Máster en Computación Paralela y Distribuida.
Junio, 2010.

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Análisis y caracterización de trabajos BLAST para la planificación eficiente en entornos Grid y Supercomputación

Autor: Abel Antonio Carrión Collado
Directores: Dr. D. Ignacio Blanquer Espert
Dr. D. Vicente Hernández García

A todos los miembros del GRyCAP

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

El sistema universitario español se encuentra actualmente (2010) en fase de adaptación al Espacio Europeo de Educación Superior (EEES). Con el fin de llevar a cabo dicha adaptación, se ha definido un sistema de titulaciones basado en dos niveles: GRADO y POSGRADO. A su vez, el segundo nivel (POSGRADO) conduce a la obtención de dos títulos: MÁSTER y DOCTORADO (segundo y tercer ciclo respectivamente).

Como primer paso hacia dicha adaptación y siguiendo las pautas indicadas por la Generalitat Valenciana, la Universidad Politécnica de Valencia ha puesto en marcha un conjunto de Programas Oficiales de Posgrado. Cada uno de estos Programas Oficiales de Posgrado está formado por uno o varios Másteres oficiales y el Doctorado.

Uno de los requisitos exigidos para la finalización de cualquier Máster comprende la elaboración y presentación de la Tesis de Máster, en forma de asignatura de 20 créditos ECTS. Así pues, el presente documento constituye la memoria de Tesis realizada en el marco de uno de los másteres oficiales de la UPV: *Máster en Computación Paralela y Distribuida*.

Tal y como se deduce del título de la tesis, el presente documento recoge un conjunto de trabajos de investigación orientados al desarrollo de técnicas y herramientas que permitan la ejecución eficiente de experimentos científicos masivos en infraestructuras de e-Ciencia (concretamente Grid y Supercomputación). Para poner de manifiesto la aplicabilidad de todos estos trabajos, se ha utilizado como caso de uso, una de las tareas más representativas dentro del campo de la Bioinformática: el alineamiento de secuencias mediante la herramienta BLAST (Basic Local Alignment Search Tool).

Las infraestructuras de e-Ciencia suponen una herramienta fundamental para el avance de numerosas disciplinas científicas. La disponibilidad de centenares y miles de elementos de proceso y el almacenamiento eficiente de Petabytes de datos está permitiendo el avance del conocimiento en la física de partículas, la astronomía o la genética. En estas disciplinas se requiere el despliegue y ejecución de grandes experimentos sobre grandes volúmenes de datos, lo que implica la ejecución simultánea y coordinada de una gran cantidad de procesos sobre las infraestructuras disponibles. Sin embargo, el uso eficiente de estas herramientas supone un reto importante, especialmente considerando que el coste económico de estas infraestructuras es considerable y por tanto es fundamental que se garantice un buen uso de las mismas. Por tanto, esta tesis recoge los resultados de una serie de trabajos que se centran en desarrollar mecanismos eficientes para la gestión y planificación de los trabajos ejecutados en las mismas.

Asimismo, es importante destacar que la aplicabilidad de los trabajos de investigación, contenidos en esta tesis, ha permitido establecer colaboraciones con otros institutos y grupos de investigación, tales como: el Instituto Cavanilles de Biodiversitat i Biologia Evolutiva, el Instituto de Biología Molecular y Celular de Plantas (IBMCP) o el Centro de Investigación en Métodos de Producción de Software (ProS).

A lo largo de los capítulos que componen esta memoria, se hará un repaso cronológico a los trabajos realizados, remarcando aquellas carencias y/o necesidades que han motivado la realización de cada uno de ellos.

La memoria consta de 7 capítulos, incluyendo la presente introducción que cubre el primero de ellos. El segundo capítulo hace una revisión del estado del arte de los

principales estándares, tecnologías y paradigmas que constituirán la base del trabajo posteriormente descrito. En el tercer capítulo y como punto de partida, se plantea una metodología para el despliegue y ejecución de experimentos masivos, tanto en infraestructuras Grid en producción como en Supercomputación.

Tomando como base la metodología descrita en el capítulo anterior, el cuarto capítulo realiza una comparativa entre la ejecución de la versión secuencial de BLAST en la infraestructura Grid EGEE y una implementación paralela (mpiBLAST) en el Supercomputador Tirant (uno de los 7 nodos de la actual Red Española de Supercomputación).

A raíz de los resultados obtenidos en el cuarto capítulo, los dos siguientes capítulos plantean diversas técnicas avanzadas con el propósito de suplir las carencias detectadas en el sistema de lanzamiento de experimentos masivos de las infraestructuras Grid. Tales técnicas abarcan desde el desarrollo de un sistema de gestión de trabajos sofisticado hasta la definición de un modelo computacional, que permita estimar el tiempo de respuesta de trabajos BLAST en el Grid.

En el séptimo y último capítulo se citan las principales conclusiones extraídas del trabajo realizado, las publicaciones a las cuales ha dado lugar el trabajo generado y una serie de líneas de trabajo que podrían seguirse para continuar este trabajo en un futuro inmediato. Finalmente, se incluyen un conjunto de referencias necesarias para comprender las ideas y conceptos mencionados a lo largo de esta memoria.

2. ESTADO DEL ARTE

A la hora de abordar un trabajo de investigación, es fundamental realizar, primeramente, un análisis del estado actual de los estándares, tecnologías y paradigmas sobre los que se apoyará todo el estudio. Por tanto, esta sección comenzará describiendo una serie de conceptos imprescindibles para comprender el caso de uso utilizado en la tesis: el alineamiento de metagenomas. A continuación, se hará un repaso a los algoritmos y herramientas más importantes para el análisis de secuencias. En último lugar se analizarán las tecnologías que hacen posible la ejecución de experimentos masivos (como el caso de uso presentado): la computación Grid y la Supercomputación, haciendo especial hincapié en los middlewares, infraestructuras y proyectos más relevantes.

2.1 Bioinformática

2.1.1 Definición

No existe un consenso claro acerca del significado del término 'Bioinformática', pudiéndose encontrar, en la literatura, diferentes definiciones del mismo concepto. Sin embargo, todos los autores parecen estar de acuerdo en que la 'Bioinformática' es la aplicación de las Ciencias Computacionales (junto con otras disciplinas tales como la Estadística) al campo de la Biología Molecular [1].

De esta forma, la 'Bioinformática' puede ser vista desde dos ámbitos totalmente distintos: el biológico y el informático. Por un lado, los biólogos defienden que la Bioinformática es un sinónimo de biología molecular computacional, es decir, la utilización del computador para caracterizar los componentes moleculares de los seres vivos. Por otro lado, desde el punto de vista informático, el término Bioinformática (contracción de Informática Biológica) se refiere al procesamiento y gestión de las grandes cantidades de información, contenidas en los datos biológicos.

2.1.2 Era pre-genómica

Los eventos que sucedieron entre el descubrimiento de la estructura del ADN y su papel como molécula hereditaria y los primeros experimentos con secuenciadores de alta productividad, delimitan aproximadamente, lo que se conoce como era pre-genómica de la biología molecular. En este periodo, la labor de la Bioinformática, denominada 'Clásica', consiste en utilizar el computador con el fin de almacenar, recuperar, analizar, predecir e incluso simular la composición de la estructura de las biomoléculas.

2.1.3 Era post-genómica

La finalización del Proyecto del Genoma Humano, el mayor logro de la Bioinformática hasta la fecha, marcó el final de la era pre-genómica y el inicio de la era post-genómica de la Bioinformática. Este hecho hizo que la naturaleza y las prioridades de la investigación en el campo de la Bioinformática sufrieran un cambio muy importante. De esta manera:

- Al disponer de múltiples genomas completos, ahora es posible buscar diferencias y similitudes entre los genes de múltiples especies. Gracias a este

tipo de prácticas (conocido como Genómica Comparativa) es posible extraer conclusiones acerca de las especies y su evolución.

- Ha sido posible desarrollar tecnologías orientadas a medir el número de copias de un mensaje genético (niveles de expresión de un gen) en diferentes estados de desarrollo de una enfermedad.
- La Genómica Funcional se ha vista favorecida por la aparición de métodos más directos y de gran escala para identificar las funciones y asociaciones de los genes.
- Se ha producido un cambio de paradigma en el área del Análisis de Secuencias, centrándose más en los productos de los genes (es decir, las proteínas) que en los propios genes. Esto ha dado lugar a:
    - La Proteómica, encargada de catalogar las actividades y caracterizar las interacciones entre todas las proteínas de los humanos.
    - La Genómica Estructural, cuya misión es predecir las estructuras de todas las proteínas (en los humanos).

2.1.4 Áreas de investigación más notables

A continuación se describen, brevemente, las sub-disciplinas o áreas de investigación más destacables de la Bioinformática.

- Análisis de secuencias

Desde finales de los años 70, se han decodificado miles de secuencias de ADN de diferentes organismos, siendo finalmente almacenadas en bases de datos. Esta información es posteriormente analizada para determinar qué genes codifican: determinadas proteínas, RNA, etc. No obstante, debido al crecimiento exponencial de las bases de datos, el análisis manual de estas secuencias de ADN pronto se convirtió en un proceso impracticable. Afortunadamente, en la actualidad, este proceso ha sido automatizado mediante el uso de programas de ordenador (como BLAST), permitiendo el análisis del genoma de miles de especies (billones de nucleótidos de información). La misión de estos programas consiste en llevar a cabo el alineamiento entre dos secuencias, es decir, obtener el conjunto de operaciones de edición (inserción, deleción y sustitución) que logran transformar una secuencia en la otra, con el fin de identificar secuencias relacionadas, pero no idénticas. Puesto que el alineamiento de secuencias constituye el caso de uso utilizado en esta tesis, apartados posteriores profundizarán en esta cuestión.

Asimismo, una variante del alineamiento de secuencias, el ensamblaje de genomas, es empleado en el propio secuenciamiento (denominado secuenciamiento 'Shotgun'). En lugar de producir cromosomas enteros, la mayoría de institutos de Genómica optan por generar las secuencias de miles de pequeños fragmentos de ADN. Puesto que los finales de estos fragmentos se solapan, un programa de ensamblaje de genomas es capaz de reconstruir el genoma completo. Estos programas, sin embargo, tienen un alto coste computacional y, por esta razón, la mejora de los mismos supone una cuestión de gran interés en la investigación Bioinformática.

- Anotación de genomas

En el contexto de Genómica, la anotación es el proceso por el cual los genes y otras características biológicas son identificados en una secuencia de ADN. El primer programa software de anotación de genomas fue diseñado en 1995 por el Dr. Owen

White, uno de los miembros del Instituto para la Investigación Genómica, responsables de secuenciar y analizar el primer genoma de un organismo vivo, la bacteria *Haemophilus influenzae.* El Dr. White implementó un software para poder analizar los genes, la transferencia de ARN y otras características de las secuencias de ADN. Aunque la mayoría de programas de anotación actuales conservan la base establecida por el programa del Dr.White, cada vez son más precisos y eficientes.

- Biología evolutiva computacional

La biología evolutiva es el área encargada de estudiar el origen y la descendencia de las especies, así como también los cambios sufridos con el transcurso del tiempo. En los últimos años, el uso asistido del computador ha permitido a los investigadores:

  - Descubrir la evolución de un gran número de organismos mediante la observación de cambios en el ADN, en lugar de basarse únicamente en la taxonomía física u observaciones fisiológicas.
  - Comparar genomas completos, permitiendo, de esta forma, estudiar procesos evolutivos más complejos, tales como la duplicación o la transferencia horizontal de genes.
  - Construir complejos modelos computacionales de poblaciones.

- Análisis de la expresión génica

La expresión de muchos genes puede determinarse midiendo sus niveles de ARN mediante múltiples métodos, siendo los microarrays, probablemente, la técnica más popular. No obstante, un problema común a todas estas técnicas es su clara sensibilidad al ruido. Por esta razón, numerosos investigadores han desarrollado herramientas estadísticas que permitan separar la señal del ruido en la adquisición, en proyectos de expresión génica de alta productividad. De esta forma, ahora es posible determinar qué genes están implicados en un cierto desorden (como un cáncer).

- Análisis de mutaciones en cáncer

En un cáncer, los genomas de las células afectadas se organizan de forma compleja e impredecible. Por ello, los esfuerzos de muchos proyectos masivos de secuenciamiento están destinados a identificar los 'puntos de mutación' en los genes, en presencia de un cáncer. Estos biomarcadores permitirían un diagnóstico más temprano y preciso, la evaluación del riesgo de desarrollar cáncer y la administración de terapias personalizadas. En este sentido, la tarea de los Bioinformáticos consiste en desarrollar automatismos especializados en la gestión de grandes volúmenes de datos y en crear nuevos programas para comparar los resultados de la secuenciación con la colección de secuencias del genoma humano disponibles.

- Genómica comparativa

La base de la genómica comparativa consiste en el análisis del establecimiento de la correspondencia entre genes u otras características genómicas en diferentes organismos. Gracias a estos mapas intergenómicos es posible descubrir los procesos evolutivos responsables de la divergencia entre dos genomas. La complejidad de evolución genómica es tal, que los Bioinformáticos, encargados de desarrollar modelos y algoritmos, se han visto forzados a recurrir a un amplio abanico de técnicas algorítmicas, estadísticas y matemáticas.

- Predicción de la estructura proteica

La secuencia de aminoácidos que componen una proteína, la llamada estructura de la proteína, puede determinarse a partir de la secuencia del gen que codifica dicha proteína. El conocimiento de dicha estructura es fundamental para comprender la función de una proteína y los procedimientos utilizados en la predicción suelen estar basados en heurísticas que ofrecen, por lo general, buenos resultados.

Uno de los métodos de predicción con mayor aceptación en la comunidad Bioinformática es el basado en la homología: si la secuencia de un gen A, cuya función es conocida, es homóloga a la secuencia de un gen B de función desconocida, entonces es razonable pensar que B y A comparten la misma función. Actualmente, la única forma fiable de predecir estructuras proteicas consiste en utilizar la información acerca de la estructura de una proteína homóloga.

Un buen ejemplo de lo anterior es homología proteica existente entre la hemoglobina en humanos y la hemoglobina en las legumbres. Aunque ambas proteínas tienen secuencias de aminoácidos completamente diferentes, sus estructuras son prácticamente idénticas, indicando que el propósito de ambas es idéntico: transportar el oxígeno en el organismo.

- Modelado de sistemas biológicos

La biología de sistemas se encarga de utilizar el ordenador para simular los sistemas sub-celulares con el propósito de analizar y visualizar las complejas conexiones entre los procesos celulares.

- Análisis de imágenes de alta productividad

En la actualidad el computador es utilizado para acelerar o completamente automatizar el proceso, cuantificación y análisis de grandes cantidades información, extraídas de imágenes biomédicas. De hecho, los sistemas de análisis de imagen modernos han logrado aumentar la habilidad de los observadores, consiguiendo mejorar la precisión, objetividad o velocidad de sus análisis. Si además, el sistema de análisis se encuentra totalmente automatizado, la necesidad de un observador desaparece.

- Acoplamiento de proteínas

En las últimas dos décadas, se han descubierto decenas de miles de estructuras tridimensionales de proteínas. El cometido del acoplamiento de proteínas se basa precisamente en utilizar dichas estructuras tridimensionales para predecir posibles interacciones entre proteínas, sin llevar a cabo físicamente los experimentos. Aunque en la actualidad existen numerosos métodos que abordan este problema, el acoplamiento de proteínas es un campo con un largo camino por recorrer, especialmente porque la estructura 3D de las proteínas no es un elemento estático, sino que varía constantemente con el tiempo.

2.2 Alineamiento de secuencias.

Tal y como se ha mencionado anteriormente, el caso de estudio de esta tesis de máster es el alineamiento de secuencias, en particular, el alineamiento de metagenomas, debido a las necesidades computacionales que implica este problema. Por ello, el presente

punto se va a centrar en describir todos los aspectos relativos al alineamiento de secuencias, desde su origen y definición hasta los principales métodos que pueden encontrarse en la literatura.

2.2.1 Definición

El problema computacional del alineamiento de secuencias no es una cuestión nueva. De hecho, su origen se remonta a finales de los años 60, momento en el que se define, por primera vez, la distancia de edición entre dos cadenas, es decir, el mínimo número de operaciones de edición (inserciones, borrados y sustituciones) necesarias para transformar una cadena en otra. Sin embargo, desde el punto de vista bioinformático, el alineamiento de secuencias es el método que permite identificar regiones similares entre dos secuencias de ADN, ARN o proteínas, con el fin de inferir relaciones funcionales, estructurales o evolutivas entre ambas.

2.2.2 Tipos de alineamiento

El alineamiento de secuencias muy cortas o similares es una tarea poco costosa que puede ser realizada manualmente. No obstante, los problemas más interesantes de la bioinformática requieren el alineamiento de un gran número de secuencias largas y distintas que no pueden ser alineadas sin ayuda del computador. En lugar de alinear 'a mano' las secuencias, la tarea de los investigadores se ha centrado en desarrollar algoritmos que produzcan, de forma eficiente, alineamientos de alta calidad. Puesto que el alineamiento es un problema complejo, es posible abordarlo de maneras muy distintas, con diferentes niveles de sensibilidad, especificidad y prestaciones, según el objetivo que se persiga. Por tanto, el presente punto se ocupará de hacer un repaso a los principales tipos de alineamiento.

2.2.2.1 Alineamiento global

El alineamiento global entre dos cadenas $x = x1…xm$ y $y = y1…ym$ se define formalmente como la ubicación de huecos (representados con el carácter '-') al principio, al final o entre cualquier par de caracteres de las cadenas, de manera que las cadenas resultantes tengan la misma longitud, teniendo en cuenta que las asociaciones entre dos huecos no están permitidas. De forma equivalente, dada una matriz de dimensión (m+1) x (n+1) en la que eje x representa la secuencia 0$x1…xm$ de izquierda a derecha, y el eje y representa la secuencia 0$y1…yn$ de arriba abajo, un alineamiento se define como cualquier camino desde (0,0) a (m,n) utilizando movimientos abajo (+0,+1), derecha (+1,+0) y abajo-derecha (+1,+1), es decir, huecos en x, huecos en y y correspondencia entre dos caracteres, respectivamente. El alineamiento global se suele emplear en aquellos casos en los que las secuencias a alinear son similares y, además, poseen una longitud parecida.

Posiblemente el primer intento de resolver el problema del alineamiento global es el algoritmo de Needleman-Wunsch, aproximación que calcula la distancia de edición óptima entre dos cadenas mediante programación dinámica.

Los métodos de alineamiento global disponibles son, sin embargo, poco adecuados para alinear genomas completos por dos motivos. El primer inconveniente es el elevado coste temporal de estos algoritmos, aproximadamente cuadrático con la longitud de las secuencias, haciendo, por tanto, inabordable el alineamiento de miles de secuencias en un tiempo razonable. En segundo lugar, a diferencia de las secuencias de proteínas, las

secuencias de ADN de organismos relacionados suelen poseer pocas similitudes (normalmente concentradas en una región pequeña) siendo el alineamiento global menos útil en estas situaciones que otros tipos de alineamiento.

2.2.2.2 Alineamiento local

Dadas dos secuencias, el alineamiento local busca encontrar todos aquellos pares de sub-secuencias que posean un valor de similitud por encima de un determinado umbral. En contraste al global, el alineamiento local es una técnica útil cuando se pretende alinear secuencias que difieren claramente en tamaño y contenido, pero que son tentativas de poseer pequeños segmentos o regiones con un alto grado de similitud. Este tipo de alineamiento es, por tanto, idóneo cuando se pretende encontrar pares de genes (regiones de una secuencia) o elementos evolutivos entre dos genomas.

A continuación se muestra un ejemplo que refleja las diferencias existentes entre llevar a cabo el alineamiento local y global, utilizando las mismas secuencias.

```
Global FTFTALILLAVAV
       F--TAL-LLA-AV

Local  FTFTALILL-AVAV
       --FTAL-LLAAV--
```

Figura 1: Ejemplo que ilustra las diferencias entre el alineamiento local y global.

2.2.2.3 Alineamiento múltiple de secuencias

El alineamiento múltiple de secuencias es una extensión del alineamiento tradicional, que permite incorporar más de dos secuencias a la vez. De esta manera, los métodos de alineamiento múltiple tratan de producir el alineamiento del conjunto de secuencias proporcionadas como entrada. Este tipo de alineamiento es fundamental para identificar regiones comunes a una serie de secuencias con (hipotéticamente) relación evolutiva. Incluso cuando las regiones comunes a todas las secuencias presentan poca similitud, pueden estar revelando importantes funciones biológicas, inalteradas durante la evolución. Otra de las aplicaciones más importantes del alineamiento múltiple, en los últimos años, ha sido la construcción de árboles filogenéticos que establezcan relaciones evolutivas entre las especies.

Obviamente, calcular un alineamiento múltiple supone un esfuerzo computacional mucho mayor que el necesario para realizar un alineamiento simple. De hecho, la mayor parte de las formulaciones realizadas dan lugar a problemas NP-completos. No obstante, la utilidad de esta clase de alineamiento para la Bioinformática ha suscitado la aparición de diversos métodos.

El método más directo, sin lugar a dudas, es la programación dinámica. Aunque teóricamente es aplicable a cualquier número de secuencias, la adición de una nueva secuencia al conjunto, multiplica el tiempo y memoria necesarios para calcular el alineamiento óptimo por un factor proporcional a la longitud de dicha secuencia. Por este motivo, un esquema simple de programación dinámica pocas veces es utilizado para alinear más de tres o cuatro secuencias. En su lugar, suele ser preferible emplear aproximaciones basadas en heurísticas, entre las que destaca, por su extendido uso, el 'alineamiento progresivo'. El procedimiento del alineamiento progresivo consiste en

alinear, en primer lugar, las secuencias más parecidas utilizando un método de alineamiento simple y, a continuación, ir añadiendo 'progresivamente' secuencias o grupos menos relacionados hasta que el conjunto de secuencias haya sido incluido por completo en la solución. A continuación se ofrece un ejemplo (Figura 2) que trata de explicar el procedimiento expuesto.

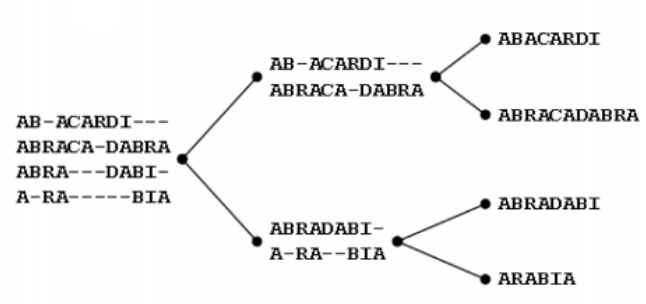

Figura 2: Ejemplo de alineamiento múltiple de cuatro secuencias utilizando el método progresivo. Nótese que el procedimiento mostrado comienza por la derecha.

2.3 Herramientas software para el alineamiento local de secuencias

Una de las conclusiones más importantes a las que se llegó en el punto anterior es que, debido a las características inherentes a las secuencias de ADN (muy diferentes en longitud y contenido), el alineamiento local es el más apropiado para comparar dicho tipo de secuencias [1]. Puesto que el caso de uso de esta tesis se basa en la utilización de secuencias de ADN, el presente punto se centrará en describir los algoritmos e implementaciones más relevantes del alineamiento local.

2.3.1 El algoritmo Smith-Waterman

Propuesto por primera vez en 1981 por Temple F. Smith y Michael S. Waterman, el algoritmo Smith-Waterman [2] es una variante del algoritmo Needleman-Wunsch [3] para el caso del alineamiento local de secuencias. Al igual que el algoritmo en el que está inspirado, Smith-Waterman hace uso de un esquema de programación dinámica, garantizando, de esta manera, el descubrimiento del alineamiento local óptimo (dependiente del sistema de puntuación que esté siendo utilizado).

2.3.1.1 Algoritmo

La definición formal del algoritmo es como sigue:

Dadas dos secuencias moleculares, A = a1 a2 … an y B = b1 b2 … bm, para hallar segmentos de secuencia muy similares, el primer paso consiste en construir la matriz H, inicializada de la siguiente manera:

$$H(i,0) = 0,\ 0 \leq i \leq m$$
$$H(0,j) = 0,\ 0 \leq j \leq n$$

Una vez inicializada la matriz H, el resto de valores se calculan siguiendo el esquema recursivo mostrado debajo. Los valores de H(i,j) representan la máxima similitud existente entre dos segmentos acabados en ai y bj, respectivamente.

$$H(i,j) = \max \begin{Bmatrix} 0 & \\ H(i-1,j-1) + w(a_i,b_j) & \text{Match/Mismatch} \\ H(i-1,j) + w(a_i,-) & \text{Deletion} \\ H(i,j-1) + w(-,b_j) & \text{Insertion} \end{Bmatrix}, \ 1 \le i \le m, 1 \le j \le n$$

Donde:

a, b = Cadenas sobre el alfabeto Σ

m = longitud(a)

n = longitud(b)

w(c,d) es el sistema de puntuación o pesos.

El esquema recursivo para calcular Hij tiene en cuenta los posibles de segmentos acabados en cualquier ai y bj:

(1) El cero es la condición de contorno de la recursión que se incluye para evitar obtener valores de similitud negativos (diferencia fundamental con el algoritmo Needleman-Wunsch).
(2) La segunda línea representa que ai y bj son el mismo o distinto símbolo.
(3) La tercera línea indica que se ha producido un borrado (desde el punto de vista de la secuencia de referencia A)
(4) Finalmente, la cuarta línea significa que se ha producido una inserción (desde el punto de vista de la secuencia de referencia A).

El par de segmentos con máxima similitud se encuentra localizando, primero, el máximo valor de H. A continuación se lleva a cabo un procedimiento 'backwards' o 'hacia atrás', mediante movimientos a la posición con el máximo valor de entre las siguientes posiciones: (i-1,j), (i,j-1) o (i-1,j-1). El proceso continua hasta que se alcanza una celda con valor 0. Para reconstruir el alineamiento debe tenerse en cuenta que un movimiento diagonal (i-1,j-1) se corresponde con alineamiento entre caracteres, un movimiento (i-1,j) implica un borrado y un movimiento (i,j-1) representa una inserción.

Si se quisiera encontrar el par de segmentos con la siguiente mejor similitud, el único requisito necesario sería comenzar el proceso desde el siguiente valor máximo de H, no asociado con la anterior traza.

2.3.1.2 Ejemplo de alineamiento con el algoritmo Smith-Waterman

Con el fin de aclarar la anterior explicación, se incluye el siguiente ejemplo:

- Secuencia 1 = ACACACTA
- Secuencia 2 = AGCACACA
- w(match) = +2
- w(a,-) = w(-,b) = w(mismatch) = -1

Según lo anterior, se construye la matriz H mostrada en la Figura 3.

$$H = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix}$$

Figura 3: Matriz H resultante para el ejemplo del algoritmo Smith-Waterman.

A la vista de la matriz H de la Figura 3, el valor máximo se encuentra en la posición (8,8). El consiguiente procedimiento 'hacia atrás' sería:

(8,8) -> (7,7) -> (7,6) -> (6,5) -> (5,4) -> (4,3) -> (3,2) -> (2,1) -> (1,1) -> (0,0)

Por tanto, el alineamiento final que se obtiene es:

- Secuencia 1 = A – C A C A C T A
- Secuencia 2 = AG C A C A C – A

2.3.1.3 Coste computacional

Puesto que Smith-Waterman es un algoritmo de programación dinámica, el coste computacional que lleva asociado es considerable: *O*(*m* x *n*) en tiempo y memoria, siendo *m* y *n* las longitudes de las secuencias a alinear. Por tanto, utilizar el algoritmo Smith-Waterman es muy ineficiente cuando se pretende alinear un número significativo de secuencias de gran longitud, como en el caso de un genoma. Este hecho ha motivado que, en muchas situaciones, el algoritmo Smith-Waterman sea reemplazado por herramientas más eficientes, entre las que destaca por su extendido uso e importancia en esta tesis, BLAST.

2.3.2 BLAST (Basic Local Alignment Search Tool)

BLAST o "Basic Local Alignment Search Tool" [4] es posiblemente la herramienta bioinformática más popular. El programa fue creado por Eugene Myers, Stephen Altschul, Warren Gish, David J. Lipman y Webb Miller en el NCBI (National Center for Biotechnology Information) [5] y publicado en 1990, convirtiéndose en uno de los artículos más citados de la década de los 90 (en la actualidad tiene más de 32.000 referencias).

La característica que distingue a BLAST de herramientas más tradicionales, como Smith-Waterman, es su énfasis en reducir el tiempo necesario para realizar el alineamiento local, a cambio de sacrificar la sensibilidad. Dicha eficiencia temporal se logra gracias al uso de un algoritmo heurístico, no garantizándose, por tanto, el descubrimiento de la solución (alineamiento) óptimo.

2.3.2.1 Algoritmo

En líneas generales, el funcionamiento del alineador es el siguiente: dada una secuencia de búsqueda (o 'query') y una secuencia contra la que buscar (también llamada secuencia objetivo o 'target') o una base de datos que contenga múltiples secuencias, BLAST encuentra todas las sub-secuencias en la base de datos que sean similares a las sub-secuencias contenidas en la base de datos, por encima de un determinado umbral. Típicamente, la secuencia de búsqueda suele ser mucho más pequeña que la secuencia de la base de datos.

La idea fundamental del algoritmo BLAST es que, en cualquier alineamiento estadísticamente significativo, suelen existir pares de segmentos muy similares (denominados en la literatura como HSPs o High Scoring Pairs). BLAST busca todos los HSP entre la secuencia de búsqueda y las secuencias de la base de datos, utilizando el algoritmo heurístico que se describe seguidamente:

1. Eliminar regiones simples y sub-secuencias repetidas de la secuencia de búsqueda. Se entiende por regiones simples a partes de la secuencia de búsqueda compuestas por pocos tipos de elementos. Una tarea de pre-proceso importante consiste en filtrar estas regiones que generan ruido y perjudican el descubrimiento de regiones realmente similares.
2. Construir un diccionario de palabras con k-letras. Tal y como se muestra en la Figura 4, el método consiste simplemente en ir listando las palabras de k-letras secuencialmente, hasta que la última letra de la secuencia de búsqueda haya sido incluida. Con fines didácticos se ha utilizado un valor de k igual a 3, aunque para el caso de secuencias de ADN se suele emplear un k igual a 11.

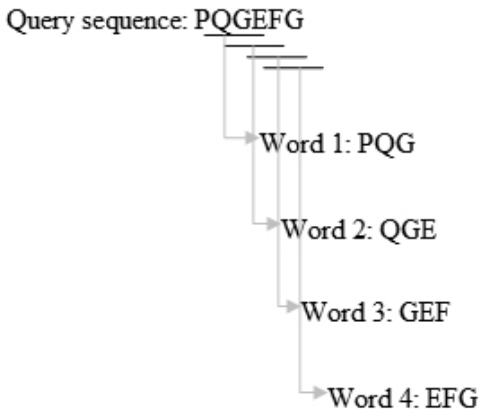

Figura 4: Método para construir el diccionario de palabras de k-letras (k=3).

3. Determinar palabras de k-letras que son tentativas de producir un HSP. Tomando una sola palabra del diccionario anterior, BLAST compara dicho elemento con cada una de las posibles palabras de k-letras, mediante una matriz de sustitución. Posteriormente, aquellas posibles palabras de k-letras con un valor de puntuación por debajo de un umbral T son desechadas.
4. Construir un árbol de búsqueda eficiente con las palabras que superaron el umbral. El objetivo de este punto es poder comparar rápidamente las palabras con alta puntuación con las encontradas en la base de datos.
5. El proceso 3-4 se repite para cada una de las palabras de la secuencia de búsqueda (diccionario creado en el punto 2).

6. Buscar en la base de datos coincidencias exactas con las palabras de alta puntuación. Tales coincidencias exactas servirán como punto de partida para tratar de descubrir regiones con alta similitud o HSPs.
7. Extender las coincidencias exactas con el fin de encontrar HSPs. A partir de las coincidencias exactas, se extiende el alineamiento en ambas direcciones (izquierda y derecha). El proceso de extensión no finaliza hasta que el valor total acumulado del HSP empieza a decrecer, es decir, se encuentran valores de puntuación negativos (véase el ejemplo simplificado de la Figura 5).

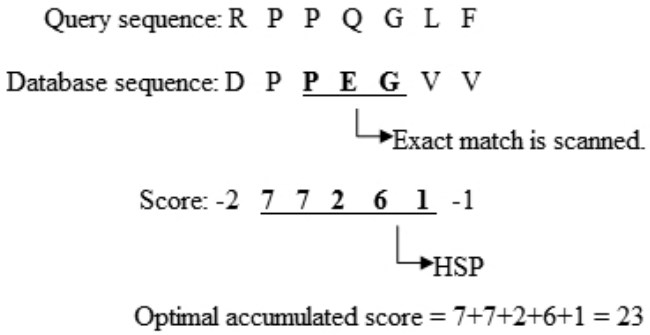

Figura 5: Ejemplo de búsqueda de un HSP.

8. Listar todos los posibles HSPs en la base de datos, cuya puntuación sea mayor que un valor empírico de corte S (determinado mediante la comparación de secuencias aleatorias).
9. Evaluar el significado estadístico de las puntuaciones de los HSPs restantes. Para ello, se hace uso del siguiente principio: las puntuaciones de alineamientos locales entre dos secuencias aleatorias siguen la distribución de Gumbel EVD (Extreme Value Distribution). Gracias a dicha distribución es posible calcular el valor esperado E (‘expected value’ en la literatura) de un HSP con puntuación X, es decir, el número de veces que dos secuencias aleatorias obtendrían una puntuación S mayor que X, debido a simple aleatoriedad.
10. Producir el alineamiento local con Smith-Waterman cuando una secuencia de la base de datos contiene más de un HSP. De esta manera, es posible aumentar la sensibilidad con alineamientos que contemplen la inclusión de huecos (inserciones o borrados). Lógicamente, en estos casos será necesario recalcular el valor esperado E.
11. Mostrar aquellas coincidencias con un ‘expected value’ E por debajo de un determinado umbral (parámetro del programa).

2.3.2.2 Sintonización del algoritmo BLAST

Una de las características más importantes de los métodos basados en índices, como el implementado por BLAST, es la posibilidad de sacrificar la velocidad de proceso a cambio de sensibilidad y viceversa. Por ello, este punto ofrece una serie de consejos, que permiten acelerar el proceso o aumentar la sensibilidad de los resultados.

1. La primera técnica consiste en aumentar o reducir el tamaño de las palabras del diccionario (k). Con un ‘k’ más pequeño es más probable encontrar un mayor número de coincidencias entre la secuencia de búsqueda y la base de datos. Sin embargo, también se aumenta el número de posibles coincidencias, requiriendo un mayor tiempo de análisis (mayor número de procesos de extensión).

2. Un método para aumentar la sensibilidad consiste en reducir el valor de 'k' a cambio de encontrar dos coincidencias cercanas, antes de llevar a cabo el proceso de extensión (ver Figura 6).

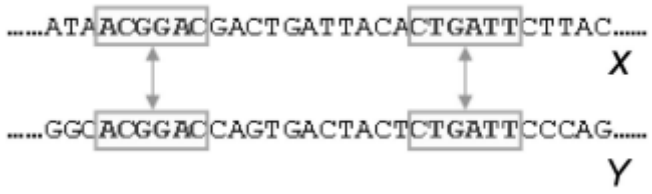

Figura 6: Ejemplo del método expuesto en el punto 2.

3. Otra forma de aumentar la sensibilidad conlleva aumentar el tamaño de k pero permitiendo que las coincidencias no sean exactas (una o varias diferencias como máximo). En la Figura 7 se incluye un ejemplo.

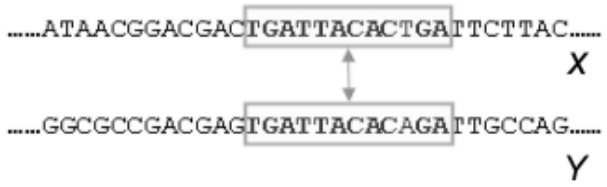

Figura 7: Ejemplo del método expuesto en el punto 3.

2.3.2.3 Programa BLAST

El propio NCBI proporciona una versión de código abierto del programa BLAST (http://www.ncbi.nlm.nih.gov/BLAST). Este hecho ha permitido la aparición de un gran número de variantes, desde optimizaciones hasta implementaciones ad-hoc que resuelven con soltura determinados problemas. Asimismo, el NCBI tiene habilitado un servidor web público que permite realizar búsquedas contra las bases de datos contenidas en el NCBI. Aunque, de esta manera, se garantiza que las bases de datos utilizadas están actualizadas, el uso limitado del servidor impide alinear un gran número de secuencias.

Para hacer uso de BLAST, en primer lugar es necesario conocer sus parámetros principales. Por un lado están los diferentes tipos de programas integrados en la herramienta. Hay un total de cinco programas BLAST diferentes, los cuales se pueden distinguir por el tipo de secuencia a buscar (ADN o proteína) y el tipo de base de datos contra la que se hace la búsqueda:

- BLASTP compara una secuencia de aminoácidos contra una base de datos de proteínas.
- BLASTN compara una secuencia de nucleótidos contra una base de datos de nucleótidos.
- BLASTX compara una secuencia de nucleótidos contra una base de datos de proteínas.
- TBLASTN compara una proteína contra una base de datos de nucleótidos.

- TBLASTX compara una secuencia de un nucleótido contra una base de datos de nucleótidos.

Además de la base de datos a utilizar, también es necesario especificar la base de datos sobre la que se quiere realizar la búsqueda. Una de las más utilizadas es la base de datos *nr* (colección de secuencias 'no redundantes' de GenBank y otros bancos de secuencias).

El otro parámetro fundamental es la secuencia de entrada, o fichero con las secuencias de entrada. Dichas secuencias deben estar en el formato estándar denominado FASTA o GI (número de registro asociado a la secuencia, en caso de tenerlo).

Existen otros parámetros importantes que deben ser ajustados. Entre ellos, destaca el E-Value y el Filter. Como ya se comentó en el apartado 2.3.1.1, el E-Value ('Expected value') es el umbral de importancia estadística para las coincidencias con las secuencias de la base de datos. Por otra lado, el valor Filter enmascara segmentos de la secuencia de entrada que tienen una complejidad composicional baja (es decir, el pre-proceso explicado en el punto 1 del algoritmo).

Además de todos los parámetros vistos, también es necesario especificar el número máximo de hits (resultados) y el formato de salida del propio resultado del proceso BLAST.

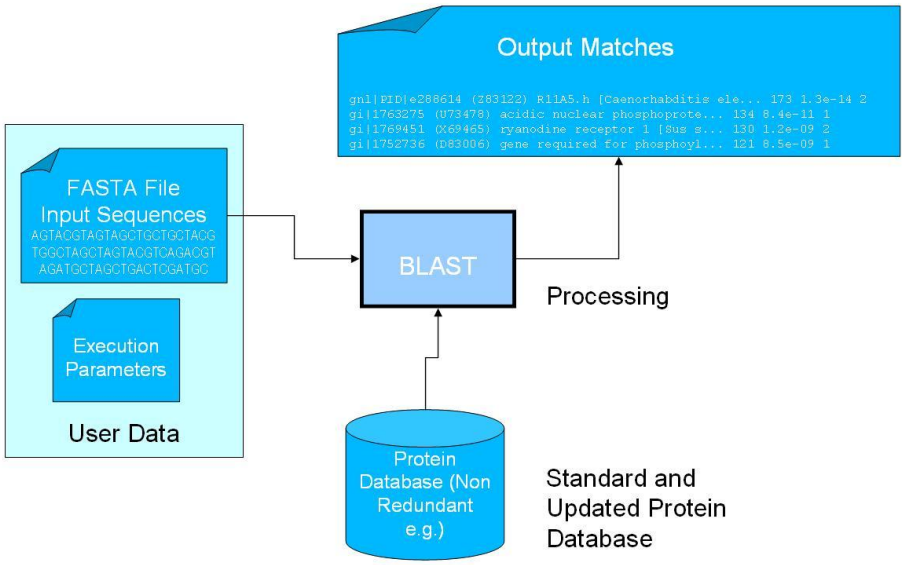

Figura 8: Proceso BLAST genérico.

La implementación mantenida por el NCBI consta de un toolkit escrito principalmente en C++ [6], aunque el núcleo del algoritmo está programado en C [7]. Dicho toolkit ofrece una serie de programas, de los cuales los más importantes para el propósito de esta tesis son:

- *blastall*: Este es el ejecutable encargado de realizar las comparaciones entre las secuencias de entrada y una base de datos. Los principales parámetros son el tipo de programa a emplear (blastn, blastp, blastx, tblastn, tblastx), la base de datos contra la que hacer las búsquedas (nr por defecto), el fichero de secuencias de entrada (ruta completa del fichero o stdin), el fichero de resultados, el E-value (el *expected value*), el formato de salida (XML [8], ASCII u otros) y el número máximo de BLAST hits (resultados).

- *formatdb*: el propósito de *formatdb* es formatear las bases de datos de entrada de proteínas o nucleótidos antes de ser utilizadas en una búsqueda. La base de datos fuente tiene que estar en formato FASTA o ASN.1. Este proceso sólo es necesario realizarlo una vez por versión de la base de datos. Una vez formateada, es posible trabajar sobre ella. En este caso los parámetros básicos son el fichero con la base de datos, el nombre de la base de datos formateada, el tipo de base de datos y su formato.

2.3.2.4 Implementaciones paralelas

Las versiones paralelas de BLAST disponibles están implementadas utilizando MPI y Pthreads y han sido llevadas a diversas plataformas incluyendo Windows, Linux, Solaris, Mac OS X y AIX. Puesto que mpiBLAST es la versión paralela utilizada en uno de los trabajos de la presente tesis, a continuación se describen las características más importantes de dicho programa.

- mpiBLAST

mpiBLAST [9] es una paralelización del programa BLAST proporcionado por el NCBI, de distribución pública, libre y abierta. mpiBLAST lleva a cabo una segmentación de la base de datos escogida y la distribuye entre los nodos de un clúster, permitiendo de esta manera que las búsquedas de BLAST sean procesadas en diferentes nodos de forma simultánea. Como su nombre indica, mpiBLAST está basado en MPI [10] y funciona sobre Linux, Windows y diversas variantes de Unix. A pesar de sus virtudes tiene como desventaja un comportamiento inestable con problemas de gran talla. Desde este punto de vista, se hace imprescindible el uso de la tecnología Grid para abordar problemas grandes con garantías. En cuanto a las ventajas respecto a la versión secuencial existen dos aspectos que merecen ser mencionados. En primer lugar, el hecho de que mpiBLAST divida la base de datos entre los nodos de un clúster. Esta característica permite que los segmentos de la base de datos puedan ser más pequeños, pudiendo residir por completo en la memoria del nodo, obteniendo de esta manera una aceleración significativa del proceso, como resultado de la eliminación del acceso a disco. Por otro lado, mpiBLAST permite a los usuarios aprovecharse de su eficiencia, incluso en clúster heterogéneos, ya que las demandas de comunicación entre procesadores son muy bajas.

Además, los requerimientos de mpiBLAST no son demasiado exigentes. Uno de los requisitos es tener una implementación de MPI instalada (funciona correctamente con las dos implementaciones libres más importantes, MPICH, LAM/MPI y OpenMPI). En las configuraciones habituales, MPIBLAST necesita también que los computadores tengan algún directorio de almacenamiento compartido. Este directorio puede ser una partición montada NFS, una compartición SAMBA, AFS o casi cualquier tipo de sistema de ficheros compartido por red. La localización de los directorios compartidos después se especifica en un fichero de configuración alojado en el directorio home. Con el fin de construir mpiBLAST desde los propios fuentes, también es necesario compilar una versión adecuada del *Toolbox* del NCBI, disponible via *internet*.

Como se podrá comprobar a continuación, los ejecutables son muy parecidos a los de la versión secuencial, siendo de interés para esta tesis los siguientes:

- *mpiblast*: es el equivalente a *blastall*. Los parámetros son prácticamente los mismos, por lo que no es necesario hacer grandes cambios en la línea de comandos.

- *mpiformatdb*: es el equivalente a formatdb. También comparte los mismos parámetros que el respectivo ejecutable secuencial. Es necesario formatear adecuadamente la base de datos de acuerdo con el número de procesadores a utilizar, previamente a una ejecución paralela, aunque este formateo puede ser reutilizado en sucesivas ejecuciones si el número de procesadores no cambia y se utiliza la misma base de referencia.

2.3.2.5 Coste computacional

La memoria requerida por BLAST para llevar a cabo los alineamientos es proporcional a la longitud de la secuencia de búsqueda mientras que el tiempo de ejecución es proporcional al número de coincidencias que inician una extensión. Por tanto podemos decir, que a diferencia de Smith-Waterman (coste cuadrático en tiempo y memoria), BLAST tiene una complejidad sub-cuadrática, obteniendo según algunas fuentes resultados 50 veces más rápido. De hecho, si el valor 'k' es suficientemente grande, pocas coincidencias dan lugar a la extensión, finalizando rápidamente el algoritmo. Esta mejora en eficiencia es lo que ha convertido a BLAST en la herramienta predilecta para llevar a cabo el alineamiento de grandes bases de datos de genomas. No obstante, puesto que BLAST está basado en un algoritmo heurístico, los resultados mostrados (hits) pueden no ser los mejores resultados. En efecto, en muchos casos, BLAST no logra devolver coincidencias que son difíciles de encontrar.

2.3.3 SSAHA

En 2001, Zemin Ning, Anthony J.Cox y James C. Mullikin presentaron un algoritmo de alineamiento local de nueva generación conocido como SSAHA (Sequence Search and Alignment by Hashing Algorithm) [11]. Este algoritmo aprovecha el hecho de que los ordenadores disponibles en 2001 poseen una memoria RAM suficientemente grande para albergar una tabla hash que describa una base de datos de múltiples gigabases de ADN. Ello permite realizar búsquedas mucho más rápido que aproximaciones basadas en índices y métodos de diccionario (como el caso de BLAST).

2.3.3.1 Algoritmo

- Construir la tabla Hash

El primer paso del algoritmo consiste en convertir la base de datos en una estructura tabla Hash. Dicha tabla Hash se almacena en memoria como dos estructuras de datos: una lista de posiciones L y un vector de punteros a L. Suponiendo que vaya a utilizarse para los alineamientos secuencias de ADN (4 símbolos: A, C, G, T), el número de elementos del vector de punteros será el número de posibles k-tuplas, es decir, 4^k punteros. Cada puntero apunta a la primera ocurrencia de la k-tupla en la base de datos y, a partir de esta posición, se almacenan consecutivamente el resto de ocurrencias de dicha k-tupla hasta que se encuentre un puntero asociado a una k-tupla distinta.

El proceso de construcción de la tabla Hash se lleva a cabo en dos pasos. En cada pasada sólo se consideran k-tuplas no superpuestas, es decir, aquellas k-tuplas de cualquier secuencia que comiencen en 0, k, 2k, etc. Por tanto, el cometido de la primera pasada es contar el número de ocurrencias para cada una de las posibles 4^k palabras, con el fin de reservar correctamente la cantidad de memoria necesaria para la lista de posiciones L. Una vez finalizada la primera pasada, es posible realizar un filtrado de

elementos de baja complejidad, eliminando aquellas palabras con una frecuencia por encima de un determinado umbral N. Finalmente, en la segunda pasada, se completa la información de la lista de posiciones L con la información de las posiciones. En la Figura 9 se muestra un ejemplo de tabla hash construida para el caso de k=2 (normalmente se usa un 'k' mayor) y una base de datos, formada por únicamente tres secuencias:

S1=GTGACGTCACTCTGAGGATCCCCTGGGTGTGG

S2=GTCAACTGCAACATGAGGAACATCGACAGGCCCAAGGTCTTCCT

S3=GGATCCCCTGTCCTCTCTGTCACATA

w	E(w)	Positions						
AA	0	(2, 19)						
AC	1	(1, 9)	(2, 5)	(2, 11)				
AG	2	(1, 15)	(2, 35)					
AT	3	(2, 13)	(3, 3)					
CA	4	(2, 3)	(2, 9)	(2, 21)	(2, 27)	(2, 33)	(3, 21)	(3, 23)
CC	5	(1, 21)	(2, 31)	(3, 5)	(3, 7)			
CG	6	(1, 5)						
CT	7	(1, 23)	(2, 39)	(2, 43)	(3, 13)	(3, 15)	(3, 17)	
GA	8	(1, 3)	(1, 17)	(2, 15)	(2, 25)			
GC	9							
GG	10	(1, 25)	(1, 31)	(2, 17)	(2, 29)	(3, 1)		
GT	11	(1, 1)	(1, 27)	(1, 29)	(2, 1)	(2, 37)	(3, 19)	
TA	12	(3, 25)						
TC	13	(1, 7)	(1, 11)	(1, 19)	(2, 23)	(2, 41)	(3, 11)	
TG	14	(1, 13)	(2, 7)	(3, 9)				
TT	15							

Figura 9: Tabla Hash construida para el ejemplo propuesto, siendo 'w' la lista de todas las posibles 2-tuplas, 'E(w)' el vector de punteros asociado a cada palabra y 'Positions' las lista de posiciones L.

- Búsqueda de secuencias

El siguiente paso del algoritmo es utilizar la tabla Hash para buscar ocurrencias de una secuencia de búsqueda Q en la base de datos objetivo. Para ello, se itera base a base de la secuencia Q, desde la base 0 hasta la base n-k (siendo n la longitud de Q). En cada paso 't', el algoritmo consulta la entrada de la tabla hash asociada a la k-tupla y compone una lista de posiciones:

(i1,j1), (i2,j2), …, (ir,jr)

A continuación, tomando las posiciones anteriores se crea una lista de 'hits':

H1=(i1,j1-t,j1), H2=(i2,j2-t,j2), …, Hr=(ir,jr-t,jr);

De izquierda a derecha, los componentes de los hits se denominan *índice*, *desplazamiento* y *offset*. Los hits obtenidos en cada paso 't', se van acumulando en una lista M que posteriormente será ordenada, primero por índice y después por desplazamiento. La parte final de proceso de búsqueda simplemente requerirá buscar una serie de hits de M (ordenada) con el mismo índice y desplazamiento (consecutivos). Si se quisiera que los alineamientos contemplasen inserciones/borrados, únicamente habría que permitir que el desplazamiento de hits consecutivos pudiera diferir en una pequeña cantidad de bases. Si además se desea incluir gaps o "huecos" en el alineamiento, se podría pensar en unir varios conjuntos de hits consecutivos, siempre y cuando pertenezcan a la misma secuencia de la base de datos.

Por ejemplo, si la secuencia de búsqueda fuera Q=TGCAACAT, la búsqueda de secuencias contra la base de datos anterior, se realizaría tal y como se muestra en la Figura 9. Por un lado, la columna 'Positions' muestra todas las ocurrencias de las

palabras de la secuencia 'query'. Por otro lado, la columna H muestra los hits creados a partir de las posiciones anteriores. Finalmente, la última columna de la tabla es la ordenación de los hits de H por índice y desplazamiento. Los cuatro hits resaltados en negrita poseen el mismo índice y desplazamiento, indicando la existencia de un alineamiento entre la secuencia de búsqueda o 'query' y la segunda secuencia de la base de datos.

2.3.3.2 Coste computacional

La complejidad espacial de SSAHA depende claramente de la memoria necesaria para albergar la tabla hash: 4^(k+1)+8W, siendo W el número de k-tuplas en la base de datos objetivo. Por lo que respecta a la complejidad temporal, esta puede ser dividida en dos porciones, el tiempo 'Thash' necesario para generar la tabla hash y el tiempo consumido por la búsqueda 'Tsearch'. Comparándolo con BLAST, el valor 'Thash' nunca llega a ser dos veces mayor que el tiempo necesario para dar formato a la misma base de datos con *formatdb*. En cualquier caso, si se utiliza varias veces la misma base de datos, sólo es necesario darle formato la primera vez, pudiendo ser guardada en disco o retenida en la memoria RAM para futuras búsquedas. Por otro lado, el tiempo de búsqueda 'Tsearch' está dominado por la ordenación de los hits, con un coste normalmente *O*(nlog(n)), siendo n la longitud de secuencia de búsqueda. Según sus autores, en un caso típico, SSAHA es tres o cuatro órdenes de magnitud más rápido que BLAST.

t	$w_t(Q)$	Positions	H	M
0	TG	(1, 13)	(1, 13, 13)	(1, 5, 9)
		(2, 7)	(2, 7, 7)	(1, 13, 13)
		(3, 9)	(3, 9, 9)	(2, −2, 3)
1	GC			(2, 1, 3)
2	CA	(2, 3)	(2, 1, 3)	(2, 1, 5)
		(2, 9)	(2, 7, 9)	(2, 4, 9)
		(2, 21)	(2, 19, 21)	**(2, 7, 7)**
		(2, 27)	(2, 25, 27)	**(2, 7, 9)**
		(2, 33)	(2, 31, 33)	**(2, 7, 11)**
		(3, 21)	(3, 19, 21)	**(2, 7, 13)**
		(3, 23)	(3, 21, 23)	(2, 16, 19)
3	AA	(2, 19)	(2, 16, 19)	(2, 16, 21)
4	AC	(1, 9)	(1, 5, 9)	(2, 19, 21)
		(2, 5)	(2, 1, 5)	(2, 22, 27)
		(2, 11)	(2, 7, 11)	(2, 25, 27)
5	CA	(2, 3)	(2, −2, 3)	(2, 28, 33)
		(2, 9)	(2, 4, 9)	(2, 31, 33)
		(2, 21)	(2, 16, 21)	(3, −3, 3)
		(2, 27)	(2, 22, 27)	(3, 9, 9)
		(2, 33)	(2, 28, 33)	(3, 16, 21)
		(3, 21)	(3, 16, 21)	(3, 18, 23)
		(3, 23)	(3, 18, 23)	(3, 19, 21)
6	AT	(2, 13)	(2, 7, 13)	(3, 21, 23)
		(3, 3)	(3, −3, 3)	

Figura 10: Proceso de búsqueda de secuencias para el ejemplo propuesto.

2.3.4 BWA (Burrows-Wheeler Alignment Tool)

La última herramienta de alineamiento local que se describirá en este apartado es un software actual, conocido como BWA o Burrows-Wheeler Alignment tool. Desarrollado en 2009 por Heng Li y Richard Durbin del Wellcome Trust Sanger Institute, BWA mejora la eficiencia de los métodos que hacen uso de tablas hash. El algoritmo que implementa esta herramienta es la transformada de Burrougs-Wheeler, utilizada en herramientas de compresión y búsqueda eficiente, por lo que su funcionamiento se describirá de forma muy somera. Para comprender al detalle todos

los aspectos del algoritmo, se recomienda revisar el artículo original, referenciado en la bibliografía de esta tesis [12].

2.3.4.1 Descripción general del algoritmo

El algoritmo de BWA se basa en dos conceptos: los árboles de prefijos y la transformada de Burrows-Wheeler.

Un árbol de prefijos para una cadena X se define como aquel árbol en el que cada rama está etiquetada y la concatenación de las etiquetas de las ramas desde un nodo cualquiera a la raíz es una sub-cadena de X. De esta forma, explorando un árbol de prefijos es posible comprobar si una determinada secuencia de búsqueda W es sub-cadena exacta de una secuencia de referencia en tiempo O(|W|). Si por el contrario, se pretende encontrar coincidencias no exactas (permitir fallos) la búsqueda deberá realizarse por todos los posibles caminos del árbol (con el consiguiente aumento de ramas consultadas y el correspondiente aumento en el tiempo de búsqueda). Por ejemplo, si la cadena de referencia fuera 'GOOGOL' el árbol de prefijos resultante tendría el aspecto mostrado en la Figura 11.

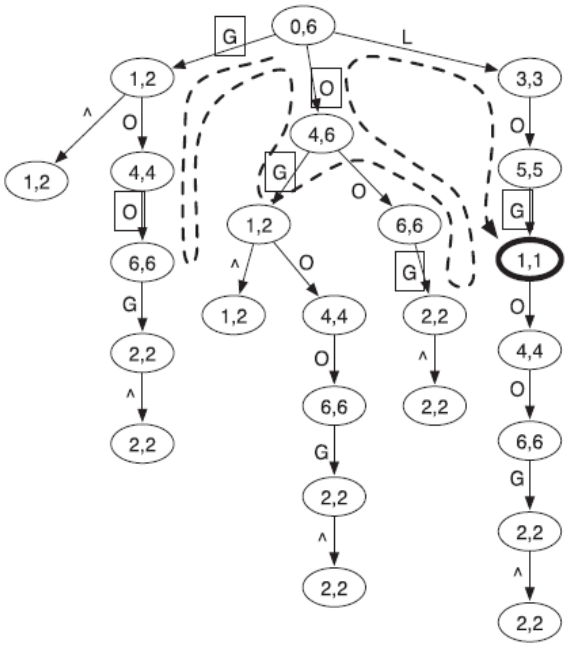

Figura 11: Árbol de prefijos para la cadena de búsqueda 'GOOGOL'.

La utilidad de la transformada de Burrows-Wheeler es obtener de información sobre los prefijos de la cadena de búsqueda. Para ello, se calcula el vector de sufijos y la transformada, tal y como muestra el ejemplo de la Figura 12.

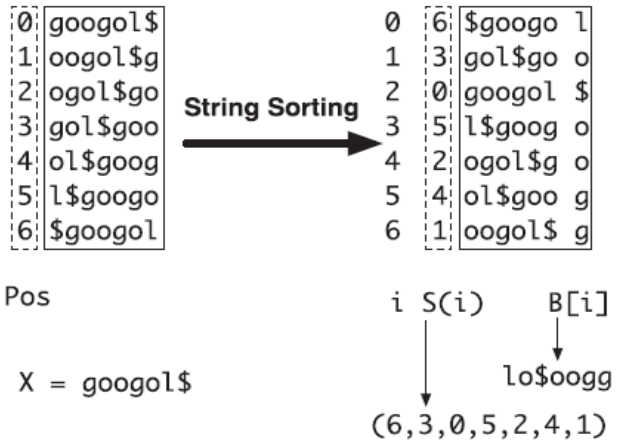

Figura 12: Construcción del vector de sufijos y la trasformada BW para la cadena de búsqueda X=googol$. La cadena X se rota para generar siete cadenas, que seguidamente son ordenadas lexicográficamente. Tras la ordenación, los índices de las cadenas definen las posiciones de los símbolos del vector de sufijos (6, 3, 0, 5, 2, 4, 1) y la concatenación de los últimos símbolos de las cadenas rotadas dan lugar a la transformada de BW: lo$oogg.

Haciendo uso de esta información es posible llevar a cabo una poda efectiva en las búsquedas (especialmente efectivo en el caso de exactas pero adecuado para inexactas si el número de fallos permitidos es reducido), mejorando notablemente la eficiencia del proceso. Por tanto, el alineamiento de secuencias en BWA puede ser visto como una búsqueda en un árbol de prefijos con poda (derivada de la transformada de BW).

2.3.4.2 Coste computacional

Según el análisis de complejidad espacial realizado por los autores de BWA, la cantidad de memoria requerida por dicho software de alineamiento es un poco superior a la requerida por herramientas basadas en tablas hash, como SSAHA. BWA utiliza la memoria para almacenar, entre otras cosas, el árbol de prefijos, el vector de sufijos y la transformada de Burrows-Wheeler. No obstante, mientras la memoria necesaria por BWA es independiente del número de secuencias a examinar (las estructuras de datos se crean únicamente para la secuencia de búsqueda), en los métodos basados en tablas Hash la memoria necesaria es lineal con el número de secuencias de la base de datos. Respecto al coste temporal, BWA es alrededor de 10-20 veces más rápido que los métodos de tablas Hash, obteniendo además una precisión similar en los resultados. El único inconveniente de BWA son las malas prestaciones que se obtienen cuando el tamaño de la secuencia a alinear es muy grande o cuando el número de fallos permitidos es muy grande.

2.4 Metagenómica

La presente tesis utiliza como caso de estudio para los desarrollos realizados el alineamiento de metagenomas. Además de dotar de realismo a los experimentos expuestos en el apartado de desarrollos, el caso de estudio es útil para los proyectos de colaboración realizados en el marco de esta tesis. Con el fin de comprender mejor dicho caso de uso, este punto comenzará introduciendo el término metagenómica y finalizará describiendo brevemente el metagenoma más representativo utilizado en esta tesis.

2.4.1 Concepto

La posibilidad de secuenciar genomas completos de microorganismos ha sido uno de los avances científicos más importantes de los últimos años en el campo de la biología. La genómica ha mejorado no sólo el conocimiento sobre los procesos evolutivos que dan forma a los seres vivos sino también ha permitido el desarrollo de interesantes aplicaciones clínicas y biotecnológicas. El secuenciamiento de genomas sólo es posible si el microorganismo puede obtenerse en grandes cantidades, es decir, si puede ser cultivado. Pero en la mayoría de los casos, el secuenciamiento aislado del ADN de muchos microorganismos (se estima que el 99%) no es viable, debido a la coexistencia necesaria entre muchos microorganismos. El término metagenómica es un nuevo tecnicismo, también denominado genómica ambiental, ecogenómica o genómica comunitaria, que denota el estudio de material genético recuperado directamente de muestras ambientales. Uno de los pilares sobre los que se apoya la metagenómica es la aparición de nuevas tecnologías de secuenciamiento (métodos 'Shotgun'), las cuales permiten procesar secuencias de ADN a una mayor velocidad. En lugar de enriquecer el contenido de ADN de las muestras, la metagenómica opta por secuenciar enormes cantidades de ADN ambiental, eliminando de esta manera la necesidad de cultivar el microorganismo. Por tanto, la clave de la metagenómica reside en el hecho de poder analizar un conjunto de genomas de forma análoga al estudio de un único genoma.

En algunos proyectos metagenómicos, la cantidad de secuencias de ADN que se ha obtenido es equivalente al genoma de 200 bacterias. El análisis de estas secuencias puede realizarse con procedimientos y algoritmos bioinformáticos convencionales (como el alineamiento de secuencias), aunque el enorme número de secuencias a procesar supone un gran reto computacional. Afortunadamente, tal y como podrá comprobarse en los desarrollos de esta tesis, el uso combinado de herramientas bioinformáticas como BLAST e infraestructuras de e-Ciencia hace posible abordar proyectos metagenómicos de gran magnitud.

2.4.2 Metagenoma del mar de los Sargazos

El mar de los Sargazos es una región del Océano Atlántico septentrional situada entre los meridianos 70º y 40º O y los paralelos 25º y 35º N, con una superficie total aproximada de 3.500.000 kilómetros cuadrados. Se caracteriza por una ausencia frecuente de viento, corrientes marinas y abundancia de plancton y algas (que constituyen auténticos bosques marinos superficiales que se extienden de horizonte a horizonte).

En las aguas superficiales, donde llega la luz, abunda el plancton vegetal, que consume sales como los fosfatos y los nitratos. Debido a la diferencia de densidad, el agua de la superficie apenas se mezcla con el agua fría y rica en minerales de las capas inferiores, que podrían restaurar las sales consumidas. Por esta razón, en las regiones superiores del mar de los Sargazos prácticamente no hay vida animal, y no tendría ningún interés biológico si no fuera por el alga que le da nombre, el sargazo, el cual forma grandes campos, llenos de microorganismos marinos. Esta riqueza biológica supone una fuente importantísima para la investigación farmacéutica.

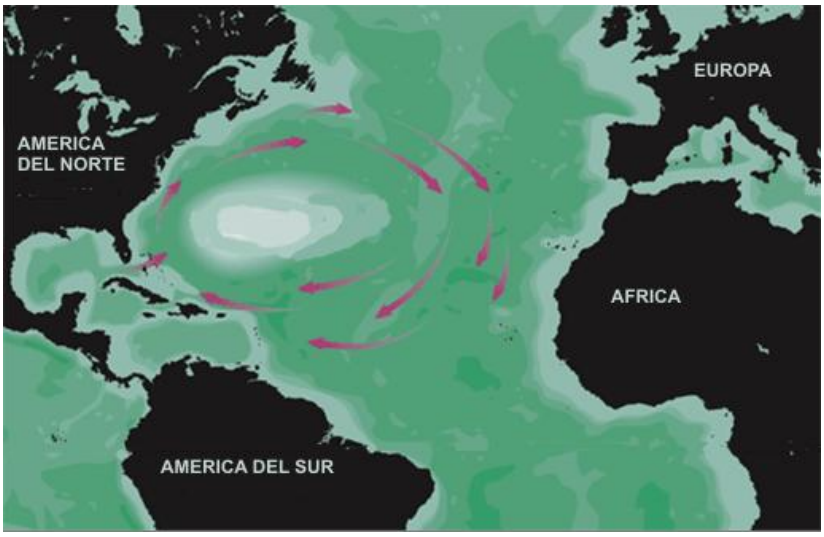

Figura 13: Ubicación geográfica del mar de los Sargazos.

2.5 La tecnología Grid

Esta sección pretende realizar una introducción a la tecnología Grid, analizando brevemente la historia de la computación científica que ha propiciado la aparición de dicha tecnología. Posteriormente, se describen los dos middleware Grid más extendidos para la implementación de estas arquitecturas. Por un lado, el estándar *de facto* actual empleado para la construcción de Grids, Globus Toolkit. Por otro lado, el middleware gLite, desarrollado en el marco del proyecto europeo EGEE, que soporta el mayor despliegue computacional en Grid existente en la actualidad. Finalmente, se hace un repaso a los proyectos EGEE, EELA y la NGI Española, que ofrecen la infraestructura física donde se han llevado a cabo gran parte de los experimentos realizados en esta tesis.

2.5.1 Orígenes

La evolución de la ciencia siempre ha venido marcada por la necesidad de utilizar las máximas prestaciones de los recursos de computación existentes. De hecho, el incremento de la capacidad de cómputo de los sistemas ha permitido abordar problemas de mayor envergadura.

En este sentido, la evolución de la capacidad de cómputo ha venido marcada fundamentalmente por la Ley de Moore, que establece que el número de transistores de los circuitos integrados se duplica cada 18 meses aproximadamente. Por lo tanto, cada vez existen máquinas más potentes con mayor capacidad de proceso. No es de extrañar por tanto que un PC actual disponga de una potencia de cómputo comparable a la de un supercomputador de la década anterior.

La solución tradicional que se ha utilizado para la ejecución de las aplicaciones científicas ha sido una computación centralizada de altas prestaciones, basada en un servidor central, al que se conectan los usuarios para ejecutar sus aplicaciones. Estas máquinas multiprocesador de memoria compartida sufren problemas de falta de escalabilidad, siendo equipos muy costosos que la mayor parte del tiempo

desaprovechan ciclos de reloj, ya que la demanda de potencia suele ser muy puntual. Una alternativa que surge a los sistemas multiprocesadores viene dada por los sistemas multicomputador, dando lugar a la computación basada en clusters de PCs, que ofrecen un buen ratio coste/prestaciones para la ejecución de aplicaciones paralelas.

Una extensión al concepto de computación basada en clusters de PCs es la llamada computación basada en Intranet, que permite el uso de los equipos de una organización para la ejecución de trabajos secuenciales o paralelos por medio de una herramienta de control de la carga. Estas herramientas permiten la ejecución de tareas en los recursos de una organización para aprovechar los ciclos ociosos de los procesadores. De esta manera, se consigue aumentar la productividad al realizar múltiples ejecuciones concurrentes en los recursos infrautilizados. Esta estrategia de computación aprovecha los recursos informáticos y facilita la escalabilidad al permitir añadir nuevas máquinas al sistema de forma sencilla. Existe bastante software dedicado a esta filosofía de computación, como Sun Grid Engine [13] de Sun Microsystems, Condor [14] de la Universidad de Wisconsin, InnerGrid [15] de GridSystems o Nimrod [16] de la Universidad de Monash.

Sin embargo, cuando el ámbito de las aplicaciones atraviesa las fronteras de una única organización, surgen una serie de problemas que no aparecen en la computación basada en Intranet. El hecho de trabajar con diferentes dominios de administración, que pueden estar sujetos a diversas políticas de seguridad, gestión y localización, supone un nuevo desafío para las estrategias de computación. Por lo tanto, la creación de una infraestructura de computación distribuida, que aglutine recursos de diferentes centros geográficamente dispersos, requiere tecnologías y herramientas avanzadas.

2.5.2 La computación basada en Grid

Con el incremento en el ancho de banda de las redes de comunicación se impulsó la idea de enlazar distintos recursos, geográficamente distribuidos, para crear una infraestructura global de computación conocida como el Grid. El término Grid fue acuñado a mediados de los 90 para proponer una infraestructura de computación distribuida para la ingeniería y ciencia avanzada.

Inicialmente, los conceptos y tecnologías relacionados con el Grid fueron desarrollados para permitir la compartición de recursos en las colaboraciones científicas. La necesidad de este tipo de colaboraciones no era únicamente la compartición de datos experimentales sino también aplicaciones, recursos computacionales e instrumental específico como telescopios y microscopios. Ante esta situación emergieron multitud de aplicaciones precisando computación y almacenamiento de datos distribuido para la ejecución de aplicaciones computacionalmente complejas, el acceso a bancos de datos distribuidos, la visualización colaborativa de datos y la compartición de instrumentos. Por lo tanto, existía una demanda de compartición de recursos coordinada para la resolución de problemas, en el marco de colaboraciones dinámicas, entre múltiples instituciones [17].

En realidad, el Grid puede ser considerado como un servicio para la compartición de potencia computacional y de capacidad de almacenamiento, al igual que el Web es un servicio para la compartición de información a través de Internet.

Por lo tanto, la computación en Grid es una tecnología que permite la compartición de recursos entre máquinas pertenecientes a dominios de administración diferentes de manera transparente, eficiente y segura.

La principal idea que subyace a las tecnologías Grid es ofrecer una interfaz homogénea y estándar para poder acceder a los recursos. Estos recursos pueden ser máquinas de cálculo como supercomputadores, clusters de PCs o sistemas de almacenamiento, aunque también se pueden compartir fuentes de información como bases de datos o incluso instrumentos científicos.

Uno de los puntos importantes que ofrece esta tecnología es que permite la interconexión de recursos entre organizaciones diferentes respetando las políticas internas de seguridad, así como todo el software de gestión de recursos que ya disponga la organización. Además, las tecnologías Grid permiten el uso colaborativo de diferentes recursos hardware para crear un conjunto de recursos de computación formado por equipos heterogéneos y geográficamente distribuidos. Mediante el empleo de esta tecnología es posible superar los límites computacionales de una única organización, accediendo a nodos de cómputo de otras entidades.

2.5.3 La Alianza Globus y el Middleware Globus Toolkit

La alianza Globus [18] es una comunidad de organizaciones e individuos que desarrollan tecnologías fundamentales para el Grid. Para esta alianza, un Grid permite a los individuos la compartición de potencia computacional, bases de datos, instrumentos y otras herramientas on-line de manera segura, a través de las barreras institucionales o geográficas sin sacrificar la autonomía local.

El consorcio Globus, formado tanto por académicos como por grandes empresas como HP, IBM e Intel, se encarga en gran parte del desarrollo de código abierto de Globus Toolkit (GT) junto con una gran comunidad de usuarios.

GT es una herramienta de libre distribución que permite la creación de Grids, facilitando la compartición de recursos computacionales de manera segura, a través de diferentes organizaciones, sin sacrificar la autonomía local de los diferentes dominios de administración que integren el Grid. Este paquete incluye servicios y librerías para la monitorización y gestión de recursos así como para la seguridad y gestión de ficheros. En un entorno de múltiples organizaciones aparece el problema de la heterogeneidad, donde existen ordenadores de arquitecturas diferentes y redes de datos diversas, que pueden dar lugar a problemas de incompatibilidad.

GT ha sido diseñado para eliminar los principales obstáculos que surgen de una colaboración entre organizaciones. Los principales servicios, interfaces y protocolos que ofrece Globus permiten a los usuarios el acceso a los recursos remotos como si se trataran de recursos locales, al mismo tiempo que preserva el control local sobre quién y cuándo se accede a los recursos compartidos.

La versión 1.0 de Globus (GT1) [19] apareció en el año 1998. La versión 2.0 (GT2) estuvo disponible en el año 2002. Globus Toolkit 3.0 (GT3) [20] apareció en el año 2003 y la versión 4.0 (GT4) [21] apareció en el año 2005, siendo la actual la versión 5.0 (GT5). Inicialmente, tanto GT1 como GT2 empleaban interfaces propietarias para la comunicación entre los diferentes servicios aunque generalmente se implementaban utilizando protocolos estándares como LDAP (Lightweight Directory Access Protocol), HTTP (Hypertext Transfer Protocol) o FTP (File Transfer Protocol). La aparición de

GT2 supuso la adopción de Globus Toolkit por parte de numerosos proyectos de investigación relacionados con el Grid, y se convirtió automáticamente en un estándar de facto para el despliegue de Grids computacionales. Con el lanzamiento de GT3 se realizó un acercamiento de este software a la tecnología de Servicios Web (Web Services, WS) [22] con el objetivo de exponer interfaces y comunicación basada en protocolos estándar para la interacción entre diferentes servicios. GT4 representó un importante avance en términos de funcionalidad, conformidad a estándares y usabilidad.

Globus Toolkit no es el único middleware Grid existente en la actualidad. Sin embargo, ha sido el marco de referencia de desarrollos posteriores llevados a cabo por diferentes proyectos de investigación. Globus Toolkit será una herramienta básica en los desarrollos del presente trabajo, puesto que se utilizarán muchas de las herramientas que proporciona y por el hecho de que el middleware utilizado, gLite, hace uso de Globus Toolkit.

2.5.3.1 Componentes Principales de Globus Toolkit

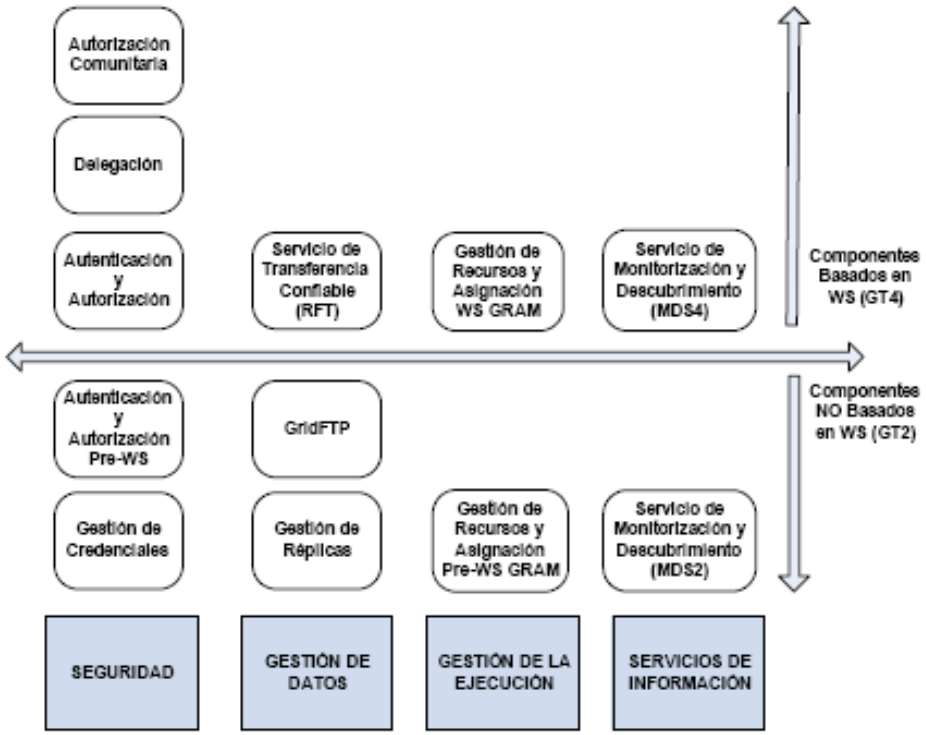

Figura 13: Diagrama de los principales componentes de GT5 en cada una de las cuatro áreas principales.

La Figura 13 muestra los principales componentes ofrecidos por GT en cada una de las principales áreas involucradas en el Grid: Seguridad, Gestión de Datos, Gestión de la Ejecución y Servicios de Información. En ella se observa cómo se realiza una distinción entre los componentes heredados de GT2, que no están orientados a servicios, y los más recientes incluidos en GT5. A continuación se detalla brevemente la funcionalidad ofrecida en cada una de estas áreas:

- Gestión de la ejecución

El servicio GRAM (Globus Resource Allocation Manager) simplifica el uso de sistemas remotos proporcionando una interfaz estándar para la ejecución de tareas en ellos. GRAM reduce el número de mecanismos necesarios para el uso de recursos remotos ya que cada sistema puede emplear, a priori, multitud de mecanismos de gestión como planificadores locales, gestores de colas o sistemas de reservas diferentes (Local Resource Management System, LRMS). De esta manera, el usuario y desarrollador de aplicaciones únicamente ha de conocer los servicios que ofrece GRAM, en lugar de conocer los mecanismos particulares necesarios para interactuar con los LRMS para la ejecución de aplicaciones en cada una de las máquinas del Grid. Nótese en la Figura 13 que se distingue entre el servicio pre-WS GRAM, desarrollado originalmente en GT2, y el servicio WS GRAM, desarrollado en GT4.

- Servicios de Información

Este apartado incluye al servicio MDS (Monitoring and Discovery Service) [23], encargado de proporcionar información sobre los recursos existentes en un Grid. Este servicio proporciona un mecanismo estándar para la publicación y el acceso a la información de los recursos computacionales. La implementación de MDS, como componente pre-WS (MDS2), involucra la existencia de una estructura jerárquica compuesta por dos elementos principales. Por un lado está el servicio GRIS (Grid Resource Information Service) [23], encargado de la recopilación de información en un recurso. Por otro lado, el servicio GIIS (Grid Index Information Service) [23] permite la creación de un directorio de información agregada que aglutina los datos recopilados por parte de múltiples servidores GRIS. El servicio MDS2 utiliza el protocolo LDAP para el acceso a la información publicada tanto por el recurso computacional como el servidor GIIS. En GT4, el servicio MDS4 es un servicio Grid que proporciona la información en formato XML a través de las interfaces requeridas empleando el lenguaje de consulta XPath (XML Path Language) [24].

La importancia del servicio MDS radica en que permite publicar el estado de los recursos de un Grid como puede ser la carga del procesador, la memoria disponible, el sistema operativo, así como información proporcionada por los LRMS instalados en el recurso.

- Gestión de Datos

La gestión de datos incluye la habilidad de acceder y gestionar los datos dentro de un entorno Grid. Este módulo incluye el componente GridFTP, encargado de realizar transferencias de datos seguras y de altas prestaciones entre los recursos computacionales de un Grid. GridFTP es un protocolo de transferencia de datos basado en el popular protocolo FTP, y optimizado para redes de área extensa (WANs) con grandes anchos de banda.

- Seguridad

Los aspectos de seguridad dentro de un Grid computacional son realmente importantes, dado que involucra el acceso a recursos de diferentes organizaciones.

En primer lugar, es importante realizar un proceso de *autenticación* que verifique la identidad de todos los participantes en el Grid, ya sean usuarios, recursos

computacionales o servicios Grid. En segundo lugar, se debe garantizar que los recursos únicamente sean accedidos por usuarios con ese privilegio y, por lo tanto, debe producirse un proceso de *autorización*. Nótese que la autorización está estrechamente ligada a la autenticación, puesto que antes de conceder un permiso de acceso es necesario verificar la identidad del cliente. En tercer lugar, a menudo es imprescindible garantizar la *privacidad* en la comunicación entre un cliente y el servicio al que accede, de manera que ésta únicamente sea entendible por ambos. Si una tercera persona consigue interceptar un mensaje entre ambos no debe ser capaz de entender el contenido de la conversación. Finalmente, la propiedad de *integridad* garantiza que el mensaje transmitido llegue al destinatario tal y como fue enviado por el cliente. Por lo tanto, si una tercera persona consigue introducir alguna modificación en el mensaje, el destinatario debe poder detectar que ha sido modificado desde que fue enviado.

Para poder dar una respuesta a los problemas de seguridad que surgen en un Grid, GT incluye una capa de seguridad denominada GSI (Grid Security Infrastructure). GSI utiliza los conceptos de criptografía de clave pública, también conocida como criptografía asimétrica. La funcionalidad ofrecida se puede resumir en tres puntos:

1. Ofrecer servicios de comunicación segura, autenticada y confidencial, entre los elementos de un Grid computacional.
2. Soportar esquemas de seguridad a través de las diferentes organizaciones involucradas en un Grid, descartando la creación de un sistema de seguridad gestionado de manera central.
3. Permitir la identificación única para los usuarios del Grid, incluyendo la delegación de credenciales entre recursos.

Globus proporciona un API de autenticación basado en tecnologías de infraestructura de clave pública (PKI) como los certificados X.509 y TLS. En particular, los experimentos descritos en esta tesis hacen uso de certificados X.509 para la autenticación.

2.5.4 El middleware gLite

El middleware gLite [25] pretende dar solución a la problemática del despliegue y utilización de una infraestructura Grid. Los usuarios de una infraestructura Grid basada en gLite se agrupan en diferentes VOs.

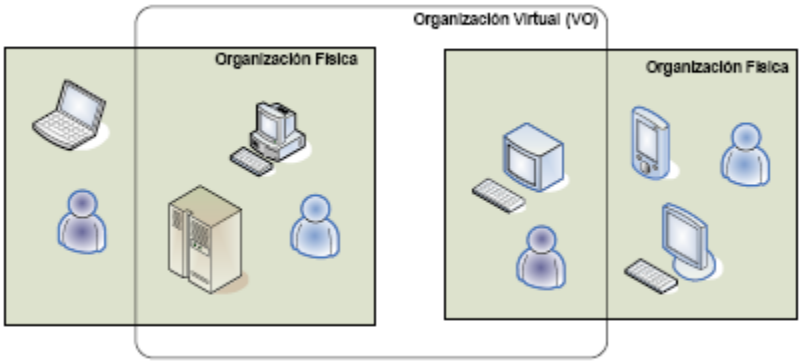

Figura 15: Estructura de una Organización Virtual que aglutina usuarios y recursos de diferentes organizaciones físicas.

Tal y como se muestra en la Figura 15, una organización virtual engloba un conjunto de individuos y recursos computacionales que permiten la resolución de problemas de manera colaborativa u otros propósitos. Las organizaciones virtuales proporcionan el contexto adecuado para vincular a los usuarios un conjunto de recursos. La compartición de recursos dentro de una VO es altamente controlada, donde los proveedores y consumidores de recursos están claramente diferenciados. Actualmente, las VOs mantienen una correspondencia directa con proyectos relacionados con los experimentos llevados a cabo, como por ejemplo ALICE, ATLAS, CMS y LHCb. En el marco del proyecto EGEE se han incoporado un total de 140 VOs, como *biomed*, que agrupa a la comunidad de usuarios a la que pertenece el trabajo de esta tesis (bioinformática).

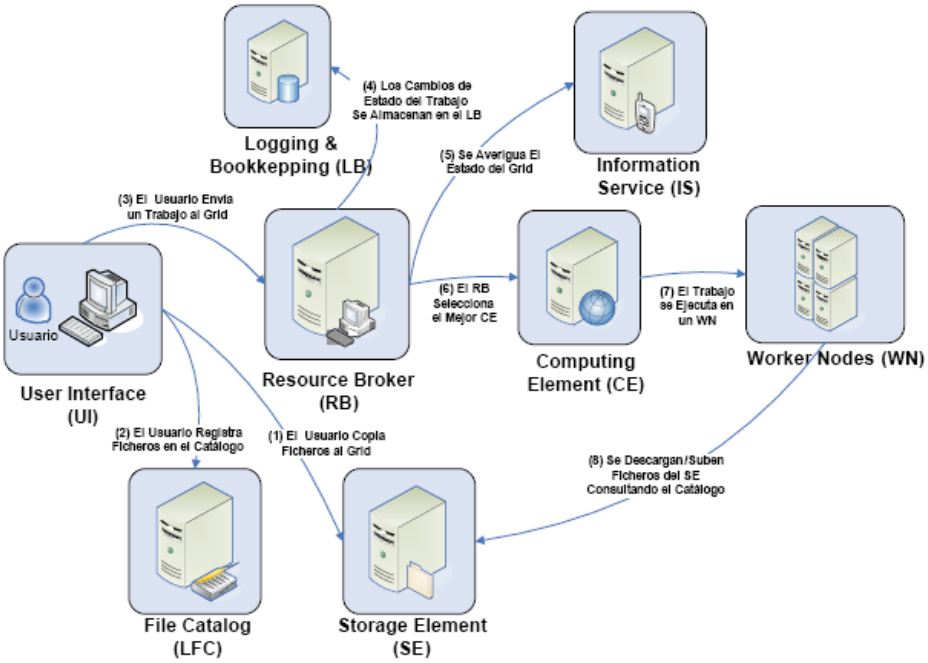

Figura 16: Arquitectura general del middleware gLite.

La Figura 16 muestra un esquema de la arquitectura general del middleware gLite. A continuación se describen los principales componentes encontrados en un despliegue gLite. Como norma general, cada servicio corresponde a una máquina diferente, aunque es posible combinar varios servicios en una misma máquina:

- User Interface (UI). Es el punto de entrada a un Grid basado en gLite donde el usuario dispone tanto de su certificado personal como de su clave privada. Desde un UI el usuario puede ser autenticado y autorizado para acceder a los recursos de gLite. Las principales acciones que pueden realizarse desde un UI son: lanzamiento de trabajos para su ejecución; obtener un listado de máquinas disponibles para ejecutar un trabajo; averiguar el estado de los trabajos y cancelarlos; obtener el resultado de aquellos finalizados y gestionar ficheros y réplicas para ser almacenadas en el Grid.
- Workload Management System (WMS). Este servicio se encarga de recibir las peticiones de ejecución de trabajos desde los UI y gestionar su ejecución en los diferentes elementos de computación (CE). Para ello debe consultar el estado del

Grid al Sistema de Información (IS). Este servicio es, por tanto, el encargado de realizar la distribución de la carga computacional entre los diferentes recursos del Grid. En un despliegue gLite pueden haber múltiples WMS.

- Computing Element (CE). Este servicio representa a un elemento de computación encargado de recibir solicitudes de ejecución de trabajos. Un CE viene identificado por una cadena que referencia a una determinada cola de ejecución en la máquina. Diferentes colas sobre una misma máquina son consideradas diferentes CEs. Por lo general, este servicio se instala en máquinas que constan de un gestor de trabajos (LRMS) para poder asignar las tareas recibidas a los diferentes nodos de computación (Worker Nodes). El ejemplo más claro consiste en instalar el CE en el nodo principal de un cluster, donde los nodos internos de cálculo son WN. Worker Node (WN). Este servicio se encarga de gestionar la ejecución de los trabajos recibidos por un CE. Por lo general, los WN disponen de conectividad a la red, permitiendo el acceso a otros servicios.
- Storage Element (SE). Este servicio proporciona un acceso uniforme a los recursos de almacenamiento. Un SE puede controlar desde simples servidores de disco, arrays de disco de tamaño medio e incluso dispositivos de almacenamiento masivo. Por lo general, cada organización que aporta recursos de computación a gLite también comparte espacio de almacenamiento ofreciendo un SE para el almacenamiento temporal de ficheros.
- Logging & Bookkeeping (LB). El LB permite almacenar la evolución del estado de los trabajos a lo largo del ciclo de vida. Por ello, las diferentes fases que atraviesa un trabajo son registradas por este servicio. Esta información puede ser accedida tanto por el usuario como por otros servicios para conocer en todo momento la evolución de las tareas.
- LCG File Catalog (LFC). En gLite los ficheros de datos pueden estar replicados en diferentes SE e, idealmente, no es necesario que los usuarios conozcan la localización de los mismos. LFC proporciona un catálogo que permite gestionar las replicas de los ficheros así como la traducción entre nombres lógicos (Logical File Names (LFN)), independientes de la localización y los nombres físicos (Storage Uniform Resource Locator (SURL)), que identifican el SE donde se almacena. Este es un servicio crítico por su centralización y a veces supone un importante cuello de botella.
- Information Service (IS). Este servicio se encarga de proporcionar información sobre los recursos gLite y su estado. Dicha información es esencial para el buen funcionamiento de todo el Grid. Mediante esta información, los recursos de computación (CE) pueden ser encontrados y mediante la publicación de información sobre el estado del Grid es posible detectar posibles situaciones de fallo o analizar el uso de la infraestructura. En la actualidad, la información publicada en el IS sigue el esquema *GLUE* (Grid Laboratory for a Uniform Environment) [26]. Esta iniciativa trata de definir un esquema conceptual de la información para ser usada en la monitorización y descubrimiento de recursos. En LCG-2 se ha adoptado el servicio MDS como la base principal del servicio de información. Sin embargo, recientemente se está desplegando un nuevo tipo de servicio basado en una arquitectura relacional, denominado R-GMA (Relational Grid Monitoring Architecture). Adicionalmente, existe un componente llamado BDII (Berkeley Database Information Index) [27] que permite agregar la información de todos los recursos computacionales de una organización.

La Figura 16 recoge el esquema de interacción con los componentes de gLite para la ejecución de tareas. Todas las operaciones llevadas a cabo por el usuario se realizan desde el UI. En primer lugar, el usuario copia y registra en el Grid los ficheros necesarios para la ejecución de la aplicación en uno o varios SE, dado que es posible disponer de diferentes replicas para un mismo fichero. Además, la localización de las réplicas debe almacenarse en el catálogo LFC junto con un identificador para, posteriormente, referenciar el fichero y tener acceso al mismo. A continuación el usuario proporciona la descripción del trabajo empleando el lenguaje JDL (Job Description Language) que especifica los principales atributos necesarios para su ejecución. En esta descripción se define el fichero ejecutable, la salida estándar y la de error, los argumentos de línea de comandos para el ejecutable y la VO a la que pertenece el trabajo que se pretende ejecutar.

El trabajo se envía al RB que es el encargado de gestionar su ejecución. Para ello busca el CE más apropiado a partir de la información sobre el estado del Grid obtenida por el IS y delega en el CE la ejecución del trabajo que, finalmente, se llevará a cabo en un WN. El RB monitoriza los cambios de estado del trabajo, que se registran en el LB, información que puede ser accedida por el usuario desde el UI.

Una vez que el trabajo ha finalizado, el usuario puede acceder a los resultados generados accediendo bien al OutputSandbox, que puede ser transferido a la máquina del usuario, o a los ficheros que hayan sido almacenados en algún SE.

2.5.5 Proyectos de Computación en Grid

En la actualidad existen multitud de proyectos relacionados con las tecnologías Grid que proporcionan las infraestructuras necesarias para desplegar y ejecutar experimentos científicos a gran escala. No obstante, este punto se ocupará únicamente de describir los proyectos (y sus correspondientes infraestructuras) empleados en esta tesis: EGEE, EELA y la NGI Española.

2.5.5.1 EGEE

El proyecto EGEE (Enabling Grid for E-sciencE) [28], actualmente finalizando su tercera fase, es la confluencia de la experiencia de más de 150 centros de más de 27 países con el propósito común de contribuir a los avances recientes de la tecnología Grid y desarrollar un servicio de infraestructura Grid en producción que esté disponible para los científicos las 24 horas del día.

La finalidad principal del proyecto es proveer a los investigadores de las universidades y de la industria del acceso a más recursos computacionales y almacenamiento de datos, independientemente de su localización geográfica. El proyecto, además, tiene como fin ampliar el número de usuarios nuevos del Grid.

En términos generales, el proyecto se enfoca en las siguientes áreas:

- La primera área es la construir una infraestructura Grid consistente, robusta y segura, que pueda atraer recursos computacionales adicionales.
- La segunda área es la de comprobar, mejorar y mantener el middleware (gLite) con la finalidad de poder dar servicio eficiente y confiable a los usuarios.
- La tercera área es la promover la búsqueda de nuevos usuarios y la consolidación de los actuales, tanto de la industria como del mundo científico, y

asegurar que reciben el proceso de aprendizaje y posterior soporte que puedan necesitar.

El Grid EGEE aprovecha la red europea de búsqueda GEANT e integra los conocimientos generados por decenas de proyectos Grid de la Unión Europea, nacionales e internacionales desarrollados hasta la fecha actual.

Los centros proveedores de recursos de EGEE se dividen en 12 federaciones, comprendiendo un total de más de 240 centros, proporcionado más de 41000 CPUs.

Dos dominios de aplicación piloto fueron seleccionados para guiar la implementación y certificar las prestaciones y funcionalidades de la infraestructura resultante. Uno es el Large Hadron Collider Computing Grid que da soporte a los experimentos de la Física y el otro es el Biomedical Grids, donde diversas comunidades están afrontando grandes retos con el fin de explotar la gran cantidad de datos bioinformáticos disponibles. En la actualidad, EGEE integra más de 5000 usuarios organizados en 100 comunidades de 45 paises.

Con una financiación europea superior a 30 millones de euros anuales a través de la Comisión Europea, EGEE da por finalizado, este año, su tercer y último periodo. En su lugar, se pondrá en marcha el proyecto EGI (European Grid Initiative), que a diferencia de EGEE no estará financiado por la Comisión Europea y que estará compuesto por las Iniciativas Nacionales de Grid de los diferentes países. Se espera que esta estructura federal convierta a EGI en la principal infraestructura de e-Ciencia europea.

2.5.5.2 EELA

EELA (E-science grid facility for Europe and Latin America) [29], actualmente en su segunda fase, aspira a construir una infraestructura Grid escalable de alta capacidad y calidad de producción, suministrando las 24 horas, acceso global a recursos de cómputo y almacenamiento, requeridas por un gran número de aplicaciones provenientes de la colaboración científica entre Europa y América Latina. El Proyecto EELA tiene principalmente tres objetivos:

- Establecer una red de colaboración entre instituciones Europeas que ya tienen experiencia en Grid (como el Proyecto EGEE), e instituciones Latino Americanas donde las actividades Grid están emergiendo.
- Establecer una e-Infraestructura en América Latina, interoperable con la de EGEE en Europa, lo que permitirá la diseminación del conocimiento a través de la experiencia en la tecnología Grid y permitiendo probar y ejecutar grandes aplicaciones.
- Establecer un marco seguro para la colaboración en e-Ciencia entre Europa y América Latina.

Para ello, la primera fase del proyecto se centró en establecer una plataforma Grid sostenible, proporcionando a sus usuarios una infraestructura basada en 16 centros, con un total de más de 730 núcleos de CPU y 60 Terabytes de espacio de almacenamiento. Esta fase permitió comprobar no sólo que la e-Infraestructura europea-latinoamericana era viable, sino que además responde a la necesidad real de una significativa parte de la comunidad científica.

Respecto a la fase actual (EELA-2), esta tiene dos objetivos:

- Consolidar y expandir la actual e-Infraestructura EELA para convertirse en una e-Infraestructura que proporcione un conjunto completo de servicios mejorados para todo tipo de aplicaciones de las múltiples áreas científicas de las comunidades científicas de Europa y América Latina.
- Establecer las condiciones que permitan la durabilidad de la e-Infraestructura, aún después del fin de proyecto.

2.5.5.3 NGI española

La NGI ('National Grid Inititative' o Iniciativa Grid Nacional) española integra actualmente cerca de 18 centros proveedores de recursos y 2400 cores, dando soporte a más de 300 usuarios directos de las comunidades de la física de altas energías, biomedicina, fusión, química computacional, astrofísica y geología, entre las comunidades más relevantes. Además, la NGI española mantiene un fuerte vínculo con la NGI portuguesa, bajo el acuerdo IBERGrid.

2.6 Supercomputación: El Supercomputador Tirant

Tirant, el supercomputador utilizado en uno de los trabajos de esta tesis, es uno de los 7 nodos de la Red Española de Supercomputación.

Actualmente, Tirant está compuesto de 256 nodos de cómputo JS20 (procedentes de la primera versión del supercomputador Marenostrum del Barcelona Supercomputing Center) y 5 servidores p515. A su vez, cada nodo de cómputo, posee dos procesadores IBM Power4 con una frecuencia de reloj de 2.0 GHz, 4 GB de memoria RAM y 36 GB de espacio de disco duro. Entre todos los nodos, proporcionan cerca de 10 TB de espacio de disco duro, accesible mediante el sistema de ficheros paralelo GPFS (Global Parallel File System). Los nodos de Tirant están interconectados mediante una red Myrinet (utilizada por las aplicaciones con comunicaciones paralelas) y una red Gigabit.

3. METODOLOGÍA PARA EL LANZAMIENTO DE EXPERIMENTOS MASIVOS EN e-INFRAESTRUCTURAS GRID EN PRODUCCIÓN Y SUPERCOMPUTACIÓN

La finalidad del presente capítulo es ofrecer una primera aproximación al despliegue y ejecución de experimentos masivos, tanto en e-Infraestructuras Grid en producción como en entornos de Supercomputación. En posteriores capítulos, esta metodología básica será analizada y progresivamente refinada.

3.1 Metodología en e-Infraestructuras Grid en producción

Por lo que respecta al despliegue de experimentos masivos en e-Infraestructuras Grid en producción, como EGEE, EELA o la NGI española, es aconsejable seguir una serie de pasos básicos que, dependiendo del tipo de experimento tendrán algunas variaciones. Evidentemente no se realizarán las mismas acciones en un experimento secuencial que en uno paralelo, o en un experimento con el software necesario previamente instalado y en un sistema auto-contenido. También es necesario tener en cuenta si los ficheros requeridos para la ejecución de los trabajos se encuentran almacenados en los recursos Grid o si directamente se indican en la especificación del trabajo.

En la Figura 17 puede apreciarse un sencillo esquema con las principales tareas que componen el despliegue de un experimento masivo en una e-Infraestructura grid en producción. El orden indicado es una posibilidad, dado que hay tareas que pueden realizarse en un orden distinto, como la especificación de trabajos y la selección de elementos de cómputo. Sin embargo, existen otras tareas que necesariamente deben realizarse secuencialmente como, por ejemplo, la selección de recursos de cómputo que siempre será anterior a la selección de recursos de almacenamiento.

A modo de resumen, a continuación se enumeran los pasos que habría que seguir para desplegar un experimento, aunque posteriormente se profundizará en cada uno de ellos.

En primer lugar, es necesario observar cuáles son los CEs que hay disponibles para el experimento. A partir del listado de CEs elegidos, será necesario decidir también cuáles son los SEs escogidos para albergar las distintas réplicas de los ficheros de entrada (bases de datos, ficheros de secuencias, etc.).

Con los recursos escogidos, darán comienzo las primeras pruebas con el programa básico que gestionará las operaciones realizadas por un trabajo, desde la recepción de los ficheros de entrada hasta el registro de los ficheros de salida.

Una vez se dispone de un conjunto de recursos destinatarios definitivos y un programa básico para resolver el problema, el último paso consiste en automatizar el sistema.

3.1.1 Selección de CEs

A la hora de elegir los recursos de cómputo que ejecutarán los trabajos, el primer paso es obtener la lista de los disponibles en la organización virtual a la que pertenece el certificado con el que el usuario se identifica. En el caso del middleware gLite, se

utilizaría el comando *glite-job-list-match* para determinar los recursos de cómputo disponibles. Por desgracia, no todos estos recursos estarán capacitados para ejecutar los trabajos del experimento masivo.

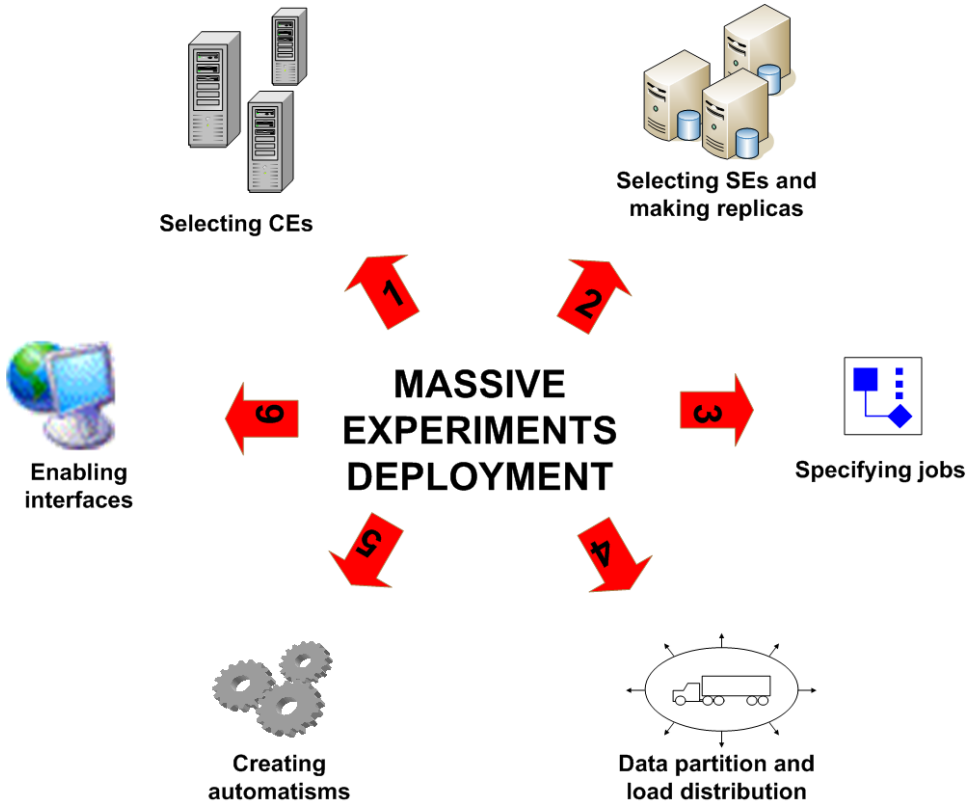

Figura 17: Esquema con las tareas que conforman el despliegue de experimentos masivos.

Uno de los primeros problemas que se suele encontrar está relacionado con la compilación de los programas a ejecutar. La experiencia aconseja utilizar pre-compilaciones estáticas, aunque esta medida no será siempre posible, además de aumentar de forma considerable el tamaño de los ficheros binarios. En otros casos, debido a la simplicidad de los ficheros fuente utilizados y a la ausencia de dependencias con librerías externas, será más conveniente intentar una compilación en el propio recurso de cómputo. Otras restricciones están relacionadas con el conjunto de programas instalados en el recurso. En caso de no poder realizar la instalación dentro del propio trabajo Grid, será obligatorio contactar a los administradores de los diferentes recursos para que procedan a instalar el software a utilizar, lo que limitará claramente el número de recursos y las posibilidades a la hora de lanzar los trabajos Grid libremente.

Por tanto, parece obvio, que para definir un grupo de recursos de cómputo hay que recurrir a las pruebas empíricas, con el fin de salir de dudas con respecto a los problemas de configuración que tienen los diferentes recursos. Estas pruebas, al mismo tiempo, pueden aportar más información sobre el recurso, como la velocidad de cómputo, la capacidad de la memoria primaria y secundaria o la velocidad de I/O. Esta información, en caso de abundancia de recursos, constituirá criterios objetivos para elegir un recurso u otro. Otro tipo de información interesante será la configuración del recurso de cómputo con su entorno, como por ejemplo, su elemento de almacenamiento más próximo o los sistemas de información y catálogo.

3.1.2 Selección de SEs y réplicas de ficheros

Los trabajos Grid, como cualquier tipo de trabajo, necesitan una serie de datos de entrada, que pueden dividirse en parámetros y ficheros de datos. En algunos de los middlewares Grid más utilizados (como Globus Toolkit o gLite) no suele haber muchos problemas con el paso de parámetros, aunque el caso de los ficheros de entrada es diferente. Existe la posibilidad de adjuntar ficheros de datos pequeños en la bandeja de entrada de los trabajos, aunque normalmente los ficheros necesarios serán excesivamente grandes. De hecho, *gLite* impone una restricción al tamaño total máximo de los ficheros de entrada incluidos en la bandeja de entrada (10 MB), la denominada *input sandbox*.

El uso de la bandeja de entrada de una forma desmesurada es evidente que contribuye a saturar el ancho de banda del gestor de trabajos. Esto se debe al hecho de que los datos incluidos en las bandejas de entrada y salida se transfieren y copian temporalmente al Gestor de Recursos (Workload Management System), y de aquí a los CEs. Además de ineficiente, es una opción poco aconsejable porque puede saturar el espacio en disco de los gestores de tareas. Este tipo de situaciones son evitables mediante el uso de elementos de almacenamiento, disponibles en las infraestructuras utilizadas. El uso de los SEs, en primer lugar, logra superar los problemas de ancho de banda de la interfaz de usuario trasladándolos al propio SE (más preparado para este tipo de requerimientos) y además, permite aprovechar el hecho de que se trate de un sistema de almacenamiento distribuido con todas las facilidades correspondientes, como la creación de réplicas de ficheros.

La selección de SEs y la réplica de ciertos ficheros fundamentales para la ejecución de un trabajo Grid supondrá otra de las tareas prioritarias en el despliegue de un experimento masivo Grid. Una de los aspectos claves será buscar un acoplamiento entre los CEs escogidos y los SEs donde estarán las distintas réplicas de los ficheros de entrada utilizados en las ejecuciones de los trabajos.

Para comenzar, existen dos estrategias básicas respecto a la localización de los almacenes contenedores de ficheros de entrada: centralización (todos los CEs acceden al mismo SE) y descentralización (cada CE accede a su propio SE). La primera de las opciones convierte al SE escogido en el cuello de botella de todas las operaciones de entrada y salida, mientras que la segunda opción, pese a que lograría obtener las mejores prestaciones posibles, tiene un impacto negativo en el global de la infraestructura Grid utilizada, debido al espacio de memoria secundaria utilizado.

Teniendo en cuenta ambas tendencias, con sus ventajas e inconvenientes, se puede optar por una solución de compromiso que se sitúe entre ambas estrategias. La solución utilizada requerirá la realización de un estudio para encontrar los mejores candidatos a albergar réplicas de los distintos ficheros. Este estudio consistirá en elaborar un listado con todos los SEs más próximos asociados a los CEs escogidos en el punto anterior, y filtrar aquellos que no tengan las características mínimas aconsejables (por ejemplo, un ancho de banda mínimo y una cantidad de memoria física disponible mínima).

Después de este filtrado, habrá que decidir el número de réplicas (y por tanto de SEs) que se quieren realizar, dependiendo del número de elementos de cómputo utilizados en el experimento y el máximo espacio de memoria que se pretende ocupar en toda la infraestructura Grid. Una vez decidido el número, se buscarán los elementos de almacenamiento en base a un criterio de proximidad administrativa y geográfica de los CEs.

Por lo que respecta a los ficheros de salida, el tipo de problemática es muy diferente, ya que no tiene sentido tener diversas réplicas de estos ficheros, en todo caso, se podría

hacer una copia de los ficheros resultado al SE más cercano del recurso de cómputo y una réplica al SE más cercano a la interfaz de usuario.

3.1.3 Especificación del trabajo

Una vez escogidas las máquinas donde van a lanzarse los trabajos y las máquinas que almacenarán los datos necesarios para su ejecución, el siguiente paso será especificar correctamente los trabajos.

En este sentido, habrá que elaborar la especificación en el lenguaje de descripción de los trabajos en *gLite*, es decir, en JDL (Job Description Language). Un ejemplo de fichero JDL se puede observar en la Figura 18.

```
Type = "Job";
VirtualOrganisation = "biomed";
Executable = "shell-BiG-seq.sh";
Arguments = "ramses.dsic.upv.es 0 11thExp seqfile290.sqf";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"/home/acarrion/blast/shell-BiG-seq.sh"};
OutputSandbox = {"std.out","std.err"};
```

Figura 18: Ejemplo de JDL.

Como puede verse en la Figura 18, en la especificación del trabajo se indica: la organización virtual donde se lanzará el trabajo (*VirtualOrganisation*), cuál es el fichero principal que iniciará la ejecución del trabajo (*Executable*), cuáles son los argumentos que se pasarán al ejecutable anterior (*Arguments*), en qué ficheros se depositará las salidas estándar y las salidas de error (*StdOutput* y *StdError*), cuáles son los ficheros que se depositan en la bandeja de entrada (*InputSandbox*), y finalmente cuáles son los ficheros que habrá que retornar al usuario en la bandeja de salida (*OutputSandbox*).

Por tanto, parece obvio que el peso fundamental de la ejecución recaerá en el ejecutable especificado en este fichero, tratándose normalmente de un shell-script (habitualmente interpretable por *sh* o *bash*). Se utiliza un shell-script y no directamente un ejecutable dado que el ejecutable conlleva una serie de problemáticas, tales como tener que resolver el impacto de los entornos heterogéneos en la compilación de los ficheros fuente o tener que fusionar distintos procesos independientes dentro de un mismo ejecutable. Como además, se recomienda a los administradores instalar sistemas operativos tipo Unix en los nodos de proceso, el uso de shell-scripts asegura que los programas desarrollados podrán ser ejecutados sin problemas en cualquier centro.

Por tanto, el proceso que ejecute dicho shell-script será el encargado de realizar todo el trabajo, desde invocar a otros ejecutables hasta transferir los ficheros desde o hacia un SE. Por esta razón, el citado shell-script cobra una importancia crucial en la ejecución de un trabajo y será conveniente analizar su contenido seguidamente.

El programa comienza con la recogida de los parámetros de entrada, que en este caso son: el nombre del recurso de cómputo, el identificador del trabajo *gLite*, el identificador del experimento y el nombre del fichero de secuencias de entrada. Estos parámetros facilitan el conocimiento, por parte del propio trabajo, de qué papel juega dentro del experimento global realizado, estando perfectamente identificado. Una vez se han procesado los parámetros de entrada, es el momento de preparar el entorno para la ejecución, es decir, la construcción de la estructura de directorios, los cambios

pertinentes en los ficheros pasados vía la bandeja de entrada (cambios de directorio, cambios de permisos, etc.) y el establecimiento del directorio actual.

Con los pasos anteriores, se dan por finalizadas las acciones referentes a los parámetros, ficheros de entrada y preparación del entorno. Llegados a este punto, es el momento de comenzar con la ejecución de las acciones específicas del problema concreto. Como es habitual en cualquier trabajo, primeramente se hará una búsqueda de los ficheros necesarios para su ejecución (obviamente, ficheros que no se han pasado por la bandeja de entrada). Por ello, será habitual el uso en este primer momento de comandos de descarga de ficheros, como *wget* para ficheros en la infraestructura general de internet o *lcg-cp* cuando se trata de ficheros almacenados en elementos de almacenamiento de la infraestructura Grid. Una vez descargados, algunos ficheros requerirán ser descomprimidos, instalados, formateados (el fichero de base de datos), etc.

Una vez están disponibles todos los ficheros necesarios para la ejecución, es momento de proceder con la ejecución propiamente dicha. El programa central normalmente habrá sido instalado en la fase anterior (en el caso de ejecuciones BLAST secuenciales, será el ejecutable *blastall* de NCBI) y necesitará de una serie de parámetros, algunos comunes a todos los trabajos del experimento y otros indicados en el propio fichero JDL como argumentos. El objetivo de esta ejecución será la obtención de una serie de resultados que habrá que empaquetar en la última fase, eligiendo la mejor manera de ofrecer estos datos al usuario.

La fase final consistirá precisamente en el tratamiento de los resultados, creados a partir de la ejecución. Los resultados estarán disponibles en dos formatos principales: ficheros de salida y salidas estándar (salida y error). Las salidas estándar estarán disponibles (una vez finalice el trabajo) en la bandeja de salida, mientras que los ficheros de salida normalmente recibirán algún tipo de tratamiento previo, como el filtrado de los datos o la compresión mediante *gzip*. Al igual que sucedía con los ficheros que se enviaban conjuntamente con el trabajo en la bandeja de entrada, en la fase de salida sucede exactamente igual. Dependiendo del tamaño de los ficheros de salida se podrá optar por obtener los ficheros por medio de la bandeja de salida (en el caso de ficheros pequeños) o buscando alternativas como los elementos de almacenamiento (en el caso de ficheros grandes). Después de resolver el problema de los ficheros de salida, es momento de proceder a abandonar el elemento de cómputo, sin olvidarse de borrar los ficheros y directorios previamente creados. De no hacerlo así, no sólo se estarán dejando ficheros innecesarios en el sistema sino que se contribuirá a la desestabilización de los sistemas de cómputo utilizados, y el sistema en un sentido global. Este tipo de buenas prácticas ayudan a administrar los sistemas y a mantener unas buenas prestaciones en el futuro próximo.

3.1.4 Automatización del sistema

Uno de los puntos más importantes de cualquier despliegue Grid es la construcción de un sistema automatizado que lleve un control y monitorización de la evolución de los experimentos. El sistema tendrá que encargarse principalmente del lanzamiento de los distintos trabajos *gLite* en los que se haya dividido el experimento y su monitorización, relanzamiento (en caso de que sea necesario) y obtención de resultados.

Este sistema ha de ser eficaz, eficiente y robusto. Como podrá comprobarse en posteriores capítulos, muchas veces es complicado intentar mejorar al mismo tiempo estas tres características fundamentales. La eficacia es la garantía de que más tarde o

más pronto el sistema llegará a un estado en el que el trabajo global habrá finalizado con éxito. Por otro lado, la eficacia garantizará que la distribución de la carga del trabajo es óptima y se alcanza a un estado final lo más pronto posible, adecuándose el sistema periódicamente a posibles cambios de la estructura global de la infraestructura Grid. Finalmente, el sistema debe ser robusto de manera que sea capaz de retomar una ejecución interrumpida en cualquier momento.

Obviamente, son objetivos muy ambiciosos que necesitan la aplicación de acciones concretas para poder lograrlos y en muchas ocasiones será necesario optar por un compromiso cuando el efecto de las acciones mejore un objetivo pero, a cambio, perjudique otro.

Uno de los puntos donde se pone de manifiesto la incapacidad para aumentar a la vez la eficacia y la eficiencia es en el establecimiento de un tiempo máximo de vida de un trabajo, a partir del cual un trabajo inacabado debe ser cancelado y relanzado. El objetivo de esta medida es evitar que existan trabajos que permanezcan en ejecución debido a errores o simplemente por haberse quedado bloqueados. La cancelación de los trabajos es obvio que es imprescindible para garantizar la eficacia del sistema, ya que son necesarios mecanismos para desbloquear las ejecuciones, pero también constituyen un riesgo dado que es complicado (o incluso imposible) distinguir entre un trabajo bloqueado y un trabajo que evoluciona con lentitud (todos los recursos no tienen las mismas prestaciones y puesto que las aplicaciones son de tipo ‘legacy’ es imposible añadir mecanismos para informar de su progreso).

La robustez se logrará creando un sistema de información actualizado con una frecuencia configurable. Este sistema de información contendrá los datos necesarios para restaurar la ejecución a pesar de los posibles errores ocasionales. Por tanto, habrá un registro de todos los trabajos que componen el trabajo global, donde estará disponible el identificador del trabajo, información sobre el estado puntual de dicho trabajo y un contador con el número de veces que un determinado trabajo ha sido relanzado. Por otro lado, también habrá una estructura con ficheros que contendrá los identificadores de los trabajos *gLite* que constituyen el trabajo global. Con ambas estructuras se mantiene un enlace con los trabajos *gLite*, que son los que en realidad ejecutan el trabajo.

3.1.5 *Particionamiento de los datos y distribución de la carga*

Uno de los pasos más importantes, previos al sometimiento de los diferentes trabajos es el particionamiento de los datos y la distribución de la carga.

Como se comentaba en el apartado referente a la automatización del sistema, los trabajos han de ser caracterizados de acuerdo a un tiempo máximo o *wall-time*, a partir del cual el automatismo descartará cualquier trabajo en ejecución, considerando que tal trabajo se ha quedado bloqueado y no será productivo. Este parámetro temporal es muy crítico, dado que un valor muy pequeño descartará muchos trabajos lentos y un valor elevado hará que el sistema espere de forma innecesaria y excesiva para descartar los trabajos bloqueados.

No obstante, elegir un buen valor para el *wall-time* es un problema complejo. En una primera versión del automatismo se han elegido valores relacionados con la duración media de un trabajo, por ejemplo, el doble de la duración media. Aunque parece una

estrategia razonable, cuando se trabaja con ficheros de secuencias altamente heterogéneos (respecto al número de secuencias) no es posible garantizar que el valor de *wall-time* elegido vaya a ser superior al trabajo más largo. En el peor de los casos, el experimento nunca finalizará dado que ninguno de los recursos disponibles es capaz de ejecutar el trabajo más largo dentro del tiempo especificado por el *wall-time*.

Por tanto, se hace imprescindible homogeneizar el coste computacional de los trabajos correspondientes a cada bloque, mediante un particionamiento adecuado de los datos a procesar: dividir el fichero de secuencias de entrada equitativamente en múltiples bloques.

Por otro lado, la distribución de la carga es otro aspecto que es necesario analizar con el fin de incrementar las prestaciones de un experimento. No todos los CEs poseen las mismas prestaciones, y la distribución de la carga debe ser realizada de acuerdo con estas capacidades. El factor clave es el número de trabajos concurrentes que un CE puede ejecutar a la vez. Este valor estará estrictamente relacionado con el tiempo de cola y la velocidad del procesador (los bytes de una secuencia de entrada que pueden ser procesados por unidad de tiempo), y los trabajos deben ser asignados de forma asimétrica de acuerdo con este número de trabajos máximos concurrentes.

Sin embargo, como podrá comprobarse en el último capítulo de esta tesis, es posible desarrollar técnicas que permitan calcular, con una precisión aceptable, el tiempo estimado que un trabajo (con unas características determinadas) estará en ejecución en un recurso con unas prestaciones concretas.

3.2 Metodología en un supercomputador

Puesto que en el caso de un supercomputador se trata de una infraestructura paralela, no tiene sentido hablar del despliegue de experimentos, sino más bien de la preparación de los mismos. Por ello, a diferencia de una infraestructura Grid, el número y complejidad de las tareas necesarias para ejecutar un determinado experimento se reduce considerablemente. El objetivo de este punto será justamente describir, brevemente, las tareas que conforman la preparación de experimentos para ser lanzados en un Supercomputador, como Tirant.

3.2.1 Particionamiento de la carga

La necesidad de particionamiento de la carga en un supercomputador, como Tirant, se debe a dos motivos. El primero de ellos es que, en muchas ocasiones, el coste computacional de los experimentos (como el alineamiento de un gran número de secuencias genómicas contra las almacenadas en una base de datos) excede las capacidades de un supercomputador, aunque esté provisto de una gran capacidad de cómputo y almacenamiento. Por otra parte, debe tenerse en cuenta que los supercomputadores son recursos compartidos con muchos otros usuarios, por lo que normalmente no será posible disponer de toda la potencia computacional del mismo. Además, y con el fin de favorecer el progreso de todos sus usuarios, los supercomputadores imponen una restricción respecto al tiempo máximo (*wall-time*) que un determinado experimento puede permanecer ininterrumpidamente en ejecución. Por tanto, siempre y cuando sea posible, se aconseja dividir el fichero de entrada en diversos

ficheros con (aproximadamente) el mismo coste computacional. Al llevar a cabo este paso habrá que tener la precaución de que el tiempo de ejecución estimado para las tareas, no supere, en ningún caso, el *wall-time* máximo del supercomputador (en el caso de Tirant es 3 días).

3.2.3 Especificación de los trabajos

Una vez particionado el experimento, el siguiente paso es especificar las tareas que deberán ser ejecutadas en el supercomputador, haciendo uso del lenguaje disponible para ello. Por ejemplo, en el supercomputador Tirant, los trabajos se definen en un fichero de texto, con una serie de directivas y comandos que informan al sistema *batch* (Slurm+MOAB) de las características del mismo (véase Figura 19).

```
#!/ bin /bash
# @ job name = test_serial
# @ initialdir = .
# @ output = serial%j.out
# @ error = serial%j.err
# @ totaltasks = 1
# @ wallclocklimit = 00:02:00
. /serialbinary
```

Figura 19: Ejemplo de fichero para la especificación de un trabajo secuencial para la utilidad SLRUM+MOAB.

Los campos a indicar en dicho fichero son: el nombre del trabajo, el directorio que se tomará como base en la ejecución, los nombres de los ficheros de salida estándar y de error, el número total de tareas (lógicamente para el caso de trabajos secuenciales será uno mientras que para trabajos paralelos será mayor que uno), el tiempo de duración máximo estimado y el comando que inicia la ejecución. Por último, un aspecto muy importante a tener en cuenta es asegurarse de que el programa a ejecutar (en el ejemplo 'serialbinary') realice el *stagein* (copia de los ficheros de entrada desde el directorio 'home' del usuario hacia el nodo de cómputo) y el *stageout* (volcado de los ficheros resultado desde el nodo de proceso al directorio 'home' del usuario). Gracias a este tipo de medidas se evita saturar el sistema de ficheros paralelo (en el caso de Tirant es GPFS).

3.2.4 Lanzamiento de los trabajos

Una vez especificadas adecuadamente las tareas, el último paso consiste en lanzar los trabajos mediante el comando de sometimiento del supercomputador (*mnsubmit* en Tirant). Tras ser lanzados, estos serán encolados y será responsabilidad del planificador instalado decidir dónde y cuándo se ejecutan los trabajos. Para ello, el planificador tendrá en cuenta, entre otras cosas: las características del trabajo, la prioridad asignada al usuario por el comité de acceso al supercomputador, la cantidad de recursos consumidos hasta la fecha y la carga actual del sistema.

Al igual que se comentó en el apartado de las infraestructuras Grid, una buena práctica consiste en eliminar, de los nodos de cómputo, aquellos ficheros que se hayan generado durante la ejecución de los trabajos.

Por último, cabe mencionar que en el caso de la supercomputación, el usuario no tiene la posibilidad de desarrollar programas o automatismos que controlen la ejecución de los trabajos y que además, mejoren la eficiencia de los experimentos. Esto es debido a que el punto de acceso de todos los usuarios con el supercomputador suelen ser únicamente uno o dos nodos, llamados 'interactivos'. La finalidad de estos nodos es permitir a los usuarios que lleven a cabo sus tareas de pre-proceso (movimiento de ficheros, particionamiento de datos, generación de ficheros de especificación de los trabajos, etc.), siempre y cuando los procesos encargados de ello no requieran un uso del nodo interactivo superior a un pequeño periodo de tiempo (en Tirant este tiempo es de tan solo 10 minutos). Por tanto, en estos casos la única opción es monitorizar de forma manual y periódica el estado de los trabajos y relanzar inmediatamente aquellos que hayan finalizado incorrectamente.

4. COMPARATIVA ENTRE MPIBLAST EN UN SUPERCOMPUTADOR Y BLAST SECUENCIAL EN UNA INFRAESTRUCTURA GRID

En el anterior capítulo se expuso la metodología que debe seguirse a la hora de desplegar y ejecutar un experimento masivo en ambos tipos de infraestructuras: el Grid y la Supercomputación. Tomando como base la metodología anterior, el presente capítulo realiza una comparativa entre ambas infraestructuras, utilizando como caso de uso el alineamiento de un gran número de secuencias genómicas contra las almacenadas en una determinada base de datos.

Como ya se comentó en el capítulo de 'Estado del arte', el programa de alineamiento local de secuencias más popular en la comunidad bioinformática es, sin lugar a dudas, BLAST (Basic Local Alignment Search Tool) del National Centre for Biotechnology Information (NCBI). La gran aceptación de esta herramienta ha suscitado la aparición de múltiples versiones que, normalmente, pertenecen a uno de los siguientes tipos: de 'Altas Prestaciones' o de 'Alta Productividad'. Las versiones de 'Altas Prestaciones' se ejecutan eficientemente en clusters o supercomputadores mientras que las versiones de 'Alta Productividad' son apropiadas para ser ejecutadas en infraestructuras Grid.

Por tanto, la finalidad de este capítulo será, en primer lugar, describir las características de cada aproximación. A continuación, se comentarán los experimentos realizados con el fin de analizar las ventajas e inconvenientes de ambos enfoques. Finalmente se presentarán los resultados obtenidos en los experimentos anteriores.

4.1 Aproximaciones paralelas

Como se ha comentado anteriormente, el alineamiento de grandes muestras genómicas (como los metagenomas) requiere la ejecución de programas computacionalmente intensivos que exceden los recursos disponibles en la mayoría de centros de investigación. No obstante, mediante estrategias de paralelización, es posible dividir el problema en múltiples tareas individuales, logrando, de esta manera, reducir considerablemente el tiempo de ejecución requerido por el experimento. En el presente punto se describen las principales aproximaciones paralelas que suelen utilizarse cuando se persigue paralelizar el programa BLAST.

4.1.1 Aproximación paralela basada en el particionamiento del fichero de secuencias de entrada

La aproximación paralela basada en el particionamiento del fichero de secuencias de entrada (búsqueda) se ajusta al paradigma de 'Alta Productividad' de las infraestructuras Grid. De hecho, esta estrategia de paralelización puede considerarse una instancia de la metodología de despliegue y ejecución de experimentos en infraestructuras Grid, descrita extensamente en el capítulo anterior. A modo de recordatorio, dicha metodología implicaba la realización de las siguientes tareas: particionar los datos de entrada, seleccionar los recursos de cómputo y almacenamiento

apropiados, replicar adecuadamente los ficheros requeridos por los trabajos, decidir la distribución de la carga inicial y ejecutar el programa o automatismo encargado del lanzamiento, monitorización y relanzamiento de los trabajos fallidos o bloqueados.

El único aspecto a tener en cuenta en este método de paralelización es la forma en la que se realiza la primera tarea de la metodología anterior, es decir, el particionamiento de datos. Como su nombre sugiere, esta aproximación se basa en la fragmentación del fichero con las secuencias de entrada a analizar. Para ello, existen dos posibles estrategias de partición: dividir el fichero de entrada por número de secuencias (los fragmentos resultantes tendrán el mismo número de secuencias) o por tamaño (todos los ficheros producidos por la fragmentación tendrán el mismo tamaño). La elección de una estrategia u otra dependerá de las características del fichero de entrada utilizado, siendo recomendable optar por aquella que logre homogeneizar mejor el coste computacional de las tareas. Esta medida permitirá, al automatismo responsable de la ejecución, diferenciar con mayor precisión entre trabajos lentos y bloqueados.

Cada uno de los fragmentos generados en la fase de particionamiento dará lugar a un trabajo BLAST secuencial, que deberá ser ejecutado, independientemente, en uno de los elementos de cómputo de la infraestructura utilizada. La ausencia de comunicación entre los trabajos que conforman el experimento es lo que convierte a esta aproximación en adecuada para ser ejecutada en infraestructuras de 'Alta Productividad' como el Grid.

4.1.2 Aproximación paralela basada en el particionamiento de la base de datos

El segundo método de paralelización es el utilizado por determinadas versiones paralelas de BLAST, como *mpiBLAST*. Debido a su naturaleza paralela, este tipo de programas se ejecutan eficientemente en clusters o supercomputadores. La idea básica de esta aproximación consiste en fragmentar la base de datos en diversos trozos que serán distribuidos a un nodo de cómputo distinto, paralelizando así, la búsqueda sobre la base de datos. Cuando todos los procesos han terminado de procesar la misma secuencia del fichero de entrada (todos disponen del fichero de secuencias de entrada completo) proceden a comunicarse, para combinar los hallazgos encontrados por cada uno. Puesto que cada herramienta implementa de una forma distinta el proceso anteriormente explicado, este punto se centrará en describir las acciones llevadas a cabo por mpiBLAST-PIO (de entrada/salida paralela), la versión paralela de BLAST utilizada en los experimentos de este capítulo.

Al igual que la versión de BLAST secuencial, el procedimiento de mpiBLAST-PIO comienza dando formato a la base de datos, con la particularidad de que, en este caso, es necesario indicar el número de fragmentos en los que se dividirá dicha base de datos. Tras finalizar el formateo de la base de datos, los fragmentos resultantes (y en el formato adecuado) serán depositados por el programa en un directorio compartido. Tomando estos fragmentos, el siguiente paso consistirá en ejecutar el programa, indicando entra otras cosas, el número de procesadores del supercomputador a utilizar. Si lo que se pretende es obtener, al menos, una aceleración lineal se recomienda emplear aproximadamente tantos procesadores como fragmentos se hayan generado en el anterior paso. Por ejemplo, si la base de datos fuese dividida en 75 trozos, la ejecución debería realizarse en 76 procesadores (el procesador extra se utiliza como coordinador del resto) distintos del supercomputador.

La implementación de mpiBLAST-PIO se basa en el uso de un esquema maestro-esclavo. En una búsqueda BLAST, el 'expected value' está muy relacionado con el espacio de búsqueda empleado, es decir, con la dimensión del fichero de secuencias de entrada y la base de datos. Por tanto, el alineamiento de una secuencia de búsqueda contra un fragmento de la base de datos tendrá un espacio de búsqueda más reducido que si se realizase contra la base de datos completa, obteniéndose diferentes valores de similitud en ambos escenarios. La forma en mpiBLAST-PIO resuelve esta cuestión consiste en hacer que el proceso maestro ejecute una pseudo-ejecución de BLAST contra todos los fragmentos de la base de datos. Gracias a esta medida, el maestro recibirá de los esclavos cierta información, para posteriormente, re-calcular los valores del 'e-value' como si la búsqueda se hubiese realizado contra la base de datos completa.

A partir de este momento, comienza el verdadero proceso de búsqueda. En primer lugar, el maestro difunde el fichero de secuencias de búsqueda completo a todos los esclavos, junto con información estadística necesaria para producir resultados del valor de similitud (el 'e-value') globales (como si se tratase de una búsqueda contra la base de datos completa). A continuación, el maestro procederá a asignar a cada proceso, un fragmento diferente de la base de datos (siempre y cuando se utilice un número de procesadores igual al número de fragmentos más uno). En caso de que el proceso esclavo tenga a su disposición un espacio de almacenamiento local, este transferirá el fragmento de base de datos desde el directorio compartido (ubicación donde el proceso de formateo depositó los fragmentos) con el fin de agilizar accesos posteriores. Una vez el esclavo ha recibido el fichero de secuencias de entrada y el fragmento de base de datos, está en condiciones para comenzar a ejecutar los alineamientos sobre su porción de base de datos. Según vaya produciendo resultados (una lista de registros de salida ordenados por el grado de similitud) los irá almacenando en memoria. Posteriormente, estos resultados locales serán combinados en un único resultado global y volcado en el sistema de ficheros. Además, si el supercomputador utilizado posee un sistema de ficheros paralelo (como GPFS o Global Parallel File System), la ejecución puede beneficiarse de la entrada/salida paralela implementada en la versión '-PIO' de mpiBLAST. Las primeras versiones de mpiBLAST utilizaban la estrategia de "escritura del maestro" en la que el maestro era el encargado no solo los esclavos, sino la entrada/salida también. De esta forma, no se observaba escalabilidad en las prestaciones conforme se aumentaba el número de nodos, ya que el maestro se convertía en el cuello de botella. Por este motivo, en la nueva versión, mpiBLAST-PIO, se implementó la estrategia de "escritura paralela" o "escritura de los esclavos" en la que los esclavos sólo envían metadatos (puntuaciones y tamaños de los registro de salida), permitiendo de esta manera, que el maestro calcule en qué parte del fichero de salida deben escribirse los resultados. Finalmente el maestro le enviará a los esclavos las posiciones calculadas, permitiendo que cada uno de ellos escriba, independientemente, los resultados que ha obtenido en el fichero de resultados global (aprovechando la existencia del sistema de ficheros paralelo).

La ventaja de utilizar mpiBLAST frente a la versión secuencial de BLAST es la siguiente: debido al crecimiento exponencial del tamaño de las bases de datos bioinformáticas, estas bases de datos son, actualmente, de mayor envergadura que la memoria principal de la mayoría de computadores, forzando a BLAST a recurrir a la paginación de la memoria. Sin embargo, gracias a la segmentación de la base de datos, implementada por mpiBLAST, es posible aprovecharse de la localidad de referencia, reduciendo de este modo (o incluso eliminando) el acceso externo a disco, obteniendo una mejora sustancial en las prestaciones.

El único inconveniente de mpiBLAST es la falta de un mecanismo de reinicio, que permitiera continuar una ejecución previa fallida. Puesto que se trata de un programa basado en MPI, un trabajo formado por un gran número de procesos y ejecutándose durante varios días, no es de extrañar que termine fallando. La única opción es estos casos es relanzar lo antes posible el trabajo.

4.2 Experimentos

El cometido de los experimentos llevados a cabo en este capítulo es realizar una comparativa entre las prestaciones obtenidas por la versión secuencial de BLAST en una infraestructura Grid y mpiBLAST en un supercomputador. En concreto, las infraestructuras utilizadas han sido: EGEE (Grid) y Tirant (supercomputador).

Por lo que respecta a la base de datos, se ha utilizado en ambos casos la base de datos nr (no redundante) GenBank, únicamente con las secuencias de ADN de organismos procariotas. La versión utilizada contiene 7.166.228 secuencias y su volumen aproximado es de 1 Gigabyte.

El fichero de secuencias de búsqueda fue presentado en el apartado de 'Estado del arte' y contiene el metagenoma de las especies del mar de los Sargazos. La versión empleada posee 811.372 secuencias con un volumen total de 1 Gigabyte.

Para cuantificar las prestaciones obtenidas en los experimentos se han utilizado cuatro parámetros: productividad (secuencias procesadas/hora), evolución de la finalización del experimento, tiempo de respuesta y tiempo total de CPU consumido.

Antes de comentar los resultados, cabe mencionar que los tiempos de respuesta han sido normalizados de acuerdo con los 'benchmark' oficiales proporcionados por los proveedores (SpecInt2K y SpecFloat2K). Las prestaciones de cada recurso, incluidas las del Grid, han sido multiplicadas por un factor que se obtiene de dividir el factor Spec2K promedio por 1500. Esta normalización permite hacer mucho más comparables los tiempos de respuesta entre las máquinas.

4.2.1 Resultados obtenidos en la infraestructura Grid

Los experimentos realizados en la infraestructura Grid EGEE revelan una cuestión interesante. El principal problema que afecta a las prestaciones es el alto ratio de trabajos fallidos, dado que más del 58% de los trabajos fueron relanzandos, duplicando el tiempo de respuesta necesario por el experimento. De hecho, como puede observarse en la Figura 20, la finalización del 80% de los trabajos requirió de tan sólo el 30% del tiempo total de respuesta. Por lo tanto, si se lograse reducir el número de trabajos relanzados, se conseguiría reducir considerablemente el tiempo de respuesta.

Los factores involucrados en la reducción del número de trabajos fallidos son tres: una buena selección de los elementos de almacenamiento (11% de los trabajos fallaron al acceder a los SE), una buena predicción del tiempo de respuesta necesario (12% de los trabajos parecieron estar bloqueados, aunque realmente estaban ejecutándose lentamente) y mecanismo de relanzamiento más efectivos (31% de los trabajos fueron relanzados más de una vez).

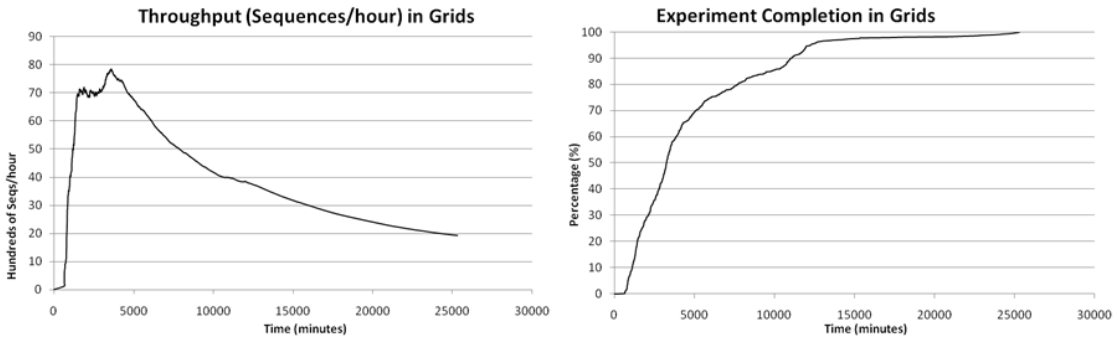

Figura 20. Prestaciones obtenidas en la infraestructura Grid

La Figura 20 muestra el número de secuencias por hora y la evolución de la finalización del experimento en el Grid. Como puede observarse, las prestaciones decaen tras haber finalizado el 75% de los trabajos. El fallo de los trabajos y los relanzamientos penalizan la productividad del experimento e incrementan considerablemente el tiempo de respuesta del mismo.

4.2.2 Resultados obtenidos en el supercomputador

Las pruebas llevadas a cabo en el supercomputador se realizaron utilizando dos conjuntos de particiones (10 y 25 bloques) con 25 y 50 procesadores, dando lugar a cuatro configuraciones posibles.

Los resultados obtenidos muestran que el incremento del número de procesadores y el número de particiones supone una mejoría de las prestaciones del experimento. Aunque teóricamente la aceleración debida al número de procesadores está limitada por las sincronizaciones de entrada/salida, los resultados muestran una tendencia incremental, que podría confirmarse utilizando otras configuraciones no probadas (con mayor número de procesadores). En cuanto a la mejoría observada como resultado del uso de un mayor número de bloques, esta podría deberse al hecho de que los fragmentos del fichero de secuencias de búsqueda cupiesen en la memoria del nodo, reduciendo el tiempo extra utilizado en accesos a memoria.

La Figura 21 muestra el número de secuencias por hora y la evolución de la finalización del experimento en el supercomputador. Una clara diferencia que se observa con respecto a los resultados obtenidos en el Grid es un comportamiento mucho más lineal. Dicho comportamiento lineal puede deberse a varias razones. La primera de ellas es el paralelismo de la ejecución: mientras en el Grid todos los trabajos fueron lanzados concurrentemente, en el supercomputador los trabajos se ejecutaron de una forma más secuencial. Además, otra diferencia fundamental es que los recursos en el Grid son heterogéneos mientras que todos los nodos de proceso de un supercomputador tienen idénticas capacidades. Finalmente, el bajo ratio de trabajos fallidos juega un importante papel en la linealidad apreciada.

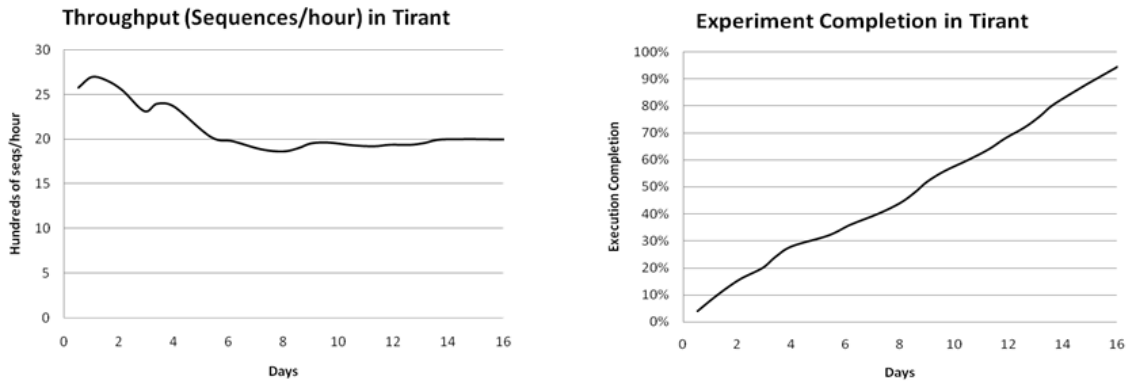

Figura 21. Prestaciones obtenidas en el supercomputador.

Del gráfico de tiempo total de CPU consumido, incluido en la Figura 22, se extraen dos conclusiones importantes. En general, la cantidad de tiempo necesario para procesar dos secuencias distintas no es el mismo debido a la diferencia de longitud entre ambas y la naturaleza heurística del algoritmo BLAST. Por esta razón, aunque el maestro distribuya el mismo número de secuencias de búsqueda entre los esclavos, algunos de ellos finalizarán antes que otros. Los esclavos que hayan finalizado estarán ociosos hasta que el maestro reciba los resultados del resto de esclavos. Por tanto, la distribución de la carga constituye un factor clave para mejorar las prestaciones. Este hecho explica por qué el tiempo total de CPU consumido por la configuración '10 bloques – 25 procesadores' es mayor que el tiempo consumido por la configuración '25 bloques – 25 procesadores'. Asimismo, el gráfico también muestra que fijando el número de particiones del fichero de entrada, el uso de un mayor número de procesadores reduce la eficiencia del experimento. Disponer de un mayor número de procesadores implica: un mayor número de comunicaciones (para distribuir las secuencias de búsqueda y recibir resultados) y una mayor sobrecarga para el maestro (encargado de procesar los metadatos recibidos por los esclavos).

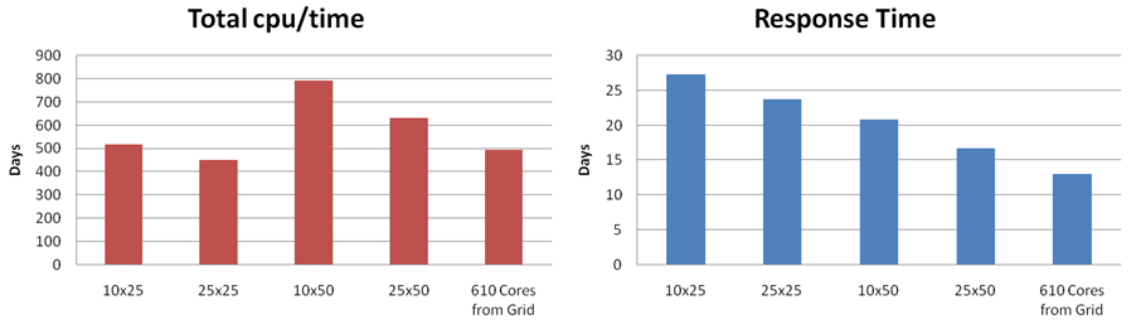

Figura 22. Tiempo de respuesta y tiempo total de CPU empleado por cada una de las 5 configuraciones.

4.3 Conclusiones

Los gráficos de la Figuras 21 y 22 muestran una comparativa entre la aproximación del Supercomputador y la aproximación del Grid. Como puede apreciarse, la eficiencia en las infraestructuras Grid es más pobre, aunque la productividad obtenida es mayor (menor tiempo de respuesta).

En el caso de la supercomputación, la única acción que podría llevarse a cabo para mejorar las prestaciones sería probar nuevas configuraciones, hasta encontrar la óptima. En las infraestructuras Grid, en cambio, es posible mejorar muchos aspectos de la metodología descrita en el capítulo anterior. Por ello, el resto de capítulos de la tesis se ocuparán de investigar diferentes modelos de lanzamiento de trabajos y medidas para predecir con mayor exactitud el tiempo de respuesta de los trabajos Grid.

5. COMPARATIVA ENTRE DOS MODELOS DE GESTIÓN DE TRABAJOS EN INFRAESTRUCTURAS GRID

En el punto anterior se realizó una comparativa entre las dos infraestructuras capaces de dar soporte a la ejecución de experimentos masivos: el Grid y la Supercomputación. Utilizando el mismo caso de uso, se llevaron a cabo experimentos en ambas plataformas y los resultados obtenidos fueron analizados. A partir de dichos resultados, se concluyó que determinados aspectos del esquema de gestión de trabajos del Grid podían ser mejorados notablemente, tales como reducir la tasa de trabajos fallidos. Por tanto, el diseño e implementación de un esquema de gestión de trabajos adecuado, supone una de las decisiones más importantes a la hora de desplegar y ejecutar experimentos masivos en las infraestructuras Grid. Por este motivo, el presente capítulo comenzará describiendo y analizando los pros y contras de dos modelos de gestión de trabajos típicos: el modelo de 'bolsa de trabajos' y el modelo de 'trabajos piloto'. A continuación, se llevarán a cabo experimentos y se compararán las prestaciones obtenidas por cada uno de ellos.

5.1 Modelos de gestión de trabajos

En esta sección se introducen dos modelos de gestión de trabajos típicos: el modelo de 'bolsa de trabajos' y el modelo de 'trabajos piloto', empleando como caso de uso el alineamiento de un gran número de secuencias genómicas contra las contenidas en una determinada base de datos.

5.1.1 Tareas de pre-procesamiento

Previo a la ejecución de los experimentos, en ambos esquemas de gestión de trabajos, es recomendable realizar una serie de tareas de pre-procesamiento. Puesto que estas tareas de pre-procesamiento ya han aparecido en capítulos anteriores, se comentarán de forma muy somera.

Como ya se expuso en el capítulo anterior, las principales aproximaciones paralelas utilizadas en herramientas bioinformaticas, como BLAST, se basan en el particionamiento de los datos de entrada. Dado que BLAST posee dos datos de tipos de entrada, es posible realizar la paralelización de dos formas distintas: particionando el fichero de secuencias de búsqueda (enfoque apropiado para las infraestructuras Grid) o la base de datos (estrategia adecuada para ser empleada en un supercomputador). Como en este capítulo la ejecución de experimentos se realizará únicamente en infraestructuras Grid, la primera tarea de pre-procesamiento consistirá en elegir la estrategia de partición más adecuada para el fichero de secuencias de búsqueda. Para ello existían dos posibles estrategias: dividir el fichero por número de secuencias (el fichero de entrada se divide en múltiples bloques con, aproximadamente, el mismo número de secuencias cada uno) o por tamaño (los fragmentos resultantes del proceso de fragmentación tienen aproximadamente el mismo tamaño). Se deberá optar por aquella estrategia que logre distribuir el coste computacional de las tareas de la forma más homogénea posible.

La siguiente tarea de pre-procesamiento consiste en seleccionar los recursos cómputo (CEs) y de almancenamiento (SEs) más apropiados, para obtener las mejores prestaciones posibles en los experimentos. La selección de los CEs deberá realizarse en torno a aspectos como la velocidad del procesador o la fiabilidad del recurso, mientras que los SEs serán elegidos atendiendo a requerimientos técnicos como un ancho de banda razonable y la memoria física disponible. Asimismo, otro aspecto clave en esta selección es poner especial atención a las prestaciones conjuntas de los elementos de cómputo y los elemento de almacenamiento.

La última tarea de pre-procesamiento está relacionada con el segundo tipo de entrada del programa BLAST: el fichero de base de datos. Como consecuencia del gran tamaño de las bases de datos típicas (del orden de los gigabytes), este tipo de ficheros no pueden ser adjuntados en la bandeja de entrada de los trabajos grid. Por tanto, para poder acceder a estas bases de datos, los trabajos se verán obligados a recuperar la base de datos en tiempo de ejecución, transfiriéndola desde un elemento de almacenamiento de la infraestructura a su espacio de memoria local. La última tarea consistirá justamente en replicar el fichero de la base de datos en los SEs seleccionados anteriormente, lo que favorecerá el acceso a la misma por parte de los trabajos.

5.1.2 Modelo 'bolsa de trabajos'

El modelo de 'bolsa de trabajos' sigue la metodología de lanzamiento de trabajos descrita en los capítulos anteriores, con unas ligeras modificaciones.

El punto de partida de este modelo es una conjunto de tareas pendientes de ser ejecutadas en los nodos de proceso. La característica que distingue a este modelo de otros, es que un trabajo siempre tendrá a su cargo la ejecución de una sola tarea (el alineamiento de un fragmento generado en la fase de particionamiento). A modo de resumen, las tareas particulares de este modelo eran: generar la especificación de los trabajos que ejecutarán las tareas, calcular la distribución de la carga (número de trabajos lanzados a cada recurso), monitorizar el estado de los trabajos y relanzar trabajos fallidos o tentativos de estar bloqueados y finalmente recuperar los resultados de los elementos de almacenamiento.

En cuanto a las modificaciones realizadas, la primera está relacionada con la estimación del tiempo máximo de un trabajo (el 'wall-time'). En anteriores experimentos se optaba por calcular únicamente un 'wall-time' para todos los trabajos. Sin embargo, dado que el ciclo de vida de un trabajo está compuesto de varias fases de duración no uniforme, esta opción resulta inadecuada. Con el fin de detectar trabajos no productivos en las primeras etapas, resultará de gran utilidad estimar un 'wall-time' para cada uno de los estados por los que atraviesa un trabajo grid.

La segunda modificación realizada es un refinamiento de la política de relanzamiento de trabajos. A la hora de decidir el destino de un relanzamiento, la experiencia aconseja dar prioridad a aquellos recursos, que durante la ejecución del experimento, consiguen ejecutar satisfactoriamente los trabajos enviados. Por tanto, los candidatos más apropiados para albergar un trabajo previamente fallido serán aquellos recursos que ya hayan completado con éxito algún trabajo.

5.1.3 Modelo de 'trabajos piloto'

El modelo de trabajos piloto empleado en esta tesis, está basado en la arquitectura maestro-esclavo implementada por las herramientas DIANE y Ganga.

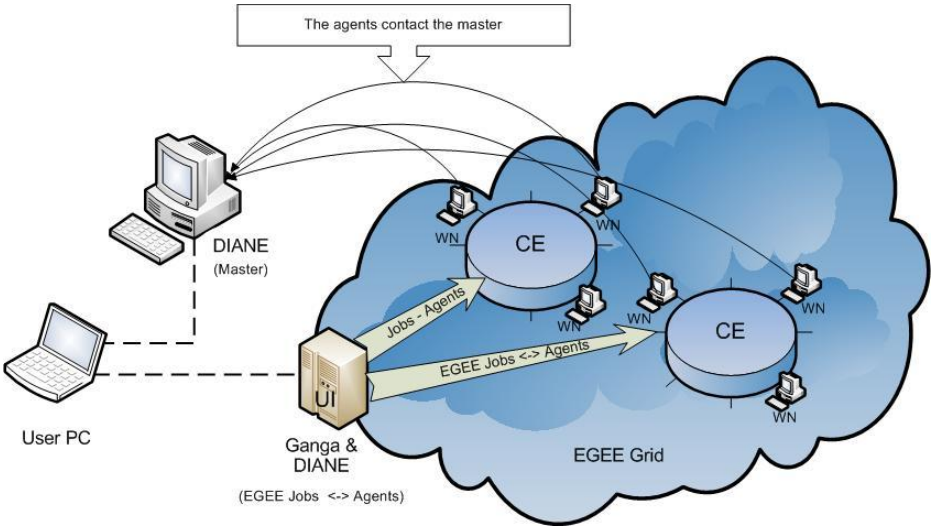

Figura 23. Esquema de funcionamiento del modelo basado en trabajos piloto.

Como puede observarse en la Figura 23, el modelo de trabajos piloto comienza con la ejecución del proceso maestro, un pequeño servidor local. Este proceso maestro será el encargado de proporcionar las tareas (alineamiento de los fragmentos generados durante el particionado de los datos) a los nodos esclavo, hasta que todas las tareas hayan sido completadas con éxito. Los esclavos son trabajos grid ejecutándose en los nodos de cómputo de la infraestructura, con la particularidad de que son capaces de comunicarse con el maestro. El maestro debe ocuparse de vigilar la evolución de las tareas y asegurarse de que todas son completadas correctamente al final del experimento, mientras que los esclavos ofrecen el acceso a un nodo previamente alcanzado.

Si, por alguna razón, una tarea falla o un esclavo pierde la comunicación con el maestro, inmediatamente, el maestro reasignará la tarea a otro esclavo. Además, en el caso de que un esclavo termine una determinada tarea y queden tareas pendientes por ejecutar, el maestro podrá asignarle una nueva al mismo esclavo. Por tanto, una diferencia sustancial con el anterior modelo es, que en este escenario, existe la posibilidad de que un esclavo procese más de una tarea. Gracias a esta característica, el sobrecoste derivado de la fase de pre-proceso (principalmente la descarga y formateo de la base de datos) puede reducirse considerablemente. Para ello, será necesario modificar adecuadamente el *shell-script* de ejecución de los trabajos.

No obstante, antes de poner en marcha la arquitectura maestro-esclavo, es necesario definir las características de la ejecución. En primer lugar, la especificación de una ejecución requiere definir la configuración del maestro (su 'heartbeat' y el timeout para los esclavos). También será necesario definir políticas de planificación del maestro como: el número máximo de veces que una tarea fallida será asignada a un mismo esclavo, qué acciones deben llevarse a cabo cuando una tarea falla, el máximo número de trabajos fallidos que se permite a un mismo esclavo.

La última tarea antes de proceder a ejecutar el experimento es especificar correctamente al maestro los argumentos de las tareas, ficheros comunes a todas ellas (el *shell-script* y

los ficheros auxiliares) y los ficheros específicos (básicamente el fragmento del fichero de secuencias de entrada).

A partir de este momento, el maestro puede ser iniciado utilizando la especificación anteriormente descrita. Los trabajos (que contienen los esclavos) serán lanzados a los nodos de cómputo del grid y el maestro quedará a la espera de recibir conexiones entrantes de los esclavos.

Como ya se ha comentado, los esclavos son trabajos genéricos que pueden ejecutar cualquier tarea que les indique el proceso maestro. Con la intención de mejorar las prestaciones del experimento, se ha desarrollado un automatismo de control de los esclavos. La finalidad de este automatismo es asegurar que el maestro siempre está en contacto con un determinado número de esclavos (en función del número de tareas pendientes). Para ello, el automatismo puede preguntar al maestro el número de esclavos vivos, es decir, que siguen comunicándose con él. El automatismo también será el encargado de lanzar los trabajos-esclavo a los CEs escogidos en la fase de pre-proceso de selección de los recursos óptimos. Tras iniciar su ejecución, lo primero que hará el proceso esclavo será registrarse en el maestro. A cambio, este le asignará una tarea pendiente, además de transferirle la información necesaria para la ejecución (argumentos y ficheros de pequeño tamaño, como los incluidos en un JDL).

Este esquema tiene diversas ventajas derivadas del hecho de que un trabajo pueda ejecutar más de una tarea. En primer lugar, la planificación dinámica utilizada por este esquema reduce el desequilibrio de la carga entre los recursos. Sólo cuando un esclavo haya finalizado su tarea, el maestro le asignará una nueva. Otra diferencia importante con el modelo anterior, es que cuando una tarea se completa, no es necesario lanzar un nuevo trabajo, reduciendo de esta manera el tiempo de espera en la cola de los recursos de cómputo. Finalmente, aquellos trabajos que ejecutan más de una tarea sólo necesitan ejecutar la fase de pre-procesamiento la primera vez. Puesto que esta etapa de pre-proceso contiene la operación más crítica de la ejecución de un trabajo (la descarga de la base de datos), el uso de un modelo basado en trabajos piloto podría influir positivamente en la tasa de trabajos fallidos obtenida.

5.2 Experimentos

Con el fin de apreciar la importancia del modelo de gestión de trabajos utilizado se ejecutaron dos experimentos en la infraestructura grid EELA, que mostraran el comportamiento de ambos esquemas.

Para la comparación se han utilizado los mismos datos en ambos escenarios. Por un lado, la base de datos empleada es la GenBank no redundante. La versión utilizada contiene 19.871.612 secuencias de ADN y su volumen (no comprimido) es de aproximadamente 3,4 Gigabytes. Por otro lado, el fichero de secuencias de búsqueda contiene las secuencias de un determinado virus. El fichero consta de 103.859 secuencias y un volumen de 10,6 Megabytes aproximadamente. Posteriormente este fichero se fragmentó utilizando la estrategia de división por tamaño, dando lugar a 54 bloques de 59 KB cada uno. Se estimó que el tiempo de cómputo secuencial aproximado para realizar el alineamiento completo de este virus es aproximadamente de 32 días.

En cuanto a los parámetros de la ejecución BLAST, evidentemente también se utilizaron los mismos para ambos modelos: el número de salidas o 'hits' (20.000), el expected value mínimo (0.01) y el algoritmo a utilizar (tblastx).

5.2.1 Experimento con el modelo 'bolsa de trabajos'

Previo a la ejecución del experimento, era necesario decidir una distribución de los trabajos óptima. Finalmente se seleccionaron 52 elementos de cómputo de entre los recursos más productivos en experimentos previos, asignando mayor cantidad de trabajos a los recursos con un historial más productivo.

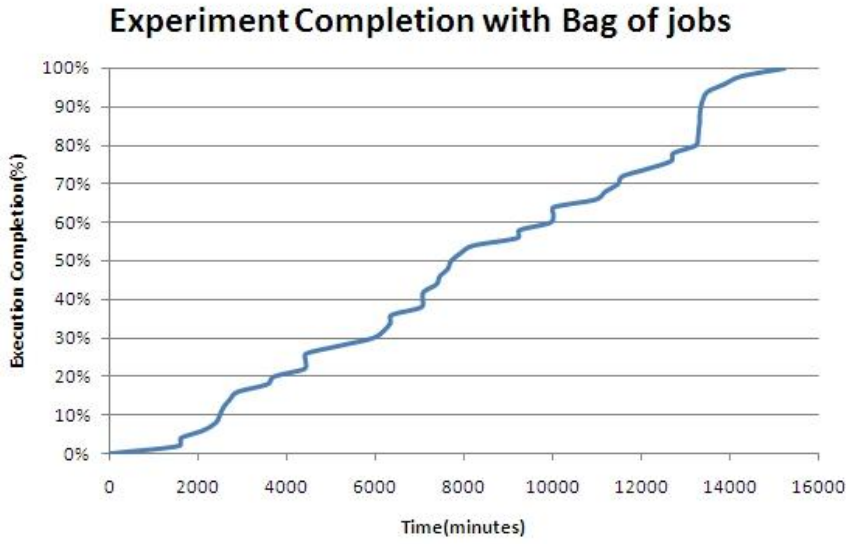

Figura 24. Gráfica de evolución de la finalización del experimento con el modelo de 'bolsa de trabajos'.

La Figura 24 muestra la evolución de la finalización del experimento. En concreto, el experimento requirió el uso de la infraestructura durante 254 horas (10 días y 14 horas) aunque el tiempo de CPU total consumido fue de 798 horas (33 días), obteniéndose un speed-up al final del experimento de 3,14 (siendo 54 el máximo téorico posible). Al final, el número de trabajos lanzados durante el experimento fue 82, lo que implica una tasa de fallos del 54%.

Una vez finalizado el proceso, se obtuvieron 54 ficheros comprimidos con gzip con un tamaño total de 3503 KB.

5.2.2 Experimento con el modelo de 'trabajos piloto'

Como se ha mencionado anteriormente, el experimento con el modelo de 'trabajos piloto' se llevó a cabo mediante el uso de dos herramientas desarrolladas en el CERN: DIANE (Distributed Analysis Environment) [30] y Ganga [31]. DIANE es una herramienta para gestionar un gran número de tareas independientes y Ganga es la interfaz que utiliza DIANE para lanzar los trabajos esclavo.

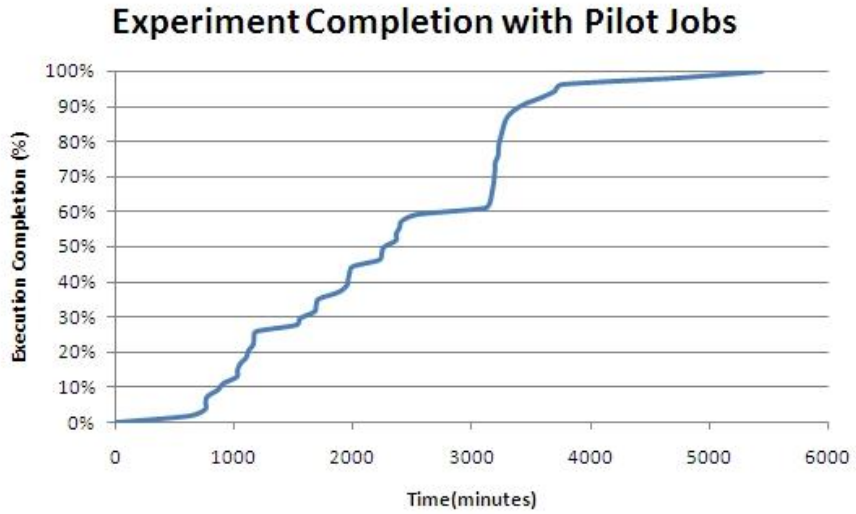

Figura 25. Gráfica de evolución de la finalización del experimento con el modelo de 'trabajos piloto'.

En la Figura 25 puede observarse la evolución de la finalización de este experimento. En esta ocasión, el experimento necesitó tan sólo 90 horas (menos de 4 días) para finalizar y el tiempo total de CPU consumido fue de 551 horas (alrededor de 23 días). Por tanto, el speed-up global del experimento es dos veces el obtenido con el otro modelo. Asimismo, gracias a las características de este modelo, se logró reducir la tasa de trabajos fallidos al 18%.

5.2.3 Comparativa del tiempo total de CPU consumido por ambos experimentos

Los resultados reflejados en la Figura 26, muestran que el modelo basado en 'trabajos piloto' consigue finalizar por completo el experimento en 6 días menos que el esquema 'bolsa de trabajos'. Esta importante reducción del tiempo de respuesta está ligada, en cierta manera, a los resultados observables en la Figura 27.

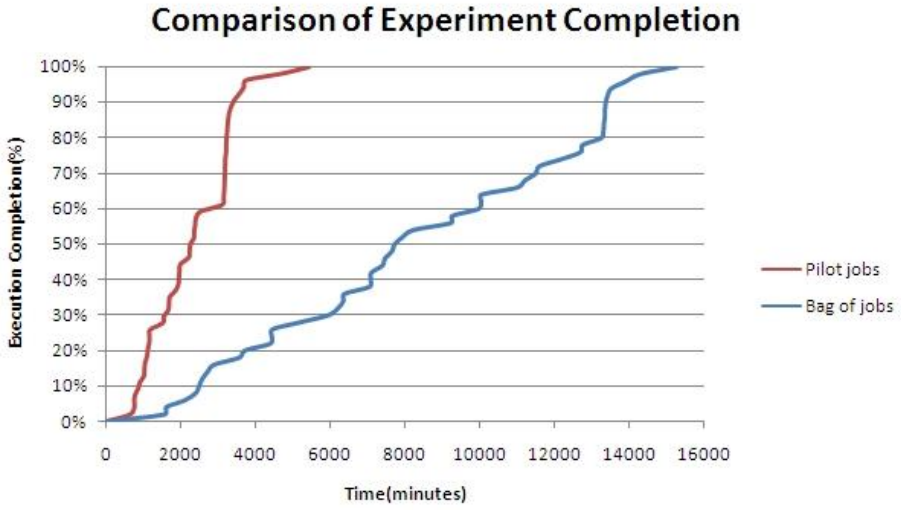

Figura 26. Comparativa de la evolución de la finalización del experimento para ambos modelos.

La Figura 27 ofrece una comparativa entre el tiempo total de CPU consumido por las tareas finalizadas con éxito, en ambos modelos. De esta forma, el proceso de ejecución

de una tarea se ha dividido en tres fases: tiempo de cola, tiempo de pre-proceso y tiempo de alineamiento. Como se observa, los trabajos en el primer experimento permanecen un total de 530 horas en la cola mientras que en el segundo experimento este valor es de 431 horas. La explicación a esta observación es bastante evidente: en el sistema de 'trabajos piloto' el número de trabajos ejecutando tareas es menor, dado que existe la posibilidad de que un trabajo ejecute más de una tarea. Específicamente, el número de tareas que no requirieron esperar en la cola de un CE fue 12. Por este mismo motivo, mientras el tiempo empleado en la etapa de pre-proceso en el primer experimento es de 36 horas en el segundo es de 20 horas. Como ya se comentó, el programa de ejecución de un trabajo fue modificado para que sólo fuese necesario realizar el pro-proceso la primera vez. Por último, se aprecia una curiosa diferencia entre el tiempo de alineamiento empleado en ambos esquemas (762 horas para el modelo 'bolsa de trabajos' y 531 para el modelo 'trabajos piloto').

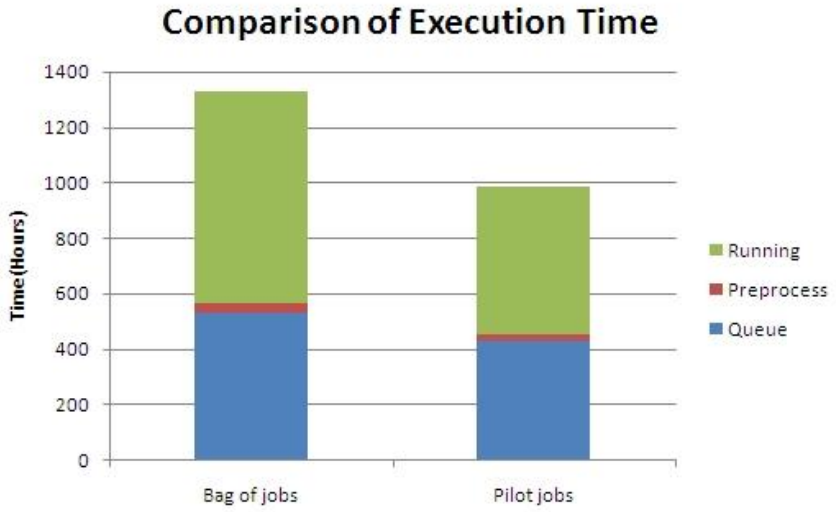

Figura 27. Comparativa entre el tiempo total de CPU consumido por ambos modelos.

5.3 Conclusiones

La principal conclusión que se extrae en este capítulo es la siguiente: el uso del modelo de 'trabajos piloto' mejora las prestaciones obtenidas por el modelo de gestión de trabajos tradicional. Respecto a los inconvenientes de este modelo, el primero de ellos es el hecho de estar basado en otras herramientas (DIANE y Ganga), las cuales son poco flexibles y no permiten, por ejemplo, aprovechar las listas blancas para relanzar los trabajos fallidos en aquellos recursos que ofrezcan un mejor comportamiento. Con esta medida, las prestaciones obtenidas con este modelo podrían mejorar todavía más. Por último, debido a las características de la arquitectura maestro-esclavo, el proceso maestro puede llegar a saturarse en aquellos experimentos con un gran número de trabajos (decenas de miles de trabajos).

6. FORMULACIÓN DE UN MODELO DE ESTIMACIÓN DEL TIEMPO DE EJECUCIÓN DE TRABAJOS BLAST

En el capítulo 5 se demostró que una de las decisiones más importantes, a la hora de realizar experimentos en infraestructuras Grid, es elegir el modelo de gestión de trabajos más adecuado. Utilizando el mismo caso de uso, se compararon las prestaciones obtenidas por dos esquemas de planificación de trabajos distintos: el modelo 'bolsa de trabajos' y el modelo 'trabajos piloto'. A partir de los resultados, se concluyó que el modelo basado en 'trabajos piloto' lograba mejorar no sólo el tiempo de respuesta, sino también la eficiencia obtenida por el otro modelo considerado. Sin embargo, la implementación del esquema 'trabajos piloto' no estaba exenta de inconvenientes. Por esta razón, el presente capítulo se centrará en mejorar las prestaciones del modelo 'bolsa de trabajos'. El principal inconveniente de este modelo era el alto porcentaje de trabajos relanzados, lo que provocaba un importante aumento del tiempo de respuesta del experimento (completar el 20% de los trabajos requería el 70% del tiempo de respuesta total). En dicho esquema, un trabajo podía ser relanzado por dos motivos: el fallo debido a problemas de configuración o disponibilidad de los recursos, y una ejecución más extensa de lo normal. Por un lado, la medida adoptada por el automatismo para reducir el número de trabajos relanzados por fallo consistía en realizar una selección óptima de los recursos. Por el otro lado, la detección de trabajos bloqueados o ejecutándose lentamente en los recursos se basaba en homogeneizar el coste computacional de las tareas (mediante un particionamiento adecuado), asignando el mismo tiempo de vida a todas las etapas del ciclo de vida de todos los trabajos. Puesto que las infraestructuras Grid se caracterizan por la gran heterogeneidad de sus recursos, asignar a la etapa de ejecución de todas las tareas el mismo tiempo de vida parece poco adecuado. De hecho, esto da a lugar a situaciones en las que trabajos productivos son cancelados prematuramente y trabajos improductivos son cancelados tardíamente. Lo ideal sería disponer de algún procedimiento que permitiese estimar con precisión la duración del ciclo de vida de un trabajo Grid. Por tanto, el presente capítulo se ocupará en realizar una primera aproximación a esta medida, mediante la formulación de un modelo que permita estimar el tiempo de alineamiento de una determinada tarea BLAST en un recuso con unas característica determinadas.

El capítulo comienza enumerando los posibles factores que pueden tener influencia en el tiempo de proceso de BLAST. A continuación se describen una serie de experimentos llevados a cabo para verificar la influencia de los factores anteriormente mencionados. Finalmente se expone el procedimiento seguido para obtener el modelo de prestaciones de tareas BLAST.

6.1 Factores relevantes

El tiempo de proceso (o alineamiento) de BLAST depende de un conjunto de factores que pueden ser clasificados en: dependientes de la aplicación (BLAST) y dependientes del recurso de cómputo.

6.1.1 Factores dependientes de la aplicación

Parece lógico pensar que dos factores que influyen, claramente, en el tiempo de proceso de un trabajo BLAST son el tamaño del fichero de entrada y el tamaño de la base de datos. También, un aspecto a tener en cuenta será el grado de similitud entre las secuencias desconocidas del fichero de entrada y las secuencias anotadas de la base de datos. Debido al comportamiento heurístico de BLAST, la comparación de dos secuencias muy distintas puede dar lugar a tiempos de proceso muy cortos. El último factor tentativo de influir en las prestaciones es el número de 'hits', es decir, el número máximo de resultados que el programa BLAST escribirá en el fichero de resultados, para cada una de las secuencias de búsqueda.

6.1.2 Factores dependientes del recurso de cómputo

Los posibles factores dependientes del elemento de cómputo utilizado son la velocidad del procesador y el tamaño de la memoria principal. A su vez, la velocidad del procesamiento se puede desglosar en otros dos parámetros: la rapidez de ejecución de las instrucciones enteras y las de coma flotante.

6.2 Experimentación

Esta sección describe, en detalle, los experimentos realizados con el fin de verificar la importancia de los factores destacados en el anterior apartado. Posteriormente, aquellos parámetros identificados como relevantes serán utilizados para formular el modelo de estimación del tiempo de ejecución.

6.2.1 Infraestructura utilizada

La infraestructura en la que se han realizado todos los experimentos es EGEE, en concreto en la Organización Virtual *biomed*. Dicha infraestructura se ha seleccionado teniendo en cuenta su madurez y la amplia variedad de recursos de cómputo disponibles (más de 139336 núcleos accesibles simultáneamente).

6.2.2 Experimento 1: Comprobación de la influencia del tamaño de los datos de entrada

Para comprobar la influencia del tamaño de los datos de entrada se optó por crear diversos ficheros de secuencias de entrada y de bases de datos con distintos tamaños, pero teniendo en cuenta que el tamaño promedio de las secuencias fuera similar para todos los ficheros. En particular, para las pruebas realizadas se generaron 10 ficheros de secuencias de entrada, siendo el fichero más pequeño de 0,5 MB y el más grande de 9 MB, con un incremento invariable de 0,5 MB entre un fichero y otro. Por lo que respecta a las bases de datos, se crearon 9 ficheros con tamaños comprendidos entre 50 y 450 MB, utilizando un incremento constante de 50 MB. Utilizando todos estos ficheros, el objetivo del experimento consistió en ejecutar un trabajo por cada fichero de entrada y base de datos generados, dando lugar a un total de 90 tareas. Evidentemente,

para poder apreciar el efecto de estos dos factores, todas las tareas fueron ejecutadas en un mismo nodo de proceso de la infraestructura Grid.

A simple vista, los resultados mostrados en la Figura 28 muestran una relación lineal entre el tamaño del fichero de secuencias de entrada y el tamaño de la base de datos. Como ya se sospechaba, cuánto mayor es el número de secuencias de búsqueda y objetivo (las contenidas en la base de datos), mayor es el tiempo requerido para procesarlas.

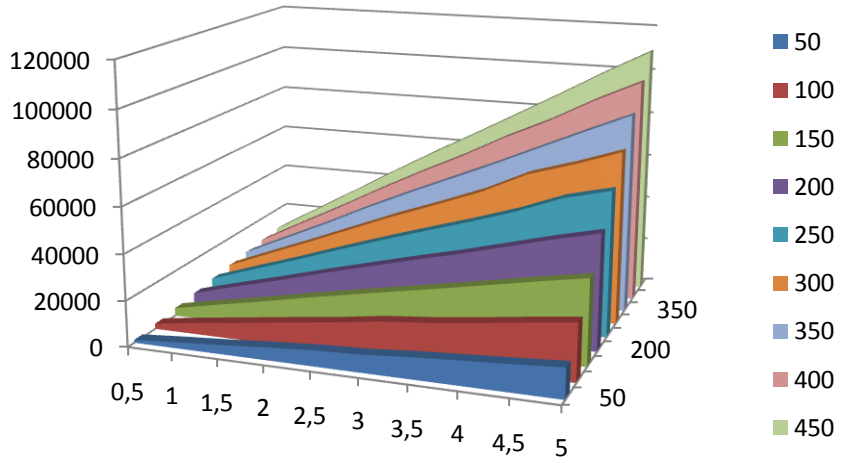

Figura 28. Tiempo de respuesta (segundos en el eje vertical) de tareas BLAST ejecutadas en el mismo nodo de cómputo, utilizando diferentes tamaños del fichero de entrada (MB en el eje inferior) y de la base de datos (MB en el eje inferior derecho).

6.2.3 Experimento 2: Comprobación de la influencia del grado de similitud entre secuencias

El cometido del segundo experimento es comprobar si el tiempo de alineamiento de un trabajo BLAST depende del grado de similitud entre las secuencias de búsqueda y de la base de datos. Para ello, se generaron tres nuevas versiones de una base de datos concreta, mediante la sustitución del 1%, 5%, y 10% de su contenido con valores aleatorios coherentes (caracteres correspondientes a una de las cuatro bases que componen el ADN: A,C,G,T). Como ficheros de secuencias de entrada, se emplearon los 9 ficheros creados en el experimento anterior.

Como puede verse en la Figura 29, los resultados son prácticamente idénticos para todas las versiones de la base de datos. Por tanto, se puede concluir que el tiempo de alineamiento de BLAST es independiente de la similitud entre las secuencias de ambos ficheros de entrada.

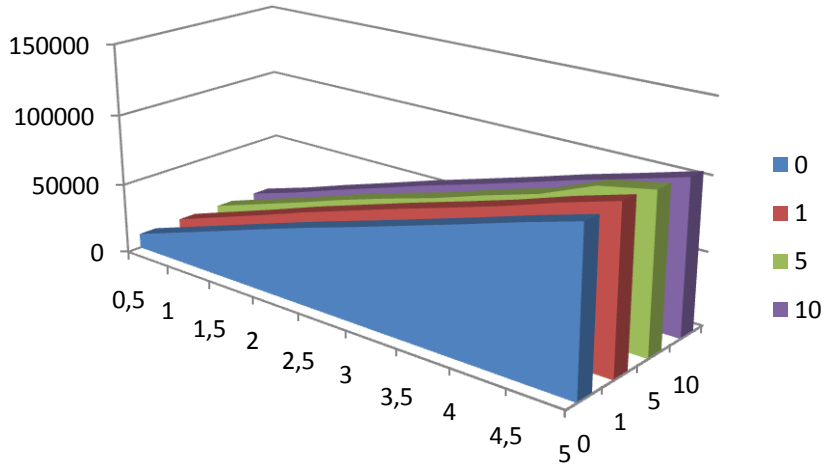

Figura 29. Tiempo de respuesta (segundos en el eje vertical) de tareas BLAST ejecutadas en el mismo nodo de cómputo, utilizando diferentes tamaños del fichero de secuencias de búsqueda (MB en el eje horizontal) y la cantidad de contenido modificado en la base de datos (% en el eje inferior derecho).

6.2.4 Experimento 3: Comprobación de la influencia del parámetro 'bhits'.

El experimento llevado a cabo para comprobar la influencia del último parámetro considerado consistió en ejecutar un conjunto de tareas BLAST en un mismo recurso y con los mismos datos de entrada, pero variando el parámetro 'bhits' desde un valor base de 5000 hasta 25000, con un incremento constante de 2500. Para minimizar el efecto de resultados anómalos, cada tarea se ejecutó varias veces y como valor final se tomó el tiempo de ejecución promedio.

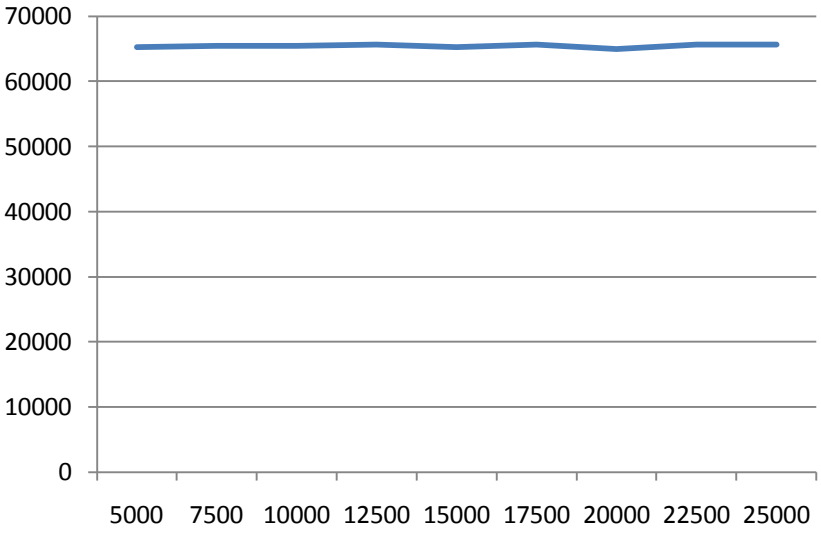

Figura 30. Tiempo de ejecución (segundos en el eje vertical) de tareas BLAST ejecutadas en el mismo nodo de cómputo, utilizando diferentes valores del parámetro 'bhits' (eje horizontal).

Como puede observarse en la Figura 30, el valor del tiempo de respuesta es aproximadamente 65000 para todos los trabajos lanzados. Por tanto, se puede afirmar que el valor del parámetro 'blast hits' no tiene influencia en el coste computacional de

una tarea BLAST. Al igual que en el caso del grado de similitud, este parámetro no será tenido en cuenta a la hora de formular el modelo de prestaciones.

6.2.5 Experimento 4: Comprobación de la influencia del nodo de computación

La finalidad del cuarto y último experimento es analizar la influencia de los factores relacionados con las prestaciones de un elemento de cómputo del grid. Para ello, lo primero que se hizo fue obtener, mediante consultas al BDII, los valores de prestaciones publicados por cada nodo de cómputo. El funcionamiento del BDII, al igual que todos los Servicios de Información del Grid, está regido por el estándar GLUE Schema, el cual define que datos deben proporcionar los recursos de información. De todos los valores ofrecidos por el BDII de EGEE, los tres valores de utilidad para este estudio son: los 'benchmarks' SpecInt2000 (indicador de la velocidad de proceso de operaciones enteras) y SpecFloat2000 (indicador de la velocidad de proceso de las operaciones en coma flotante), y el tamaño de la memoria principal. Haciendo uso de esta información, 19 CEs con diferentes velocidades y tamaños de memoria fueron elegidos para ejecutar los trabajos de test.

Tras la finalización de todos los trabajos de test, se procedió a realizar un estudio de correlación entre los benchmarks SpecInt2000 y SpecFloat2000 y los tiempos de ejecución observados (recuperados de las salidas estándar de los ficheros).

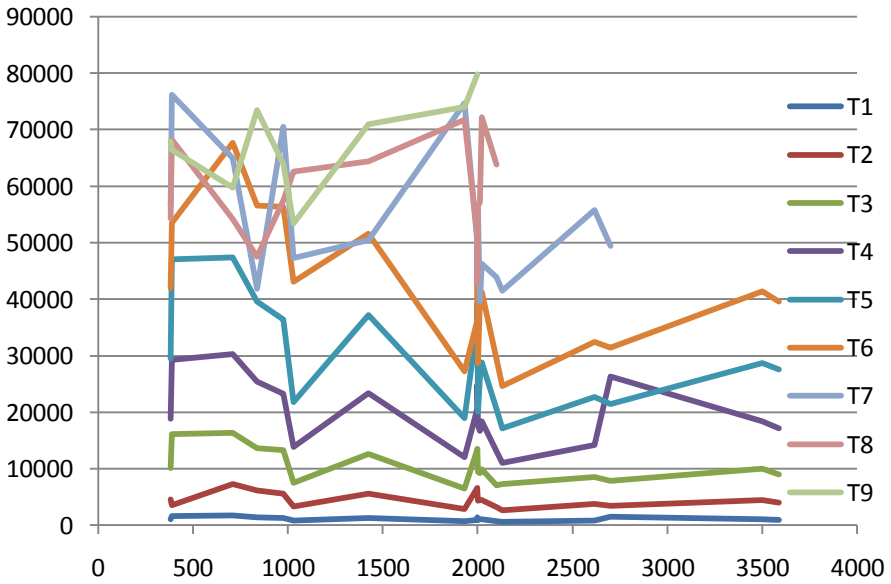

Figura 30. Tiempo de alineamiento (segundos en el eje vertical) para 9 combinaciones diferentes del tamaño del fichero de entrada y el tamaño de la base de datos en función del Spec2000 promedio (unidades en el eje horizontal).

Para poder observar la correlación existente entre los valores de los benchmarks Spec2000 y las prestaciones reales de BLAST, se realizó un gráfico como el de la Figura 31. Esta Figura muestra el tiempo de alineamiento para 9 tipos diferentes de trabajos (combinaciones de los tamaños del fichero de entrada y el fichero de base de datos) en función del Spec2000. Por Spec2000 se entiende el valor resultante de calcular la media aritmética de los valores SpecInt2000 y SpecFloat2000.

Como puede apreciarse a simple vista, los valores de los benchmarks SpecInt y SpecFloat no están correlacionados con el tiempo de alineamiento de BLAST: un mayor valor del Spec2000 no implica la obtención de un menor valor del tiempo de ejecución.

Para comprobar numéricamente la anterior afirmación, se calculó el coeficiente de correlación de Pearson para cada tipo de trabajo (r de Pearson).

	T1	T2	T3	T4	T5	T6	T7	T8	T9
Pearson	-0,433	-0,309	-0,533	-0,42	-0,521	-0,549	-0,413	0,079	0,55
p	0,052	0,175	0,014	0,06	0,017	0,011	0,081	0,772	0,089

Tabla 1. Índices de correlación para los 9 tipos de trabajos considerados.

A la vista de los resultados ofrecidos por la Tabla 1, puede afirmarse que existe poca dependencia lineal entre los benchmarks Spec2000 y las prestaciones de BLAST para cualquier tipo de trabajo: la r de Pearson varía entre 0.08 y 0.55 (muy distinta de los valores extremos 1 y -1). Asimismo, en la segunda fila de la tabla se puede comprobar que la significancia estadística es muy pobre para todos los tipos de trabajos empleados, siendo especialmente llamativo el caso en el que la relación entre ambas variables es debida a pura aleatoriedad (77% de probabilidad).

La principal explicación a la inexistencia de correlación observada es el hecho de que Spec2000 sea un benchmark de pago. Como consecuencia de esto, en lugar de ejecutar los benchmarks, muchos administradores de Sistemas Grid optan por obtener directamente los valores de las tablas correspondientes a los procesadores de que disponen.

Por tanto, se hacía necesaria la definición de un nuevo estimador del tiempo de ejecución, basado únicamente en los tiempos observados para cada nodo. En concreto, el estimador utilizado, en adelante estimador empírico, es el ratio de prestaciones promedio con respecto a un nodo fijo. De esta manera, para calcular los valores del nuevo estimador lo que se hizo fue obtener, para cada nodo, la suma de los tiempos de cómputo (considerando el mismo número y tipo de trabajos). Finalmente, a partir de estos tiempos totales de cómputo y la elección de un nodo de referencia, se hallaron las aceleraciones.

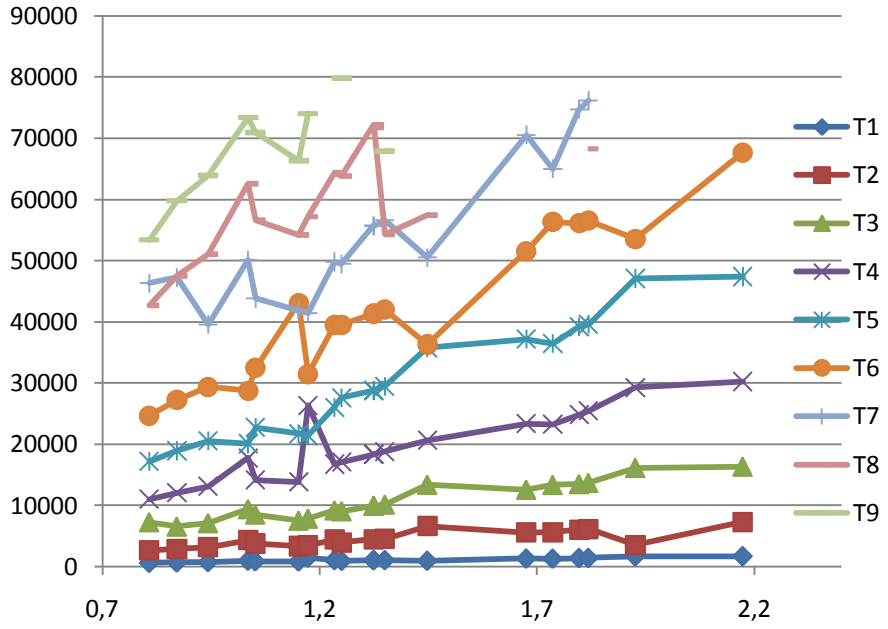

Figura 32. Tiempo de alineamiento (segundos en el eje vertical) para 9 tipos de trabajos diferentes en función del ratio de prestaciones promedio con respecto a un nodo fijo (unidades en el eje horizontal).

La Figura 32 muestra gráfica la correlación entre el tiempo de alineamiento y el nuevo estimador. Sin embargo, de forma análoga al caso del estimador antiguo, la Tabla 2 cuantifica al correlación existente entre ambos parámetros, por medio de la r de Pearson.

	T1	T2	T3	T4	T5	T6	T7	T8	T9
Pearson	0,888	0,788	0,957	0,897	0,978	0,96	0,889	0,691	0,73
p	1,86E-07	3,33E-05	6,65E-11	8,72E-08	2,10E-13	3,12E-11	8,33E-07	0,004	0,015

Tabla 2. Índices de correlación para los 9 tipos de trabajos, utilizando el estimador empírico.

En contraste con los resultados ofrecidos por la Tabla 1, la Tabla 2 muestra una mayor dependencia lineal entre ambas variables (el tiempo de cómputo y las prestaciones del estimador propuesto). Obviamente, la significación estadística es también mayor, con valores siempre por encima de 98% y en general por encima del 99%.

Por lo que respecta al tamaño de la memoria principal del nodo no se observó una dependencia significativa, por lo que será excluida del modelo de prestaciones final.

6.3 Modelo de prestaciones

Antes de proponer un modelo, será útil recordar todas las conclusiones extraídas en el punto anterior.

En primer lugar, se concluyó que había una dependencia lineal con ambos tipos de ficheros de entrada: el fichero de secuencias de búsqueda y el fichero de bases de datos. A continuación se comprobó que el grado de similitud entre las secuencias de ambos ficheros de entrada no tiene influencia alguna sobre el tiempo requerido por el alineamiento. También se observó que el parámetro número de hits era independiente del tiempo de respuesta de las tareas. En último lugar, se concluyó que existía una dependencia directa con la velocidad de procesamiento del elemento de cómputo pero no con el tamaño de la memoria.

De acuerdo con lo anterior, se propuso el modelo mostrado en la Fórmula 1.

$$T = \frac{P_{Sp}}{Sp} P_{Tm}\left(\frac{T_{inp}}{T_{inp\_basal}} \frac{T_{BD}}{T_{BD\_basal}}\right) + K .$$

Fórmula 1. Modelo de prestaciones propuesto.

Donde:

- P_{Sp}, P_{Tm} and K son los parámetros desconocidos del modelo de regresión.
- T_{inp} and T_{BD} son los tamaños de los ficheros de entrada empleados.

- Sp es la relación entre el tiempo de alineamiento en un determinado nodo y un nodo de referencia.
- T_{inp_basal} is 0,5 and T_{BD_basal} is 50 (tamaños basales).

El procedimiento para estimar los parámetros desconocidos de la Fórmula 1 se llevó a cabo en dos fases. El objetivo de la primera fase era determinar los parámetros relacionados únicamente con el tamaño de los ficheros de entrada, es decir, P_{Tm} y K. De esta forma, a partir de los resultados del nodo de referencia, se realizó una regresión lineal con el modelo expuesto en la Fórmula 2.

$$T = P_{Tm}\left(\frac{T_{inp}}{T_{inp\_basal}}\frac{T_{BD}}{T_{BD\_basal}}\right) + K$$

Fórmula 2. Modelo para la regresión lineal de la primera fase.

Mediante un método de estimación típico, como OLS (Ordinary Least Squares) se obtuvieron los valores de P_{Tm} y K: 835,62 y 1065,5 respectivamente. A continuación, era necesario completar el modelo de predicción mediante la estimación del parámetro relacionado con las prestaciones de un nodo de cómputo: P_{Sp}. Utilizando los datos de los 19 nodos considerados en el estudio, se definió un sistema compatible indeterminado con la intención de aplicar el método de mínimos cuadrados y obtener el modelo de predicción mostrado en la Fórmula 3.

$$T = \frac{1.35}{Sp}835.62\left(\frac{T_{inp}}{T_{inp\_basal}}\frac{T_{BD}}{T_{BD\_basal}}\right) + 1065.5$$

Fórmula 3. Modelo de predicción final.

Para apreciar la precisión del modelo de predicción formulado, las Figuras 33 y 34 realizan una comparativa entre el tiempo de alineamiento real y el estimado por el modelo de predicción.

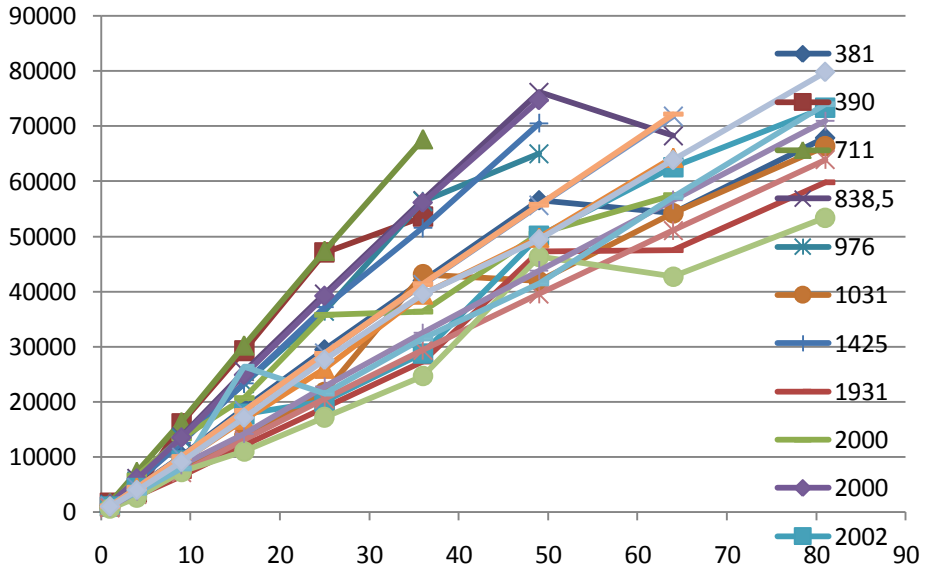

Figura 33. Tiempo de ejecución real (segundos en el eje vertical) para los 19 nodos de cómputo considerados en función de la talla del problema (unidades de tamaño relativo con respecto a un tamaño base en el eje horizontal).

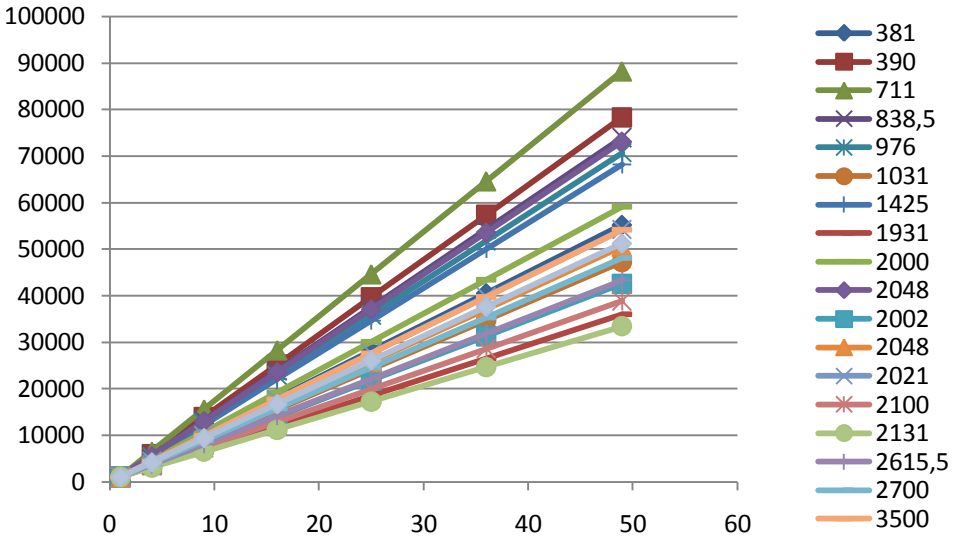

Figura 34. Tiempo de ejecución estimado (segundos en el eje vertical) para los 19 nodos de cómputo considerados en función de la talla del problema (unidades de tamaño relativo con respecto a un tamaño base en el eje horizontal).

7. CONCLUSIONES

Finalmente, ha llegado el momento de hacer un repaso de las principales conclusiones de este trabajo. Por ello, se ha dividido el capítulo de conclusiones en un sub-apartado con las contribuciones propias de este trabajo, otro sub-apartado con el conjunto de publicaciones generadas a partir del trabajo desarrollado en la presente tesis de máster y por último, un repaso a los trabajos futuros que han quedado pendientes y que deberán ser resueltos en breve.

7.1 Contribuciones propias

El objetivo de la presente tesis, como su título indica, era analizar y caracterizar una de las herramientas bioinformáticas más importantes, BLAST, para su ejecución eficiente tanto en infraestructuras Grid como Supercomputación.

Por tanto, en primer lugar se ha desarrollado un modelo genérico para el despliegue de experimentos masivos en ambas infraestructuras, sentando las bases del trabajo posteriormente realizado. En esta parte se han descrito los diferentes componentes que conforman el sistema y los elementos de configuración que tendrán un impacto directo en las prestaciones finales.

A continuación, utilizando el modelo anterior, se ha realizado una comparativa entre las prestaciones obtenidas por la versión secuencial de BLAST en una infraestructura Grid y una versión paralela de dicho programa (mpiBLAST) en un Supercomputador, utilizando un conjunto de datos real (el metagenoma del Mar de los Sargazos). A la vez, la ejecución de este caso de uso ha sido útil para uno de los grupos con los que se ha colaborado en el marco de esta tesis de máster.

A partir de los resultados obtenidos en la comparativa de infraestructuras se decidió incidir en el caso de las infraestructuras Grid, analizando dos modelos de gestión de trabajos diferentes y llevando a cabo experimentos que mostraran los pros y contras de cada de uno de ellos.

Finalmente, con vistas a mejorar la planificación de los trabajos BLAST en el Grid se realizó la caracterización de dicha aplicación. Para ello, en primer lugar se ejecutaron una serie de experimentos que permitieran identificar los factores influyentes en el tiempo de alineamiento de trabajos BLAST. Seguidamente, a partir de los factores identificados como relevantes, se procedió a formular un modelo de predicción del tiempo de respuesta de trabajos BLAST en el Grid.

7.2 Publicaciones derivadas

En este apartado se enumeran un conjunto de publicaciones, todas ellas artículos de congresos, generadas en torno al trabajo de esta tesis de máster.

- I. Blanquer, A.A. Carrión, V. Hernández, M. Pignatelli, J. Tamames: A Comparison Between mpiBLAST on Supercomputers and High-Throughput BLAST on Grid Infrastructures. First EELA-2 Conference, Bogotá, Colombia, 25-27 Febrero 2009 (ISBN 978-84-7834-600-4).

- I. Blanquer, A.A. Carrión, V. Hernández, M. Pignatelli, J. Tamames: Improving the execution of bioinformatics applications by using pilot jobs. 3rd Iberian Grid Infrastructure Conference Proceedings (IBERGRID 2009), Valencia, España, 20-22 Mayo 2009 (ISBN 978-84-9745-406-3).
- LN. Ciuffo, L.Alves, AC Alves de Melo, G.Aparicio, P.Arce, I.Blanquer, DA Burbano, A.Carrion, V.Hernández, JM. Hurtado, et. al.: Biomedical Applications in EELA-2 Project. Network Tools and Applications in Biology (NETTAB) 2009, Ninth annual workshop, Catania, Italia, 10-12 Junio 2009 (ISBN 978-88-7388-242-8).
- I. Blanquer, A.A. Carrión, V. Hernández, J. Forment, V. Conejero: Estimating the horizontal gene transfer between prokaryotes and plants by using e-Science infrastructures. Second EELA-2 Conference, Choroni, Venezuela, 25-27 Noviembre 2009 (ISBN 978-84-7834-627-1).
- I. Blanquer, A.A. Carrión, V. Hernández: Estimating the performance of BLAST runs in the EGEE Grid. 5th EGEE User Forum, Uppsala(Suecia), 12-15 Abril 2010.
- I. Blanquer, A.A. Carrión, V. Hernández: Characterizing Grid experiments in Bioinformatics for an efficient scheduling. VECPAR'10, Berkeley, CA (USA), 22-25 Junio 2010 (Aceptado, pendiente de publicación).

7.2 Trabajos futuros

Tras finalizar el trabajo se puede concluir que hay muchas líneas que se han quedado en el camino y otras muchas que quedan por cubrir. No obstante, los esfuerzos destinados en un futuro próximo deberían estar orientados, principalmente, en dos direcciones.

El primer trabajo futuro a tener en cuenta sería desarrollar un sistema de gestión de trabajos piloto propio, independiente de herramientas externas (como DIANE y Ganga). La implementación este sistema permitiría obtener las mejores prestaciones a la hora de ejecutar experimentos masivos en infraestructuras Grid.

La otra línea de trabajo a tener en cuenta sería caracterizar por completo el ciclo de vida de un trabajo BLAST en el Grid. Los desarrollos de la presente tesis se han centrado en estimar el tiempo de alineamiento requerido por el programa BLAST. Sin embargo, la ejecución de un trabajo BLAST Grid conlleva la realización de otras muchas tareas, como la descarga de la base de datos, que influye claramente en el tiempo de respuesta observado. Por tanto, en un futuro sería interesante introducir en el modelo parámetros de otros componentes de la infraestructura Grid, tales como: el metaplanificador, los planificadores locales de los recursos de cómputo o el ancho de banda entre los elementos de cómputo y de almacenamiento. Con el modelo completo determinado será posible realizar una correspondencia entre tareas y recursos óptima, obtenidas a partir de algoritmos heurísticos.

REFERENCIAS BIBLIOGRÁFICAS

[1] Introduction to Bioinformatics: Teresa K. Attwood, David Parry-Smith. Pearson Education, ISBN 0582327881, 2001.

[2] Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195--197 (1981).

[3] Needleman, Saul B.; and Wunsch, Christian D: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48, 443–53 (1970).

[4] Altschul, S.F., Gish, W., Miller, W., Meyers, E.W., Lipman, D.J.: Basic Local Alignment Search Tool. J. Mol. Biol. 215, 403--410 (1990).

[5] National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov

[6] Bjarne Stroustrup, The C++ Programming Language, Addison-Wesley Pub Co; Tercera edició (15 de febrer de 2000); ISBN 0201700735

[7] B. W. Kernighan, D. M. Ritchie, The C programming language, Prentice-Hall, Inc., Upper Saddle River, NJ, 1978

[8] Extensible Markup Language: http://www.w3.org/XML

[9] mpiBLAST: Open-Source Parallel BLAST home page, http://mpiblast.lanl.gov/

[10] Message Passing Interface Forum: MPI: A message-passing interface standard. (2003) http://www.mpi-forum.org/

[11] Ning, Z., Cox, A.J., Mullikin, J.C.: SSAHA: A fast search method for large DNA databases. Genome Res. 11:1725–1729(2001).

[12] Li, H., Durbin, R.: Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics 25:1754–1760 (2009).

[13] Sun Grid Engine, http://gridengine.sunsource.net.

[14] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor – a distributed job scheduler," in Beowulf Cluster Computing with Linux (T. Sterling, ed.), MIT Press, October 2001.

[15] "Innergrid." http://www.gridsystems.com.

[16] D. Abramson, R. Buyya, and J. Giddy, "A computational economy for grid computing and its implementation in the Nimrod-G resource broker," Future Generation Computer Systems, vol. 18, pp. 1061–1074, Oct. 2002.

[17] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organizations," International Journal of High Performance Computing Applications, vol. 15, no. 3, pp. 200–222, 2001.

[18] "The Globus Alliance." http://www.globus.org.

[19] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," International Journal of Supercomputer Applications and High Performance Computing, vol. 11, no. 2, pp. 115–128, 1997.

[20] T. Sandholm and J. Gawor, "Globus toolkit 3 core - a grid service container framework." http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3 core.pdf.

[21] I. Foster,"Globus toolkit version 4: Software for service-oriented systems,"Network and Parallel Computing, Proceedings, vol. 3779, pp. 2–13, 2005.

[22] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchad, "Web services architecture."W3C Working Draft, 2003.

[23] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10) (I. Press, ed.), 2001.

[24] W3C Consortium, "XML path language (XPath)." http://www.w3.org/TR/xpath, November 1999.

[25] LightWeight Middleware for Grid Computing. http://glite.web.cern.ch/glite

[26] "GLUE schema." http://glueschema.forge.cnaf.infn.it.

[27] "Berkeley Database Information Index."https://twiki.cern.ch/twiki//bin/view/EGEE/BDII.

[28] EGEE: Enabling Grids in e-Science. http://www.eu-eege.org/

[29] EELA: E-Infrastructure shared between Europe and Latin America. http://www.eu-eela.org/

[30] DIANE: Distributed Analysis Environment. Retrieved January 2009, from http://www.cern.ch/diane.

[31] F.Brochu, U.Egede, J.Elmsheuser, K.Harrison, R.W.L.Jones, H.C.Lee, D.Liko,A.Maier, J.T.Moscicki, A.Muraru, G.N.Patrick, K.Pajchel, W.Reece, B.H.Samset, M.W.Slater, A.Soroko, C.L.Tan, D.C.Vanderster. Ganga: a tool for computational-task management and easy access to Grid resources", (February 2009).