Universidad Autónoma de Barcelona

Escuela de Ingeniería

Departamento de Arquitectura de Computadores y Sistemas Operativos

Framework for Integrating Scheduling Policies into Workflow Engines

Tesis doctoral presentada por **Gustavo Martínez** para optar al grado de Doctor por la Universitat Autònoma de Barcelona, bajo la dierección de los Drs. Elisa Heymann y Miquel A. Senar.

Bellaterra, 6 de Junio del 2011

Framework for Integrating Scheduling Policies into Workflow Engines

Tesis doctoral presentada por **Gustavo Martínez** para optar al grado de Doctor por la Universitat Autònoma de Barcelona. Trabajo realizado en el Departament d'Arquitectura de Computadors i Sistemes Operatius de la Escola d' Enginyeria de la Universitat Autònoma de Barcelona, dentro del programa de Doctorado en Informática, Opción A "Arquitectura de computadores y procesamiento paralelo" bajo la dirección de los Drs. Elisa Heymann y Miquel A. Senar.

Barcelona, 6 de Junio del 2011

Dra. Elisa Heymann

Dr. Miquel Senar

AGRADECIMIENTOS

A Dios todopoderoso por permitirme la oportunidad de alcanzar exitosamente esta etapa de superación profesional.

A mi madre Hilda Araujo que con todo su amor y cariño, ha sabido comprender mi destino apoyándome siempre gracias mamá te adoro.

A mi esposa Alexandra Corrao quien ha vivido a mi lado todo este proyecto, gracias por estar presente cada día te amo.

A mi futura hija Hilda Naomi, que le sirva de ejemplo de constancia para llegar siempre a los objetivos planteados en la vida.

A Alexandre Strube un gran amigo, gracias mi pana por toda la ayuda prestada.

A Elisa Heymann y Miquel Senar quienes han aportado todas sus ideas y conocimientos, para completar este trabajo satisfactoriamente.

A Lola por todo su apoyo en los momentos de angustia, no hay palabras para ti más que un enorme agradecimiento.

A Porfídio Hernández y Remo Suppi, a quienes agradeceré siempre su valiosa amistad y grandes consejos.

A Emilio Luque, gracias a su confianza hoy estoy aquí presentando este trabajo.

A todos los compañeros del doctorado que de una otra manera siempre han estado ahí, aportando ideas y consejos tanto en lo profesional como personal.

A todos Muchísimas Gracias

Gustavo E. Martínez

Índice general

Índice de figuras

Índice de tablas

Resumen

En este trabajo se ha desarrollado una nueva solución para la integración de políticas de planificación de workflows en sistemas gestores de workflows (workflow engines). SchedFlow es la solución propuesta e implementada, la cual permite que un usuario final pueda utilizar la política de planificación que desee, ya sea esta estática o dinámica, sin necesidad de modificar el sistema gestor de workflow.

Además SchedFlow toma en cuenta cuando una tarea del workflow no se puede ejecutar por eventos externos, como que una máquina está siendo utilizada por el usuario principal de la misma.

En este tipo de escenarios SchedFlow realiza la replanificación de tareas, siempre bajo la política de planificación definida o elegida por el usuario. Como resultado se ha obtenido una aportación novedosa ya que normalmente si el usuario desea utilizar una política de planificación específica, deberá modificar el gestor de workflow para que soporte dicha política de planificación.

Un punto importante a destacar, es que esta tesis además de contener un estudio exhaustivo de la bibliografía existente en el área, y de realizar un análisis de problema y diseño de solución propuesta, implementa la solución en un entorno oportunístico real.

Los resultados de esta tesis para la planificación de workflows en entornos Grid, abre una nueva vía para el estudio de políticas de planificación para este tipo de aplicaciones, que será aprovechado por futuras investigaciones en la UAB.

Capítulo 1

Introducción

En la actualidad existen en el mundo de las ciencias computacionales un elevado número de aplicaciones orientadas al cómputo intensivo. Una de las aplicaciones que podemos encontrar que requieren un elevado cómputo son las aplicaciones workflow. Las aplicaciones workflow son aquellas que están compuestas por varias tareas de cómputo y que tienen una relación de dependencia entre ellos [93].

Hoy en día existen muchos ejemplos de estas aplicaciones workflow que van desde la física de altas energías [43, 33], aplicaciones del genoma humano [66], simuladores de sistemas complejos [15, 83, 82], y más recientemente procesamiento de imágenes espaciales [13, 27, 68, 49].

Todas estas aplicaciones mencionadas representan importantes avances para la comunidad científica, la cual está en una constante lucha por obtener resultados más precisos y rápidos, que ayuden a comprender mejor el mundo en el cual estamos viviendo. Para obtener esos resultados más precisos se hace necesario incluir más detalles a la aplicación lo que eleva el cómputo intensivo de la misma, que viene representado normalmente por un incremento importante en el tiempo de ejecución de la aplicación workflow y que evita inicialmente tener un resultado en menor tiempo.

Esos impresionantes desarrollos de aplicaciones workflow para comprender mejor el mundo en el que vivimos, trae con ellos una serie de problemas al momento de ejecutarlas, por ejemplo no disponer de suficientes recursos para la ejecución, carecer de políticas de planificación ajustadas a la aplicación workflow, desconocimiento del entorno de ejecución, limitaciones propias de los gestores entre otros.

Las consecuencias de estos problemas son que las aplicaciones workflow no se ejecuten de forma rápida, sin contar que de entrada estas aplicaciones tienen un elevado tiempo de cómputo, entonces si sumamos estos iniciales problemas el resultado se traduce en largas esperas para los investigadores quienes a su vez pierden valioso tiempo para poder avanzar en sus estudios particulares.

Los investigadores deben hacer frente diariamente a estos problemas, buscando un objetivo común que es reducir el tiempo de ejecución total de la aplicación. Este tiempo total se define como tiempo desde que la primera tarea comienza su ejecución hasta que la última tarea finaliza y que se conoce como *makespan* [14].

La comunidad científica ante esta situación, encontró en los últimos años una interesante propuesta para abordar esta problemática existente y sin una efectiva solución. Una de las primeras propuestas que nacieron fue realizar la ejecución de estas aplicaciones utilizando los entornos distribuidos.

Un entorno distribuido viene definido, como un conjunto de ordenadores interconectados en red y que se comunican entre sí, para coordinar sus acciones utilizando el paso de mensajes para así alcanzar un objetivo común. Dando una potencia de cálculo superior a lo que podríamos tener si utilizamos un simple ordenador que posea unas buenas prestaciones [98, 48, 21, 84, 36].

Esta propuesta buscaba aprovechar de alguna manera la potencia individual de los ordenadores en la suma conjunta de ellos, utilizando las tecnologías existentes en ese momento para las redes de interconexión.

Por otro lado en el año 1985 que se demostró que la mayoría de clústeres se encontraban desocupados una elevada cantidad de tiempo, y se propuso aprovechar esos intervalos de desocupación para realizar ejecuciones de trabajos, claro siempre pensando en que cuando llega una tarea de prioridad superior a la nuestra, el proceso nuestro será expulsado del recurso agregando un nuevo problema que es el de gestionar estos eventos propios de este entorno[98].

Un ejemplo de este tipo de eventos es el hecho de que en una red donde se utilizan diversos sistemas operativos (Linux y Windows), nos podemos encontrar con un escenario inicial para ejecutar una aplicación con 20 recursos de cómputo, pero si por alguna razón llega un usuario que desea trabajar en Windows reiniciaría el ordenador, que estaba ejecutando una tarea que será interrumpida de forma permanente teniendo que ser re-planificada a otro recurso y tomar las acciones correctivas de la planificación inicial.

Esta propuesta bastante acertada nos daba una solución para aumentar el poder de cómputo, pero también nos presentaba un nuevo problema el decidir qué

tipo de entorno distribuido se utiliza, ya que esto entornos están divididos en dos grandes grupos, estos grupos están contemplados en entornos dedicados y entornos no dedicados.

Estos grupos son conceptos claves para poder definir la estrategia correcta para atacar los problemas propios presentes en cada entorno, puesto que el funcionamiento de cada uno de ellos está basado en modelos totalmente diferentes. A continuación damos la definición de cada uno de ellos:

- Entornos distribuidos dedicados [61, 38], son aquellos en donde los usuarios saben con exactitud los recursos de cómputo de los que disponen, de tal manera que suelen ser muy predecibles al momento de la ejecución conocer el tipo de recurso y la cantidad disponibles, además de ser controlables de manera más sencilla, pero su principal problema es que suelen ser costosos.
- Entornos distribuidos no dedicados [70, 99, 19], son aquellos sistemas en los que suele ser complicado conocer de antemano cuantos recursos tenemos disponibles, que características poseen, cuánto tiempo los tendremos disponible, lo que hacen que sean muy cambiantes en el tiempo, pero tienen una gran ventaja y es que su costo es casi nulo.

Esto nos da una idea de la complejidad de este tipo de entornos puesto que su dinamismo intrínseco, no nos proporciona la certeza exacta de los recursos con los que contamos para realizar la ejecución de una aplicación, lo que supone un reto importante para cualquier estudio donde intentemos disminuir tiempos de ejecución.

Otro punto importante que debemos mencionar es la existencia de muchas otras arquitecturas, como son el Internet Computing [34, 28, 45], Web Computing [67, 55, 74], Grid Computing [35, 8, 52], y lo más reciente el Cloud Computing [16], que vienen a representar de forma mucho más específica los entornos existentes de ejecución.

Mencionado todo lo relevante a entornos distribuidos elemento esencial para poder ejecutar las aplicaciones, llega el momento de indicar que nuestra investigación estará centrada por un lado en cómo mejorar las prestaciones de las aplicaciones workflow, utilizando para ello los entornos distribuidos no dedicados.

1.1. Los Entornos de Ejecución

En esta sección se explicará un poco más lo que representa un entorno de ejecución, ya que de el depende que una planificación se complete de forma

correcta, o al menos se ajusten lo más posible la estimación planificada inicialmente.

Hace ya unos años que se vienen desarrollado tecnologías como Internet Computing, cuyo fundamento teórico se basa en aprovechar el tiempo de inactividad de los computadores que se encuentran conectados en las redes de interconexión de hoy, por esta razón hemos pensado en adaptar esta gran idea para nuestra investigación asumiendo que el entorno de ejecución que tenemos disponible nos proporciona los recursos suficientes para realizar las pruebas de nuestras ideas.

Adicionalmente para dar soporte a nuestra selección tenemos ejemplos muy claros, donde este tipo de entornos distribuidos han sido altamente utilizados, y evidentemente esto nos hace pensar que los resultados finales que obtengamos en ellos se ajustarán bastante bien a la realidad, como ya ha ocurrido en estudios previos que la comunidad científica del Grid Computing ya ha realizado.

La tecnología Grid Computing [35, 8] cuyo término se refiere a una infraestructura que permite la integración y uso colectivo de ordenadores de alto rendimiento, redes y bases de datos que están administradas por diferentes instituciones. Este último si hacemos buen uso de los recursos nos puede proporcionar muy buenos resultados en términos de tiempo de ejecución.

Dentro de este mismo contexto existe también la tecnología de Web Computing término caracterizado por Mark Weiser [6], cuya meta es el incremento en el uso de sistemas de cómputo a través del ambiente físico, haciéndolos disponibles y a la vez invisibles al usuario. Esta propuesta se ha posicionado como la tercera generación o paradigma en la computación, que cuenta ya con múltiples aplicaciones y equipos de investigación que procuran su evolución y desarrollo.

Finalmente nos resta comentar la emergente tecnología del Cloud Computing cuyo paradigma permite ofrecer servicios de computación a través de Internet, en este tipo de computación todo lo que puede ofrecer un sistema informático se ofrece como servicio, de modo que los usuarios puedan acceder a los servicios disponibles en la nube de Internet sin conocimientos, o al menos sin ser expertos en la gestión de los recursos que utilizan [47, 94].

Toda la investigación se enmarca en el contexto de entornos no dedicados, concepto que encaja de forma muy precisa al entorno de ejecución del que disponemos para llevar a cabo todos los experimentos. Debemos señalar que este tipo de entornos tiene mucha similitud pero en una escala menor con el representado en un sistema Grid, en el cual existen muchas variables a tener en cuenta y que suelen ser muy complejos al momento de modelarlo.

1.2. Aplicaciones Workflow

En esta sección pretendemos introducir al lector algunos conceptos básicos sobre este tipo de aplicaciones, que luego detallaremos más adelante en el capítulo 2 de esta memoria para poder tener una idea más clara de las mismas.

Actualmente muchas de las aplicaciones desarrolladas para la comunidad científica son modeladas mediante un workflow, esto porque por definición un workflow se refiere al flujo de trabajo a seguir para la consecución de una tarea o proceso predeterminado. Este flujo de trabajo debe ser correctamente planificado para minimizar el tiempo de ejecutar la aplicación, agilizando de esta manera la evaluación de por parte de los expertos.

Dicho esto se puede intuir que los gestores de planificación juegan un papel importante al momento de ejecutar estas aplicaciones, esto ha motivado a la comunidad científica a obtener constantemente diferentes sistemas capaces de ejecutar de forma óptima las aplicaciones workflow. Para esto han propuesto diferentes entornos de ejecución, decenas de políticas de planificación y por supuesto diversos gestores capaces de ejecutar estas complejas aplicaciones.

¿Por qué una aplicación workflow?, la respuesta a esta pregunta no es trivial ya que una de las ventajas principales de estas aplicaciones, es poder organizar todo un conjunto de tareas para que se ejecutan coordinadamente, pero de forma independiente aprovechando de esta manera la ejecución de las mismas en entornos no dedicados, ahorrando así el coste que significa ejecutar un solo proceso en un potente recurso de cómputo [31, 76, 79].

En nuestra investigación vamos a utilizar este tipo de aplicaciones, por un lado para definir una alternativa diferente al momento de planificar un workflow, además de aprovechar las ventajas que representan poder ejecutarlas en un entorno no dedicado, donde los recursos no necesitan tener una potencia de cómputo elevado para completar las tareas que componen la aplicación. En la figura 1.1 se puede ver un ejemplo gráfico de como se representa una aplicación workflow.

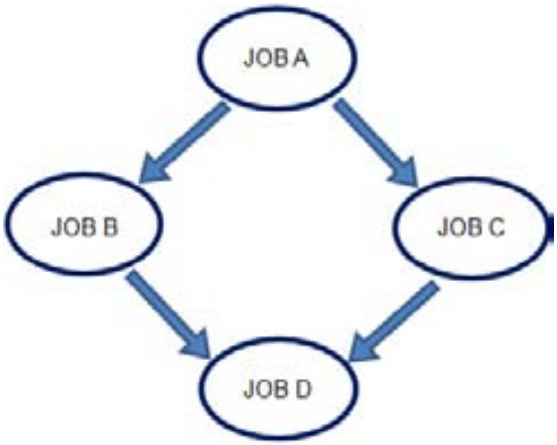

Figura 1.1: Ejemplo de aplicación workflow.

Una vez definidas las aplicaciones a utilizar en nuestra investigación, se hace necesario mencionar un nuevo elemento que se incorpora a la misma y son los **Gestores de Workflow**. Un gestor de workflow es un sistema que se encarga llevar la gestión de la ejecución de una aplicación workflow, teniendo que encontrar los recursos disponibles del entorno de ejecución cada vez que se necesita alguno durante la ejecución. La descripción de este importante elemento de nuestra investigación está desarrollada en la sección 1.3 de esta memoria.

1.3. Los Gestores de Aplicaciones Workflow

Esta sección mencionamos algunos de los gestores de aplicaciones workflow importantes y ampliamente utilizando en numerosas investigaciones, lo que nos da una buena retroalimentación sobre cada uno de ellos.

En la literatura existen un gran número de gestores de workflow desde los más clásicos instalados en los máquinas locales Condor DAGMan [39, 88], Taverna [72, 53, 73], Karajan [96, 97], Triana [86, 87, 95], hasta los ejecutados desde servicios web como Pegasus [23, 62], WebFlow [42, 51], todos ellos persiguen un objetivo común gestionar de la mejor manera la ejecución de una aplicación workflow.

Para ello utilizan una serie de funciones, con las cuales pueden obtener la información necesaria de donde enviar una tarea y en qué momento. Su modo de funcionamiento suele ser ubicar un recurso disponible y una vez localizado seleccionar una tarea y enviarla para su ejecución.

El problema es que en los entornos no dedicados esto no es tan simple, ya que la propia complejidad del entorno de ejecución, hace muy difícil disponer siempre de recursos de cómputo donde poder ejecutar las aplicaciones.

Entonces parece que no toda la responsabilidad recae en el gestor, sino que existen muchos otros factores asociados al entorno de ejecución, tipo de aplicación, y selección de la política.

Otro problema que encontramos es que la mayoría de estos gestores funcionan con una política de planificación interna, la cual permite obtener la combinación tarea-máquina, que finalmente será enviada a los recursos disponibles, una vez cumplidos los requisitos establecidos por el usuario para la ejecución de la aplicación workflow.

Debido a la naturaleza de las aplicaciones y del entorno de ejecución distribuido, en ocasiones resulta complicado evaluar el resultado de una política de planificación.

En una primera etapa se suelen realizar evaluaciones de una forma relativamente sencilla, haciendo uso de simuladores, que permitan ver el comportamiento de estas aplicaciones sobre un entorno no dedicado.

Estos cambios se pueden aplicar a diferentes políticas de planificación, para así tener una idea más aproximada sobre cuáles pueden ser las mejores condiciones para la ejecución.

Estas mejoras se traducen en una reducción de los tiempos de ejecución, y esto depende de dos factores importantes el tipo de política que se usa, y los recursos que tenemos disponibles en el entorno de ejecución.

Pero esto no siempre es posible conseguirlo ya que aparece un nuevo reto, y es como vincular esas políticas externas a los gestores de manera rápida y sencilla, sin que represente un gran esfuerzo al usuario que la desee implementar.

Recordemos que las arquitecturas de los gestores de workflow no son siempre las mismas, y esto supone hacer un estudio exhaustivo para conocer los elementos que me darán la información y luego ver cómo hacer para comunicar una política de planificación con esos elementos que pertenecen a una determinada arquitectura propia del gestor.

Entonces parece ser interesante crear un *framework* que nos permita vincular de forma externa las políticas de planificación sobre un determinado gestor de workflow, y así tener un punto de partida para nuestra investigación que traerá como consecuencia directa una mejora sobre el tiempo de ejecución de la aplicación.

1.4. La Problemática Existente

Ahora que hemos descrito los elementos y las tecnologías necesarias para ejecutar las aplicaciones workflow, se hace necesario mencionar el problema existente al momento de ejecutar estas aplicaciones workflow.

Partamos del hecho que un usuarios cualquiera decide ejecutar una aplicación workflow, entonces debe seleccionar el tipo de entorno donde lo desea ejecutar, luego debe decidir un gestor de workflow sencillo de utilizar, adaptar la descripción del workflow a la admitida por el gestor, y finalmente realizar la ejecución.

Planteado de este modo parece que no hay ningún problema para el usuario, pero la realidad es muy diferente al simple hecho de hacer la selección de los componentes para ejecutar la aplicación. El primer punto es que normalmente se suele seleccionar un entorno no dedicado, esto porque representa un costo casi nulo. Después tenemos que la adaptación del workflow a la descripción del gestor puede no ser tan simple, a menos que tengamos cierta experiencia utilizando el mismo.

Pero una vez que se tiene todo seleccionado, es cuando el usuario debe empezar a plantearse algunas cuestiones, por ejemplo: ¿Cómo de estable es nuestro entorno de ejecución? ¿Cuántos recursos tengo disponibles realmente para ejecutar mi aplicación? ¿Qué tipo de política es la utilizada para la planificación del workflow?

Entonces supongamos que las tres preguntas han sido superadas y el usuario ejecuta la aplicación, y resulta que según su conocimiento la misma debería finalizar de modo estimado en dos horas, pero finaliza en cinco horas, también hay casos donde el sistema simplemente nunca da una respuesta.

Aquí el usuario empieza a plantearse la posibilidad de utilizar una política de planificación, que ofrezca mejores resultados en tiempo de ejecución a los obtenidos, ajustando al máximo posible a lo obtenido de forma estimada.

En este momento aparece un vacio en todo nuestro espacio de ejecución, porque los actuales gestores de workflow no dan soporte de forma sencilla para que el usuario incorpore una nueva política de planificación en su arquitectura. Este es uno de los problemas al que se ha dado solución con nuestra investigación.

Pero no solo tenemos este problema a resolver, ya que siendo un entorno distribuido no dedicado, corremos el riesgo de que los gestores de workflow en cada organización participante sean diferentes. Lo que se traduce en que nuestra solución debe ser transparente al gestor de workflow que se esté utilizando.

Adicionalmente y producto del dinamismo propio del entorno, se hace necesario tener un mecanismo que nos permita replanificar las tareas, puesto que no todos los gestores de workflow son capaces de hacerlo, y estamos seguros que eventos inesperados ocurren en este tipo de entornos.

Entonces mostrado todos estos problemas parece que no es tan sencillo ejecutar una aplicación workflow, pues hay que tomar en cuenta todos estos detalles que son los que nos permitirán obtener una ejecución rápida de nuestra aplicación.

1.5. Nuestra Propuesta

Una vez identificados de manera global los problemas que podemos encontrar cuando un usuario desea ejecutar una aplicación workflow, se propuso las siguientes soluciones en este trabajo de investigación.

Primero realizar pruebas en un entorno no dedicado en el cual los recursos pueden considerarse como clústeres locales, pero donde están presentes los problemas de disponibilidad, ancho de banda, suspensiones, características diversas en los recursos de cómputo (heterogeneidad), entre otras. Nuestra contribución en este punto fue identificar los eventos que más se presentan en entornos no dedicados, para posteriormente introducir soluciones apropiadas que aportarán mejoras a nuestro sistema.

Las aplicaciones que hemos seleccionado para esta investigación, son aquella cuyo comportamiento sea del tipo workflow donde están presentes los problemas de coordinación, tiempos de cómputo, tiempos de comunicación, tamaños de carga inicial, no se consideran para nuestro trabajo los DAG recurrentes.

Para trabajar con los clústeres locales se necesitan unos servicios (middleware), situados entre el sistema operativo de cada máquina y las aplicaciones de los usuarios. Las aplicaciones utilizarán estos servicios a partir de la interfaz de aplicaciones (API) propia de los middlewares, que supone tener un control coordinado al momento de la ejecución.

La ejecución eficiente de una aplicación sobre un entorno distribuido, requieren además de servicios que permitan al usuario interactuar cómodamente y de forma transparente con el sistema, herramientas que controlen y planifiquen automáticamente la ejecución de la aplicación sobre los distintos recursos del entorno.

Teniendo en cuenta el estado dinámico del sistema y con el objetivo de aprovechar los recursos de la mejor manera posible. La función principal que

debe tener un planificador, es la de controlar la ejecución de la aplicación sobre los recursos disponibles del entorno distribuido.

Esto se logra determinando cuantos recursos son necesarios, y cuáles de esos recursos son los más apropiados para ejecutar esa tarea particular de la aplicación, para poder reducir los el tiempo de ejecución global de la aplicación.

Un planificador eficiente debe tener en cuenta tres aspectos, velocidad de procesamiento del recurso, latencia y ancho de banda de la red, intuitivamente una buena selección por parte del planificador sería la colocar las tareas en los recursos más rápidos.

Pero en muchas ocasiones el planificador debe gestionar de manera adecuada los recursos disponibles, ya que en función de ello puede ser que se gane en tiempo de ejecución, pero se pierda en tiempos de transferencia si existe una elevada comunicación en el entorno de ejecución.

Desafortunadamente, el uso de las buenas herramientas gestoras de workflow están lejos de poder brindar la posibilidad de control más cercana al usuario, el cual pueda de manera sencilla no puede integrar una política de planificación diferente a la por defecto, y que con seguridad proporcionará unos mejores resultados en el tiempo de ejecución.

Los sistemas utilizados actualmente muestran estas características, pero también esas limitaciones al momento de la ejecución de aplicaciones workflow, sin embargo dentro de la comunidad científica existen numerosas aplicaciones capaces de gestionar los workflows, pero con la limitación que supone tener que usar su política de planificación por defecto.

Teniendo en cuenta que hoy día las aplicaciones workflow representan un grupo importante de aplicaciones en el mundo de las ciencias computacionales, y existiendo esta necesidad por parte de los usuarios, para dar una solución a ello esta es la propuesta:

Desarrollar un *framework* que nos permita integrar las políticas de planificación existentes en la bibliografía, y que son comúnmente utilizadas por los usuarios y comunidades científicas para dar solución eficiente a estas aplicaciones, dando la posibilidad al usuario de seleccionar la política de planificación que mejor se ajuste a la aplicación.

También proponemos que el *framework* tenga la posibilidad de acoplarse con algunos de los gestores de workflow existentes en la actualidad, esto de forma transparente tanto al gestor como al usuario, dando un abanico más amplio de ejecución.

Nuestra idea concibe además tener un mecanismo automático que sea capaz de reaccionar ante un cambio en el sistema, tomando la acción correctiva evitando que el tiempo consumido hasta ese momento por la aplicación no se pierda, utilizando la teoría de replanificación.

Todo esto lo hemos realizado a través de una API (Interfaz de Programación de Aplicaciones) [81, 41], que por un lado implemente los métodos necesarios para poder gestionar toda esta arquitectura a la que nos enfrentamos en esta investigación.

Así daremos al usuario final la posibilidad de experimentar con diferentes políticas de planificación, sin que ello resulte un engorroso camino para poder llegar a un solo gestor, la consecuencia directa de esto es tener todo un *framework* que nos ayude al objetivo de la investigación que es reducir el índice de prestaciones que en nuestro caso es el tiempo total de ejecución conocido como *makespan* [14].

Ahora bien nuestro trabajo se centra fundamentalmente en conseguir acoplar todos los elementos en un entorno de ejecución real, donde las características son totalmente dinámicas y los cambios pueden ocurrir en cuestión de segundos, dada estas condiciones dinámicas se realizaron caracterizaciones tanto de las políticas de planificación, como de los gestores de workflow, para así poder extraer los diferentes elementos, que nos proporcionen una solidez a nuestra investigación.

1.6. Objetivos

Nuestra investigación tiene unos objetivos claramente definidos, ya que tenemos todos los elementos identificados, y asumiendo como un hecho la propuesta antes mencionada ahora indicamos nuestros objetivos.

- Diseñar e implementar un *framework* capaz de gestionar la integración de las políticas de planificación en los diferentes gestores de workflow, de forma sencilla, flexible y transparente.

- Pruebas conceptuales para reducir el *makespan* de las aplicaciones workflow utilizando SchedFlow.

- Experimentar con diferentes políticas de planificación, incluyendo gestión de cambios en un entorno distribuido.

1.7. Contribuciones

La ejecución de aplicaciones workflow en un entorno no dedicado, surge de una serie de retos por parte de la comunidad científica y a las que no se han atendido algunas necesidades.

La mayoría de los sistemas de gestión para la ejecución de workflow se dedican a la ejecución de las aplicaciones, sin tomar en cuenta la necesidad de planificarlas adecuadamente. Dejando de lado todos los aportes que hacen los investigadores dedicados a la creación de políticas de planificación.

En este trabajo se propone una solución integral que se ocupa de vincular de forma transparente y sencilla, ese vacío existente en ambas áreas. Las principales aportaciones de este trabajo son las siguientes:

1. La definición de una arquitectura para la integración de una política de planificación dentro de un gestor de workflow. Esta arquitectura contempla los cambios dinámicos del entorno, el gestor de workflow a utilizar, las implementaciones librerías dinámicas, y un mecanismo para la gestión de eventos, que apoyo a la política de planificación integrada. La combinación de estos módulos permite la integración de una amplia gama de políticas de planificación en diferentes gestores de workflow.

2. La posibilidad de seguir una metodología diferente al momento de planificar una aplicación workflow, siendo el usuario final quien decida qué política utilizar en la ejecución. Esta acción antes estaba muy ligada al gestor de workflow utilizado, nuestro *framework* permite desvincular este lazo dando un mayor control al usuario.

3. Un mecanismo para gestionar los eventos inesperados durante la ejecución de la aplicación workflow, siendo un punto de apoyo a la política de planificación, permitiendo la replanificación de una tarea que puede ocurrir muy seguido en un entorno no dedicado. Todo de manera automática sin que el usuario intervenga en el proceso, evitando así tiempos importantes de volver a lanzar la aplicación.

4. Mejoras importante en el makespan de la aplicación, producto de la integración de las políticas de planificación, que ya traen por definición una mejor forma de llevar a cabo la ejecución de un workflow.

5. Un estudio experimental en el que medir los beneficios del *framework* propuesto en un entorno no dedicado. También medir los efectos de variar la carga inicial del workflow con diferentes políticas de planificación.

1.8. Organización de la Memoria

La memoria aquí presentada consta de las siguientes partes:

El capítulo 1, donde se hace una introducción general de todos los elementos fundamentales que iniciaron este trabajo de investigación así como la problemática existente, la propuesta que hemos realizado y los objetivos generales de la investigación.

El capítulo 2, se hace una completa introducción a los entornos distribuidos, donde se presentan las características más relevantes de este tipo de entornos, las arquitecturas existentes sus ventajas y desventajas al momento de trabajar con ellas.

El capítulo 3, se definen los principales gestores de workflow tomados como referencia para nuestro trabajo de investigación, también se detallan las políticas de planificación que tenemos disponibles en la literatura actual, y que forma una pieza clave en la investigación realizada.

El capítulo 4, presenta **SchedFlow** tanto a nivel de arquitectura como en el modo de funcionamiento, también se definen claramente cada uno de los sus módulos, la funciones de la API que han desarrollado, y la interacción existente entre cada una de ellas.

El capítulo 5, se muestran los experimentos realizados usando SchedFlow, también se detallan las aplicaciones utilizadas, el entorno de ejecución donde se hicieron, y un completo análisis de los resultados obtenidos de cada escenario de experimentación planteado

El capítulo 6, contiene las principales conclusiones del trabajo de investigación realizado, así como las líneas abiertas que se pueden seguir tomando como referencia nuestra investigación

En este capítulo hemos visto una breve introducción de los elementos claves en la investigación, haciendo especial énfasis a los problemas que daremos solución, la propuesta que hacemos y los objetivos propios del trabajo. Se definen los aspectos principales de los gestores de workflow, aplicaciones workflow y los entornos en los que comúnmente se ejecutan estas aplicaciones.

Capítulo 2

Aplicaciones y Gestores de Workflow

En este capítulo queremos hacer una explicación más amplia de las aplicaciones workflow, para tener una idea general de los conceptos claves que fueron necesarios estudiar para poder llevar adelante toda nuestra investigación.

2.1. Definición de Workflow

Hay diversas definiciones sobre el significado y alcances de un workflow, en nuestra investigación se utilizará el significado propio de la gestión de procesos para aplicaciones computacionales de altas prestaciones.

Los workflow son aplicaciones que definen una manera de realizar la ejecución a lo largo de los niveles de dependencia existentes en ellos, para lo cual se establecen tareas previas que deben completarse exitosamente antes de comenzar un siguiente nivel de ejecución, que dará como resultado una salida que dependerá del tipo de aplicación y datos iniciales cargados para su ejecución.

La gestión de ejecución de una aplicación workflow suelen utilizar los lineamientos del Workflow Management Task [92], el cual nos permite la gestión de cualquier tipo de proceso o procedimiento, sea este dependiente o independiente. Esto nos garantiza que los procesos se ejecuten en un orden correcto evitando así fallos en la ejecución por inconsistencia de datos.

2.2. Taxonomía de un Workflow

La taxonomía caracteriza y clasifica las aplicaciones workflow en el contexto de la computación Grid. Como se muestra en la figura 2.1, consta de cuatro elementos: (a) El diseño de Workflow, (b) La programación de workflow, (c) La tolerancia a fallos (d) El movimiento de datos [100]. En esta sección, muestra cada elemento y su taxonomía ya que es importante comprender, en qué puntos nuestro trabajo inside en mejoras notables de acuerdo a esta taxonomía.

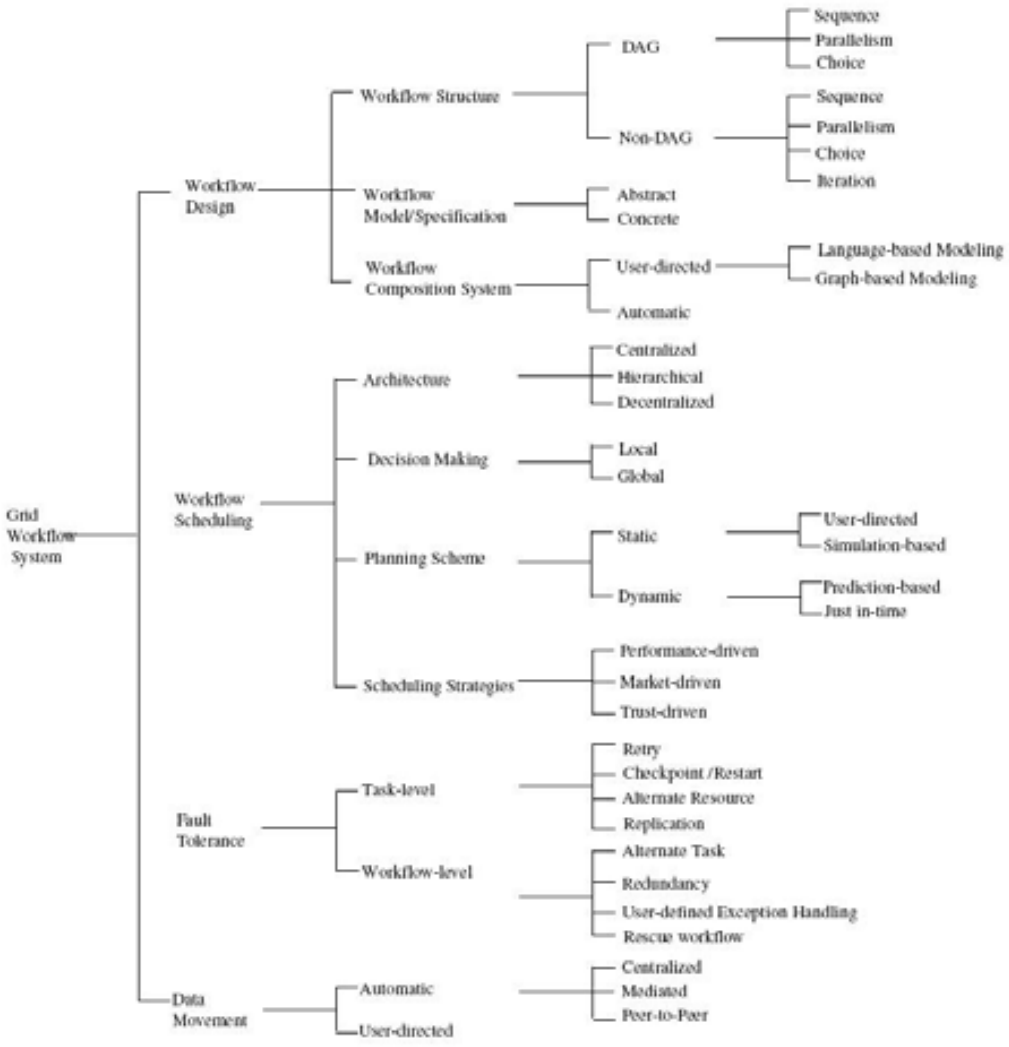

Figura 2.1: Taxonomía de los worflows científicos para sistemas distribuidos.

Cómo se puede ver esta taxonomía hace una clasificación muy completa de las aplicaciones workflow, puesto que las mismas tienen una enorme diversidad de formas, esto porque en el mundo tenemos muchos modelos físicos, químicos, industriales que podemos representar como una aplicación workflow, de tal manera que los aportes que realizamos en nuestro trabajo de investigación son extrapolable a cualquier rama de la ciencia que pueda tener una aplicación workflow.

2.2.1. Modelos de Workflow

Un modelo del Workflow (también llamado especificación del workflow) define un workflow incluyendo su definición de la tarea y la definición de la estructura. Los modelos de workflow se pueden clasificar en dos tipos los workflow abstracto, y los workflow concretos [24, 25]. Según [50] en la figura 2.2 vemos esa clasificación de los workflows.

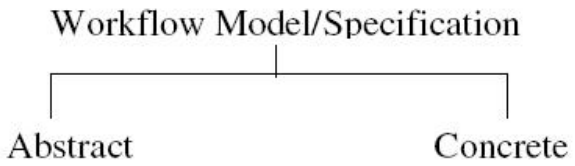

Figura 2.2: Modelos de un workflow.

En un modelo abstracto un workflow se describe en una forma abstracta en la que el workflow se especifica, sin hacer referencia específica a los recursos para la ejecución de las tareas. Un modelo abstracto proporciona una forma flexible para que los usuarios puedan definir workflows, sin preocuparse por los detalles de implementación de bajo nivel. Las tareas en un modelo abstracto son portátiles y se pueden asignar en cualquier servicio del entorno, incluso en tiempo de ejecución mediante el uso adecuado de los mecanismos de planificación.

El uso de modelos abstractos también facilita el intercambio de descripciones de workflow entre los usuarios de la red [25], donde particularmente salen beneficiados los participantes de Organizaciones Virtuales (VO). Una organización virtual en términos de Grid Computing, se refiere a un conjunto dinámico de recursos individuales, que tienen definidas un conjunto de normas para el reparto de los recursos cuando algún participante de la misma solicita ejecutar una determinada aplicación.[37].

Por el contrario, un modelo concreto une las tareas de flujo de trabajo a los recursos específicos. En algunos casos, un modelo concreto pueden incluir tareas

de movimiento de datos para transferirlos fuera de ella y hacerlos públicos dentro de una organización virtual [25]. En otras situaciones, las tareas de un modelo concreto también pueden incluir la aplicación necesaria para trasladar el código computacional para analizar a gran escala.

Una vez definidos los conceptos de modelado para un workflow, es importante señalar que dada la naturaleza dinámica del entorno de ejecución en los sistemas distribuidos, es más conveniente que los usuarios definan el workflow en un modelo abstracto, sencillamente porque no tenemos idea precisa de lo que puede pasar de un momento a otro. Si utilizáramos modelos concretos completos ó parciales tendríamos que tener un entorno más robusto como un clúster donde podemos tener el control de los recursos, sin embargo este trabajo esta completamente enmarcado en sistemas distribuidos no dedicados, por esa razón trabajaremos con aplicaciones workflow abstractas.

2.2.2. Estructura de un Workflow

Un workflow está compuesto por múltiples tareas conectadas entre si según las dependencias existentes en la aplicación. La estructura del workflow indica la relación temporal entre las tareas allí descritas. Un workflow se puede representar generalmente como DAG (Grafo Acíclico Dirigido)[78] ó un no-DAG (Grafo Acíclico no Dirigido), pero nuestro trabajo se centra en las aplicaciones workflow representadas como un DAG. Según [50] la figura 2.3 muestra la estructura de un workflow.

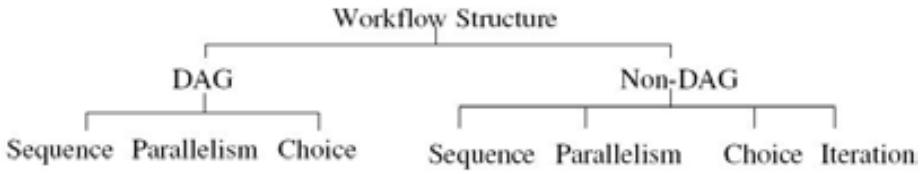

Figura 2.3: Estructura de un workflow.

En un workflow basado en grafos acíclicos dirigidos la estructura puede ser clasificada como secuencia, paralelismo, y elección. La secuencia está definida como una serie ordenada de las tareas, donde una tarea comienza después que la anterior se ha completado exitosamente. Paralelismo representa las tareas que pueden realizarse al mismo tiempo, en lugar de una serie. En el patrón de elección, una tarea se ha seleccionado para ser ejecutada solo cuando las condiciones asociadas a ella son verdaderas [100].

Además de todos los modelos que figuran en un grafo acíclico dirigido, también tenemos aquellos que funcionan como un grafo acíclico no dirigido (no-DAG), que también incluyen en su estructura las iteraciones en la que las secciones de algunas tareas permiten que se repita, creando un bucle sobre la misma tarea.

2.2.2.1. Terminología utilizada en la estructura de una aplicación workflow

- **Workflow**: Por definición es una aplicación con una estructura que está compuesta por un conjunto de tareas, que están vinculadas de forma dependiente por aristas que representan las transiciones del mismo.
- **Tarea**: Son todos aquellos elementos que componen de forma independiente un workflow, en ella están definidas las acciones, programas o subprogramas que se deben ejecutar, y que forman parte de la aplicación workflow.
- **Transición**: Se define como la transición entre las diversas tareas que componen el workflow y que tienen un coste de comunicación para enviar los resultados al siguiente nivel de ejecución.
- **Niveles**: Son todas las fases que debe ejecutar el workflow para obtener el resultado final, múltiples actividades se pueden ejecutar inclusive en paralelo, si existen los recursos suficientes para tal fin.

2.2.3. Composición de un Workflow

Un workflow está diseñado para permitir a los usuarios ensamblar componentes en esta representación. Estos componentes se necesitan para proporcionar una vista de alto nivel en la construcción de aplicaciones workflow, para así lograr ocultar la complejidad subyacente de los entornos no dedicados. La figura 2.4 y según [50][44] muestra la composición de los workflows.

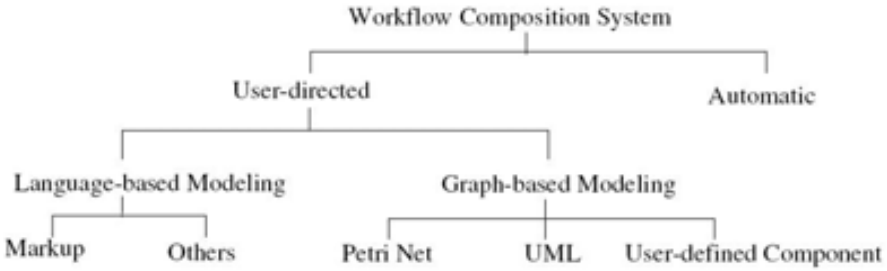

Figura 2.4: Composición de un Sistema Workflow

Esta representación tiene dos grandes caminos una dirigida por los usuarios, que son quienes editan directamente el workflow. Por otro lado tenemos la composición automática que generan el workflow de forma automática para los usuarios. En general, los usuarios pueden utilizar cualquier tipo de lenguaje para modelar una aplicación workflow, o utilizar herramientas gráficas para obtener la composición de estas aplicaciones.

2.2.4. Planificación de Workflow

Casavant [18] clasifica la programación de tareas en los sistemas de computación distribuida en programación de tareas "Local" y "global". La programación local implica el manejo y la asignación de tareas en un único recurso, mientras que la programación global consiste en decidir dónde ejecutar una tarea. Según [50] la figura 2.5 planificar un workflow implica tomar en cuenta cinco elementos importantes, como son la arquitectura, toma de decisiones, esquema de planificación, estrategias, y la estimación del rendimiento. Esto se traduce en buscar opciones que den soporte a cada elemento de una manera óptima, ya que si no los resultados pueden ser muy diferentes de los esperados.

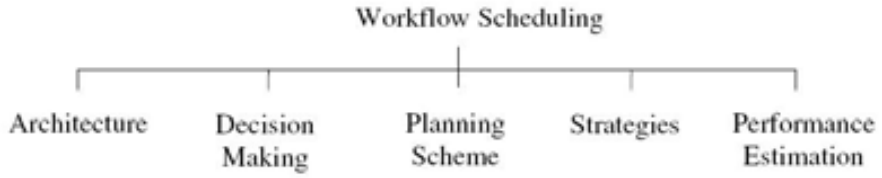

Figura 2.5: Planificación de un Workflow

De acuerdo con esta definición, la programación de workflow es una especie de programación de tareas global, ya que se centra en la gestión de la ejecución de tareas interdependiente de los recursos compartidos que no están directamente bajo su control.

El programador de workflow debe coordinar con los diversos sistemas la gestión tanto de recursos (que en un entorno no dedicado son heterogéneos), como de las políticas de planificación locales. Teniendo en cuenta las limitaciones de los usuarios es también importante en el proceso de programación, satisfacer las necesidades mínimas que los usuarios esperan de cada ejecución.

2.2.4.1. Arquitectura de planificación

La arquitectura en la planificación de un workflow es muy importante sobre todo para la escalabilidad, autonomía, calidad y rendimiento del sistema [46]. De ahí que tengamos tres grandes categorías de arquitectura para la programación de los workflow, según [50] la figura 2.6 estás pueden ser centralizadas, jerárquicas, o descentralizadas.

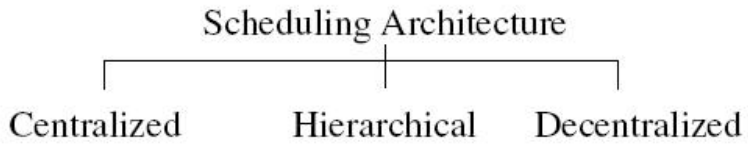

Figura 2.6: Arquitectura de Planificación de Workflow

En un esquema de workflow centralizado, el planificador central realiza la toma de decisiones para todas las tareas. El planificador tiene la información sobre el workflow y recoge información de todos los recursos disponibles. Se cree que el sistema centralizado puede ser muy eficiente porque tiene toda la información necesaria.

El principal problema es que este esquema no es escalable con respecto al número de tareas, las clases, y el número de recursos del entorno no dedicado. Por lo tanto un sistema adecuado para ejecutar un workflow a pequeña o gran escala, cada tarea debe tener un mismo objetivo (un ejemplo de esto puedes ser que tengan el mismo tipo de recurso).

A diferencia de la planificación centralizada, la planificación jerárquica o descentralizada permite que las tareas sean planificadas por diferentes planificadores. Por lo tanto, un planificador sólo mantiene la información relacionada con un sub-workflow. Así, en comparación con la planificación centralizada es mucho más escalable, ya que limita el número de tareas gestionadas por un planificador.

Sin embargo, la mejor decisión para un workflow parcial puede inducir a tener un rendimiento por debajo del nivel óptimo para la ejecución del workflow general. Por otra parte, los conflictos y los problemas son más graves [7]. Un ejemplo de conflicto, es que las tareas de los diferentes sub-workflow planificados por diferentes planificadores pueden estar competiendo por el mismo recurso.

Para la planificación jerárquica, no hay un gestor de central sino varios de nivel inferior, que son los responsable de controlar la ejecución del workflow y asignar sub-workflow a los planificadores de bajo nivel. Por ejemplo, en el proyecto

GridFlow [17], el cual es un gestor de workflow con múltiples planificadores de nivel inferior. Donde cada planificador de bajo nivel es responsable de planificar cada una de las tareas en un sub-workflow donde los recursos son propiedad de un mismo dominio organizacional.

Las principales ventajas de la utilización de la arquitectura jerárquica es que la planificación de diferentes políticas, se puede implementar en una especie de gestor central [46]. Sin embargo, el hecho de que todo sea administrado de manera centralizada, puede ocasionar un fallo en el sistema si se hace una mala distribución de las tareas.

Por el contrario, en la planificación descentralizada existen múltiples planificadores sin un controlador central. En este cado cada planificador puede comunicarse unos con otros y planificar un sub-workflow a otro planificador con menor carga. En comparación con la planificación jerárquica, la planificación des-centralizada es más escalable, pero se enfrenta a más desafíos para generar soluciones óptimas que den un buen rendimiento a la ejecución de una aplicación workflow, y conseguir así reducir al mínimo los problemas de conflicto existentes entre ellos.

2.2.4.2. Planificación

No existe una solución única y mejor para la asignación de las aplicaciones workflow en los recursos, ya que las aplicaciones pueden tener características muy diferentes. Esto depende en cierta medida de los modelos utilizados para ser planificados.

En general, las decisiones sobre la asignación de tareas de un workflow en los recursos del entorno de ejecución, se puede basar en la información de la tarea actual o del workflow y esta a su vez puede ser de dos tipos, basada en decisiones locales o globales [24] como se muestra en la figura 2.7.

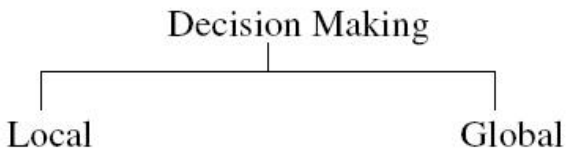

Figura 2.7: Modelos de toma de decisiones de un workflow.

La planificación basada en decisión local sólo tiene una tarea o workflow, con lo que suele se más sencillo hacer una buena planificación para la tarea actual,

o workflow secundario pero también podría reducir el rendimiento final de la ejecución de un workflow. Mientras que la planificación de tareas en un workflow utilizando decisión global, ofrece un mejor rendimiento de la ejecución del workflow. Existen algunos algoritmos para grafos dirigidos que permiten la planificación de tareas en paralelo como lo muestra [54] con el método de análisis prospectivo (FLAM).

Un detalle importante en este esquema es que a veces puede tardar mucho más tiempo en la planificación de la tarea, que en realizar la ejecución de la misma y esto hay que tenerlo presente, para que la sobrecarga producida por la planificación global logre reducir el beneficio general e incluso muy parecidos a los que se pueden obtener en la planificación local.

2.2.4.3. Planificación por esquema

La planificación por esquema es un método para convertir los workflow abstracto en workflow concretos, logrando que la complejidad de la aplicación sea menor. Cómo se muestra en la figura 2.8, los sistemas para la planificación de una aplicación workflow se pueden clasificar en sistemas estáticos o dinámicos. En un sistema estático, los modelos concretos tienen que ser generados antes de la ejecución de acuerdo con información actualizada sobre el entorno de ejecución y el estado cambiante dinámico de los recursos no se tiene en cuenta.

Por el contrario en un esquema dinámico, se utiliza tanto la información dinámica y estática, sobre los recursos para hacer la planificación basadas en análisis de las decisiones en tiempo de ejecución.

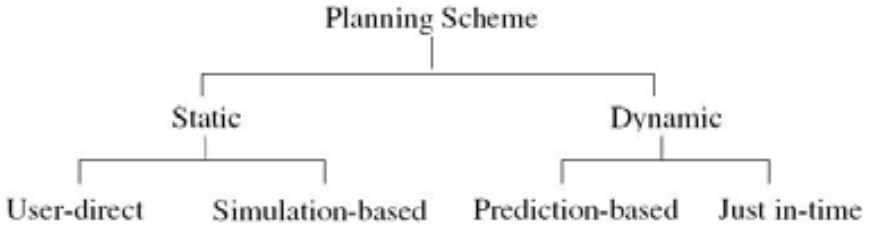

Figura 2.8: Esquema de planificación de un workflow.

Los sistemas estáticos también conocido como planificación por delante, son dirigidas por el usuario y son basados en la simulación. En la planificación dirigida por el usuario, lo que suelen hacer es emular la planificación del proceso y tomar decisiones de asignación de recursos de acuerdo a sus conocimientos,

preferencia o criterios de desempeño. Por ejemplo, los usuarios prefieren mapear las tareas a los recursos en los que no han experimentado fracasos.

En la programación basada en simulación, el "mejor" programa se logra mediante la simulación de la ejecución de tareas en un determinado conjunto de recursos, antes de que un workflow inicie la ejecución.

La simulación puede ser procesada sobre la base de información estática, o el resultado de la estimación del rendimiento. Por ejemplo, en *GridFlow*, el "mejor" recurso seleccionado para la planificación de una tarea se basa en la predicción de tiempo de ejecución que prevé el recurso.

Los sistemas dinámicos incluyen la predicción basada en la planificación justo a tiempo. Donde la predicción de la planificación está ajustada a la información dinámica vinculada a los resultados. Es similar a la planificación estática basada en la simulación, en la cual el planificador está obligado a predecir el rendimiento de la ejecución de tareas sobre los recursos, para así generar un programa casi óptimo para la tarea antes de que comience la ejecución.

Sin embargo, cambios en la planificación inicial pueden ser soportados de forma dinámica durante la ejecución. Por ejemplo *GrADS* [52] genera un mapeo preliminar mediante el uso de la predicción de resultados, pero puede migra la ejecución de una tarea a un mejor recurso cuando este ha fallado o un mejor recurso se encuentra disponible.

Sakellariou [77] desarrollo una replanificación de bajo costo para las políticas de planificación de workflow en el Grid. Se considera la replanificación de las tareas de un workflow cuando la misma puede ser reasignada durante la ejecución de una forma dinámica ajustándose a los cambios del entorno.

El problema es que no todas las replanificaciones suelen ser exitosas, sobre todo cuando la heterogeneidad del entorno es muy grande. Sin embargo se pueden establecer unos recursos pre-seleccionados para estas tareas que necesiten una nueva planificación.

2.2.4.4. Estrategia de planificación

En general, la planificación de aplicaciones workflow en un sistema distribuido es un problema **NP-completo** [32]. Por lo tanto, muchas heurísticas han sido desarrollados para obtener soluciones casi óptimas para que coincida con las limitaciones presentadas por los usuarios.

En la figura 2.9, y según [50] nos categorizan las estrategias de los principales enfoques de planificación, en función de los resultados obtenidos

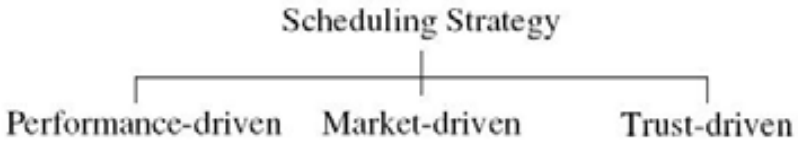

Figura 2.9: Estrategia de planificación de un workflow.

Las estrategias en función de los resultados tratan de encontrar una asignación de tareas de los workflow en los recursos que logran un rendimiento óptimo de ejecución, uno de los objetivos principales es reducir al mínimo el *makespan* (tiempo general de ejecución).

La mayoría de los sistemas de planificación de workflow entra en esta categoría, por ejemplo *GrADS* [52] intenta optimizar los workflows utilizando la heurística Min-Min [75], con la esperanza de obtener los tiempos mínimos de finalización en cada ejecución pero la verdad es que muy pocas veces lo consigue.

Dado que la mayoría de las aplicaciones están basadas en el rendimiento, toda nuestra investigación ha sido centrada en conseguir integrar las políticas de planificación en los gestores de workflow, e intentar aportar algunas mejoras a los resultados aportados por las diferentes políticas de planificación que encontramos en la literatura.

2.2.4.5. Estimación del rendimiento

Los planificadores de workflow puedan predecir cómo las tareas se comportarán en los recursos heterogéneos distribuidos, y así tomar decisiones sobre cómo y dónde ejecutar las tareas. Como se indica en la figura 2.10[50], hay varios métodos de estimación de resultados: simulación, análisis de modelos, datos históricos, aprendizaje en línea, e híbridos.

Los enfoques de simulación [29] proporcionan entornos de recursos para emular la ejecución de tareas del workflow antes de su ejecución real. En el análisis de modelado [22, 20], un planificador predice el desempeño de las tareas del workflow en un determinado conjunto de recursos sobre la base de un análisis métrico.

Los datos de aproximación histórica [80] se basan en datos históricos para predecir el desempeño de la tarea de ejecución. Los datos históricos relacionados con el rendimiento de las aplicaciones de un usuario o experiencia del mismo. También se puede utilizar para predecir el porcentaje de los recursos disponibles

para ese usuario. El aprendizaje en línea es un enfoque que predice el desempeño de tareas de ejecución, basándose en la experiencia en línea sin pero obviando el conocimiento previo de la dinámica del entorno.

Los enfoques de aprendizaje histórico y en la línea de uso datos experimentales, pueden definirse en general como en métodos de modelización empírica para el rendimiento de estimación, en ciertas condiciones estos enfoques pueden son utilizados juntos formando el enfoque híbrido para la generación de la evaluación del desempeño de las tareas de un workflow.

Figura 2.10: Estimación de rendimiento

2.2.4.6. Tolerancia a fallos

En un entorno distribuido la ejecución de un workflow puede no realizarse por varias razones: la variación en la configuración del entorno de ejecución, la no disponibilidad de los servicios requeridos o de componentes de software, las condiciones de sobrecarga de los recursos, el sistema se quede sin memoria, y fallas en los elementos de cómputo o de red.

La gestión de workflow debe ser capaz de identificar y controlar los errores, para poder realizar la ejecución con cierto grado de fiabilidad ante la presencia de estos posibles fallos que suelen ocurrir en estos entornos.

En la figura 2.11, Hwang [50] divide el manejo de fallos de los workflow en dos niveles diferentes, por una lado a nivel de tarea, y por el otro a nivel de workflow.

Las técnicas a nivel de tareas enmascaran los efectos de la falta de ejecución de las tareas del workflow, mientras que técnicas a nivel de workflow manipulan la estructura del mismo para hacer frente a las condiciones de error. Estas técnicas de nivel han sido ampliamente estudiadas en los sistemas paralelos y distribuidos.

Estos pueden ser catalogados en varios sub-grupos entre los que tenemos "reintento", "búsqueda de recursos alternativos", "reinicio de tarea", o "replicación de tarea".

- **Reintento**: es una de la técnica más simple de implementar ante la falta de recuperación de una tarea que falla, ya que simplemente intenta ejecutar la misma tarea en el mismo recurso tras el fallo detectado [2].

- **Búsqueda del recurso alternativo**: esta es una técnica que va un poco más allá pues intenta obtener algún recurso del entorno que se encuentre disponible y lanzar nuevamente la tarea fallida a otro recurso [2].

- **Reinicio**: es una técnica [1] que mueve las tareas de forma transparente a los recursos, de modo que la tarea pueda continuar su ejecución desde el punto de falla.

- **Replicación**: este procedimiento lo que hace es ejecutar la misma tarea de manera simultánea en diferentes recursos de la red, para garantizar la ejecución de esas tareas partiendo del supuesto que las dos máquinas utilizadas no fallen al mismo tiempo [3].

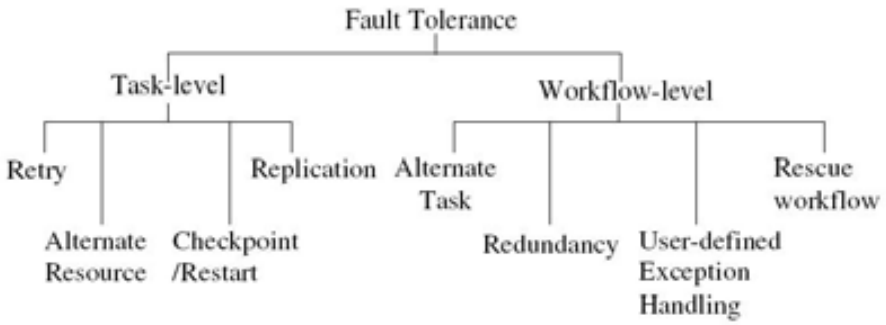

Figura 2.11: Tolerancia a fallos de los workflow.

Uno de los gestores muy utilizado es Cóndor DAGMan quien ha desarrollado La técnica de rescate pero el problema es que si falla este no seguirá ejecutando el workflow y se corre el riesgo de perder un importante tiempo si el fallo se produce antes de finalizar.

En nuestro trabajo de investigación desarrollamos un *framework* de capa superior con un potente mecanismo de replanificación, capaz de soportar y replanificar en tiempo de ejecución la tarea, sin perder más que el tiempo de replanificación y puesta en marcha de esta tarea que ha fallado.

2.2.4.7. Movimiento de datos

Para las aplicaciones workflow los archivos de entrada de las tareas son por etapas, y las mismas no pueden ejecutarse hasta que los datos han llegado desde las tareas predecesoras.

Del mismo modo los archivos de salida de estas últimas pueden ser requeridos por sus tareas de nivel inferior, que pueden estar siendo ejecutadas en otros recursos. Por lo tanto, los datos intermedios tienen que ser por etapas a los correspondientes puntos del entorno distribuido donde estamos ejecutando la aplicación.

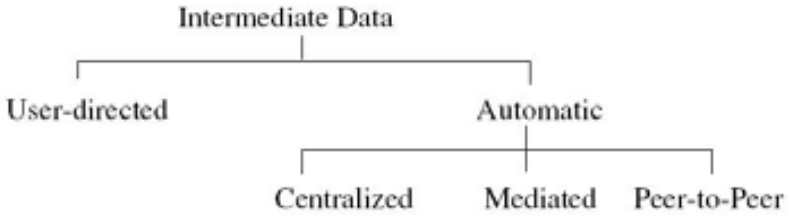

Figura 2.12: Movimiento de datos en un workflow.

Algunos sistemas requieren que los usuarios gestionen la transferencia de datos intermedios, en lugar de proporcionar mecanismos automáticos de transferencia de datos. Como se indica en la Figura 2.12[50], tenemos una clasificación de estos enfoques para el movimiento automático de datos intermedios, los mismos pueden ser centralizado, con mediación, o peer-to-peer.

Básicamente un enfoque centralizado de las transferencias de datos intermedios entre los recursos suelen ser realizado a través de un punto central. Los enfoques centralizados son fáciles de implementar y adaptarse a las aplicaciones workflow. En un enfoque mediado, en lugar de utilizar un punto central la ubicación de los datos son manejados por un sistema de gestión de datos distribuido.

Finalmente un sistema *peer-to-peer* para la transferencias de datos es un enfoque que envía los datos entre los recursos que procesan las aplicaciones, aunque puede ser el más eficiente no son sencillos de implementar, pero si el workflow a ejecutar es de gran escala puede llegar a ser la única solución viable.

2.3. Estados de una tarea en un workflow

En la siguiente sección veremos las definiciones de los estados por los que normalmente debe parar una tarea, al momento de ejecutar un workflow. Estos

estados son los que marcan el inicio, ejecución y resultado de las tareas, y que son de vital importancia para poder gestionar correctamente el proceso de planificación de una aplicación workflow.

A continuación se definen los 5 estados que en nuestra investigación son relevantes, ya sea para tomar decisiones o simplemente para monitorizar contantemente todos los estados de las tareas.

- **Estado no iniciado**: Un estado "no iniciado" se define como aquel en el las tareas de la aplicación del workflow están planificadas pero no han sido iniciadas en ningún recursos, ya sea porque no se han completado las dependencias o porque no hay recursos disponibles para poder ser enviada a la cola del gestor.

- **Estado en ejecución**: Un estado "en ejecución", viene definido como aquel en el que las tareas de la aplicación workflow han sido envidas al recurso y su ejecución ha comenzado.

- **Estado completado**: Un estado "completado", es aquel en el que las tareas que han sido envidas al recurso de cómputo, han finalizado con éxito la ejecución.

- **Estado suspendido**: Un estado "suspendido", se define como aquel en el que la tarea una vez enviada al recurso de cómputo, es suspendida por un proceso de mayor jerarquía, evitando la ejecución de la tarea por un período de tiempo no definido.

- **Estado abortado**: Un estado "abortado", es aquel en el que las tareas de la aplicación workflow, una vez iniciada la ejecución en el recurso por algún motivo es abortada, ya sea por el usuario o por que el recurso haya fallado.

En la figura 2.13 se muestra el diagrama de estados de un workflow [31].

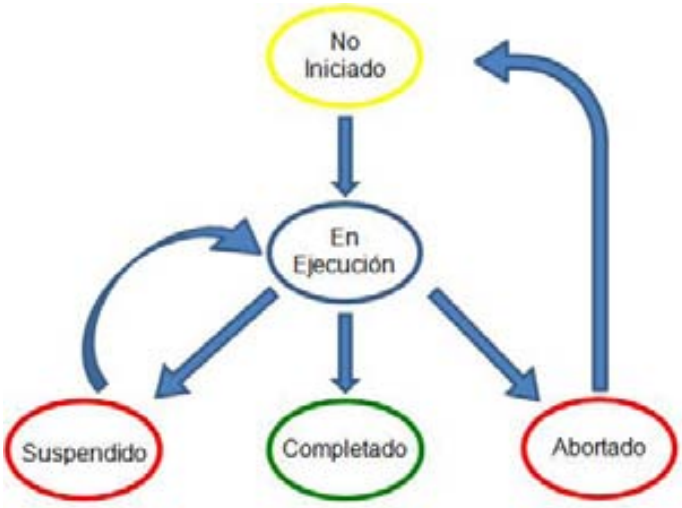

Figura 2.13: Estados de una tarea de un workflow.

2.4. Gestores de Workflow

2.4.1. Condor-DAGMan

Una de las herramientas más utilizada que brinda soporte a la ejecución de aplicaciones workflow es el gestor DAGMan [56]. DAGMan (Direct Acyclic Graph Manager) es un meta-planificador para los trabajos de condor, DAGMan realiza el envió de las tareas a Condor en un correcto orden que representa a la aplicación workflow, pero también realiza la asignación de estas tareas para que sean procesadas exitosamente. Esto lo hace siempre que un recurso del entorno de ejecución se encuentre disponible para llevar a cabo esa ejecución.

DAGMan realiza el envió de la tarea al sistema de colas de condor, quién a su vez utiliza su *matchmaking* [85] [11] para finalmente enviar la tarea al recurso que se encuentre disponible en el entorno de ejecución. DAGMan por su lado realiza la planificación del workflow dinámicamente para cada nivel de ejecución, de modo que los componentes del workflow que se encuentren en el estado listo para ejecución, puedan ser enviados para comenzar su ejecución.

DAGMan específica las dependencias entre los trabajos y Condor las ejecuta, con lo cual si se poseen dependencia entre dos tareas cualesquiera, donde se necesita ejecutar antes una tarea X, entonces Y deberá esperar hasta que X culmine exitosamente. DAGMan se encarga de manejar este tipo de situaciones sin mayor

problema, pues ya ha sido utilizado muchísimas veces por la comunidad científica que trabaja con las aplicaciones workflow.

En la actualidad muchos de los sistemas existentes utilizan a DAGMan como gestor, ya que el mismo representa una herramienta muy completa y segura en cuanto al manejo de dependencias para aplicaciones workflow, dando así una mayor fiabilidad al momento de ejecutar las aplicaciones workflow, dejando a un lado el problema de dependencias y para poder enfocar su esfuerzo a la política de planificación.

La política de planificación que DAGMan tiene por defecto es la aleatoria, con lo cual no aplica ningún mecanismo sofisticado que logre mejorar eficientemente el rendimiento de la aplicación, sin embargo como el manejo de dependencias es bueno, se ha popularizado entre los investigadores de workflows.

2.4.2. Taverna

Taverna es el sistema de gestión de workflow desarrollado por el proyecto myGrid [72], que tiene como objetivo explotar la tecnología Grid para desarrollar un middleware de alto nivel para apoyar de forma personalizada los experimentos de biología.

El propósito de Taverna se utiliza para ayudar a los científicos con el desarrollo y ejecución de workflows bioinformáticos en el Grid. Taverna proporciona modelos de datos, extensiones y las interfaces gráficas para los usuarios.

FreeFluo [72] también está integrado en Taverna como un motor de workflow para apoyar la transferencia de datos intermedios. En Taverna los modelos de datos pueden ser representados ya sea en un formato gráfico o en un lenguaje XML.

Al igual que los otros sistemas gestores tiene una política de planificación basada en la velocidad de la CPU, esto resulta particularmente útil si puedes asignar todas las tareas con mayor volumen de cómputo a las máquinas más potentes, pero la realidad es que al ejecutar con este gestor de workflow los resultados son bastantes diferentes de los que se pueden predecir utilizando simulación.

2.4.3. Karajan

Karajan desarrollado por el Laboratorio Nacional Argonne, tiene como objetivo proporcionar un sistema integrado para la ejecución de workflow. El mismo es un *framework* que puede ser fácilmente utilizado por terceros para proveer

soluciones en la ejecución de aplicaciones workflow. Se deriva de GridAnt y proporciona capacidades de escalabilidad y control de errores [96, 97].

Karajan es parte de Java COG kit. Java COG Kit se basa en un diseño modular y proporciona mecanismos para el desarrollo de aplicaciones rápido y fácil integración a los middleware del Grid.

Los elementos utilizados para la descripción de las tareas de flujo de trabajo son definibles por el usuario, de esta manera no tenemos limitación al momento de utilizar algún parámetro específico.

Su política de planificación está basada en la selección cíclica de recursos, esta selección la hace desde un fichero, donde va marcando los recursos uno a uno hasta finalizar la lista, y vuelve de nuevo a empezar con el recurso inicial sin actualizarlo. Esto trae el problema de asignar tareas a recursos no disponibles, o que estén siendo utilizados por otra tarea de la misma ejecución.

2.4.4. Triana

Triana es un entorno de datos visual orientado para el análisis y desarrollo de workflow creado en la Universidad de Cardiff. En 2002, Triana se extendió a aplicar un modelo para el Grid utilizando un enfoque peer-to-peer [86] .

Recientemente, Triana ha sido rediseñado e integrado con sistemas Grid a través de la interfaz GridLab GAT (Grid Application Toolkit) [4]. GAT define una API de alto nivel para el acceso de servicios básicos a través Grid JXTA [30], y los servicios web OGSA [91].

Triana ofrece una interfaz de programación visual con funcionalidad representada por unidades. Las aplicaciones se escriben arrastrando las unidades requeridas en el lugar de trabajo y se conectan para construir un workflow.

La política de distribución de triana se basa en el concepto de unidad de grupo y se aplican dos políticas, una paralela y otra de punto a punto ambas son generalmente muy lentas, aunque la punto a punto si los recursos son buenos puede dar mejores resultados en algunas ejecuciones.

2.4.5. WebFlow

Este sistema está diseñado para regular el flujo de tareas a lo largo de la ejecución del workflow, para manejar en línea las peticiones del cliente. Este gestor de worklow en línea verifica como fase inicial si se encuentra un flujo nulo es decir una tarea que no contenga sucesores. Si este no existe entonces da comienzo a

la ejecución de un nuevo nivel de flujo a través de la web para la gestión del workflow.

El mismo utiliza JDOM que es una librería de código fuente para manipulaciones de datos XML optimizados para java, con esto puede tener acceso al fichero de entrada definido en XML (lenguaje de marcas extensible) donde esta descrito el workflow, con las definiciones de tareas que se deben ejecutar.

Se proporcionan varios métodos para la ejecución del workflow uno es por flujo horizontal, donde todas las tareas del nivel deben completarse ó de acceso aleatorio donde se ejecutan cualquier tarea en un servidor web disponible.

Una vez que se haga esto y todo el estado de las tareas se haya verificado correctamente finalizado, se toman las decisiones de las tareas del siguiente nivel que se ejecutaran esto lo repite en cada nivel hasta finalizar el workflow exitosamente.

El estado está asociado a un estado del nivel de ejecución se almacena en un fichero histórico, el cual proporciona información al servidor para que planifique el siguiente nivel del workflow. Este servidor web es el responsable de la gestión correcta de las tares a ejecutar del workflow definido por el usuario. Este sistema utiliza una política de planificación estática definida mediante un histórico, y solo utiliza como parámetro el tiempo de ejecución más pequeño, pero no generaliza sobre si la tarea realmente tenía un gran volumen de cómputo, muy poco fiable al momento de ejecutar en los entornos muy heterogéneos.

2.4.6. Kepler

Kepler [5, 60] es uno de los sistemas workflow populares con características avanzadas para componer aplicaciones científicas. Se deriva del sistema de Ptolomeo II [57] y en la actualidad en desarrollo a través de una serie de proyectos científicos de gestión de datos.

Además de una fácil interfaz de usuario y un código abierto extensible a diferentes plataformas, este sistema permite una composición de componentes independientes para las aplicaciones workflow que se comunican a través de interfaces bien definidas.

Este diseño modular permite que los diferentes modelos de ejecución o los mecanismos a implementar sean fácilmente conectados a los workflow sin cambiar ninguno de los componentes del workflow.

El detalle es que la política de planificación que utiliza por defecto es random, y si bien puede ser muy amigable para integrar servicios, el mismo todo lo ejecuta

sobre Internet, con lo cual se hace necesario crear todo un web service para poder gestionar políticas diferentes, es por eso que este sistema no fue utilizado como parte en nuestra investigación.

2.4.7. GriPhyN - Pegasus

El gestor de workflow Pegasus [62] fue concebido del proyecto de GriPhyN [25] se ocupa de automatizar la generación de los workflow y ejecución de trabajos en Grid. Estos workflow se describen mediante la ejecución de un número complejo de componentes, todos construidos individualmente para su realizar su ejecución.

El generador del workflow concreto (CWG), está basado en un workflow abstracto definido en términos de los componentes de cada nivel y del uso de workflows concretos, definidos en términos de ejecutables y nombres de archivos de entrada-salida. El workflow concreto traza sobre el sistema de recursos disponibles en el Grid, y utiliza la infraestructura y componentes de Globus para el descubrimiento de los recursos (MDS).

Los puntos de transferencia de datos se agregan en el workflow por si es necesario para el siguiente nivel. Una vez que se genere el workflow concreto final, se ejecuta el workflow usando el Condor-G y DAGMan [25].

2.4.8. GridFlow

El gestor GridFlow [17] realiza la gestión del workflow basado en el paso de mensajes entre los trabajos dependientes. Actúa generalmente como un planificador de propósito jerárquico a la hora de ejecutar el workflow.

Dada la característica de planificación que posee con su uso jerárquico, esto representa un problema ya que existen sub-workflows que pueden desplazar de alguna manera la ejecución de una tarea en un recurso determinado. Los recursos también son vistos de manera jerárquica, y el Grid es visto como un esquema global o local para la ejecución. Hay un paso importante antes de la ejecución del workflow y es programar localmente los sub-workflows que se pretenden lanzar.

El workflow busca una planificación secundaria a través del módulo titan, para así obtener todo el esquema de tareas a ejecutar en el GridFlow. La salida del workflow es elegida y se configura con el tiempo más elevado, este último con sus predecesores hasta completar el workflow.

Los detalles del workflow son enviados al servicio de infomación de Globus, hasta que se completa la ejecución del workflow. Este sistema se utiliza sobre

todo para optimizar la ejecución del workflows con múltiples tareas dentro de los recursos, lo que permite hacer la separación del workflow grande a subworkflows más pequeños.

2.4.9. AppLes

El desarrollo inicial de AppLes es como aplicación gestora de workflow, tiene la particularidad de describir, cada uno de los niveles de planificación para la porción que se selecciona a lo largo de la ejecución, seleccionando primero la política de planificación que se desea utilizar.

Posteriormente el módulo actuador una vez recibidas cada una de las tareas asociadas al estado de listas para ejecutar, realiza la transferencia de los mismos a los sistemas de colas los cuales se encargaran de enviar los trabajos a los recursos y dando uso sintético a los recursos del Grid.

Este gestor fue concebido fundamentalmente para ejecutar SARA, una aplicación desarrollada que ameritaba ser ejecutada con diferentes políticas y diferentes sistemas de colas.

Este gestor demuestra un uso efectivo en la ejecución pero al igual que la mayoría de las herramientas existentes, giran la brújula en la constitución de políticas específicas de la literatura, sin embargo esta herramienta muestra de manera explícita la eficacia del uso de planificadores para entornos distribuidos ó para entornos Grid con múltiples usuarios [12].

2.4.10. TrellisDAG

El sistema TrellisDAG que es el encargado de realizar la cargar de los trabajos que se necesitan ejecutar, este realiza la verificación de las dependencias que se necesitan completar funciona con el algoritmos HEFT, para poder lanzar los workflow para su posterior ejecución en el sistema es especialmente útil que la descripción este dentro de los Makefiles que permiten hacer el envió de los trabajos. Los mecanismos que proporciona se canalizan en aquella dependencia que sea de prioridad mayor de acuerdo al tiempo de cómputo [40].

Las políticas de planificación proporcionan una programación detallada de las tareas a ejecutar. Las dependencias de los trabajos actúan bajo DAGMan como gestor de workflow.

Las dependencias cíclicas en el workflow, puesto que se apoya sobre todo debido a la arquitectura interpretativa que posee. Se debe tener cuidado en estas

dependencias cíclicas son manejadas en el módulo de translation, y esta es una característica del TrellisDAG pues acepta dependencias sobre los mismos recursos para la ejecución de las tareas.

Adicionalmente su infraestructura como se puede ver en la capa de *"translation and submission"*, no proporcionan ninguna ayuda que permita al usuario integrar un algoritmo de planificación que considere más adecuado para ejecutar la aplicación. TrellisDAG solo posee en su arquitectura la política incluida por defecto, y resultaría realmente complicado incluir otra sin que esto represente modificar la arquitectura del mismo.

En este capítulo hemos mostrado a fondo los fundamentos teóricos de las aplicaciones workflow, también se han detallado el funcionamiento de los gestores de workflow más importantes. Otro aspecto que se ha detallado es la taxonomía, y modelos de workflow que podemos encontrar en la literatura, para poder dar una idea más clara al lector.

Capítulo 3

Políticas de Planificación para Workflow

En este capítulo explicaremos las políticas de planificación más utilizadas en la actualidad, en en las cuales la comunidad científica esta dedicada para obtener nuevas mejoras en la ejecución de aplicaciones workflow.

Actualmente existen numerosos grupos de investigación trabajando en la planificación workflow, todo ello con la finalidad de brindar siempre a los usuarios mejores estrategias que logren minimizar el tiempo de ejecución de la aplicación utilizando sus heurísticas. Para conseguirlo se apoyan en características claves en la ejecución de una aplicación workflow.

Algunas de esas características son el tiempo de cómputo, tiempo de comunicación, estimación inicial, camino crítico, dependencia de tareas entre otras. Por existen infinidad de políticas de planificación, algunas abordan tiempo de cómputo, otras están enfocadas al tiempo de comunicación, también están las que se basan en mejorar la ejecución del camino critico, pero todas van tras un objetivo en común reducir el *makespan* de la aplicación, trabajo difícil de conseguir cuando el entorno de ejecución es muy dinámico.

Los problemas generales de todas estas políticas, es que cuando funcionan bien para una determinada aplicación, no está garantizado que sea igualmente eficiente para otra, o incluso si se varia algún parámetro de la aplicación, los resultados finales pueden ser totalmente diferentes a los esperados.

Teniendo en cuenta este interés por parte de la comunidad científica, señalaremos algunas de las políticas comúnmente utilizadas para la ejecución aplicaciones workflow, y por esta razón nuestro *framework* ha sido utilizado para ver el

comportamiento de estas heurísticas independientemente del gestor de workflow utilizado.

3.1. Planificación de un Workflow

Existen numerosas aplicaciones científicas con niveles de rendimiento y escala sin precedentes, pero no existen medidas claras o estándar para las aplicaciones workflow, que son consideradas una parte más del mundo científico, por tal motivo existen se hace necesario estudiar un gran número de políticas con las cuales poder realizar la planificación de aplicaciones workflow.

Por tal motivo se hace necesario el estudio de estas políticas para apoyar a los investigadores, las mismas se deben analizar y ejecutar bajo distintos entornos de ejecución para poder tener una idea de su comportamiento. En nuestro caso tomaremos particular interés de aquellas políticas de planificación que benefician en si a las aplicaciones workflow ejecutadas en entornos distribuidos.

Recordemos que la planificación de un workflow consiste en la asignación de un conjunto de tareas a diferentes recursos, definiendo orden donde se debe respetar las dependencias existentes entre las tareas que lo componen. Para lograr la exitosa ejecución de las tareas existen tres elementos importantes para la planificación:

- **Recursos**: Son todos los elementos de cómputo que se tienen disponibles en el entorno de ejecución donde se pueden asignar las tareas, estos recursos deben ser capaces de procesar una tarea en un determinado tiempo no establecido con anterioridad.
- **Gestor de Recursos**: El tercer elemento que no hace falta es el gestor de recursos, cuyo objetivo es proporcionar en un instante determinado los recursos disponibles del entorno de ejecución, información que necesita el planificador para poder completar el *mapping* entre las tareas y los recursos.
- **Planificador de tareas**: Este elemento es la pieza clave dentro de la planificación, ya que el mismo posee el algoritmo capaz de realizar el *mapping* de las tareas a los recursos disponibles del entorno de ejecución.

Las tareas son planificadas por una determinada heurística, quien se encarga de asignar distintos recursos a cada tarea utilizando un protocolo de acceso a los recursos. La planificación debe ser lo suficientemente buena para garantizar

un uso adecuado de los recursos, con el fin de poder garantizar los requisitos de la aplicación workflow. Un elemento planificación tiene dos piezas claves que debemos mencionar:

- **Algoritmo de planificación:** Este elemento de la plnificación define las maneras en que las tareas serán enviadas a los recursos de cómputo, utilizando para ello variables ya establecidas que seleccionan un determinado recurso, ajustando lo mejor posible los requisitos de ejecución de esa tarea.
- **Método de análisis:** Este elemento permite calcular el comportamiento temporal del sistema, así se puede comprobar si los requisitos temporales están garantizados en todos los casos posibles, en general se estudia el peor comportamiento para predecir posibles errores en la planificación.

Existen diferentes algoritmos para la planificación de tareas de una aplicación workflow, a continuación mencionamos algunos de ellos dando una breve definición de los mismos para que el lector tenga más claro cómo funcionan estos algoritmos de planificación:

- **Planificación Estática:** Las políticas de planificación estáticas, son aquellas en las que los parámetros a evaluar son fijos y se encuentran calculados antes de comenzar la ejecución de las tareas.
- **Planificación Dinámica:** Las políticas de planificación dinámicas, son aquellas en las que los parámetros a evaluar pueden sufrir algunos cambios durante la ejecución de las tareas, y que son más difíciles de gestionar.
- **Planificación Hibrida:** Las políticas de planificación hibridas, son aquellas en las que al iniciar la planificación se conocen los parámetros antes de iniciar la ejecución, pero pueden ir cambiando durante el proceso de ejecución de las tareas.

Para entender un poco el uso de los diferentes algoritmos que podemos encontrar hoy día en la literatura, daremos la explicación de cada algoritmo con un ejemplo muy sencillo para orientar en la justa medida lo que hace cada una de ellas. La planificación de ejemplo viene estará ajustada a por un workflow sintético que tiene las siguientes características en su estructura.

Tres niveles de dependencias, distribuidos en cuatro sencillas tareas, donde cada tarea solo consume un determinado tiempo de cómputo y una pequeña cantidad tiempo para la transferencia de los ficheros de salida. El ejemplo busca fundamentalmente mostrar la planificación que realiza cada algoritmo según el modelo teórico que posee, sin entrar en grandes detalles.

El workflow está compuesto de cuatro tareas, donde el máximo nivel de paralelismo es dos, es decir que hay dos tareas que se pueden ejecutar de manera simultánea, siempre que existan los recursos disponibles en el entorno de ejecución.

La tabla 3.1 agrupa los tiempos de ejecución estimados para cada tarea del workflow, mientras que la figura 3.1 muestra la estructura del workflow que se debe planificar con las diversas políticas de planificación.

Recursos	Texec A	Texec B1	Texec B2	Texec C
M1	188 seg	225 seg	98 seg	115 seg
M2	172 seg	200 seg	59 seg	100 seg
M3	194 seg	215 seg	150 seg	144 seg
M4	235 seg	259 seg	194 seg	186 seg

Tabla 3.1: Tiempo de ejecución de las tareas en las máquinas.

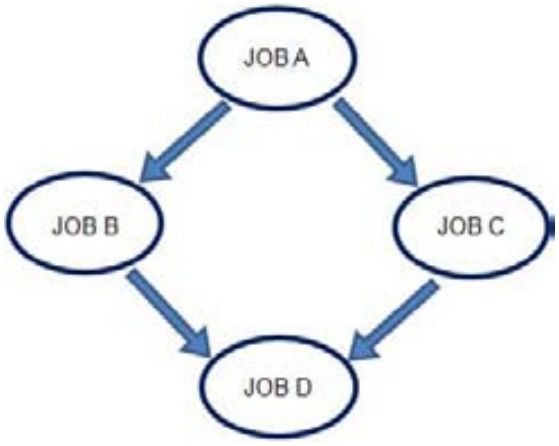

Figura 3.1: Workflow con cuatro tareas y tres niveles de ejecución.

Para este ejemplo utilizaremos un entorno de ejecución con cuatro máquinas con características diferentes, así la planificación final estará ajustada a la disponibilidad y el *benchmark* de los recursos de cómputo al cual se lanza las tareas. Hay que destacar que todos los algoritmos utilizados deben conocer al menos tres de los parámetros fundamentales para poder realizar su planificación exitosamente. Estos parámetros son los siguientes:

- **Tiempo de cómputo de las tareas (**Tcomp**)**: Este parámetro se define como el tiempo necesario que consume una tarea en particular, hasta que finaliza exitosamente su ejecución.
- **Tiempo de comunicación (**Tcomm**)**: Este parámetro está definido como el tiempo necesario que se consume, para transferir los ficheros de salida del nodo padre al nodo hijo.
- **El ancho de banda del entorno de ejecución** (Bw): Este parámetro viene definido como la cantidad de bits por segundos, con las que se cuenta a la hora de realizar el envío de las tareas, ficheros de entrada / salida de la aplicación workflow.

3.1.1. Algoritmo de Planificación Aleatorio

El algoritmo de planificación aleatorio [58] está basado en la asignación de las tareas a los recursos disponibles del entorno de ejecución, en el cual no existe ningún criterio específico, salvo el recurso se encuentre disponible. Los pasos que sigue este algoritmo para realizar la planificación de una aplicación workflow son los siguientes:

1. Para cada tarea lista para ejecutar se debe encontrar un recurso disponible.
2. De estos recursos disponibles, debe elegir cualquiera de ellos y asignar la tarea correspondiente.
3. Actualizar las máquinas disponibles en el entorno de ejecución.
4. Repetir los pasos 1, 2, y 3 hasta que no queden tareas por asignar de la aplicación workflow que se encuentra ejecutando.

En la siguiente tabla 3.2 se muestran los resultados obtenidos con la planificación aleatorio para nuestro ejemplo, obteniendo un tiempo total de ejecución con este algoritmo de **634 segundos**, utilizando la máquina que el algoritmo encuentra sin analizar si la tarea es más compleja o no.

3.1.2. Algoritmo de Planificación Min-Min

Este algoritmo de planificación Min-Min [71] asigna la tarea al recurso que tiene el mínimo tiempo estimado de finalización (TEC) que encuentra disponible. A partir de ahí construye un modelo (C, R, T), donde C es el componente

Tareas/Máquina	Planificación (Aleatoria)
A/M2	172 seg
B1/M4	259 seg
B2/M1	59 seg
C/M3	144 seg
tiempo total	634 seg

Tabla 3.2: Planificación Aleatoria del workflow.

(tarea), R es el recurso para el que se alcanza el mínimo tiempo y T es la TEC correspondientes.

En el siguiente paso, el componente que tiene el valor mínimo de la TEC se elige para planificar la siguiente tarea. Esto se realiza iterativamente hasta que todas las tareas han sido asignadas. La intuición detrás de esta heurística es que el *makespan* aumente menos en cada paso de interacción, con la esperanza de que el tiempo total de ejecución sea lo más pequeño posible. Los pasos a seguir este algoritmo de planificación son los siguientes:

1. Para cada tarea debe encontrar el tiempo de finalización más pequeño, y la máquina con la cual se obtiene este tiempo.

2. De estos tiempos, debe elegir el mínimo y asignar la tarea a la máquina correspondiente.

3. Actualizar las máquinas disponibles en el entorno de ejecución.

4. Repetir los pasos 1, 2, y 3 hasta que no queden tareas por asignar del workflow lanzado.

En la siguiente tabla 3.3 se muestra los resultados obtenidos con la planificación *Min-Min* de nuestro ejemplo, obteniendo un tiempo total de ejecución con este algoritmo de **546 segundos**, utilizando la máquina con mejor *benchmark* en la ejecución de casi todas las tareas.

Tareas/Máquina	Planificación (min-min)
A/M2	172 seg
B1/M3	215 seg
B2/M2	59 seg
C/M2	100 seg
tiempo total	546 seg

Tabla 3.3: Planificación Min-Min del workflow.

3.1.3. Algoritmo de Planificación Max-Min

Este algoritmo Max-Min [10] está muy estrechamente relacionado con la heurística Min-Min, su principal diferencia es que al realizar el cálculo del tiempo estimado de finalización (TEC), este selecciona el recurso con un valor máximo tiempo estimado de finalización (TCE) obtenido sobre todas las duplas para asignar la tarea.

La intuición detrás de esta heurística es que al dar preferencia a los más tareas que tarden más, hay esperanza de que las tareas con menos cómputo pueden ser solapadas con las que tardan más para así lograr reducir el *makespan* de la aplicación workflow que se está ejecutando. Los pasos que sigue para realizar la planificación de un workflow es la siguiente:

1. Para cada tarea debe encontrar el tiempo de finalización más grande, y la máquina con la cual se obtiene este tiempo.
2. De estos tiempos, debe elegir el máximo y asignar la tarea a la máquina correspondiente.
3. Actualizar las máquinas disponibles en el entorno de ejecución.
4. Repetir los pasos 1, 2, y 3 hasta que no queden tareas por asignar del workflow lanzado.

En la siguiente tabla 3.4 se muestra los resultados obtenidos con la planificación max-min de nuestro ejemplo, obteniendo un tiempo total de ejecución con el algoritmo Max-Min fue de **666 segundos**, utilizando la máquina con mejor *benchmark*, en la ejecución de casi todos los tareas.

Tareas/Máquina	Planificación (Max-Min)
A/M2	172 seg
B1/M2	200 seg
B2/M4	194 seg
C/M2	100 seg
tiempo total	666 seg

Tabla 3.4: Planificación Max-Min del workflow.

3.1.4. Algoritmo de Planificación Heterogeneous Earliest Finish Time (HEFT)

El algoritmo Heterogeneous Earliest Finish Time (HEFT) [90] esta divido en dos fases: una fase de prioritización de tareas, y otra fase de selección de

recursos basados en el rendimiento de los procesadores de estos. En la fase de dar prioridad a la tarea se calcula en valor *rank*, el cual se basa en el tiempo promedio de cómputo y comunicación de la tarea.

Una vez completada la fase inicial, entonces se procede a la selección del procesador para la tarea con la más alta prioridad, que viene producto del valor *Rank*. Esta tarea es elegida para la asignación de un procesador que garantiza los mejores tiempos de finalización para la misma. Los siguientes son los pasos a realizar para poder realizar una planificación HEFT:

1. Para cada tarea debe recorrer todo el workflow.
2. Una vez recorrido el workflow debe asignar una prioridad de longitud, que debe ser igual a la longitud de las tareas hasta el final.
3. Incluye en esta prioridad el tiempo de cómputo y comunicación de las tareas.
4. Realiza la asignación de las tareas.
5. Actualizar las máquinas disponibles en el entorno de ejecución.
6. Repetir los pasos 1, 2, 3, 4, y 5 hasta que no queden tareas por asignar del workflow lanzado.

En la siguiente tabla 3.5 se muestra los resultados obtenidos con la planificación HETF de nuestro ejemplo, obteniendo un tiempo total de ejecución con el algoritmo HETF fue de **552 segundos**, utilizando la máquina con mejor *benchmark* en la ejecución de casi todas las tareas.

Tareas/Máquina	Planificación (HEFT)
A/M2	178 seg
B1/M3	215 seg
B2/M2	59 seg
C/M2	100 seg
tiempo total	552 seg

Tabla 3.5: Planificación HEFT del workflow.

3.1.5. Algoritmo de Planificación Balanced Minimum Completion Time (BMCT)

El algoritmo Balanced Minimum Completion Time (BMCT) [77] es una heurística conocida como equilibrada, donde después de una asignación inicial

de las tareas a las máquinas que reduzcan al mínimo el tiempo de ejecución, hay una fase que trata de reducir al mínimo el tiempo total de intercambio de tareas entre las máquinas que ejecutan más rápido.

Esta heurística utiliza un conjunto de tareas independientes; un conjunto de máquinas y una matriz, donde se indicará el coste de ejecución de cada tarea en cada máquina. Cada tarea es planificada la primera vez con el objetivo de completar la ejecución de todas las tareas tan pronto como sea posible (es decir, minimizando el *makespan*).

El algoritmo se compone también de dos fases, una destinada a la asignación inicial de tareas en los equipos la ejecutarán, y otra segunda fase llamada de optimización, donde las tareas seleccionadas se mueven entre las máquinas con el fin de minimizar el *makespan* general de la aplicación.

1. Para cada tarea debe recorrer todo el workflow.

2. Una vez recorrido el workflow debe asignar una prioridad del camino crítico.

3. Las tareas con esta prioridad serán enviadas a los recursos de cómputo con mejor rendimiento.

4. Se realiza la asignación de las tareas.

5. Actualizar las máquinas disponibles en el entorno de ejecución.

6. Repetir los pasos 1, 2, 3, 4, y 5 hasta que no queden tareas por asignar del workflow lanzado.

En la siguiente tabla 3.6 se muestra los resultados obtenidos con la planificación BMCT de nuestro ejemplo, obteniendo un tiempo total de ejecución con este algoritmo BMTC fue de **535 segundos**, utilizando la máquina con mejor *benchmark*, en la ejecución de casi todas las tareas.

Tareas/Máquina	Planificación (BMCT)
A/M2	148 seg
B1/M3	198 seg
B2/M2	60 seg
C/M2	99 seg
tiempo total	535 seg

Tabla 3.6: Planificación BMCT del workflow.

3.1.6. Algoritmo de Planificación Sufferage

El algoritmo de planificación Sufferage [59] se deben seleccionar los dos valores mínimos de tiempo estimado de finalización (TEC), que se encuentran para cada tarea al inicio de la planificación. La diferencia entre estos dos valores se define como el *valor Sufferage*. Posteriormente este algoritmo selecciona el componente que tiene el máximo valor de Sufferage para enviarlo al recurso de cómputo con mejores prestaciones. Esto lo repite para cada tarea que vaya siendo planificada.

La intuición detrás de esta heurística es que los trabajos se les otorgan una prioridad relativa. El trabajo que tiene un valor Sufferage alto sugiere que no debe ser asignado al recurso para el cual hay mínimos TEC, porque puede tener un efecto adverso en el *makespan* debido a que el siguiente mejor valor de la TEC estará lejos del mínimo. Los pasos que realiza esta política para la planificación son los siguientes:

1. Para cada tarea debe obtener el primer menor tiempo y el segundo menor tiempo.
2. La diferencia entre ellos es el valor sufferage asignado.
3. Realiza la asignación de las tareas.
4. Actualizar las máquinas disponibles en el sistema distribuido.
5. Repetir los pasos 1, 2, 3, y 4 hasta que no queden tareas por asignar del workflow lanzado.

En la siguiente tabla 3.7 se muestra los resultados obtenidos con la planificación Sufferage de nuestro ejemplo, obteniendo un tiempo total de ejecución con el algoritmo Sufferage fue de **568 segundos**, utilizando la máquina con mejor *benchmark* en la ejecución de casi todas las tareas.

Tareas/Máquina	Planificación (Sufferage)
A/M2	178 seg
B1/M1	137 seg
B2/M2	59 seg
C/M2	100 seg
tiempo total	474 seg

Tabla 3.7: Planificación Sufferage del workflow.

Este capítulo hemos tratado las diferentes políticas de planificación que se disponen en la bibliografía. Estas políticas son las que permiten ejecutar de

manera más eficiente una aplicación workflow, pero claro teniendo en cuenta que existen muchas variables a controlar los resultados pueden variar si el entorno de ejecución cambia. Estos conceptos son claves para poder mejorar el *makespan* de una aplicación workflow, y serán utilizadas a lo largo de esta investigación. Finalmente vemos algún ejemplo sencillo de como tratan un workflow estas políticas de planificación.

Capítulo 4

Arquitectura y componentes de "SchedFlow"

En este capítulo se describe el diseño y la implementación de SchedFlow, donde se muestran los principales componentes de la arquitectura, y cada una de las funciones que tiene la API. También hacemos una descripción detallada de cómo interactúan los módulos entre sí, para gestionar la integración de las políticas de planificación. Para finalizar se muestra un ejemplo sencillo de la integración de una política de planificación, en la que destacamos la flexibilidad, simplicidad y transparencia de trabajo que tiene nuestro *framework*.

SchedFlow es un *framework* desarrollado como una capa transparente de software, con el objetivo de poder integrar políticas de planificación en los gestores de workflow. Esta integración permitirá la ejecución de las aplicaciones workflow utilizando diferentes políticas de planificación, todo de forma muy sencilla y con las ventajas de mejorar el *makespan* final de la aplicación.

Una vez mencionada la definición de nuestro *framework* salta una pregunta muy importante ¿Por qué utilizar SchedFlow?. Analizando la respuesta desde el punto de vista de la planificación y gestión SchedFlow ofrece algunas ventajas interesantes para los usuarios que lo utilicen. En primer lugar porque podemos integrar cualquier política de planificación, de manera sencilla y transparente al usuario. Además con un framework como SchedFlow, ya se obtienen cómo mínimo el rendimiento ofrecido por la política de planificación integrada, que siempre será mayor a no tener ninguna política para planificar la ejecución de un workflow.

También tenemos la posibilidad de realizar la replanificación de las tareas, gracias al gestor de eventos (que será detallado más adelante en este capítulo)

integrado en nuestro *framework*. Este gestor juega un papel importante en la ejecución de un workflow, y suele estar ausente en muchos de los gestores de workflow que existen en la bibliografía. Podemos decir que SchedFlow amplia las posibilidades ofrecidas por otras plataformas. En las siguientes secciones de este capítulo se tratarán los detalles tanto de la arquitectura como del funcionamiento de SchedFlow.

4.1. Arquitectura de SchedFlow

SchedFlow es un *framework* desarrollado como una capa transparente de software, con el objetivo de poder integrar políticas de planificación en los gestores de workflow. Esta integración permitirá la ejecución de las aplicaciones workflow utilizando diferentes políticas de planificación, todo de forma muy sencilla, flexible y transparente buscando mejorar el *makespan* final de la aplicación.

Esta arquitectura ha sido construida por capas para permitir la mayor flexibilidad al momento de utilizarla, para ello se han desarrollado una *API* que contiene las funciones orientadas al usuario, y luego tenemos una serie de adaptors que están orientados al gestor y que serán definidos más adelante en este capítulo.

También se dispone de *logs* informativos que permiten a los módulos de SchedFlow, obtener los datos de lo que está ocurriendo con los diferentes estados de las tareas (estados que fueron definidos en la sección 2.3 de esta memoria). Ahora veremos las definiciones de los módulos que están presentes en la arquitectura, cómo interactúan entre sí, y los mecanismos de control que hemos incorporado a nuestro *framework*.

4.1.1. Descripción del DAG

Dag description es un parámetro de entrada que SchedFlow necesita para iniciar el proceso de planificación, en el se encuentran definidas todas y cada una de las tareas del workflow con sus respectivas dependencias, esta descripción contiene la estructura de la aplicación workflow, y el correcto orden en que han de ejecutarse las tareas en los diferentes niveles de la aplicación.

Sin este fichero es imposible poner en marcha nuestro a SchedFlow, ya que es ahí donde esta definida la estructura propia de la aplicación, número de tareas, funciones asociadas a cada tarea, tiempos de cómputo, tiempos de comunicación,

y el ancho de banda del entorno de ejecución. Los detalles más técnicos de esta descripción están definidos más adelante en la sección 4.2 de este capítulo.

4.1.2. Módulo Controller

El módulo **Controller** es el encargado de poner en marcha las tareas en SchedFlow y su modo de funcionamiento es el siguiente: una vez recibida la planificación de las tareas que previamente ha realizado el módulo **Scheduler.** El **Controller** da inicio a la operación de envío de estas tareas al gestor de workflow, para que el gestor de workflow las comience a enviar a los recursos a los cuales que han sido planificadas.

Previó al envió, las tareas de la aplicación workflow deben ser adaptadas al gestor de workflow que se esté utilizando. Esta adaptación consiste en ajustar el formato descriptivo enviado por el usuario, al utilizado por el gestor de workflow; esto ocurre de forma automática y transparente al usuario, donde un proceso interno realizado por SchedFlow detecta cuál es el gestor disponible en el entorno de ejecución.

4.1.3. Módulo Observer

El módulo **Observer** es uno de los principales componentes de SchedFlow, ya que es el encargado de obtener los recursos de cómputo disponibles en el entorno de ejecución, información que debe proporcionar posteriormente al módulo **Scheduler**. Este módulo **Observer** también se encarga monitorizar los estados que van pasando tareas del workflow mientras se están ejecutando.

El funcionamiento del módulo es el siguiente: una vez que el usuario ha enviado el workflow y el módulo **Scheduler** lo ha planificado, entonces el módulo **Controller** comienza a enviar las tareas a la cola de ejecución. Por su parte el módulo **Observer** inicia el proceso de monitorización de la(s) tarea(s) que se han enviado. Toda esta actividad la obtiene del el *log de información* generado por el gestor de workflow, así se mantiene un estricto seguimiento de lo que está ocurriendo con cada tarea enviada al recurso de cómputo para su ejecución.

Si el módulo **Observer** detecta que una tarea ha finalizado correctamente, entonces informa al **Scheduler** para que este libere la(s) siguiente(s) tarea(s) que se encontraban a la espera de ser liberadas de su dependencia por parte de la tarea padre. Este procedimiento se realiza cada vez que una tarea pasa a estado finalizada.

Adicionalmente el módulo **Observer** cumple otra función la de monitorizar las tareas en ejecución. Esta monitorización sirve para detectar e informar al módulo **Scheduler**, cuando una tarea ha sufrido algún evento inesperado (suspensión, fallo) durante la ejecución. Así el módulo Scheduler puede realizar las acciones correctivas necesarias para que la ejecución se complete exitosamente, sin tener que iniciar nuevamente toda la ejecución.

4.1.4. Módulo Scheduler

El módulo **Scheduler** como su nombre lo indica, es el encargado de realizar la planificación de las tareas que vienen definidas en el fichero descriptivo con la aplicación workflow. La planificación de estas tareas la realiza utilizando las funciones de la *API*, que el usuario debe incluir en la política de planificación que desea integrar.

Cuando todo está correctamente definido en la política del usuario, entonces el módulo **Scheduler** haciendo uso de la *API*, empieza a realizar el mapping entre las tareas definidas en el fichero de descripción, y los recursos disponibles del entorno de ejecución que el módulo Observer le ha informado. Una vez que el mapping se ha completado, el módulo **Scheduler** envía las tareas del primer nivel de la aplicación workflow al módulo **Controller** para que este las comience a enviar a la cola de planificación.

Los recursos utilizados por el módulo **Scheduler** para realizar la planificación, provienen de una lista generada por el módulo **Observer** utilizando el método *get_resource()* de nuestra *API*. Esto garantiza que el **Scheduler** realice el mapping de las tareas a los recursos disponibles, justo al momento de iniciar la planificación.

El módulo **Scheduler** por otro lado se queda escuchando los cambios de estado de las tareas proporcionados por el módulo **Observer,** para así continuar enviando las tareas que ya han satisfecho sus dependencias al módulo **Controller**.

Si el módulo **Observer** notifica al módulo **Scheduler** sobre algún fallo de una tarea, el proceso de mapping se vuelve a realizar completamente para las tareas pendientes en el caso estático, y para la(s) tarea(s) dependiente de esta tarea y que estaban próximas a ser enviadas al módulo **Controller** para que este las enviará al recurso para ser ejecutadas.

4.1.5. Módulos Adaptors

Los módulos Adaptors (adaptadores) ofrecen una interfaz que permite la conexión de los diferentes gestores de workflow, aprovechando de esta manera los servicios ofrecidos por los gestores de workflow. Estos servicios pueden ser lanzamiento de una tarea para su ejecución, obtención de recursos, información sobre los estados de ejecución de las tareas por mencionar algunos.

SchedFlow incluye tres *Adaptors* que nos permiten trabajar con tres gestores de workflow populares entre la comunidad científica, a saber DAGMan [56], Taverna [72], y Karajan [96]. Estos adaptors fueron diseñados con la finalidad de permitir el enlace entre SchedFlow y los gestores de workflow de fácil, flexible y transparente al usuario.

La figura 4.1 muestra los tres principales servicios que los adaptors utilizan, para obtener información de los gestores que luego es utilizada por SchedFlow. Los servicios que han sido implementados para los *Adaptors* son los siguientes:

Task adaptor: Este servicio interactúa con el módulo **Controller**, el cual adapta el envío de las tareas asignadas a la correspondiente descripción utilizada por el gestor de workflow que se está utilizando. Las funciones de los gestores utilizadas para tal fin son las siguientes:

- En el caso de Cóndor DAGMan, el comando que se utiliza es *condor_submit_dag* .
- En el caso de Triana, el comando utilizado es *run_workflow* .
- En el caso de Taverna, el comando utilizado es *job_submission*.

Resource adaptor: Este servicio permite al módulo **Observer** obtener la lista de recursos, que se encuentran disponibles en el entorno de ejecución utilizando para ello las siguientes funciones:

- En el caso de Cóndor DAGMan, el comando que se utiliza es *condor_status*.
- En el caso de Triana, el directorio a utilizar es *TrianaV3Resources*.
- En el caso de Taverna, el comando que se utiliza es *g_resource*.

Event adaptor: Este servicio es necesario para que los módulos **Observer** y **Scheduler** puedan replanificar las tareas cuando ocurre un evento inesperado, produciendo la suspensión o eliminación de la tarea del recurso donde se estaba ejecutando. Esta información es obtenida utilizando las siguientes funciones:

- En el caso de DAGMan, los eventos se obtienen de la información del fichero de *log* creado al momento de iniciar la ejecución.
- En el caso de Triana, utilizamos la herramienta de *show_properties* de la tarea que proporciona su estado en tiempo de ejecución.
- En el caso de Taverna, el contenido del *log_book_data* es utilizado para tal fin.

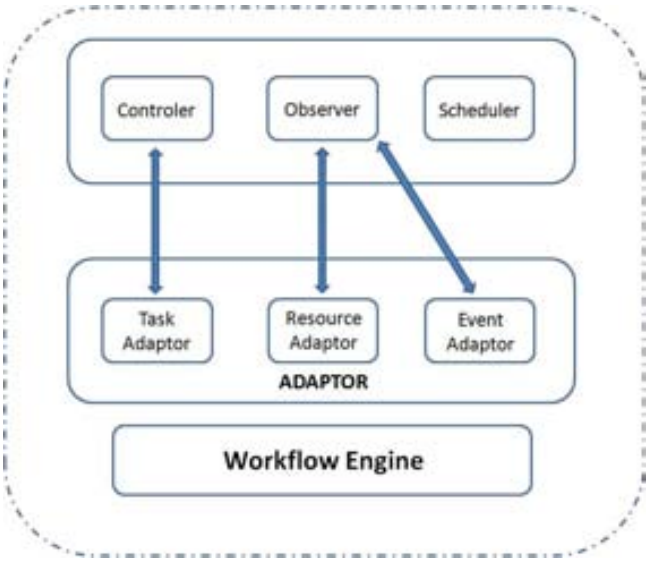

Figura 4.1: Capa de los módulos Adaptor.

4.1.6. Mecanismo de Replanificación de SchedFlow

En esta sección, se describe el mecanismo de replanificación que se ha implementado en SchedFlow, donde parte del proceso principal de este mecanismo es gestionado por el módulo **Observer**.

Este sencillo pero potente mecanismo, nos permite reaccionar ante diferentes eventos que ocurren mientras las tareas de la aplicación workflow se están ejecutando. Para conseguirlo el módulo **Observer** hace uso de una función llamada *log_monitor()*, que sirve de apoyo a las políticas de planificación sin importar el tipo que sean (estática, dinámica o hibrida). La ventaja de utilizar este mecanismo es que podemos replanificar las tareas que sufren algún evento inesperado (suspensión o fallo).

El funcionamiento de este mecanismo es el siguiente: si una tarea es enviada al recurso de cómputo para su ejecución, y se registra en el *log* de información un

estado diferente a *"Ok"*, entonces el módulo **Observer** lo detecta de inmediato y a su vez le notifica al módulo **Scheduler** para que este último realice la acción correctiva. Para entender mejor este mecanismo ilustramos su funcionamiento con un ejemplo:

Supongamos que se ha enviado una aplicación workflow utilizando SchedFlow, entonces suelen ocurrir una serie de procesos hasta que se tienen planificadas todas las tareas de la aplicación, y todos los módulos son iniciados. Entonces si una tarea se encuentra está lista para ser ejecutada SchedFlow asigna la tarea a un recurso de cómputo a través del módulo **Scheduler**, dejando esta tarea en una lista de tareas planificadas (*mapped_list*).

Luego, utilizando el módulo **Controller**, la tarea es adaptada y enviada al gestor de workflow, quien finalmente la pondrá en marcha en el recurso asignado por el módulo **Scheduler**. Una vez que la tarea completa estos pasos y la misma se ejecuta satisfactoriamente pasa a otra lista de trabajos completados (*done_list*) esto es lo que ocurre en el caso ideal. ¿Pero qué sucede si tenemos un evento que impide la finalización exitosa de la tarea?, ahí es donde entra en escena el mecanismo de replanificación, que da la posibilidad de replanificar nuevamente la(s) tarea(s) que han fallado y actualizar el mapping restante eliminando el recurso que ha dado el fallo.

ShedFlow gestiona estos eventos inesperados de forma dinámica, básicamente este mecanismo hacen un uso intensivo de los *logs* de información, los cuales como habíamos mencionado anteriormente están siento monitorizados por el módulo **Observer**, quién es capaz de detectar dos clases de eventos; estos son tareas suspendidas y superación de tiempo de ejecución planificado de una tarea. A continuación hacemos la definición de estos tipos de eventos, que podemos gestionar actualmente en ScheFlow:

- **Tareas Suspendida**: Este evento se detecta cuando una tarea enviada a un recurso informático puede ser interrumpida, o suspendida por los procesos locales se ejecutan en esa máquina, por lo que la tarea se suspende por un período de tiempo aleatorio. La consecuencia es un retraso en la finalización de la misma. Para apoyar este evento, SchedFlow utiliza el módulo de **Observer**, así que cuando se produce este evento, la tarea podría ser replanificada a un nuevo recurso.
- **Error de tiempo en planificación de tareas**: Este evento se detecta cuando una tarea que fue enviado a un recurso de cómputo está tardando más de un factor definido por el usuario, del tiempo que ha sido planificado inicialmente por el **Scheduler**. Entonces esto marca un tiempo máximo de espera para que nuestro sistema reaccione realizando una replanificación

de la tarea, eliminando de la lista de recursos este que ha sino ineficiente para ejecutar esta tarea.

Cabe señalar que esta opción de seguimiento de tareas se puede configurar por el usuario, pero eso significaría que el usuario debe ser consciente de la si la tarea produce un error solamente le queda la opción de volver a enviarla manualmente.

Dado que los workflow pueden tener muchas tareas, la manipulación manual de error puede ser difícil. Aunque cada motor de workflow tiene su propia función para esto, las mismas son muy básicas, y no solucionan ningún problema que ayude a mejorar o mantener el tiempo planificado para la ejecución del workflow, incidiendo en el *makespan* de la aplicación.

La figura 4.2 representa de forma gráfica todos los componentes y módulos que constituyen la arquitectura de nuestro *framework* **SchedFlow**.

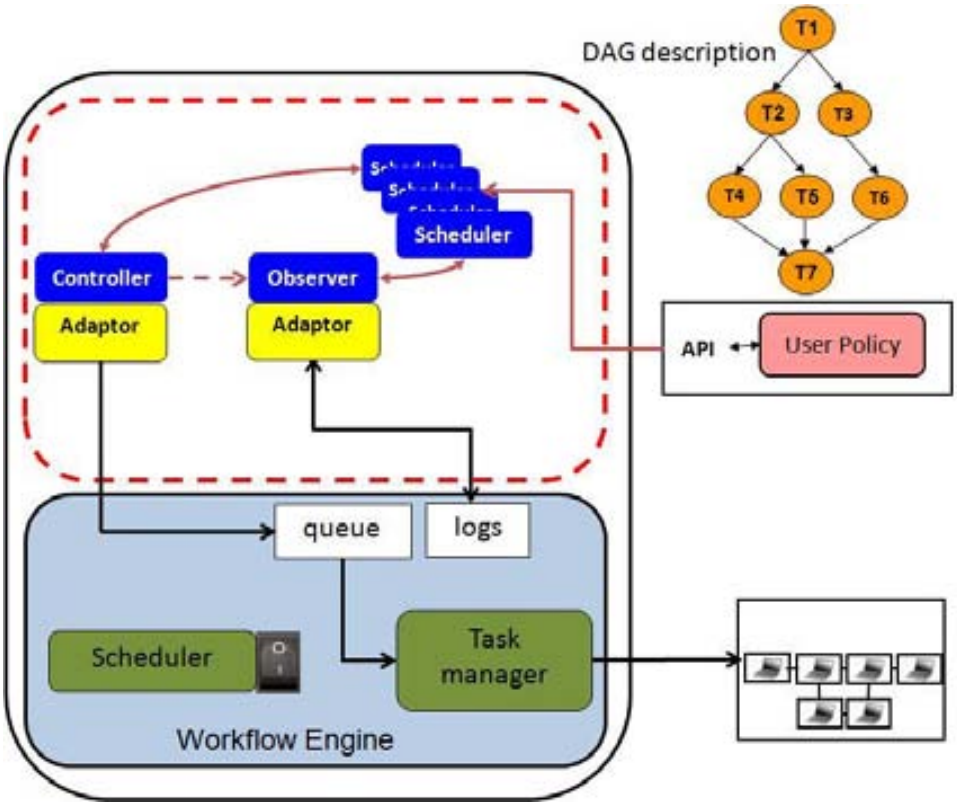

Figura 4.2: Arquitectura de SchedFlow.

4.2. Descripción de Trabajos para SchedFlow

A la hora de especificar las características y los requisitos mínimos de los trabajos por parte del usuario, se necesita un lenguaje de descripción que cumpla con los siguientes requisitos:

- Debe ser concreto y fácil de utilizar por el usuario final, de forma que tanto la escritura manual, cómo la generación a partir de herramientas gráficas sean sencillas.
- Que permita expresar las necesidades concretas de las aplicaciones workflow, que son en las que hemos seleccionado para nuestra investigación, con un modelo de datos semi-estructurado en el que sea necesario un esquema específico.

La descripción de los trabajos que se enviaran a SchedFlow, se encuentran definidas mediante las *ClassAds* de Condor [89], pero esto no significa que exista un vinculo directo con este gestor, solo tomamos como referencia la forma descriptiva para completar los módulos de SchedFlow. Así que cuando se envía algún trabajo a nuestro *framework* se deben satisfacer los requisitos antes mencionados.

Debemos recordar al lector que un *ClassAd* [89] por definición son, un conjunto de expresiones con nombre llamados atributos, donde una expresión contiene literales y referencias a otros atributos, y suelen estar compuestas de operadores. En la especificación de trabajos para Condor se hace necesario definir algunos de los nombres de los atributos con un significado especial, a continuación se muestran las definiciones de estos atributos en un trabajo Condor:

- **Executable:** define el nombre del ejecutable que será enviado y ejecutado finalmente.
- **Arguments:** una cadena que contiene los argumentos en línea de comandos pasados al ejecutable.
- **Universe:** especificación del tipo de universo donde se ejecutará el workflow.
- **StdInput:** fichero de entrada estándar.
- **StdErr:** fichero de error estándar.
- **Log-File:** ficheros donde se almacenan todos los eventos que ocurren durante la ejecución del workflow.

- **Requirements:** Expresión representando un conjunto de requerimientos de la aplicación.
- **Queue:** Cola donde se irán colocando los trabajos a espera de recursos disponibles en el entorno de ejecución.

A continuación se muestra un ejemplo de la definición de una tarea para poder ser ejecutado en DAGMan, el cual debe tener las siguientes características:

- El nombre del ejecutable es **test.sh**. Los ficheros /usr/local/gustavo/test.sh (el ejecutable) que es residente en la máquina desde donde se envía el trabajo y serán transferidos a la máquina donde se ejecute finalmente la aplicación.
- La salida y error estándar son **node_A.out y node_A.err** respectivamente. Estos ficheros se transferirán de forma automática a las máquinas desde donde ha sido lanzada la tarea una vez que finalice la ejecución de test.sh.

```
Executable = test.sh;
Arguments = node A;
StdOutput = node_A.out;
StdError = node_A.err;
log = dag.log
Requirements = other.GlueHostArchitecture == INTEL;
```

El ejemplo anterior estamos indicando que la aplicación requiere, recursos de cómputo que contengan una arquitectura Intel para poder ser ejecutada. Estas son las definiciones que cualquier usuario debe hacer para todas las tareas del workflow.

4.2.1. Especificación del Workflow usando SchedFlow

SchedFlow tiene una estructura de especificación muy similar a la del gestor de workflow DAGMan, esto motivado a la sencillez que tiene este formato como se pudo ver en la sección 4.2. La especificación de un trabajo para SchedFlow la haremos con un sencillo ejemplo, utilizando un grafo acíclico dirigido (DAG), que tiene cuatro tareas y tres niveles de ejecución donde cada tarea ejecuta un pequeño *script* llamado *Sflow.sh*.

Recordemos que un DAG es un grafo con aristas unidireccionales en el que no existen bucles, cada una de las tareas (vértices) del grafo representa a

los trabajos que componen la aplicación y los arcos (aristas) identifican las dependencias entre ellos.

La ejecución de este DAG consistiría en tres pasos consecutivos:

1. Ejecución del nodo Nodo_A del primer nivel.
2. Ejecución paralela de las dos tareas (Nodo_B, Nodo_C) del segundo nivel. La ejecución se iniciará si y sólo si la ejecución del nodo del primer nivel (del cual dependen estas tareas) tuvo éxito.
3. Ejecución del último nodo (Nodo_D) del tercer nivel. La ejecución empezará si y sólo si, todas las tareas del segundo nivel de las cuales depende este nodo han finalizado con éxito.

Los flujos de tareas, de forma similar a los trabajos normales, se especifican con ficheros de texto siguiendo como ya dijimos antes el formato Cóndor-DAGMan convenientemente para soportar la gestión y estructura de DAGs.

```
Executable = Sflow.sh;
Arguments = node_A;
StdOutput = node_A.out;
StdError = node_A.err;
log = dag.log GM = DAGMan (para conocer el gestor utilizado);
Transfer_input_files= /home/gustavo/bin/prueba.jpg;
Should_transfer_files = YES;
When_to_transfer_output = ON_EXIT;
Requirements = other.GlueHostArchitecture == INTEL;
queue;
```

Esta es la forma en que un usuario debe especificar una tarea para SchedFlow, y conseguir así que nuestro *framework* pueda realizar la ejecución de la misma.

4.2.2. Integración de Políticas de Planificación en SchedFlow

En esta sección procederemos a describir, el funcionamiento de nuestra API para la planificación de un workflow, detallando las funciones con las cuales un usuario cuenta para ejecutar aplicaciones tipo workflow, haciendo uso de nuestra herramienta de planificación. La *API* consta de tres clases principales implementadas en *C++*.

Las clases son las siguientes **Schtask, Schmachine y Schplanning**, las cuales cuentan a su vez de unos métodos que permiten al usuario de nuestro sistema, realizar planificación de workflow en un entorno no dedicado. Siguiendo el orden de ideas planteado en esta sección, procedemos a describir las funciones ó módulos de cada una de las clases:

- **Schtask**, es una de las clases desarrolladas en nuestro *framework*, la cual controla todo lo relacionado con la descripción de las tareas del workflow y las dependencias que poseen las tareas entre si. Estas funciones son las siguientes:

1. *set_workflow* (nodename, node.submit, t_computing), esta función recibe como entradas el nombre del nodo, el archivo o tarea a ejecutar (node.submit), que no es más que el fichero de envió que necesita el gestor para ejecutar los trabajos del nodo y así enviarlo a la cola donde esperar hasta que pueda ser asignado a un recurso. Por último tenemos el parámetro del tiempo de cómputo del nodo, esto debe conocerse de antemano en caso contrario se posee un método que realiza este cálculo de forma estimada.

2. *task_dependence* (nodename1, nodename2, t_communication), esta función recibe como parámetros de entrada los pares de tareas que se relacionan en el workflow, así mismo como el tiempo de comunicación entre ellos, este valor si se desconoce existe un método que también lo calcula, y regresa las dependencias entre cada par de tareas del workflow con su respectivo tiempo de cómputo, estos valores se incluyen en el fichero de entrada para el SchedFlow.

- **Schmachine**, es otra de las clases del planificador, encargada del manejo de los recursos con los que contaremos a lo largo de la planificación, así como la notificación de los eventos que ocurren a lo largo de la ejecución del workflow, esta clase cuenta con las siguientes funciones:

1. *get_resource_list* (machine_list), esta función recibe como parámetro de entrada, una lista de recursos total de los que se dispone en nuestro entorno de ejecución, y retorna los recursos disponibles con sus *benchmark* en el instante que sea ejecutada. El valor que retorna es un lista simple con dos campos.

2. *get_communication_time* (schtask1, schtask2, bandwidth, latency), esta función recibe como parámetros de entrada las tareas a planificar, el ancho

de banda estimado disponible, y el valor de latencia estimado del entorno de ejecución. La misma retorna la media de comunicación entre las tareas del workflow, esto en el caso de que el usuario desconozca toda la jerarquía de comunicación entre las tareas de la aplicación. El valor es un número entero.

3. *get_ computing_ time* (schtask, task_time), esta función tiene una característica similar a la de *get_ communication_ time*, recibe como parámetros de entrada la tarea asociada del workflow, el tiempo estimado de ejecución de la tarea, y retorna el tiempo de ejecución para cada una de las tareas, así se puede obtener un valor estimado de ejecución para cada tarea del workflow. El valor es un número entero.

- **Schplanning**, es la última clase con la que cuenta nuestra API, encargada de administrar todo lo relacionado con la asignación de tareas en las máquinas disponibles de nuestro entorno de ejecución, es decir permite a través de la función de planificación organizar todas las tareas y la forma en que estas se ejecutaran, cuenta con las siguientes funciones:

1. *map* (schtask, schmachine, priority), esta es la función más importante de nuestro planificador, la misma recibe como entrada la tarea a planificar, el recurso planificado y la prioridad de ejecución de esta tarea. Esto permite obtener no solo planificación tarea / recurso, sino también como desea el usuario que sea realice la ejecución del workflow. Esta función retorna la acción de planificación de acuerdo a la política de planificación utilizada.

2. *unmap* (schtask), esta función es utilizada para eliminar una tarea que se encuentra en ejecución. La misma recibe como parámetro de entrada la tarea planificada, para su eliminación del recurso donde fue asignada. Esto puede ser por decisión del usuario para mejorar el entorno de ejecución. La función retorna un valor nulo si ha sido exitosa la eliminación de la tarea.

4.2.2.1. Ejemplo de integración de políticas en el SchedFlow

Todas estas funciones antes descritas anteriormente son las que permiten a un usuario, integrar una nueva políticas de planificación a nuestro sistema, completando entre todas una interface, sencilla, flexible y transparente para ser llamadas como instancias desde el SchedFlow. Ahora pasamos a ilustrar en pseudo-código lo que se debe hacer un usuario, cuando decida integrar una nueva política de planificación a un gestor de workflow utilizando SchedFlow.

Partamos del supuesto que un usuario decide utilizar SchedFlow, para llevar a cabo la ejecución de una aplicación workflow, utilizando para ello una política de planificación nueva, que supone una notable mejora en el *makespan* de la aplicación. Para completar esa integración de SchedFlow este usuario debe seguir estos sencillos pasos que detallamos a continuación:

- Realizar la descripción del fichero de entrada en el cual se especifican tareas y dependencias.
- Definir cada uno de los ficheros de envío que las tareas poseen en el workflow.
- Indicar los tiempos de cómputo y comunicación entre las tareas.
- Indicar la política de planificación con la que desea ejecutar el workflow, donde tiene dos opciones:

1. Ejecutar con las políticas existentes en SchedFlow.
2. Integrar una política externa definida por el usuario.
3. Ejecutar el workflow utilizando nuestro *framework*.

El esquema que mostraremos en la figura 4.3 será para el caso de que se integre una política nueva.

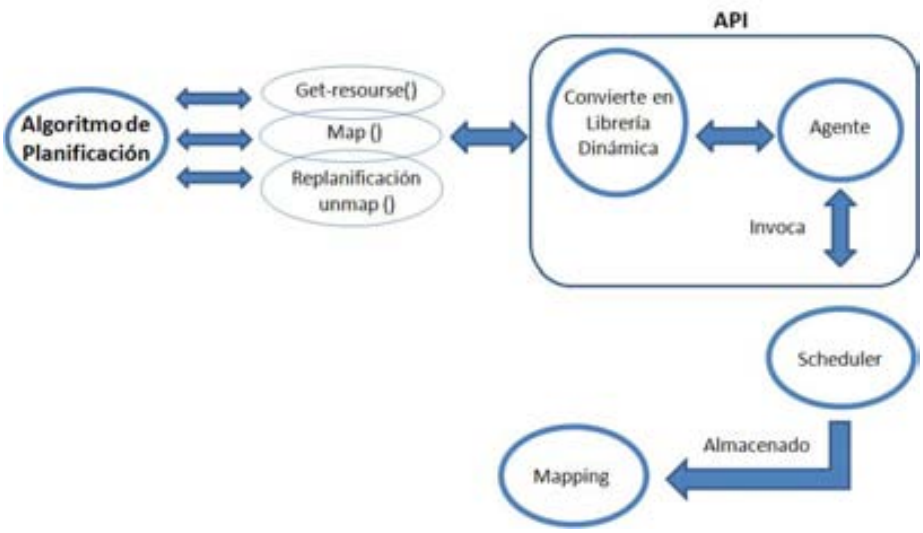

Figura 4.3: Esquema de integración de una política de planificación en SchedFlow.

Como se ve en la figura 4.3, ahora queda por mencionar como un usuario podría agregar una política nueva que no esté incluida en nuestro *framework*. Para ello deberá hacer uso de tres de los métodos descritos anteriormente, ya que siempre en la políticas de planificación hay puntos donde se debe solicitar los recursos, realizar el mapeo de las tareas, y como adicional tenemos que añadir el método *unmap(),* esto solo si el usuario desea que nuestro mecanismo de re-planificación sea utilizado.

Una vez que el usuario tenga implementada las funciones de la *API* en su algoritmo de planificación, entonces se invoca desde SchedFlow la función *creaScheduler(),* que ha sido desarrollada para poder realizar la acción de convertir la política del usuario en una librería dinámica. Esta librería a su vez es llamada por el agente de nuestra *API* para quedarse escuchando la petición del módulo **Scheculer** cuando sea necesario realizar el mapping, utilizando la política de planificación que el usuario este integrado en SchedFlow.

El método programado para *creaScheduler()* es el que se muestra a continuación:

```
creaScheduler(){
return new RoundRobinScheduler;
}
 void *hndl = dlopen("libnewshapes.so", RTLD_NOW);
 if(hndl == NULL){
 cerr << dlerror() << endl;
 exit(-1);
}
void *creaScheduler = dlsym(hndl, creaScheduler);
Scheduller *aScheduller **= static_cast;
Static_cast= <Scheduller()>(creaScheduler)();
```

Esto se guarda en un agente que es el encargado de recibir la petición del módulo **Scheduler**, cada vez que se pretende realizar un mapeo con alguna de las políticas existentes, o si es una nueva entonces es asignada de forma automática a un procedimiento llamado *new_police().* Como se puede apreciar no es complejo el procedimiento sin embargo los resultados son muy favorables al momento de ejecutar una aplicación workflow.

A lo largo de este capítulo hemos visto la arquitectura y principales componentes de software, que se implementaron en SchedFlow. Se describen cada uno de los módulos con su respectiva funcionalidad, el mecanismo de replanificación que es un punto clave en nuestra investigación.

También tenemos las funciones implementadas para poder gestionar la planificación y enlace con los gestores de workflow. Para finalizar mostramos un ejemplo de cómo un usuario puede realizar la integración de una política de planificación utilizando SchedFlow.

Capítulo 5

Experimentación y Resultados

En este capítulo se detallan los experimentos realizados para mostrar la funcionalidad del SchedFlow, y donde queda reflejado las ventajas de utilizar un *framework* flexible, transparente y sencillo de utilizar. Las aplicaciones workflow reales en las que nos hemos apoyado para la experimentación han sido Montage **[27]** y LIGO [9, 26].

5.1. Aplicación Montage

Montage [68] es una aplicación de cómputo intensivo, y que tiene como finalidad el refinamiento para el servicio de imágenes astronómicas portables. Esta aplicación realiza un cómputo intensivo para el procesamiento de las imágenes espaciales, todo esto para apoyar a los científicos del observatorio nacional virtual (NVO).

Aunque Montage está desarrollado al servicio de cálculo y de procesamiento intensivo de datos, para la astronomía y la comunidad científica que realiza investigaciones sobre imágenes astronómicas. También resulta útil para los que trabajamos aplicaciones workflow ya que la misma tiene un comportamiento de flujo de tareas que se van completando hasta construir un mosaico que luego es analizado por los expertos del área.

El proyecto Montage trata de dar soluciones al problema que atraviesa la tierra y la ciencia del espacio, en relación a cómo tener acceso eficiente a multi-terabyte de procesos, datasets distribuidos entre otras cosas más.

Por lo tanto resulta muy interesante utilizar esta aplicación en nuestra investigación, ya que hay un uso intensivo de los recursos de cómputo. En esta memoria describiremos de manera general la aplicación, para dar una idea al lector de los experimentos realizados. Dada la complejidad de los módulos que comprenden la aplicación, fue un buen punto de partida para toda nuestra investigación y sobre todo para poder medir la eficiencia de los elementos que tenemos en la arquitectura de SchedFlow.

Montage tiene la meta de proveer a los astrónomos el software para el cómputo de las imágenes astronómicas en mosaicos, cuyo formato venga dado por el estándar que utilizan los expertos como son las imágenes tipo FITS (Flexible Image Transport System). Los mosaicos de entrada suelen ser trozos de imágenes reales tomadas por los telescopios, y luego son procesadas para preservar la fidelidad espacial astrométrica de las mismas.

Montage construye un mosaico de la imagen en cuatro etapas:

1. Re-proyección de las imágenes de entrada a una escala espacial común, sistema coordenado, y proyección de WCS,

2. El modelo de la radiación de fondo en las imágenes alcanzan escalas de flujo, y niveles comunes de fondo reduciendo al mínimo las diferencias entre las imágenes,

3. Rectificación de imágenes a una escala común del flujo y nivel del fondo,

4. Co-adición de imágenes re-proyectadas como un fondo astrométrico mapeadas hasta obtener un mosaico final. Montage logra estos pasos en módulos independientes, escrito en ANSI C para la portabilidad. Este acercamiento de las imágenes ayuda y proporciona flexibilidad considerable a los usuarios para el posterior estudio de las mismas.

Por ejemplo, utilizar el Montage como sistema de re-proyección simple de imágenes espaciales, pueden manejar el fondo de ejecución del algoritmo sin impacto en los otros pasos siguientes, o que define el proceso específico que atraviesa los datos de entrada.

La estructura de la aplicación Montage que se muestra en la figura 5.1 consta de 9 niveles de profundidad, donde cada módulo aplica diferentes cambios a la imágenes cargada con en los parámetros de entrada del Montage. Cada módulo depura y refina los trozos de imagen para obtener una imagen que los investigadores puedan analizar.

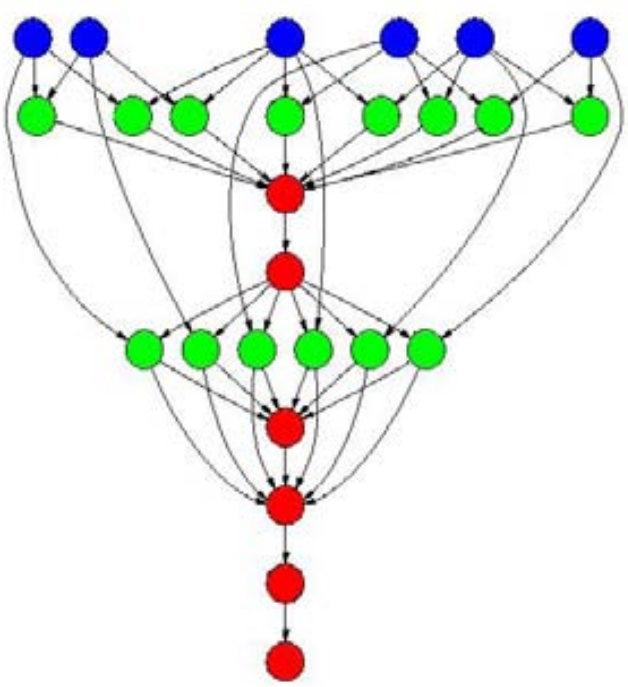

Figura 5.1: Descripción del workflow Montage.

5.2. Aplicación LIGO

El Laser Interferometer Gravitational Wave Observatory (**LIGO**) [9] es un ambicioso proyecto, que permite a los investigadores detectar las ondas gravitacionales producidas por eventos violentos en el universo, como por ejemplo la colisión de dos agujeros negros o la explosión de supernovas [26]. Los registros experimentales de esta aplicación obtienen aproximadamente un terabyte de datos por día, que luego es analizada por los científicos en un proyecto colaborativo que se extiende por cuatro continentes.

Esta aplicación viene definida como un workflow de dimensiones muy grandes, pero que suele ser utilizado en pequeñas proporciones para experimentar la comunidad científica para realizar estudios de políticas de planificación. En este trabajo de investigación nosotros hemos seleccionado una porción de ese gigantesco workflow tan solo compuesto por 81 nodos, para poder realizar la evaluación de carga inicial de los datos de entrada, para ver el comportamiento del *makespan* final utilizando SchedFlow.

La representación gráfica del LIGO se puede ver en la siguiente figura 5.2[26], donde se puede apreciar la complejidad de esta aplicación pero que para efectos

de nuestro trabajo de investigación ha sido determinante en la evaluación de carga inicial de una aplicación workflow.

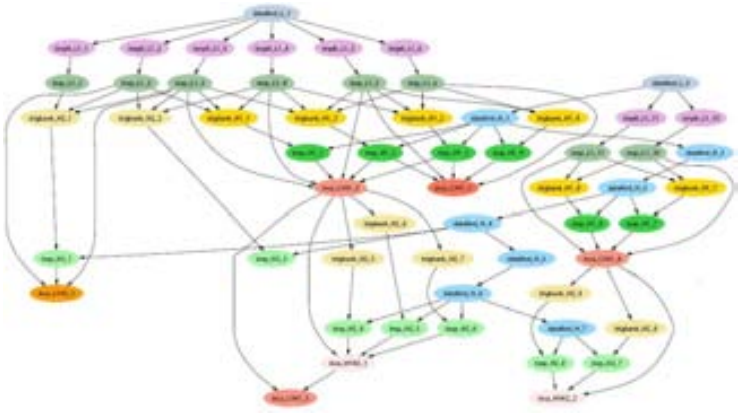

Figura 5.2: Descripción general de la aplicación LIGO.

Hemos seleccionado estas dos aplicaciones ya que las mismas son ampliamente utilizadas por la comunidad científica, pero existen algunas razones y que tanto Montage como LIGO, representan el tipo de aplicaciones que nuestro framework gestionaría de manera correcta, ya que son grafos acíclicos dirigidos, donde existen tareas con un intenso cálculo de cómputo, existen un buen volumen de comunicación y no tenemos loop sobre alguna tare de la aplicación workflow. Esto significa que aplicaciones con estas características pueden sacar el máximo provecho de nuestra plataforma, pudiendo ser ejecutadas en un entorno distribuido no dedicado, y extrapolable a sistemas Grid.

5.3. Entorno de Experimentación

En esta sección presentaremos el entorno de ejecución que hemos utilizado para realizar todos los experimentos de nuestra investigación. Nuestro entorno de ejecución seleccionado encaja perfectamente dentro de la definición de entorno distribuido no dedicado. Esto significa que todos los experimentos realizados deben enfrentar los mismos problemas existentes en estos entornos. Algunos de los problemas que podemos destacar son por ejemplo: disponibilidad de los recursos, latencia de red, ancho de banda y heterogeneidad de los recursos de cómputo.

Sin embargo los entornos no dedicados nos ofrecen la capacidad de cómputo necesaria para realizar la planificación de un workflow. Además este tipo de

entorno nos permitirá a ver lo que ocurre al ejecutar un workflow con SchedFlow en un entorno tan dinámico como el mencionado.

Por otro lado cabe destacar que este tipo de entornos, en lo que respecta a SchedFlow, no presenta diferencias significativas respecto a un entorno Grid. Sin embargo, un entorno no dedicado es más sencillo de administrar. También podemos decir que el haber utilizado SchedFlow un entorno no dedicado, nos da una visión más clara de lo que ocurrirá en un entorno de grandes dimensiones como el Grid. Salvo que para toda la gestión de configuración los usuarios deben tener los acuerdos correctos con los administradores de las organizaciones que participan en el Grid. Es importante destacar que en un entorno Grid habrá una política sobre el middleware que gestiona los recursos del mismo, y que en este middleware se tendrán que integrar los gestores de workflow con los que se desee trabajar.

En nuestro caso el entorno no dedicado donde se ejecutaron todas las pruebas está compuesto por *135 recursos de cómputo*, con una arquitectura *Intel*, y sistema operativo *Linux*. Siendo la media del *benchmark* obtenida para estos recursos de cómputo de *0,55 Gflops* aproximadamente.

La tabla 5.1, muestra las tres categorías de recursos que tenemos, la cual se ha elaborado utilizando los rangos *benchmark* en los que han caído. Esto además nos muestra que estamos en presencia de un entorno de ejecución heterogéneo y dinámico, en el cual cada recurso de cómputo tiene diferentes prestaciones y según cómo se haga la planificación los resultados de la ejecución serán mejores o peores.

La medida de rendimiento que hemos seleccionado para los recursos utilizados, está dada por la cantidad de operaciones de punto flotante por segundo que puede resolver (GFlops) [69].

Rango de GFlops (X)	Categoria del Recurso	% del Total de Recursos
X < 0,4	Recurso muy lento	6,66 %
0,4 <= X <= 0,6	Recurso estandar	15,55 %
X > 0,6	Recurso muy rápido	77,79 %

Tabla 5.1: Categorización de los recursos.

- Si la cantidad de operaciones de punto flotante por segundo (GFlops) es menor que 0,4. Este grupo de recursos están categorizados como muy lentos en el gráfico, y estas representan el 6,66 % del total de recursos con el que disponemos para ejecutar la aplicación.
- Si la cantidad de operaciones de punto flotante por segundo (GFlops) están en el rango mayor o igual a 0,4 pero menor o igual a 0,6 este se considera

como un grupo de recursos estándar, capaces de ejecutar las tareas de cómputo en tiempos relativamente aceptables, en el gráfico anterior esto representa el 77,79 % del total de recursos con el que disponemos en nuestro entorno de ejecución.

- Si la cantidad de operaciones de punto flotante por segundo (GFlops) es mayor que 0,6. Este grupo de recursos están categorizados como muy rápido en el gráfico esto representa el 15,55 % del total de recursos con el que disponemos para ejecutar la aplicación.

5.4. Experimentos Realizados

En esta sección describiremos todos los experimentos que hemos llevado a cabo en nuestra investigación, y que fueron necesarios para poder ver los alcances de SchedFlow.

El caso más interesante para la planificación de workflow son aquellos que están formados por numerosos trabajos, distribuidos en múltiples niveles a lo largo de su estructura, y que requieran recursos con altas prestaciones para obtener un mejor rendimiento en la ejecución de las aplicaciones.

Por esta razón procedemos a dar los detalles de la experimentación realizada, los tipos de algoritmos de planificación que se utilizaron, que gestores de workflow se enlazaron a SchedFlow, que métrica buscamos en los experimentos, y otras cuestiones menos importantes.

A la hora de realizar la planificación de una aplicación workflow, hay que tener en cuenta dos factores importantes como son los tiempos de ejecución de cada una de las tareas que conforman la aplicación, así como los tiempos de transferencia de ficheros existentes entre ellos.

Para solucionar este primer problema se realizaron más de 300 ejecuciones donde se midieron los tiempos de ejecución para cada tarea, utilizando una máquina piloto, y cuyos resultados nos permitió establecer una aproximación inicial del tiempo consumido por cada tarea. Estos tiempos fueron posteriormente utilizados como datos iniciales que alimentaban la planificación que se hizo con SchedFlow.

El siguiente problema que debíamos solventar era tener una estimación del ancho de banda existente en el entorno de ejecución, para solucionarlo realizamos más 3000 mediciones de transferencia de ficheros entre pares diferentes de los recursos del entorno de ejecución, y la media global de ello fue seleccionado como ancho de banda.

Una vez obtenidas las estimaciones de los parámetros importantes a la hora de planificar una aplicación workflow como son los tiempos de cómputo y comunicación, además del ancho de banda entonces nos planteamos realizar los experimentos en tres fases para demostrar la facilidad, flexibilidad, y transparencia de SchedFlow al momento de integrar una política de planificación en un gestor de workflow.

La primera fase de experimentación consistió en realizar la ejecución del Montage utilizando SchedFlow, donde integramos las políticas de planificación, aleatoria [58], Min-Min [71], y una variante del algoritmo aleatorio incluyendo replanificación, en este caso solo utilizamos Condor-DAGMan como gestor de workflow.

Los experimentos que hicimos en esta fase se llevaron a cabo en el entorno de ejecución descrito en la sección 5.3 de este capítulo, aquí nuestro interés principal era demostrar la flexibilidad ofrecida por SchedFlow, para ello integramos tres políticas de planificación (aleatoria [58], Min-Min [71]) en el gestor de workflow Condor-DAGMan [56].

Para esto se ejecutaron dos aplicaciones workflow con la política de planificación que tiene por defecto el gestor de workflow Condor-DAGMan. Luego realizamos la integración de las políticas de planificación antes mencionadas, en ambos casos se midió el *makespan* promedio global de la aplicación.

El *makespan* proviene de 120 ejecuciones para cada una de las políticas de planificación, donde el Montage utilizado era de 56 nodos, y cuya carga inicial era de 100MB. En el caso de LIGO se utilizo una porción de este workflow con 81 nodos y una carga inicial de 100MB. Estos resultados nos darían un primer feedback de la potencialidad ofrecida por SchedFlow.

Las políticas utilizadas no son de las más complejas y para estos experimento solo se tenía enlazado el gestor de workflow Condor-DAGMan. La importancia de este experimento es que gracias a los resultados obtenidos, se detecto la necesidad de incluir un mecanismo de replanificación a SchedFlow que posteriormente incluimos en la segunda fase de experimentación.

La segunda fase de experimentación dimos un paso adelante ya que incluimos políticas de planificación más sofisticadas, como la HEFT [90] y BMCT [77] manteniendo las políticas de planificación Aleatoria y Min-Min. También se incluye en estas pruebas un mecanismo de replanificación disponible que tenemos en SchedFlow, y que fue definido en la sección 4.1.6 de esta memoria.

Otra nueva variante que hemos incluido en esta experimentación es que se utilizaron dos nuevos gestores de workflow, estos han sido Taverna y Karajan,

repitiendo cada escenario planteado con los diferentes gestores workflow. Los detalles de cada escenario se muestran en la sección 5.5.2.

La tercera fase de experimentación ha llevado un enfoque hacia el efecto que puede causar la modificación de la carga inicial que pasamos como parámetro a las aplicaciones. Esto con la finalidad de ver como era el comportamiento de una aplicación workflow planificada con la misma política de planificación y gestor de workflow, cuando los tamaños de carga inicial se han variado.

Los experimentos ha sido llevados a cabo modificando el tamaño inicial de carga de 400MB a 1024 MB, hemos utilizado las mismas políticas de planificación, así como los gestores de workflow. También en este caso se ha medido el *makespan* de la aplicación, y el entorno de ejecución fue el descrito en la sección 5.3 de este capítulo. Los detalles de los escenarios utilizados se encuentran en la sección 5.5.3.

Las tres fases tenían como finalidad mostrar la flexibilidad y transparencia de ShcedFlow, también ver la potencialidad de tener un mecanismo de replanificación que apoye las políticas de planificación. Finalmente observar el comportamiento del *makespan* en diferentes escenarios de prueba y con diferentes aplicaciones utilizando SchedFlow.

5.5. Resultados Experimentales

En la siguiente sección se presentan los resultados obtenidos de todos los experimentos realizados en nuestro trabajo de investigación, los cuales incluyen un análisis relacionado a cada experimento planteado en la sección 5.4 de este capítulo.

5.5.1. Experimentos de integración y medición del Makespan para las aplicaciones Montage y LIGO

Las figuras 5.3 y 5.4 muestran los resultados obtenidos de la experimentación comentada en la fase 1, donde el eje "X" representa la política de planificación, y el eje "Y" representa el *makespan* promedio calculado de 120 ejecuciones.

Analizando este experimento se puede ver que la ejecución de ambas aplicaciones workflow tienen un comportamiento similar, donde a simple vista vemos que la política de planificación que da los mejores resultados es la Min-Min, algo previsible porque esta política intenta buscar siempre aquellos recursos de

cómputo que dan los menores tiempos de ejecución (aunque esto siempre está estrechamente vinculado al rendimiento de los recursos).

Ahora bien luego vemos que los resultados que se obtienen con la política de planificación aleatoria, está prácticamente igual que el obtenido por la política por defecto que tiene el gestor de workflow DAGMan, esto tiene una explicación sencilla, ya que justamente DAGMan hace una planificación aleatoria de las tareas, aquí se demuestra que la integración de la política aleatoria se ajusta a la realidad, demostrando así la efectividad en la integración que se obtiene con SchedFlow[65].

Otro punto importante a destacar es el hecho de que incluso con una política de planificación que incluye un pequeño grado de inteligencia como es el caso de Min-Min, ya se obtiene una ganancia en el *makespan* de la aplicación que se encuentra en el orden del *10 % para la aplicación Montage, y 18 % para la aplicación LIGO*.

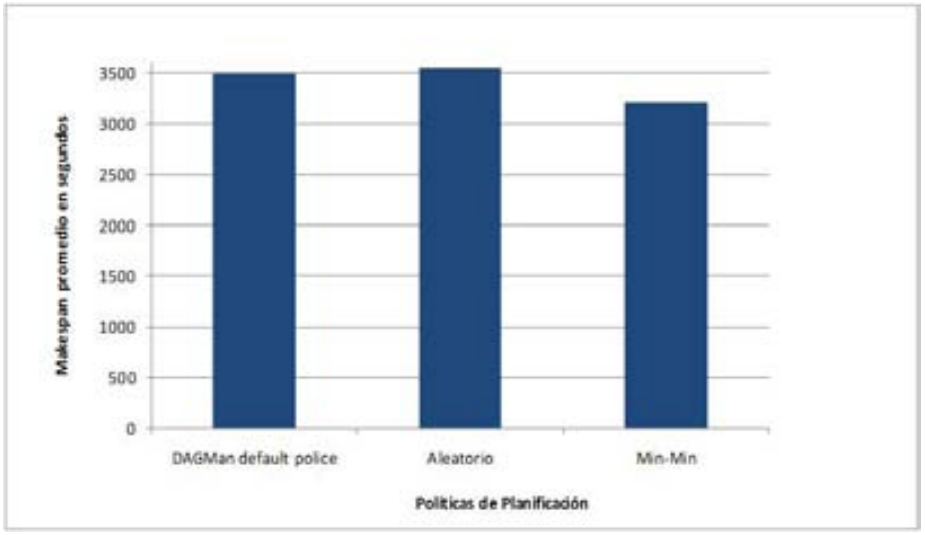

Figura 5.3: Ejecución de Montage utilizando SchedFlow y diferentes políticas de planificación.

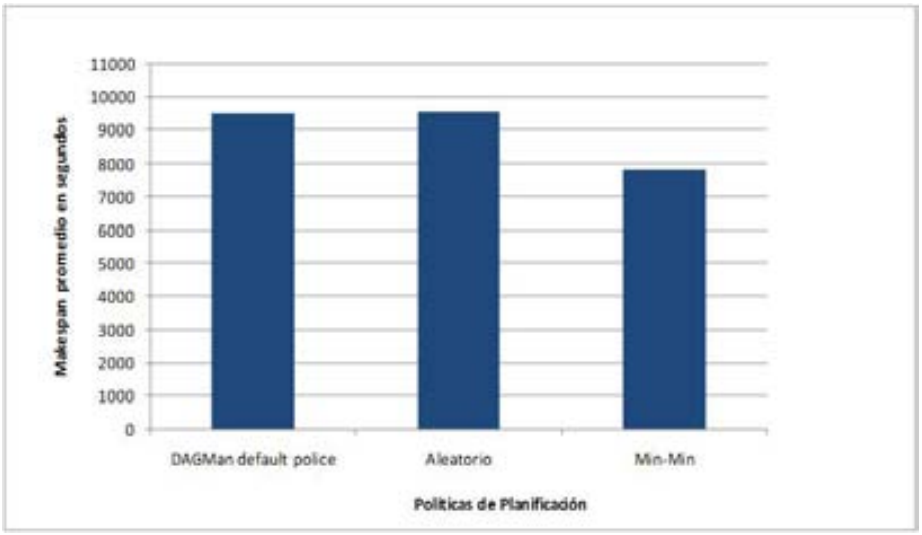

Figura 5.4: Ejecución de LIGO utilizando SchedFlow y diferentes políticas de planificación.

5.5.2. Experimentación con diferentes gestores de workflow en condiciones ideales y normales

Se han realizado un conjunto total de cuatro experimentos que estaban destinados a demostrar la flexibilidad, transparencia, y facilidad de uso de SchedFlow para integrar políticas de planificación en diferentes gestores de workflow.

El primer escenario considera la integración de las políticas de planificación Random, Min-Min, HEFT, y BMTC, en el gestor de workflow condor-DAGMan . Para esto hemos realizado la ejecución de las aplicaciones Montage (56 nodos) y LIGO (81 nodos). Otros elementos que se han tomado en cuenta son, el entorno de ejecución descrito en la sección 5.3, el número de ejecuciones para cada política fueron 120, y en este caso particular se configuro el entorno de ejecución para que funcionara en condiciones ideales.

Este escenario ha considerado unas condiciones ideales del entorno de ejecución, perseguía el objetivo de conocer cuáles eran los valores ideales del *makespan* que podíamos obtener, si los mejores recursos de cómputo estaban disponibles. Para luego hacer las comparativas de los resultados en condiciones ideales, con los obtenidos cuando se hacen con las condiciones que normalmente se tienen en la realidad.

Las figuras 5.5 y 5.6 muestran los resultados donde el eje "X" define la política de planificación, mientras que el eje "Y" define el *makespan* promedio.

Luego se planteó un segundo escenario [63], donde se utilizaron las mismas políticas de planificación, pero se inyectó aleatoriamente pero de forma controlada tres tareas con una prioridad mayor a la de cualquier tarea que se estuviese ejecutando en los recursos. Esto con la finalidad de que algunas de las tareas fueran suspendidas temporalmente (de acuerdo con las políticas aplicadas por Cóndor), lo cual afectó la ejecución general del workflow conjunto. Esta fue la forma de reflejar una situación muy habitual en los entornos no-dedicado.

El problema es que cuando esto ocurre y dado que nuestras tres políticas de planificación se aplicaron de forma estática, ninguna tuvo una acción correctiva que replanificará las tareas que fueron suspendidas. Esta acción solo estuvo gestionada por Cóndor DAGMan que solamente reintenta de forma automática en el mismo equipo, porque la asignación original no fue modificada. Los resultados de esos experimentos se muestran en las figuras 5.7 y 5.8, como era de esperar todas las políticas de planificación obtuvieron tiempos de ejecución muy elevadas en comparación con el caso anterior.

Los resultados obtenidos, cuando no se produzcan suspensiones (primer escenario), podemos ver que la utilizan políticas de planificación que tengan en cuenta características como rutas críticas, tiempos de ejecución, evidentemente reducirán el *makespan* de la aplicación. Además nos indica la necesidad de contar con una arquitectura como la que nos proporciona SchedFlow, que ayudará al usuario final para seleccionar la política de planificación que da el mejor rendimiento de su aplicación.

Por otro lado los resultado obtenido en el segundo escenario, donde inducimos de forma controlada algunas suspensiones, el *makespan* final se ve afectado de manera diferente en cada política de planificación. La consecuencia para una estrategia simple como Min-Min es que obtenga tiempos de ejecución similares al de BMCT. Esto es sencillo de explicar porque esta política de planificación, no toma en cuenta información adicional sobre la ruta crítica del workflow.

HEFT se vio afectada significativamente por los eventos de suspensión inyectados en tiempo de ejecución, que modificó la ejecución de las estimaciones utilizadas durante el proceso de mapping (teniendo una desviación de hasta un 26 % en los casos más graves). Esto es fácil de entender porque HEFT es una política de planificación estática, y si algún recurso de cómputo falla, HEFT debe re-programa todas las tareas pendientes.

Una vez analizados los resultados de los dos primeros escenarios, nos decidimos a ver qué sucede en un tercer escenario donde ejecutamos el Montage con el mecanismo de replanificación mencionado en la sección 4.1.6 del capítulo 4, siempre mirando el *makespan* final obtenido. Para poder ver el efecto se realizo

con el mismo gestor de workflow Condor-DAGMan, y el entorno de ejecución descrito en la sección 5.2.

En este escenario, se utilizaron las mismas políticas de planificación e inyectamos las tareas que produzcan las suspensiones. Bajo la aparición de estas suspensiones en tiempo de ejecución, SchedFlow llama a la función de gestión de eventos reaccionando para volver a asignar la tarea suspendida, y algunas de las otras tareas que aún se encontraban a la espera de ser ejecutadas. Los resultados de este experimento se pueden ver en las figuras 5.9 y 5.10.

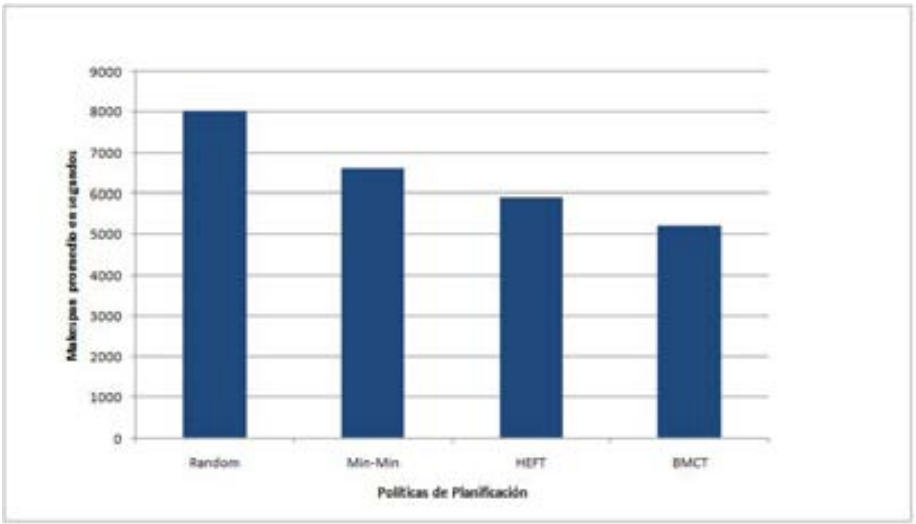

Figura 5.5: Ejecución del Montage utilizando diferentes políticas de planificación en condiciones ideales con el gestor DAGMan.

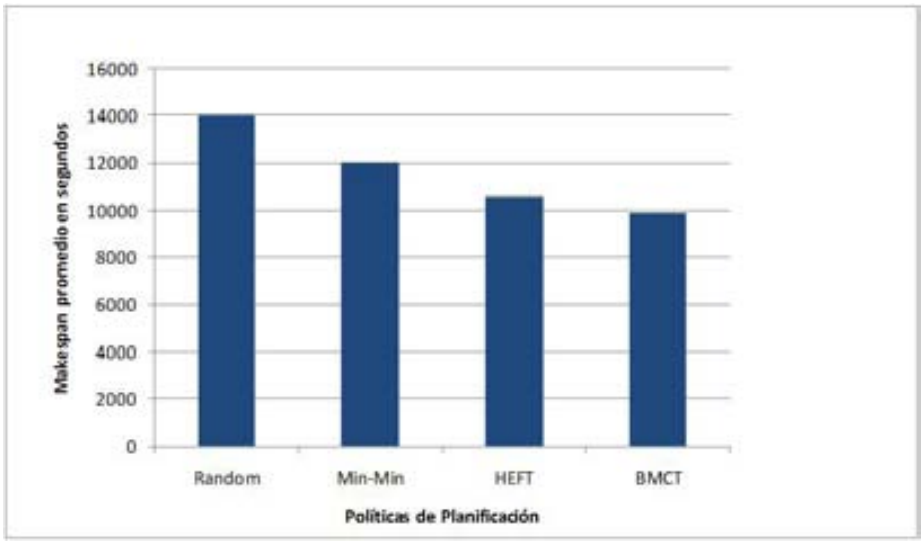

Figura 5.6: Ejecución de LIGO utilizando diferentes políticas de planificación en condiciones ideales con el gestor DAGMan.

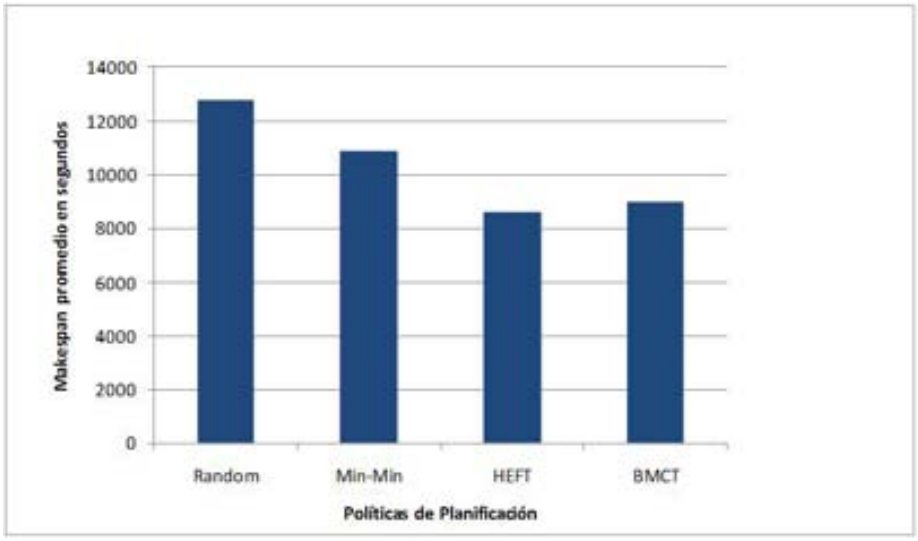

Figura 5.7: Ejecución del Montage con diferentes políticas de planificación integradas en DAGMan provocando suspensiones.

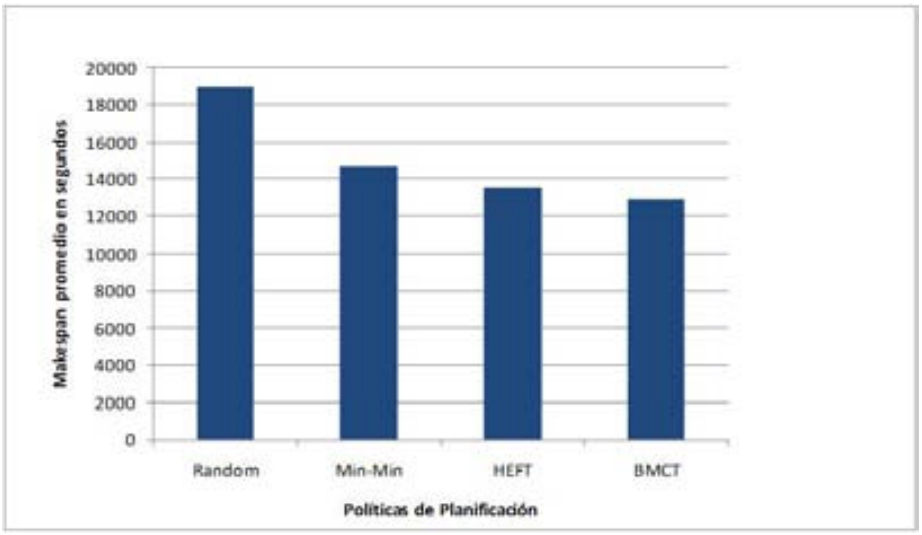

Figura 5.8: Ejecución de LIGO con diferentes políticas de planificación integradas en DAGMan provocando suspensiones.

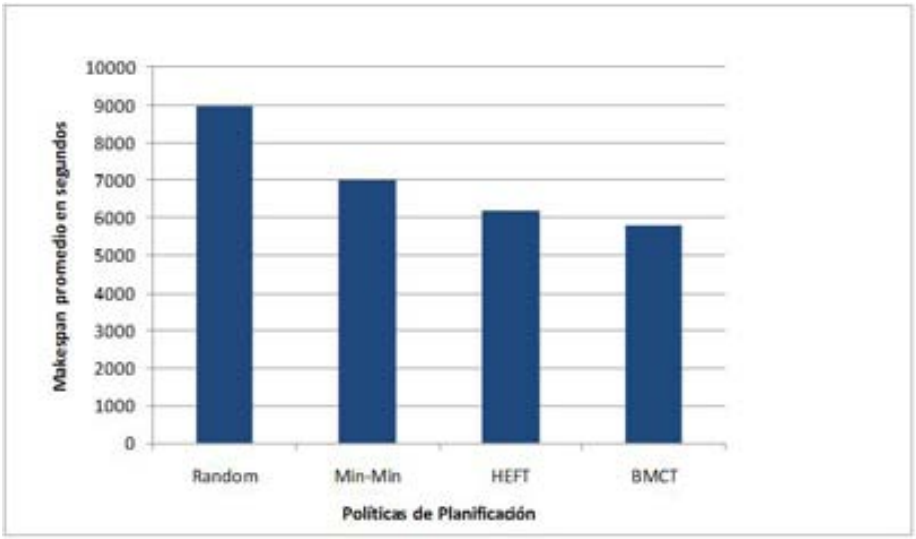

Figura 5.9: Ejecución del Montage con diferentes políticas de planificación integradas en DAGMan, incluyendo el mecanismo de replanificación.

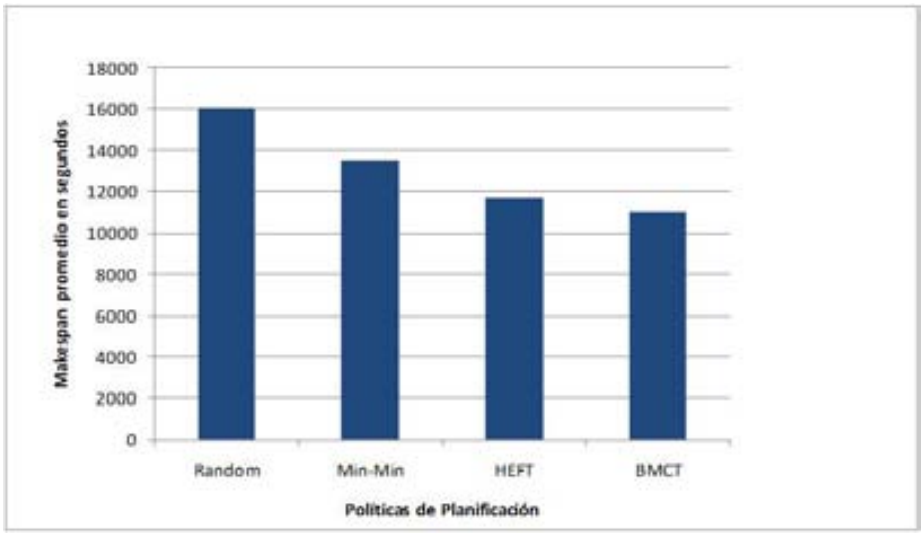

Figura 5.10: Ejecución de LIGO con diferentes políticas de planificación integradas en DAGMan, incluyendo el mecanismo de replanificación.

Los resultados observados en el tercer escenario fueron muy satisfactorios, y demuestran que el uso de un mecanismo dinámico de replanificación, capaz de reaccionar en caso de detectar algún evento es beneficioso para todas las estrategias. Se ve claramente que el *makespan* es muy similar al obtenido en el caso ideal donde no ocurren eventos.

En la mayoría de los casos la diferencia individual entre las ejecuciones, siempre fueron de menos de un 5 %, lo que significa que los resultados son muy homogéneos si tenemos en cuenta el hecho de que cada suspensión, requiere una ejecución total de la estrategia de planificación para replanificar todas las tareas que está en la espera de ser ejecutadas. Es importante señalar que esta sobrecarga adicional es muy pequeña, pues solo se inyectan un número de suspensiones no mayor a tres.

Dado que en la literatura contamos con muchos gestores de workflow, decidimos hacer una experimentación exactamente igual a la anteriormente descrito para utilizando los gestores de workflow Taverna y Karajan. Esto con la finalidad de ver si SchedFlow nos proporcionaba buenos resultados, en diferentes plataformas que gestionan los workflow. Los resultados con el gestor de workflow Taverna se pueden ver en las figuras 5.11y 5.12 (Condiciones ideales), 5.13 y 5.14 (inyectando suspensiones), 5.15y 5.16 (inyectado suspensiones pero incluyendo un gestor de eventos).

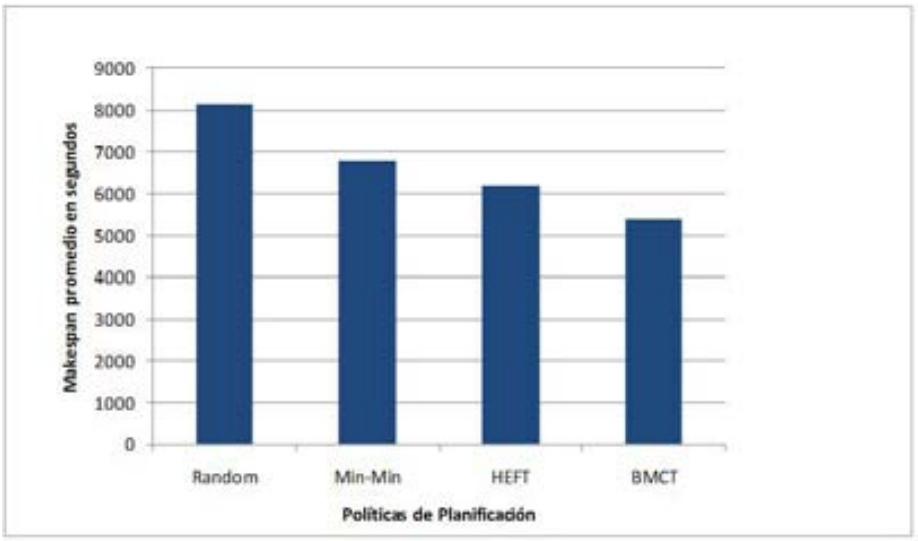

Figura 5.11: Ejecución del Montage en Taverna con diferentes políticas de planificación en condiciones ideales.

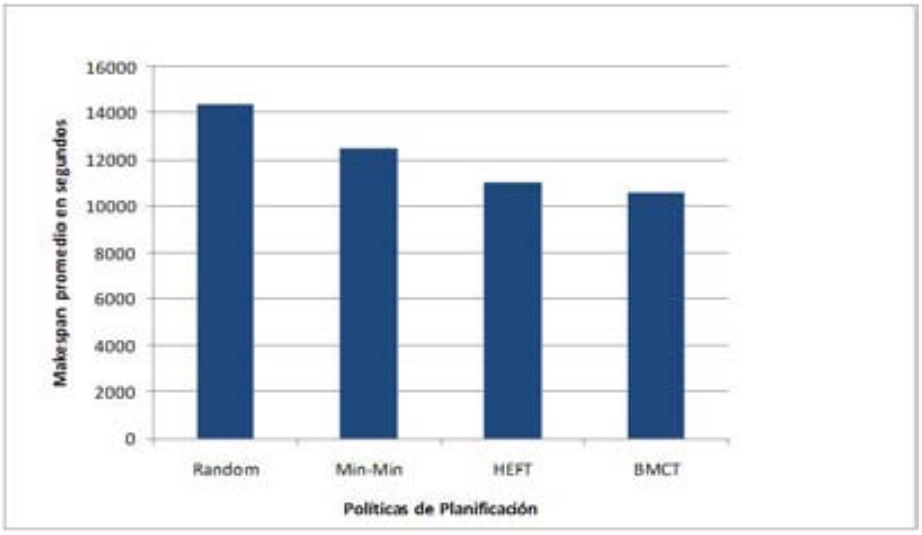

Figura 5.12: Ejecución de LIGO en Taverna con diferentes políticas de planificación en condiciones ideales.

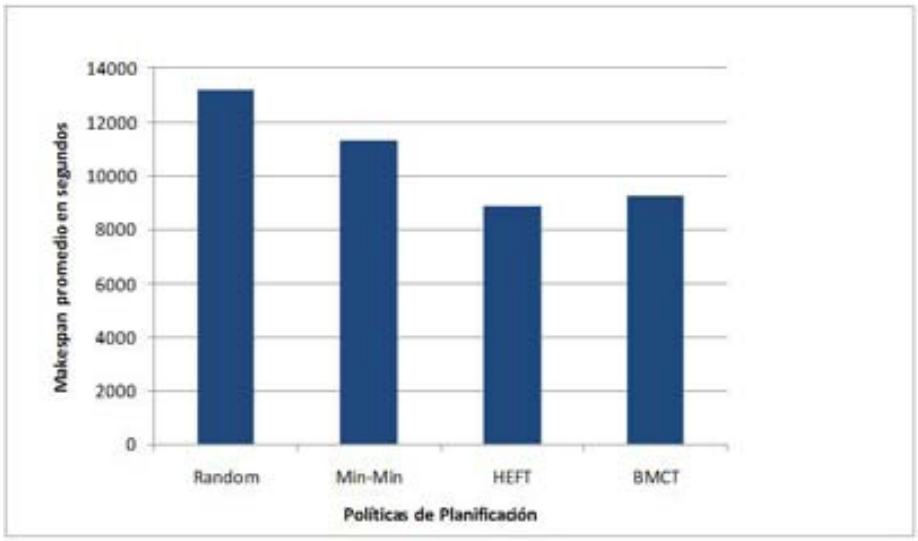

Figura 5.13: Ejecución del Montage en Taverna con diferentes políticas de planificación provocando suspensiones intensionalmente.

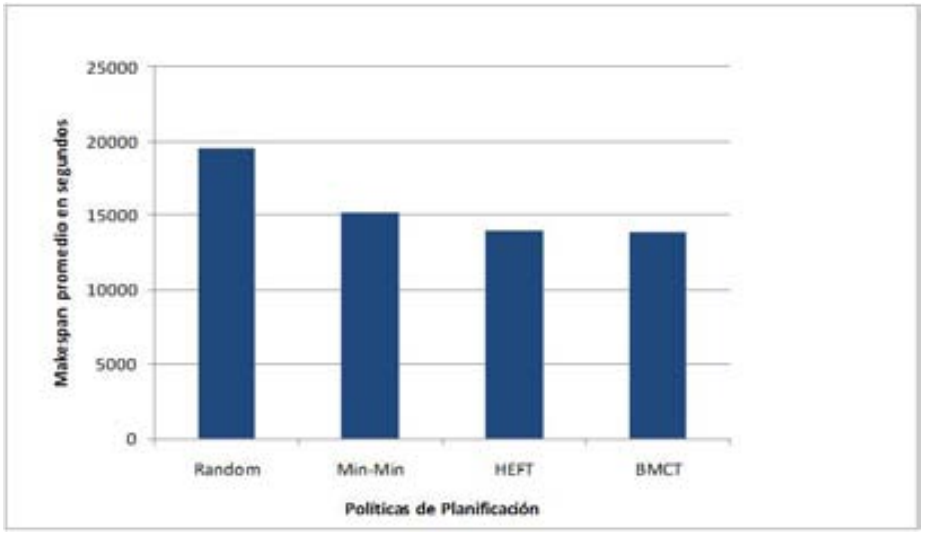

Figura 5.14: Ejecución del LIGO en Taverna con diferentes políticas de planificación provocando suspensiones intensionalmente.

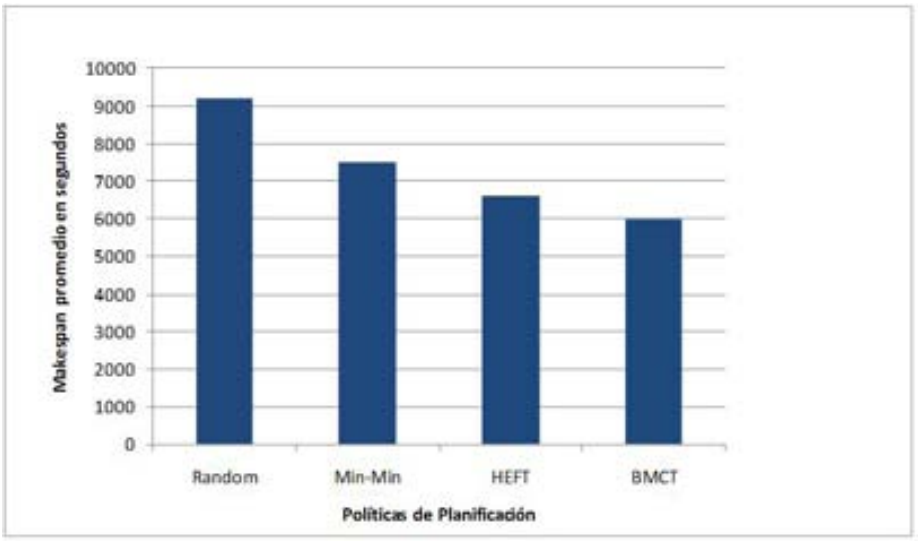

Figura 5.15: Ejecución del Montage en Taverna con diferentes políticas de planificación provocando suspensiones, e incluyendo un mecanismo gestor de eventos.

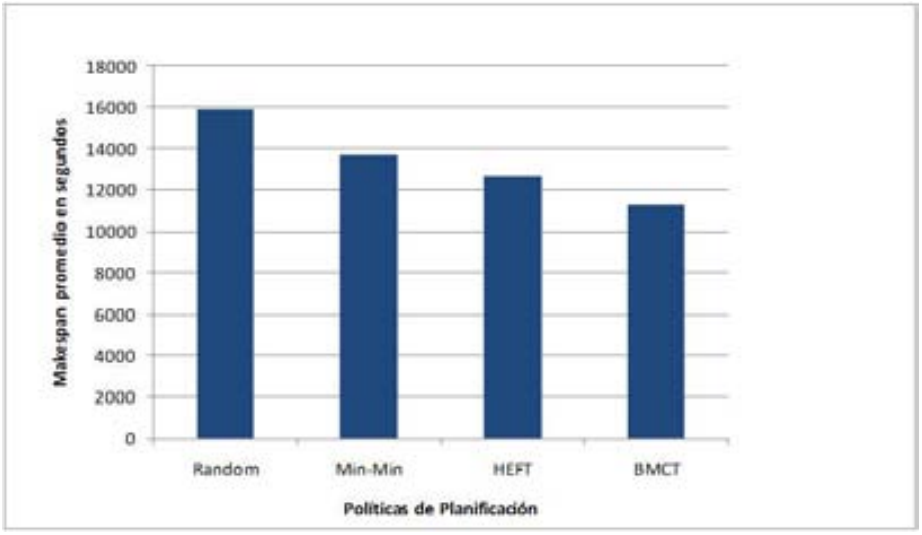

Figura 5.16: Ejecución de LIGO en Taverna con diferentes políticas de planificación provocando suspensiones, e incluyendo un mecanismo gestor de eventos.

Los resultados al repetir los experimentos variando el gestor de workflow Condor-DAGMan por Taverna, pudimos observar que los resultados son bastante similares a los obtenidos con DAGMan, esto nos indica que las políticas

de planificación no se ven afectadas de forma significativa al tipo de gestor utilizado, incluso cuando las aplicaciones son diferentes.

Esto tiene sus ventajas ya que damos más posibilidades de poder enlazar diferentes gestores a nuestro *framework*, abriendo alternativas al usuario final para hacer pruebas no solo con diferentes políticas de planificación, sino también incluir diversos gestores en función del utilizado en su organización.

Finalmente para continuar con las pruebas integración de políticas en diversos gestores de workflow utilizando SchedFlow, hemos cambiado esta vez el gestor de workflow Taverna, por otro llamado Karajan que también nos permite enlazar de forma sencilla nuestro *framework*, repitiendo nuevamente los experimentos iniciales.

Los resultados con el gestor de workflow Karajan pueden verse en las figuras 5.17 y 5.18 (condiciones ideales), 5.19y 5.20 (inyectando suspensiones), 5.21 y 5.22 (inyectado suspensiones pero incluyendo un gestor de eventos). Nuevamente en los casos de condiciones ideales y cuando incluimos el gestor de eventos, los resultados se repiten como en los dos escenarios anteriores. Sin embargo en el caso donde se incluyen las suspensiones sin gestión de eventos, hay un pequeño ya que la política HEFT no se ve tan afectada, esto ha sido porque al ver los recursos que estaban disponibles en esos experimentos siempre fueron las que tenían mejor rendimiento, pero en general la situación se mantiene.

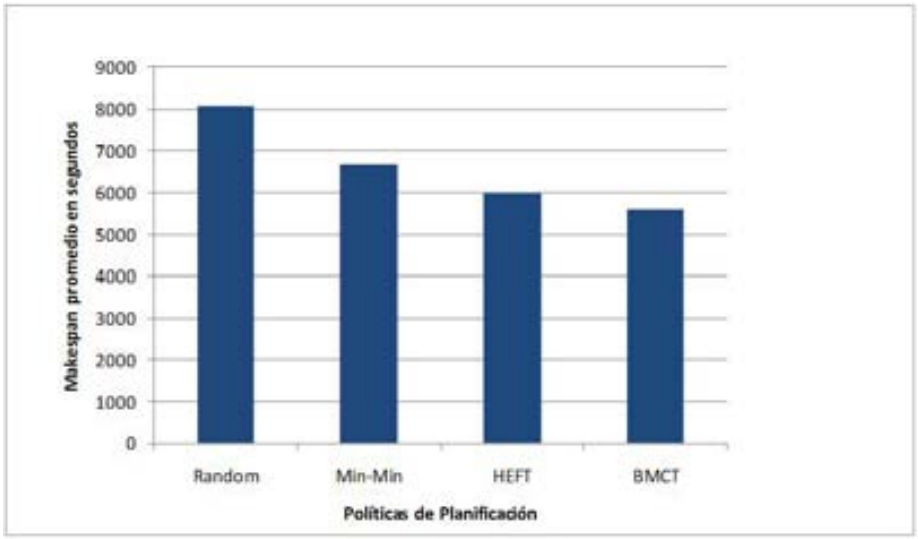

Figura 5.17: Ejecución del Montage en Karajan con diferentes políticas de planificación en condiciones ideales.

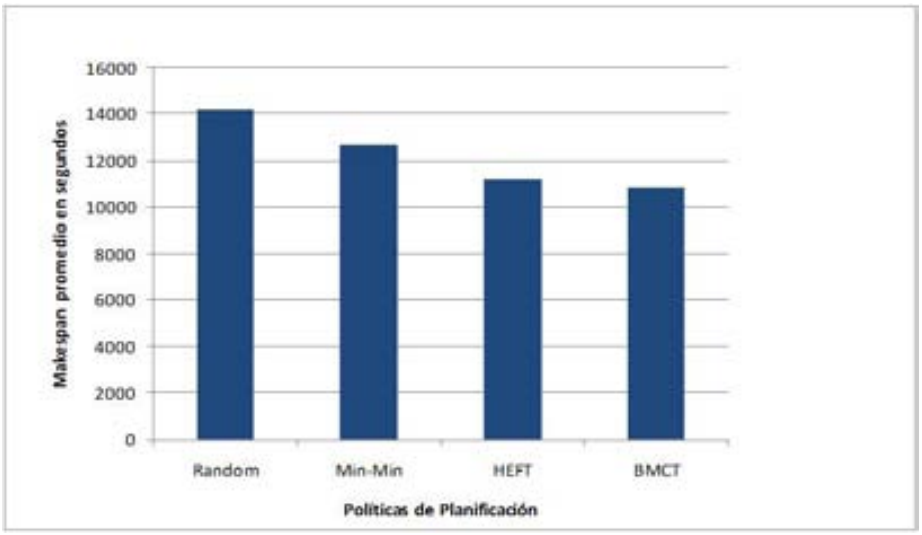

Figura 5.18: Ejecución de LIGO en Karajan con diferentes políticas de planificación en condiciones ideales.

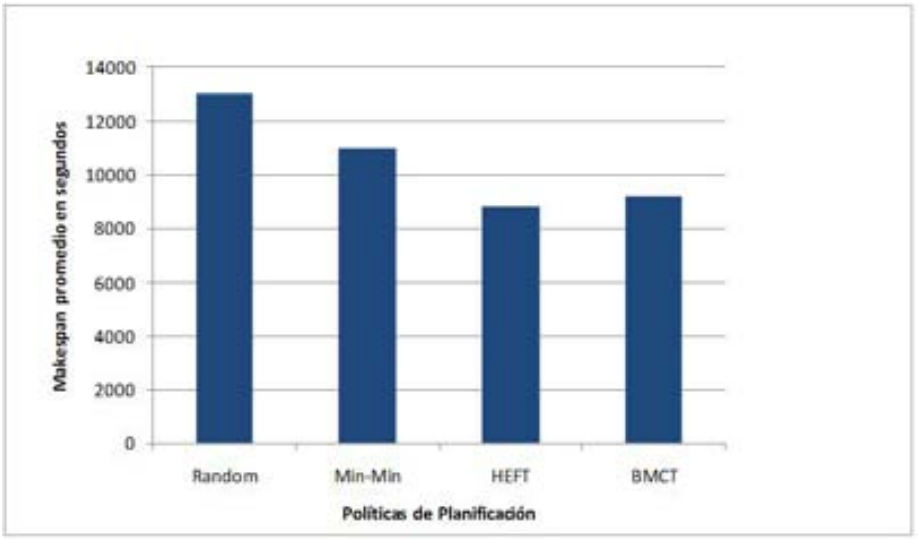

Figura 5.19: Ejecución del Montage en Karajan con diferentes políticas de planificación provocando suspensiones intensionalmente.

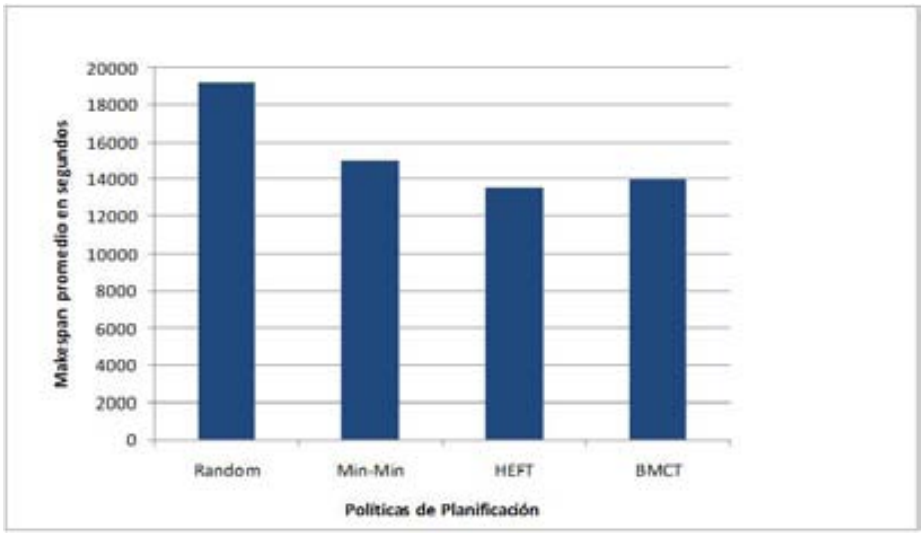

Figura 5.20: Ejecución de LIGO en Karajan con diferentes políticas de planificación provocando suspensiones intensionalmente.

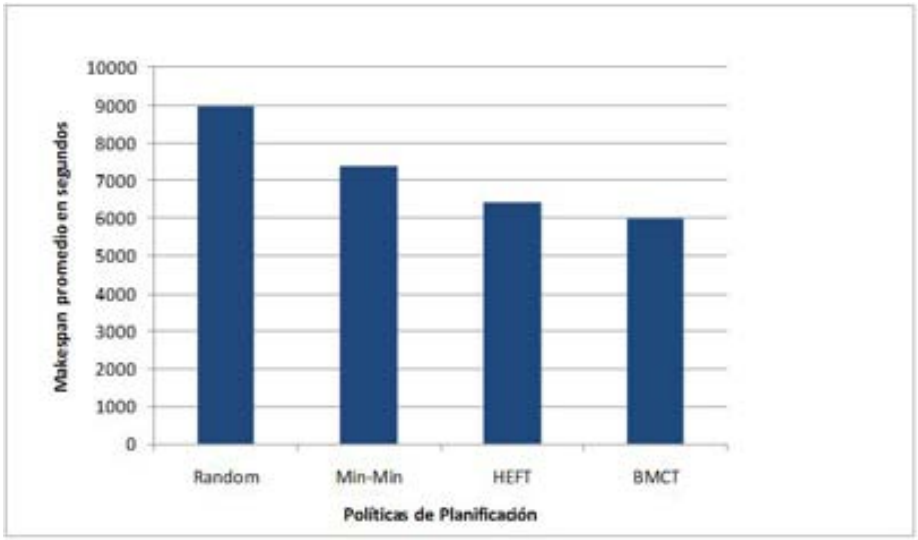

Figura 5.21: Ejecución del Montage en Karajan con diferentes políticas de planificación provocando suspensiones, e incluyendo un mecanismo gestor de eventos.

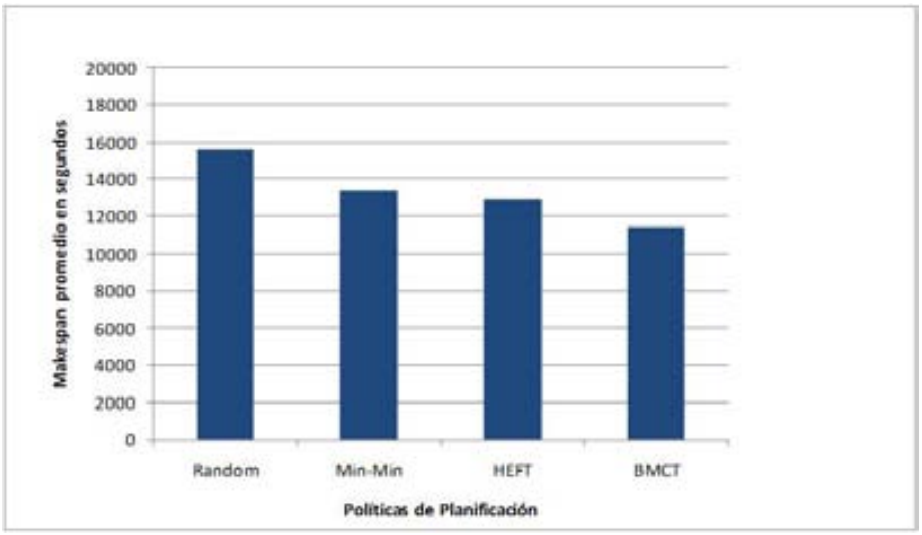

Figura 5.22: Ejecución de LIGO en Karajan con diferentes políticas de planificación provocando suspensiones, e incluyendo un mecanismo gestor de eventos.

5.5.3. Experimentos de la variación en la carga inicial de las aplicaciones

Utilizando el entorno de ejecución descrito en la sección 5.3 se llevaron a cabo cuatro series de experimentos, donde SchedFlow ejecuto diferentes políticas de planificación, con las aplicaciones Montage y LIGO variando en este caso la carga inicial de las aplicaciones, donde nuestro interés esta es medir nuevamente en *makespan*.

Pero nuestro objetivo principal observar el comportamiento de las políticas de planificación ante los cambios iniciales de carga, es decir queremos ver cuán sensibles son los resultados finales cuando variamos el *workload* inicial pero utilizando el mismo gestor, aplicación, y política de planificación. Tomando como índice de prestaciones el *makespan* de la aplicación. La experimentación que presentamos tiene algunas características que debemos mencionar, para aclarar lo que hemos utilizado en cada escenario de prueba:

Nuestro primer escenario consiste en realizar la ejecución de la aplicación Montage, utilizando la funcionalidad ofrecida por los gestores de workflow. Esto significa que utilizaremos las políticas de planificación que poseen estos tres gestores de workflow (Taverna, DAGMan, y Karajan), que además poseen diferentes esquemas de planificación. Todas muy básicas pero que nos sirve como punto de partida para comprobar la eficiencia que cada uno de ellos

ofrece, también nos permitirá realizar posteriormente comparativas con nuestro *framework* una vez completados los experimentos.

El workload utilizado en este experimento ha sido de 400 MB, con el cual hemos realizado un total de 120 ejecuciones, con la aplicación Montage cuyo número de nodos son 53, las acciones correctivas ante un fallo eran soportadas con los mecanismos proporcionados por el gestor de workflow, el entorno lo mantuvimos controlado para garantizar 45 recursos disponibles de los 140 que teníamos en total. El índice de prestaciones que hemos seleccionado observar ha sido el *makespan* [64].

La figura 5.23 nos muestra los resultados obtenidos, donde el eje de las "X" representa el gestor de workflow utilizado (con su política de planificación por defecto), mientras el eje de las "Y" representa el índice de prestaciones promedio (*makespan*) de las 120 ejecuciones expresado en segundos.

Una vez visto los resultados se ha repetido el experimento, pero esta vez utilizando SchedFlow, y ejecutando diferentes políticas de planificación aplicadas sobre cada uno de los gestores de workflow, para ver el comportamiento y posibles mejoras que obtendremos en el índice de prestaciones, esto en parte demostrara la ventaja de utilizar nuestro sistema y poder comparar rápidamente que política puede ir mejor con una determinada aplicación.

En este caso todos los parámetros del escenario se mantienen, seguimos garantizando 45 recursos, el workload aun sigue siendo de 400 MB, pero el soporte de replanificación lo hacemos con SchedFlow. Los gestores son utilizados para gestionar las dependencias, y poder monitorizar desde SchedFlow toda la información que ellos proveen, para luego utilizarla tomando acciones que ayuden a mejorar el *makespan*. La figura 5.24 nos muestra los resultados obtenidos de este experimento.

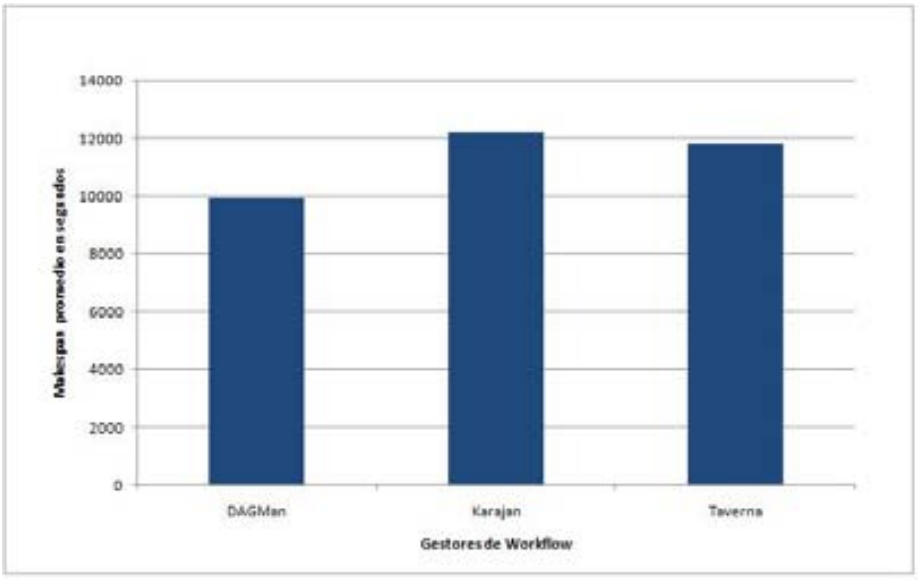

Figura 5.23: Ejecución del Montage con las políticas por defectos de los gestores de workflow y carga inicial de 400 MB.

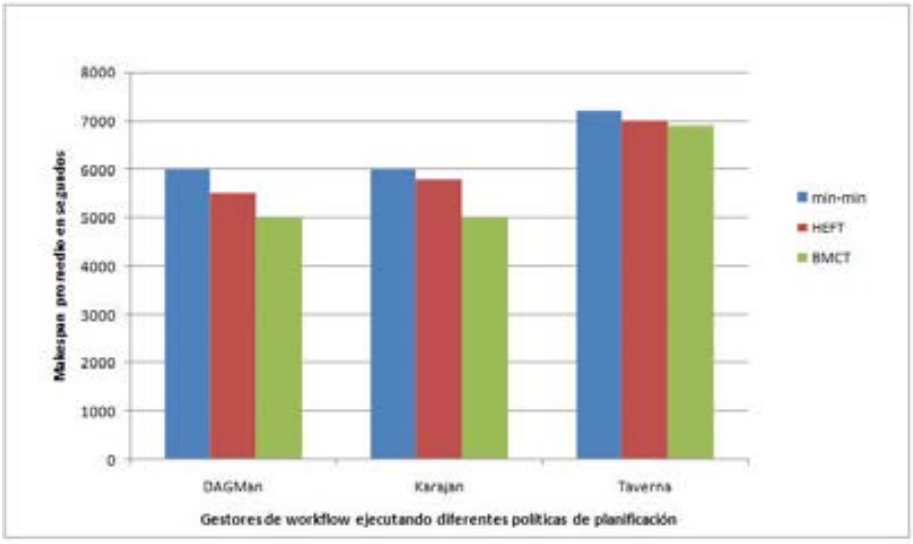

Figura 5.24: Ejecución del Montage utilizando SchedFlow con una carga inicial de 400 MB.

Analizando los resultados anteriores se pueden comentar algunos detalles importantes, a simple vista tenemos que una vez integradas las políticas de planificación en los gestores de workflow, el *makespan* promedio de la aplicación

mejora de forma inmediata, algo que era previsible al usar una política más compleja.

Por otro lado se aprecia que indistintamente del gestor de workflow utilizado, la política que consigue mejores resultados con la carga inicial de 400 MB es la BMCT, siendo los resultados muy similares tanto en DAGMan como en Karajan esto nos indica claramente que tienen una gestión similar, cosa que no sucede en Taverna. Pero lo importante es que si utilizamos SchedFlow para integrar políticas más sofisticadas, es evidente que el índice de prestación (*makespan*) mejora sus resultados, con reducciones que están por el orden del **30 %**, demostrando así la potencia de SchedFlow.

Continuando con el orden de ideas e investigación principal que nos toca, ahora presentamos los resultados obtenidos al variar la carga inicial de 400MB a 1024MB. Las Figuras 5.25 y 5.26 se corresponden a los resultados obtenidos.

Inicialmente vemos que los resultados obtenidos de la ejecución, con los gestores de workflow utilizando su política de planificación, estos resultados a nivel de gestores se mantienen solo que tenemos tiempos de ejecución mucho más elevados producto de aumentar la carga inicial de la aplicación.

Ahora cuando hacemos la comparación con las diferentes políticas hemos notado cambios importantes, el primero de ellos es que la política con mejor resultado no es BMCT, ahora pasa a ser HEFT, esto motivado a solo el cambio del workload. La explicación a este cambio viene dado por el hecho que la política HEFT organiza la planificación de una mejor manera en la medida que los datos van creciendo en tamaño obteniendo así una planificación más adecuada donde la consecuencia es una mejora del *makespan* como se ve en las figura 8.

Por otro lado BMCT disminuye su rendimiento porque su planificación está basada en un valor llamado rank, el cual le da una prioridad a las tareas. Si este valor no es lo suficientemente exacto la política reacciona de forma incorrecta y por eso los resultados no son los esperados.

Esto trae como consecuencia que se llevan a cabo experimentos donde los resultados pudieron ser mejores, y por no tener en conocimiento cierto si era esa la política correcta se pierda tiempo innecesario. Esto con aplicaciones pequeñas puede representar diferencias de horas, pero pensando en aplicaciones con cálculo intensivo, estas diferencias pueden ser de días, sin contar que en nuestro caso para poder realizar la comparativa el entorno de ejecución estuvo siempre controlado cosa que generalmente en un Grid es muy complicado hacer.

Esto nos demuestra que las políticas de planificación son sensibles a las cargas iniciales, también que no existe la mejor política para determinada aplicación,

los experimentos nos dejaron ver como son los resultados reales en los gestores de workflow. Finalmente podemos ver el potencial de nuestro *framework,* para evaluar un factor importante dentro de la planificación de workflow.

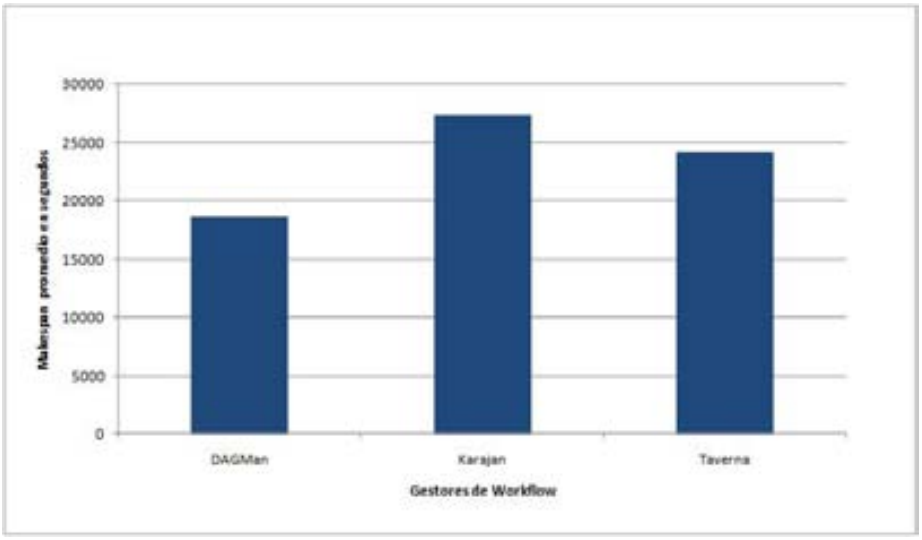

Figura 5.25: Ejecución del Montage con las políticas por defectos de los gestores de workflow y carga inicial de 1024 MB.

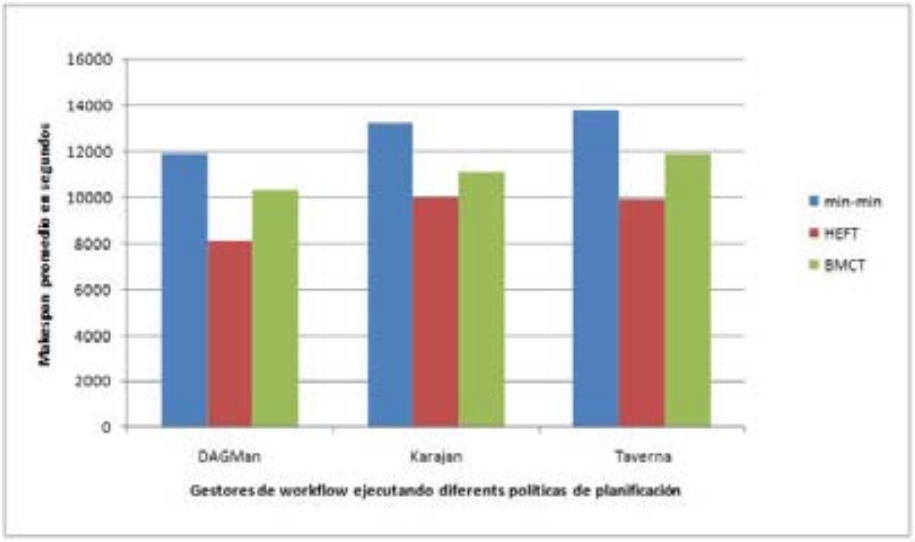

Figura 5.26: Ejecución del LIGO utilizando SchedFlow con una carga inicial de 1024 MB.

Una vez completados los experimentos con la aplicación Montage, hemos decidido repetir los escenarios antes detallados, pero utilizando una pequeña

parte de la aplicación LIGO (un total de 81 nodos), y donde existe un alto volumen de comunicaciones para analizar los resultados del *makespan*. En este caso también haremos la variante de carga inicial como hicimos con la aplicación Montage. Los resultados obtenidos para cargas de 400 MB y 1024 MB están representados en las figuras 5.27, 5.28, 5.29, 5.30 respectivamente.

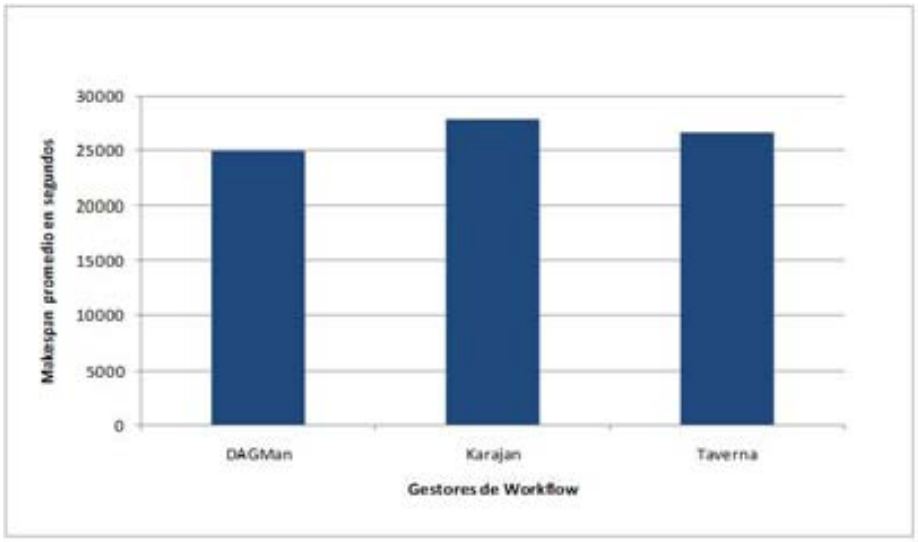

Figura 5.27: Ejecución del LIGO con las políticas por defectos de los gestores de workflow y carga inicial de 400 MB.

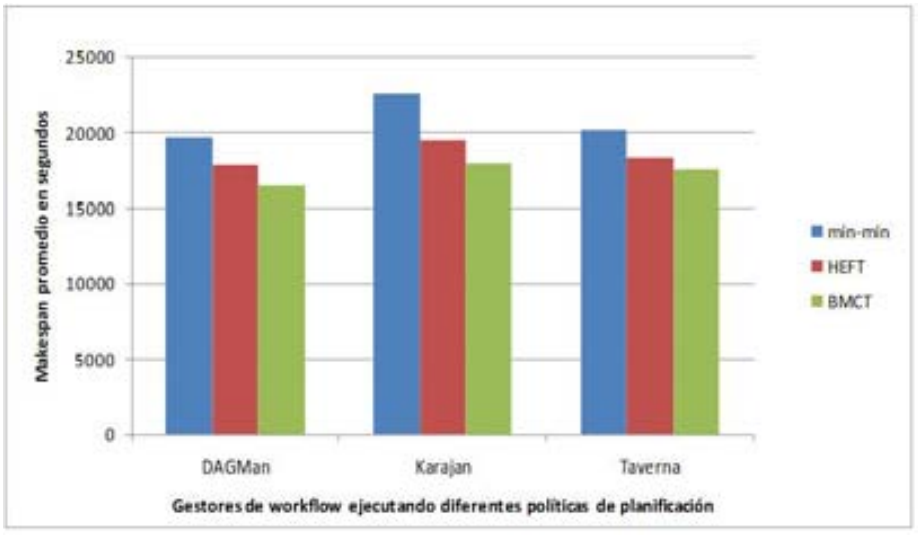

Figura 5.28: Ejecución del LIGO utilizando SchedFlow con una carga inicial de 400 MB.

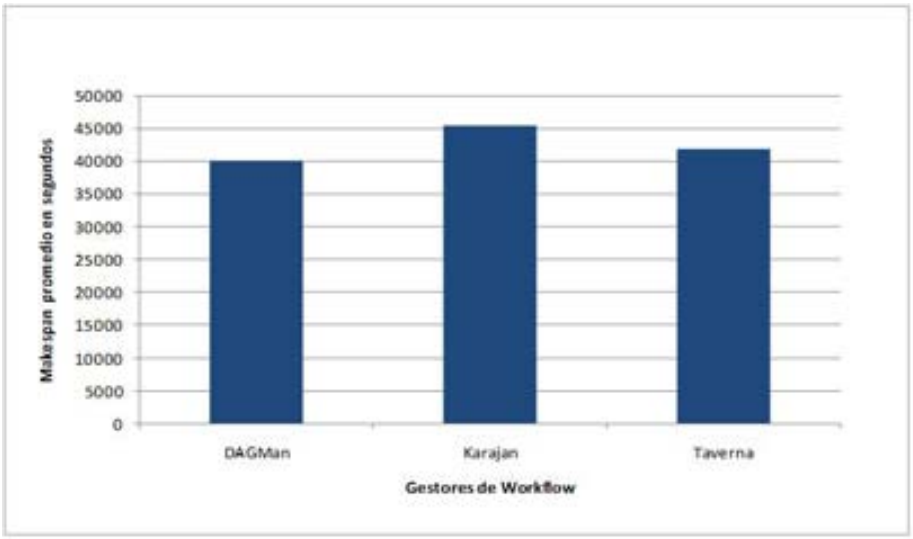

Figura 5.29: Ejecución del LIGO con las políticas por defectos de los gestores de workflow y carga inicial de 1024 MB.

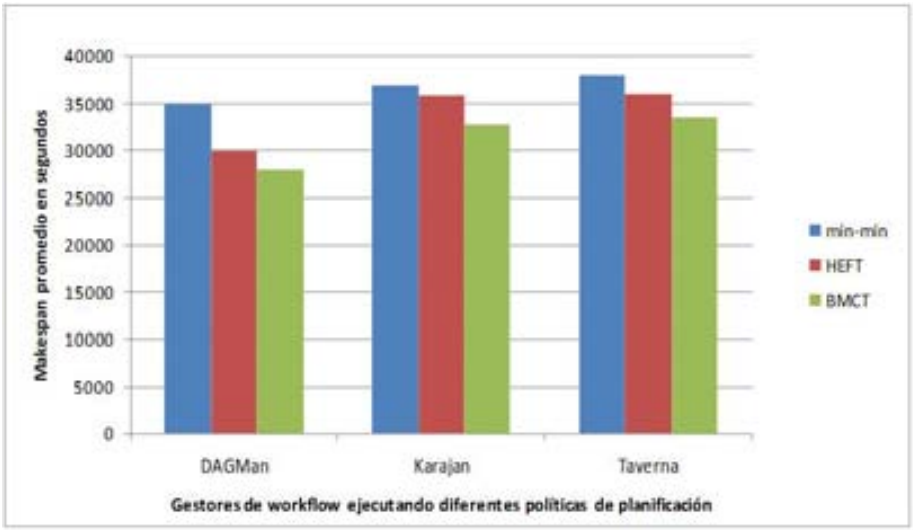

Figura 5.30: Ejecución del LIGO utilizando SchedFlow con una carga inicial de 1024 MB.

Estos experimentos realizados con LIGO nos revelan algunos datos interesantes, lo primero es la consistencia en los resultados ya que se mantiene de forma coherente el comportamiento de las diferentes políticas de planificación, tanto para la carga inicial de 400 MB como para la de 1024 MB. Siendo BMCT la política de planificación que nos da los mejores resultados de *makespan* para

todos los casos. Estos resultados se deben a que el volumen de comunicaciones es muy bien gestionado por esta política, ya que trabaja la planificación sobre la ruta crítica.

Por otro lado vemos que si comparamos estos resultados con los obtenidos en el caso de Montage con carga inicial de 1024 MB, los resultados cambian pues la mejor política de planificación es la HEFT, esto demuestra que según la variante de carga, tipo de workflow, dinámica del entorno de ejecución pueden hacer cambiar los resultados incluso en aplicaciones workflow con mucha similitud.

SchedFlow representa un gran aporte para la comunidad de investigadores, pues les permitirá poder realizar pruebas experimentales en diversos escenarios, sin que esto les suponga un gran esfuerzo. Además de poder ejecutar en plataformas con diferentes gestores de workflow ahorrando un tiempo importante al momento de ejecutar la aplicación, puesto que no debe preocuparse del gestor y dedicar más tiempo en la política de planificación.

En este capítulo hemos visto las pruebas experimentales que se han llevado a cabo con ShedFlow, los análisis realizados a cada uno de los experimentos que se hicieron con las variantes que nos permitieron demostrar la potencialidad de nuestro *framework*. También se muestra el detalle de las aplicaciones utilizadas así como el entorno de ejecución que utilizamos en este trabajo de investigación.

Capítulo 6

Conclusiones y Líneas Abiertas

En este trabajo de investigación se ha estudiado el problema de la ejecución de aplicaciones workflow en entornos no dedicado, mostrando las principales conclusiones de la tesis. También presentamos las líneas abiertas de investigación a seguir relacionadas con este trabajo.

Los entornos distribuidos presentan una alternativa para el uso compartido de recursos, con el que se puede obtener una gran potencia de cálculo con unos costos muy reducidos. El problema es que la heterogeneidad de estos entornos hace indispensable, tener a disposición un middleware ó software que proporcione una visión unificada de los recursos.

Pero también tiene su ventaja porque representa un punto de apoyo, que permite la colaboración entre grandes grupos de científicos, que lleven adelante importantes soluciones a diversos problemas. SchedFlow es un *framework* creado para extender el uso de planificadores distribuidos, a nuevos tipos de aplicaciones que hasta el momento ha sido poco tratadas en entornos de ejecución reales, como es el caso de las aplicaciones workflow.

SchedFlow representa un novedoso sistema que permite la integración de políticas de planificación, replanificación dinámica de tareas, y una ejecución eficiente de las aplicaciones workflow, todo realizado de forma sencilla, flexible, y transparente al usuario. Los problemas detectados en este trabajo de investigación fueron la falta de integración entre las áreas de planificación y gestión de workflow, carencia de mecanismos automáticos para la replanificación, un alto nivel de complejidad en los entornos distribuidos. Todo el conjunto de

ellas fueron analizadas y atacadas mediante una propuesta de un *framework* robusto pero sencillo de manejar, que combinase las áreas de planificación y gestión de forma transparente al usuario.

Nuestra propuesta fue desarrollar SchedFlow un *framework* capaz de integrar de manera sistemática los gestores existentes y las políticas de planificación, sin que esto represente un esfuerzo muy grande para el usuario final, con la ventaja asociada de poder conseguir la reducción del *makespan* que la política de planificación ofrezca. También es posible que se obtenga una reducción producto del mecanismo de replanificación incorporado a SchedFlow.

Los experimentos realizados con SchedFlow en los cuales hemos ejecutado dos importantes aplicaciones para la comunidad científica como son "*Montage y LIGO*", con diferentes políticas de planificación demuestran, por un lado la flexibilidad ofrecida por nuestro *framework*, y por el otro las mejoras importantes en el *makespan* de la aplicación.

Los experimentos adicionales también demostraron, que con solo realizar cambios en la carga inicial de una aplicación workflow, las políticas de planificación se ven afectadas en el *Makespan*. Esto nos hace pensar en el número de diferentes combinaciones que un usuario puede realizar con nuestro *framework,* para lograr tener una ejecución eficiente ajustada al entorno de ejecución que utilice.

Otra conclusión interesante es que si utilizamos un *framework* como el presentado en esta memoria, no tenemos el problema para seleccionar el tipo de gestor que este instalado en los recursos utilizados puesto que se soportan diferentes gestores de workflow. Incluso podemos hacer uso de este nuevo concepto llamado computación voluntaria, donde podemos tener recursos de cómputo en los cuales poder ejecutar algunas tareas más sencillas.

Para finalizar podemos concluir dados los resultados obtenidos en los experimentos realizados con SchedFlow, que la potencialidad de la integración de políticas de planificación alcanza mejoras del *makespan* en más del 30 %, cuando utilizamos el mecanismo de replanificación del que está dotado nuestro *framework*.

A raíz del trabajo realizado se obtuvieron las siguientes publicaciones:

JP 2008 G. Martinez, M. Lopez, E. Heymann, M. Senar, *Shedflow: Sistema Integrador de Politicas de Planificación y Gestores de Workflow*, XIX Jornadas de Paralelismo, pp. 481-487, 2008.

ICCS 2010 G. Martinez, E. Heymann, M. Senar, *Integrating scheduling policies into workflow engines*, Proceedings of the 2010 International Conference on Computational Science, vol. 1, pp. 2743-2752, 2010.

WORKS 2010 G. Martinez, E. Heymann, M. Senar, E. Luque, B.P. Miller, *Using SchedFlow for performance evaluation of workflow applications*, Workflows in Support of Large-Scale Science (WORKS), 2010 5th Workshop on, pp. 1-8, 2010.

6.1. Lineas Abiertas

Las líneas abiertas que hemos detectado sería utilizar esquemas más complejos de multiprogramación, donde exista más de un workflow ejecutándose con distintas prioridades para diferentes tipos de tareas.

Incluir mecanismos *peer-to-peer* para mejorar los tiempos de comunicación para aplicaciones workflow a gran escala, y donde las comunicaciones pueden ocasionar fallos en los sistemas centralizados.

Bibliografía

[1] DAGMan Application Manual. *http://www.cs.wisc.edu/condor/manual/*.

[2] Taverna User Manual. *http://taverna.sourceforge.net/manual/*.

[3] JH Abawajy. Fault-tolerant scheduling policy for grid computing systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 238, 2004.

[4] G. Allen, T. Goodale, T. Radke, M. Russell, E. Seidel, K. Davis, K.N. Dolkas, N.D. Doulamis, T. Kielmann, A. Merzky, et al. Enabling applications on the grid: a gridlab overview. *International Journal of High Performance Computing Applications*, 17(4):449, 2003.

[5] I. Altintas, A. Birnbaum, K.K. Baldridge, W. Sudholt, M. Miller, C. Amoreira, Y. Potier, and B. Ludaescher. A framework for the design and reuse of grid workflows. *Scientific Applications of Grid Computing*, pages 120–133, 2005.

[6] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Konya, M. Mambelli, JM Schopf, M. Viljoen, and A. Wilson. GLUE Schema Specification-Version 1.2. *OGF GLUE-WG*, 2007.

[7] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, et al. The NAS parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63, 1991.

[8] M. Baker, R. Buyya, and D. Laforenza. The Grid: International efforts in global computing. In *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*. Citeseer, 2000.

[9] B.C. Barish and R. Weiss. LIGO and the detection of gravitational waves. *Physics Today*, 52:44–50, 1999.

[10] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, et al. New grid scheduling and rescheduling methods in the GrADS project. *International Journal of Parallel Programming*, 33(2):209–229, 2005.

[11] F. Berman, G. Fox, and A.J.G. Hey. *Grid computing: making the global infrastructure a reality*. John Wiley & Sons Inc, 2003.

[12] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, et al. Adaptive computing on the grid using AppLeS. *Parallel and Distributed Systems, IEEE Transactions on*, 14(4):369–382, 2003.

[13] GB Berriman, JC Good, AC Laity, A. Bergou, J. Jacob, DS Katz, E. Deelman, C. Kesselman, G. Singh, MH Su, et al. Montage: A grid enabled image mosaic service for the national virtual observatory. In *ASTRONOMICAL SOCIETY OF THE PACIFIC CONFERENCE SERIES*, volume 314, pages 593–596. Citeseer, 2004.

[14] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 759–767. IEEE, 2005.

[15] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[16] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[17] J. Cao, S.A. Jarvis, S. Saini, and G.R. Nudd. Gridflow: Workflow management for grid computing. 2003.

[18] T.L. Casavant and J.G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *Software Engineering, IEEE Transactions on*, 14(2):141–154, 2002.

[19] K. Chanchio and X.H. Sun. (R) MpPVM: A Software System for Non-Dedicated Heterogeneous Computing. In *icpp*, page 0215. Published by the IEEE Computer Society, 1996.

[20] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, et al. New Grid scheduling and rescheduling methods in the GrADS project. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 199. IEEE, 2004.

[21] D.D. Corkill. *Hierarchical planning in a distributed environment.* University of Massachusetts, 1979.

[22] H.J. Dail. *A modular framework for adaptive scheduling in grid application development environments.* PhD thesis, Citeseer, 2002.

[23] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Pegasus: Planning for execution in grids. *GriPhyN*, 20:2002, 2002.

[24] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow management in GriPhyN. In *Grid Resource Management*, pages 99–116. Kluwer Academic Publishers, 2004.

[25] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.H. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*, pages 131–140. Springer, 2004.

[26] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda. GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists. In *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pages 225–234. IEEE, 2002.

[27] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: the montage example. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–12. IEEE, 2009.

[28] M.D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali. Cloud computing: Distributed internet computing for it and scientific research. *Internet Computing, IEEE*, 13(5):10–13, 2009.

[29] P.A. Dinda. Online prediction of the running time of tasks. *Cluster Computing*, 5(3):225–236, 2002.

[30] C.N.E. Edition. JXTA v2. 0 Protocols Specification.

[31] T. Fahringer, J. Qin, and S. Hainzer. Specification of grid workflow applications with AGWL: an Abstract Grid Workflow Language. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 676–685. IEEE, 2005.

[32] D. Fernández-Baca. Allocating modules to processors in a distributed system. *Software Engineering, IEEE Transactions on*, 15(11):1427–1436, 2002.

[33] A. Ferrari, T. Rancati, and PR Sala. FLUKA applications in high energy problems: from LHC to ICARUS and atmospheric showers. 97:165–175, 1997.

[34] I. Foster. Internet computing and the emerging grid. *Nature Web Matters*, 7, 2000.

[35] I. Foster and C. Kesselman. *The grid*. Kaufmann, 1999.

[36] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, pages 37–46, 2002.

[37] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200, 2001.

[38] E. Frécon and M. Stenius. DIVE: A scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering*, 5:91, 1998.

[39] J. Frey. Condor DAGMan: Handling inter-job dependencies, 2002.

[40] M. Goldenberg, P. Lu, and J. Schaeffer. TrellisDAG: A system for structured DAG scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 21–43. Springer, 2003.

[41] L. Gong, G. Ellison, and M. Dageforde. *Inside Java 2 platform security: architecture, API design, and implementation*. Addison-Wesley Professional, 2003.

[42] F. Gottschalk, W.M.P. Van Der Aalst, and M.H. Jansen-Vullers. SAP webflow made configurable: unifying workflow templates into a configurable model. In *Proceedings of the 5th international conference on Business process management*, pages 262–270. Springer-Verlag, 2007.

[43] G.E. Graham, D. Evans, and I. Bertram. McRunjob: A high energy physics workflow planner for Grid production processing. *Arxiv preprint cs/0305063*, 2003.

[44] Z. Guan, F. Hernandez, P. Bangalore, J. Gray, A. Skjellum, V. Velusamy, and Y. Liu. Grid-Flow: A Grid-enabled scientific workflow system with a Petri-net-based interface. *Concurrency and Computation: Practice and Experience*, 18(10):1115–1140, 2006.

[45] R. Gupta, V. Sekhri, and A.K. Somani. Compup2p: An architecture for internet computing using peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, pages 1306–1320, 2006.

[46] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. *Grid Computing-GRID 2000*, pages 191–202, 2000.

[47] B. Hayes. Cloud computing. *Communications of the ACM*, 51(7):9–11, 2008.

[48] J. Heflin, Hendler, et al. *Towards the semantic web: knowledge representation in a dynamic, distributed environment.* PhD thesis, Ph. D. Thesis, University of Maryland, College Park. 2001, 2001.

[49] R. Huang, H. Casanova, and A.A. Chien. Using virtual grids to simplify application scheduling. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 10. IEEE, 2006.

[50] S. Hwang and C. Kesselman. Grid workflow: A flexible failure handling framework for the grid. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 126–137. IEEE, 2003.

[51] M. Jeyachandran and K. Gary. WERCCS: A Client-side Workflow Enactment Service Using AJAX. In *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*, pages 1597–1598. IEEE, 2009.

[52] J. Joseph and C. Fellenstein. *Grid computing.* Prentice Hall PTR, 2004.

[53] A. Lanzén and T. Oinn. The Taverna Interaction Service: enabling manual interaction in workflows. *Bioinformatics*, 24(8):1118, 2008.

[54] D. Li and N. Ishii. Scheduling task graphs onto heterogeneous multiprocessors. In *TENCON'94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994*, pages 556–563. IEEE, 2002.

[55] J. LIN and RG Mathieu. Web services computing: advancing software interoperability [J]. *Computer*, 36(10):35–37, 2003.

[56] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104–111. IEEE, 2002.

[57] X. Liu, J. Liu, J. Eker, and E.A. Lee. Heterogeneous modeling and design of control systems. *Software-Enabled Control: Information Technology for Dynamical Systems*, 2002.

[58] M.M. Lopez, E. Heymann, and M.A. Senar. Analysis of dynamic heuristics for workflow scheduling on grid systems. In *Parallel and Distributed Computing, 2006. ISPDC'06. The Fifth International Symposium on*, pages 199–207. IEEE, 2006.

[59] M.M. Lopez, E. Heymann, and M.A. SENAR. Sensitivity Analysis of Workflow Scheduling on Grid Systems. *Scalable Computing: Practice and Experience*, 8(3):301311, 2007.

[60] B. Lud, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[61] T.W. Malone, R.E. Fikes, and M.T. Howard. Enterprise: A market-like task scheduler for distributed computing environments. *Working paper (Sloan School of Management); 1537-84.*, 1983.

[62] N. Mandal, E. Deelman, G. Mehta, M.H. Su, and K. Vahi. Integrating existing scientific workflow systems: the Kepler/Pegasus example. In *Proceedings of the 2nd workshop on Workflows in support of large-scale science*, pages 21–28. ACM, 2007.

[63] G. Martínez, E. Heymann, and M. Senar. Integrating scheduling policies into workflow engines. *Procedia Computer Science*, 1(1):2743–2752, 2010.

[64] G. Martínez, E. Heymann, M.A. Senar, E. Luque, and B.P. Miller. Using schedflow for performance evaluation of workflow applications. In *Workflows in Support of Large-Scale Science (WORKS), 2010 5th Workshop on*, pages 1–8. IEEE.

[65] G. Martínez, M.M. Lopez, E. Heymann, and M. Senar. Shedflow: Sistema integrador de politicas de planificación y gestores de workflow. *XIX Jornadas de Paralelismo*, pages 481–487, 2008.

[66] A. Milano, F. Forti, C. Sala, G. Riccardi, and D. Ghisotti. Transcriptional regulation of furA and katG upon oxidative stress in Mycobacterium smegmatis. *Journal of Bacteriology*, 183(23):6801, 2001.

[67] M. Miller. *Cloud computing: web-based applications that change the way you work and collaborate online*. Que Publishing Company, 2008.

[68] E. Nakamae, K. Harada, T. Ishizaki, and T. Nishita. A montage method: The overlaying of the computer generated images onto a background photograph. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 207–214. ACM, 1986.

[69] S. Nesmachnow and E. Salsano. Evaluación del desempeño computacional del cluster Medusa. *InCo, Facultad de Ingeniería, Universidad de la República, Uruguay*, 2007.

[70] R. Novaes, P. Roisenberg, R. Scheer, C. Northfleet, J. Jornada, and W. Cirne. Non-dedicated distributed environment: A solution for safe and continuous exploitation of idle cycles. In *Proceedings of the Workshop on Adaptive Grid Middleware*, pages 107–115. Citeseer, 2003.

[71] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, and K. Kennedy. Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 119. ACM, 2006.

[72] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, M.R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 2004.

[73] T. Oinn, P. Li, D.B. Kell, C. Goble, A. Goderis, M. Greenwood, D. Hull, R. Stevens, D. Turi, and J. Zhao. Taverna/my Grid: Aligning a Workflow System with the Life Sciences Community. *Workflows for e-Science*, pages 300–319, 2007.

[74] G. Pandurangan, P. Raghavan, and E. Upfal. Using pagerank to characterize web structure. *Internet Mathematics*, 3(1):1–20, 2006.

[75] R. Prodan and T. Fahringer. Dynamic scheduling of scientific workflow applications on the grid: a case study. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 687–694. ACM, 2005.

[76] M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In *Modern Database Systems: The Object Model, Interoperability, and Beyond*. Citeseer, 1995.

[77] R. Sakellariou and H. Zhao. A hybrid heuristic for DAG scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 111. IEEE, 2004.

[78] R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Scientific Programming*, 12(4):253–262, 2004.

[79] A. Sharp and P. McDermott. *Workflow modeling: tools for process improvement and applications development.* Artech House Publishers, 2008.

[80] W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. In *Job Scheduling Strategies for Parallel Processing*, pages 122–142. Springer, 1998.

[81] K. Sowizral, K. Rushforth, and H. Sowizral. *The Java 3D API Specification.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1997.

[82] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya. A toolkit for modelling and simulating data Grids: an extension to GridSim. *Concurrency and Computation: Practice and Experience*, 20(13):1591–1609, 2008.

[83] A. Sulistio, G. Poduval, R. Buyya, and C.K. Tham. On incorporating differentiated levels of network service into GridSim. *Future Generation Computer Systems*, 23(4):606–615, 2007.

[84] A.S. Tanenbaum and M. Van Steen. *Distributed systems.* Citeseer, 2002.

[85] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. 1 Condor© A Distributed Job Scheduler. *Beowulf Cluster Computing with Linux, The MIT*, pages 0–262, 2002.

[86] I. Taylor, S. Majithia, M. Shields, and I. Wang. Triana workflow specification. *GridLab Specification available at: www. gridlab. org/WorkPackages/wp-3/ D*, 3.

[87] I.J. Taylor, E. Deelman, D. Gannon, and M. Shields. *Workflows for e-Science.* Springer-Verlag London Limited, 2007.

[88] C. Team. Dagman (directed acyclic graph manager). *See website at http://www. cs. wisc. edu/condor/dagman.*

[89] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[90] H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, 2002.

[91] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling. Open grid services infrastructure (ogsi) version 1.0. In *Global Grid Forum Draft Recommendation*, volume 6, page 27, 2003.

[92] W. Van der Aalst and K.M. Van Hee. *Workflow management: models, methods, and systems*. The MIT press, 2004.

[93] W.M.P. van der Aalst and S. Jablonski. Dealing with workflow change: identification of issues and solutions. *Computer systems science and engineering*, 15(5):267–276, 2000.

[94] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[95] G. Von Laszewski, K. Amin, M. Hategan, N.J. Zaluzec, S. Hampton, and A. Rossi. GridAnt: A client-controllable grid workflow system. In *37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, pages 5–8. Citeseer.

[96] G. von Laszewski and M. Hategan. Java CoG Kit Karajan/GridAnt Workflow Guide. Technical report, Technical Report, Argonne National Laboratory, Argonne, IL, USA, 2005.

[97] G. Von Laszewski and D. Kodeboyina. A repository service for grid workflow components. 2005.

[98] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review*, 36(SI):255–270, 2002.

[99] M. Wu and X.H. Sun. A general self-adaptive task scheduling system for non-dedicated heterogeneous computing. In *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, pages 354–361. IEEE, 2005.

[100] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3):171–200, 2005.