad de asemejarla lo más posible a la otra secuencia con la que se está alineando

Figura 4.2: Ejemplo fichero de secuencias en formato FASTA

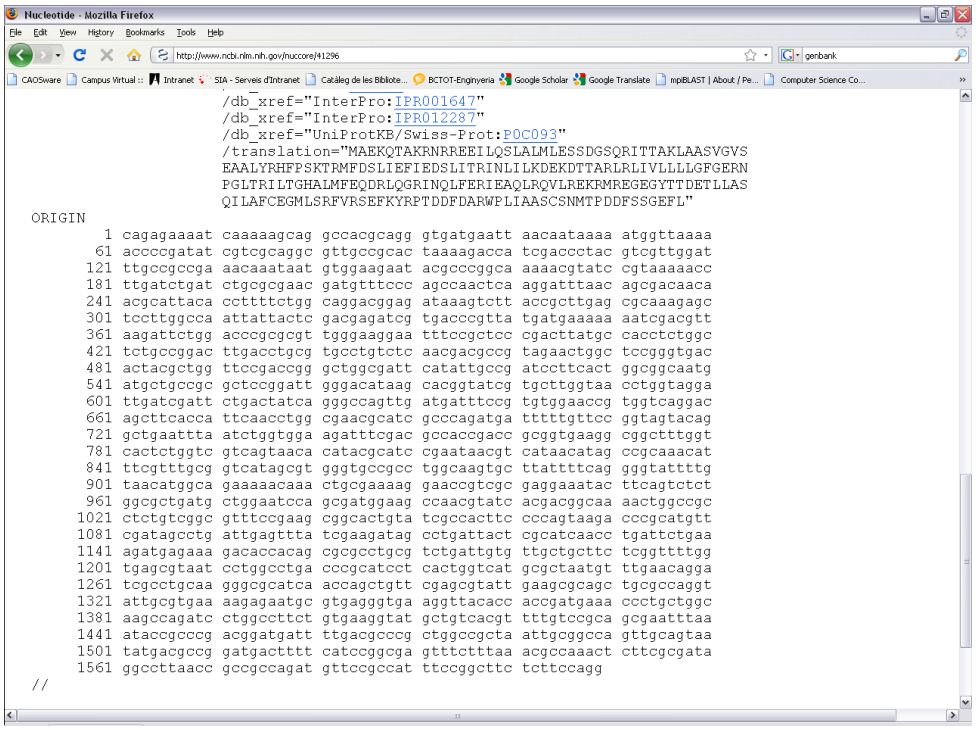

Figura 4.3: Nucleótido disponible en el GenBank

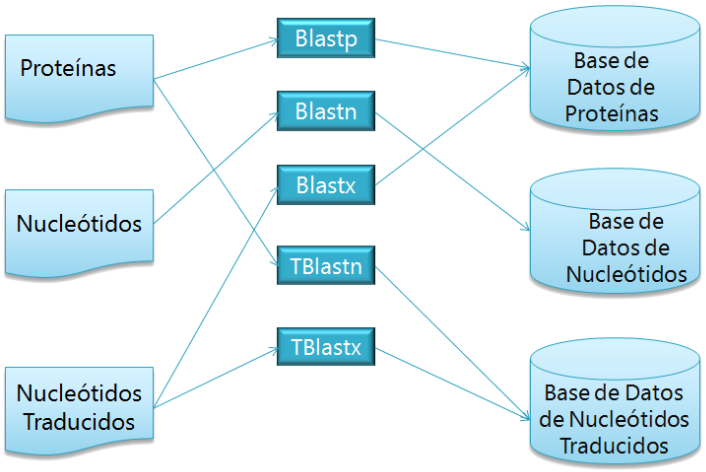

Figura 4.4: Programas de la familia de BLAST.

por un conjunto finito de palabras de longitud w que aparecen dentro de una secuencia dada. Por ejemplo, la secuencia CTAGGT contiene cuatro palabras de longitud 3: CTA, TAG, AGG, GGT. El algoritmo de BLAST asume que alineamientos significativos tienen palabras en común. La ubicación de todas las palabras compartidas, de w cantidad de letras, entre dos secuencias dadas está determinada por la coincidencia exacta entre las palabras; la secuencia de consulta, también puede ser comparada con una base de datos genómica, generando las palabras compartidas entre la secuencia y las secuencias dentro de la base de datos; estas palabras identificadas son conocidas como *palabras idénticas* y solo las regiones con palabras idénticas pueden ser usadas como semillas (*seeds*, de ahí el nombre de la fase) para el alineamiento. Esta lista es evaluada usando la matriz de substitución y el concepto de vecindario. El vecino de una palabra incluye la palabra en sí misma y cualquier otra palabra que puntúe en al menos igual a un umbral T, cuando sea comparada con la matriz de substitución. El puntaje umbral (T) es empleado para reducir el número de posibles aciertos. Una selección apropiada de w y T, y de la matriz de substitución es una forma efectiva de controlar el rendimiento y la sensibilidad de BLAST.

2. Extensión. Las semillas obtenidas en la fase anterior, deben ser extendidas con la intención de generar un alineamiento. Esto es hecho a medida que se inspeccionan los caracteres cercanos a la semilla en ambas direcciones y concantenándolos a la semilla hasta que se alcance un puntaje umbral X. Este umbral define cuanto puede ser reducido el puntaje, considerando de último el máximo valor. Luego de eso, el algoritmo regresa al mejor puntaje para obtener el alineamiento.

3. Evaluación. Los alineamientos generados en la fase de extensión deben ser evaluados con la intención de remover aquéllos no significativos. Los alineamientos significativos llamados *High Score Segment Pairs* (HSP) son aquellos cuyos puntajes son iguales o superiores a un umbral S. De igual forma, los grupos consistentes de HSP son generados sin incluir los HSP no solapados que están más cercanos a la misma diagonal (de programación dinámica); y son los que serán comparados contra un umbral final, conocido como el *e-value*, y sólo los alineamientos que superen este valor serán considerados. El e-value (valor esperado) da un estimado del número de alineamientos de secuencias no relacionados que tendrán valores muy altos, mientras que los valores más bajos del e-value, mayor es la probabilidad de estar frente a un alineamiento significativo.

En términos globales el comportamiento de BLAST para cada una de las secuencias se puede observar en la gráfica 4.5 y la descripción general del algoritmo que sigue está descrito en el pseudocódigo 4.1.

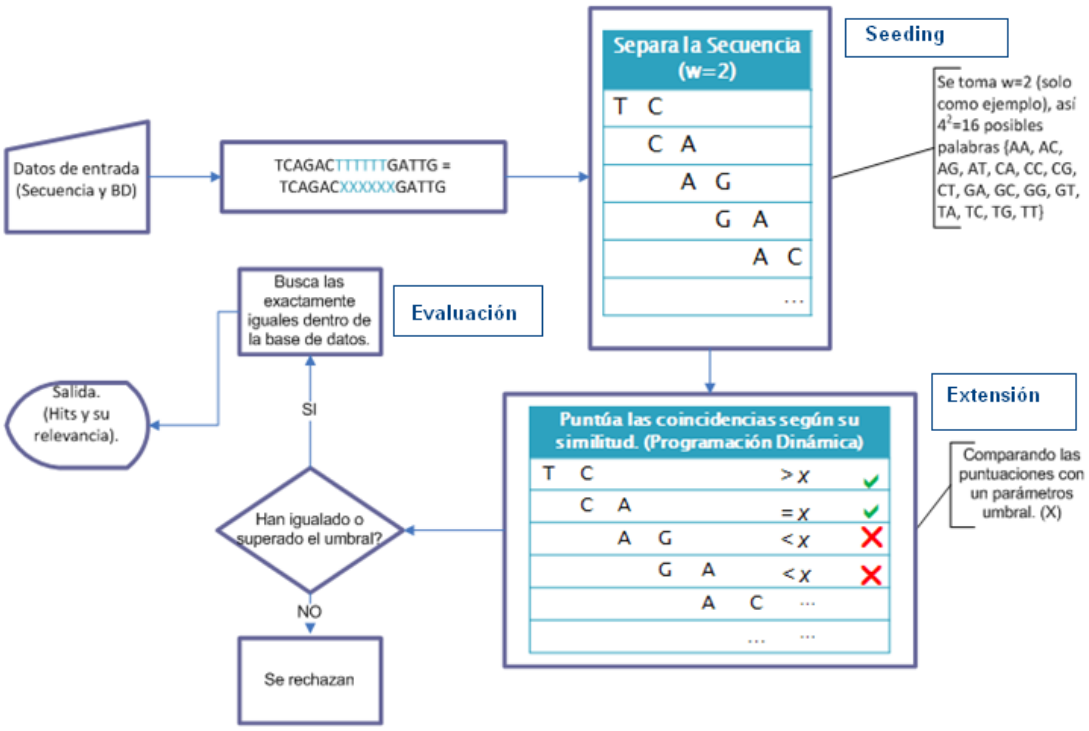

Figura 4.5: Flujo del Algoritmo de BLAST

```
1  /* Ejecuci\'on de BLAST */
2
3  Inicio
4
5  Para cada secuencia de consulta S,
6          Para cada base de datos D_i,
7                  Para cada secuencia D en D_i,
8                          Compara S con D usando BLAST;
9                          Actualiza las estadísticas;
10                 Fin Para
11         Fin Para
12         Reporta los resultados para le secuencia S;
13 Fin Para
14 Fin Pseudocodigo
```

Código fuente 4.1: Ejecución de BLAST

4.2.2. Gapped BLAST y PSI-BLAST

Gapped BLAST y PSI-BLAST son herramientas muy útiles provistas por el servidor de BLAST (conocidas como la versión 2.0)[2]. El BLAST original, no tomaba en cuenta el uso de *gaps* (espacios sin información), ya que acostumbraba buscar alineamientos sencillos de al menos longitud T y luego cada "acierto positivo" era extendido; en cambio, Gapped BLAST permite realizar dos alineamientos simultáneos no solapados de longitud T con una distancia A el uno del otro y posteriormente estos eran los que serían extendidos.

El algoritmo de Gapped BLAST permite la inserción y borrado de gaps dentro de los alineamientos que serán devueltos. Permitir esto significa que regiones similares no van a ser particionadas en varios segmentos. La puntuación de estos alineamientos con gaps tienden a reflejar más cercanamente las relaciones biológicas.

BLAST	GappedBLAST
ABCDE	ABCDE
ACD- -	A-CD-

Tabla 4.2: Alineamientos BLAST y GappedBLAST

Mientras que PSI-BLAST (*Position-Specific Iteraded BLAST*) provee una búsqueda automatizada del perfil, de fácil uso, la cual es una forma bastante sensitiva de buscar secuencias homólogas. El programa inicialmente realiza una búsqueda de gappedBLAST en la base de datos y luego el programa PSI-BLAST usa esta información.

Para cualquier alineamiento significativo que ha sido retornado en la fase anterior construye una matriz de puntuación con posiciones específicas, la que reemplazará la secuencia para la siguiente iteración de búsqueda dentro de la base de datos.

PSI-BLAST debe realizar iteraciones hasta que no se encuentren más alineamientos significativos. Hasta el momento PSI-BLAST puede ser usado solo para comparar secuencias de proteínas contra bases de datos de proteínas. Antes de la llegada de PSI-BLAST, sus técnicas ya habian sido usadas, pero un alto grado de experticia e intervención por parte de los biólogos era requerida.

PSI-BLAST, hace necesario empezar con una secuencia de consulta y una matriz de puntuación (por ejempo BLOSUM62, que es un tipo característico de matriz BLOSUM). Las secuencias homólogas son encontradas empleando los fundamentos de BLAST (alineando la base de datos con la secuencia de consulta). A diferencia de BLAST el *e-value* es empleado como criterio para insertar la secuencia encontrada dentro del perfil que se esta generando. De donde, un perfil (p1) es construído una vez que se ha pasado la secuencia por la matriz de puntuación. Se realiza una búsqueda nueva de homólogos empleando BLAST (ahora alineando la base de datos con el perfil obtenido), nuevamente el e-value determina si se inserta en el perfil y se genera un perfil (p2) con las secuencias aprobadas y la matriz de puntuación.

4.3. Paralelización en Alineamiento de Secuencias

A excepción de secuencias de consulta sencillas y pequeñas bases de datos, el procesamiento de BLAST consume gran cantidad de tiempo al contrastar secuencias biológicas completas contra bases de datos de cientos de miles de secuencias biológicas más en un sólo ordenador, para dar una idea, imaginen que tienen que comparar tu solito/a todas y cada una de las líneas de una obra literaria como El Quijote de Cervantes con una frase nueva que acabas de encontrar, sólo para saber si pertenece al mismo tiempo literario en el que fue escrita esta obra, (esto sin sumar cuanto te tomaría realizar esa consulta cada vez que le agregaran una página nueva a ese libro), sería mucho más rápido y si repartieras el trabajo entre personas de tu grupo de lectura favorito.

De esta forma, muchas ideas han sido planteadas para mejorar los tiempos de ejecución de BLAST. Las estrategias de cómputo paralelo y distribuido en clústers y grids ha sido de las más atractivas. Desde el punto de vista de la base de datos, existen dos enfoques básicos: la replicación de la información genómica en todos los nodos de procesamiento (nodos sencillos o clústers) junto con la segmentación de la secuencia de consulta; o la información genómica es separada en fragmentos disjuntos y la secuencia de consulta es introducida en una sola pieza para ser ejecutada la consulta en todos los sitios [12].

Mientras que la estrategia de replicacion de la base de datos está determinada por el paralelismo en la relación de los segmentos de la secuencia de consulta, la situación de fragmentar la base de datos lidera una situación un poco más complicada, dado que la ejecución de fragmentos más pequeños pudiera no generar el resultado correcto (y secuencial) si los parámetros estadísticos en tiempo de ejecución no están bien definidos. Aún más, igual que en otros problemas de computación paralela, cuando no hay una carga de trabajo pareja, los beneficios que vienen con estos enfoques se pierden.

4.3.1. Segmentación de la secuencia de consulta

La segmentación de la secuencia de consulta, fragmenta dicha secuencia de modo que cada nodo dentro de un clúster sea capaz de realizar la búsqueda con una fracción de la secuencia de consulta. Haciendo esto, muchas búsquedas de BLAST pueden ser ejecutadas con diferentes secuencias de consulta; ésta estrategia aplicada a un clúster típicamente replica la base de datos por completo en el sistema local de almacenamiento de cada nodo. Si la base de datos es mayor que la memoria del nodo, las búsquedas con segmentación de la secuencia de consulta sufre los mismos efectos adversos de e E/S que el BLAST tradicional. Cuando la base de datos cabe en la memoria local, efectivamente, la segmentación de la consulta puede conseguir casi escalabilidad linear para todos los tipos de búsqueda de BLAST.

ScalaBLAST

ScalaBLAST es una implementación paralela del algoritmo original de NCBI de alineamiento de secuencias (BLAST); puede ser usado para identificar secuencias rápidamente que son

similares a un conjunto de secuencias de proteínas suministradas por el usuario. Una lista de secuencias de consulta suele contener miles o millones de secuencias, cada una de las cuales se espera que se compare contra una base de datos inmensa de información genética y disponible públicamente, tal como la base de datos de proteínas no reduntante (nr). ScalaBLAST ha sido construído empleando el compilador intel y gnu y ha funcionado en arquitecturas tanto de 32 como 64 bits. Funciona eficientemente enviando los resultados y leyendo los archivos sobre sistemas de ficheros montados globalmente (Lustre), desde espacio de disco local (/scratch) y sobre sistemas de ficheros en red (/home)[28].

Scala BLAST alcanza un speedup considerable en entornos multiplocesador, gracias a dos métodos concurrentes:

1. Segmentando la lista de secuencia de consulta en pequeñas listas y gestionando que la búsqueda de BLAST sea llevada a cabo con cada lista generada por un determinado grupo de procesamiento.

2. Gestionando eficientemente el acceso a la base de datos genómica empleando Arreglos Globales[28], una implementación de una interfaz de memoria compartida que puede ser usada en arquitecturas de memoria compartida o distribuída.

Esta combinación única de separar la lista de secuencias de consulta y gestionar la memoria en una forma eficiente da cierta escalabilidad para trabajos de tamaño considerable. ScalaBLAST estuvo usando 1000 secuencias de consulta contra la base de datos nr. ScalaBlast emplea comunicación no bloqueante disponible en el pack de herramientas de Arreglos Globales para ocultar virtualmente todo el costo de comunicación dentro de entornos de clúster. ScalaBLAST ha sido portado a una variedad de arquitecturas, incluídos sistemas con memoria compartida y distribuída. Ha sido ejecutada con diferentes grados de interconexión: gigabit ethernet, myrinet, quadrics.

4.3.2. Segmentación de la base de datos

Cuando se habla de segmentación de la base de datos, se realiza la búsqueda independiente de cada fragmento en cada procesador o nodo, y los resultados son recolectados en un único fichero final de salida. Existen diferentes implementaciones de segmentación de bases de datos, la primera de ellas fue la de los NCBI BLAST, ya que implementa la segmentación de la base de datos a través de la búsqueda multihilo de forma que cada procesador dentro de un sistema SMP tiene asignada una porción diferente de la base de datos. Ésta estrategia también fue implementada en una versión comercial (de código cerrado) de la empresa TurboWorx, Inc. denominada TurboBLAST [6].

TurboBLAST

TurboBLAST provee una segmentación de la base de datos y una estrategia de distribución diseñada explícitamente para ser usadas en NoWs (Networks of Workstations). Empleando el

gestor y balanceador de carga propietario de TurboWorx denominado TurboHub, TurboBLAST puede adaptarse dinámicamente al entorno de clúster sobre el que se está ejecutando. Sin embargo por tratarse de una aplicación propietaria el uso es casi nulo. También está disponible otra versión de aplicación bioinformática que emplee segmentación de base de datos, denominada ParallelBLAST, que está compuesto por un conjunto de scripts que se pueden ejecutar sobre entornos PVM/Sun Grid Engine. Además de requerir un entorno PVM/SGE, difiere de mpiBLAST por no tener integrado directamente el NCBI Toolbox y no provee un mecanismo explícito de balanceo de carga.

TurboBLAST se basa en realizar un trabajo individual de BLAST donde especifica el número de secuencias de consulta de entrada que van a ser buscadas contra una o más bases de datos genómicas. TurboBLAST separa los trabajos de BLAST en muchas piezas pequeñas; que serán procesadas de forma paralela y posteriormente combinadas en un fichero final de salida. Aplicando esta estrategia, el fichero de salida presenta el mismo formato que el generado por NCBI BLAST en su sitio local.

Para conseguir coordinar las actividades en múltiples máquinas, lleva a cabo los siguientes procesos:

- Creando las tareas de BLAST, cada una de las máquinas necesita comparar un pequeño grupo de secuencias (aproximadamente entre 10 y 20 secuencias) contra una partición de tamaño modesto de la base de datos, de forma que la tarea pueda ser llevada a cabo sin generar paginación de disco.
- Aplica el programa estándar de NCBI BLAST para completar cada tarea.
- La integración de las tareas culmina en un fichero de salida unificado.

Este enfoque tiene la ventaja de garantizar la generación de un fichero de salida idéntico al obtenido en la versión serie de BLAST. Primero, el tamaño de cada tarea de BLAST es configurada de forma que el procesamiento en cada procesador sea lo más eficiente posible. Segundo, un conjunto de tareas lo suficientemente grande es creado de forma que todos los procesadores realicen trabajo útil y se obtenga un balanceo de carga casi perfecto.

La creación de tareas ocurre en dos pasos:

1. En el momento del envío del trabajo, se crean las tareas iniciales que buscan un grupo de 10 a 20 secuencias contra la*s* base*s* de dato*s*.
2. Si alguna de las tareas iniciales es demasiado grande para el procesamiento de BLAST en la máquina sobre la que está siendo llevada a cabo, la tarea es separada dinámicamente en sub-tareas lo suficientemente pequeñas para ser ejecutadas en esa máquina.

PackageBLAST

PackageBLAST [35] es una implementación de BLAST bajo entornos Grid, que puede operar bajo dos modos de ejecución: un primer modo, una secuencia larga sencilla es comparada con

una base de datos genómica. En este caso, cada máquina que figura como worker compara la misma secuencia con un conjunto de diferentes segmentos de la base de datos. La salida de éste modo es un reporte de BLAST parcial, que necesita ser procesado más adelante. En el modo 2, un conjunto de secuencias es comparado con la base de datos genómica completa. La salida de este modo es el reporte final de BLAST para cada secuencia comparada. En la figura 4.6[35], se puede observar las estrategias de ejecución que han diseñado.

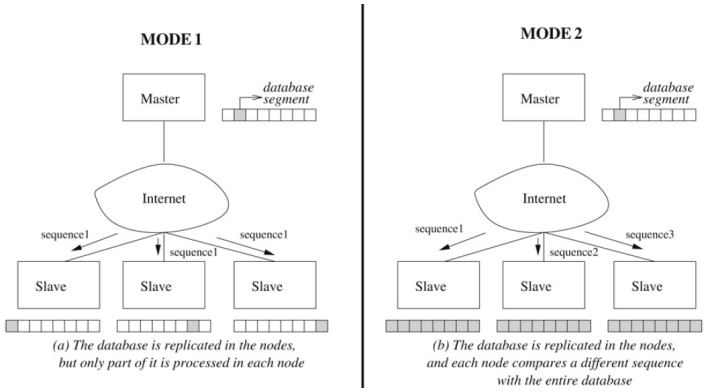

Figura 4.6: Modos de ejecución PackageBLAST.

Habiendo decidido que la base de datos genética sería fragmentada y replicada, seleccionan cuales unidades de trabajo serían procesadas por cada nodo worker del grid. Las unidades de trabajo son definidas como segmentos de base de datos (en el caso del modo 1) o secuencias de consulta (en el modo 2). Esto involucra muchas políticas de alocación de tareas, por lo que han desarrollado un framework que permite diferentes politicas de alocación que el usuario puede emplear según sus necesidades y entorno disponible.

MpiBLAST

mpiBLAST [12] es una herramienta de código fuente abierto, desarrollada el Los Alamos National Laboratory (LANL) para ejecutar NCBI-BLAST 4.2.1 en clústers de nodos de cómputo. El algoritmo tiene dos fases. Primero, la base de datos genética es segmentada y colocada en un medio de almacenamiento compartido. Posteriormente, la secuencia de consulta es distribuída a los nodos del clúster. Si el nodo no tiene el fragmento de la base de datos, se realiza una copia local. El método ha sido propuesto para asociar los fragmentos de datos con los nodos del clúster, tratando de minimizar el número de copias. Cuando los workers finalizan el procesamiento, ellos envían los reportes locales al máster, y éste habiendo recibido los reportes locales para le secuencia de consulta, los combina todos para crear el reporte final de BLAST.

El formateo de la base de datos en mpiBLAST es una función que es llevada a cabo de la siguiente manera:

- Analiza la correspondencia entre las secuencias y comprobar las opciones de la línea de comando.

- Lee la base de datos de entrada en formato FASTA; indexando la ubicación y longitud de cada secuencia dentro del archivo de entrada.
- Escribe un fichero temporal con las secuencias reordenadas para que las más largas estén primero.

  Llama a la función *formatdb* dentro del Main, que está encargada de crear el número deseado de fragmentos de la base de datos. Dado que esta función siempre agrega la siguiente secuencia en la lista temporal al fragmento más pequeño, garantiza que los tamaños de los fragmentos serán iguales.
- Crea y actualiza los archivos soportados por la aplicación, tales como .mbf, .dbs, y los .{n, p} del nombre del fichero.

Una vez que mpiBLAST ha formateado la base de datos, y dependiendo del tipo de base de datos que está siendo formateada, se generan 7 ficheros con extensiones diferentes, los cuales están descritos en las tablas 4.3 y 4.4.

Extensión	Contenido	Formato
nhr	Definición de Líneas	Binario
nin	Indices	Binario
nsq	Datos de la Secuencia	Binario
nnd	Datos GI	Binario
nni	Indices GI	Binario
nsd	Datos que no son GI	Binario
nsi	Indices de los datos que no son GI	Binario

Tabla 4.3: Extensión ficheros de la base de datos de nucleótidos formateada

Extensión	Contenido	Formato
phr	Definición de Líneas	Binario
pin	Indices	Binario
psq	Datos de la Secuencia	Binario
pnd	Datos GI	Binario
pni	Indices GI	Binario
psd	Datos que no son GI	Binario
psi	Indices de los datos que no son GI	Binario

Tabla 4.4: Extensión ficheros de la base de datos de proteínas formateada

Asimismo, el trabajo correspondiente a cada uno de los procesos, dependiendo de si es máster o worker, está descrito en los algoritmos presentados a continuación que han sido tomados de [12], de esta forma se puede visualizar el comportamiento teórico que habría de presentar la aplicación.

```
1  /* Máster de mpiBLAST */
2
3  Inicio
4  resultados es el conjunto de resultados de BLAST.
5  F = {f_1, f_2, ...} es el conjunto de fragmentos de la base de datos.
6  SinBuscar ⊆ F es el conjunto de fragmentos de la base de datos en los que no se ha buscado.
7  SinAsignar ⊆ F es el conjunto de fragmentos de la base de datos en los que no se han asignado.
8  W = {w_1, w_2, ...} es el conjunto de workers disponibles.
9  D_i ⊆ W es el conjunto de workers que tienen el fragmento f_i en almacenamiento local.
10 Distribuídos = {D_1, D_2, ...} es el conjunto de Datos para cada fragmento.
11 Requiere que: |W| ≠ 0
12 Asegúrese que: |SinBuscar| = 0
13 SinBuscar ← F
14 SinAsignar ← F
15 resultados ← vacio
16 Envía las secuencias a los workers vía Broadcast
17 Mientras estadoActual ≠ 0 haga
18 Recibe un mensaje desde un worker w_j
19 Si mensaje es un estado de solicitud entonces
20 Si |SinAsignar| = 0 entonces
21         Envía al worker w_j el estado BÚSQUEDA_COMPLETA
22 De lo contrario
23         Envía al worker w_j el estado de BUSCAR_FRAGMENTO
24 Fin Si
25 De lo contrario Si mensaje es una solicitud de fragmento entonces
26         Encuentra f_i de forma que sea el min D_i ∈ Datos_Distribuidos |D_i| y f_i ∈ SinAsignar
27         Si |D_i| = 0 entonces
28                 Agrega w_j a D_i
29         Fin Si
30         Remueve f_i de SinAsignar
31         Envía la asignación del fragmento f_i al worker w_j
32 De lo contrario Si mensaje es un conjunto de resultados para el fragmento f_i entonces
33         Combina mensaje con resultados
34         Remueve f_i de SinAsignar
35 Fin Si
36 Fin Mientras
37 Imprime resultados
```

Código fuente 4.2: Máster de mpiBLAST

El proceso máster usa un algoritmo ambicioso (descrito en el pseudocódigo 4.2) para determinar cuáles fragmentos asignar a cada worker. Inicialmente, si un worker inactivo tiene en su almacenamiento local un fragmento sobre el que aún no se ha realizado la búsqueda, el worker es asignado para realizar el alineamiento con ese fragmento. Si el worker no tiene un único fragmento, se le asigna buscar en aquél que esté repartido el menor número de veces entre los demás workers. Finalmente, si un worker inactivo no tiene fragmentos en los que aún no se ha buscado, se le indica que copie un fragmento no utilizado existente en la menor cantidad de workers. El conjunto de fragmentos que se esté copiando en ese instante es controlado por el máster para prevenir la copia duplicada de asignaciones a workers diferentes.

```
1  /* Worker de mpiBLAST */
2
3  Inicio
4  secuencias ← Recibe las secuencias de consulta desde el máster
5  estadoActual ← Recibe el estado desde el máster
6  Mientras estadoActual ≠ BÚSQUEDA_COMPLETA haga
7          fragmentoActual ← Recibe una asignación de fragmento desde el máster
8          Si fragmentoActual no está en el almacenamiento local entonces
9                  Copia fragmentoActual al almacenamiento local
10         Fin Si
11         resultados ← BLASTsecuencias,fragmentoActual
12         Envía resultados al máster
13         estadoActual ← Recibe el estado desde el máster
14 Fin Mientras
```

Código fuente 4.3: Worker de mpiBLAST

El algoritmo seguido por el worker se describe en el pseudocódigo 4.3, donde una vez que el worker completa su búsqueda, reporta los resultados al máster. El máster es el encargado de fusionar todos los resultados de cada worker y ordenarlos de acuerdo al puntaje que traen. Una vez que todos los resultados han sido recibidos, son escritos a un fichero de salida especificado por el usuario empleando las funciones de salida de BLAST de la librería de desarrollo de NCBI. Este enfoque de generar resultados combinados le permite a mpiBLAST producir resultados en cualquier formato soportado por NCBI-BLAST, incluyendo XML, HTML, texto delimitado por espacios en blanco, y ASN.1. Todo el proceso que realiza mpiBLAST, se puede ver a modo general en la figura 4.7.

En el trabajo de Feng [14] han descrito el comportamiento de mpiBLAST en entornos de supercomputación, clasificando cada una de las fases involucradas en la ejecución, tal como se muestra en la gráfica 4.8.

Adicionalmente han mencionado la influencia que podrían tener en la ejecución factores como:

- El tipo de base de datos biológica con la que se está trabajando;
- El tamaño de la secuencia de consulta;

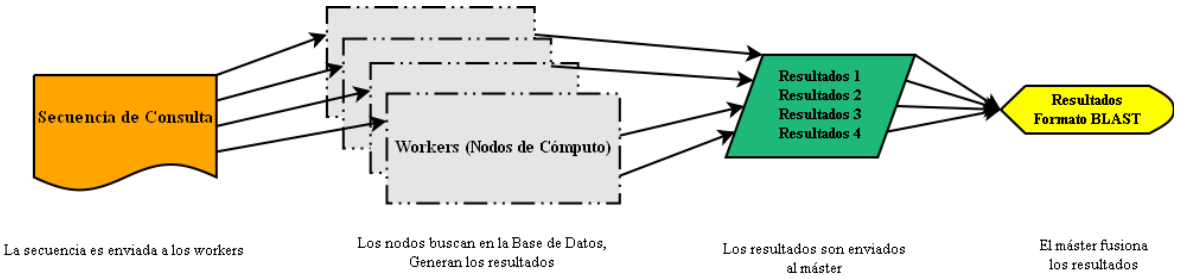

Figura 4.7: Algoritmo de mpiBLAST

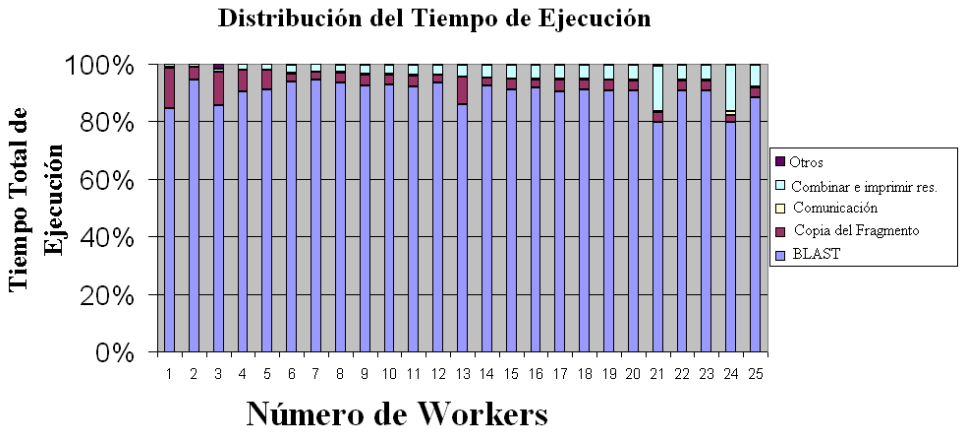

Figura 4.8: Fases de mpiBLAST

- El tamaño de la base de datos biológica contra la que se realiza la consulta;
- El número de workers disponibles; y
- El nivel de similitud entre familias biológicas de las bases de datos y las secuencias.

Que repercutirían en el tiempo total de ejecución de la aplicación; de esta forma constituye un enfoque interesante que sustenta nuestra intención de análisis de rendimiento durante este trabajo de investigación.

Capítulo 5

Experimentos

"El genio es un uno por ciento de inspiración, y un noventa y nueve por ciento de transpiración."

Thomas Alva Edison(1847-1931).

5.1. Introducción

Cuando se desea diseñar el grupo de experimentos necesarios para verificar la hipótesis que se plantea en cada uno de ellos, es primordial que una vez que se ha reconocido y delimitado el problema, se proceda a seleccionar aquéllos factores involucrados y los niveles que se van a a tomar en consideración, ya que a partir de estos de va a definir cual es nuestra variable de respuesta esperada. Variable que lleva la motivación global por la que se están realizando los experimentos. Aún así una vez que éstas variables han sido establecidas, es necesario determinar el diseño experimental que habrá de llevarse a cabo para responder la pregunta que nos hemos planteado en un inicio. Luego se dedicará gran parte de nuestro tiempo en la realización de la experimentación para la obtención de los datos, y su posterior análisis de forma que seamos capaces de emitir las conclusiónes y/o recomendaciónes finales; además, suele ser necesario realizar un estudio en que se confirme si los resultados y la conclusión a la que se ha llegado es cierta o no.

El objetivo detrás de los experimentos descritos en este capítulo giran entorno al análisis de rendimiento de la aplicación mpiBLAST, donde se persigue con cada uno de ellos identificar aquéllos factores significativos en el consumo de tiempo dentro de la ejecución de la aplicación. Como primera aproximación se ha optado por realizar un análisis de rendimiento estático post-mortem, que se espera dinamizar en trabajos futuros, monitorizando el comportamiento de la aplicación dentro de un entorno paralelo controlado, un clúster Beowulf dedicado de 32 nodos.

5.2. Escenario

El proceso de evaluación de rendimiento para la aplicación mpiBLAST requiere cierto conocimiento de la interacción que presenta ella en un determinado entorno paralelo. Dado que nuestro principal objetivo es conseguir la mejor configuración de los parámetros involucrados en la ejecución de mpiBLAST en aras de conseguir el menor tiempo de ejecución posible, una vez que la hemos seleccionado como objeto de estudio, es necesario alojarla en nuestras máquinas paralelas con la finalidad de poder conocerla y evaluarla.

Como principal requisito para ejecutar mpiBLAST en el entorno paralelo (clúster de ordenadores) es necesario contar con alguna versión de MPI para poder compilar la aplicación y posteriormente utilizarla. En nuestro caso contamos con un cluster Beowulf con 32 nodos Intel Single-core a 3.00 GHz, 1 gigabyte de RAM por nodo de cómputo, 80 gigabytes de disco de almacenamiento, red Gigabit/Ethernet, sistema de ficheros NFS y configuración RAID1; en el cual hemos instalado la versión de MPI mpich2 [16] que permite realizar el traceado con instrucciones MPE (para luego visualizarlas como análisis inicial de mpiBLAST), y hemos compilado las versiones disponibles en: http://www.mpiblast.org, mpiBLAST-1.4 y mpiBLAST-1.5-PIO, que son la última versión estable y la que versión más reciente con entrada y salida paralela, respectivamente.

Para instalar mpiBLAST es necesario desempaquetar el archivo de código fuente (disponible en: www.mpiblast.org) en un fichero temporal. Se debe disponer de una versión actual de la caja de herramientas de NCBI (NCBI Toolbox), facilmente descargable desde el sitio web de NCBI. Una vez que la caja de herramientas ha sido instalada apropiadamente, se procede a realizar el `./configure` y posteriormente `make` y `make install`.

La aplicación mpiBLAST debe ser configurada de acuerdo a las librerias y herramientas disponibles en el entorno paralelo sobre el que se va a ejecutar, es necesario identificar el MPI con el que se está trabajando, junto con la dirección donde se van a instalar los archivos binarios y la dirección donde se ha instalado el pack de herramientas de NCBI.

Una vez que se ha instalado mpiBLAST, en la carpeta local están disponibles los binarios:

- `mpiformatdb`, empleado para formatear la base de datos biológica en el número de fragmentos necesario;
- `mpiblast`, que ejecuta la consulta de mpiBLAST como tal;
- `mpiblast_cleanup`, encargada de limpiar los "residuos" de base de datos que han quedado en las carpetas locales de cada uno de los worker.

5.3. Análisis de los Factores

En la siguiente sección detallamos cada uno de los experimentos realizados para analizar y evaluar la herramienta bioinformática mpiBLAST como aplicación paralela basada en MPI.

Con la siguiente experimentación se espera responder las interrogantes sobre la existencia de ineficiencias de la aplicación, carencia de paralelismo y/o poca explotación de los recursos disponibles. De manera que sea posible verificar los posibles cuellos de botella existentes dentro de la ejecución que pudieran ser sintonizados para mejorar el rendimiento que presenta mpiBLAST.

Paralelismo de mpiBLAST

Uno de los principales problemas de las aplicaciones paralelas, es el nivel de escalabilidad que éstas pudieran tener en clústers de ordenadores, para ello es necesario comprobar las ventajas que presenta el paralelismo dentro de la aplicación dentro de un ambiente paralelo específico.

Hipotesis:

"mpiBLAST no explota el paralelismo para un tamaño de problema demasiado pequeño".

Esto debido a que el no sacar provecho del paralelismo del diseño de la aplicación o del entorno paralelo sobre el que se trabaja, se convierte en el principal "talón de Aquiles" de las aplicaciones paralelas. Es por ello que nos hemos planteado comprobar el nivel de ganancia que presenta o no la aplicación al ejecutarla en su versión paralela. Usualmente por problemas de diseño o limitaciones del entorno, las aplicaciones paralelas tienden a tener peor rendimiento que cuando han sido ejecutadas secuencialmente, esto debido a cierto nivel de *overhead* que se pudiera estar presentando o ineficiencias en el paradigma de paralelismo empleado al momento del diseño.

Si recordamos cuando hablamos de una aplicación como mpiBLAST, estamos tratando con una clase de aplicación diseñada bajo un enfoque *embarrasingly parallel*, en el que siguiendo un paradigma Máster/Worker el trabajo es repartido a los workers, que se encargan de realizar la actividad de forma paralela y luego enviar los resultados obtenidos al máster. Sin embargo, no todas las veces que se emplea este paradigma aparentemente sencillo se consigue un mejor desempeño que si se continuara empleando una sola máquina.

Proceso:

Para determinar el grado de ventaja existente al usar o no BLAST en su versión paralela, es necesario partir de la comparación del parámetro *tiempo de ejecución*, con la intención de poder observar cuantitativamente si existe o no una mejora entre ambos enfoques. Sin embargo, para comprobar esto es necesario realizar las mediciones pertinentes de las ejecuciones secuenciales de BLAST bajo un solo ordenador y determinar el speedup existente con mpiBLAST.

Para lograr esto, hemos decidido ejecutar en un cluster, las consultas a BLAST (la versión secuencial) con las bases de datos *mito.nt* de 934 KB, la *yeast.nt* de 3.635,20 KB, *drosoph.nt* de 33.280,00 KB y la *NT* con 7.130.316,80 KB (todos estos tamaños son de las bases de datos comprimidas en un fichero *.gz*), con la secuencia *e_chrysanthem.fas* de 300KB de longitud. Los experimentos serán llevados a cabo en un cluster Beowulf compuesto de nodos de cómputo Intel Pentium 4 a 3.00Ghz, con 1 GB de memoria RAM, 80 GB de disco duro SATA, red Gigabit Ethernet, sistema de ficheros paralelo con RAID1. Posteriormente se calculará el *speedup* de mpiBLAST para esa misma carga descrita.

Se han observado los tiempos de ejecución para 1, 4, 8, 16, 32 procesadores respectivamente, los cuales se encuentran reflejados en la gráfica 5.1. Se controlará el tiempo de ejecución tanto de la versión serie como de la versión paralela y se podrá definir hasta qué punto favorece el paralelismo de mpiBLAST la realización de dichas consultas.

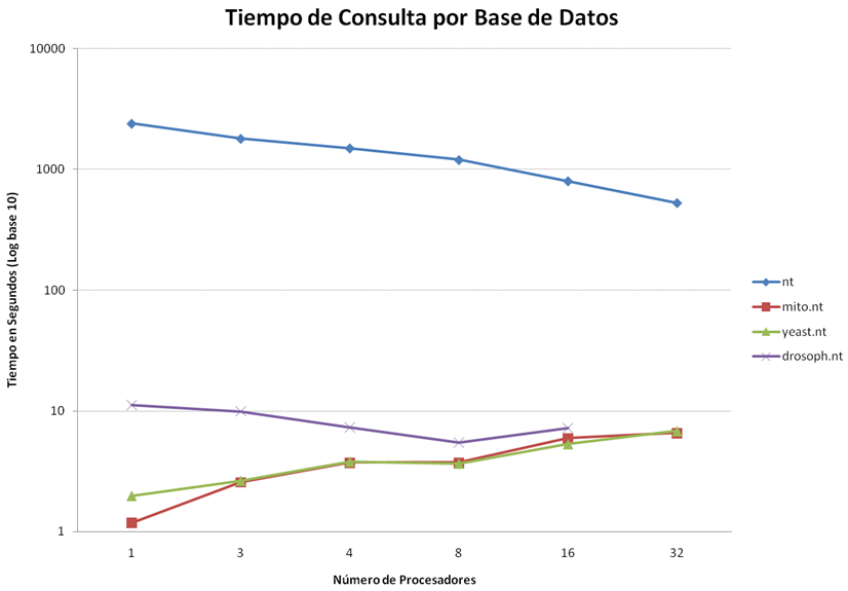

Figura 5.1: Tamaño base de datos vs. Número de Workers

Inicialmente obtenemos un speedup de 4,5 para 32 procesadores con la base de datos NT, tal y como se puede ver en la figura 5.3, en la que se encuentran los cálculos de speedup para cada una de las bases de datos que hemos planteado, es posible afirmar la existencia de ganancias de rendimiento al emplear mpiBLAST.

Si contrastamos los speedup obtenidos con lo que sería el speedup ideal (ver figura 5.3, podemos observar que las ganancias, aunque pocas, se consiguen a medida que se aumenta el tamaño del problema. Lo que queda por determinar es por qué el speedup es tan bajo para este tipo de ejecuciones.

Los tiempos de ejecución están directamente relacionados con la carga de trabajo que se le está proporcionando al algoritmo, *si el tamaño del problema es demasiado pequeño, el paralelismo no brinda ventaja alguna, en cambio si el tamaño del problema se vuelve significativo, a medida que se incorporan procesadores que se comportarán como workers y realizar la búsqueda de forma simultánea el tiempo de ejecución reduce considerablemente.*

Adicionalmente, con los datos obtenidos en este experimento, somos capaces de calcular el nivel de eficiencia que tiene la aplicación (ver figura 5.4), lo que se consigue observar es que la aplicación a medida que el número de procesadores aumenta se presenta alguna situación dentro de la ejecución que deja entrever que no se está aprovechando el paralelismo del sistema como tal. Situación que se espera descubrir en experimentos siguientes.

Conclusiones:

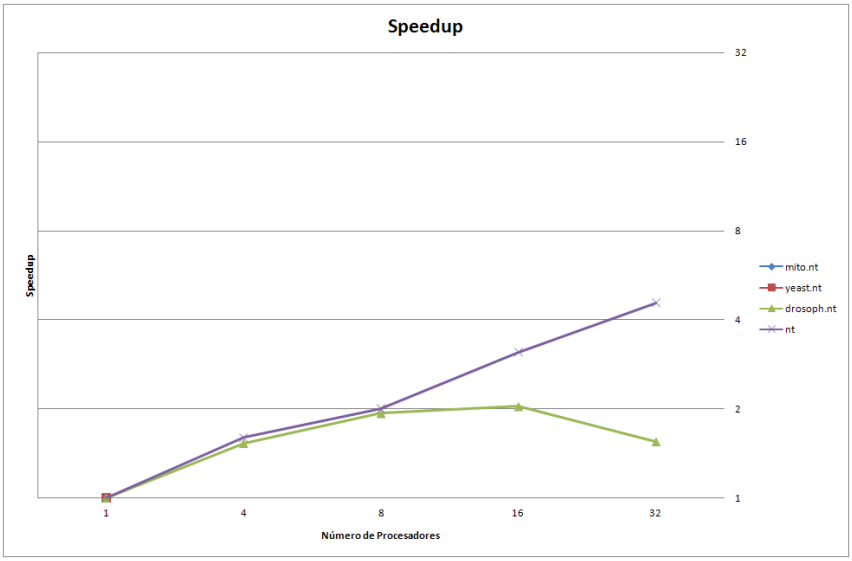

Figura 5.2: Speedup de mpiBLAST

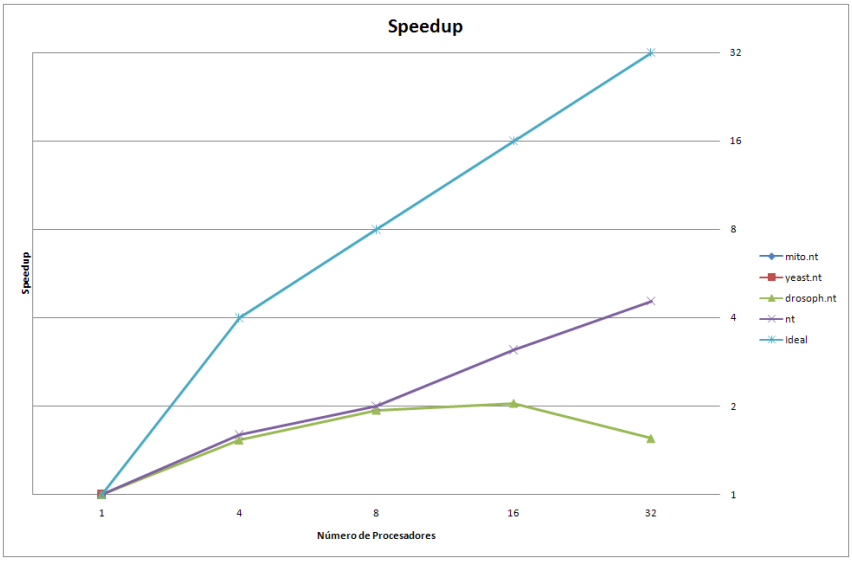

Figura 5.3: Speedup Ideal vs. Speedup de mpiBLAST

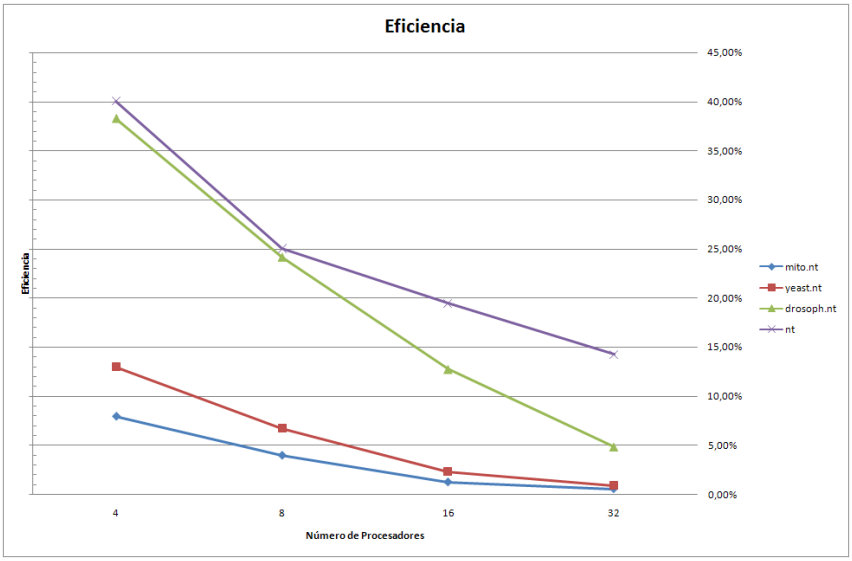

Figura 5.4: Eficiencia

Se puede observar en las gráficas presentadas que a medida que aumenta el tamaño de la base de datos biológica, las exigencias para el algoritmo secuencial se hacen cada vez mayores y requiere cada vez más cantidad de recursos. Sin embargo, a medida que se aumenta el número de procesadores sobre los que se ejecuta la aplicación de forma paralela (aún cuando el tamaño de la base de datos se incremente), y manteniendo fijo un fragmento para cada procesador se obtiene una ganancia considerable en tiempo de ejecución. Por lo cual se puede concluir que a medida que el tamaño del problema crece es posible observar la ventaja de realizar las consultas de forma paralela. Esto debido al tamaño de los datos de entrada con los que se está trabajando, que al introducir el concepto de fragmentación es posible llevar a cabo varias tareas de forma simultánea sin entorpecer unas con otras en las ejecuciones.

Con esto hemos podido determinar que el tamaño de base de datos biológica con la que se esté trabajando va a determinar el nivel de ganancia que se puede obtener al realizar las consultas de forma paralela. Con ello a medida que el tamaño de la base de datos aumente y se disponga de un clúster de 32 nodos de cómputo se podrá reducir el tiempo de consulta en aproximadamente 75 %. Bajando de una media de 40 minutos de ejecución en el BLAST secuencial a apenas una media de 8 minutos en la ejecución en paralelo.

Fases de mpiBLAST

Hipotesis:

"mpiBLAST está desaprovechando el paralelismo dentro de su diseño en algún punto durante la ejecución".

Igual como se determinó en el experimento anterior que mpiBLAST escala según el tamaño del problema, ahora se plantea determinar si como aplicación paralela presenta puntos de ineficiencia; puntos que se determinan una vez que se ha realizado el análisis de rendimiento de la

aplicación.

Tal como ha sido descrito en capítulos anteriores, este proceso se puede llevar a cabo de diferentes maneras pero el objetivo continúa siendo el mismo, el por qué de esto es que normalmente se busca identificar puntos de ineficiencia en la aplicación en los que se está desaprovechando el paralelismo inherente al entorno paralelo sobre el cual está siendo ejecutada, e identificar la causa de su presencia para generar una conclusión adecuada para posteriores mejoras en la aplicación.

Proceso:

Se ha decidido identificar la cantidad de tiempo de ejecución que invierte la aplicación en las diferentes fases que la componen, una vez que ha sido formateada la base de datos en un determinado número de fragmentos y se desea realizar una consulta contra la base de datos de secuencias biológicas disponible, es importante tomar en consideración la importancia que refieren cada una de las etapas por las que el programa ha de pasar para generar el fichero de salida con la información de la secuenciación. Basándonos en parte en la descripción que plantean los diseñadores de la aplicación se han identificado las siguientes fases significativas:

- Repartición de los fragmentos de la base de datos a cada uno de los workers.
- Proceso de búsqueda de la secuencia de consulta en el(los) respectivo(s) fragmento(s) asignado(s) a cada worker.
- Envío de los resultados por parte de los workers hacia el máster, empleando MPI.
- Unión e impresión de los resultados en el fichero de salida.

Para ello se han tomado como puntos de medición aquéllos en los que la aplicación está ejecutando cada una de las funciones anteriormente descritas, consiguiendo de esta manera una primera impresión de la cantidad de tiempo que se está perdiendo dentro de la ejecución.

Presentando un tiempo medio de ejecución para 7 workers y 7 fragmentos (uno para cada worker) igual a 627,0916 segundos, se puede determinar una distribución del tiempo total en cada una de las fases involucradas en mpiBLAST representada en la figura 5.5 en el que se puede observar que la mayor parte del tiempo de la ejecución se está consumiendo en la repartición de fragmentos para cada worker, esto debido a la estrategia de repartición diseñada para el algoritmo, en la que el worker no inicia su trabajo hasta que ha recibido todos los fragmentos sobre los que le corresponde realizar la búsqueda.

Otra información que se pudo obtener durante la experimentación es que el tiempo medio de búsqueda para cada worker es practicamente similar, como se puede observar en la tabla 5.1 con lo que se concluye que el algoritmo de BLAST independientemente de los datos con los que está trabajando consigue realizar la búsqueda en el fragmento asignado en un tiempo muy similar para cada worker.

De igual forma cuando se aumenta en la ejecución el número de nodos a emplear a 32, podemos observar el porcentaje de tiempo empleado en cada una de las fases en la figura 5.6. De

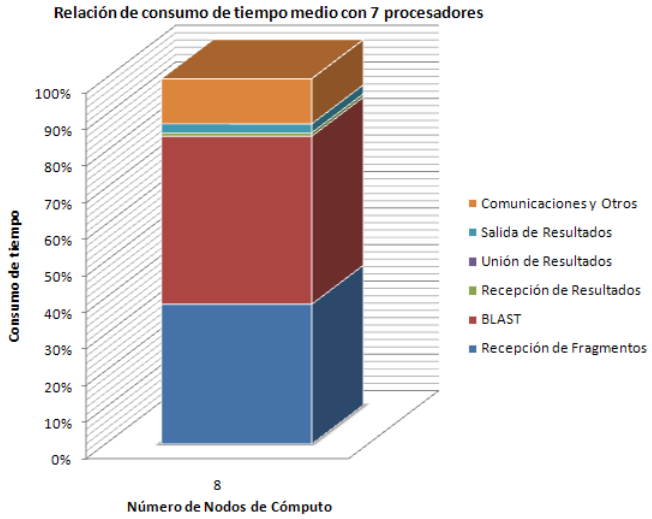

Figura 5.5: Fases de mpiBLAST con siete Fragmentos y siete workers.

forma que se puede percibir que a medida que la base de datos se fragmenta en un mayor número de piezas, cada fragmento disminuye en su tamaño. Por lo tanto el tiempo de distribución de los fragmentos y de búsqueda dentro de cada uno se ve reducido, como se ve reflejado en la tabla 5.1. De forma que se puede observar un mayor solapamiento entre cómputo y distribución de datos, que básicamente es lo que se persigue al momento de sacar provecho de las características de paralelismo disponibles en la aplicación.

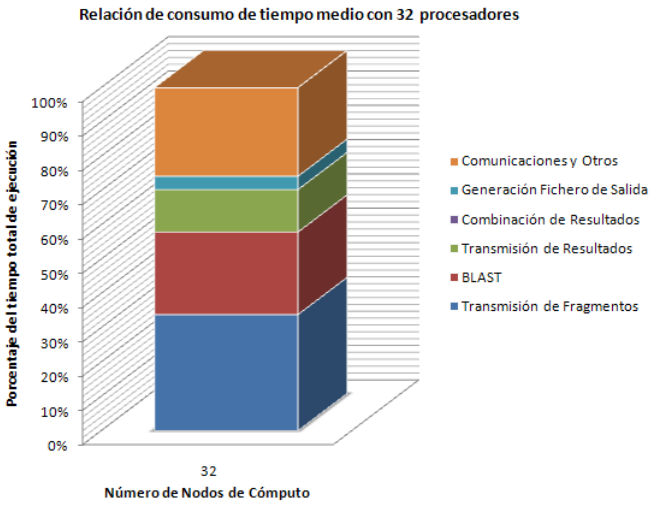

Figura 5.6: Fases de mpiBLAST con 31 fragmentos y 32 nodos.

Hemos podido determinar la estrategia de repartición de fragmentos por parte del máster hacia los workers, visualizando la traza de ejecución reflejada en la figura 5.7, en la que los cuadros pequeños de tamaño más pequeño reflejan el momento en el que el worker está recibiendo el fragmento de base de datos que le corresponde. De esta forma hemos podido confirmar que parte del proceso de repartición afecta significativamente el tiempo global de ejecución, ya que

Rank Worker	Tiempo Medio *sg*
2	276,34
3	277,07
4	288,39
5	280,23
6	271,22
7	266,69
8	279,46
Rank Worker	**Tiempo Medio *sg***
2	108,76
3	119,49
4	114,06
5	115,85
6	116,56
7	110,08
8	119,69
9	112,19
10	114,18
11	113,41
12	108,36
13	114,89
14	122,20
15	116,61
16	110,41
17	111,93
18	109,97
19	116,18
20	113,21
21	108,93
22	113,75
23	118,42
24	117,31
25	113,36
26	113,32
27	116,48
28	113,97
29	107,42
30	111,36
31	117,75

Tabla 5.1: Tiempo medio de búsqueda BLAST con para 8 y 32 nodos, con 7 y 31 fragmentos respectivamente.

el último worker debe esperar que todos los fragmentos anteriores sean repartidos, para el recibir el suyo y comenzar el proceso de búsqueda con BLAST.

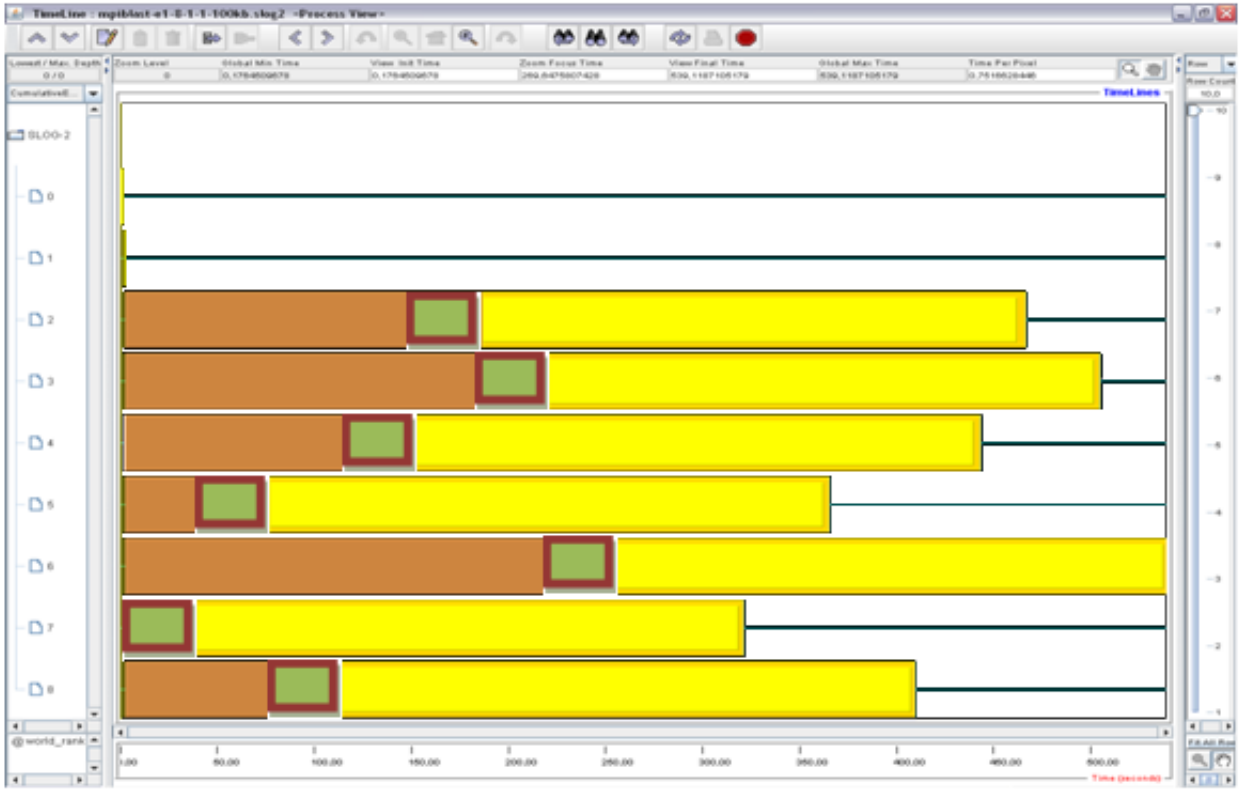

Figura 5.7: Distribución de Fragmentos

N° de Nodos	mito.nt (sg)	yeast.nt (sg)	drosoph.nt (sg)	NT (sg)
8	4,10	3,65	5,76	627,09
32	6,56	6,82	36,10	465,74

Tabla 5.2: Tiempo medio total de búsqueda en mpiBLAST para 8 y 32 nodos con varias bases de datos.

Esta estrategia, una vez que se ha revisado el código fuente igualmente muestra que los diseñadores se han planteado repartir la totalidad de fragmentos a todos los workers para que luego ellos comiencen el procesamiento. Y si alguno llegara a terminar antes, y requiriera un fragmento adicional, el algoritmo busca aquél fragmento en que otro worker no haya buscado y se lo transmite al worker desocupado.

Los tiempos medios totales de las ejecuciones para 8 y 32 nodos respectivamente, se encuentran representados en la tabla 5.2, donde se puede percibir la diferencia existente en la duración de las ejecuciones cuando se varía el número de workers y el tamaño de las bases de datos.

Conclusiones:

Gracias a lo que se puede observar en las primeras gráficas en las que se presentan la repartición de un fragmento para cada worker, y basándonos en los resultados obtenidos luego de la monitorización, hemos podido observar que las ejecuciones de mpiBLAST bajo un entorno con memoria y procesamiento limitados (en este caso a 32 nodos con 1 GB de RAM cada uno) tenemos una serialización en la repartición de los fragmentos entre los workers involucrados, es por ello que a medida que se aumenta el número de workers se obtiene un mejor desempeño de la aplicación, esto porque el tamaño del fragmento se ve reducido y con ello consigue ser alojado en la memoria local de cada uno de los nodos de cómputo. Sin embargo, la fase de repartición al no tener un sistema de entrada/salida paralelo presenta ciertas ineficiencias al momento de hacer llegar los fragmentos a cada nodo. Lo cual cuando se carece de la totalidad de los procesadores, pudiera generar un cuello de botella considerable en el que se esté perdiendo entre el 35 y 40 % del tiempo total de ejecución.

Algo que vale la pena mencionar es que el tiempo de consulta para cada fragmento en cada nodo aún cuando poseen datos diferentes es prácticamente similar, esto debido a la estrategia empleada por el algoritmo de recorrer la totalidad del fragmento buscando posibles similitudes. Aparentemente no existe dependencia entre los datos distribuídos en cada fragmento. Situación que va a ser comprobada más adelante.

Rendimiento de la memoria con mpiBLAST

Hipotesis:

"La cantidad de memoria disponible restringe el desempeño de mpiBLAST".

Recordando lo que hemos venido señalando a todo lo largo de la investigación el tamaño de las bases de datos con los que se están enfrentando aplicaciones bioinformáticas como mpiBLAST está convirtiendo, aparentemente, la capacidad de memoria disponible en los nodos de

cómputo de las máquinas paralelas en uno de los principales cuellos de botella dado que si se está trabajando con tamaños muy por encima de esta capacidad, la velocidad con que se accede a los datos se ve afectada por una posible paginación a disco.

Proceso:

Es por ello que se ha considerado, medir el grado de influencia existente en el tiempo de ejecución cuando se ha incrementado exponencialmente el tamaño de la base de datos sobre la que se va a llevar a cabo la búsqueda y la cantidad de memoria disponible empleando para la consulta un mayor número de nodos de cómputo. De esta forma se espera comprobar frente a que tipo de limitaciones se encuentra mpiBLAST. Para ello se ha considerado una base de datos de la cual se han generado ficheros de tamaños variables hasta llegar a los 2 gigabytes de datos, y se contrastará con el tiempo de ejecución promedio que tome realizar la consulta.

Se ha realizado el experimento con la base de datos NT($\approx$ 6.8 GB) y la secuencia de consulta *e_chrysantem.fas* de 300 KB, con la finalidad de observar el consumo total de memoria al variar el número de workers (de forma que sea demasiado grande para ser alojada en la memoria de un nodo de cómputo).

Inicialmente con la base de datos NT segmentada en siete piezas se realiza la consulta con 8 nodos de cómputo del cluster (descrito como entorno en páginas anteriores); cada fragmento ocupa poco más de 1GB (mpiBLAST permite formatear la base de datos como fichero comprimido con la extensión `.gz`), no dejando espacio libre suficiente en memoria para el búffer de resultados que se genera a medida que avanza la ejecución. Lo que obliga al sistema a tener que paginar en disco para poder alocar todos los elementos necesarios durante la ejecución.

Para la gráfica 5.9 se han seleccionado 7 nodos, tomando en consideración aquéllos que habían recibido su fragmento en los primeros 25 segundos de ejecución, y los que lo habian recibido su fragmento a 50, 100 y 150 segundos, de forma que se viera el comportamiento que adoptaba la memoria a medida que éstos eran cargados. Para el caso del nodo 7, 18 y 15, se han seleccionado por tener el mayor uso de memoria durante la ejecución, esto debido a que ya que los fragmentos no reciben igual número de secuencias de acuerdo a su tamaño, existen algunos fragmentos de la base de datos que tendrán secuencias de mayor longitud que otras y habrán de consumir memoria durante mucho más tiempo. Entiéndase que la base de datos se ha fragmentado en 7 y 31 piezas, y se ha realizado la ejecución con 8 y 32 nodos en cada caso.

Nótese la presencia de dos procesos adicionales en ambas gráficas que son el *Writer* y el *Scheduler*, que son aquéllos procesos ejecutados dentro del nodo máster que se encargan de escribir el fichero de salida en formato BLAST y de gestionar la distribución de los fragmentos a cada worker respectivamente.

Si se monitorea el consumo de memoria en dichos procesos (el de escritura y, el de gestión del fichero de salida y fragmentos de la base de datos), los datos obtenidos están representados en las gráficas 5.8 y 5.9, en los que se visualiza que el consumo de memoria no supera los 20 MB mientras que los workers dependiendo del tamaño de la secuencia alcanzan medias por encima a los 500MB, esto se debe a que no se está alojando gran cantidad de información durante tiempo

de ejecución.

Asimismo, se tomaron las mediciones del consumo de memoria por parte del máster y los workers en un sistema con 32 nodos, manteniendo el tipo de la base de datos y secuencia de consulta fijo. Y se puede percibir que el nodo que más consume no alcanza a superar el máximo de memoria física disponible y de forma similar que como sucede con 8 nodos, la carga de trabajo y consumo de memoria para el máster se mantiene extremadamente baja.

Conclusiones:

Dado el tamaño considerable que han adoptado las bases de datos biológicas, la disponibilidad de recursos en cuanto a memoria para alojar el fragmento, se convierte en un factor determinante en cuanto a uso eficiente de los recursos, porque de lo contrario se está generando un *overhead* ocasionado por las repetidas veces que el nodo de cómputo se ve obligado a paginar sus datos a disco debido a la ausencia de espacio libre para alojar en memoria.

Será interesante comprobar si aumentando el número de fragmentos, existe alguna ventaja para los clusters con poca memoria disponible, de manera que se trabaje con piezas que consuman menos de memoria.

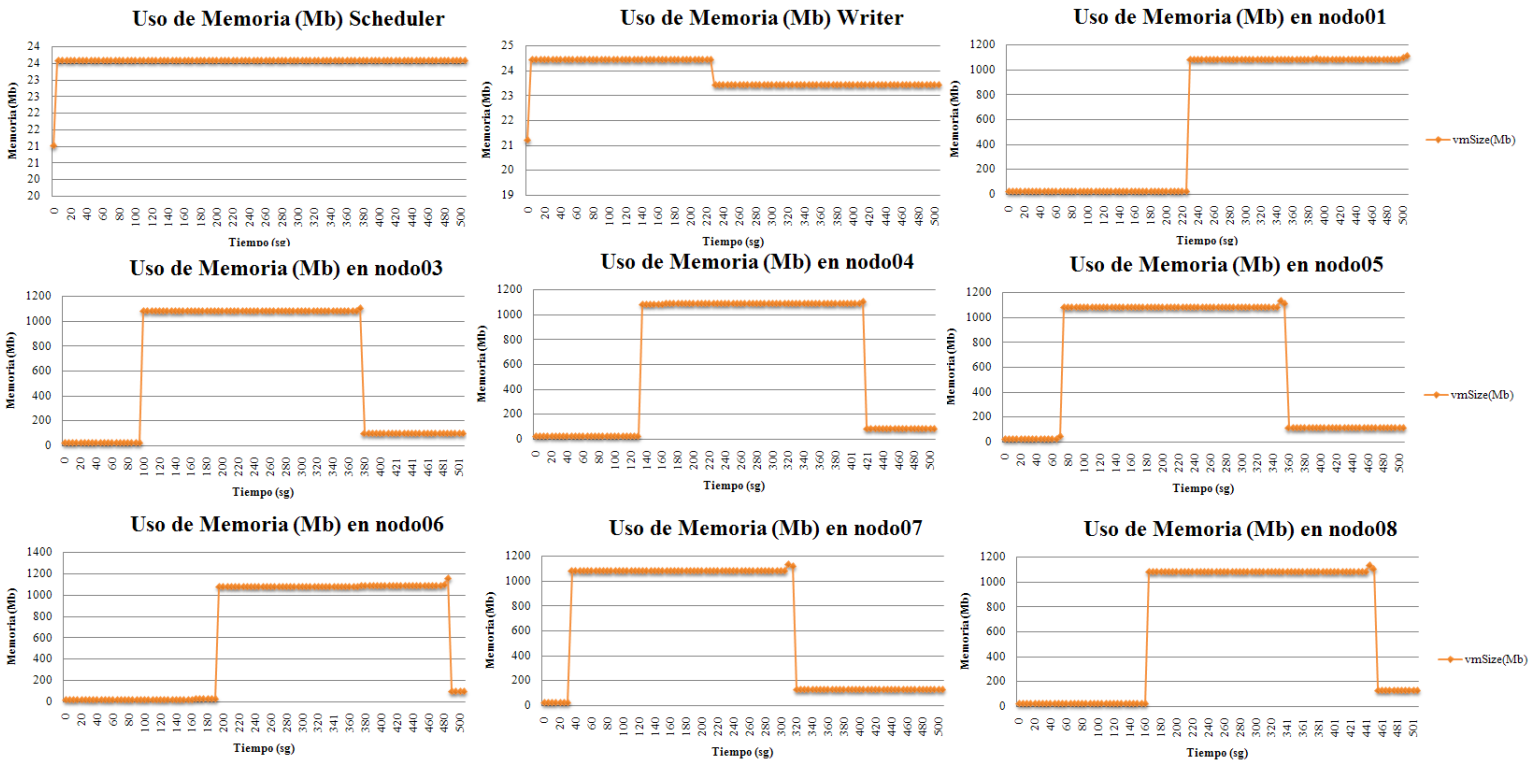

Figura 5.8: Consumo de memoria en los nodos. Ejecución con siete fragmentos.

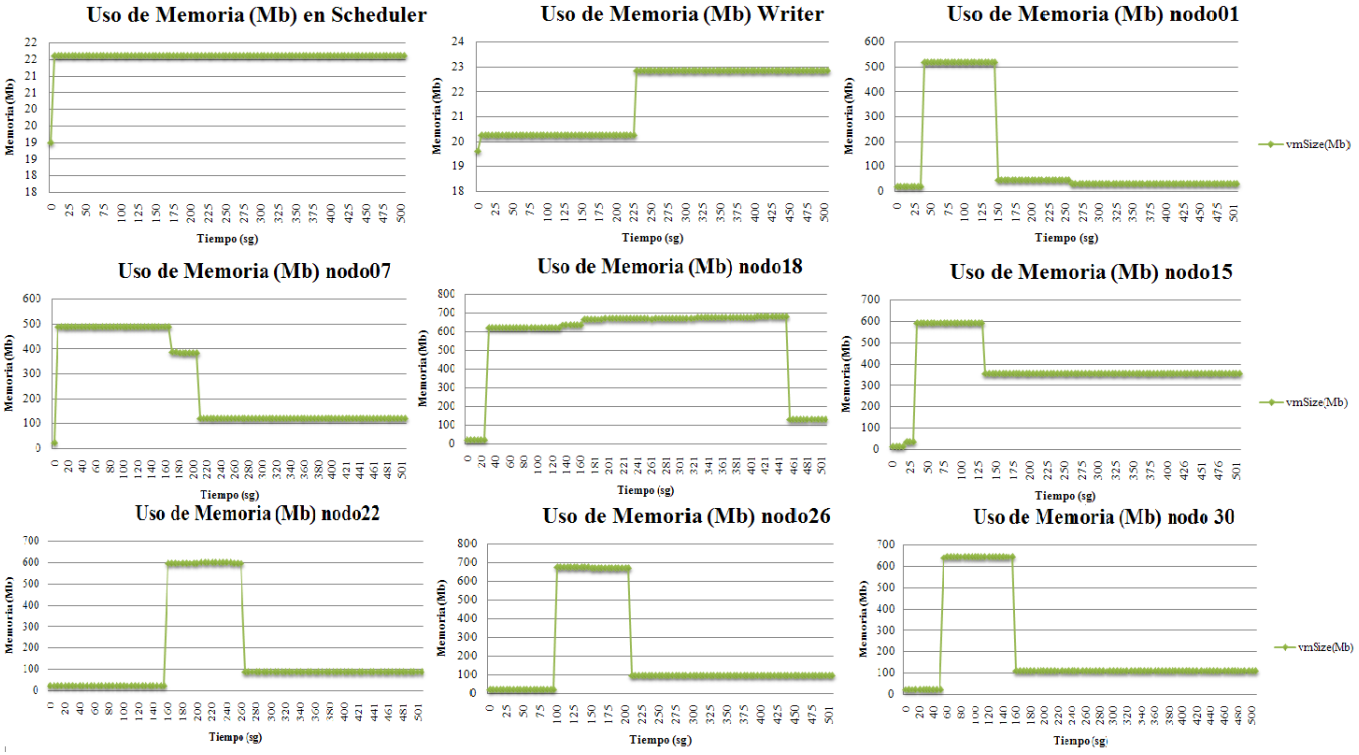

Figura 5.9: Consumo de memoria en nodos. Ejecución con treinta y un fragmentos.

Influencia de MPI en la Aplicación

Hipotesis:

"MPI consume un porcentaje alto del procesamiento".

Cuando se paralelizan aplicaciones con la ayuda de interfaces como MPI, muchas veces suele presentarse la problemática de que se puede consumir un alto porcentaje del tiempo de ejecución en funciones propias de la interfaz. La motivación de este experimento en particular es la gran cantidad de tiempo que invierten los procesos relacionados al nodo máster, en que espera contínuamente en un bucle el envío de mensajes por parte de los workers. Aún así y dado que el envío de los fragmentos a los workers se está haciendo a través de MPI, es relevante conocer el nivel de intrusión que genera dentro de la ejecución global de la aplicación.

Proceso:

Para determinar la influencia de los pasos de mensajes dentro de la aplicación, se ha decidido realizar una ejecución en la cual se han insertado como datos de entrada la base de datos NT y la secuencia *e_chrysantem.fas* empleando 32 nodos de cómputo y fragmentando la base de datos en 31 piezas (una para cada worker). Adicionalmente, con la ayuda de la instrumentación realizada para que el traceador de Vampir identifique las funciones que nos interesan, ha sido posible obtener y visualizar los resultados presentados en la gráfica 5.10, siendo una captura de pantalla de la herramienta VAMPIR, la cual permite mostrar los datos obtenidos. Particularmente en este caso, estamos interesados en observar el porcentaje de tiempo de ejecución y procesamiento que consume mpiBLAST en llamado a funciones propias de MPI.

Conclusiones:

De esta forma podemos visualizar que las funciones que hemos instrumentado para medir consumo de recursos están identificadas en la columna CAOS, (por asignar un nombre a la monitorización insertada por nosotros), de igual forma aquélla relacionada con mpiBLAST exclusivamente es la que pone *Application* y por último Entrada/Salida y MPI están al final y encima del gráfico respectivamente. Señalando de ésta forma que del número total de llamadas a funciones que realiza mpiBLAST poco más de un 51 % está destinado a funciones MPI, lo que se traduce en un consumo excesivo de funciones no relacionadas directamente con el algoritmo de la aplicación.

La función con mayor consumo de tiempo durante la ejecución es un *MPI_Iprobe* que vigila los mensajes que están siendo enviados desde el worker hacia el máster, el cual, tal como se puede ver en la gráfica 5.11 está consumiendo el 45 % del tiempo total de ejecución.

Sin embargo es necesario recordar que dado que las comunicaciones máster/worker están siendo llevadas a través de MPI esto puede generar un incremento en el porcentaje a diferencia de si se emplera la estrategia de directorio compartido que se pudiera acceder a través de un sistema de ficheros paralelo.

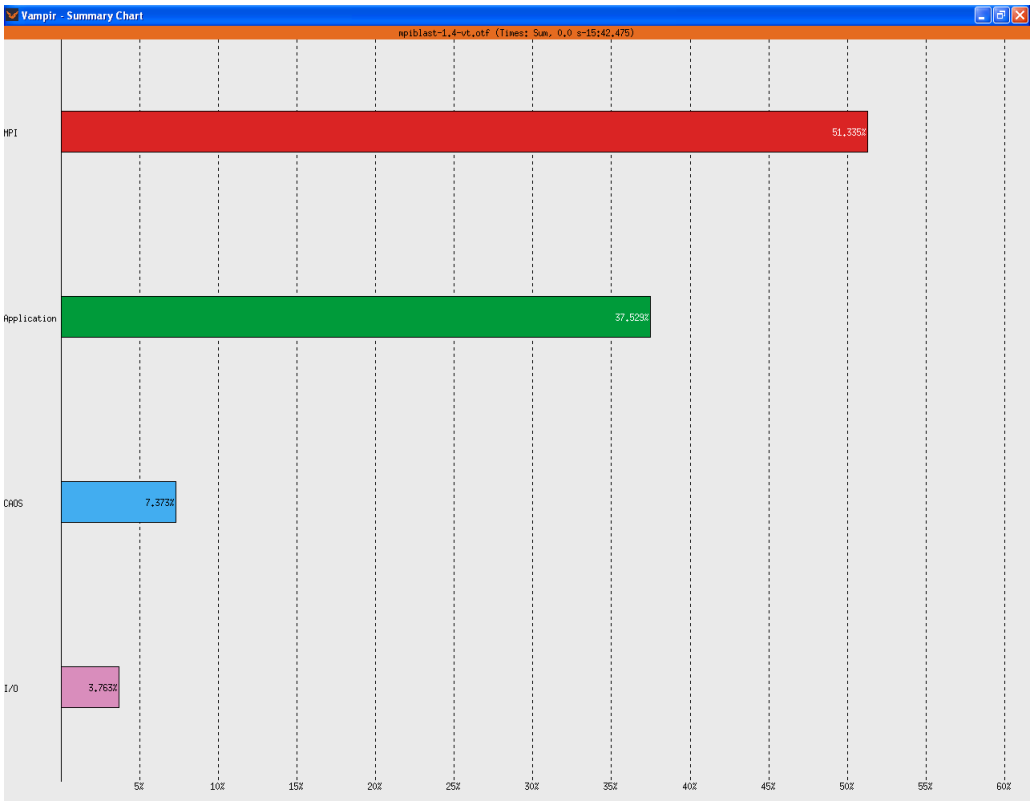

Figura 5.10: Porcentajes Globales de MPI en una ejecución de mpiBLAST

Granularidad de la carga de trabajo

Hipotesis:

"El fragmentar la base de datos en un mayor número de segmentos agiliza la ejecución".

Al plantearnos este experimento la respuesta que pretendemos responder es, qué sucede cuando los fragmentos de la base de datos son cada vez más pequeños, se minimiza el tiempo de ejecución o en caso contrario la gestión de más piezas de base de datos incrementa el overhead por comunicación y envío de los fragmentos por parte del máster a los workers. Situaciones que suelen aparecer a medida que la cantidad de datos que se hace necesario hacer llegar a los nodos de cómputo aumenta en cantidad y no en volumen.

Proceso:

Para ello hemos realizado un conjunto de consultas para obtener los tiempos medios de ejecución y distribución para cada fragmento. Hemos empleado la totalidad de los nodos de cómputo disponibles (32 nodos del cluster Beowulf) y se han repetido las ejecuciones 10 veces por cada caso. Adicionalmente hemos fijado la base de datos (NT en este caso) y la secuencia de consulta, nuevamente estamos probando con la *e_chrysantem.fas*. El factor que hemos variado es el número de fragmentos en el que está segmentada la base de datos con la finalidad de observar qué comportamiento presenta la aplicación con cargas de trabajo variables.

Los resultados de la variable de respuesta tiempo de ejecución medio están detallados en la

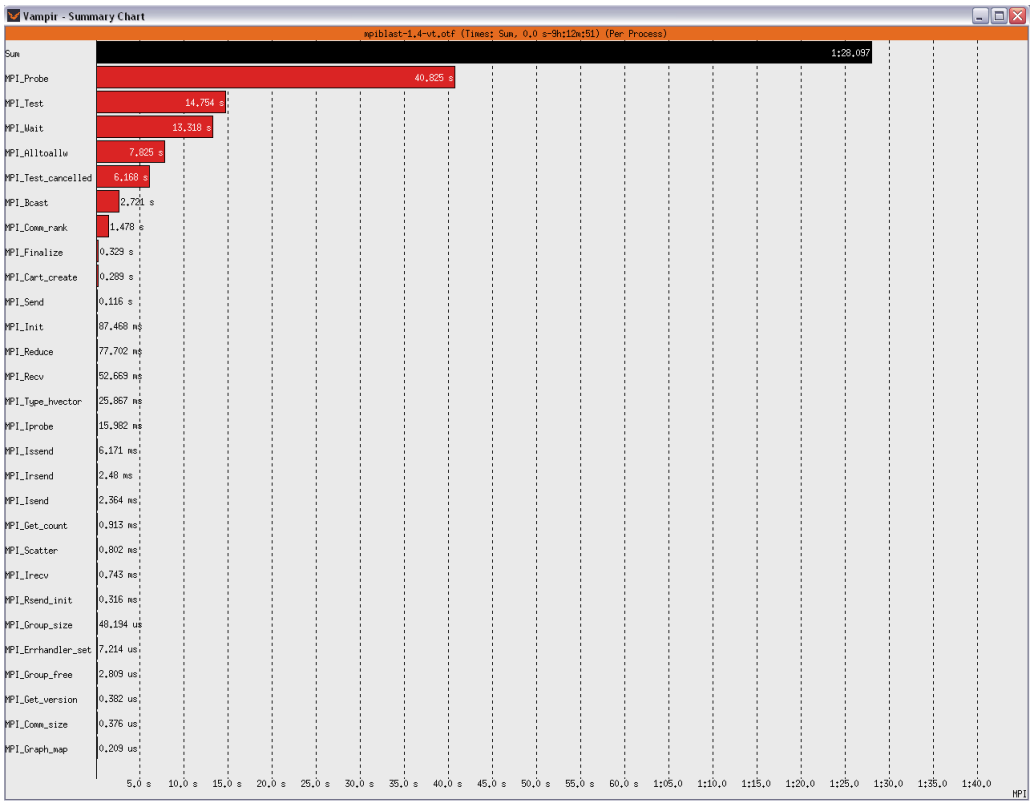

Figura 5.11: Consumo funciones MPI en la ejecución del nodo Máster.

tabla 5.3 que se presenta a continuación, donde se puede visualizar que a medida que se aumenta el número de fragmentos de la base de datos, los tiempos de repartición y procesamiento de cada uno de ellos es cada vez menor, sin embargo, tal como se sospechaba se observa una variación interesante en el tiempo total de ejecución a medida que el número de fragmentos crece, ya que involucra un mayor tiempo de espera para algunos workers para recibir su fragmento.

Conclusiones: Con lo que podemos concluir que existe cierto beneficio a medida que se disminuye el tamaño del grano de la carga de trabajo, esto debido al balanceo de trabajo que se produce dentro de la aplicación y el solapamiento que se presenta a medida que se computa y distribuyen los fragmentos. Sin embargo, al momento de la repartición sólo los primeros nodos que reciban la totalidad de fragmentos correspondientes iniciarán el procesamiento y serán los primeros en terminar. Ya que al carecer de acceso simultáneo a los fragmentos que le corresponden a cada nodo, los últimos habrán de esperar que todos los anteriores hayan recibido su porción de trabajo para poder obtener el suyo y comenzar la consulta al algoritmo.

De igual forma, a nosotros nos parece un punto importante que se podría mejorar en un futuro dentro del sistema de gestión de fragmentos existente actualmente, de forma que pudiera realizarse la repartición de forma dinámica a cada uno de los workers que vaya quedando desocupado, de forma que se reduzca el tiempo de inactividad que presentan lo más posible.

Ejecución con 8 nodos de cómputo			
Número de Fragmentos	**7**	**14**	**21**
Tiempo Medio de Distribución por Nodo (sg)	32,89	17,41	10,17
Tiempo Medio de Consulta BLAST por Nodo (sg)	288,39	144,20	122,29
Tiempo Medio de Envío de los Resultados por Nodo(sg)	5,86	3,68	1,92
Tiempo Total de Ejecución (sg)	627,09	626,14	592,92
Ejecución con 32 nodos de cómputo			
Número de Fragmentos	**31**	**62**	**93**
Tiempo Medio de Distribución por Nodo (sg)	5,55	2,79	1,89
Tiempo Medio de Consulta BLAST por Nodo (sg)	113,94	57,77	38,93
Tiempo Medio de Envío de los Resultados por Nodo(sg)	2,02	1,01	0,75
Tiempo Total de Ejecución (sg)	508,13	417,56	434,29

Tabla 5.3: Tiempos medios variando la granularidad de la carga de trabajo en mpiBLAST para 8 y 32 nodos de cómputo

Composición de la base de datos NT

Hipotesis:

"Los tamaños de las secuencias dentro de la base de datos son de tamaños similares".

Si estamos hablando de que la mayor base de datos biológica existente hoy en día es la NT (por supuesto excluyendo los genomas humanos), deseamos conocer la naturaleza de los datos que la conforman, ya que se trata de un fichero de secuencias biológicas que descomprimido (extraído del fichero `.gz` ocupa disponible en el repositorio) ocupa aproximadamente poco más de unos 22 GB, que al momento de trabajar con ellos reflejan un consumo masivo de recursos de almacenamiento y alojamiento en memoria.

Proceso:

Para ello se realizó un estudio con la ayuda del paquete estadístico SPSS en el que se introdujeron todos los datos referentes a la base de datos y que permitió obtener el número de secuencias existentes dentro de la base de datos, el tamaño medio de las secuencias que la componen y la distribución de las mismas en el fichero. De ésta forma pudimos identificar el volumen de datos con los que estamos trabajando y poder predecir la influencia que tiene uno u otro tamaño de secuencia en la ejecución de la consulta y en la conformación de los fragmentos que serán enviados a los workers.

La distribución de secuencias dentro de la base de datos está reflejada en la figura 5.12, donde se puede observar que la NT, un fichero que está compuesto por más de 7 millones de secuencias diferentes, que el tamaño medio de una secuencia es de 3000 caracteres (estamos despreciando por el momento todo significado biológico que éstas secuencias pudieran tener según su tamaño), ya que estamos tratanto cada secuencia biológica como una cadena de caracteres contra los que se contrastarán las secuencias que conforman la secuencia de consulta.

Conclusiones:

Al estar trabajando con una aplicación cuya esencia es la comparación de cadenas de ca-

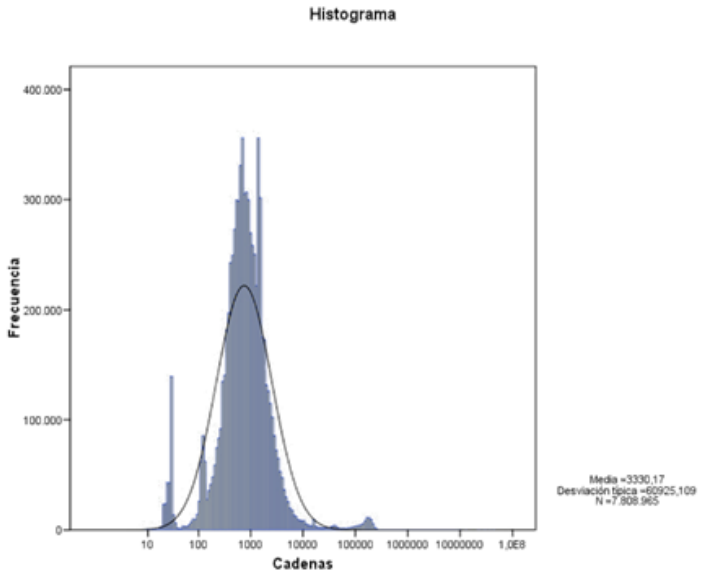

Figura 5.12: Histograma de distribución de secuencias dentro de la base de datos biológica NT

racteres, era de vital importancia identificar el "mayor" reto que representaban las secuencias dentro de la base de datos, de forma que a medida que se realizaran las ejecuciones se conociera la naturaleza de la carga de trabajo que se introduce a la aplicación.

Dado que el número medio de caracteres dentro de una cadena oscila entre unos cientos de KB (en contraste con los millones que conforman un genoma humano, TB) podemos saber que los fragmentos en los que se ha segmentado la base de datos están compuestos por n cadenas de tamaño variable. Denotando así que la presencia de cadenas que ocupen MB no es el factor común dentro de cada fragmento.

Resumen

Una vez que hemos finalizado la experimentación, podemos afirmar que mpiBLAST como aplicación paralela de bioinformática presenta algunas situaciones de ineficiencias cuando se ejecuta en clústers con características muy específicas de memoria, sistema de fichero y disponibilidad de nodos, como el nuestro. Estas ineficiencias están directamente relacionadas con los parámetros que hemos evaluado, logrando así determinar la influencia que generan estos en la ejecución.

El tamaño de la base de datos biológica con la que se va a trabajar, incluso consultas de bases de datos contra ellas mismas (para buscar alguna incongruencia entre una y otra) refiere el mayor consumo de recursos disponibles; es decir, ella fija parte de la cantidad de recursos que debes tener al alcance para obtener respuestas en un mejor tiempo, tales como número de workers a emplear, de acuerdo a la cantidad de memoria disponible.

De igual forma, la distribución de trabajo cuando se aumenta la granularidad de los fragmentos de la base de datos, permite el manejo de ficheros de menor tamaño, lo que se refleja

en una mejora en los tiempos de ejecución momentánea, ya que si el número de fragmentos es demasiado grande, todo el proceso de gestión y distribución del máster a los workers introduce overhead en las comunicaciones e incrementa el tiempo total de ejecución.

5.4. Diseño del Modelo de Rendimiento Preliminar

Partiendo de los resultados observados a lo largo de los experimentos realizados y de las trazas de ejecución obtenidas, podemos plantear una primera aproximación matemática del comportamiento presenta mpiBLAST bajo el entorno paralelo sobre el que se está ejecutando.

Señalando que en esta aproximación se está despreciando parte del tiempo consumido al momento de generar el fichero de salida con las secuencias resultantes de la búsqueda; de forma que se perciba la influencia existente por parte del factor de fragmentación de la base de datos de secuencias biológicas.

Si el tiempo total de ejecución T_{total} viene dado por la siguiente ecuación:

$$T_{total} = T_{send} * \frac{N_{frag}}{N_{proc}} + \sum_{n=1}^{k} T_{proc_n} * \frac{N_{frag}}{N_{proc}} \tag{5.1}$$

Y considerando que el tiempo de envío de los fragmentos, T_{send}, a cada worker, podría expresarse como la relación entre el tamaño del fragmento s, expresado en *megabytes* y el ancho de banda b disponible en la red del clúster, y se ve reflejado en la ecuación 5.2:

$$T_{send} = \frac{s}{b} \tag{5.2}$$

Una vez definido el tiémpo de envío por fragmento, el volumen de datos por procesar de mpiBLAST se puede expresar como:

$$V = s * F \tag{5.3}$$

Lo que representaría la cantidad total de datos con los que habría de tratar la aplicación en una aplicación determindada. De forma de que si se estuviera tratando con F fragmentos, el tiempo de procesar cada uno de ellos realizando la búsqueda de mpiBLAST, determinaría la carga total de trabajo E y estaría determinada por:

$$W = T_{proc_n} * F \tag{5.4}$$

donde: T_{proc}, es el tiempo que le toma al proceso n dentro de cada worker para llevar a cabo la función de búsqueda de BLAST.

Pudiendo concluir que la carga W está condicionada por el volumen en una relación $Proc(V) = W$ de forma que determinase el tiempo total de ejecución a través de:

$$T_{total} = \frac{V}{b * P} + \frac{W}{P} \tag{5.5}$$

Cabe destacar que el tiempo de procesamiento T_{proc} está condicionado por la capacidad de memoria disponible dentro del nódo de cómputo. Este modelo analítico es la aproximación inicial para la descripción del comportamiento de mpiBLAST como aplicación bioinformática paralela, bajo un paradigma máster/worker.

Capítulo 6

Conclusiones y Trabajos Futuros

"Si dos hombres vienen caminando por una carretera, cada uno cargando un pan, y, cuando se encontrasen, intercambiaran los panes, cada hombre se iría con uno... Pero, si dos hombres vienen caminando por una carretera cada uno cargando una idea, y, cuando se encontrasen, intercambiaran las ideas, cada hombre iría ahora con dos..."

Proverbio Chino.

6.1. Conclusiones

Las aplicaciones paralelas hoy en día representan un gran avance para los campos y disciplinas científicas existentes, ya que ellas hacen posible la realización de diversas tareas de forma simultánea, el procesamiento de datos de forma más rápida (y eficiente) y a su vez facilitan el trabajo con volúmenes de datos cada vez mayores. Aplicaciones paralelas como mpiBLAST [12] no son la excepción, ya que es una herramienta que ha sido propuesta como una mejora sustancial al algoritmo de alineamiento de secuencias BLAST [1] publicado en 1990, y que ha conseguido mantenerse como aplicación estrella en el el campo de la bioinformática, dado que muchas tareas más complejas están basadas en ella.

Recordando que la esencia de BLAST es realizar alineamientos locales (búsqueda de similitudes) entre dos secuencias biológicas (cadenas de ADN o Proteínas) ó entre una secuencia y una base de datos, que está compuesta por varias secuencias más; se parte de la premisa planteada en la publicación de 1990 de que al momento de una consulta es necesario que el nodo que va a efectuar la búsqueda tenga en memoria la base de datos contra la que se va a realizar la búsqueda, de lo contrario no era posible llevar a cabo el algoritmo, claro las capacidades que tenían los ordenadores de la época no se comparan con las que se han alcanzado hoy en día.

El problema se presenta cuando una ejecución secuencial típica de BLAST se ralentiza a medida que las bases de datos biológicas aumentan en tamaño. Por este motivo, los creadores de mpiBLAST [12] plantearon *segmentar* la base de datos y repartir el trabajo entre los nodos,

siguiendo un paradigma de paralelismo basado en Máster/Worker; existe un nodo maestro que se encarga de distribuír y gestionar el trabajo repartido entre n cantidad de nodos worker. Todo ésto diseñado y programado con la ayuda de la interfaz de paso de mensajes (MPI, Message Passing Interface).

El proceso de comparación de las cadenas biológicas es la fase que consume mayor cantidad de tiempo dentro de la ejecución de mpiBLAST y si el tamaño de los fragmentos supera la capacidad de memoria del nodo y pagina a disco, este tiempo incrementará considerablemente. Por lo tanto se necesita un entorno de ejecución en paralelo (un cluster) que permita llevar a cabo las ejecuciones de la forma más eficiente. Nuestra hipótesis ha sido durante todo el proceso de investigación que si el problema es demasiado grande para ser resuelto con pocos nodos de cómputo sin presentarse ineficiencias se hace necesario determinar estrategias que permitan realizar la ejecución sacando provecho de los recursos disponibles.

La aplicación mpiBLAST, posee toda una estrategia de manejo de la entrada/salida que pudiera llegar a interpretarse como un problema cuando la carga de trabajo aumenta, el algoritmo de gestión de E/S no llega a ser capaz de repartir el trabajo de forma eficiente, generando cuellos de botella al serializarse el inicio y final de la ejecución.

Como resultados finales obtenidos luego de toda la experimentación presentada en el capítulo 5, muchos de ellos giraban entorno a un problema existente de gestión de los datos; es decir, a medida que el volumen de la carga de trabajo aumenta se hace necesario mejorar aún más las estrategias de rendimiento existentes, esto debido a que tal como se señaló en su momento, el consumo de recursos disponibles incrementa considerablemente y la presencia de ineficiencias se hace inevitable. Por ejemplo, cuando hemos trabajado con bases de datos con un tamaño que no superaba 1 GB, la presencia de ineficiencias no afectaban el tiempo total de ejecución. La carga de trabajo era tan pequeña que el retraso existente en la distribución de los datos para ser procesados era imperceptible. Situación contraria cuando se toman ficheros de varios GB que necesitan ser formateados en k segmentos y repartidos a n workers. La escala de tiempo empieza a tornarse más significativa, los tiempos pasan de ser unos pocos segundos a casi una hora por cada formateo; sucede de forma similar con el tiempo total de ejecución de la consulta, a medida que la secuencia que se contrasta y la base de datos aumentan de tamaño, el tiempo de respuesta aumenta considerablemente, la cantidad de datos influyen directamente con el volumen de información a enviar durante la entrada y la salida.

Otra observación que se obtuvo a medida que avanzaba la investigación ha sido la serialización de las comunicaciones, en la que gran parte del tiempo total de cómputo estaba siendo gastado en el envío de los fragmentos, no decimos que consume mucho tiempo enviar un fragmento, sino que lo que retrasa la ejecución es la forma en que éstos están siendo enviados al worker; esto se obtuvo una vez que monitorizamos cada una de las fases involucradas en la ejecución y nos percatamos que, el nodo n no recibirá su fragmento k a menos que los nodos 0 hasta *n-1* hayan recibido los fragmentos 0 hasta *k-1* respectivamente. La causa a la que atribuímos la presencia de esta ineficiencia es que dado que se supone la presencia de un sistema de ficheros distribuido,

se desprecia la posibilidad de que esta fase se secuenciase, y estaríamos hablando de tiempos de inactividad, para algunos nodos, superiores al 50 % de lo que toma realizar la consulta como tal.

También se presenta una serialización de la ejecución a medida que se genera el fichero de salida. Bajo la configuración de comunicación que se ha adoptado durante nuestro estudio de emplear la interfaz de pasos de mensaje MPI y ya que la construcción del fichero resultante corresponde al proceso escritor (*writer*) del máster; todos los workers a medida que van terminando de procesar habrán de esperar que dicho proceso sea capaz de atender su petición, generándose una cola de tareas por realizar por el máster, que a medida que el número de procesadores aumente termina convirtiéndose en uno de los cuellos de botella más significativos dentro de la aplicación.

El número de ineficiencias relacionadas con gestión de E/S, en mpiBLAST (un M/W con prerepartición de fragmentos), es relevante, ya que no se está explotando el paralelismo propio del diseño de la aplicación; y consideramos que estas problemáticas habrán de ser resueltas con propuestas que tomen en consideración la presencia de una fuerte dependencia a los datos que se deben procesar.

Adicionalmente hemos obtenido resultados interesantes cuando se varía el factor de granularidad de la carga de trabajo, entiéndase como este factor el fragmentar la base de datos en 2 y 3 veces el número de workers disponibles, de forma que cada worker procesara piezas de base de datos aún más pequeñas, ganancia que se vería reflejada en la disminución del tiempo de espera por el worker n para recibir su porción de trabajo, sin embargo, nos ha sorprendido la estrategia de distribución de fragmentos presente en la herramienta. Ya que al basarse en un sistema de ficheros distribuído, deciden entregar la totalidad de la carga de trabajo a cada worker antes de que empiece a procesar el primer fragmento, lo que ocasiona una serialización de la ejecución.

Si no existiera el problema que comentamos líneas más arriba sobre la serialización presente en la fase de repartición, debido a la carencia de sistema de ficheros paralelos, la estrategia sería la más adecuada, pero por el contrario, si no se cuenta con un entorno con estas características obliga a cada nodo esperar hasta que sus predecesores hayan recibido su carga de trabajo, para luego recibir de igual forma el siguiente fragmento y así sucesivamente con los dos o tres más que pudieran tocarle. Incrementándose de esta forma el tiempo de inactividad en todos los nodos por el tiempo de espera que habrán de pasar mientras reciben sus fragmentos asignados.

El algoritmo de gestión de fragmentos (*scheduler*, dentro del código fuente) presente en el nodo máster, refleja un intento de dinamismo en esta variable, ya que si un worker ha finalizado la totalidad de su trabajo busca aquél fragmento en el que ningún otro worker está buscando y le señala al worker inactivo que lo copie. Pero desde nuestro punto de vista y la respuesta que se plantea a futuro para hacer más dinámica ésta distribución, es variar la estrategia de forma que los workers reciban sólo un fragmento por vez y luego a medida que van quedando inactivos soliciten más trabajo de forma que se solapen el cómputo y la comunicación. Adicionalmente al lograrse esto se podría introducir conceptos de números de workers ideales [11] para los nodos disponibles.

Al concretarse la propuesta de distribución dinámica de fragmentos, que queda por ahora como trabajo futuro para este proyecto se conseguiría aportar mayor flexibilidad al algoritmo de distribución de fragmentos presente en el *scheduler* que sin duda alguna se reflejaría como ganancia en rendimiento de mpiBLAST como aplicación paralela.

Un factor relevante que vale la pena destacar es que la investigación ha permitido aplicar directamente los fundamentos teóricos de análisis de rendimiento existentes, sobre aplicaciones científicas paralelas de uso diario. Descubriendo de esta forma caminos sobre los que se pueden realizar aportaciones basándonos en la teoría conocida.

Generalizando, las aplicaciones como mpiBLAST que están planteadas bajo un modelo muy sencillo pero que les toca procesar volúmenes masivos de datos presentan problemáticas de rendimiento relacionadas con:

- La gestión de E/S, que a medida que aumenta el número de datos a procesar la repartición de los datos junto con la recolección de resultados se ven afectados.
- La granularidad de la carga de trabajo, requieren una mejor estrategia para balancear la distribución del trabajo (fragmentos de base de datos) entre los nodos de cómputo disponibles.
- La estrategia de distribución de fragmentos no toma en consideración que esta fase se esté serializando.

6.2. Trabajo Futuro

Una vez alcanzado el objetivo planteado para éste trabajo de investigación en particular, quedan algunas lineas abiertas para aportar aún más conocimiento en cuanto a análisis de rendimiento y sintonización de aplicaciones paralelas. Para ello hemos definido como posibles trabajos futuros en los que se encuentren involucradas aplicaciones máster/worker del tipo de mpiBLAST que puedan adaptarse a los modelos de rendimiento existentes y generar aún mayor información para los grupos de investigación relacionados.

Todo esto con la intención de que este trabajo de investigación emplee los aportes obtenidos durante las fases de diseño teórico de modelos de rendimiento y sintonización dinámica y además permita incorporar nuevos enfoques de modelado y análisis de rendimiento de las nuevas aplicaciones científicas disponibles en el mercado.

Nos encontramos en este momento en un punto en el que los datos con los que se están trabajando están tomando protagonismo en el consumo de recursos. Estamos saliendo del paradigma de cómputo complejo y datos sencillos para entrar en una época de cómputo sencillo y cantidad masiva de datos. Los cuales están determinando la cantidad de recursos necesarios para poder llevar a cabo dichos procesos. Es acá donde cada vez más la Computación de Altas Prestaciones, se acerca a los grupos científicos de diferentes áreas a lo largo del mundo, brindando una opción

para sacar provecho de las máquinas paralelas con las que cada uno de ellos cuenta en sus sitios de trabajo.

Un factor relevante en nuestras próximas investigaciones es la posibilidad de poder reproducir el ambiente sobre el que pudieran estarse ejecutando las aplicaciones a evaluar, si bien el análisis realizado a mpiBLAST pudiera ser muy específico en cuanto a aplicación como tal, el conocimiento generado a través de su evaluación de rendimiento nos es útil para considerar nuevos enfoques que adopten las aplicaciones científicas y generen aportes a los modelos diseñados anteriormente.

Es interesante la idea de sondear un entorno colaborativo, pero tal como señalamos esto estará directamente relacionado con la capacidad de reproducción del entorno y de los experimentos. Es decir, subir un nivel siempre y cuando pudiera mantenerse la capacidad de control de los parámetros que se posee ahora en los clústers tradicionales.

Conjuntamente, quedó sobre nuestro grupo de investigación la posibilidad de estudiar la influencia que pudiera tener aspectos económicos dentro del aprovechamiento de la computación de altas prestaciones, introducir conceptos de cómputo colaborativo como Cloud de forma que aplicaciones del tipo de mpiBLAST puedan ser ejecutadas en entornos de miles de nodos de cómputos conectados bajo este enfoque, lo que significaría para el grupo que trabaje con este tipo de herramientas una opción de procesar grandes cantidades de datos sin la necesidad de tener un Marenostrum (por poner un ejemplo) en su laboratorio. Por el momento se seguirá tratando como un punto de referencia a futuro ya que dependerá de las posibilidades de *High Performance Computing* o *High Throughput Computing* (HPC o HTC) que se planteen, ya que aún restan por definir.

Queda como línea abierta la gestión de E/S en las aplicaciones biológicas que integrarán el trabajo con grandes volúmenes de datos y nuevas tecnologías de secuenciación. De donde se puede plantear la gestión de aplicaciones con datos con un tamaño superior a TB y la implementación de modelos conocidos de cómputo. Junto con el uso eficiente de los recursos, empleando la menor cantidad posible de ellos en el manejo de grandes cantidades de datos de forma total y exhaustiva; de forma que se mantenga la investigación ligada directamente con la filosofía de cómputo de altas prestaciones y se convierta en una solución para las aplicaciones científicas actuales y siguientes.

Sin duda alguna, la bioinformática y sus aplicaciones continuarán formando parte de los principales beneficiarios del cómputo de altas prestaciones, de forma que todo lo que se pueda concebir para generar aún más beneficios en el procesamiento de los datos biológicos será un aporte bien recibido por los grupos científicos existentes, y servirá de base para aún más avances y aportes en el *High Performance Computing.*

Bibliografía

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic Local Alignment Search Tool," *J Mol Biol*, vol. 215, pp. 403 – 410, 1990.

[2] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped blast and Psi-blast: a new generation of protein database search programs," *Nucl. Acids Res.*, vol. 25, no. 17, pp. 3389 – 3402, 1997. [Online]. Available: http://nar.oxfordjournals.org/cgi/content/abstract/25/17/3389

[3] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference.* New York, NY, USA: ACM, 1967, pp. 483–485.

[4] R. M. Badia, J. Labarta, J. Giménez, and F. Escalé., "Dimemas: Predicting mpi applications behaviour in grid environments." in *Workshop on Grid Applications and Programming Tools (GGF8)*, June 2003.

[5] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler, "Genbank," *Nucleic Acid Research*, vol. 36, pp. 25–30, 2008.

[6] R. Bjornson, A. Sherman, S. Weston, N. Willard, and J. Wing, "Turboblast®: A parallel implementation of blast built on the turbohub," in *IPDPS 2002 Workshops*, April 2002, p. 1.

[7] D. G. Brown, *Bioinformatics Algorithms: Techniques and Applications.* Wiley-Interscience, 2008, ch. A survey of seeding for sequence alignment, pp. 117–141.

[8] A. Chan, D. Ashton, R. Lusk, and W. Gropp, *Jumpshot-4 Users Guide*, Mathematics and Computer Science Division, Argonne National Laboratory, July 2007.

[9] A. Churbanov, "Pam matrix for Blast algorithm," University of Nebraska, Tech. Rep., April 11 2002.

[10] G. Costa, J. Jorba, A. Morajko, T. Margalef, and E. Luque, "Performance models for dynamic tuning of parallel applications on computational grids," in *2008 IEEE International Conference on Cluster Computing*, 29 2008-Oct. 1 2008, pp. 376–385.

[11] E. César, A. Morajko, T. Margalef, J. Sorribes, A. Espinosa, and E. Luque, "Dynamic performance tuning supported by program specification," *Scientific Programming*, vol. Volume 10, Number 1/2002, pp. 35–44, 2002.

[12] A. E. Darling, L. Carey, and W.-C. Feng., "The design, implementation, and evaluation of mpiblast," in *Cluster World Conference & Expo.*, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.3974

[13] M. Dayhoff, "Survey of new data and computer methods of analysis," in *Atlas of Protein Sequence and Structure.* National Biomedical Research Foundation: National Biomedical Research Foundation, 1978, vol. 5, pp. 3–29.

[14] W.-C. Feng, "Green destiny + mpiblast = bioinfomagic," in *10th International Conference on Parallel Computing: Bioinformatics Symposium*, 2003, pp. 1–6.

[15] M. Geimer, F. Wolf, B. Wylie, E. Abraham, D. Becker, and B. Mohr, "The Scalasca performance toolset architecture," in *International Workshop on Scalable Tools for High-End Computing*, B. M. M. Gerndt, J. Labarta, Ed., June 2008, pp. 51 – 65.

[16] W. Gropp, E. Lusk, D. Ashton, P. Balaji, D. Buntinas, R. Butler, A. Chan, D. Goodell, J. Krishna, G. Mercier, R. Ross, R. Thakur, and B. Toonen, *MPICH2 User's Guide*, Mathematics and Computer Science Division. Argonne National Laboratory, June 2009.

[17] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proc. Natl. Acad. Sci. USA*, vol. 89, pp. 10 915–10 919, 1992.

[18] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Commun. ACM*, vol. 18, no. 6, pp. 341–343, 1975.

[19] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques forExperimental Design, Measurement, Simulation, and Modeling.* Wiley - Interscience, 1991.

[20] J. Jorba, "Análisis automático de prestaciones de aplicaciones paralelas basadas en paso de mensajes." PhD. Thesis, Universitat Autònoma de Barcelona., Barcelona, Febrero 2006. [Online]. Available: http://www.tesisenxarxa.net/TESIS_UAB/AVAILABLE/TDX-1013106132034/

[21] H.-Y. Liao, Y. M.-L. Yin, and Y. Cheng, "A parallel implementation of the smith - waterman algorithm for massive sequences searching," in *Conference Proceedings. 26th Annual International Conference of the Engineering in Medicine and Biology Society. EMBC 2004.*, vol. 2, 2004, pp. 2817–2820 Vol 4. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1403804

[22] D. J. Lilja, *Measuring Computer Performance: A Practitioner's guide*, C. U. Press, Ed. Cambridge University Press, 2000.

[23] H. Lin, P. Balaji, R. Poole, C. Sosa, X. Ma, and W.-C. Feng, "Massively parallel genomic sequence search on the blue gene/p architecture," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, no. 33. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–11.

[24] B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Irvin, K. Karavanic, K. Kunchithapadam, and T. Newhall, "The paradyn parallel performance measurement tool," *Computer*, vol. 28, no. 11, pp. 37–46, Nov 1995.

[25] A. Morajko, "Dynamic tuning of parallel/distributed applications," PhD Thesis, Universitat Autònoma de Barcelona, 2003. [Online]. Available: http://www.tdx.cbuc.es/TDX-1124104-171625/

[26] W. E. Nagel, A. Arnold, M. Weber, and K. Solchenbach, "Vampir: Visualization and analysis of mpi resources," Central Institute for Applied Mathematics, Tech. Rep., 1996.

[27] S. Needleman and C. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins." *J. Mol. Biol.*, vol. 48, pp. 443–453, 1970.

[28] C. Oehmen and J. Nieplocha, "Scalablast: A scalable implementation of blast for high-performancedata-intensive bioinformatics analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 8, pp. 740–749, 2006.

[29] W. R. Pearson, "Rapid and sensitive sequence comparison with fastp and fasta," *Methods in enzymology*, vol. 183, pp. 63–98, 1990.

[30] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "Paraver: A tool to visualize and analyze parallel code," *WoTUG*, vol. 44, pp. 17–31, 1995.

[31] R. Schaller, "Moore's law: past, present and future," *Spectrum, IEEE*, vol. 34, no. 6, pp. 52–59, Jun 1997.

[32] S. S. Shende and A. D. Malony, "The tau parallel performance system," *International Journal of High Performance Computing Applications*, vol. 20, pp. 287–311, 2006.

[33] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences." *J Mol Biol*, vol. 147, no. 1, pp. 195–197, March 1981. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/7265238

[34] M. S. Sousa and A. C. M. A. Melo, "Packageblast: an adaptive multi-policy grid service for biological sequence comparison," in *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2006, pp. 156–160.

[35] B. Wilkinson and M. Allen, *Parallel Programming. Techniques and Applications Using Networked Workstations and Parallel Computers.* Prentice Hall, 2005.

[36] F. Wolf and B. Mohr, "Kojak - a tool set for automatic performance analysis of parallel applications," in *Proc. of the European Conference on Parallel Computing (Euro-Par)*, ser. Lecture Notes in Computer Science, vol. 2790. Klagenfurt, Austria: Springer, August 2003, pp. 1301–1304, demonstrations of Parallel and Distributed Computing.

[37] F. Wolf, "Automatic performance analysis on parallel computers with smp nodes," Ph.D. dissertation, Forschunszentrum Jülich, 2002. [Online]. Available: http://www.kfa-juelich.de/nic-series/volume17/nic-series-band17.pdf

[38] C.-T. Yang, T.-F. Han, and H.-C. Kan, "G-blast: a grid-based solution for mpiblast on computational grids," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 2, pp. 225–255, 2009. [Online]. Available: http://dx.doi.org/10.1002/cpe.1346

Índice alfabético