Universitat Autònoma de Barcelona

Planificador consciente del almacenamiento para Multiworkflows en Cluster Galaxy

César Esteban Acevedo

Escuela de Ingeniería
Departamento de Arquitectura de Computadores y Sistemas Operativos

Planificador consciente del almacenamiento para Multiworkflows en Cluster Galaxy

Memoria doctoral presentada por **César Esteban Acevedo Giménez** para optar al grado de Doctor por la Universitat Autònoma de Barcelona, bajo la dirección del Dr. Porfidio Hernández Budé y el Dr. Antonio Miguel Espinosa Morales

Cerdanyola del Vallés, Julio 2017

Planificador consciente del almacenamiento para Multiworkflows en Cluster Galaxy

Memoria presentada por **César Esteban Acevedo Giménez** para optar al grado de Doctor por la Universitat Autònoma de Barcelona. Este trabajo ha sido desarrollado en el Departamento de Arquitectura de Computadores y Sistemas Operativos, de la Escuela de Ingeniería dentro del programa de Doctorado en Informática (Computación de Altas Prestaciones) bajo la dirección del **Dr. Porfidio Hernández Budé** y el **Dr. Antonio Miguel Espinosa Morales**.

Dr. Porfidio Hernández Budé (Tutor)

Dr. Antonio Miguel Espinosa Morales

Directores de Tesis

César Esteban Acevedo Giménez

Autor de la Tesis

Cerdanyola del Vallés, Julio 2017

Dedicado a:

Mis hermanas y sobrinos, por el continuo apoyo y afecto.

Muy especialmente a mis padres, por ser ejemplos de perseverancia y superación. Por poner los cimientos de la persona que soy y seré. Por haberme dado todas las herramientas para desarrollarme como un profesional. Por la confianza y el apoyo ilimitado a todos mis emprendimientos.

Y a mi novia, por todo el amor, apoyo y comprensión durante mi prolongada ausencia.

Agradecimientos

Mi agradecimiento especial al Dr. Porfidio Hernández Budé, por su incansable paciencia, consejos y ojo crítico para la valoración y el desarrollo del trabajo realizado. Así como al Dr. Antonio Espinosa por sus continuos aportes. Al Dr. Victor Méndez por su colaboración en distintas etapas del trabajo.

A mis compañeros de CAOS, por hacer que la estancia sea como en casa, en especial a Francisco Borges, Carlos Nuñes, Hugo Meyer, Marcela Castro, Viola Saveta, Tomas Artes, Pilar Gomez, Josefina Lenis, Ronald Muressano, Aprigio Bezerra, Cecilia Jaramillo, Joe Carrion y Manuel Brugnoli†.

Al Prof. Emilio Luque y la Prof. Dolores Rexachs por la inmensa amistad.

Abstract

In the bioinformatic field, experimentation is performed through sequential execution of applications, each application uses as input file the one generated by the previous application. This analysis process consisting of a list of applications describing a dependency chain is called Workflow. Two relevant characteristics of bioinformatic workflows refer to the handling of large volumes of data and the complexity of data dependencies. Many of the current resource managers ignore the location of the files, this implies a high cost if the processing elements are not close to the files and have to be moved. The direct acyclic graph (DAG) model, used to represent the execution order of workflow jobs, does not help to establish the best location of input or temporary data files for efficient execution. The solution to this challenge may be the data-aware scheduling, where an intelligent file placement strategy, added to a resource scheduling according to this knowledge; Will help prevent system downtime caused by the waiting time of data file on processing elements. With the current computing power of clusters, it is possible that multiple workflows to be executed in parallel. In addition, clusters allow multiworkflows to share input and temporal data files in the storage hierarchy. We propose a storage hierarchy composed by the distributed file system, a Local RamDisk, Local Disk and Local Solid State Disk (SSD). In order to solve the assignment of multiworkflows applications to the cluster resources, we extended the multiworkflow heuristic called HEFT (Heterogeneous Earliest Finish Time). This comprises two phases: first a task prioritization phase is performed, and then the processors selection is performed, which consists of assigning the applications to the node that minimizes the execution time of each one of them. The data-aware scheduler considers placing

the files in the storage hierarchy before starting the execution. The data files pre-fetching on the compute nodes makes the applications that use them, can be assigned to the same node as the data files, reducing the access time to disk. To determine the initial location of the input and temporal data files, the scheduler performs the merging of all workflows into a single meta-workflow, then the algorithm sets according to application precedence, file size and sharing degree; The proper storage of each file within the hierarchy. The goal of the research is to implement a multi-workflow data-aware scheduler policy that improves data-intensive applications. To evaluate the scalability of the proposal and to compare it with other policies in the literature, we use simulators. This is a common method for validating scheduling heuristics and saving computation time by looking for the best option. To do this, we extend WorkflowSim by providing it with a data-aware scheduler with a storage hierarchy. Our work was validated, with synthetic workflows, implemented from the characterization of real bioinformatics applications, and workflows benchmark as Montage and Epigenomics because they generate a large amount of temporal files. The experimentation was performed in two scenarios: real cluster system of 128 cores and a simulated cluster in WorkflowSim with up to 1024 cores. In the real scenario, we achieve a *makespan* improvement of up to 70%. In the simulated scenario, the *makespan* improvement was 69% with errors between 0.9% and 3%.

Resum

En l'àmbit bioinformàtic, l'experimentació es realitza a través de seqüències d'execucions d'aplicacions, cada aplicació utilitza com a arxiu d'entrada el generat per l'aplicació anterior. Aquest procés d'anàlisi format per una llista d'aplicacions descrivint una cadena de dependència es diu Workflow. Dues característiques rellevants dels workflows bioinformàtics, fan referència al maneig de grans volums de dades i de la complexitat de les dependències de dades. Molts dels gestors de recursos actuals, ignoren la ubicació dels arxius, això implica un elevat cost si els elements de processament no estan propers als arxius i se'ls ha de moure. El model de graf dirigit acíclic (DAG), utilitzat per representar l'ordre d'execució dels treballs del workflow, no ajuda a establir la millor ubicació dels arxius d'entrada o temporals per a una execució eficient. La solució per a aquest desafiament, pot ser la planificació de recursos conscient de l'emmagatzematge, on una estratègia intelligent de collocació d'arxius, afegida a una planificació de recursos d'acord a aquest coneixement; contribuirà a evitar els períodes d'inactivitat en els sistemes, causats pels temps d'espera d'arxius en els elements de processament. Amb la capacitat de còmput actual dels clústers, és possible que múltiples workflows puguin ser executats en parallel. A més, els clústers permeten que els multiworkflows, puguin compartir els arxius d'entrada i temporals en la jerarquia d'emmagatzematge. Proposem una jerarquia d'emmagatzematge composta pel sistema d'arxius distribuït, un RamDisk Local, Disc Local i Disc d'Estat Sòlid (SSD) Local. A fi de resoldre l'assignació d'aplicacions de multiworkflows als recursos del clúster, vam estendre l'heurística basada en llista per multiworkflows anomenada HEFT (Heterogeneuos Earliest Finish Time). Aquesta comprèn dues fases: primer es realitza una fase de priorització de

tasques, per a posteriorment realitzar la selecció de processadors, que consisteix a assignar les aplicacions al node que minimitza el temps de finalització de cadascuna d'elles. El planificador conscient de l'emmagatzematge proposat, considera ubicar els arxius en la jerarquia d'emmagatzematge abans de començar l'execució. La pre-ubicació d'arxius en els nodes de còmput fa que les aplicacions que les utilitzen, puguin ser assignades al mateix node que els arxius, reduint el temps d'accés a disc. Per determinar la ubicació inicial dels arxius d'entrada i temporals, el planificador realitza la fusió de tots els workflows en un sol meta-workflow, a continuació, l'algoritme estableix segons les precedències d'aplicacions, mida dels arxius i grau de compartició de els mateixos; l'emmagatzematge adequat de cada arxiu dins de la jerarquia. L'objectiu del nostre treball és implementar una nova política de planificació conscient de l'emmagatzematge per multiworkflows que millori el temps de *makespan* d'aplicacions amb còmput intensiu de dades. Per avaluar l'escalabilitat de la política de planificació i a més poder comparar-la amb altres polítiques clàssiques de la literatura, hem utilitzat simuladors. Aquest és un mètode bastant comú per validar heurístiques de planificació i estalviar temps de còmput buscant la millor opció. Per a això, hem extès WorkflowSim i l'hem dotat d'un planificador conscient de la jerarquia d'emmagatzematge. El planificador resultant ha sigut validat en diversos escenaris, amb una càrrega composta per workflows sintètics de bioinformàtica, implementats a partir de la caracterització d'aplicacions bioinformàtiques reals, i workflows de referència àmpliament utilitzats, com Montage i Epigenomics, ja que són workflows que generen una gran quantitat d'arxius temporals. Per validar la nostra proposta hem utilitzat el planificador en dos escenaris: sistemes de clúster real de 128 nuclis i simulador de clúster en WorkflowSim fins a 1024 nuclis. L'escenari real, llanço millores de *makespan* de fins a 70%. A l'escenari simulat, la millora de *makespan* va ser del 69% amb errors entre 0,9% i 3%.

Resumen

En el ámbito bioinformático, la experimentación se realiza a través de secuencias de ejecuciones de aplicaciones, cada aplicación utiliza como archivo de entrada el generado por la aplicación anterior. Este proceso de análisis formado por una lista de aplicaciones describiendo una cadena de dependencia se llama Workflow. Dos características relevantes de los workflows bioinformáticos, hacen referencia al manejo de grandes volúmenes de datos y a la complejidad de las dependencias de datos. Muchos de los gestores de recursos actuales, ignoran la ubicación de los archivos, esto implica un elevado costo si los elementos de procesamiento no están próximos a los archivos y hay que moverlos. El modelo de grafo dirigido acíclico (DAG), utilizado para representar el orden de ejecución de los trabajos del workflow, no ayuda a establecer la mejor ubicación de los archivos de entrada o temporales para una ejecución eficiente. La solución para este desafío, puede ser la planificación de recursos consciente del almacenamiento, donde una estrategia inteligente de colocación de archivos, añadida a una planificación de recursos acorde a este conocimiento; contribuirá a evitar los periodos de inactividad en los sistemas, causados por los tiempos de espera de archivos en los elementos de procesamiento. Con la capacidad de cómputo actual de los clústers, es posible que múltiples workflows puedan ser ejecutados en paralelo. Además, los clústers permiten que los multiworkflows, puedan compartir los archivos de entrada y temporales en la jerarquía de almacenamiento. Proponemos una jerarquía de almacenamiento compuesta por el sistema de archivos distribuido, una RamDisk Local, Disco Local y Disco de Estado Solido (SSD) Local. Con objeto de resolver la asignación de aplicaciones de multiworkflows a los recursos del clúster, extendimos la heurística basada

en lista para multiworkflows llamada HEFT (Heterogeneuos Earliest Finish Time). Esta comprende dos fases: primero se realiza una fase de priorización de tareas, para posteriormente realizar la selección de procesadores, que consiste en asignar las aplicaciones al nodo que minimiza el tiempo de finalización de cada una de ellas. El planificador consciente del almacenamiento propuesto, considera ubicar los archivos en la jerarquía de almacenamiento antes de comenzar la ejecución. La pre-ubicación de archivos en los nodos de cómputo hace que las aplicaciones que las utilizan, puedan ser asignadas al mismo nodo que los archivos, reduciendo el tiempo de acceso a disco. Para determinar la ubicación inicial de los archivos de entrada y temporales, el planificador realiza la fusión de todos los workflows en un solo meta-workflow, a continuación, el algoritmo establece según las precedencias de aplicaciones, tamaño de los archivos y grado de compartición de los mismos; el almacenamiento adecuado de cada archivo dentro de la jerarquía. El objetivo del trabajo es implementar una política de planificación consciente del almacenamiento para multiworkflows que mejore el *makespan* de aplicaciones con cómputo intensivo de datos. Para evaluar la escalabilidad de la propuesta y compararla con otras políticas de la literatura, utilizamos simuladores. Este es un método común para validar heurísticas de planificación y ahorrar tiempo de cómputo buscando la mejor opción. Para ello, extendimos WorkflowSim dotándolo de un planificador consciente de la jerarquía de almacenamiento. El trabajo fue validado, con workflows sintéticos, implementados a partir de la caracterización de aplicaciones bioinformáticas reales, y workflows ampliamente utilizados como Montage y Epigenomics debido a que generan una gran cantidad de archivos temporales. La experimentación se realizó en dos escenarios: sistemas de clúster real de 128 núcleos y simulador de clúster en WorkflowSim hasta 1024 núcleos. El escenario real, arrojo mejoras de *makespan* de hasta 70 %. En el escenario simulado, la mejora de *makespan* fue del 69 % con errores entre 0,9 % y 3 %.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La posibilidad que nos brinda la tecnología de hoy, a la hora de abordar tanto aplicaciones de cómputo científico (simulación de clima, análisis genómico, predicción de terremotos, entre otras), como de aplicaciones distribuidas (buscadores de páginas web, subastas, vídeo bajo demanda, votaciones, predicciones en bolsa, etc.) ha originado la necesidad de más recursos de cómputo, red y almacenamiento por parte de las aplicaciones; y por ende, de una gestión de recursos más eficiente de las plataformas de cómputo destinadas a la ejecución de las mismas.

Al mismo tiempo que las tecnologías de instrumentación generan más y mejores datos (secuenciadores de ADN, aceleradores de partículas, imágenes en HD, etc.), las aplicaciones científicas creadas con el fin de analizar estos datos, en la mayoría de los casos, son invocadas por *scripts* para describir cada una de las etapas de las que se compone el proceso de análisis. El modelo de resolución de problemas aplicado, para representar la complejidad que supone la resolución de las diferentes tareas planteadas, se denomina *workflow* (WF).

Los recientes avances científicos hacen que la generación de grandes cantidades de datos necesiten ser almacenados y analizados de manera más eficiente. El creciente volumen de datos a ser procesados por aplicaciones puede ser observado en casi todas las áreas de conocimiento, como simulación, sensores de datos, procesamiento de genomas e imágenes médicas. Se estima por ejemplo que los programas genómicos y proteómicos pueden generar hasta 100 TBytes de nuevos

datos cada día [43].

La creciente complejidad, a la hora de definir las etapas del workflow en cuanto a localización de datos de entrada / salida, y la reutilización de los *scripts* para reproducir la necesidad de automatizar experimentos, así como la dificultad añadida que supone el manejo de las infraestructuras de cómputo paralelo (clústers, grid y clouds), han acelerado la necesidad de desarrollar *middlewares* y *frameworks* que permitan liberar al usuario de esta complejidad, poniendo a su disposición GUI amigables como *front-end* (Galaxy, Taverna, Pegasus, etc.) que permiten simplificar la creación de WF de aplicaciones científicas, y herramientas automáticas (tolerancia de fallos, ejecución eficiente, paralelización automática, etc).

En una primera instancia, nuestro objetivo inmediato a perseguir consistirá en el diseño e implementación de *middlewares*, gestores de recursos, librerías y herramientas automáticas. Estos elementos permiten liberar al científico de la complejidad que supone la ejecución eficiente de sus aplicaciones y centrarse en lo que es su especialidad, dejando el resto de tareas para su implementación en el *back-end* al experto en computación de altas prestaciones (HPC).

La computación de altas prestaciones tradicional está implementada mediante arquitecturas paralelas del tipo *shared disk*, con nodos de cómputo y nodos de datos diferenciados. Los nodos de cómputo tienen un mínimo de almacenamiento local, además de ser habitual la disponibilidad de Sistemas de Ficheros Distribuidos (como Lustre o GPFS), o protocolos del tipo NFS para la utilización de sistemas de almacenamiento remoto. Los nodos de cómputo se encuentran conectados entre ellos por redes de alta velocidad (Giga Ethernet, Infiniband) y los servidores de almacenamiento por redes de comunicación del tipo NAS o SAN. Aunque los avances tecnológicos permitan redes de conexión cada vez más rápidas, el gran volumen de datos que necesita ser transferido a través de la red de datos por las aplicaciones intensivas de datos, hace que el uso de estas arquitecturas no sea el más adecuado.

Los modelos de programación paralela como MPI y OpenMP, proporcionan un cierto nivel de abstracción sobre las operaciones de comunicación y sincronización. Sin embargo, los programadores aún necesitan abordar muchas de las decisiones de diseño mencionadas anteriormente (tolerancia a fallos, *mapping* y *scheduling*, número de réplicas de los ficheros a procesar, etc). Además, los modelos tradicionales

de programación paralela, se centran en los problemas relacionados a aplicaciones intensivas en CPU y no proporcionan un soporte adecuado para el procesamiento de gran cantidad de datos.

Como solución a estas necesidades, Google propuso utilizar el paradigma MapReduce, centrado en los problemas del procesamiento de grandes volúmenes de datos de entrada. Ante el dilema que se nos plantea de utilizar plataformas de HPC para la ejecución de aplicaciones intensivas de datos (DIC); como objetivo a largo plazo pretendemos recuperar técnicas utilizadas en arquitecturas *shared-nothing*, a efectos de mover las tareas a los datos (incidir en la localidad de los datos), manejo de caches distribuidas, uso de Ramdisk, y en particular en la generación de planificadores que permitan la ejecución sin interferencias y en clústers de workflows con características diferentes (i/o bound, cpu bound, ambos, etc). Nuestro objetivo es hacer efectiva la simbiosis que a nuestro entender se debería producir al ejecutar aplicaciones web y aplicaciones típicas de cómputo de altas prestaciones.

Las aplicaciones científicas utilizan multitud de archivos de entrada, generalmente de gran tamaño, requiriendo un tiempo de cómputo largo y un gran volumen de operaciones de E/S a realizar. Estas aplicaciones que hacen uso de grandes volúmenes de datos son llamadas aplicaciones intensivas de datos.

Para generar descubrimientos de relevancia y/o tomar decisiones importantes, los datos deben ser procesados en un tiempo razonable. Para ello, las aplicaciones intensivas de datos pueden aprovechar los sistemas de computación paralelos y distribuidos mediante la correcta utilización de métodos de balanceo de carga de trabajo, pre-ubicación de archivos de entrada en nodos de cómputo y el uso eficiente de las jerarquías de almacenamiento. Sin embargo, los sistemas de gestión de recursos también necesitan adaptarse a estos requerimientos. Deben conocer dónde se almacenan los archivos de entrada de las aplicaciones, tener la capacidad de mover archivos a los dispositivos de almacenamiento de los nodos de cómputo, y tener políticas de planificación de aplicaciones específicas para sistemas de tipo clúster

En la actualidad, la tecnología de computación de altas prestaciones posee una amplia jerarquía de almacenamiento. Esta jerarquía está compuesta de discos compartidos entre nodos conectados a través de redes de conexión de alta velocidad y los nodos de cómputo cuentan con almacenamiento local. Esta jerarquía permite

que los archivos que van a ser utilizados en varias ocasiones sean almacenados de manera local en el nodo de cómputo y así evitar la posible latencia de red.

El área científica de la bioinformática es un ejemplo clásico donde las aplicaciones han desarrollado características particulares. Aplicaciones como las de transformación de formato, análisis de variantes, mapeo de secuencias, y alineación de secuencias presentan características comunes como:

- Un número importante de archivos de entrada.
- Los archivos de entrada se procesan en su totalidad, típicamente de forma secuencial.
- Existen archivos en común utilizados por aplicaciones similares: normalmente archivos como el genoma de referencia.
- La gran cantidad y el gran tamaño de archivos temporales generados por algunas aplicaciones. Los archivos de entrada pueden generar archivos de salida con tamaños aún mayores.
- Las aplicaciones utilizan como archivos de entrada, archivos que fueron generados por la ejecución previa de otras aplicaciones.

Los análisis de datos en el ámbito científico se realizan a través de cadenas de ejecución de aplicaciones. Cada aplicación tiene un orden de precedencia, determinado por los datos de salida de la aplicación anterior. Estos workflows son creados por profesionales del área científica para resolver un determinado análisis de datos científicos o industriales. Un ejemplo típico en bioinformática sería la búsqueda de variantes en un conjunto de genomas de interés, como en la figura 1.1.

Como se muestra en la figura 1.1, la ejecución de un workflow genómico típico se inicia con Fast2Sanger, una aplicación de transformación de formato de archivos de entrada. A continuación, mediante la ejecución de la aplicación BWA, se realiza una comparación de los nuevos archivos generados con la base de datos de referencia y se alinean respecto a las secuencias del genoma. Posteriormente, SAMTools ajusta la calidad del archivo de salida de la alineación, eliminando redundancias. Por último, la aplicación GatK busca las variantes en la secuencia, y genera un archivo

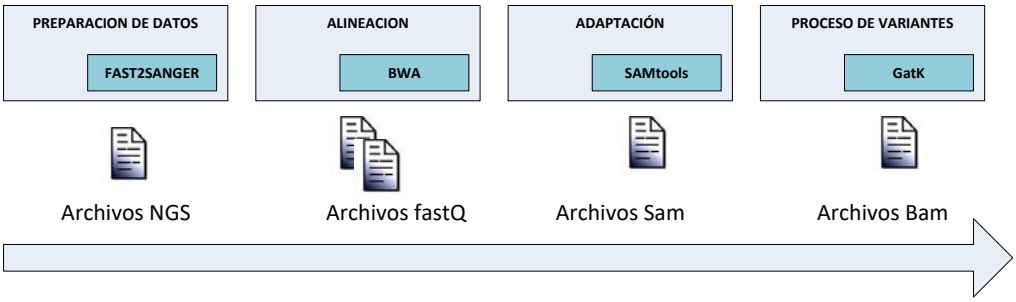


Figura 1.1: Workflow Bioinformático de búsqueda de variantes.

con los resultados para ser revisados por los científicos. En este caso se muestra un patrón secuencial de workflow. No obstante, existen una gran variedad de patrones de dependencias siendo el pipeline y el paralelo dos ejemplos relevantes de ellos.

Los workflows están típicamente escritos en un lenguaje de *scripting* en el que se definen los archivos de entrada, los archivos de salida y las dependencias entre las aplicaciones utilizadas. Estas aplicaciones son creadas con el fin de cumplir con el objetivo de analizar los datos provistos por la tecnología de instrumentación y están en constante desarrollo por la comunidad científica.

La complejidad de la administración de los workflows científicos ha ido creciendo en los últimos años, de forma que se han desarrollado herramientas que permiten compartir workflows por la comunidad científica. Estos gestores de workflows, permiten replicar los resultados de los análisis publicados por científicos en todo el mundo, utilizando las mismas aplicaciones, condiciones y parámetros. En este sentido, se ha popularizado el uso de herramientas de administración de workflows como Galaxy[34], el cual simplifica la compartición de workflows y la reproducibilidad de los resultados obtenidos. La gran mayoría de ellos, como Galaxy pueden ser utilizados tanto en infraestructuras de cómputo locales como distribuidas.

Debido a las peculiaridades de las aplicaciones que componen los workflows bioinformáticos, donde los archivos de entrada y temporales pueden ser compartidos por diferentes aplicaciones, ubicar los archivos en el lugar de la jerarquía de almacenamiento adecuado puede mejorar el rendimiento de las aplicaciones. Uno de los objetivos a lograr a la hora de configurar nuestros gestores, consistirá en

conseguir la localidad a nivel de archivo, asignando los mismos a los nodos donde se encuentran las aplicaciones que los utilizarán, evitamos así que la lectura y escritura de archivos compartidos sobrecargue la red y el acceso a disco. Un análisis de los archivos de entrada asociados a las aplicaciones a ejecutar, en relación a su tamaño, grado de compartición de los archivos entre aplicaciones y situación actual de los archivos, deberá ser tenido en cuenta a la hora de determinar el lugar más adecuado para ubicar los archivos de entrada, salida y temporales dentro de la jerarquía de almacenamiento.

Del abanico de posibilidades de entornos de cómputo hemos decidido utilizar sistemas clúster. Los sistemas clúster son un conjunto de computadoras construidas mediante la utilización de componentes de hardware que se comportan como si fuesen una única computadora [19]. Los sistemas clúster de altas prestaciones van adquiriendo cada vez más capacidad de cómputo y almacenamiento, y suelen estar disponibles en la mayoría de los centros de computación para el análisis de datos científicos.

1.1 Entornos de cómputo

En esta sección, situamos los clústers dentro de los distintos tipos de cómputo paralelo habitual, al ser el entorno elegido para la experimentación del trabajo. La caracterización de dichos entornos nos permitirá fijar aspectos críticos de los mismos, en relación al planificador que será más adecuado.

El problema de asignación de tareas o trabajos a los nodos de procesamiento, con objeto de mejorar una figura de mérito, es y ha sido un problema abierto y ampliamente debatido en la comunidad científica. Un tópico de los considerados clásicos a resolver que ha ido reformulándose en el tiempo, intentando adaptarse tanto a los nuevos tipos de cargas a ejecutar, como a los nuevos entornos de cómputo distribuido.

La relevancia del planificador de aplicaciones en un entorno de cómputo paralelo, se debe principalmente a la capacidad de administrar los recursos disponibles de manera eficiente para que las aplicaciones en el sistema cumplan con

los requisitos de tiempo de respuesta, *makespan*, *turnaround* y otras métricas tanto del sistema como del usuario. En nuestro trabajo tendremos en cuenta como métrica el *makespan* global de los workflows, debido a que buscamos ejecutar en el menor tiempo posible múltiples grupos de workflows, posiblemente de varios usuarios que conviven en el mismo sistema de cómputo.

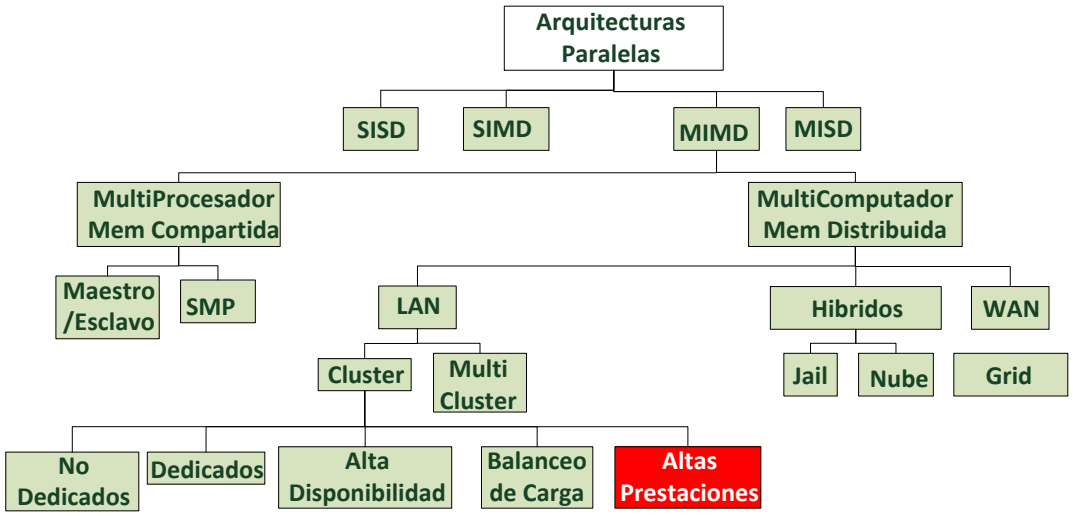


Figura 1.2: Taxonomía de arquitecturas de cómputo paralelo.

La figura 1.2 muestra una de las posibles taxonomías en lo referente a las posibles arquitecturas de cómputo paralelo. Se han utilizado las de Flynn [30] y Tanenbaum [69] como base para su generación.

A partir de las arquitecturas MIMD, los dos grupos que se generan obedecen a la existencia o no de memoria compartida para resolver los problemas de comunicación y/o sincronización. Siendo los multiprocesadores, mediante memoria compartida; o los multicomputadores mediante el paso de mensajes; las dos ramas del árbol desde donde cuelgan las posibles alternativas. Llegados a este punto, la predecibilidad del ancho de banda de la red, determina un nuevo punto de ruptura en la taxonomía que se describe; las redes de área local, predecibles donde ubicamos los clústers, frente a las redes de área remota donde situamos los sistemas Grid; marcan un nuevo punto de inflexión en la descripción que nos ocupa.

Una de las clasificaciones mayoritariamente aceptada en relación a los diferentes tipos de clústers es la que se muestra a continuación. Los clústers de tipo dedicado, tienen como característica más relevante la sintonización de la infraestructura hardware y software a una sola aplicación (Beowulf son un claro ejemplo). Por otro lado, los clústers del tipo no dedicado, permiten la ejecución de aplicaciones que comparten recursos entre ellas, o permiten la ejecución de aplicaciones serie o paralelas de otros usuarios cediendo parte de su potencia de cómputo (sistemas de cómputo voluntario, o redes de estaciones de trabajo son claros ejemplos). Además, también existen los sistemas clúster de alta disponibilidad para aplicaciones críticas, caracterizados por poner el foco en la tolerancia a fallos mediante la utilización de redundancia a todos los niveles. Siguen en la taxonomía los clúster de balanceo de carga, cuyo criterio de prestaciones se centra en distribuir la carga de trabajo de manera equitativa entre los diferentes nodos. Finalmente, consideramos los clústers de altas prestaciones, orientados a la ejecución eficiente de aplicaciones paralelas, como cierre de nuestra clasificación y nos sitúa en el entorno motivo del presente estudio.

Por lo tanto al observar la figura 1.3, la tecnología actual con mayor uso es el de sistemas clúster, de los sistemas de cómputo del top500 desde el año 1993 y su evolución hasta el 2016.

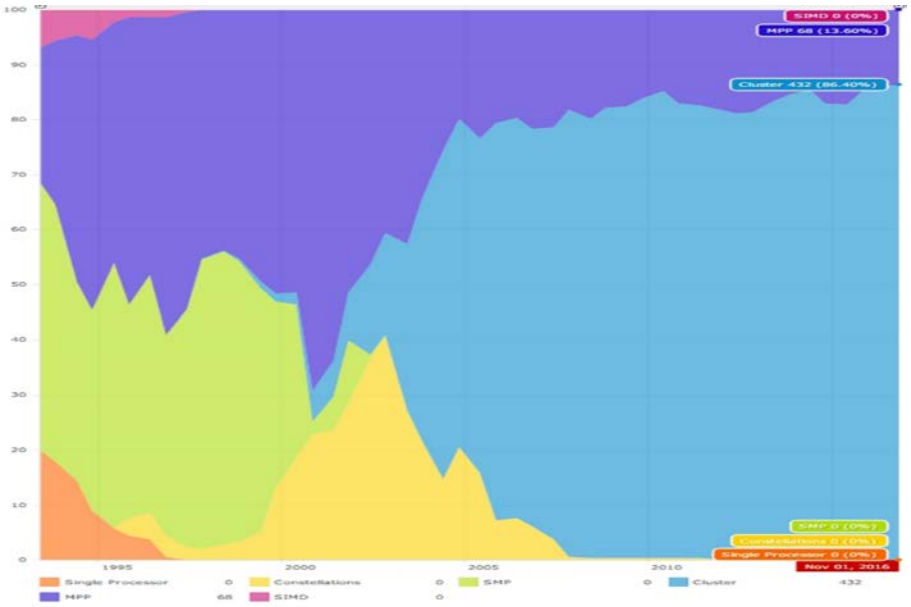
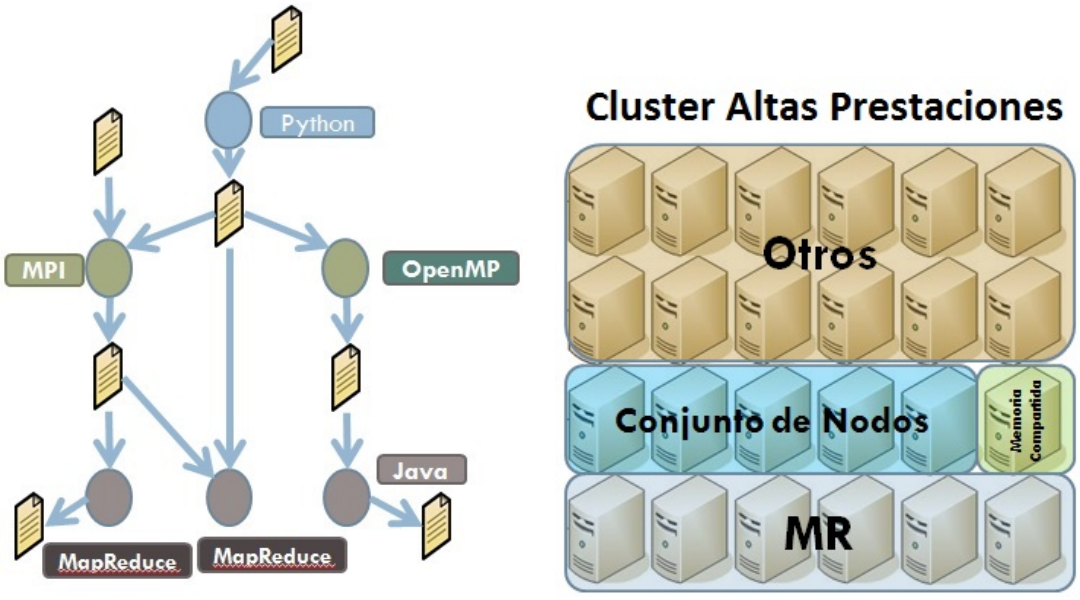


Figura 1.3: Porcentaje de uso de tecnologías de cómputo en el TOP500[70]

Debido a que los clústers actuales poseen cada vez más capacidad de cómputo, uno de los problemas a resolver en la actualidad reside en encontrar la manera adecuada de particionarlos para poder así ejecutar aplicaciones desarrolladas bajo distintos paradigmas de programación y de esta forma, obtener la mayor eficiencia de estos sistemas.

El constante desarrollo de aplicaciones que coexisten en el sistema genera nuevos tipos de requerimientos como tiempo de*turnaround*, calidad de servicio, *throughput* o tiempo de espera desde que la aplicación esta lista para ejecutarse hasta que efectivamente inicia su ejecución. Cada una de estas aplicaciones tiene necesidades propias de memoria, CPU, ancho de banda de red y almacenamiento en disco. En muchos casos estas aplicaciones pueden aprovechar recursos compartidos en un clúster. Lo más habitual sería utilizar aplicaciones con la misma magnitud de tiempo de ejecución y así evitar un tiempo de turnaround o tiempo de espera excesivo.

La utilización de planificadores de aplicaciones nos permite por lo tanto que workflows con aplicaciones de distintos paradigmas 1.4.a coexistan en un mismo sistema como el que podemos ver en la figura 1.4.b.



(a) Paradigmas de Aplicaciones del workflow (b) Clúster No Dedicado

Figura 1.4: Paradigmas de Aplicaciones de Workflows planificados en Clústers No-Dedicados.

El problema que encontramos en estos sistemas de clúster compartido, es el de poder determinar la cantidad de recursos que asignaremos a cada paradigma. Es el administrador del sistema, quien establece el conjunto de particiones del clúster, y el conjunto de recursos asociados a cada partición, y que se deberán utilizar para cada caso en particular. En el caso de aplicaciones programadas mediante la utilización del paradigma MapReduce, se puede definir una cola del gestor de recursos dedicada exclusivamente para los nodos con *framework* Hadoop instalado y a utilizar si se desea de modo exclusivo, por este tipo de trabajos.

En esta memoria presentamos una política de planificaciones de multiworkflows bioinformáticos y de tipo genérico basados en aplicaciones intensivas de datos que comparten archivos de entrada y temporales, ubicadas en una jerarquía de almacenamiento clásica de nodos de clúster. En las siguientes secciones 1.2,1.3,1.4 detallamos las distintas alternativas existentes de planificación.

1.2 Políticas de planificación de Workflows

En el ámbito bioinformático, los workflows constituyen un modelo clásico para expresar la solución a un problema complejo, debido a que se suele dividir el problema en etapas parciales realizadas por las aplicaciones que intervienen. Éstas están relacionadas formando cadenas de dependencias donde se envían y reciben archivos de entrada y archivos temporales generados por las aplicaciones en cada una de las etapas.

A pesar de su amplio uso para representar las etapas del workflow, los grafos dirigidos acíclicos (DAGs) carecen de información relevante sobre la localización de los archivos pertenecientes a las aplicaciones. Es decir, no muestran ningún detalle sobre dónde están ubicados inicialmente los archivos de entrada o temporales, y cómo se transferirán a los nodos de cómputo donde se ejecutarán las aplicaciones.

La figura 1.5 muestra un ejemplo de como el DAG que representa el workflow carece de información sobre la ubicación de los archivos de entrada y salida. Esta información sobre la ubicación es necesaria para encontrar el mejor lugar en la jerarquía de almacenamiento donde situar la información. Si la jerarquía

de almacenamiento está constituida de 3 niveles donde alojar los archivos, en el ejemplo se observan las posibles ubicaciones de los archivos en una jerarquía de almacenamiento típica de una arquitectura de clúster, así como también las posibles asignaciones de las aplicaciones a los nodos de cómputo. A la hora de decidir cuál es el medio más adecuado para almacenar los diferentes archivos relacionados con los workflow, hay que considerar varias características del sistema de almacenamiento como: las diferencias de velocidades de acceso en lectura y escritura de los diferentes dispositivos y la capacidad de almacenamiento de cada nivel de la jerarquía.

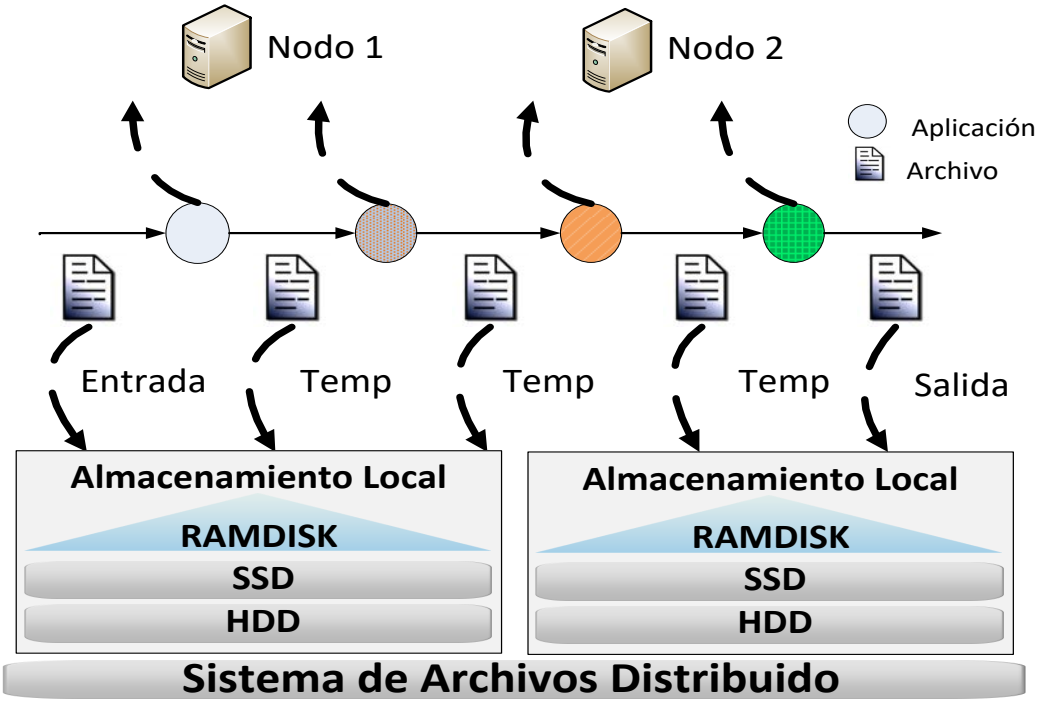


Figura 1.5: Grafo de Aplicaciones con sus relaciones según localización de datos

Los nodos de cómputo del clúster tienen su propia jerarquía local de almacenamiento. Se pueden describir como dispositivos de almacenamiento de propósito general con tiempos de acceso elevados. Habitualmente los nodos suelen estar conectados a un sistema de archivos distribuido a través de una red de interconexión de altas prestaciones. Desde el punto de vista del manejo de datos, las etapas del workflow necesitan administrar los archivos de entrada, salida y de uso temporal [25]. Se convierte así en un desafío poder determinar cuál es la mejor

ubicación de los archivos necesarios para las diferentes aplicaciones y así obtener un mejor *makespan*.

La planificación en clústers se reduce a resolver un problema en dos niveles: encontrar el conjunto de recursos más adecuado (planificación espacial) y establecer el orden de ejecución en cada uno de los nodos del clúster, dado que el grado de multiprogramación en clústers compartidos será mayor que uno habitualmente exclusivo (planificación temporal).

Los planificadores son los encargados de garantizar que las aplicaciones cuenten con los recursos necesarios para su ejecución. Sin embargo, no siempre estarán todos los recursos disponibles y por tanto, se deben administrar los recursos de cómputo de manera eficiente para garantizar la disponibilidad de los mismos en un tiempo razonable.

La figura 1.6 representa una taxonomía que recoge las posibles alternativas a la hora de diseñar planificadores. Nos apoyamos en esta taxonomía para situar nuestro planificador dentro de lo que sería el estado del arte actualmente.

En una primera fase, analizamos la problemática y alternativas derivadas de la planificación para workflows, multiworkflows y workflows conscientes del almacenamiento, a efectos de posicionar el nuevo planificador propuesto dentro de las posibles opciones.

La problemática de planificación de workflows cuando se formula sin restricciones, es un clásico en la teoría de planificación y se ha demostrado que es del tipo NP-Completo[7].

Existen estudios sobre el diseño y las alternativas de creación de planificadores para un único workflow, específicamente al intentar asignar un trabajo en dominios heterogéneos como lo describe Topcuoglu[71]. Kwok [45], por su parte describe una taxonomía que clasifica los algoritmos estáticos de acuerdo a las funcionalidades. El método de agrupamiento reduce los costes de comunicación mediante la definición de grupos de aplicaciones con alto coste de comunicación en un sistema clúster [79]. La heurística de duplicación [56] proporciona un método de transmisión de archivos en duplicados a nodos de cómputo, y así incrementar la probabilidad de ejecución de aplicaciones en distintos nodos reduciendo el tiempo de espera. El método basado en

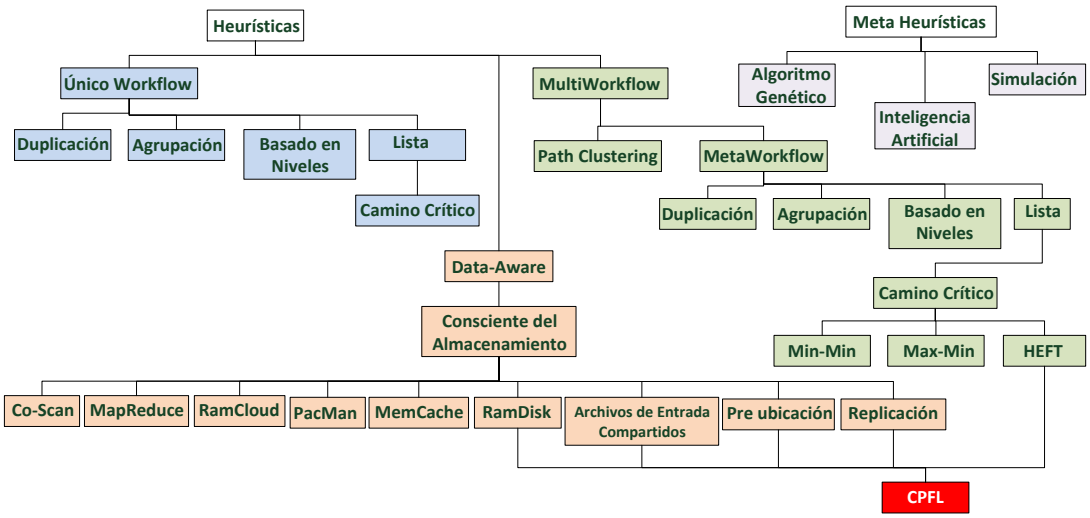


Figura 1.6: Taxonomía de Planificadores para Workflows

niveles, primero divide el workflow en aplicaciones independientes y seguidamente aplica la asignación a los recursos [49] de modo que aumenta el paralelismo de las aplicaciones.

La planificación basada en Lista [37] utiliza heurísticas de baja complejidad para entornos heterogéneos. Bolze et. al [16] propone como alternativa minimizar el *makespan* del workflow, proporcionando una lista ordenada de aplicaciones de acuerdo a prioridades, calculadas según el camino crítico [71] considerando el tiempo de cómputo y coste de comunicación.

En este trabajo, consideramos una generalización del problema de planificación de aplicaciones de workflows científicos. Para este caso podemos encontrar muchos algoritmos basados en heurística de lista para DAGs. Algunas de las propuestas más relevantes son: Predictive Earliest Finish Time (PEFT) [6] la cual utiliza una tabla optimista para definir prioridades, Lookahead [14] y Heterogeneous Earliest Finish Time (HEFT) [71]. La heurística HEFT propone seleccionar la aplicación con la prioridad más alta en cada nivel del grafo. El algoritmo calcula primero, para cada aplicación su peso, utilizando el tiempo de cómputo y el coste de comunicación. El peso calculado es seguidamente utilizado en el camino crítico para determinar la prioridad en el orden de ejecución del workflow. A continuación, se asigna aplicaciones en el orden de precedencia definida en el workflow a los

procesadores. Se utiliza la lista de prioridad en caso de empate en la precedencia del workflow. El algoritmo HEFT es una de las propuestas más utilizadas, debido a su robustez y baja complejidad. Su complejidad temporal es $O(n^2 * m)$ donde n es el número de trabajos y m es el número de procesadores. Aunque HEFT es un algoritmo para entornos heterogéneos, lo utilizamos en este trabajo con sistemas clúster homogéneos pero con la línea de investigación abierta a la posibilidad de utilizar nodos de cómputo heterogéneos.

1.3 Políticas de planificación de múltiples Workflows

Con el aumento de la capacidad de cómputo disponible en los servidores actuales, la ejecución de un solo workflow puede no utilizar todos los recursos del sistema. Esto permite plantearse la ejecución simultánea de múltiples workflows dado que hay recursos disponibles. A partir de aquí, vamos a evaluar las soluciones clásicas de planificación para la ejecución de múltiples workflows.

Definir qué aplicación de los múltiples workflows se ejecutará en un nodo de cómputo específico del sistema clúster se ha descrito en varios trabajos como [68], que proponen encapsular aplicaciones para que puedan ser asignadas a los recursos. Por otro lado, existen propuestas de planificación dinámica. Yu et al. [81] propone un planificador basado en un contenedor de aplicaciones listas para ser ejecutadas que van siendo seleccionadas de acuerdo al planificador HEFT. Otros estudios buscan seleccionar la cantidad adecuada de recursos para aplicaciones de workflows [53]. Así también, existen otras propuestas como [58] que utilizan el análisis de camino crítico como base para un planificador dinámico.

La planificación de múltiples workflows ha sido estudiada ampliamente [80]. La mayoría de las propuestas aplican un principio similar: transformar múltiples workflows en uno solo, conectando un número de nodos ficticios en ambos extremos del grafo. Seguidamente, se aplican diferentes heurísticas para planificar un grupo de aplicaciones del workflow.

Zhao et al. [82] propone una composición de workflows como la descrita

anteriormente, con el objetivo de combinar múltiples DAGs en un solo grafo, antes de aplicar algoritmos estáticos, teniendo en cuenta que todos los parámetros representativos de los diferentes grafos son conocidos de antemano. Del mismo modo, Hönig et al. [36] describen un meta-planificador para múltiples DAGs el cual implica la fusión de múltiples DAG en un solo grafo para minimizar el tiempo de inactividad de los recursos y mejorar el paralelismo global. Estos estudios están enfocados al caso estático y no tienen en cuenta las cargas de trabajo dinámicas.

Es muy común encontrar heurísticas basadas en lista para resolver problemas de planificación de múltiples workflows. Estas heurísticas mantienen una lista de todas las aplicaciones de un DAG específico de acuerdo a sus prioridades. Cada aplicación del grafo tiene una prioridad. Primero, se crea una lista de aplicaciones considerando un orden decreciente de prioridad. Por último, siguiendo el orden de prioridades creado y teniendo en cuenta la precedencia dada por el grafo, se asigna la aplicación con mayor prioridad a un recurso de cómputo.

Muchos estudios como [15] [17] y [21], muestran que la estrategia estática puede obtener resultados casi óptimos para workflows reales y sintéticos [76]. Existen trabajos de simulación [15] que sugieren que los algoritmos estáticos proporcionan un buen resultado para workflows de aplicaciones intensivas de datos, incluso cuando no se dispone de la información de los siguientes trabajos de forma completa.

Los workflows bioinformáticos, por lo general, hacen uso de aplicaciones que comparten archivos de entrada. Nuestra propuesta extiende la heurística basada en listas (HEFT) adaptando su uso al contexto multiworkflow, fusionando workflows en un único meta-workflow, y añadiendo información sobre el almacenamiento de los archivos.

1.4 Políticas de planificación consciente del almacenamiento para Workflows

Las investigaciones sobre estrategias conscientes del almacenamiento estudian las ubicaciones de los archivos en las jerarquías de almacenamiento como criterio relevante para la planificación de las aplicaciones.

Pocos trabajos anteriores han considerado la localidad de los archivos de entrada, de salida y temporales en sistemas de clúster compartidos a pesar de que la ubicación de los archivos ha demostrado ser un criterio crítico en la gestión de workflows de aplicaciones intensivas de datos[59] [60]. El método utilizado para determinar el rendimiento de las propuestas presentadas es reducir el tiempo de ejecución del workflow al mínimo así como el *makespan*.

La estrategia de ubicación de archivos en una jerarquía de almacenamiento se ha utilizado previamente para reducir la carga de E/S de sistemas en la Nube [74] y sistemas Grid. En estos sistemas, cuando los archivos compartidos se utilizan varias veces, el rendimiento depende de la capacidad de la red de interconexión. En estos casos, los archivos compartidos se encuentran en nodos específicos de almacenamiento para minimizar al máximo las transferencias.

En un entorno clúster con una red de altas prestaciones, el disco se convierte en el recurso limitante, lo que genera tiempos de espera de E/S cuando muchas aplicaciones solicitan al mismo tiempo el acceso a los archivos. En los sistemas de altas prestaciones, el rendimiento de E/S ha sido estudiado por Wickberg [75] que propone el uso de Ramdisk como un sistema de almacenamiento con técnicas de localización de archivos en una sección de la memoria principal del sistema.

La relevancia del almacenamiento de datos para el planificación ya ha sido introducida por Bent et al. [9] al presentar el sistema de archivos distribuido por lotes (BAD-FS). Su objetivo es proponer que el sistema de archivos puede exportar algunas características de la arquitectura y comportamiento que podrían ser consideradas al diseñar el planificador. Además, [64] introduce el concepto de una herramienta de indexación, se centra en la distribución de datos para aplicaciones de uso intensivo de datos sobre un sistema de archivos distribuido. El enfoque de SmallClient, se refiere a la idea de dividir los archivos en bloques más pequeños, almacenados en un formato de clave-valor en un sistema de archivos distribuido.

Nosotros sin embargo, no necesitamos dividir los archivos, pero necesitamos recuperar los archivos completos para utilizarlos varias veces en caso de archivos compartidos de entrada y temporales. En ese caso, incluso cuando trabajamos en una red de altas prestaciones queremos evitar la necesidad de leer repetidamente desde un sistema de archivos distribuido. Buscamos aprovechar el archivo que ya

pre-ubicamos en la memoria principal usando una RamDisk, y enviar las aplicaciones que comparten el archivo para que sean ejecutadas en el mismo nodo de cómputo donde está ubicado el archivo. Debido a la imposibilidad de tener una RamDisk ilimitada, introducimos el uso de niveles de almacenamiento jerárquico, con el sistema de archivos distribuido, disco local y RamDisk local, que SmallClient y otros planificadores no tienen en cuenta.

Las transferencias de archivos entre nodos de un sistema pueden convertirse en la carga principal cuando la red de interconexión se congestiona. Stork [42], describe un planificador que enfoca el problema de transferir k archivos de m orígenes a n destinos. Stork se centra en encontrar los mejores protocolos, los parámetros para afinar las transferencias y el orden de las solicitudes de transferencia, entre otros. En nuestro caso, nuestro factor limitante es el disco compartido, y no la red de interconexión, por lo que necesitamos estudiar técnicas alternativas para mantener los datos cerca de los nodos de cómputo en sistemas de jerarquía de almacenamiento rápidos.

Una de las características de las aplicaciones que estudiamos es que suelen procesar los archivos temporales de entrada y salida desde el principio hasta el final de cada archivo. Además, algunos de esos archivos son compartidos por varias aplicaciones, por lo que se leen varias veces durante la ejecución por más de una aplicación. Hacer un buen uso de la jerarquía de almacenamiento para almacenar estos archivos compartidos y ejecutar aplicaciones en nodos donde ya se encuentran almacenados dichos archivos mejora el *makespan* de los workflows al reducir la latencia de acceso a los datos.

En un escenario similar, existen modelos como [22] que evalúan el rendimiento en sistemas en la Nube y clústers que poseen su propio sistema de almacenamiento. En ese caso, se utiliza un procedimiento de copia de seguridad del almacenamiento global, para comparar el tiempo de ejecución entre sistemas en la Nube y de clúster cuando ambos sistemas procesan la misma cantidad de solicitudes de archivos. Dicho estudio establece la relevancia de gestionar adecuadamente las solicitudes de ejecución de aplicaciones cuando la red es el recurso limitante. En contraste, nuestro objetivo es beneficiarse de los patrones de localidad comúnmente mostrados por las aplicaciones científicas en estudio. En este sentido, nuestro recurso limitante

es el disco y necesitamos optimizar el acceso a memoria de archivos compartidos para mejorar el rendimiento de las aplicaciones.

De esta forma, buscamos adaptar el planificador consciente del almacenamiento para multiworkflows a entornos clúster. Seleccionamos una heurística de planificación basada en lista con camino crítico, implementando un modelo de meta-workflow abstracto. Suministramos a nuestro planificador información sobre el tiempo de cómputo, la ubicación de los datos y el tamaño de los archivos de entrada compartidos para reducir la latencia de acceso a la jerarquía de almacenamiento.

Debido a las múltiples posibilidades de ubicación de la gran cantidad de archivos utilizados, buscamos un método ágil para seleccionar las mejores políticas de ubicación. Una de las más comunes es utilizar un simulador para explorar las combinaciones de asignación antes de aplicar la mejor política de localización en la jerarquía de almacenamiento real del clúster.

1.5 Simuladores de Workflows

Dado que las infraestructuras, sistemas y aplicaciones son cada vez más complejas; y que su comportamiento es difícil de reproducir en un sistema en producción, una parte de nuestro trabajo, se centrará en el uso de la simulación como herramienta de análisis y de generación de nuevas propuestas. La simulación dirigida por eventos, ha sido uno de las vías utilizadas tradicionalmente, para ayudar en el diseño y evaluación de gestores de recursos. Para la planificación de tareas por lotes, algunos planificadores disponen de modos de simulación, como SLURM y Cobalt. Chen y Deelman propusieron WorkflowSim[23], una versión extendida de CloudSim[20] que soporta la gestión de workflows y su planificación.

El núcleo central del trabajo a desarrollar será explorar diferentes técnicas de planificación consciente de los datos, así como presentar propuestas de gestión de recursos para workflows bioinformáticos intensivos de datos.

En nuestro trabajo, utilizaremos simuladores para decidir qué políticas de planificación de aplicaciones y archivos es la adecuada, así como determinar la

escalabilidad de las soluciones de planificación propuestas. En la misma línea, el uso de la simulación nos permitirá contrastar los resultados de nuestras aportaciones, con otras políticas de planificación típicas, referidas en la literatura.

Existen varios simuladores de ejecución de aplicaciones, y en la figura 1.7 presentamos una taxonomía de los principales.

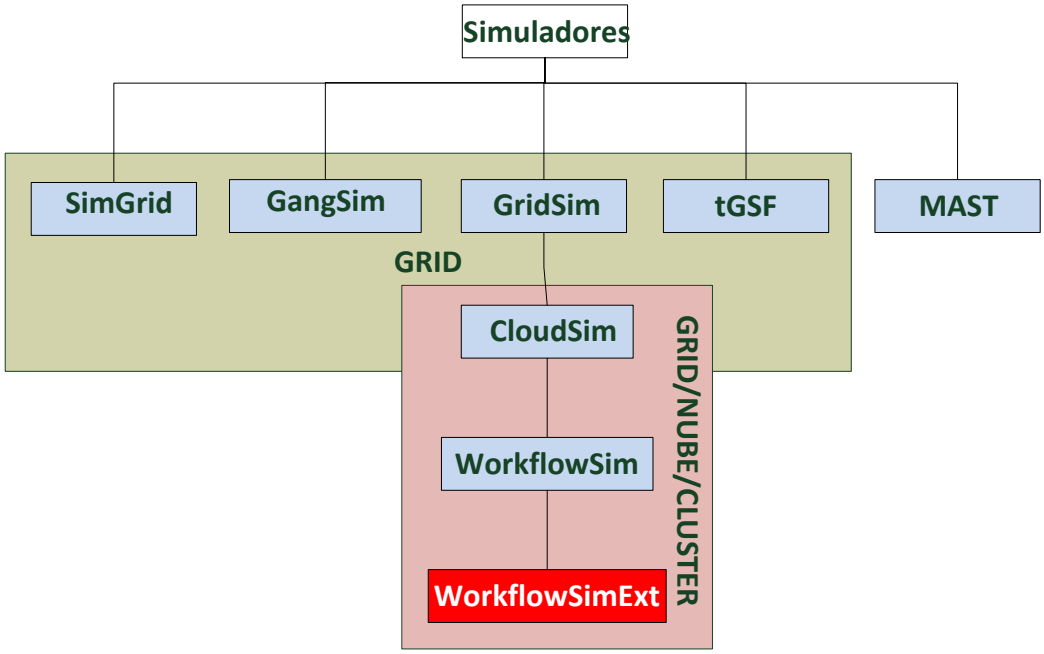


Figura 1.7: Taxonomía de Simuladores para Workflows

Existen simuladores para distintos sistemas y trabajos. Entre los simuladores encontramos que *Manufacturing Simulation Tool Agent* (MAST) [51], está basado en la interacción multi-agente, donde cada agente realiza la programación local de los recursos usando reglas de *dispatching*. Además, [35] propone tGSF como simulador para planificar workflows. Ambos simuladores fueron diseñados específicamente para analizar algunos aspectos del planificador de workflows considerando infraestructuras Grid.

CloudSim [20], es un simulador que modela cargas de trabajo únicas en infraestructuras de cómputo en la Nube. CloudSim carece del concepto de dependencias y no considera los costes generales en las configuraciones de

infraestructuras específicas.

WorkflowSim [23] extiende CloudSim para implementar dependencias entre aplicaciones como las existentes en los workflows. Esto nos permite reproducir la carga a nivel de archivos correspondiente a una jerarquía de almacenamiento típica de un clúster, diferenciándola de un sistema en la Nube o Grid. En cualquier caso, WorkflowSim carece de un planificador que considere como extensión la ubicación de los archivos en la jerarquía de almacenamiento del sistema. Es nuestro interés extender su funcionalidad en este sentido, con objeto de simular el efecto de considerar el uso de sistemas de almacenamiento rápido como RamDisk o SSD.

Hemos decidido utilizar WorkflowSim, debido a que permite simular ejecuciones de aplicaciones en clústers y administrar las dependencias de workflows. A su vez, incorpora planificadores clásicos como HEFT, Min-Min, Max-Min, FCFS y Random. Sin embargo, originalmente no contaba con ningún planificador consciente del almacenamiento para ser utilizado en el simulador. Para mejorar WorkflowSim, agregamos la capacidad de utilizar ubicaciones de archivos en niveles de la jerarquía de almacenamiento, como Ramdisk y SSD a efectos de utilizarlos junto con NFS y el disco duro local. Con las extensiones realizadas en WorkflowSim, podemos implementar otros nuevos planificadores o adaptar los existentes a la nueva situación.

WorkflowSim nos ha permitido comparar nuestro planificador consciente del almacenamiento, con planificadores clásicos como HEFT, Min-Min, Max-Min, FCFS y Random para aplicaciones con archivos de entrada compartidos con tamaños de 2048MB, 1024MB y 512MB y hasta 1024 núcleos. Algunos de los *bugs* encontrados en el código, así como restricciones encontradas en nuestros nodos de cómputo, no nos han permitido realizar análisis con un número mayor de núcleos.

1.6 Sistemas de Administración de Workflows Científicos

Existen varias plataformas que tienen como fin simplificar la creación de workflows de aplicaciones científicas como Taverna [54], Kepler [47], Pegasus [29] y Ergatis [55] que utiliza Sun Grid Engine (SGE)[32] y sistemas de archivos compartidos.

Además de UUPMAX que se ejecuta con SLURM[66]. Finalmente, Galaxy[33] se ha posicionado como uno de los referentes en este tipo de herramienta de administración de workflow, proporciona entorno web y aplicaciones pre-compiladas, puede ser instalado en un entorno local, servidores en la nube, clúster o grids.

Diseñar workflows implica definir cada una de las etapas que lo componen con los parámetros necesarios para cada aplicación que realiza cada etapa. El proceso de definir las cadenas de ejecución presenta algunas limitaciones como las que citamos a continuación:

- Dificultad para extraer paralelismo de los workflows que encadenan aplicaciones secuenciales. Las principales limitaciones de este tipo de composición son tener que lidiar con la forma en que los archivos van de una aplicación a la otra y el diseñador del workflow típicamente debe dedicarse a asignar carpetas, archivos y permisos para preparar el entorno de ejecución de la próxima aplicación en la secuencia.

- Es difícil establecer el paralelismo de aplicaciones no dependientes. Cada operación debe ser descrita en términos de qué archivos se están utilizando y dónde ubicarlos. Un problema clásico es el tener que definir los archivos de entrada, escrituras concurrentes e independencias de datos entre distintas aplicaciones.

- Las aplicaciones desarrolladas en nuevos paradigmas requieren la re-programación de workflows complejos. Las aplicaciones que proponen un nuevo modelo de programación como GATK o cualquier aplicación basada en Java como Hadoop obligan a dar soporte a aplicaciones difíciles de mantener.

- Los workflows visuales pueden no ser fácilmente escalables cuando se implementan en tecnologías web. Las herramientas como Taverna son buenas para descubrir y agregar servicios WSDL a un proceso, pero enviar grandes conjuntos de datos a servicios web remotos es demasiado costoso.

- Herramientas de administración como Galaxy no proveen métodos para administrar datos distribuidos para ejecutar workflows.

Las herramientas de creación de workflows científicos descritas ayudan a simplificar la tarea de diseño de experimentos de análisis de datos. Elegimos Galaxy, el cual originalmente hasta la versión 13.0 no contaba con características suficientes para nuestro estudio, como ejecución paralela de aplicaciones de workflows, ejecuciones de multiworkflows con soporte multiusuario, administración de archivos distribuidos, la escalabilidad de aplicaciones multicore y los permisos de seguridad correspondientes. De esta forma, hemos extendido la plataforma de gestión de workflows basada en Galaxy 13.0. Esto nos permitió ejecutar múltiples workflows en un clúster, independientemente del gestor de recursos como SLURM [66], o SGE [32] y con sistemas de ficheros como Lustre [48] y NFS3 [50] y paradigmas de programación como Hadoop [12].

Otro de nuestros objetivos es resolver la distribución de archivos usados por los workflows desde el entorno Galaxy, que inicialmente sólo consideraba el uso del sistema de archivos distribuido para aplicaciones de uso intensivo de datos. Galaxy, con la extensión de nuestra propuesta, es capaz de proporcionar una política de planificación para experimentos científicos aprovechando la potencia de cómputo de un clúster. Inicialmente Galaxy sólo soportaba la ejecución por medio de módulos de ejecución que se encargaban de enviar a una cola de trabajo las aplicaciones definidas por el orden de precedencia del workflow. No se tenía en cuenta ninguna política que no fuera la utilizada por el Administrador de Recursos Distribuido (DRM) del clúster. Algunas otras herramientas de administración de workflows, están desarrollando políticas de planificación integradas, pero insertan una capa extra de dificultad para el usuario final.

1.7 Objetivos

En este trabajo se estudian las características de las aplicaciones que comparten un mismo conjunto de archivos de entrada en sistemas clúster, y el factor añadido provocado por las aplicaciones que generan una gran cantidad de archivos temporales. Se propone el diseño e implementación de una solución, que gestione de manera eficiente la planificación de las aplicaciones que procesan los mismos archivos de entrada y archivos temporales, aprovechando los recursos disponibles en

los distintos niveles de la jerarquía de almacenamiento.

Hemos extendido una nueva heurística de planificación basada en lista, para implementar una política consciente del almacenamiento para múltiples workflows bioinformáticos en clústers, denominada *Critical Path File Location* (CPFL), que puede sustituir las políticas de planificación de workflows tradicionales sin necesidad de cambios en el diseño del workflow.

La propuesta Critical Path File Location tiene dos enfoques. Por un lado, realizar la planificación de los workflows que comparten el mismo conjunto de archivos de entrada, a través de la asignación de las aplicaciones que procesan los archivos compartidos a un mismo nodo. Por otro lado, gestionar la mejor ubicación de los archivos de entrada y temporales compartidos en una jerarquía de almacenamiento de propósito general con distintos niveles en términos de latencia y capacidad como la RamDisk, discos y SSD locales.

Por otra parte, la gestión de las aplicaciones que comparten el mismo conjunto de archivos de entrada, está estructurada en 2 niveles. En el primer nivel, la política agrupa los workflows que comparten los mismos archivos de entrada y los procesa por grupos. En el segundo nivel, las aplicaciones de los diferentes workflows procesados en el mismo grupo, y que utilizan el mismo archivo de entrada, son a su vez agrupadas para ser ejecutadas en el mismo nodo. Este enfoque permite aumentar la localidad de archivos de entrada, lo que propicia un menor volumen de archivos transferidos por la red. Dado que existe la necesidad de transferir los archivos de entrada entre nodos, estos archivos son leídos desde el sistema de jerarquía de almacenamiento local del nodo donde están ubicados y escritos otra vez en los nodos que los reciben, proporcionando así una menor latencia de acceso al disco.

Para la gestión de los archivos de entrada compartidos y temporales propusimos introducir dispositivos como RamDisk, discos duros locales y SSDs locales para el almacenamiento temporal de los archivos. En nuestra propuesta, cuando un nivel de la jerarquía de almacenamiento alcanza su capacidad límite, el planificador vuelca los datos temporales desde la RamDisk hacia el disco duro local o SSD local, y no hacia el sistema de archivos distribuido como ocurre normalmente. Utilizamos el sistema de archivos distribuido para volcar datos de salida finales, que no tienen dependencias. La RamDisk está implementada sobre la memoria principal

del nodo de cómputo. Sin embargo, como la memoria utilizada para construir la RamDisk tiene un tamaño limitado y es volátil, definimos una política de sustitución Last Recently Used (LRU) y una política de replicación en varios nodos para el caso de nodos sin capacidad de almacenamiento suficiente, cuando más aplicaciones que precisen de datos compartidos la necesiten. Este enfoque reduce el coste de acceso a archivos de entrada compartidos y temporales, generados por las aplicaciones intensivas de datos.

1.7.1 Objetivo Principal

Nuestra investigación propone adaptar una heurística de planificación de lista basada en el camino crítico para multiworkflows en clústers, teniendo en cuenta la ubicación de los datos y tomando como criterio relevante la compartición de datos por varias aplicaciones. De esta forma, el planificador consciente del almacenamiento tiene en cuenta una jerarquía con distintos niveles como sistemas de archivos distribuidos, discos locales, Ramdisk y las caches del sistema y lo llamamos CPFL (Critical Path File Location)[2].

1.7.2 Objetivos Específicos

Para alcanzar el objetivo principal definimos los siguientes objetivos específicos.

- Caracterizar las aplicaciones intensivas de datos e identificar los posibles cuellos de botella en un sistema de clúster compartido. Elegimos aplicaciones bioinformáticas representativas como FastQGrommer o SamTools del tipo de control de calidad y transformación de formatos de archivos. BWA, Bowtie, ClustalW que son aplicaciones especializadas en alineación de secuencias de genomas y GATK, como representantes de aplicaciones de análisis de variantes. Estas aplicaciones comparten el conjunto de archivos de entrada entre distintas aplicaciones, además de generar grandes volúmenes de datos temporales entre cada una de ellas.
- Diseñar e implementar una política de planificación que aumente la localidad

de datos para aplicaciones de workflows bioinformáticos, que compartan el mismo conjunto de archivos de entrada.

- Evaluar el uso de una RamDisk, que pueda ser utilizada por la política de planificación de workflows como un mecanismo de almacenamiento de archivos temporales y de entrada compartidos.
- Evaluar el tiempo de *makespan* obtenido por la política de planificación de workflows.
- Extender el simulador de ejecuciones de workflows WorkflowSim, para evaluar la escalabilidad de la política desarrollada en clústers de mayor tamaño, y comparar nuestra propuesta con otros planificadores del estado del arte.
- Extender el administrador de workflows científicos Galaxy para que pueda ser capaz de ejecutar la política desarrollada en un sistema clúster.

1.8 Organización de la Memoria

Este documento está organizado en cinco capítulos.

Capítulo 1. Describimos la problemática asociada a la planificación de aplicaciones que comparten el mismo conjunto de archivos de entrada, y al problema de gestión de grandes volúmenes de archivos temporales y finales. También presentamos en el estado del arte, diferentes enfoques para solucionar estos problemas. Definimos el objetivo de nuestro trabajo y ofrecemos una descripción de nuestra propuesta de solución.

Capítulo 2. Presentamos la arquitectura del planificador consciente del almacenamiento como solución a la problemática de planificación de workflows con aplicaciones de cómputo intensivo de datos.

Capítulo 3. Describimos la integración del planificador al administrador de workflows científico Galaxy, y la integración del planificador al simulador WorkflowSim para probar la escalabilidad de la propuesta.

Capítulo 4. Ofrecemos una descripción de los experimentos realizados. Describimos las aplicaciones utilizadas en los experimentos, el entorno en los que fueron realizados y los indicadores de rendimiento que fueron aplicados.

Finalizamos con las conclusiones, las publicaciones realizadas durante la tesis, y proponemos las líneas futuras hacia la continuación de este trabajo de investigación.

Capítulo 2

Arquitectura del Planificador Propuesto

En esta sección describimos la propuesta del planificador consciente del almacenamiento para mejorar el *makespan* de los multiworkflows bioinformáticos en clústers compartidos. Brindamos detalles de cómo se construyen los meta-workflows, y cómo podemos utilizar la información sobre las dependencias de datos obtenidas en el análisis de patrones, para tener criterios de ubicación de los archivos en la jerarquía de almacenamiento. Por último, ofrecemos una descripción de cómo aplicar estos principios para situar los archivos en los diferentes niveles de la jerarquía utilizando un algoritmo de pre-ubicación.

Inicialmente, se ha elegido HEFT [71] como política para resolver la planificación de múltiples workflows. El objetivo principal de la planificación basada en listas como HEFT es encontrar un orden de ejecución eficiente de un conjunto de aplicaciones, y asignarlas al mejor recurso disponible basándose en la información de prioridad. La prioridad de cada aplicación está determinada por la suma del tiempo de ejecución y el costo de comunicación.

Esta política está descrita en detalle en el algoritmo 1. En concreto, entre las líneas 1 y 3 calculamos la prioridad de cada aplicación según el tiempo de cómputo y el costo de comunicación. Luego, en la línea 4, se ordena la lista de acuerdo al orden decreciente de prioridad de cada aplicación.

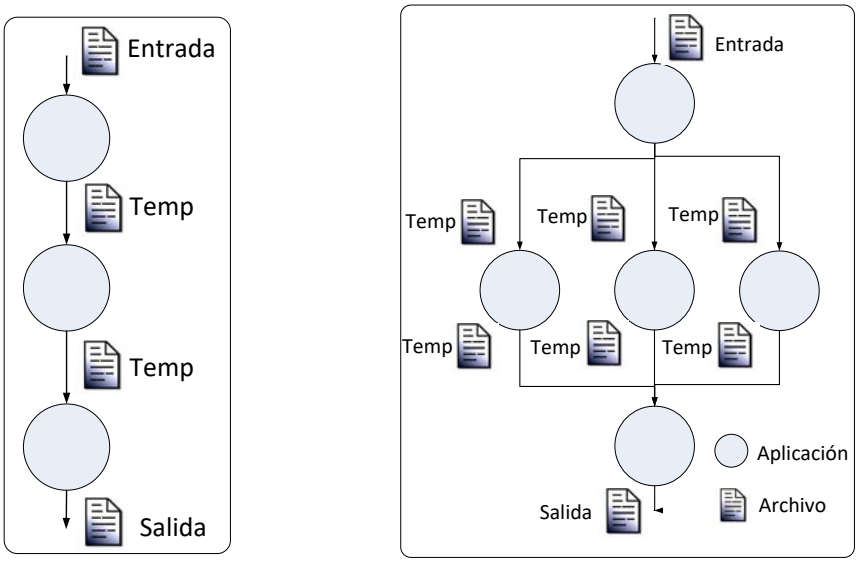
Algoritmo 1: Política HEFT

```
   input : Workflow
   output: A schedule
 1 compute the average execution time for each task according to a cost function;
 2 compute the average data transfer time between tasks and their successors according to communication
     cost function;
 3 compute level for each task; ;       // level is the sum of computation time plus communication time
 4 sort the tasks in a scheduling list Q by decreasing order of task rank value;
 5 while Q is not empty do
 6 |   t = remove the first task from Q;
 7 |   if t.predecessors == DONE then
 8 |   |   r = find a resource which can complete t at earliest time ;        // Only execute Task w/ all
 |   |   |     parents done
 9 |   |   schedule t to r;
10 |   end
11 |   else
12 |   |   add t to Q;
13 |   end
14 end
```

A pesar de que el tiempo de comunicación de archivos, se ha considerado un factor de coste importante en la planificación basada en listas, los workflows generalmente no proporcionan información referente a la ubicación de los archivos en el sistema de cómputo donde se van a ejecutar. De aquí que la propuesta presentada sea extender HEFT, para que el planificador tenga información respecto a la ubicación global de todos los archivos necesarios en la jerarquía de almacenamiento, y pueda realizar una asignación eficiente de los recursos del clúster, y mejorar por ende la ejecución de los workflows.

Para que el planificador pueda recoger la información necesaria sobre la ubicación y tamaño de los archivos, analizamos los patrones de diseño más comúnmente utilizados de los workflows en nuestra experimentación, para poder así determinar la localización de los mismos en los distintos elementos de la jerarquía de almacenamiento.

Como se muestra en la figura 2.1, basándonos en el uso de los archivos de los workflows bioinformáticos estudiados, encontramos dos patrones principales de ejecución de workflow: patrones secuenciales y paralelos. Podemos ver que un archivo de entrada para una aplicación podría generar varios archivos temporales compartidos. Entonces, podríamos tener un árbol de dependencia con diferentes aplicaciones para el caso de un patrón paralelo.



(a) Patrón Secuencial (b) Patrón Paralelo

Figura 2.1: Patrones de uso de archivos de workflows bioinformáticos

Diferenciar patrones de uso de archivos, nos permite reconocer con un análisis de diseño cuales son los archivos de entrada o temporales compartidos en el workflow. Esta información, de los archivos compartidos es de gran relevancia para nuestro trabajo, de este factor depende la ubicación de archivos compartidos en la jerarquía de almacenamiento.

La idea principal de la solución es, en primer lugar utilizar una caracterización de las aplicaciones bioinformáticas que pertenecen a los workflows en términos de utilización de los recursos de cómputo. En concreto, métricas de tiempo de ejecución, número y tamaño de las lecturas y escrituras en los diferentes sistemas de ficheros, tamaño del programa en la memoria principal (RSS), y de utilización de la CPU entre los parámetros más representativos.

En segundo lugar, creamos un meta-workflow con los workflows caracterizados, a efectos de evaluar una lista de prioridades ordenando las aplicaciones según el concepto de nivel, utilizando el tiempo de cómputo y el costo de comunicación cuando los archivos están en el sistema de archivos distribuido.

Posteriormente, recuperamos la información del patrón de uso de los archivos compartidos, así como sus tamaños y ubicación. Con la información extraída

el meta-workflow, descubrimos las aplicaciones que utilizan archivos de entrada y temporales compartidos. Esta información sobre los archivos compartidos nos ayuda a ubicarlos en la RamDisk local, el disco local o el sistema de archivos distribuido. Definimos como prioritarios los archivos compartidos. Inicialmente, todos los archivos se encuentran en el sistema de archivos distribuido. Al iniciar la planificación, aplicamos un algoritmo de pre-ubicación que mueve los archivos a la RamDisk cuando el fichero es mayor que 1024 MBytes y se ha identificado como archivo temporal o de entrada para más de una aplicación. Si no hay suficiente espacio, se utilizará un nivel de almacenamiento de mayor latencia como el disco local o el sistema de archivos distribuido. Cualquier otro archivo se almacena directamente en el sistema de archivos distribuido. Este módulo de pre-ubicación es útil para minimizar los tiempos de carga de datos, y permite la distribución de los archivos necesarios desde los sistemas de archivos remotos, y proporcionan una menor latencia de acceso a archivos.

Finalmente, mantenemos un registro de ubicaciones de archivos, y utilizamos la lista de prioridades, para ejecutar las aplicaciones en los nodos de cómputo donde se encuentran localizados los archivos.

Presentamos el diseño de la arquitectura, junto con los módulos que componen la política de planificación. A efectos prácticos, describimos cada una de las 3 capas de la arquitectura que se muestra en la figura 2.2: Nivel de Usuario, Pre-planificador y Planificador.

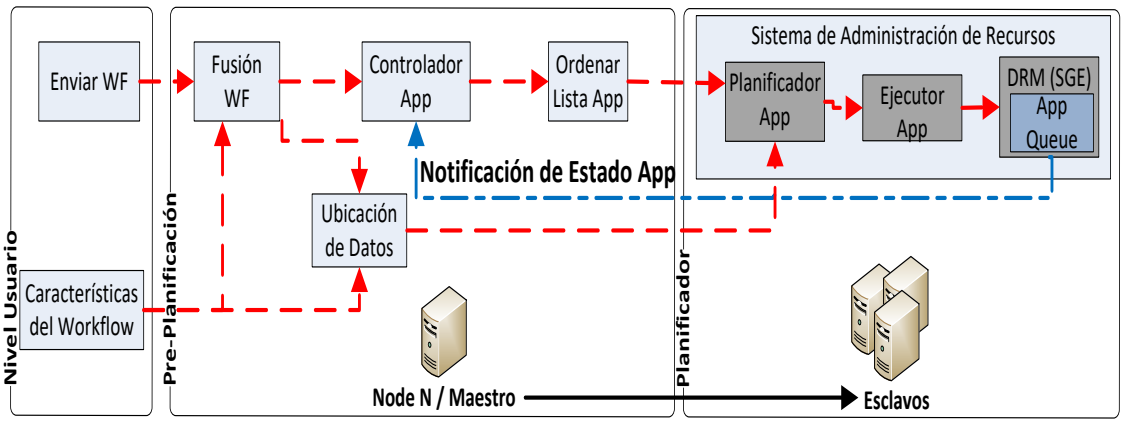


Figura 2.2: Arquitectura del Planificador

2.1 Nivel Usuario

Cada científico diseña sus workflows utilizando diversos tipos de herramientas, que pueden ir desde conjuntos de *scripts*, hasta herramientas de composición de workflows basadas en aplicaciones web. De cualquier forma, en este nivel, se definen los metadatos a partir de las características del workflow y se encapsulan en archivos XML para ser enviados a las siguientes capas.

2.1.1 Características del Workflow

El registro histórico de la herramienta de administración de workflows, alimenta al planificador con un conjunto de características de aplicaciones, como podemos ver en el XML2.1. Los metadatos generados contienen: el nombre de la aplicación, la versión de la misma, línea de comandos, archivos utilizados, tamaño de archivos, tiempo de ejecución, como parámetros más relevantes.

```
"Application": {
  "input_connections": {
    "genomeSource|ownFile": {
            "uses"="wf_in.dat"
            "size"=13631488
    "paired|input1": {
            "uses"="scan_in1.dat"
            "size"=512000
    "paired|input2": {
            "uses"="scan_in2.dat"
            "size"=512000
  },
  "inputs": [],
  "name": "Map with BWA for Illumina",
  "outputs": ["uses"="scan_in5.dat", "size"=628097024],
  "tool_id":"toolshed.g2.bx.psu.edu/repos/devteam/bwa_wrappers/bwa_wrapper/1.2.3",
  "tool_version": "1.2.3", "runtime"="24", "cores"="1"
  },
},
```

Código Fuente 2.1: XML Aplicación

2.1.1.1 Enviar WF

Mediante la herramienta de gestión de workflow a nivel de usuario, el workflow creado puede enviarse al módulo de pre-planificación para su posterior tratamiento.

2.2 Pre-Planificación

En este módulo, una vez evaluados los atributos del patrón de diseño del workflow recibido desde el nivel de usuario, decidimos dónde ubicar los archivos en la jerarquía de almacenamiento, antes de crear la lista de aplicaciones a ser utilizada por el sistema de administración de recursos.

La capa de Pre-Planificación, es la encargada de generar el meta-workflow y ubicar los archivos compartidos en la jerarquía de almacenamiento. Como se muestra en la figura 2.3.

Una vez creado el meta-workflow correspondiente, se construye una lista de prioridades basada en el camino crítico extendido del HEFT. Al mismo tiempo, por medio del análisis del patrón del meta-workflow determinamos los archivos de entrada compartidos que serán almacenados en la jerarquía de almacenamiento. Los archivos serán ordenados por prioridad según su tamaño de mayor a menor, considerando igualmente el factor cantidad de aplicaciones los comparten.

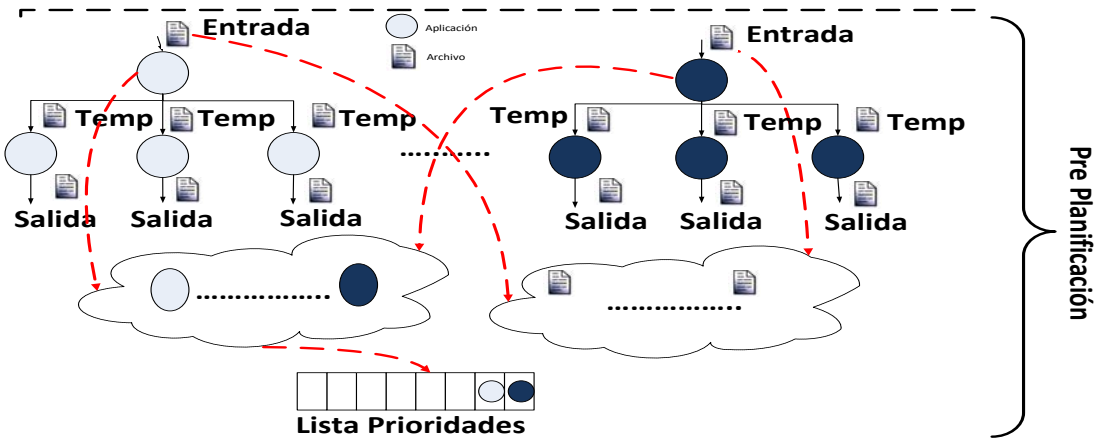


Figura 2.3: Diseño del Módulo del Pre Planificador CPFL

2.2.1 Fusión WF

Todas las aplicaciones de los workflows que ingresan al sistema se combinan en un único meta-workflow. El meta-workflow generado se extiende con dos nodos ficticios, uno inicial y otro final para luego generar una nueva lista basada en camino crítico.

2.2.2 Ubicación de Datos

Este módulo se encarga de ubicar los archivos en la jerarquía de almacenamiento. Con la información expuesta de la fusión de los workflows en el módulo Fusión WF, conocemos el número de aplicaciones que están utilizando archivos de entrada y temporales compartidos. Se selecciona la RamDisk local, disco local, o sistema de archivos distribuido, como las ubicaciones destino de los archivos en el sistema jerárquico de almacenamiento. En una primera fase, los archivos de entrada se encuentran en el sistema de archivos distribuido, y se copian a la jerarquía de almacenamiento local del nodo de cómputo. Los archivos son ubicados según el tamaño, y en base al número de veces que serán requeridos hasta que el medio de almacenamiento este completo. Los siguientes archivos serán almacenados, en el siguiente nivel de la jerarquía de almacenamiento. La ubicación de cada archivo se mantiene en una estructura de datos. Todos los archivos temporales generados por las aplicaciones para las próximas etapas del workflow deben almacenarse en el mismo disco duro local, o en Ramdisk en la medida de su capacidad.

2.2.3 Controlador App

Partimos de un primer grupo de workflows. Inicialmente, aplicamos el camino crítico al primer grupo de workflows. Cuando se planifica un nuevo grupo de workflows, el Controlador App vuelve a ejecutar el algoritmo de camino crítico a las nuevas aplicaciones en el sistema, pero sin cambiar el estado de las aplicaciones en ejecución. No se asignan aplicaciones a ninguna unidad de cómputo si hay aplicaciones previas en la precedencia del workflow sin finalizar.

2.2.4 Ordenar Lista App

Una vez que el Controlador App finaliza el camino crítico de un grupo de workflows, el módulo Ordenar Lista App se encarga de mantener una cola de aplicaciones ordenada de mayor a menor prioridad para enviarla a la capa del Planificador.

2.3 Planificador

El planificador asigna aplicaciones a nodos de cómputo específicos del clúster, siguiendo la lista de prioridades construida en el módulo Controlador App, y ordenada por el módulo Ordenar Lista App. Las asignaciones se realizan considerando el nodo que contiene los archivos de entrada para esas aplicaciones.

Como podemos ver en la figure 2.4 las aplicaciones son encoladas según el orden de la lista realizada por el módulo Ordenar Lista App en la capa de Pre-Planificación. Las aplicaciones son asignadas a las colas de los nodos donde los archivos fueron previamente ubicados.

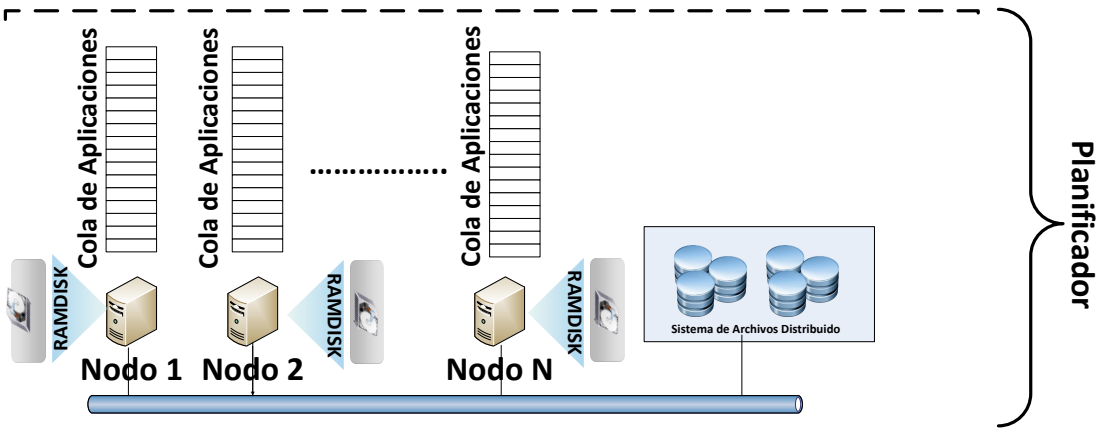


Figura 2.4: Diseño del Módulo Planificador de workflows CPFL

2.3.1 Sistema de Administración de Recursos

Este módulo se encarga de gestionar el estado de los recursos de cómputo del sistema, y de las aplicaciones que se ejecutan en la plataforma.

2.3.1.1 Planificador App

El módulo en cuestión, administra las aplicaciones interdependientes de los workflows, y realiza la planificación. La lista de aplicaciones proporcionada por el módulo Ordenar Lista App, se ordena en una única lista de aplicaciones con estado de "Listo". La aplicación de mayor prioridad estará primera en la lista, esperando ser asignada a un recurso. Será enviada al Ejecutor App cuando todas sus predecesoras estén finalizadas. La prioridad de una aplicación puede cambiar, mientras permanece en la cola sin ser ejecutada, independientemente de si pertenece a diferentes workflows o usuarios. En este trabajo, no realizamos un análisis para seleccionar la cantidad de recursos de cómputo óptimos. En su lugar, limitamos la selección máxima al factor de paralelismo máximo del workflow. El factor de paralelismo se calcula según el mayor número de aplicaciones del mismo nivel en el grupo de workflows.

2.3.1.2 Ejecutor App

Este módulo ejecuta la primera aplicación de la lista de prioridades. La aplicación es finalmente enviada a través del gestor específico del clúster (SGE en el caso que nos ocupa), a un nodo de cómputo concreto, que contiene todos los archivos necesarios para su ejecución.

2.3.1.3 DRM

Para las aplicaciones de altas prestaciones clásicas, la gestión de recursos se realiza a través de un Administrador de Recursos Distribuido (DRM) que proporciona información sobre el clúster y las aplicaciones, por lo que se puede planificar en la cola de aplicaciones interna. Aprovechamos el DRM para procesar los registros históricos

del sistema y las secuencias de ejecución proporcionadas por un administrador de Workflows. Las instrucciones sobre qué nodo y recurso utilizar se especifican aquí, para el Controlador App. Al mismo tiempo, el modulo Ubicación de Datos utiliza la información proporcionada por el DRM para mover archivos a través de los diferentes niveles del sistema de jerarquía de almacenamiento.

Por último, podemos analizar algunos detalles de la implementación de los módulos con el objetivo de profundizar sobre el funcionamiento del algoritmo en cada etapa.

El módulo de Nivel de Usuario no tiene asignado ningún algoritmo ya que depende de la herramienta de administración de workflows que el usuario desea utilizar para crear los patrones. Las características de los workflows son proporcionados por un registro histórico como el archivo XML 2.2.

```
1  <job id="a" namespace="motif" name="Pre Interproscan"
2  runtime="24" cores="1">
3      <uses file="wf_in.dat" link="input" size="13631488"/>
4      <uses file="scan_in3.dat" link="output" size="102400"/>
5  </job>
6  <job id="b" namespace="motif" name="Inter" runtime="36" cores="1">
7      <uses file="scan_in7.dat" link="input" size="102400"/>
8      <uses file="scan_out7.dat" link="output" size="512000"/>
9  </job>
10 <job id="c" namespace="motif" name="Inter" runtime="30" cores="1">
11     <uses file="scan_in4.dat" link="input" size="102400"/>
12     <uses file="scan_out4.dat" link="output" size="512000"/>
13 </job>
14 <job id="d" namespace="motif" name="Inter" runtime="9" cores="1">
15     <uses file="scan_in10.dat" link="input" size="102400"/>
16     <uses file="scan_out10.dat" link="output" size="512000"/>
17 </job>
18     <child ref="b">
19     <parent ref="a"/>
20 </child>
21     <child ref="c">
22     <parent ref="a"/>
23 </child>
24     <child ref="d">
25     <parent ref="a"/>
26 </child>
```

Código Fuente 2.2: XML Histórico

Algoritmo 2: Etapa Pre Planificador

input : New WorkFlow, User WF Characterizations
output: Updated Meta Workflow, List Scheduling

```
1  for All Applications on New Workflow do
2      User Wf Characterization ;                  // Characteristics provided at the User Level Layer
3      User Application Characterization ; // Application characteristics, Data Files with sizes, and
         Application Parents and Branch Factors are provided by the User Level
4  end
   /* The initial batch of Workflows arrive at the system and are merged at prescheduling layer
      through the WF Merge Module here                                                              */
5  while New Workflow Applications are not in Meta Workflow do
6      Node add of application from New Workflow to Meta Workflow;
7      Application of New Workflow with no parents edge to Start node of Meta Workflow;
8      Application of New Workflow with no successor edge to End node of Meta Workflow;
9      Add New Applications to Meta Workflow;
10 end
11 List = CriticalPath on Meta Workflow ;  // The App Controller execute a CriticalPath algorithm to
     the Meta Workflow to generate the Priority List
   /* Data Placement Module, locate each data file of the Priority List Applications              */
12 if fileInSize = SMALL then
13     keep file on Distributed File System ;                  // Third Level of Storage Hierarchy
14 end
   /* BranchBrothers used to determine Application using shared files                             */
15 if (fileInSize = MEDIUM or BIG) and Shared then
       /* Ramdisk or Local Disk is selected as storage                                           */
16     Select Storage Hierarchy according to storage availability;
17     if Ramdisk is not full                                  // First Level of Storage Hierarchy
18      then
19         copy file input from Distributed File System to local Ramdisk;
20     end
21     else
22         Replace Input file from Ramdisk;
23         if Replaced file is Temporal and BranchBrothers are DONE then
24             delete replaced file ;              // We free RamDisk space from all Temporal Files
25         end
26         else if Local disk is not full                      // Second Level of Storage Hierarchy
27          then
28             copy replaced input file to Local disk ;
29         end
30         else
31             Move input file from LocalDisk to Distributed File System;
32         end
33     end
34 end
   /* Output Data Files goes to the Distributed File System always                               */
35 if FileOut is from Last Application then
36     Copy FileOut to Distributed File System;
37 end
```

En la etapa de Pre-Planificación, utilizamos el algoritmo 2. Las líneas 1 a 3 inicializan los valores para la caracterización del workflow proporcionado por el diseño del usuario. En las líneas 5 a 9, combinan todos los workflows en un meta-workflow añadiendo dos nodos ficticios para los nodos de inicio y fin. La línea 10 aplica el análisis de camino crítico a este nuevo meta-workflow.

La implementación del módulo de Ubicación de Datos se describe desde la línea 14. Aquí clasificamos los archivos para mantener archivos pequeños (SMALL) (<1MB) en el sistema de archivos distribuido. Además, decidimos qué archivos compartidos más grandes, de tamaño medio (MEDIUM)(>512MB) y grande (BIG) (>1028MB) necesitan ser copiados a la jerarquía de almacenamiento local. En la línea 21, describimos una política de reemplazo de archivos cuando las ubicaciones de almacenamiento local están llenas. En esos casos, necesitamos copiar los archivos de nuevo al sistema de archivos distribuidos para liberar el espacio de almacenamiento local. Finalmente, en la línea 33, describimos cómo se van a escribir todos los archivos de salida en el sistema de archivos NFS.

En la siguiente etapa, el del Planificador, aplicamos una heurística de planificación a la lista de aplicaciones descritas en el algoritmo 3.

Inicialmente, todas las aplicaciones están en estado READY para ser planificadas. En la línea 1 recibimos el estado del recurso a través del DRM y analizamos el registro de procesos. Una vez que tengamos toda la información sobre disponibilidad de recursos, en la línea 3 y 4 clasificamos la lista de prioridades y actualizamos el estado de todas las aplicaciones a STANDBY. De esta manera, describimos qué aplicaciones están listas para ejecutarse, pero aún no están asignadas a una unidad de cómputo.

Algoritmo 3: Planificador

```
   input : Meta Workflow, List Scheduling, Available Resources
 1 Available Resources = Resource Status trough the Distributed Resource Management and read process log;
 2 while List of Priorities not empty do
 3     upgrade application status to STANDBY ;
 4     Sort List of Priorities ;                                  // Internal List of Applications
 5     Select Resources = Max(BranchBrothers of first on List) ;   // Maximun cores according to
         parallelism
 6     for i = All Applications in List do
 7         if all Parents of application == DONE and Selected Resources != 0 then
 8             if fileInput of Application[i] already on place then
                   /* If precedence is complete then send application to resource          */
 9                 execute Application on Selected Resource;
10                 Selected Resources = Selected Resources - 1 ;
11                 add to List of Priorities at the end;
12             end
13         end
14         else
15             add to List of Priorities at first;
16         end
17     end
       /* State machine should be always updated                                          */
18     for all applications in List do
19         if application status is DONE then
20             remove application from List;
21         end
22         upgrade application status to READY/STANDBY/RUNNING/DONE;
23     end
24 end
```

Seleccionamos los recursos de cómputo de destino en la línea 5 considerando el máximo paralelismo del workflow en el nivel de la aplicación actual. De las líneas 6 a 17 hacemos efectiva la asignación de recursos y cambiamos el estado de la aplicación a RUNNING, lo que significa que la aplicación se asigna correctamente a una unidad de cómputo. Si todos los predecesores de la aplicación están en el modo DONE, es decir, estas aplicaciones finalizan la ejecución correctamente, las trasladamos al final de la lista para controlar el estado más tarde. Si el estado de uno de los padres no es del tipo DONE, entonces la aplicación se mueve a la primera posición para esperar a que se compruebe todas las precedencias y que los recursos estén disponibles. Al final, de las líneas 18 a 24 controlamos si todas las aplicaciones están en DONE. Se realiza una actualización del estado como control de parada en READY/STANDBY/RUNNING o DONE para cada aplicación.

La adaptación del planificador de multiworkflows HEFT en entornos donde

se tiene en cuenta el camino crítico, además de los archivos de entrada y temporales compartidos en la jerarquía de almacenamiento, propone un trabajo de búsqueda de la mejor combinación de almacenamiento y computo.

Esta búsqueda de mejor combinación genera un tiempo de computo que queremos evitar. Para resolver este problema con un método ágil, una de las formas más comunes es la de utilizar un simulador para explorar todas las soluciones.

Capítulo 3

Integración del Planificador en WorkflowSim y Galaxy

Para comprobar si el planificador desarrollado[2] escala en clústers de mayor tamaño, decidimos utilizar un simulador de workflows conocido. WorkflowSim [23] es una herramienta de simulación de código abierto desarrollada en la University of Southern California, y es una extensión del simulador CloudSim. Se utiliza para simular workflows que han sido modelados como DAGs, definidos a través de archivos XML, e implementa un conjunto variado de algoritmos clásicos de planificación como HEFT, Min-Min, Max-Min o FCFS; lo cual nos permiten comparar nuestras propuestas a otras referidas en la literatura.

3.1 Escalabilidad del planificador usando WorkflowSim

En la figura 3.1 podemos observar, que el simulador está dividido en dos grandes grupos de módulos. *Submit Host* y *Execution Site*, encapsulan los sub módulos principales, y resaltamos los módulos que extendimos para poder escalar nuestra propuesta.

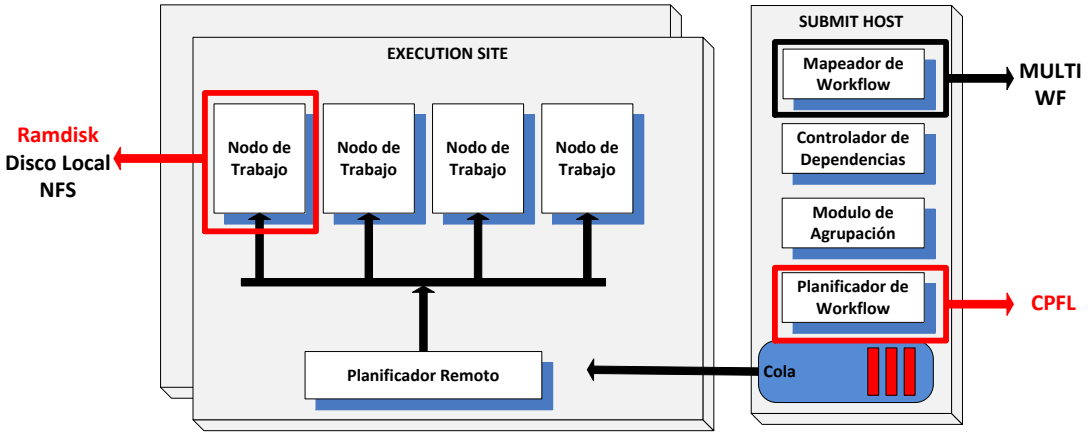


Figura 3.1: Componentes de WorkFlowSim [23]

En el *Submit Host*, el Mapeador de Workflows es responsable de importar varios DAGs que están concatenados para la ejecución de varios workflows. A continuación, crea una lista de aplicaciones que se asignarán a los recursos disponibles. En cualquier caso, debemos cumplir las dependencias originales de las aplicaciones, para respetar el orden de ejecución natural de predecesores del workflow.

Cuando sea solicitado, el Modulo de Agrupación es responsable de encapsular varias aplicaciones dentro de un solo trabajo. Un planificador de workflows, de acuerdo con los criterios definidos por el usuario, agrega efectivamente cada trabajo o aplicación a una cola de aplicaciones listas para ser asignadas a nodos de cómputo.

Algoritmo 4: Ubicación de archivos CPFL WorkflowSim

```
   input  : Storage, requiredFiles, Node, Cluster
   output: A schedule
1  double time = 0.0;
2  for FileItem on requiredFiles do
   |  /* Check if the file exists                                          */
3  |  if file.isRealInputFile(requiredFiles) then
4  |  |  List.String siteList = getStorageList(file.getName()) ;   // Retrive storage hierarchy
   |  |  /* Retrive comunication time for file on hierarchy level          */
5  |  |  for String site : siteList do
6  |  |  |  time += file.Comtime() ;                  // Sum communication cost of app files
7  |  |  end
8  |  end
9  end
```

En el planificador de workflows hemos introducido el código de CPFL como se muestra en el algoritmo 4 de ubicación de archivos, y en el algoritmo de planificación 6, siguiendo la guía para extensiones de WorkflowSim. La integración se hizo mediante la adición de un archivo CPFLScheduler de Java, que implementa el planificador consciente del almacenamiento CPFL.

Se ha modificado el módulo *Execution Site* del WorkFlowSim, específicamente la representación de los nodos de trabajo.

Se han implementado nuevos elementos en la jerarquía de almacenamiento en org.cloudbus.cloudsim. En concreto, se han implementado los módulos *SsdDriveStorage* y *RamdiskDriveStorage*. Este último según el algoritmo 5. En consecuencia, hemos introducido los parámetros necesarios para modelar la latencia, el tiempo promedio de búsqueda, y la tasa de transferencia máxima al dispositivo de almacenamiento específico de acuerdo con las especificaciones técnicas de la industria. Hemos implementado RamDisk sobre una memoria RAM DDR3-1333 SDRAM como almacenamiento local en el *Execution Site*, de tamaño fijo de 6.2GB, velocidad de transferencia máxima de 1.8GB/s, latencia de 0.003ms y 0.001ms de tiempo de búsqueda promedio. Para el SSD Intel SSD 510 Series, creamos un tamaño fijo de 1TB y velocidad de transferencia máxima de 500MB/s, latencia de 0.06ms y tiempo promedio de búsqueda de 0.1ms.

Algoritmo 5: Definición de Almacenamiento

```
   input  : capacity
   output: Communication Cost
 1 RamdiskdriveStorage(double capacity);
 2 if capacity <= 0 then
 3 |   throw new ParameterException(RamdiskdriveStorage(): Error - capacity <= 0.);
 4 end
 5 name = RamdiskdriveStorage;
 6 this.capacity = capacity;
 7 fileList = new ArrayList [File]; ;                          // Initialize parameters
 8 nameList = new ArrayList [String];
 9 gen = null;
10 currentSize = 6.2GB;
11 latency = 0.003ms;
12 avgSeekTime = 0.001ms;
13 maxTransferRate = 1.8GB;
```

Los parámetros de la infraestructura del clúster se definen en el nodo de cómputo. Por ejemplo, el número de procesadores, capacidad de RAM, capacidad de

almacenamiento local y los tipos de almacenamiento local. Necesitamos poder añadir los nuevos tipos de almacenamientos, para evaluar nuestra propuesta. Finalmente, se creó un nuevo clúster con una cantidad determinada de nodos de cómputo, con parámetros definidos como el sistema operativo y los costes de comunicación entre los niveles de la jerarquía de almacenamiento, definiendo así el archivo del planificador según el algoritmo 6.

Algoritmo 6: CPFL WorkflowSim

```
   input  : Workflows, Cluster
   output: Done
 1 CPFLSchedulingAlgorithm extends HEFTPlanningAlgorithm ;          // Extend Java HEFT algorithm
 2 daxPaths.add(/home/ceag/Downloads/data15out.xml) ;       // Setting workflow XML definition path
 3 Parameters.SchedulingAlgorithm schmethod = Parameters.SchedulingAlgorithm.CPFL ; // Setting CPFL
    as scheduler
 4 ReplicaCatalog.FileSystem filesystem = ReplicaCatalog.FileSystem.SHARED; // (OR LOCAL file system
    for the cluster
 5 OverheadParameters op = new OverheadParameters(0, null, null, null, null, 0);   // No cluster overhead
 6 vmSize = getCloudletList().size();
 7 while j <vmSize do
 8 |   vm = getVmList.get(j) ;                                  // if resource is idle, assign job
 9 |   if vm.getState = WorkflowSimTags.VMSTATUSIDLE then
10 |   |   job = cloudlet;
11 |   |   time = dataTransferTime(job.getFileList(), cloudlet, vm.getId());
12 |   |   if time <minTime then
13 |   |   |   minTime = time ;                          // assign min time to complete job as cost
14 |   |   |   closestVm = vm;
15 |   |   end
16 |   end
17 end
18 WorkflowSim.startSimulation();
```

El siguiente paso, fue ajustar los parámetros del simulador, para que se comporten lo más parecido posible a los del clúster prototipo del tipo IBM, ambos se comparan en la tabla 3.1:

Especificaciones	Simulador Estandar	Simulador (tipo-IBM)
Nodos p/Cluster	4	32
Procesadores p/ Nodo	1	4
Procesadores Frec.	1000 MIPS	6000 MIPS
RAM p/ Nodo	2 GB	12 GB
Capacidad Disco p/ Node	1 TB	10 TB
Ramdisk	-	6.2 GB
Latencia Red	0.2 ms	0 ms
Latencia Interna	0.05 ms	0.05 ms

Tabla 3.1: Especificaciones del Clúster Simulado

3.2 Administradores de workflows científicos

Para facilitar la creación de workflows, los científicos pueden acceder a una amplia variedad de herramientas, que permiten compartir experimentos en comunidades científicas distantes, además de poder replicar resultados y por tanto centrar sus esfuerzos en seleccionar las herramientas más adecuadas para solucionar problemas particulares.

En la mayoría de los laboratorios bioinformáticos, los workflows se implementan como secuencias de comandos en un lenguaje de *scripting* como Perl o Bash, donde el investigador obtiene los datos de una carpeta y llama secuencialmente a todas las aplicaciones principales necesarias con sus parámetros de ejecución. Hay algunas alternativas a este enfoque de granularidad demasiado fina, como usar workflows basados en servicios, que permiten descubrir e integrar una colección de servicios web disponibles.

Podemos citar tres familias de administradores de workflows científicos, basándonos en subdivisiones definidas por Romano[61] y con pequeños ajustes de Bux [18] son:

- Lenguajes textuales, que consisten en lenguajes de programación textual de bajo nivel, basado en batch y archivos de configuración para definir el esquema

de ejecución.

- Sistemas gráficos, se podría decir que su mayor logro es el de proveer de una interfaz gráfica para el diseño y ejecución de los workflows, mientras todavía dependen de archivos de configuración y algunas líneas de código textuales, que se tratan de mantener alejadas del usuario.

- Portales de dominio específico, preparados en un ambiente *online*, están diseñados para ejecutar y compartir workflows. Dado que son herramientas *online*, no necesitan más que de un navegador para cumplir sus objetivos, sin instalar programas adicionales en la terminal del usuario; normalmente están compuestos de aplicaciones específicas, listas para ser usadas de acuerdo a la parametrización del usuario y ejecutarlos en un servidor público remoto o local.

Administradores de workflows Textuales El objetivo principal de estos workflows, es el de facilitar la distribución eficiente de cargas grandes de trabajo enfocándose menos en la facilidad de utilización.

Swift[77]: basado en línea de comando, es un lenguaje funcional en el que se pasan por parámetros los datos de entrada y se especifican las salidas; preparado para ser ejecutado en paralelo tanto en máquinas locales como en clúster, nubes, grids. El rol principal de este planificador es el de mantener el *performance* del workflow de acuerdo a las mediciones de cada recurso asignado y de esta forma balancear el cómputo en los nodos.

Condor DAGMan[31]: metodológicamente está enfocado al planificador de tareas, donde cada tarea es mantenida en una cola de ejecución, preparado para que el sistema sea confiable, posee varios mecanismos de recuperación ante fallos.

Pegasus[29]: Parecido al sistema de planificación Condor de grafos acíclicos definidos textualmente, provee otras estrategias: random, circular, grupos, heurística de finalización más corta. Puede ser ejecutado en paralelo sobre infraestructuras de nube y grids para diferentes dominios de trabajo. Esta herramienta provee una capa de abstracción donde se puede definir que recursos son necesarios, que datos replicar y que tipos de procesos o servicios utilizar, a esto lo llaman catálogo de recursos. En cuanto al planificador, provee alternativas totalmente nuevas con respecto a Cóndor:

Ramdom, donde las tareas son asignadas de manera ramdomica a los recursos; *Round Robin*, donde las tareas son asignadas y distribuidas entre los recursos de forma independiente; *Clustering* en las tareas, pueden ser definidas por el usuario de acuerdo a sus preferencias en el mismo grupo de recursos; o por Heurística, donde el costo de comunicación de las tareas puede ser definido en tiempo de ejecución para cada tarea especificada por el usuario.

Aun cuando se tienen varias herramientas para la distribución de trabajos, un problema sigue siendo la administración de los fragmentos de datos, tanto es así que anteriormente este tipo de trabajo estaba asignado a un solo nodo. Para solucionar, esto se buscan alternativas sobre sistemas hadoop y paradigmas MapReduce sobre nodos distribuidos.

Algunas de estas herramientas son, por ejemplo:

- Turbine[78]: un administrador de workflow enfocado a motores distribuidos sobre nodos de cómputo.
- Oozie[39]: como se encontraron con algunos problemas de escalabilidad en Turbine, se buscó la forma de solucionar esto con un planificador basado en Hadoop, donde se pueden definir sub tareas y múltiples clientes, partiendo del workflow y sus datos.
- Scicumulus[27]: capaz de describir un middleware basado en nube para workflows científicos, provee componentes de subida, envío, bajada y definición de origen de archivos. A través de cartuchos o "adaptadores", el usuario puede definir como quiere que los datos sean fragmentados y vueltos a unir al final del proceso de paralelización.
- PaPy[24]: las tareas son funciones de Python bien moduladas, el paralelismo de tareas es realizado a través de un enfoque distribuido de trabajadores sobre recursos remotos o locales.
- Dryad[38] y Nephele[8] soportan paralelismos de tareas, datos y *pipelines* sobre arquitecturas multinúcleo y pueden utilizar una cola de trabajo para hacer ejecuciones en una nube, sin embargo, al ser una herramienta de uso general, no es totalmente aprovechado para solucionar los problemas de workflows

científicos, además los lenguajes de workflow requieren de un usuario que este entrenado en la sintaxis de especificación de los trabajos.

Mientras todos estos lenguajes tienen como punto fuerte determinar paralelismo en gran cantidad de datos y trabajos, siguen siendo tratados como cajas negras donde el usuario tiene que modificar parámetros para su ejecución, haciendo que salgan del alcance de usuarios no experimentados.

Administradores de workflows Gráficos

Los administradores de workflows gráficos tratan de solucionar el problema de la facilidad de construcción de workflows y especificación de procesos secuenciales, algunas herramientas soportan multihilos y están enfocadas al consumo de servicios web. Comúnmente están diseñados para componer workflows anidados jerárquicamente, al contrario de los lenguajes textuales donde la mayoría de ellos eran diseñados para ejecutar una colección de componentes predefinidos o servicios externos como los servicios web. Normalmente están instalados en servidores remotos donde los clientes pueden acceder a través de una interfaz gráfica, el planificador típicamente está implementado a través de una cola de trabajo donde los datos son replicados en trabajadores para cada hilo.

- Taverna[54]: enfocado a workflows bioinformáticos, esta herramienta no está preparada para el consumo intensivo de datos. Si bien posee la capacidad de multihilos, no tiene la habilidad de poder instalarse en un clúster para el aprovechamiento de los recursos. Provee soporte para paralelismo de tareas, datos y secuencias, sin embargo, carece de un método más sofisticado de planificación más que la de cola de tareas, por lo que la utilización de núcleos está limitada a los recursos locales.
- KNIME[11]: del área de minería de datos, hace énfasis a los servicios web. Puede ser instalado en un entorno local o un servidor que puede ser accedido por varios clientes. Para determinar el paralelismo de datos necesita que el usuario especifique estrictamente cuales datos fragmentar, sin embargo, plataformas distribuidas como grid y nubes no son soportadas
- Kepler[47][73]: de aspecto y funcionamiento parecido a los dos primeros, enfocado a las estadísticas. Como no está preparado para cómputos intensivos

de datos, implementa un enfoque MapReduce sobre Hadoop integrado a Kepler, de esta forma se puede paralelizar el workflow sin preocuparse por la interfaz del lenguaje

Administradores de workflows de Dominio Específico La mayoría de los workflows de esta familia están preparados para utilizar bases de datos específicas, aplicaciones y servicios web especializados. Su punto fuerte es la sencillez de configuración, el trabajo intuitivo y la facilidad de compartir los trabajos; en la mayoría de los casos están desarrollados en un entorno web y son ejecutados en servidores remotos públicos o privados.

Con la aparición de NGS y el creciente volumen de datos, la herramienta Galaxy[34] se ha posicionado como referente de este tipo de herramientas, con entorno web fácil de instalar y usar y herramientas precompiladas; puede ser instalado en un entorno local, servidores en la nube, clúster o grids.

Debido a que están diseñados en un entorno de dominio especifico, vienen ya con componentes precompilados en su interface. Ensamblar dichos componentes con repositorios y poder compartirlos o ejecutarlos es aún más intuitivo. Basado en este tipo de herramientas se encuentra Galaxy, con entorno web y aun basado en él se encuentra CloudMan[5] capaz de ejecutar Galaxy en *Cloud*, básicamente pudiéndose ejecutar varios workflows en máquinas virtuales.

Buscando la interoperabilidad entre las herramientas de mayor aceptación en el campo científico, aparecen otras que son algunas mezclas de dos como Tavaxy[1], la cual intenta solucionar el problema de ejecución de workflows de Taverna en entornos Galaxy en tiempo de diseño. Sin embargo, siendo que la capacidad de generación de grandes cantidades de datos para análisis sigue siendo un punto relevante, aparece Conveyor[46], una herramienta del tipo cliente/servidor, donde se hace un fuerte hincapié en la jerarquía de datos y tipos y se centra en la ejecución local a través de hilos, efectuando una paralelización del tipo Paralelismo de Tareas.

Por otro lado, se encuentra Mobyle[52], el cual intenta solucionar el problema de acceso a repositorios de datos, replicando las tareas en los diferentes clientes registrados. Chipster[41] está enfocado a una gran colección de posibles tareas a ser utilizadas; se comporta como si fuera un gran repositorio de métodos e interfaces

para análisis de datos. Cloudgene[62], es un entorno de ejecución de programas del tipo MapReduce para bioinformática, permite la ejecución de pipelines en clúster locales o públicos como Amazon EC2. CloVR por su parte permite extender dinámicamente los recursos de una nube para la ejecución de rutinas del tipo BLAST.

Estas herramientas son relevantes para nuestro trabajo ya que implementan de manera separada distintos métodos que nos pueden ser útiles para el desarrollo de una metodología correcta de paralelización.

A pesar de que muchas herramientas que se encuentran dentro de esta familia están ya preparadas para la instalación de instancias en clúster y nubes, el problema principal que mantienen es el enfoque de servicios web, en el cual consumir una gran cantidad de datos a través de paso de datos es un cuello de botella importante.

No hay un gestor de workflows para clúster, que facilite la ubicación de los archivos utilizados por dichas aplicaciones en la jerarquía de almacenamiento, para mejorar el makespan en un contexto multiworkflow.

Para el diseño científico de multiworkflows y la administración de recursos de datos compartidos, y aplicaciones de uso intensivo de datos no hay suficientes referencias. Las aplicaciones de uso intensivo de datos se están adaptando actualmente a un paradigma MapReduce [28]. Debido a esto, CRS4 [57], MyHadoop [44], Seqpig [63] y Pulsar, anteriormente llamado light-weight runner (LRW) son algunos de los adaptadores que los investigadores utilizan para integrar Galaxy a un Clúster de tipo Hadoop. Sin embargo, en este trabajo utilizamos Bioblend [65], una API que encapsula las funcionalidades de Galaxy y CloudMan. BioBlend permite proporcionar la infraestructura necesaria, y automatizar análisis complejos sobre archivos grandes dentro del ambiente familiar de Galaxy. Bioblend se enfoca en la utilización de infraestructuras del tipo Cloud, en lugar de utilizarlo para ese ambiente la utilizaremos para alimentar a nuestro planificador con parámetros de aplicación y archivo presentes en Galaxy para enviar más tarde los workflows a un clúster.

Nuestro trabajo extiende Galaxy[34], el cual es un gestor web visual y fácil de usar para crear y modificar workflows de bioinformática, permitiendo una fácil reutilización y compartición de experimentos. Esta plataforma ha demostrado tener

una comunidad activa de usuarios que dan soporte a su desarrollo [67]. Como podemos ver en la figura 3.2 la comunidad científica utiliza Galaxy, en una amplia variedad de campos de investigación.

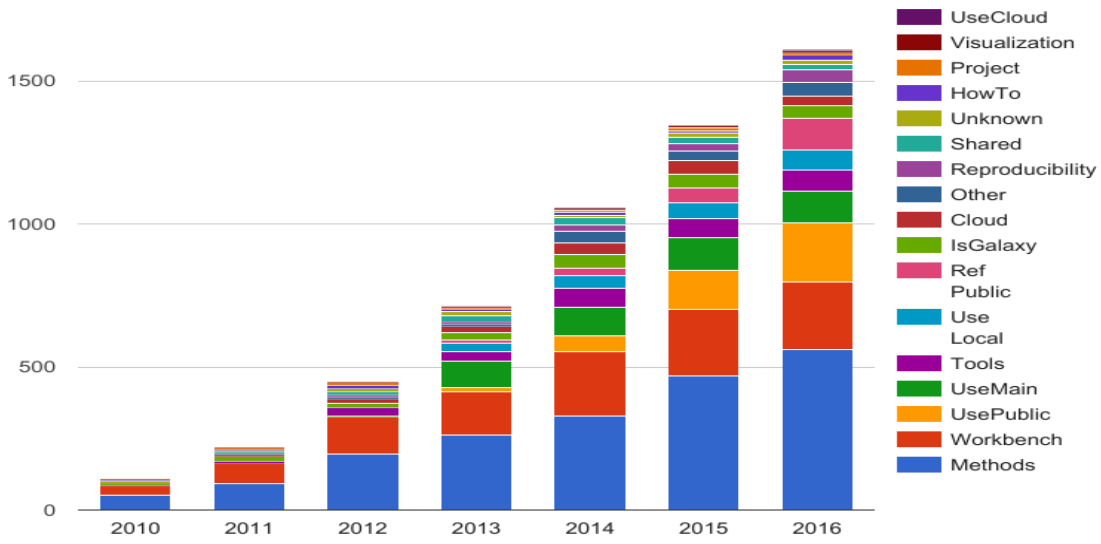


Figura 3.2: Publicaciones de la Comunidad Galaxy [67]

A pesar de la facilidad de uso de Galaxy, existen limitaciones importantes al ejecutar aplicaciones de workflows independientes en paralelo. Hemos encontrado una falta de apoyo, para definir la procedencia de los datos, que hacen que los resultados de los experimentos sean difíciles de reproducir de forma más fiable y rápida. Como así también el uso adecuado de las jerarquías de almacenamiento en plataformas del tipo clúster.

El objetivo en esta sección es el de explicar la funcionalidad de Galaxy y su extensión para ejecutar en un clúster de HPC un Planificador Consciente del Almacenamiento para Multiworkflow.

Revisión de Galaxy

Galaxy es una plataforma abierta, basada en el entorno web para realizar tareas científicas, y se muestra en la figura 3.3, donde podemos observar tres secciones principales; en el lado izquierdo vemos una lista de posibles herramientas

disponibles para el usuario. En el centro, una pizarra en la que la cadena de ejecuciones dependientes, está diseñada de acuerdo con los datos proporcionados por las aplicaciones anteriores. Al lado derecho, se puede distinguir el historial de ejecuciones e inspección de resultados. Así podemos ver cómo los usuarios pueden hacer uso de archivos de entrada y utilizar la interfaz web para interactuar con las aplicaciones. Sin embargo, la debilidad principal de la plataforma, es la capa de ejecución que no es capaz de administrar la jerarquía de almacenamiento para localizar la entrada compartida y archivos temporales. Aunque es posible definir los directorios de ubicación de datos iniciales, estos no se pueden mover.

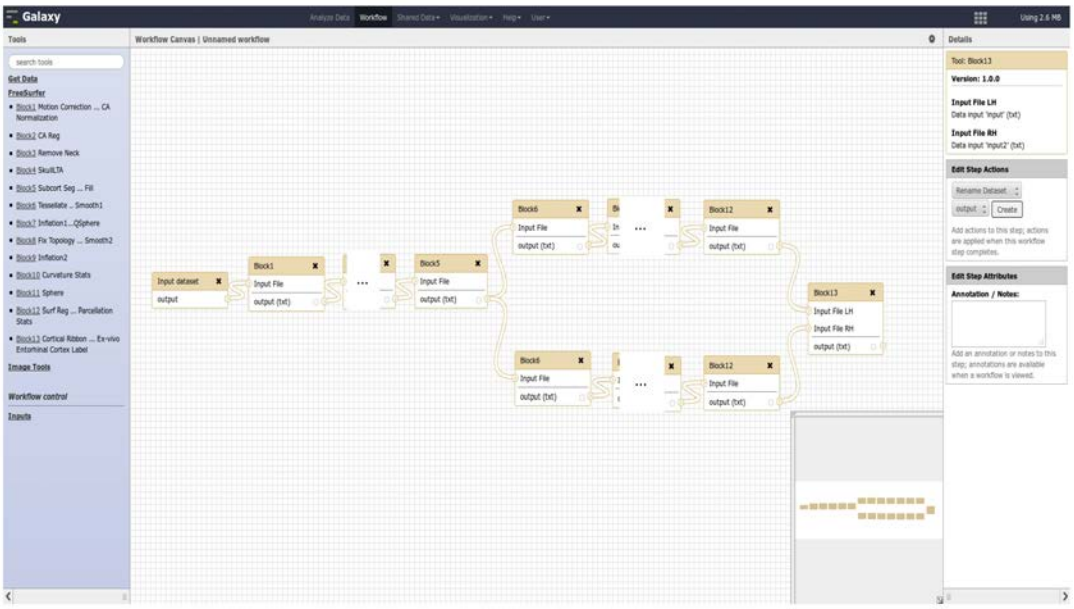


Figura 3.3: Herramienta Galaxy

A partir de la construcción de todo el ambiente de ejecución del experimento, se van guardando historiales de cada etapa de la ejecución. Este historial tiene el potencial de generar un esqueleto de ejecución para ser reutilizado a partir de ello, y ser exportado a otras instancias donde mover datos sea más complicado que mover una instancia de Galaxy para su ejecución y reproducción.

Los historiales generados pueden ser tratados como ejecuciones parciales del workflow, para la preparación de elementos de entrada a las siguientes etapas de

otros workflows más complejos.

Para definir el workflow en Galaxy tenemos que aclarar algunos puntos, como lo son: el origen de los datos, desarrollo del análisis, el historial de los datos y el almacenamiento. Los datos de entrada deben estar, de ser posible, en un formato tabulado, ingresados a partir de una fuente FTP, una dirección (URL), un servicio web o un repositorio propio de datos.

El desarrollo del análisis puede ser ejecutado en servidores públicos, nubes o localmente.

En consecuencia, los pasos básicos de ejecución de una aplicación en Galaxy serían:

- HTTP request
- Identificar la aplicación en uso
- Leer la descripción de la aplicación
- Encolar la ejecución
- Recuperar resultado
- Representar el resultado

Una vez ejecutado estos análisis, los resultados son almacenados en bases de datos de Galaxy.

Desde el punto de vista de la estructura del software, podemos describir la plataforma Galaxy como un conjunto de 3 capas: primero la interfaz web, donde los usuarios definen implementaciones funcionales de sus experimentos. En segundo lugar, la creación de trabajos, prepara y traduce cada una de las tareas con la estructura: nombre de la aplicación + parámetros de entrada + parámetros de salida; y finalmente se coloca en una lista de ejecución, ordenada según la dependencia de datos especificada en el modelo conceptual del workflow. Tercero, la lista de ejecución de aplicaciones es utilizada por la capa de ejecución de trabajos, donde las aplicaciones se envían con llamadas del sistema operativo a la plataforma, una a la vez.

La conectividad entre estas 3 capas se muestra en la Fig. 3.4, donde se extiende Galaxy a un entorno de clúster en el que la ejecución de la aplicación es llevada a cabo por varios nodos de cómputo. Describiremos en la sección 3.3 como fueron implementados los módulos representados en cuadros de color naranja, para lograr que las Políticas de Reconocimiento de Patrones y Administrador de Recursos en la Capa de Creación de Trabajo, puedan ser integradas a la política de planificación consciente de almacenamiento.

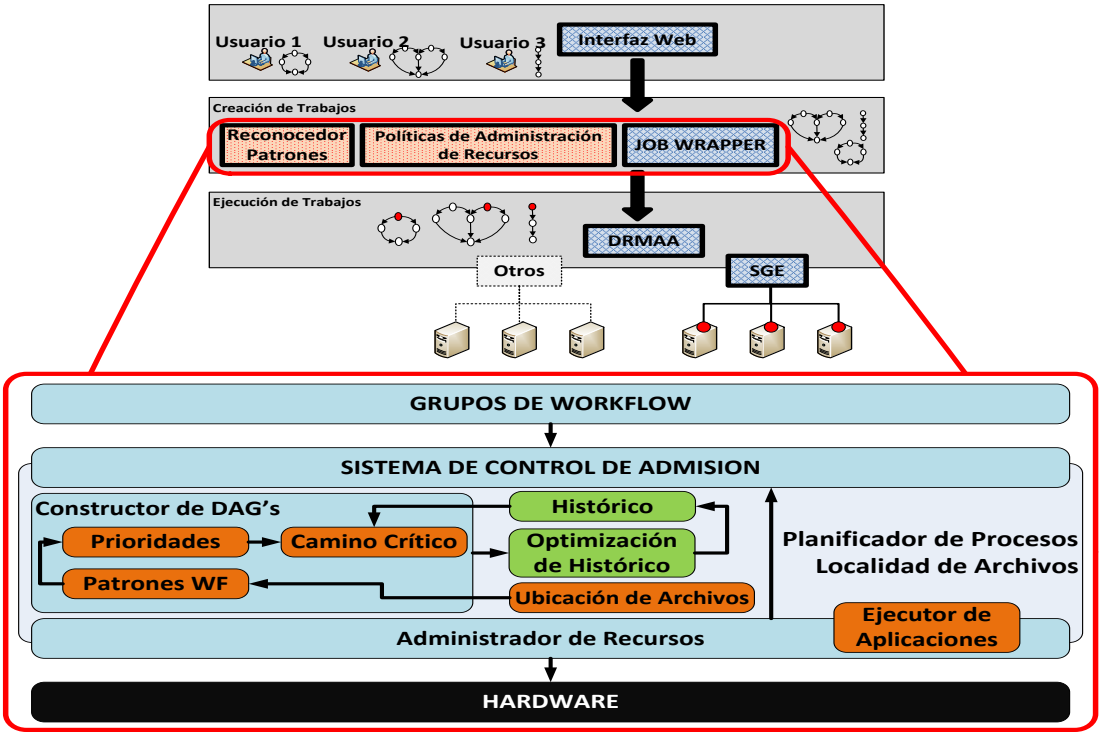


Figura 3.4: Capas Galaxy

3.3 Planificador Consciente de Almacenamiento en Galaxy

La arquitectura extendida del sistema Galaxy tiene un modelo cliente / servidor. El cliente tiene una interfaz web fácil de usar para crear, ejecutar y listar workflows. Una vez creado un workflow, se envía al servidor donde se implementa el planificador

y se guarda una copia del workflow conceptual en un archivo XML, o en una base de datos postgres con todos los atributos de las aplicaciones, y un registro de resultados históricos. Los tamaños de datos, la ubicación y el tiempo de ejecución se guardan también en una base de datos.

Con el fin de poder desarrollar nuestra propuesta de mejora a un entorno de computación escalable, como los clústers, tenemos que ampliar la capacidad de interactuar con el entorno de ejecución. Presentamos el Distributed Resource Application API (DRMAA), una API orientada a la gestión de diferentes gestores de recursos distribuidos, que nos da independencia del DRM utilizado en la plataforma.

El gestor de colas y el planificador de aplicaciones son proporcionados por Sun Grid Engine (SGE) y para dar capacidad multiusuario usamos el mecanismo de autenticación LDAP, gracias al cual también tenemos acceso a sistemas de archivos distribuidos como NFS.

Con esta extensión, tenemos varias ventajas importantes: ejecución paralela de workflows, multiusuario y multinstancias, acceso a datos distribuidos y lo más importante, la capacidad de implementar nuestra propia política de planificación.

Por lo tanto, tenemos una plataforma que integra varias tecnologías emergentes (GALAXY + DRMAA + SGE y LDAP + NFS). Ver el esquema de la plataforma en la figura 3.5.

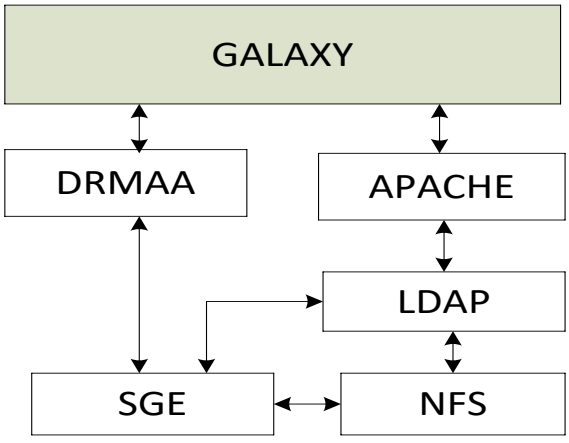


Figura 3.5: Galaxy Versión Extendida

La implementación de la política de planificación en Galaxy, se ha modificando el sistema de envío. A diferencia de los envíos de aplicaciones, secuenciales al sistema, esperamos a que Galaxy envíe un grupo de workflows y posteriormente los gestionamos como un grupo de workflows listos para ejecutarse.

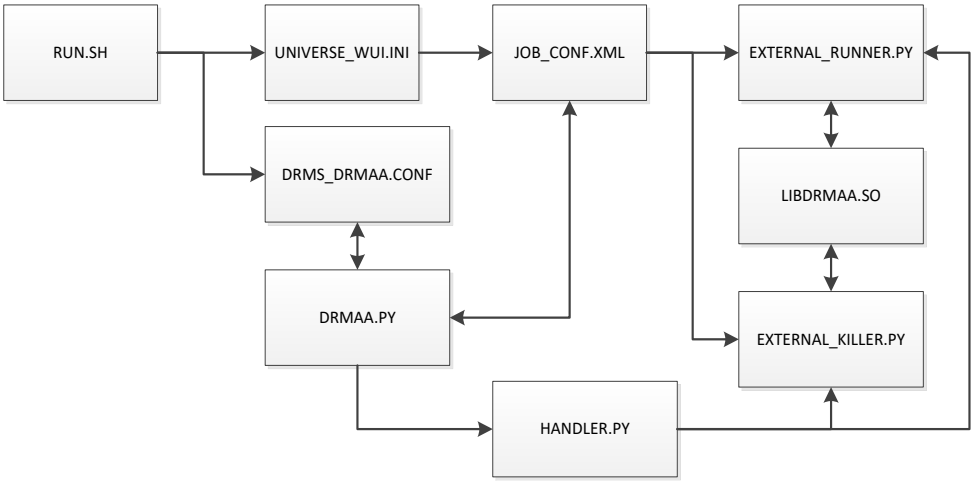


Figura 3.6: Archivos de configuración Galaxy

Para lograr introducir el planificador, necesitamos introducir modificaciones en los archivos de configuración de Galaxy mostrados en la figura 3.6:

- Run.sh: es el encargado de iniciar el *Daemon* de Galaxy. Run.sh además lee dónde están los archivos de configuración del sistema y otras variables globales necesarias para ejecutar la herramienta. Hace referencias a archivos externos de configuración extra.

- Job_config.xml: definimos la colección de adaptadores necesarios. El uso de DRMAA, LDAP y Runners externos para enviar aplicaciones al clúster se hacen aquí.

- Drmaa.py: script encargado de generar el archivo con todos los atributos necesarios para ser interpretado por un administrador de recursos como slurm o sge, empaqueta los valores en un archivo.sh

para que sea enviado a la cola de ejecución. Se encuentra en: /home/galaxy/galaxy-dist/lib/galaxy/jobs/runners/drmaa.py

- Handler.py: script desarrollado con 3 funciones principales. Enviar tareas al manejador de recursos, eliminar tareas del manejador de recursos y el control de tareas en el administrador de recursos. Se encuentra en: /home/galaxy/galaxy-central/lib/galaxy/jobs/handler.py

- Drmaa_external_runner.py: ejecuta efectivamente los archivos.sh generados por galaxy en la cola de trabajos.

- Drmaa_external_killer.py: elimina las tareas y los archivos.sh generados por galaxy en la cola de trabajos. Se encuentran en: /home/galaxy/galaxy-central/scripts

En el índice A, encontraremos ejemplos de cada uno de estos archivos de configuración.

Utilizaremos BioBlend [65], una API de Galaxy para analizar los workflows y los registros históricos guardados. Con esta API, podremos descubrir los atributos necesarios para ejecutar nuestro planificador CPFL:

- El nombre y la versión de la aplicación a ejecutar: nos permite tener control sobre la ejecución, ya que tenemos una caracterización de cada una de las aplicaciones que pueden estar en el sistema.

- Localización de archivos de entrada y salida: intentamos evitar la difícil tarea de definir manualmente la ubicación de los archivos necesarios para la ejecución de las aplicaciones. Implementamos una política para asegurar que los archivos estarán en almacenamientos de baja latencia.

- Tiempo aproximado de ejecución: al principio son proporcionados por el usuario experimentado o basados en nuestros experimentos históricos previos que nos proporcionan un cierto nivel de precisión.

- Número de nodos hijos y distancia al nodo final: permite tener un control constante sobre el patrón del workflow que se está ejecutando y una vista previa

de cómo y qué aplicaciones deben ejecutarse con mayor y menor prioridad teniendo en cuenta la entrada compartida de archivos.

Hemos incluido un nuevo archivo python, CPFL.py que es el propio código del planificador. La implementación funciona como una capa entre Galaxy y la plataforma. La extensión invocará funciones DRMAA, tipo runJob (), implementando una ejecución de workflows por lotes.

Capítulo 4

Experimentación Realizada y Resultados

En este capítulo presentamos la experimentación realizada y los resultados obtenidos por el planificador propuesto para multiworkflow CPFL cuando se comparan sus prestaciones con otras políticas de planificación de aplicaciones utilizadas en clústers.

Presentamos el entorno de experimentación utilizado y realizamos una descripción del workflow sintético que utilizamos en los experimentos. Los resultados muestran que CPFL mejora el tiempo de makespan para multiworkflows, cuando planificamos grupos de workflows que comparten el mismo conjunto de archivos de entrada y temporales. Comparamos CPFL con políticas de planificación ya existentes en el simulador WorkflowSim, como HEFT, Min-Min, Max-Min, FCFS y Random. La comparativa ha mostrado una mejora de hasta el 70 % con la utilización de 2 niveles de la jerarquía de almacenamiento real con un error del hasta 3 % en entornos simulados de 1024 núcleos.

En primer lugar, evaluamos el rendimiento de la propuesta presentada respecto a la política HEFT, basada en lista utilizando un sistema de archivos distribuido NFS en el clúster local.

Posteriormente, se evalúa el impacto del sistema de almacenamiento jerárquico cuando no se tiene en cuenta el camino crítico del workflow. Para ello se plantea una planificación aleatoria de las aplicaciones, asignando archivos en

Ramdisk y/o disco local. Para finalizar, se realiza la comparativa evaluando el algoritmo CPFL considerando la localización de los ficheros en la Ramdisk local o en los discos locales de los nodos de nuestro cluster experimental.

4.1 Entorno de Planificación y Escenarios

El clúster utilizado para las pruebas tiene 32 nodos, donde cada uno de los nodos tiene una CPU con 4 cores de 2.0Ghz, 12GB de memoria principal y una Ramdisk local de 6.2GB. Los sistemas de archivos que estamos utilizando son NFS como sistema de archivos distribuido, EXT4 como sistema de ficheros local y TMPFS como Ramdisk.

En el clúster de referencia, gran parte del análisis de datos se realiza mediante la ejecución de aplicaciones bioinformáticas, pertenecientes a un workflow. En el ejemplo proporcionado por la figura 4.1 se describe un workflow bioinformático típico. La aplicación Fast2Sanger utiliza un archivo de consulta, el cual se transforma en un fichero de salida con formato tabulado. Las aplicaciones BWA y GatK utilizan el mismo archivo de referencia, para realizar el alineamiento de secuencias y búsqueda de variantes; y finalmente Samtools utiliza ambos archivos, para convertir el resultado final a un formato compacto.

Un rasgo representativo de los workflows de análisis bioinformático, es encontrar distintas aplicaciones que utilicen el mismo fichero como entrada. Algunas de las funciones de análisis y transformación de datos, típicamente realizadas por estos workflows son: el alineamiento de secuencias respecto a una referencia como el genoma humano, el análisis de variantes, y las transformaciones de formato de archivos comunes.

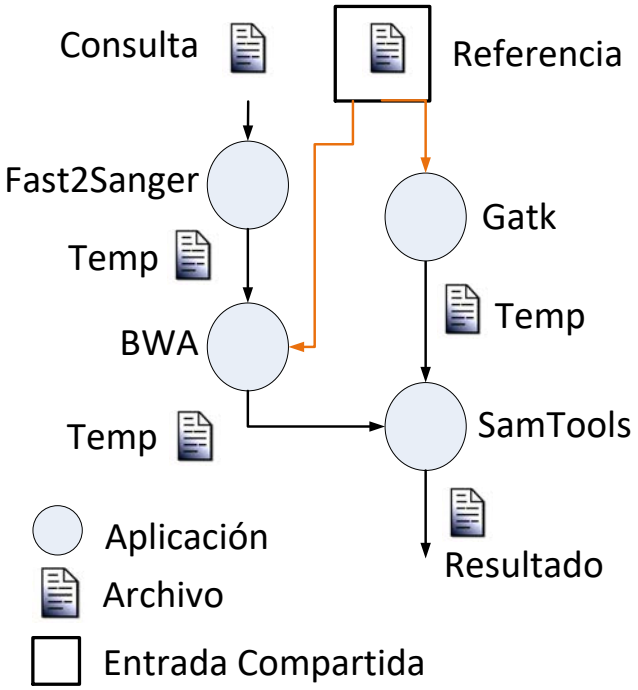


Figura 4.1: Ejemplo de archivo de entrada compartido (archivo de referencia) en un workflow bioinformático típico

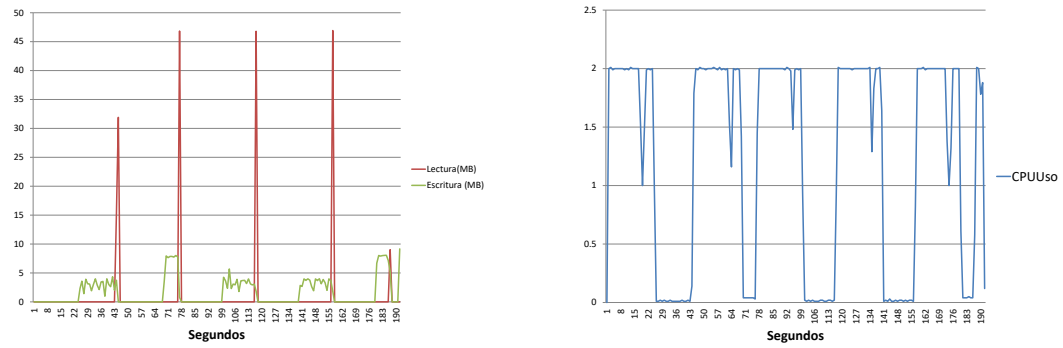
Hemos seleccionado las aplicaciones bioinformáticas que aparecen de forma frecuente en los workflows bioinformáticos analizados. Estas aplicaciones se han caracterizado, mediante el uso de la herramienta de monitorización HDF5, que posteriormente, almacena los datos de las operaciones en un repositorio de históricos de ejecución, para construir un workflow sintético con la misma estructura y comportamiento que el workflow bioinformático analizado.

4.2 Aplicaciones

Caracterizamos cuatro aplicaciones bioinformáticas: Fast2Sanger, BWA, GatK y Sam2Bam. Este conjunto de aplicaciones, nos proporciona una visión general de los tipos de aplicaciones que actualmente componen los workflows bioinformáticos. De esta manera, podemos verificar que el rendimiento de estas aplicaciones, está limitado por el acceso al sistema de E/S. El objetivo de esta caracterización es

la de proveer las bases del diseño para la implementación de nuevas heurísticas de planificación de workflows científicos que tengan en cuenta la jerarquía de almacenamiento.

Las figura 4.2 representa en el eje X un periodo de tiempo de instrumentación de la aplicación Fast2Sanger, cuyo objetivo es el de transformar los tipos de archivos de consulta a archivos tabulados. En el eje Y se representan el uso de CPU en la figura 4.2.b y los MB de escritura y lectura en el mismo periodo en la figura 4.2.a. Los picos indican que la lectura del archivo de entrada se realiza por secciones y posteriormente se escriben los resultados de la transformación. Los periodos de inactividad de CPU, son causados debido al tiempo de espera por entrada/salida generado en el sistema de almacenamiento.

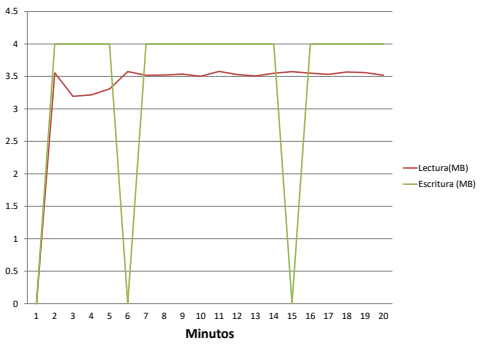


(a) Lectura/Escritura Disco

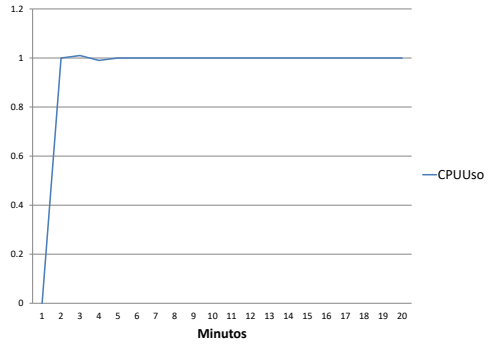
(b) Uso de CPU

Figura 4.2: Análisis Monitorización Fast2Sanger

La figura 4.3 ilustra el análisis de la aplicación BWA, cuyo objetivo es alinear los *reads* fuentes, con el genoma de referencia. Como podemos observar, en la figura 4.3.a esta aplicación realiza lecturas de tamaño considerable, y realiza cálculos comparativos para luego volcarlos a archivos de mayor tamaño, algo que podemos observar ya que el volumen de escritura es mayor al de escritura.



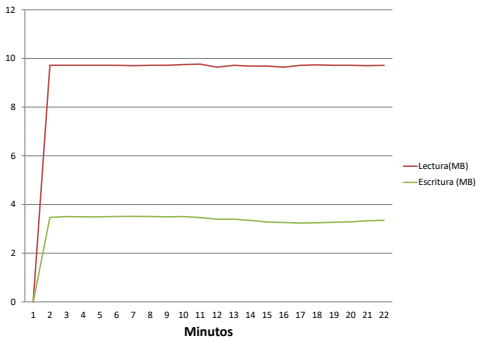
(a) Lectura/Escritura Disco



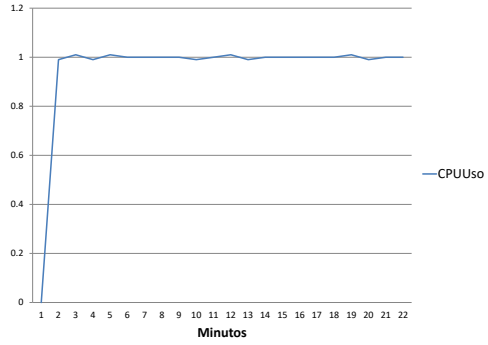
(b) Uso de CPU

Figura 4.3: Análisis Monitorización BWA

Las figura 4.4 corresponde a Sam2Bam, es una aplicación que se dedica a extraer resultados duplicados, y a comprimir los archivos a un formato binario y de menor tamaño, lo cual puede ser observado en la figura 4.4.a, dado que el volumen de escritura, es mucho menor que el de escritura.



(a) Lectura/Escritura Disco

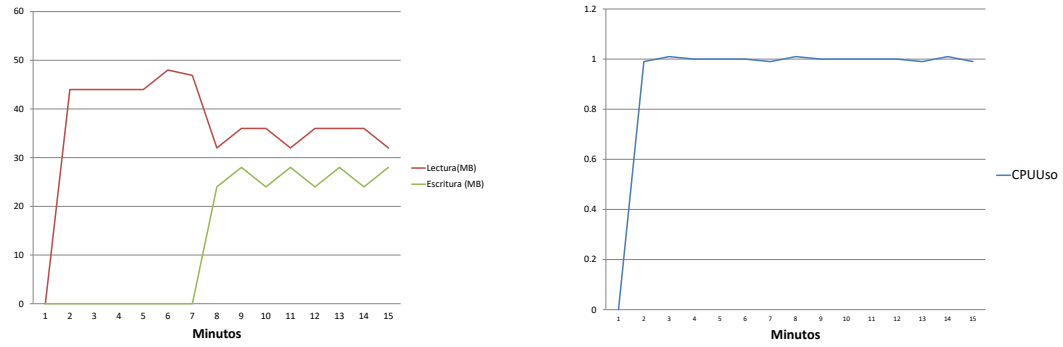


(b) Uso de CPU

Figura 4.4: Análisis Monitorización Sam2Bam

GatK es una aplicación que se utiliza para realizar la búsqueda de variantes en el genoma. La figura 4.5 representa en el eje X un periodo de tiempo de instrumentación. En esta figura 4.5.a podemos observar que GatK realiza lecturas

de gran volumen al inicio de su ejecución, ya que debe cargar toda la información de la base de referencia, y posteriormente, comparar todas las variantes posibles.



(a) Lectura/Escritura Disco (b) Uso de CPU

Figura 4.5: Análisis Monitorización GatK

Teniendo en cuenta que estas aplicaciones son representadas como nodos del workflow, y que sus dependencias son las precedencias entre ellas; hemos extraído un patrón del workflow estudiado para crear un workflow sintético que usaremos para nuestros experimentos. Las aplicaciones mostradas en la tabla 4.1, fueron seleccionadas a efectos de poder extrapolar los tiempos de ejecución relevantes, los costes de comunicación, y el uso de recursos en la tabla 4.2. Estas aplicaciones, son los elementos que componen los workflows sintéticos para nuestra experimentación.

Aplicaciones	Tiempo Ejec.(s)	E/S(MB) Lect.	Escr.	RSS (MB)	CPU Util(%)	Recurso Dominante	Objetivo
BWA (b,c,d)	11400	197	304	800	45	CPU	Alineación
Fast2Sanger (a)	1440	67	69	180	98	I/O	Transformación Formato
Sam2Bam (h)	1020	160	54	480	99	I/O	Transformación Formato
Gatk (e,f,g)	1380	10	47	300	99	CPU	Análisis Variantes

Tabla 4.1: Aplicaciones para Workflow Seleccionadas

A efectos de incrementar el abanico de aplicaciones de validación, hemos seleccionado workflows que utilizan aplicaciones de uso intensivo de datos, y que son de carácter público [13]. En este caso, hemos seleccionado Montage [10], un workflow de con E/S como recurso dominante utilizado en astronomía para generar

mosaicos del cielo., También hemos escogido el workflow Epigenomics [40] cuyo recurso dominante es la CPU.

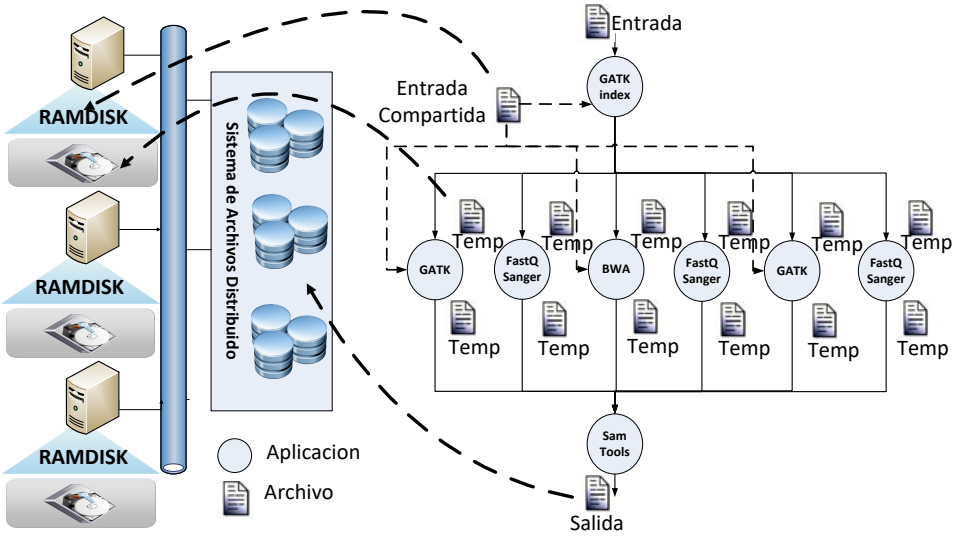
En la tabla 4.2 describimos una lista de rangos de métricas que describen los workflows Montage y Epigenomics, así como el workflow sintético desarrollado de acuerdo con la tabla 4.1.

WorkFlow	Tiempo Ejec.(s) Min-Max	E/S Lect(MB) Min-Max	E/S Escr.(MB) Min-Max	CPU Util(%) Min-Max
Montage	1-5	1-29	1-83	2-100
Epigenomics	1-4065	12-14676	10-103821	4-100
Sintético	7-179	2-201	0-305	3-98

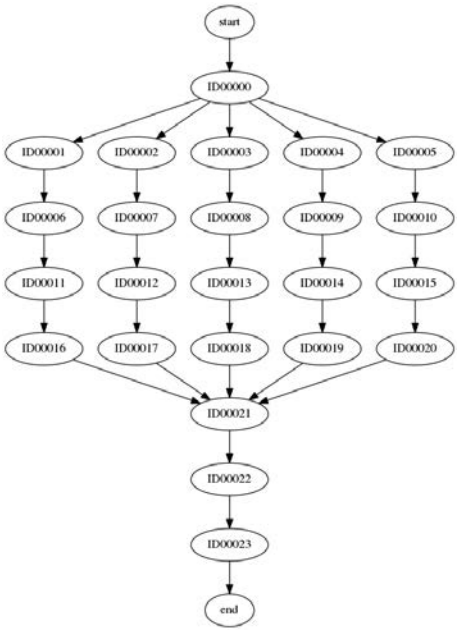
Tabla 4.2: Métricas de ejecución de Montage/Epigenomics/Sintético

En la figura 4.6 se muestran los esquemas de los grafos correspondientes a los workflows descritos. El primero de ellos hace referencia al workflow sintético utilizado, basado en el tipo de análisis de datos bioinformáticos realizado en el clúster. El segundo en la figura, es el workflow Epigenomics, y el tercero de ellos el workflow Montage. Estos dos últimos, se han obtenido de la galería de workflows[26]. En los tres casos de workflows, siempre podemos generar meta-workflows con nodos de inicio y fin.

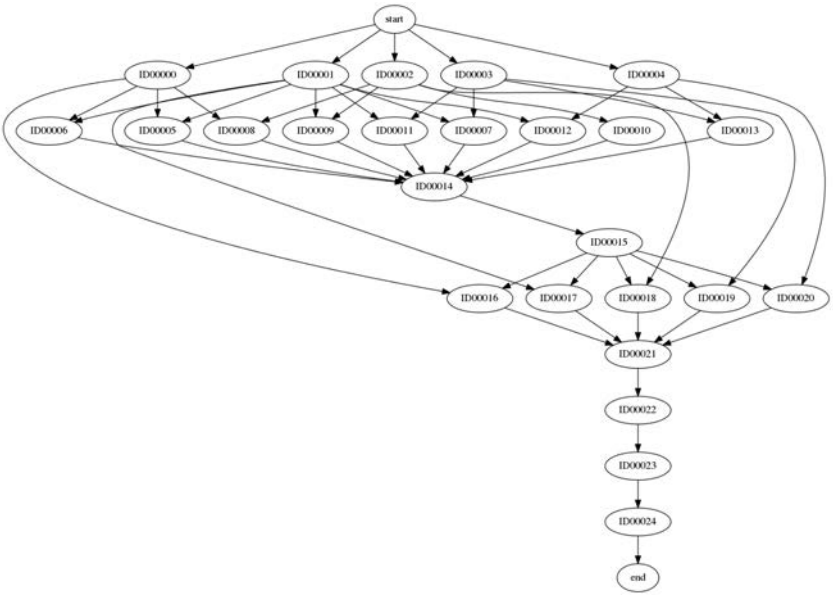
La figura 4.6.a nos ayuda a relacionar la figura 4.1 y la tabla 4.1 con la construcción del workflow sintético representado. Es decir, nos permite evaluar las opciones de localización de los datos de las aplicaciones del workflow a la vez que sus dependencias. En ese sentido, queremos representar cómo los archivos compartidos pueden ser ubicados en diferentes niveles de la jerarquía de memoria. En la misma figura representamos tres posibles tipos de archivos por aplicación (Entrada, Salida y Entrada/Temporal compartida), y su posible ubicación en la jerarquía de almacenamiento.



(a) Workflow Sintético



(b) Workflow Epigenomics



(c) Worklfow Montage

Figura 4.6: Patrones de diseño de los workflows usados para la experimentación

Al comienzo de la ejecución, los archivos de entrada estarán en el sistema de archivos distribuido donde se ha volcado el dataset de entrada. Los archivos de entrada compartidos, y los archivos temporales se localizarán en la Ramdisk local, o en el sistema de ficheros del disco local, según la disponibilidad de espacio. Los archivos de salida se encuentran directamente en el sistema de archivos distribuido. Las aplicaciones reales se representan con aplicaciones sintéticas, como se muestra en la misma figura, para simplificar la instalación de aplicaciones y el tiempo de ejecución excesivo mediante la extrapolación del tiempo de cómputo. Definimos un factor de paralelismo de 6, debido a que las aplicaciones bioinformáticas suelen tener archivos de entrada de 2GB, generando archivos temporales de hasta 6GB, que son suficientes para completar los 6GB disponibles en Ramdisk Local con 32 archivos de 2GB, uno por cada nodo (192GB es el total Tamaño de RamDisk que combina los 32 nodos del clúster). Los siguientes archivos de entrada no tienen espacio en el Ramdisk, y están ubicados en el disco local. En cada workflow sintético, introducimos 2 archivos de entrada compartidos, que producen 6 archivos temporales más de hasta 6 GB, lo que provoca que la Ramdisk y el disco local se saturen, obligando a usar NFS.

Se analizó la ejecución del workflow utilizando el makespan, como la principal métrica de rendimiento, para comparar la política motivo de estudio; utilizando el clúster experimental, definido anteriormente. Una vez analizados los resultados, utilizamos la herramienta WorkflowSim, para evaluar cómo CPFL podría escalar en clústers de mayor tamaño.

Se consideró la ejecución de cada uno de los workflows presentados, utilizando diferentes tamaños de archivo de entrada compartida. Nuestra experimentación consideró tamaños de archivo de entrada de 512 MB, 1024 MB y 2048 MB. Encontramos que los resultados obtenidos fueron muy similares, por lo que, mostraremos los datos obtenidos para un caso de uso, para cada combinación de tamaño de datos y recursos computacionales.

En todos los casos experimentales, se consideró la necesidad de planificar múltiples workflows. Para ello, hemos utilizado una lista de combinaciones como cargas de trabajo: (A) 50 workflow sintéticos (B) 10 workflows Epigenomics, (C) 10 workflows Montage. Para el caso de clúster real, hicimos hasta 10 ejecuciones con

una desviación de no más del 6 %. Para el caso B y C la ejecución realizada en un simulador, los resultados fueron los mismos siempre debido a la definición estática de atributos de WorkFlowSim.

4.3 Escenario Real

Esta experimentación, fue realizada con la ejecución en el clúster real de la carga de trabajo del tipo (A) 50 workflows sintéticos.

Iniciamos el experimento con 2 archivos compartidos, que corresponden a un archivo de referencia y un archivo de consulta. Preparamos la carga de trabajo con 5 grupos de 10 workflows cada uno, con los correspondientes archivos compartidos en cada grupo. Inicialmente, se realizan diferentes pruebas para 2 archivos de entrada compartidos de diferentes tamaños; de 512 MB, 1024 MB y 2048 MB cada uno.

Los resultados en los experimentos fueron similares, por lo que mostramos el caso de archivos con 2048MB de tamaño, en la figura 4.7 como ejemplo de la experimentación.

Considerando el uso del planificador HEFT en un sistema de archivos compartidos (NFS) que utiliza entre 8 y 128 cores, el CPFL obtuvo hasta un 50 % de mejor makespan cuando se usaron 2 niveles de almacenamiento (disco local + Ramdisk). Cuando sólo se utilizó un nivel de almacenamiento (disco local), CPFL era un 15 % más rápido.

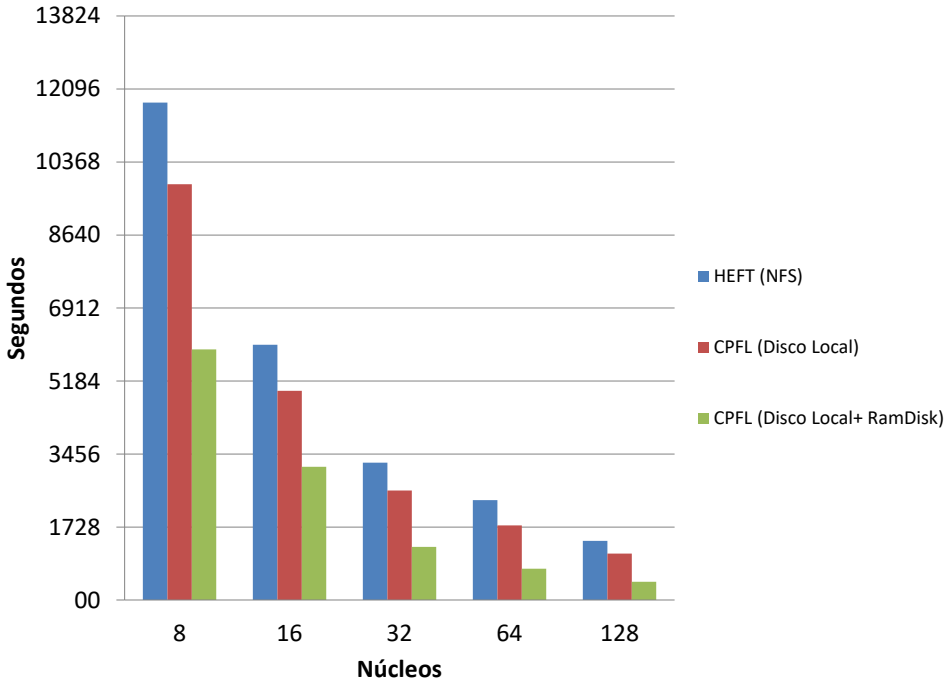


Figura 4.7: Makespan de Workflow Sintético con 2048MB de archivos de entrada compartido

Continuando con el experimento del tipo (A) con 50 workflows sintéticos, mostramos en la figura 4.8 los resultados obtenidos en el clúster de pruebas. Los resultados obtenidos fueron similares a los anteriores considerando 8, 16, 32, 64 y 128 cores. Presentamos los resultados para 128 cores donde la ganancia de makespan es hasta 70 % cuando los archivos de entrada compartidos son de 2048 MB y 40 % para 512 MB usando 2 niveles de almacenamiento (Ramdisk + Local Disk). De la misma forma, se obtienen ganancias del 20 % para archivos de 2048 MB y 12 % para archivos de 512 MB cuando usamos sólo disco local.

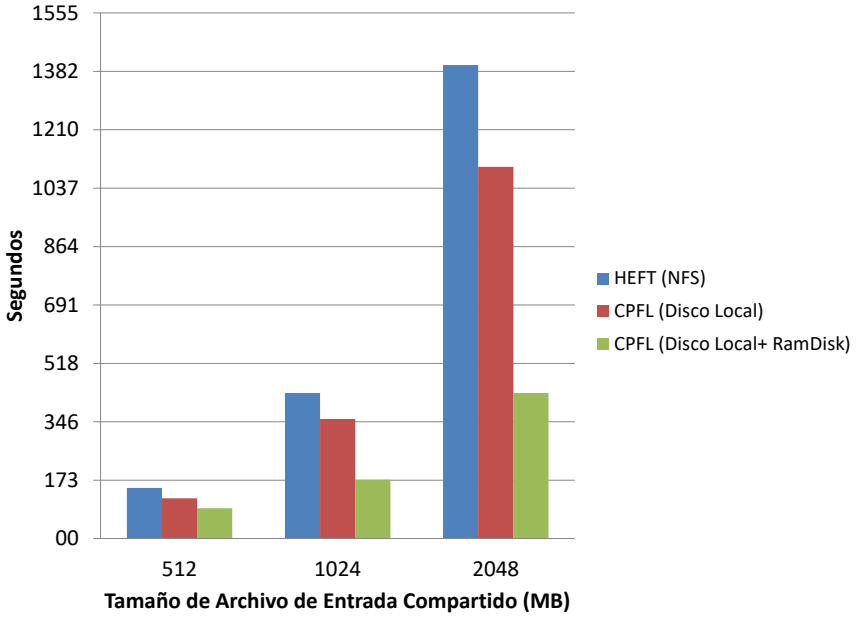


Figura 4.8: Makespan de Workflows Sintéticos en 128 cores

En la figura 4.9 podemos evaluar el impacto en el makespan de leer el mismo archivo repetidas veces durante la ejecución de multiworkflows. Ampliamos el número inicial de archivos de datos compartidos a cada uno de los 5 lotes de 10 workflows. Cada batch tiene ahora 2, 4, 8 y 16 archivos compartidos. Para 128 cores y tamaño de archivos de datos de 2048MB, tenemos una ganancia de hasta 78 % para 16 archivos compartidos en CPFL usando el enfoque de disco local y Ramdisk local en comparación con un almacenamiento HEFT en NFS.

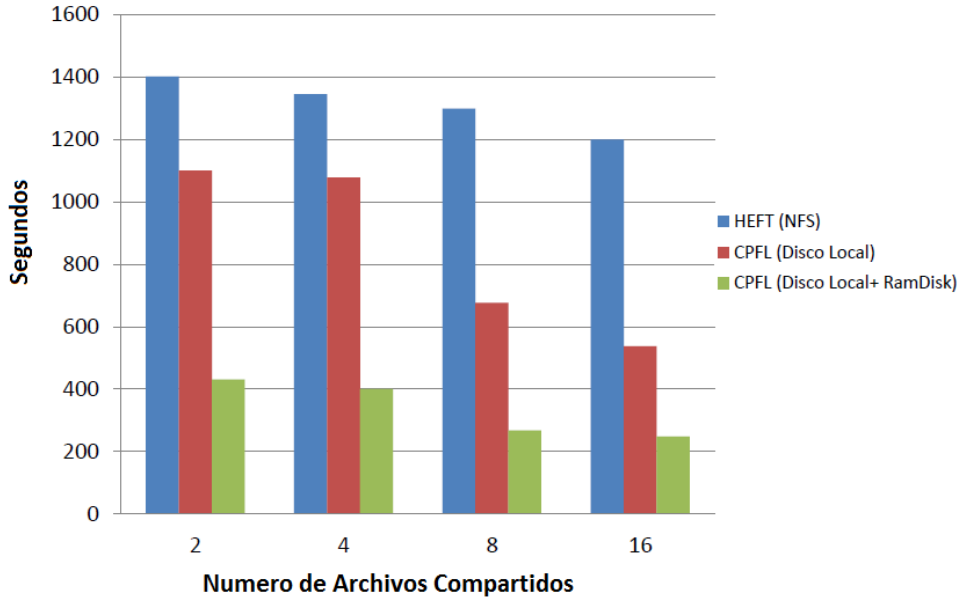


Figura 4.9: Makespan de workflow sintético en 128 núcleos con varios archivos compartidos de 2048MB

Como hemos podido probar, las ganancias en los workflows sintéticos están entre el 12 % y el 78 %, por lo que es necesario contrastar este rendimiento con otros workflows de referencia conocidos como Montage y Epigenomics.

En esta segunda parte de la experimentación en el cluster real, utilizamos como carga de trabajo un conjunto de workflows definidos como (B) 10 workflows de Epigenomics, y (C) 10 workflows de Montage. La carga de trabajo se dividió en 5 lotes de 2 workflows cada uno. Comparamos la ejecución del planificador HEFT respecto a CPFL. También consideramos el impacto de no utilizar Camino Critico (SCPFL) para la asignación de archivos en la jerarquía de almacenamiento. En SCPFL, las aplicaciones se asignan aleatoriamente a un recurso, pero siguen utilizando la jerarquía de almacenamiento.

Los resultados obtenidos al analizar Epigenomics y Montage fueron similares. En la figura 4.10 se representan los resultados de Epigenomics. La ganancia respecto a NFS y 8 cores es de 7.5 %. Utilizando Ramdisk y disco local como jerarquía de almacenamiento en 128 cores se obtiene hasta un 68 % de reducción con CPFL.

Finalmente, la ganancia llega hasta el 38 % cuando se utiliza disco local o disco local + RamDisk cuando usamos SCPFL.

De la ganancia de 68 %, CPFL tiene un 2 % del impacto, debido a la asignación previa de archivos al nodo de computo correcto, evitando tiempos de espera, mientras los archivos son copiados o movidos de un almacenamiento a otro. El disco local tiene un impacto del 11 % mientras, que el uso de Ramdisk tiene el 41 %. La combinación de Local disk + Ramdisk tiene un 46 %, debido al almacenamiento de alta velocidad, en un nivel superior de los sistemas de jerarquía de almacenamiento y la posibilidad de evitar cuellos de botella.

El workflow Montage, usando de 8 a 128 cores en el clúster de experimentación, tiene una ganancia en NFS y 8 núcleos de 9.5 %. El uso de la jerarquía de almacenamiento que combina el disco local y el Ramdisk da una ganancia en 128 cores de hasta el 71 %. Respecto a SCPFL, hemos medido una ganancia de hasta el 19 % debido al uso de disco local y el uso de disco local + RamDisk en 128 cores, porque al no utilizar un planificador para determinar qué nodo tiene el archivo, producen una desaceleración en la ganancia incluso cuando la jerarquía de almacenamiento está en uso.

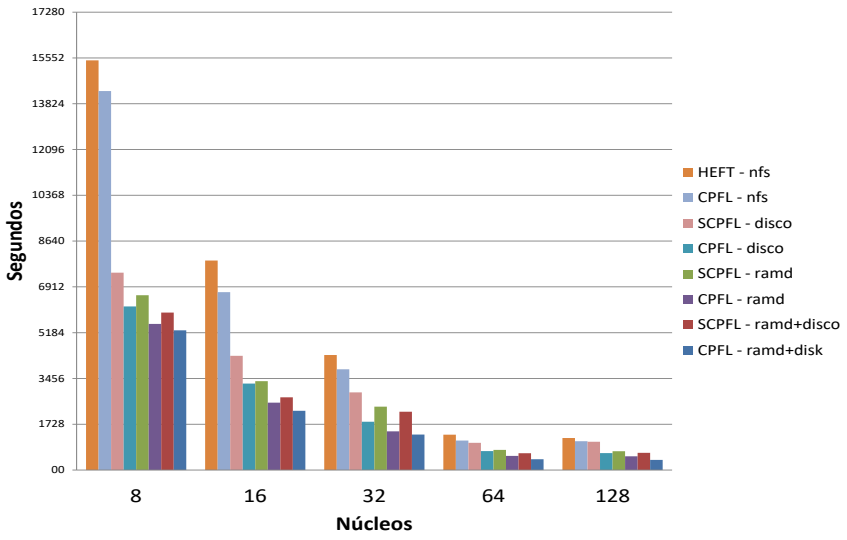


Figura 4.10: Resultados de makespan de Epigenomics en el clúster experimental

4.4 Análisis de Resultados de las Políticas de Planificación

Para corroborar la comparativa de la heurística propuesta, en la tabla 4.3 presentamos los resultados del análisis del sistema de acuerdo con la ejecución de la carga de trabajo de 5 lotes de 10 workflows cada uno con los 2 archivos compartidos correspondientes de 2048MB en cada lote.

Política	CPU Util	(%) Espera	Red/Total Reciv.	(MB) Env.	Disco/Total Lect.	(MB) Escr.	Mem Util. Usada	(MB) Libre
HEFT NFS	94	6.32	13255	3650	0.074	4.023	257-789	9981-10337
CPFL NFS	96	5.43	12848	3893	0.058	4.106	232-803	9923-11004
CPFL Disco	98	2.42	2373	239	1.838	7742	242-817	9846-10801
CPFL Disco + Ramdisk	99	0.08	712	57	0.269	51.068	9674-11325	122-1849

Tabla 4.3: Análisis comparativo de las Políticas de Planificación

En la tabla 4.3 se muestra un ejemplo de cómo afecta la jerarquía de almacenamiento, en la mejora del makespan, mediante la asignación de archivos previamente a la ejecución. En la experimentación realizada, comparado CPFL con HEFT, se ha encontrado una pequeña ganancia cuando ambos usan NFS como almacenamiento principal. Para el siguiente escenario, cuando CPFL utiliza sólo el disco local, el acceso a la red se reduce en un 70 %. Si CPFL utiliza disco local y Ramdisk, el acceso al disco se reduce en un 80 %, de 13255 MB a 712 MB en la transferencia de datos de la red en el momento de la ejecución. Según el experimento, 2 archivos de datos compartidos de 2048MB cada uno genera al menos 12GB de archivo de datos de salida. La columna *Mem Usada* refleja el tamaño total de los archivos de datos en un período de tiempo correspondiente al makespan.

La tabla 4.4 ilustra el caso de ejecutar 5 lotes de 10 workflows en un clúster de 32 nodos. La tabla muestra cómo CPFL Disk + Ramdisk utiliza los recursos. Los 32 nodos están ocupados. Tenemos un factor de paralelismo de 6 en cada workflow, es decir, 300 aplicaciones para 128 núcleos. Una vez más, dos archivos de entrada compartidos de 2048MB generan 12GB de archivos temporales compartidos. No podemos ver actividad significativa de la red porque el acceso de recursos relevante en CPFL son almacenamientos locales como Disco local y Ramdisk. La columna de

Disk/Total, muestra que en algunos casos el tamaño total de escritura de datos es cerca de 17 GB para cada lote. La columna *Mem Usada* muestra la utilización de cada lote, 12GB de archivos a los que se accede reduciendo el tiempo de acceso. Al ser el tamaño de Ramdisk local de 6.2GB, los archivos temporales no caben y por lo tanto, generamos paginación entre disco local y Ramdisk al aplicar el algoritmo de reemplazo.

Grupo	Nodos	Red/Total Reciv.	(MB) Env.	Disco/Total Lect.	(MB) Escr.	Mem Util. Usada	(MB) Libre
0	5,9,11,16,20,21,23,27,30	180	15	276	14379	9674-11280	123-1849
1	4,9,12,15,17,18,19,24,25,31	203	17	275	16071	9675-11325	122-1848
2	1,2,7,10,14,21,22,26,29,30	231	18	0.3	16435	9674-11247	270-1838
3	3,4,5,6,8,13,18,23,25,28	231	18	275	17442	10464-11325	123-1086
4	9,11,12,15,17,19,20,21,24,31	231.5	19	0.32	16725	9675-11211	281-1838

Tabla 4.4: Análisis de grupos de trabajo por Nodos en CPFL Disk + Ramdisk

4.5 Escenario Simulado

Para nuestro próximo experimento, consideramos una arquitectura de clúster de 512 cores. Para ello, proponemos el uso de WorkflowSim para evaluar la escalabilidad de CPFL para cientos de cores, y para un gran número de workflows que se ejecutan al mismo tiempo en el sistema.

En primer lugar, validamos los parámetros de la plataforma WorkflowSim, utilizando la misma carga de trabajo de nuestro escenario de clúster real anterior.

En la figura 4.11 mostramos el comportamiento del experimento del tipo (B), 10 workflows de Epigenomics en WorkflowSim. Podemos evaluar el porcentaje de error en la simulación con un máximo de 0.9 % para CPFL. Este error se debe a la implementación de CPFL en el simulador ya que algunos parámetros tales como el coste de comunicación no se aplican con precisión al planificador en ambientes simulados. La precisión se calcula de acuerdo con la fórmula de WorkflowSim Tiempo de simulación sobre tiempo real.

En caso de montaje, WorkflowSim tiene un porcentaje de error del 2 % para CPFL.

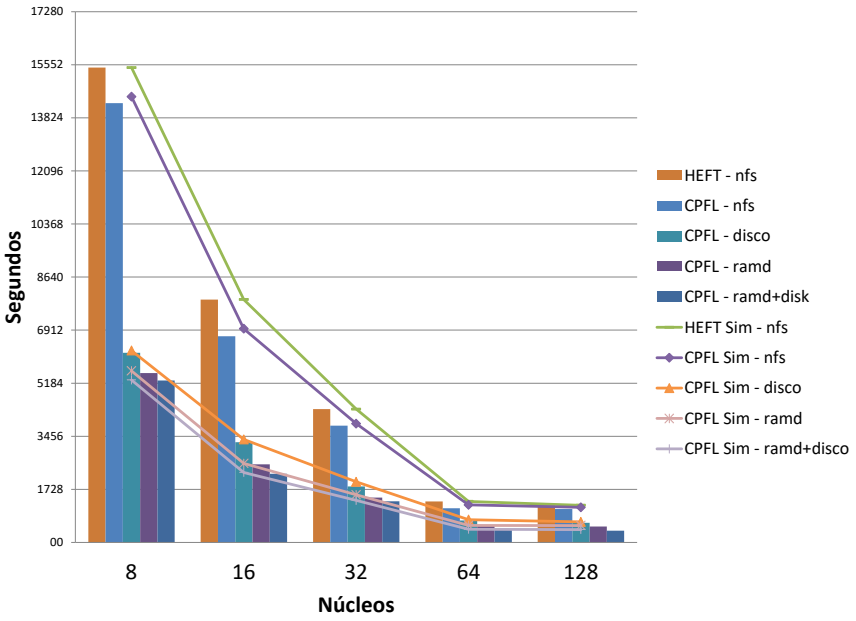


Figura 4.11: Resultados de Makespan de Epigenomics en el clúster simulado vs. Real

Después de haber establecido los parámetros y validado las condiciones de simulación en WorkflowSim, utilizamos la misma plataforma para experimentos de más volumen en plataformas más grandes.

Nuestro objetivo es evaluar la escalabilidad de CPFL en un entorno simulado de clúster. La prueba consiste en aumentar la cantidad de workflows de Epigenomics de 10 a 20 para analizar el rendimiento de CPFL con un número creciente de workflows y recursos. Definimos una nueva carga de trabajo de 20 workflows para proporcionar el estrés necesario para el clúster simulado.

Finalmente, la figura 4.12 presenta los resultados de la ejecución de 20 workflows de epigenomics en un clúster simulado de 8, 16, 32, 64, 128, 256 y 512 cores. Se comparan los resultados con otras heurísticas del estado del arte como HEFT. Como resultado, obtuvimos una ganancia de 1.69 % al comparar el uso de HEFT y CPFL en NFS en 64 cores. Usando CPFL en el disco local + Ramdisk el makespan se ha reducido a 4210 segundos. Debido a la nueva mejora, la ganancia

es del 53 % con respecto a HEFT.

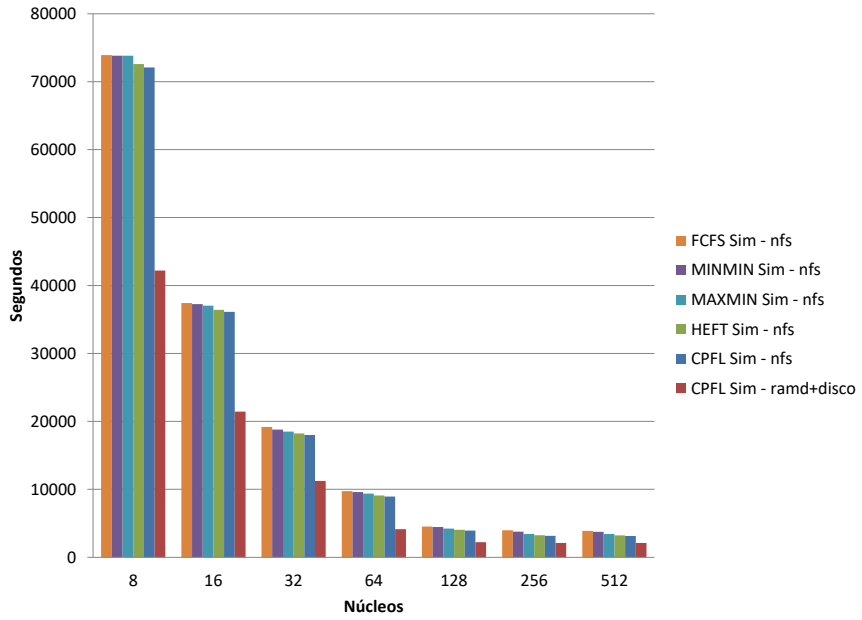


Figura 4.12: Resultados de makespan de Epigenomics usando CPFL hasta 512 cores versus otros planificadores

4.6 Resultados Simulados en 1024 núcleos

En la figura 4.13, presentamos la ejecución del flujo de trabajo sintético actual en WorkflowSim. Obtuvimos una ganancia del 3.4 % al comparar el uso de HEFT y CPFL en NFS en 64 núcleos; 5 % mejor que MINMIN; 4 % mejor que MAXMIN; 8 % con respecto a FCFS y 21 % con respecto a Random. Cuando se utiliza CPFL en disco local + Ramdisk + SSD local, el makespan se ha reducido a 4134 segundos. Por lo tanto, la ganancia obtenida es 54 % contra HEFT, 54 % contra MINMIN, 55 % con respecto a MAXMIN un 56 % con respecto a FCFS y hasta 62 % con respecto a Random.

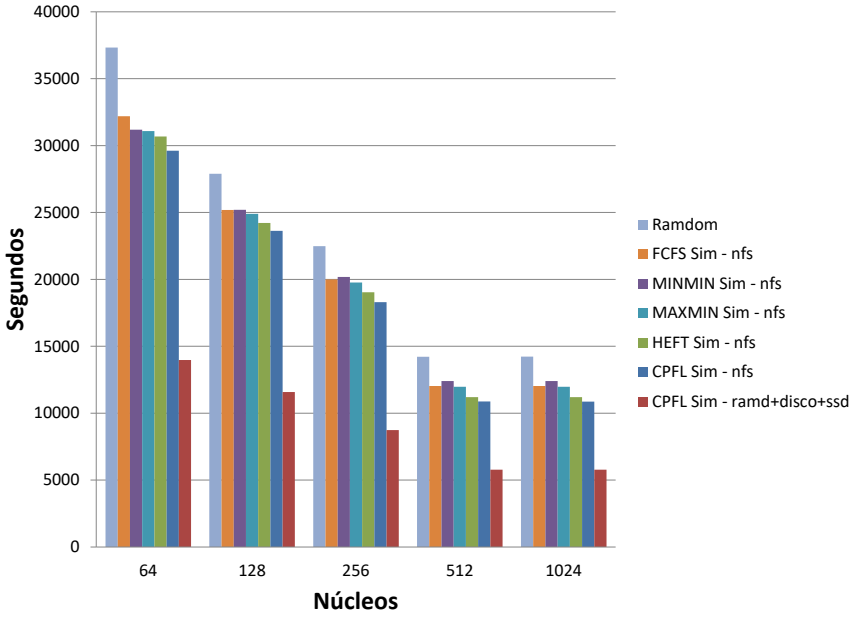


Figura 4.13: Makespan para un workflow Sintético simulado con el planificador CPFL hasta 1024 núcleos vs. otros planificadores

Figura 4.14 presenta los resultados de la ejecución de 20 workflows Epigenomic. En resumen, obtuvimos una ganancia de 1.69 % cuando comparamos usando HEFT y CPFL en NFS en 64 núcleos, un 6.9 % mejor que MINMIN, 4.6 % mejor que MAXMIN y 8.5 % con respecto a FCFS y 20 % respecto a Random. Usando CPFL en disco local + Ramdisk + SSD local el makespan se ha reducido a 4134 segundos. Debido a la nueva mejora, la ganancia es de 53 % frente a HEFT, 56 % contra MINMIN, 55 % respecto a MAXMIN y finalmente un 56 % respetando FCFS y hasta 63 % respecto a Random.

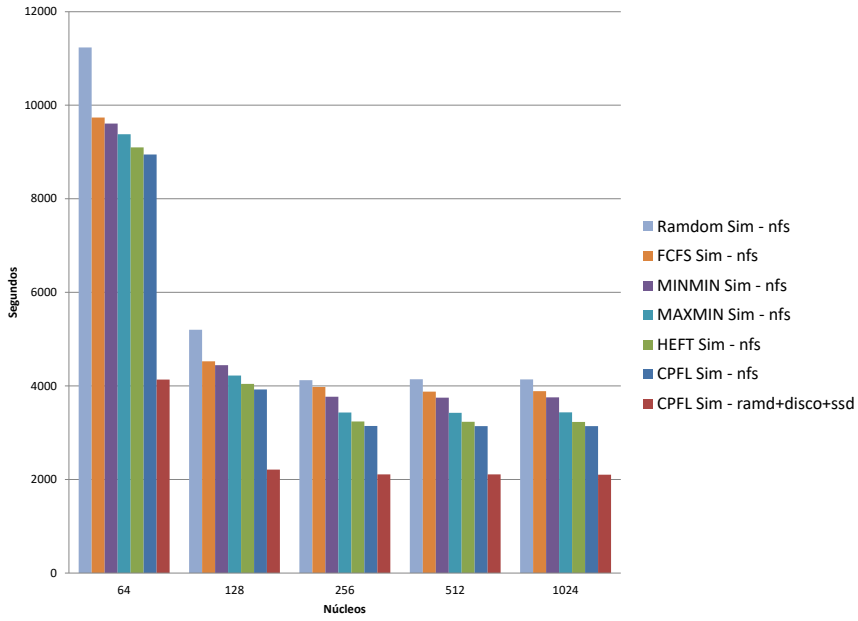


Figura 4.14: Makespan del workflow Epigenomics con un planificador CPFL hasta 1024 núcleos vs. otros planificadores

En la figura 4.15 presentamos el resultado de ejecutar 20 workflows Montage. En resumen, obtuvimos una ganancia del 8 % cuando comparamos usando HEFT y CPFL en NFS en 64 núcleos, el 21 % mejor que MINMIN, 16 % mejor que MAXMIN. Un 27 % con respecto FCFS y hasta 46 % para al Random. Usando CPFL en disco local + Ramdisk + SSD el makespan se ha reducido a 665 segundos. Debido a la nueva mejora, la ganancia es de 64 % contra HEFT, 69 % contra MINMIN, 66 % respecto a MAXMIN un 71 % respecto a FCFS y 79 % para al Random.

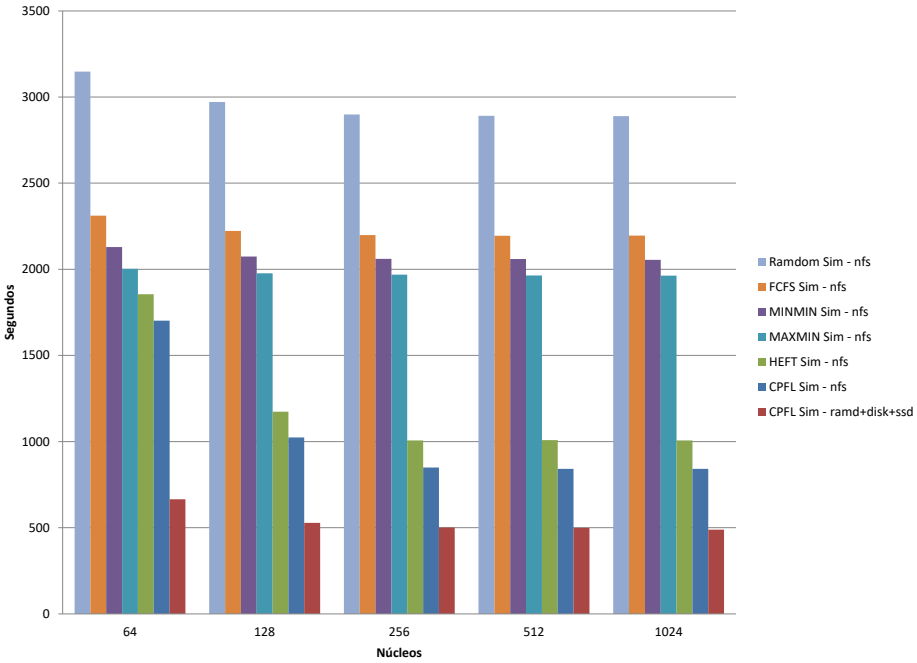


Figura 4.15: Makespan de workflow Montage con el planificador CPFL hasta 1024 núcleos vs. otros planificadores

Hemos propuesto escenarios de experimentación reales y simulados. Con la caracterización de aplicaciones bioinformáticas fuimos capaces de construir un workflow sintético capaz de comportarse como workflows bioinformáticos. Junto con el workflow sintético, utilizamos workflows utilizados en la literatura como Montage y Epigenomics para evaluar nuestra política de planificación. Con la ayuda del simulador WorkflowSim, además hemos evaluado la escalabilidad de la propuesta y la comparación con otras heurísticas de planificación. El siguiente capitulo, ofrece las conclusiones sobre nuestro trabajo y las lineas abiertas.

Capítulo 5

Conclusiones y Líneas Abiertas

En el presente trabajo, se han propuesto soluciones a las diferentes cuestiones planteadas, con el fin de desarrollar una política de planificación para múltiples workflows consciente del almacenamiento, en clústers basados en Galaxy. Una de las metas a conseguir por el planificador desarrollado (CPFL), reside en gestionar para múltiples workflows, las aplicaciones que comparten archivos de entrada y temporales, a efectos de mejorar el makespan de un conjunto de workflows, en un sistema de cómputo del tipo cluster; donde el cuello de botella del sistema lo generan las lecturas y escrituras de las aplicaciones a los dispositivos de almacenamiento masivo.

A continuación, se resumen los puntos tratados en el presente trabajo, incluyendo las aportaciones científicas a las que han dado lugar.

Para diseñar el planificador, evaluamos las características particulares de las aplicaciones que consumen y generan grandes volúmenes de datos, específicamente las aplicaciones bioinformáticas y como estas forman parte de los workflows científicos. Analizamos además, como afectan estas aplicaciones a un sistema de cómputo de altas prestaciones como lo son los clústers cuando carecemos de detalles relativos a la ubicación de los archivos.

El hecho de garantizar la localidad de los archivos antes de la asignación de aplicaciones a los nodos, genera tráfico en la red, debido a que hay accesos repetidos a disco para que los archivos sean copiados del almacenamiento del nodo de origen

al nodo de cómputo.

Ejecutar workflows con aplicaciones que comparten archivos de entrada hace que el problema sea aún mayor, porque el mismo archivo debe ser copiado a varios nodos de cómputo y aun mas, cuando los archivos temporales generados por estas aplicaciones deben ser copiados usando la red de comunicación del clúster a otros nodos.

Por otra parte, las aplicaciones que consumen grandes cantidades de datos y generan una gran cantidad de archivos intermedios, hacen que los discos del sistema de archivo distribuido se sobrecarguen. En los gestores tradicionales de workflows, los archivos son almacenados de forma aleatoria, tanto para lectura como para escritura. En estos sistemas el recurso dominante se vuelve por tanto el acceso a esos discos.

5.1 Evaluación de administradores de workflows científicos

Propusimos una nueva política de planificación de workflows, que intenta solucionar los problemas planteados en el análisis, y ha sido integrada a una plataforma de administración de workflows científicos como lo es Galaxy.

El análisis de distintos administradores de workflows científicos, la evaluación de los métodos para integrar nuevas aplicaciones en el administrador de workflows, y las diferentes tecnologías que pueden estar asociadas a la plataforma Galaxy para aplicaciones de uso intensivo de datos, en entornos de clúster fueron publicados en la Jornada de Paralelismo de España:

JP2014 - Jornadas de Paralelismo 2014:
Autores: César Acevedo, Porfidio Hernández, Antonio Espinosa, Gonzalo Vera, Javier Navarro, Jordi Delgado, Yolanda Vives y Manuel Delfino
Título: Plataforma de ejecución paralela en Workflows científicos en Galaxy.
Lugar: Valladolid, España.
Año: 2014.

5.2 Desarrollo de la política de planificación propuesta

En la propuesta, el planificador consciente del almacenamiento para multiworkflows, inicia el proceso a partir de la ubicación de los archivos de entrada en la jerarquía de almacenamiento del clúster previo al inicio de la ejecución, y en segundo lugar, en la agrupación en grupos de aplicaciones que utilizan el mismo archivo, para que puedan ser asignados y ejecutados en el mismo nodo donde los archivos residen.

La mejora en el tiempo de makespan para procesar grupos de workflows según la ubicación de los archivos compartidos de entrada y temporales; y de salida fueron publicados en la conferencia internacional Complexis 2016, donde fue elegido *Best Student Paper* e invitado a enviar una extensión del trabajo a una revista de alto impacto:

Complexis 2016 - Conference on Complex Information Systems 2016:
Autores: César Acevedo, Porfidio Hernández, Antonio Espinosa y Victor Méndez
Título: A Data-Aware Multiworkflow Cluster Scheduler[2].
Lugar: Roma, Italia.
Año: 2016.

El análisis del planificador consciente del almacenamiento para multiworkflows en clústers, fue probado con wokflows sintéticos, basados en las aplicaciones estudiadas y con Montage y Epigenomics, que son workflows que generan gran cantidad de archivos temporales. Presentamos escenarios donde los grupos de múltiples workflows hacen uso de la política de archivos compartidos en la jerarquía de almacenamiento compuesta por RamDisk, Disco Duro Local y Disco de Estado Solido Local. Además, presentamos resultados de escalabilidad del planificador en entornos de ejecución de más de 128 cores, así también una mayor cantidad de niveles en la jerarquía de almacenamiento.

Para la gestión de los archivos compartidos de entrada y temporales, propusimos utilizar una RamDisk como un nivel de almacenamiento temporal. Cada nodo del clúster proporciona una parte de su memoria principal para la construcción

de la RamDisk Local. Los archivos temporales generados por las aplicaciones del grupo de workflows, pasan a estar en lo que quede de RamDisk, en caso contrario se ubicaran en un nivel más bajo que sería el Disco Duro Local, o en el siguiente caso el sistema de archivos distribuido.

El criterio para seleccionar los archivos de entrada y temporales que serán ubicados en la RamDisk fue el tamaño, definido con 3 limites: Pequeño (menores a 1MB), Medianos (de hasta 512MB) y Grandes (mayores a 512MB). Además, se calcula el factor de compartición según la cantidad de aplicaciones que utilizan un archivo. Con prioridad se ubican los archivos de mayor tamaño y mayor factor de compartición.

Con los escenarios de prueba pudimos observar, que con el crecimiento del tamaño de los archivos a 2048MB, el makespan comienza a aumentar. Esto se debe a la necesidad de CPFL de encontrar la ubicación en la jerarquía de almacenamiento de archivos grandes. Para mejorar el makespan introdujimos diferentes niveles en la jerarquía.

Esta parte del trabajo fue propuesta y aceptada en una revista de alto impacto como lo es Future Generation Computer Systems:

FGCS - Future Generation Computer Systems:
Autores: César Acevedo, Porfidio Hernández, Antonio Espinosa y Victor Méndez
Título: A Critical Path File Location (CPFL) algorithm for data-aware multiworkflow scheduling on HPC clusters[3].
DOI: https://doi.org/10.1016/j.future.2017.04.025
Impact Factor: 2.768
Quartile: 1
Año: 2017.

Además, dada la importancia de la revista FGCS, fuimos invitados a participar de una mesa de dialogo con otros expertos del área, en un fórum especializado en publicaciones de alta calidad:

HQJ Forum 2017 - High Quality Journal Forum 2017:
Autores: César Acevedo, Porfidio Hernández, Antonio Espinosa y Victor Méndez
Título: A Critical Path File Location (CPFL) algorithm for data-aware multiworkflow scheduling on HPC clusters.
Lugar: Porto, Portugal.
Año: 2017.

5.3 Evaluación de la política de planificación en WorkflowSim e integración de la política en Galaxy

Finalmente, se desarrolló la integración del planificador al simulador WorkflowSim, con las extensiones adecuadas para implementar políticas de planificación consciente del almacenamiento para múltiples workflows. Con el simulador, probamos la escalabilidad y la comparación con otras heurísticas de planificación clásicas de la literatura. El trabajo fue publicado en el congreso internacional Complexis 2017:

Complexis 2017 - Conference on Complexity, Future Information Systems and Risk 2017:
Autores: César Acevedo, Porfidio Hernández, Antonio Espinosa y Victor Méndez
Título: A Data-Aware Multiworkflow Scheduler for Cluster on WorkflowSim[4].
Lugar: Porto, Portugal.
Año: 2017.

Finalmente para evaluar la aceptación del planificador CPFL en la comunidad científica bioinformática, hemos integrado la solución a un sistema de administración de multiworkflows llamado Galaxy, el cual está preparado para ser instalado en entornos del tipo Escritorio, Clústers y Nubes.

Para poder integrar dicho planificador, hemos recurrido al uso de técnicas de

scripting junto con APIs específicas de la comunidad Galaxy como lo es BioBlend. Esta API proporciona acceso a datos históricos de ejecuciones y estructuras de los workflows desarrollados, y diferentes parámetros de ejecución de aplicaciones.

5.4 Líneas Abiertas

Pocos trabajos relacionados con la planificación de múltiples workflows, han tenido en cuenta la localización de los ficheros de datos de entrada, salida e intermedios, en entornos de clusters compartidos. Aspecto que ha demostrado ser un factor crítico, en la ejecución de flujos de trabajo en aplicaciones intensivas de datos, como es el caso que se asume en este trabajo. En este sentido, proponemos las siguientes líneas de trabajo¿

- Como continuidad de esta investigación, se ha visto la necesidad de evaluar políticas de reemplazo en la jerarquía de almacenamiento. En el mismo sentido, se hace necesario evaluar la RamDisk cuando los archivos sean de mayor tamaño y el abanico de aplicaciones más variado.
- Nuestra propuesta propone en primera instancia, enfoques basados en la composición de WFs con el fin de combinar múltiples DAG en uno único, antes de aplicar algoritmos de naturaleza estática, considerando que todos los parámetros representativos de los diferentes grafos son conocidos de antemano. Sería interesante el plantear, la implementación de una función de admisión de peticiones a nivel dinámico, que permita ajustar la cantidad de workflows en el sistema.
- Evaluar la política de planificación en entornos con sistemas de archivos NFS4 y Lustre, podría ser otra de las líneas de continuación del presente trabajo.
- Considerar la posibilidad de implementar una RamDisk distribuida, podría permitir incrementar el uso de archivos de mayor tamaño, así como decrementar la utilización de los algoritmos de reemplazo, para los diferentes niveles de la jerarquía del almacenamiento.

- Integrar a la herramienta de administración de workflows la tolerancia a fallos, garantizando la persistencia de datos en la jerarquía de almacenamiento. Estudiar como las copias de los archivos en los niveles de la jerarquía del almacenamiento, aseguran la redundancia y facilita los algoritmos de recuperación de datos. Así como sería deseable, abordar el problema de la volatilidad de la RamDisk.

- Con DRMAA [72] como API de comunicación a los administradores de recursos distribuidos, les permite en la versión implementada actualmente, utilizar SGE pero podríamos utilizar otros gestores de recursos, lo que nos facilitaría usar una máquina de estados que asegurase la tolerancia de fallos, a través de varias tecnologías de cómputo.

- En relación al simulador de workflows, se podría ajustar a distintos tipos de sistemas de archivos distribuidos. Igualmente, sería de especial relevancia la implementación en WorkflowSim de una función de predicción de costos de cómputo, basada en la ubicación de los archivos en la jerarquía de almacenamiento.

Bibliografía

[1] Mohamed Abouelhoda, Shadi Issa, and Moustafa Ghanem. Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. *BMC Bioinformatics*, 13(1):77, jan 2012.

[2] Cesar Acevedo, Porfidio Hernandez, Antonio Espinosa, and Victor Mendez. A data-aware multiworkflow cluster scheduler. In *Proceedings of the 1st International Conference on Complex Information Systems*, pages 95–102. SCITEPRESS, 2016.

[3] César Acevedo, Porfidio Hernández, Antonio Espinosa, and Víctor Méndez. A critical path file location (cpfl) algorithm for data-aware multiworkflow scheduling on hpc clusters. *Future Generation Computer Systems*, 2017.

[4] Cesar Acevedo, Porfidio Hernandez, Antonio Espinosa, and Victor Mendez. A data-aware multiworkflow scheduler for cluster on workflowsim. In *Proceedings of the 2st International Conference on Complex Information Systems*. SCITEPRESS, 2017.

[5] Enis Afgan, Dannon Baker, Nate Coraor, Brad Chapman, Anton Nekrutenko, and James Taylor. Galaxy cloudman: delivering cloud compute clusters. *BMC bioinformatics*, 11(12):S4, 2010.

[6] Hamid Arabnejad and Jorge G Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):682–694, 2014.

[7] J Barbosa and AP Monteiro. A list scheduling algorithm for scheduling multi-user jobs on clusters. *High Performance Computing for Computational Science-VECPAR 2008*, pages 123—-136, 2008.

[8] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/pacts: a programming model and execution framework for web-scale analytical processing. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 119–130. ACM, 2010.

[9] John Bent, Douglas Thain, Andrea C Arpaci-Dusseau, Remzi H Arpaci-Dusseau, and Miron Livny. Explicit control in the batch-aware distributed file system. In *NSDI*, volume 4, pages 365–378, 2004.

[10] G Bruce Berriman, Ewa Deelman, John C Good, Joseph C Jacob, Daniel S Katz, Carl Kesselman, Anastasia C Laity, Thomas A Prince, Gurmeet Singh, and Mei-Hu Su. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 221–232. International Society for Optics and Photonics, 2004.

[11] Michael R Berthold, Nicolas Cebron, Fabian Dill, Thomas R Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Kilian Thiel, and Bernd Wiswedel. Knime-the konstanz information miner: version 2.0 and beyond. *AcM SIGKDD explorations Newsletter*, 11(1):26–31, 2009.

[12] Aprigio Augusto Lopes Bezerra. *Planificacion de trabajos en clusters Hadoop compartidos*. Tesis Doctoral-Universidad Autonoma de Barcelona, Barcelona, 2014.

[13] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *2008 Third Workshop on Workflows in Support of Large-Scale Science*, pages 1–10. IEEE, 2008.

[14] Luiz Fernando Bittencourt and Edmundo RM Madeira. Towards the scheduling of multiple workflows on computational grids. *Journal of grid computing*, 8(3):419–441, 2010.

[15] James Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task scheduling strategies for workflow-based applications in grids. In *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, volume 2, pages 759–767. IEEE, 2005.

[16] R Bolze, F Desprez, and B Isnard. Evaluation of Online Multi-Workflow Heuristics based on List-Scheduling Algorithms. *Gwendia report L*, 2009.

[17] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.

[18] Marc Bux and Ulf Leser. Parallelization in Scientific Workflow Management Systems. (1):24, mar 2013.

[19] Rajkumar Buyya. High performance cluster computing: Architecture and systems, volume i. *Prentice Hall, Upper SaddleRiver, NJ, USA*, 1:999, 1999.

[20] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[21] Louis-Claude Canon, Emmanuel Jeannot, Rizos Sakellariou, and Wei Zheng. Comparative evaluation of the robustness of dag scheduling heuristics. In *Grid Computing*, pages 73–84. Springer, 2008.

[22] Victor Chang and Gary Wills. A model to compare cloud and non-cloud storage of big data. *Future Generation Computer Systems*, 57:56 – 76, 2016.

[23] Weiwei Chen and Ewa Deelman. WorkflowSim: A toolkit for simulating scientific workflows in distributed environments. In *2012 IEEE 8th International Conference on E-Science, e-Science 2012*, 2012.

[24] Marcin Cieslik and Cameron Mura. Papy: Parallel and distributed data-processing pipelines in python. *arXiv preprint arXiv:1407.4378*, 2014.

[25] Lauro Beltrão Costa, Hao Yang, Emalayan Vairavanathan, Abmar Barros, Ketan Maheshwari, Gilles Fedak, D Katz, Michael Wilde, Matei Ripeanu, and Samer Al-Kiswany. The case for workflow-aware storage: An opportunity study. *Journal of Grid Computing*, 13(1):95–113, 2015.

[26] Rafael Ferreira da Silva, Weiwei Chen, Gideon Juve, Karan Vahi, and Ewa Deelman. Community resources for enabling research in distributed scientific workflows. In *e-Science (e-Science), 2014 IEEE 10th International Conference on*, volume 1, pages 177–184. IEEE, 2014.

[27] Daniel de Oliveira, Eduardo Ogasawara, Fernanda Baião, and Marta Mattoso. Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 378–385. IEEE, 2010.

[28] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[29] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, and Others. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[30] Michael J Flynn and Kevin W Rudd. Parallel architectures. *ACM Computing Surveys (CSUR)*, 28(1):67–70, 1996.

[31] James Frey. Condor dagman: Handling inter-job dependencies. *University of Wisconsin, Dept. of Computer Science, Tech. Rep*, 2002.

[32] W Gentzsch. Sun Grid Engine: towards creating a compute power grid. *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 35–36, 2001.

[33] Belinda Giardine, Cathy Riemer, Ross C Hardison, Richard Burhans, Laura Elnitski, Prachi Shah, Yi Zhang, Daniel Blankenberg, Istvan Albert, James Taylor, Webb Miller, W James Kent, and Anton Nekrutenko. Galaxy: a platform for interactive large-scale genome analysis. *Genome research*, 15(10):1451–5, oct 2005.

[34] Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team. Galaxy : a comprehensive approach for supporting accessible , reproducible , and transparent computational research in the life sciences. 2010.

[35] Adan Hirales-Carbajal, Andrei Tchernykh, Thomas Röblitz, and Ramin Yahyapour. A grid simulation framework to study advance scheduling strategies for complex workflow applications. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

[36] Udo Hönig and Wolfram Schiffmann. A meta-algorithm for scheduling multiple dags in homogeneous system environments. In *Proceedings of the eighteenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'06)*, 2006.

[37] E Ilavarasan and P Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer sciences*, 3(2):94–103, 2007.

[38] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, volume 41, pages 59–72. ACM, 2007.

[39] Mohammad Islam, Angelo K Huang, Mohamed Battisha, Michelle Chiang, Santhosh Srinivasan, Craig Peters, Andreas Neumann, and Alejandro Abdelnur. Oozie: towards a scalable workflow management system for hadoop. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, page 4. ACM, 2012.

[40] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, mar 2013.

[41] M Aleksi Kallio, Jarno T Tuimala, Taavi Hupponen, Petri Klemelä, Massimiliano Gentile, Ilari Scheinin, Mikko Koski, Janne Käki, and Eija I Korpelainen. Chipster: user-friendly analysis software for microarray and other high-throughput data. *BMC genomics*, 12(1):507, 2011.

[42] Tevfik Kosar and Miron Livny. Stork: Making data placement a first class citizen in the grid. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 342–349. IEEE, 2004.

[43] Richard T Kouzes, Gordon A Anderson, Stephen T Elbert, Ian Gorton, and Deborah K Gracio. The changing paradigm of data-intensive computing. *Computer*, 42(1):26–34, 2009.

[44] S Krishnan, M Tatineni, and C Baru. myHadoop-Hadoop-on-Demand on Traditional HPC Resources. *San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego*, 2011.

[45] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[46] Burkhard Linke, Robert Giegerich, and Alexander Goesmann. Conveyor: a workflow engine for bioinformatic analyses. *Bioinformatics*, 27(7):903–911, 2011.

[47] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[48] Lustre. http://wiki.lustre.org/index.php/Main_Page.

[49] Anirban Mandal, Ken Kennedy, Charles Koelbel, Gabriel Marin, John Mellor-Crummey, Bo Liu, and Lennart Johnsson. Scheduling strategies for mapping application workflows onto the grid. In *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, pages 125–134. IEEE, 2005.

[50] Richard P. Martin and David E. Culler. NFS sensitivity to high performance networks, 1999.

[51] Munir Merdan, Thomas Moser, Dindin Wahyudin, Stefan Biffl, and Pavel Vrba. Simulation of workflow scheduling strategies using the mast test management system. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pages 1172–1177. IEEE, 2008.

[52] Bertrand Néron, Hervé Ménager, Corinne Maufrais, Nicolas Joly, Julien Maupetit, Sébastien Letort, Sébastien Carrere, Pierre Tuffery, and Catherine

Letondal. Mobyle: a new full web bioinformatics framework. *Bioinformatics*, 25(22):3005–3011, 2009.

[53] Tchimou N'Takpe and Frederic Suter. Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE, 2009.

[54] T Oinn, M J Addis, J Ferris, D J Marvin, M Senger, T Carver, M Greenwood, K Glover, M R Pocock, a Wipat, and P Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, nov 2004.

[55] Joshua Orvis, Jonathan Crabtree, Kevin Galens, Aaron Gussman, Jason M Inman, Eduardo Lee, Sreenath Nampally, David Riley, Jaideep P Sundaram, Victor Felix, Brett Whitty, Anup Mahurkar, Jennifer Wortman, Owen White, and Samuel V Angiuoli. Ergatis: a web interface and scalable software system for bioinformatics workflows. *Bioinformatics (Oxford, England)*, 26:1488–1492, 2010.

[56] Gyung-Leen Park, Behrooz Shirazi, and Jeff Marquis. Dfrn: A new approach for duplication based scheduling for distributed memory multiprocessor systems. In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pages 157–166. IEEE, 1997.

[57] Luca Pireddu, Simone Leo, Nicola Soranzo, and Gianluigi Zanetti. A Hadoop-Galaxy adapter for user-friendly and scalable data-intensive bioinformatics in Galaxy. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics - BCB '14*, pages 184–191, New York, New York, USA, sep 2014. ACM Press.

[58] Mustafizur Rahman, Srikumar Venugopal, and Rajkumar Buyya. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. In *e-Science and Grid Computing, IEEE International Conference on*, pages 35–42. IEEE, 2007.

[59] Kavitha Ranganathan and Ian Foster. Design and evaluation of dynamic replication strategies for a high performance data grid. In *International*

Conference on Computing in High Energy and Nuclear Physics, volume 2001, 2001.

[60] Kavitha Ranganathan and Ian Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pages 352–358. IEEE, 2002.

[61] Paolo Romano. Automation of in-silico data analysis processes through workflow management systems, 2008.

[62] Sebastian Schönherr, Lukas Forer, Hansi Weißensteiner, Florian Kronenberg, Günther Specht, and Anita Kloss-Brandstätter. Cloudgene: A graphical execution platform for mapreduce programs on private and public clouds. *BMC bioinformatics*, 13(1):200, 2012.

[63] A Schumacher and L Pireddu. SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop. *Bioinformatics*, 2014.

[64] Aisha Siddiqa, Ahmad Karim, and Victor Chang. Smallclient for big data: an indexing framework towards fast data retrieval. *Cluster Computing*, pages 1–16, 2016.

[65] C Sloggett, N Goonasekera, and E Afgan. BioBlend: automating pipeline analyses within Galaxy and CloudMan. *Bioinformatics*, 2013.

[66] Slurm. https://computing.llnl.gov/linux/slurm.

[67] Galaxy User Statistics. https://wiki.galaxyproject.org/GalaxyProject/Statistics.

[68] Georgios L. Stavrinides and Helen D. Karatza. Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. *Simulation Modelling Practice and Theory*, 19(1):540–552, 2011.

[69] Andrew S Tanenbaum. *Structured computer organization*. Pearson, 2006.

[70] top500. https://www.top500.org/.

[71] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Task scheduling algorithms for heterogeneous processors. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 3–14. IEEE, 1999.

[72] Peter Troger, Hrabri Rajic, Andreas Haas, and Piotr Domagalski. Standardization of an api for distributed resource management systems. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 619–626. IEEE, 2007.

[73] J Wang, D Crawl, and I Altintas. Kepler+Hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems. *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, 2009.

[74] Xiaodan Wang, Christopher Olston, Anish Das Sarma, and Randal Burns. Coscan: cooperative scan sharing in the cloud. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 11. ACM, 2011.

[75] Tim Wickberg and Christopher Carothers. The ramdisk storage accelerator: a method of accelerating i/o performance on hpc systems using ramdisks. In *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, page 5. ACM, 2012.

[76] Marek Wieczorek, Radu Prodan, and Thomas Fahringer. Scheduling of scientific workflows in the askalon grid environment. *ACM SIGMOD Record*, 34(3):56–62, 2005.

[77] Michael Wilde, Mihael Hategan, Justin M Wozniak, Ben Clifford, Daniel S Katz, and Ian Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.

[78] Justin M Wozniak, Timothy G Armstrong, Ketan Maheshwari, Ewing L Lusk, Daniel S Katz, Michael Wilde, and Ian T Foster. Turbine: A distributed-memory dataflow engine for extreme-scale many-task applications. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, page 5. ACM, 2012.

[79] Tao Yang and Apostolos Gerasoulis. Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):951–967, 1994.

[80] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *ACM Sigmod Record*, 34(3):44–49, 2005.

[81] Zhifeng Yu and Weisong Shi. A Planner-Guided Scheduling Strategy for Multiple Workflow Applications. In *2008 International Conference on Parallel Processing - Workshops*, pages 1–8. IEEE, sep 2008.

[82] Henan Zhao and Rizos Sakellariou. Scheduling multiple dags onto heterogeneous systems. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 14–pp. IEEE, 2006.

Apéndice A

Archivos de Configuración Galaxy

El código fuente A.1 es un ejemplo del archivo de configuración job_conf.py, el cual define con que método serán ejecutados los trabajos en la infraestructura. En este caso, definimos como destino un cluster y drmaa como ejecutor en la línea 6 y 14. En la línea 7 hacemos la declaración del tipo de ejecutor será utilizado. En la línea 8 definimos cuantos nodos serán utilizados en el cluster.

```
1  <?xml version="1.0"?>
2  <!-- A sample job config that explicitly configures
3  job running the way it is configured by default
4  <job_conf>
5      <plugins>
6          <plugin id="cluster" type="runner"
7           load="galaxy.jobs.runners.local:ExternalJobRunner"
8           workers="32"/>
9      </plugins>
10     <handlers>
11         <handler id="main"/>
12     </handlers>
13     <destinations>
14         <destination id="cluster" runner="drmaa"/>
15     </destinations>
16 </job\_conf>
```

Código Fuente A.1: XML job_conf.py

El código en A.2 es un fragmento del archivo galaxi.ini, aunque el archivo

es extenso, este fragmento es el que nos compete, debido a que es donde se definen los directorios de almacenamiento de archivos, con estas líneas podremos definir explícitamente donde estarán los directorios de usuario

```
1 # -- Files and directories
2 # Dataset files are stored in this directory.
3 file_path = database/files
4
5 # Temporary files are stored in this directory.
6 new_file_path = database/tmp
```

Código Fuente A.2: Galaxy.ini

El código en A.3 del archivo external_runner.py, un conjunto de funciones y clases para el manejo de aplicaciones en un entorno drmaa y llamadas al API de ejecución. Como ejemplo, el inicio de sesión se hace con la llamada en línea 7 drmaa.session(), se crean los *templates* con los parámetros de la aplicación en la línea 9 y finalmente se envía para ejecución en la línea 12.

```
1
2 def main():
3     userid, json_filename, assign_all_groups = validate_paramters()
4     set_user(userid, assign_all_groups)
5     json_file_exists(json_filename)
6     os.environ['BSUB_QUIET'] = 'Y'
7     s = drmaa.Session()
8     s.initialize()
9     jt = s.createJobTemplate()
10    load_job_template_from_file(jt, json_filename)
11    # runJob will raise if there's a submittion error
12    jobId = s.runJob(jt)
13    s.deleteJobTemplate(jt)
14    s.exit()
15
16    # Print the Job-ID and exit. Galaxy will pick it up from there.
17    print(jobId)
```

Código Fuente A.3: external runner.py

El código A.4 del archivo handler.py, para el manejo de estados de las aplicaciones en el sistema, es quien controla que las aplicaciones estén siendo

ejecutadas o en qué estado se encuentran, así mantener una lista de aplicaciones. De las líneas 1 a la 11 se definen macros globales de estado, a continuación de las líneas 18 a 24 se realiza el control de las colas de aplicaciones en ejecución y espera. y de las líneas 26 a la 31, se inician y finalizan las colas. Este archivo contiene definiciones de funciones que omitimos nombrarlas aquí, por brevedad e importancia.

```
1  # States for running a job. These are NOT the same as data states
2  JOB_WAIT, JOB_ERROR, JOB_INPUT_ERROR, JOB_INPUT_DELETED, JOB_READY,
3  JOB_DELETED, JOB_ADMIN_DELETED, JOB_USER_OVER_QUOTA,
4
5  JOB_USER_OVER_TOTAL_WALLTIME = 'wait', 'error', 'input_error',
6  'input_deleted', 'ready', 'deleted', 'admin_deleted',
7  'user_over_quota', 'user_over_total_walltime'
8
9  DEFAULT_JOB_PUT_FAILURE_MESSAGE = 'Unable to run job due to a
10 misconfiguration of the Galaxy job running system.
11 Please contact a site administrator.'
12
13
14 class JobHandler( object ):
15     """
16     Handle the preparation, running, tracking, and finishing of jobs
17     """
18     def __init__( self, app ):
19         self.app = app
20         # The dispatcher launches the underlying job runners
21         self.dispatcher = DefaultJobDispatcher( app )
22         # Queues for starting and stopping jobs
23         self.job_queue = JobHandlerQueue( app, self.dispatcher )
24         self.job_stop_queue = JobHandlerStopQueue(app, self.dispatcher)
25
26     def start( self ):
27         self.job_queue.start()
28
29     def shutdown( self ):
30         self.job_queue.shutdown()
31         self.job_stop_queue.shutdown()
```

Código Fuente A.4: handler.py

```
class DRMAAJobRunner( AsynchronousJobRunner ):
...
    # Descriptive state strings pulled from the drmaa lib itself
        self.drmaa_job_state_strings = {
            drmaa.JobState.UNDETERMINED:
            'process status cannot be determined',
            drmaa.JobState.QUEUED_ACTIVE:
            'job is queued and active',
            drmaa.JobState.SYSTEM_ON_HOLD:
            'job is queued and in system hold',
            drmaa.JobState.USER_ON_HOLD:
            'job is queued and in user hold',
            drmaa.JobState.USER_SYSTEM_ON_HOLD:
            'job is queued and in user and system hold',
            drmaa.JobState.RUNNING:
            'job is running',
            drmaa.JobState.SYSTEM_SUSPENDED:
            'job is system suspended',
            drmaa.JobState.USER_SUSPENDED:
            'job is user suspended',
            drmaa.JobState.DONE:
            'job finished normally',
            drmaa.JobState.FAILED:
            'job finished, but failed',
        }
...
  def get_native_spec( self, url ):
 """Get any native DRM arguments specified by the site configuration"""

    def queue_job( self, job_wrapper ):
 """Create job script and submit it to the DRM"""
    def external_runjob(self, external_runjob_script,
    jobtemplate_filename, username):
""" runs an external script the will QSUB a new job.
    def _job_name(self, job_wrapper):
        external_runjob_script =
       job_wrapper.get_destination_configuration("", None)
        galaxy_id_tag = job_wrapper.get_id_tag()
```

Código Fuente A.5: drmaa.py

El código A.5 del archivo drmaa.py, crea los *templates* con los parámetros y definiciones propias de cada aplicación, de modo a que el *runner* pueda enviar la aplicación a ser ejecutada. Es responsable de mantener una máquina de estados de cada una de las aplicaciones. De las líneas 4 a 24 se inicializan todos los posibles estados. La línea 27, recupera los parámetros de la infraestructura. La línea 32 utiliza el ejecutor externo para enviar la aplicación a la infraestructura. La líneas finales, recuperan datos para el administrador, como el ID asignado por el DRM a la aplicación.

Este último código A.1 es un ejemplo de archivo shell generado por el drmaa.py con las definiciones de la aplicación a ser ejecutada. Podemos observar en la parte inicial del archivo.sh los parámetros de ejecución como el número de tareas a ser ejecutadas, los nodos a ser utilizados y los parámetros de la aplicación Python. Finalmente, el archivo contiene todos los parámetros específicos de localización de los archivos de entrada y salida para la aplicación a ser ejecutada.

```
#!/bin/sh

export GALAXY_SLOTS_CONFIGURED="1"
if [ -n "$SLURM_NTASKS" ]; then
    # May want to multiply this by ${SLURM_CPUS_PER_TASK:-1}.
    # SLURM_NTASKS is total tasks over all nodes so this is
    # shouldn't be used for multi-node requests.
    GALAXY_SLOTS="$SLURM_NTASKS"
elif [ -n "$NSLOTS" ]; then
    GALAXY_SLOTS="$NSLOTS"
elif [ -f "$PBS_NODEFILE" ]; then
    GALAXY_SLOTS=`wc -l < $PBS_NODEFILE`
else
    GALAXY_SLOTS="1"
    unset GALAXY_SLOTS_CONFIGURED
fi

export GALAXY_SLOTS
GALAXY_LIB="None"
if [ "$GALAXY_LIB" != "None" ]; then
    if [ -n "$PYTHONPATH" ]; then
        PYTHONPATH="$GALAXY_LIB:$PYTHONPATH"
    else
        PYTHONPATH="$GALAXY_LIB"
    fi
    export PYTHONPATH
fi

cd /home/galaxy/galaxy-central/database/job_working_directory/000/388
bash /home/galaxy/galaxy-central/tools/freesurfer/block1.sh /home/galaxy/galaxy-central/database/files/000/dataset_380.dat /home/galaxy/galaxy-central/database
iles/000/dataset_468.dat; return_code=$?; cd /home/galaxy/galaxy-central; /home/galaxy/galaxy-central/set_metadata.sh ./database/files /home/galaxy/galaxy-cent
l/database/job_working_directory/000/388 . /home/galaxy/galaxy-central/universe_wsgi.ini /home/galaxy/galaxy-central/database/tmp/tmpPXDLff /home/galaxy/galaxy
entral/database/job_working_directory/000/388/galaxy.json /home/galaxy/galaxy-central/database/job_working_directory/000/388/metadata_in_HistoryDatasetAssociat
n_470_SLDPL0,/home/galaxy/galaxy-central/database/job_working_directory/000/388/metadata_kwds_HistoryDatasetAssociation_470_S5nw0P,/home/galaxy/galaxy-central/
tabase/job_working_directory/000/388/metadata_out_HistoryDatasetAssociation_470_RdzGio,/home/galaxy/galaxy-central/database/job_working_directory/000/388/metad
a_results_HistoryDatasetAssociation_470_PMzXBX,,/home/galaxy/galaxy-central/database/job_working_directory/000/388/metadata_override_HistoryDatasetAssociation_
0_W7TP5g; sh -c "exit $return_code"
echo $? > /home/galaxy/galaxy-central/database/job_working_directory/000/388/galaxy_388.ec
~
~
```

Figura A.1: Archivo de aplicacion.sh generado por DRMAA