Estudio de algoritmos de calendarización para flujos de trabajo

Fernando Aguilar Reyes

Tesina para obtener el título de Ingeniero en Computación
Instituto Tecnológico Autónomo de México
Río Hondo #1, Progreso Tizapán, Del. Álvaro Obregón, 01080
México, Distrito Federal

24 de enero de 2014

Índice general

Índice de figuras

Capítulo 1

Introducción

Cada día, los cómputos son más complejos. Hay bancos que procesan millones de transacciones diarias. Las películas de animación requieren vastos tiempos de cómputo. Muchos proyectos de computación científica requieren hacer numerosos cálculos para llegar a resultados pertinentes. ¿Qué tienen en común todas estas aplicaciones? Ya hemos mencionado que toman mucho tiempo calcularse. Entonces, es natural pensar que se puede distribuir el gran trabajo que requieren estos proyectos entre varias computadoras para disminuir el tiempo total de ejecución y de esta forma, tener una solución escalable.

Lograr esta paralelización requiere un esfuerzo por parte del desarrollador de la solución. Aunque estas técnicas de paralelización son muy efectivas, éstas son aplicadas cuando el problema a resolver ha sido bien definido y cuando sólo hay una instancia del problema. Ahora bien, cuando la solución involucra varios pasos que están relacionados entre sí, por ejemplo, que el programa C requiera de la salida del programa A y del programa B para que pueda funcionar. O, cuando estamos definiendo los grandes bloques de solución del problema, hay una cierta secuencia que debemos seguir y, dentro de dicha secuencia, hay algunos pasos que podemos resolver de manera concurrente. Por lo tanto, estamos haciendo un *modelo* de nuestro problema.

Este modelo también es llamado *flujo de trabajo*. De manera muy abstracta, un modelo de trabajo es un conjunto de pasos que modelan la ejecución de un proceso. Este concepto ha sido aplicado en numerosas áreas. En el ámbito de los negocios, se utiliza diagramas dibujados con los bloques y reglas del *Bussiness Process Execution Language* para modelar procesos de negocio.

En el ámbito de las ciencias en computación, los flujos de trabajo son aplicados para modelar problemas que requieran varios pasos para su solución. Algunos flujos de trabajo requieren mucho tiempo tiempo de ejecución para sus pasos, como son el caso de los flujos de trabajo científicos. Otros flujos de trabajo son muy simples pero se requieren que se ejecuten muchos de éstos. De esta forma, es desable distribuir la ejecución de éstos flujos de trabajo entre varias computadoras. Si bien es posible paralelizar algunos pasos de la ejecución de nuestro flujo de trabajo utilizando las técnicas antes mencionadas, hay restricciones de orden que se deben respetar, por lo cual, se debe planear la ejecución del flujo de trabajo.

Hay varias maneras de hacer esta planeación de la ejecución, también llamada *calendari-*

zación. Cabe aclarar que definimos la calendarización como la asignación de recursos a tareas de tal modo que se ejecuten todas las tareas. Con ello, se desea encontrar una forma óptima de hacer esta calendarización, como reducir el tiempo de ejecución total del flujo de trabajo. Sin embargo, con la aparición del cómputo en la nube, es posible ejecutar nuestro flujo de trabajo con otras restricciones, como minimizar el presupuesto necesario para la ejecución del flujo afectando el tiempo de ejecución.

En esta tesina se hace un estudio de los principales algoritmos de calendarización de flujos de trabajo, con énfasis en los algoritimos utilizados en cómputo distribuido, en especial en cómputo en la nube. En el capítulo 2 se habla de un estudio detallado del concepto de flujos de trabajo y su aplicación en computación. El capítulo 3 trata los principales enfoques de cómputo distribuido para ejecutar estos flujos. En el capítulo 4 se hace un estudio de los principales algoritmos de calendarización de los flujos. Finalmente, en el capítulo 6 discutiremos algunas conclusiones sobre el análisis de estos algoritmos.

Capítulo 2

Flujos de trabajo

2.1. Definición y ejemplos

Un flujo de trabajo es un conjunto de pasos que modelan la ejecución de un proceso [6]. En la literatura especializada, los flujos de trabajos también son conocidos como *workflows*. En particular, se estudian a los flujos de trabajo utilizados para vislumbrar la ejecución de un proceso de cómputo. A continuación, se muestran algunos ejemplos de estos flujos de trabajo.

2.1.1. Anotación de proteínas

En el proyecto *e-Protein*, realizado por la Escuela Imperial de Londres, se realizó un flujo de trabajo para la anotación de proteínas. El objetivo del proyecto [13] era la identificación y anotación de partes de proteínas que expliquen su estructura y su función. En la figura 2.1 se muestra el flujo de trabajo desarrollado, donde las cajas representan los programas que son ejecutados para cada paso del proceso de anotación, y las líneas que conectan a las cajas representan las dependencias de datos entre los programas, es decir, si una caja tiene una línea que apunta a ella, significa que dicho programa depende de otro porgrama determinado por el otro extremo de la flecha.

2.1.2. Curvas de amenaza sísmica

Otro notable ejemplo es el proceso para generar curvas de amenaza que describen las probabilidades de que ocurra un temblor en una determinada área. Para elaborar estas curvas, los científicos del Centro de Terremotos del Sureste de California (SCEC por sus siglas en inglés) tienen que realizar una gran cantidad de simulaciones para que sus resultados puedan ser combinados y expresados en la curva de amenaza [5]. El flujo de trabajo para generar la curva de amenaza se muestra en la figura 2.2.

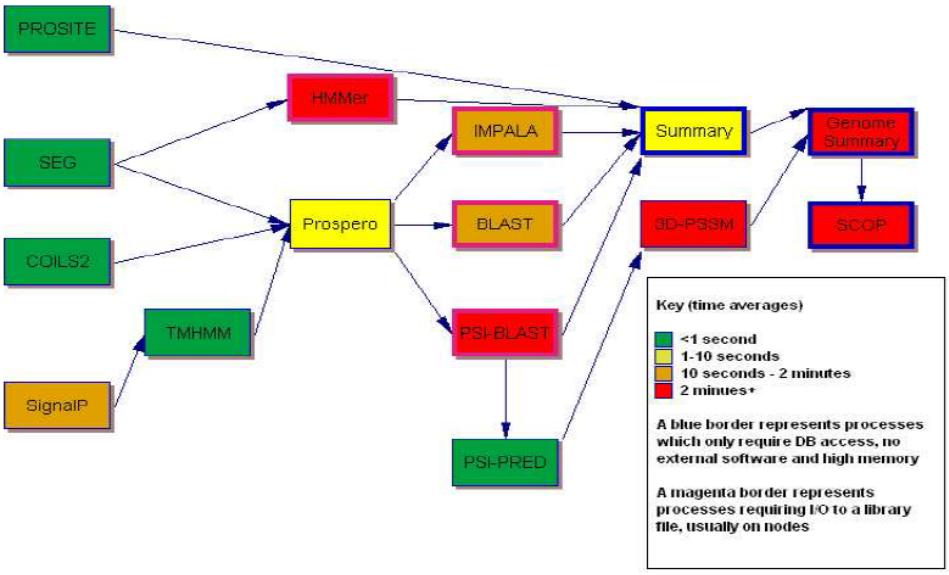

Figura 2.1: Flujo de trabajo para la anotación de proteínas, diseñado en la Escuela Imperial de Londres para el proyecto *e-Protein*.

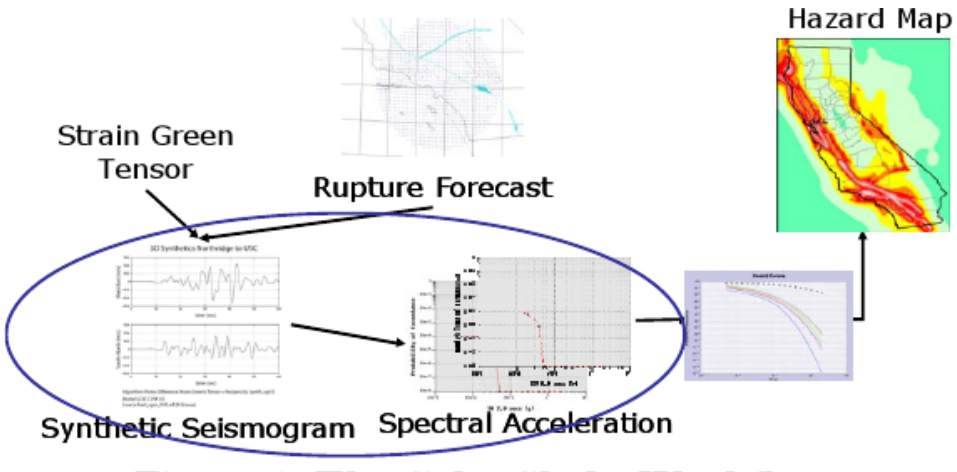

Figura 2.2: Flujo de trabajo para generar curvas de amenaza, elaborado por el SCEC.

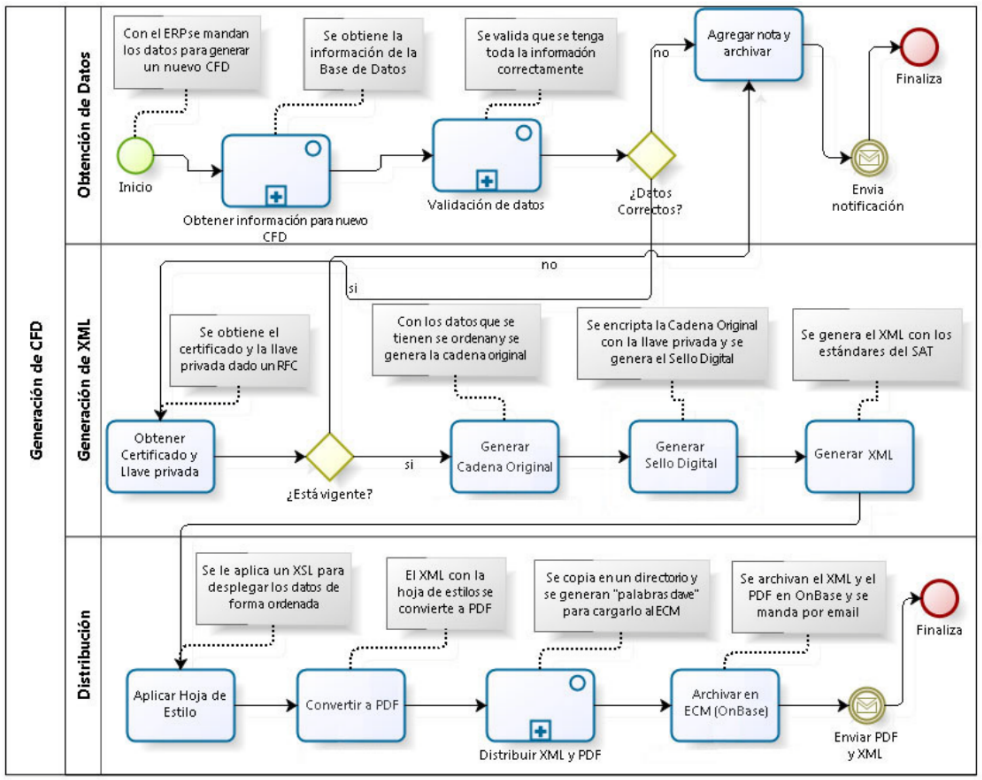

Figura 2.3: Flujo de trabajo para generar facturas electrónicas, utilizado por una empresa de cementos.

2.1.3. Generación de facturas

Recientemente en México, el Sistema de Administración Tributaria emitió los lineamientos para que las operaciones de compra-venta entre personas físicas y morales puedan ser registradas por medio de facturas electrónicas. Para generar estos Comprobantes Fiscales Digitales, las empresas tienen que hacer procesos de validaciones de RFC y encriptar el contenido de la factura con un mecanismo de llave privada. Naturalmente, este proceso de generación de factura requiere de varias actividades. En la figura 2.3 se muestra el flujo de trabajo que utiliza una empresa para generar sus facturas. Este flujo se encuentra documentado en una tesina presentada por Alcerreca [3].

2.2. Estructura de los flujos de trabajo

Como hemos visto en los dos ejemplos anteriores, los flujos de trabajo presentados describen los *pasos* que necesitan para calcular la solución de un problema. En éstos, no se ha hablado sobre detalles para hacer los cómputos necesarios para cada flujo. Tampoco definen las plataformas de cómputo en que se ejecutan estos flujos. Tan sólo definen los grandes

pasos para solucionar el problema y las dependencias que tienen estos pasos.

Ahora, es muy común que estas dependencias estén dictadas por los datos que requiere cada paso para funcionar. Sin embargo, hay situaciones en los que los pasos del flujo no requieren los datos del paso anterior para funcionar, sino que las dependencias están marcadas por el orden temporal que deben seguir estos pasos. Así, se puede notar una ambigüedad en la definición de un flujo de trabajo.

2.2.1. Perspectivas de los flujos de trabajo

El trabajo de Van der Aalst et al. [17] identifica una serie de perspectivas vislumbrados en las definiciones de las especificaciones de un flujo de trabajo en las que se basan los sistemas que administran la ejecución de éstos. Las cuatro perspectivas se enuncian a continuación:

- **Perspectiva de control de flujo.** Describe las relaciones de las actividades (pasos) con estructuras de control, tales como: secuencia, decisión, ejecución de actividades en paralelo y punto de sincronización conjunta[1].
- **Perspectiva de datos.** En ella, los flujos de trabajo describen las entradas y salidas de datos, tanto de ejecución como de control, que se tienen en cada actividad del flujo. También se toma en cuenta los datos locales a cada actividad; es decir, que sólo son necesarios dentro del contexto de ésta.
- **Perspectiva de recursos.** Muestra cuáles son los recursos con los que se cuentan para ejecutar el flujo de trabajo y la forma en que estos recursos se encuentran organizados. Estos recursos pueden ser desde entidades de cómputo hasta roles con responsabilidades específicas cumplidas por actores humanos.
- **Perspectiva operacional.** Aquí se detallan las operaciones elementales necesarias en cada actividad para ejecutar el flujo de trabajo. Estos detalles incluyen las transferencias de datos entre las operaciones y su correspondencia en programas.

Cabe aclarar que estas perspectivas están relacionadas entre sí de modo que el control de flujo es la base en la que descansan las demás perspectivas. Esto es porque la perspectiva de datos requiere que el control de flujo tenga los datos de entrada y salida como prueba de que se cumplieron las pre y post-condiciones de cada actividad, respectivamente; la perspectiva de recursos define con qué se ejecutarán y almacenará los datos del flujo de trabajo; mientras que la perspectiva operacional trata los detalles sobre cómo se utilizan *físicamente* los recursos, los datos y los programas a lo largo del flujo de trabajo.

2.3. Calendarización como control de flujo

El hecho de que la perspectiva de control de flujo sea la base de las demás perspectivas indica que la forma en que controla la ejecución del flujo determina de manera fundamental

[1]Esta estructura de control también es conocida como *join synchronization*

el rendimiento de la ejecución total de una instancia del flujo. Por lo tanto, es de vital importancia encontrar métodos de calendarización que permitan encontrar correspondencias entre recursos y actividades que cumplan con los requisitos dictados en las especificaciones examinadas en la perspectiva del control del flujo y que también maximicen el rendimiento de la ejecución de todo el flujo en general.

En este trabajo, se establecerán las siguientes consideraciones para limitar el estudio de los algoritmos de calendarización, a saber:

- Los flujos de trabajo están representados como *grafos dirigidos acíclicos*, con el objetivo de simplificar el estudio
- Las tareas (o actividades) de los flujos de trabajo son consideradas *atómicas*, i.e., una tarea no puede ejecutarse incompletamente. Algunos sistemas de administración de ejecución de flujos de trabajo tienen mecanismos para lidiar con estos errores, pero el estudio de éstos queda fuera de los límites de este trabajo.
- La información de las *demás perspectivas* se puede requerida, dependiendo si el mecanismo de calendarización lo considere necesario para tomar mejores decisiones.

Capítulo 3

Cómputo distribuido

Los flujos de trabajo requieren de recursos de cómputo para su ejecución. En los ejemplos anteriores se mencionaron aplicaciones científicas que, por su naturaleza, requieren vasto tiempo de ejecución para llevarse a cabo. Por otro lado, también existen aplicaciones de negocio que, si bien son computacionalmente sencillas, se requiere ejecutar una gran cantidad de instancias de estos flujos de trabajo. Por ello, ejecutar cualquiera de estos flujos de trabajo en una sola computadora resulta prohibitivo. Así, se hace uso de varias computadoras para distribuir el esfuerzo para correr estos grandes procesos.

Dependiendo de cómo estén organizados estas computadoras, se definen los enfoques para correr los flujos de trabajo con cómputo distribuido.

3.1. Enfoques de cómputo para flujos de trabajo

Diversos enfoques se han aplicado para distribuir la ejecución de un flujo de trabajo entre varias computadoras. De acuerdo a Buyya et al., los enfoques de cómputo más importantes para los flujos de trabajo son los *clusters*, los *grids* y las *nubes* [4]. A continuación, explicaremos cada uno de los enfoques.

3.1.1. Clusters

Los *clusters* son sistemas distribuidos, paralelos, compuestos de varias computadoras conectadas entre si que funcionan como un único recurso de cómputo [4].

Un ejemplo de un cluster es la instalación de la Universidad Autónoma Metropolitana, campus Iztapalapa, llamada *Aitzaloa*, compuesta por 270 nodos de cómputo, cada uno equipado con dos procesadores Intel Xeon Quad-Core y 16GB en RAM; los nodos están conectados entre sí por medio de switches Ethernet e Infiniband. El cluster también cuenta con un sistema de archivos distribuido basado en Lustre. La capacidad real de cómputo del cluster Aitzaloa es de 18.4 teraFLOPS [2].

El detalle de la topología del cluster se puede apreciar en la figura 3.1, en donde podemos apreciar que los switches son los puntos de conexión entre el nodo maestro, el sistema de

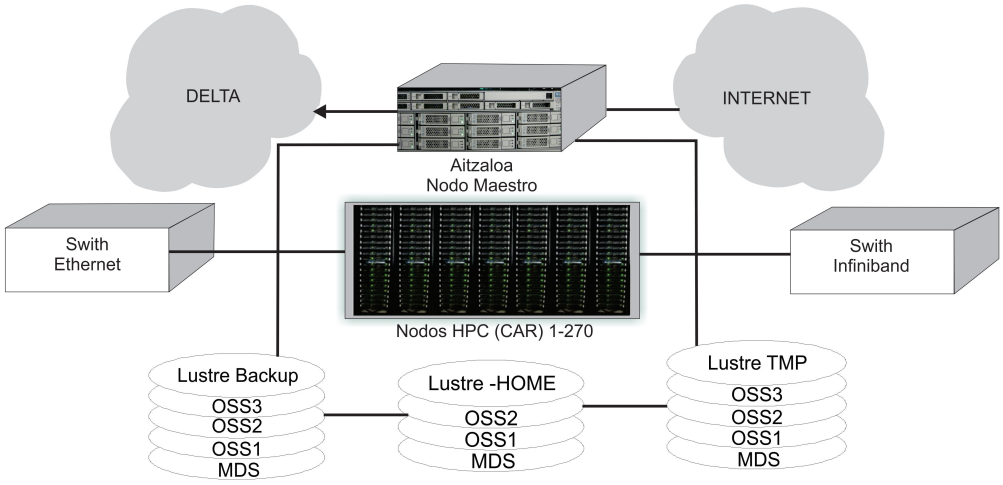

Figura 3.1: Topología del cluster Aitzaloa

almacenamiento distribuido y los nodos de cómputo.

3.1.2. Grids

Los *grids* son sistemas distribuidos, paralelos, compuestos de computadoras autónomas y geográficamente distribuidas que pueden trabajar en conjunto o de manera independiente de acuerdo a los objetivos, políticas y mecanismos de uno o varios administradores del sistema, es decir, un grid puede ser compartido entre varias instituciones [4].

El proyecto *LANCAD*[1] es un buen ejemplo, pues une el cluster *Aitzaloa* de la UAM, el cluster de la UNAM *KamBalam*, y el cluster *Xiuhcoatl* del CINVESTAV por medio de una red de fibra óptica instalada en las estaciones del Sistema de Transporte Colectivo Metro. La suma de la potencias reales de cada *nodo robusto* del grid es de 48.55 teraFLOPS [1].

En la figura 3.2 se muestra los tramos de línea de Metro que cuentan con fibra óptica para conectar cada uno de las supercomputadoras (clusters) de las tres instituciones.

3.1.3. Nubes

Las *nubes* (clouds) son sistemas distribuidos, paralelos, compuestos de computadoras o máquinas virtuales interconectadas que son aprovisionadas para usarse como uno o varios recursos de cómputo, de acuerdo a un contrato de nivel de servicio acordado entre el proveedor de la nube y el cliente [4].

Empresas nuevas y existentes proveen servicios de cómputo en la nube, tales como GoGrid, Rackspace, Amazon, Microsoft, IBM, Oracle, entre otras. La forma en que operan es la siguiente: se paga cierta cantidad por utilizar servicios de cómputo o almacenamiento durante determinado tiempo. Así, los clientes no tienen que invertir grandes cantidades de

[1]Laboratorio Nacional de Cómputo de Alto Rendimiento

Figura 3.2: Detalle de la red de fibra óptica para conectar los nodos robustos del grid del proyecto LANCAD

dinero para contar con una gran infraestructura como en el caso de los los clusters y los grids.

3.2. Comparación de los enfoques

Si bien estos enfoques difieren, principalmente, en la forma en que están organizadas las unidades de cómputo, podemos enumerar algunas observaciones.

Primero, es natural ver que los grids están compuestos de clusters, pero también podría verse a un grid como un gran cluster. Hasta cierto punto este razonamiento parece correcto. Sin embargo, la diferencia importante entre clusters y grids radica en el hecho de que el grid es comúnmente administrado por varios técnicos de varias instituciones que, pueden tener objetivos y necesidades diversas, mientras que un cluster requiere de menos administradores y se asume una mayor flexibilidad para enfrentar los fallos de un cluster.

Segundo, tanto los grids como las nubes son físicamente muy similares: ambos consisten de varios clusteres interconectados y distribuidos geográficamente. Entonces, ¿cuál es la diferencia? Lo que distingue a las nubes de los grids son dos características: (1) en una nube, se utilizan *máquinas virtuales* para distribuir el recurso. El usuario tiene la vista de una parte del grid como si fuera una computadora dedicada exclusivamente para éste. Por otro lado, en los grids, es común que el usuario acceda a él con una cuenta asignada por un administrador, de tal suerte que se tiene una vista de una gran máquina que es compartida entre varios usuarios. Muchos de los administradores de grids en el mundo administran los trabajos encargados por los usuario de tal modo que se ejecuten lo más rápido posible con los recursos disponibles en un tiempo dado. Los usuarios, en algunos casos obtienen acceso al grid y trabajan sus proyectos de manera gratuita o, en caso contrario, pagan una cuota por un tiempo fijo de cómputo. Pero, en el caso de las nubes, se agrega un modelo económico para acceder a los recursos llamado *pay as you go*, en donde el usuario paga una cuota por los recursos utilizados, sin alguna limitación mas que el presupuesto. Asi, el usuario podría pagar una gran cantidad para que su flujo de trabajo se ejecute en el menor tiempo posible o también podría pagar una menor cantidad sacrificando el tiempo total de ejecución del flujo.

Capítulo 4

Calendarización

La calendarización[1] es el proceso de asignación de recursos a tareas, de tal modo que se define un orden de ejecución de las tareas, teniendo lugar diferentes combinaciones de recursos y tareas. En este capítulo se define de la forma más elemental el problema de la calendarización para estudiar sus propiedades y también se explica la relación de este problema fundamental con los flujos de trabajo.

4.1. Definición del problema

De acuerdo a Ullman et al. [16], el problema de la calendarización se define de la siguiente manera:

Definición 1. *El **problema de la calendarización** consiste en:*

1. *Un conjunto de tareas* $S = \{J_1, J_2, \ldots, J_n\}$
2. *Un ordenamiento parcial* $\prec$ *sobre* S
3. *Una función de costo* $W : S \mapsto \mathbb{Z}^+$*, la cual indica el tiempo que tarda en completarse cada una de las tareas en* S
4. *Un número de computadoras (procesadores)* k

Para este caso, se asume que cuando una computadora ejecuta una tarea, ésta es ejecutada completamente. Así, el objetivo es minimizar *el tiempo total de ejecución, denotado por* t_{max}*, respetando el orden parcial definido por* $\prec$*, asignando tareas de* S *a los* k *procesadores .*

[1] También conocida en la literatura especializada como *scheduling*

4.2. La complejidad de calendarizar

Como puede notarse, hay varias maneras de acomodar las k computadoras para que se ejecuten todas las tareas de S. Sin embargo, Ullman et al. han demostrado que este problema pertenece a la categoría NP-completo [16].

Esto signifca que no se ha encontrado un algoritmo que pueda resolver el problema en tiempo polinomial. Entonces, la solución ingenua de probar ordenadamente todas las posibles asignaciones de tareas a computadoras resulta computacionalmente muy caro.

Así, la forma de atacar estos problemas NP-completos es utilizar métodos de aproximación [7] que obtengan soluciones subóptimas o utilizar heurísticas que resuelvan este problema sumando ciertas restricciones.

4.3. Calendarización de flujos de trabajo

Hasta ahora, se ha mencionado el problema básico de calendarización con restricciones. Con ello, se pretende plantear el problema de la calendarización de flujos de trabajo y apoyarse en la definición del problema básico de calendarización para enumerar propiedades sobre este problema.

Para plantear el problema de la calendarización de flujos de trabajo, primero se demostrará que bajo ciertas condiciones, un flujo de trabajo puede ser reducido a un grafo dirigido acíclico haciendo las transformaciones adecuadas. Luego, se utilizará la definición del problema básico de calendarización con restricciones y su semejanza para flujos de trabajo. Finalmente, se hará una descripción de la complejidad de calendarizar flujos de trabajo.

4.3.1. Reducción de flujos de trabajo a grafos dirigidos acíclicos

En el trabajo de de Mair et al. [11] proponen un formato para una representación intermedia de flujos de trabajo, con el fin de transformar una especificación de un flujo de trabajo detallada –escrita en un lenguaje basado en XML llamado AGWL– a una representación intermedia basada en grafos dirigidos acíclicos.

Aunque no existe un consenso general sobre cuál es una definición completa de un flujo de trabajo [17], el lenguaje AGWL es una especificación suficiente para una gran cantidad de flujos de trabajo de ámbito científico, porque con las construcciones if, while, for, parallel, secuence son adecuadas para expresar vastos flujos.

El hecho importante a recalcar es que los grafos dirigidos acíclicos sirven como representación para utilizarlos en los algoritmos de calendarización.

4.3.2. Definición del problema de calendarización de flujos de trabajo

Una vez que se ha establecido a los grafos dirigidos acíclicos como nuestra representación básica de flujos de trabajo, se definirá el problema que conlleva asignar recursos a las tareas

del flujo.

Con el fin de no perder generalidad, tomaremos la definición de Wieczorek-Prodan [18] del problema:

Definición 2. *Un **flujo de trabajo** es un grafo dirigido $w \in \mathcal{W}, w = (\mathcal{V}, \mathcal{E})$, compuesto de un conjunto de nodos $\mathcal{V}$ y un conjunto de aristas $\mathcal{E}$, donde los nodos y las aristas representan tareas $\tau \in \mathcal{T}$ y transferencias de datos $\rho \in \mathcal{D}$.*

Hay que notar que en la definición anterior que la forma en que se relacionan las tareas y las transferencias de datos con los nodos y aristas del grafo está determinada por la representación *concreta* del flujo de trabajo.

Ahora, se definirán los recursos de cómputo en donde se ejecuta el flujo. La definición de Wieczorek-Prodan habla de grids, pero es fácilmente aplicable a otros enfoques de cómputo

Definición 3. *Un **servicio** es una entidad de cómputo que puede ejecutar una tarea $\tau \in \mathcal{T}$. El conjunto de todos los servicios disponibles para ejecutar el flujo de trabajo está denotado por $\mathcal{S}$.*

De este modo, la calendarización es definida como una función:

Definición 4. *La **calendarización de un flujo de trabajo** w es una función $f : \mathcal{T} \mapsto \mathcal{S}$ que asigna servicios a las tareas del flujo. El conjunto que contiene todas las posibles calendarizaciones de w es denotado por $\mathcal{F}$.*

Cada posible calendarización determina un costo de ejecución. A continuación, definiremos el modelo de costo determinado por la utlización de los servicios.

Definición 5. *Un **modelo de costos** es un conjunto de criterios $C = \{c_1, c_2, \ldots, c_n\}$ que determinan las restricciones en las que se debe ejecutar una calendarización, por ejemplo, un límite en el tiempo de ejecución, en el costo monetario o la tolerancia a fallos, entre otros.*

Definición 6. *Para cada criterio $c_i \in C$, existe una **función de costo parcial** $\Theta_i : \mathcal{S} \mapsto \mathbb{R}$, en la que a cada servicio disponible para ejecutar el flujo de trabajo se le asocia un costo por ejecutar dicho servicio con las restricciones dictadas con el criterio c_i.*

Definición 7. *Para cada criterio $c_i \in C$, existe una **función de costo total** $\Delta_i : \mathcal{W} \times \mathcal{F} \mapsto \mathbb{R}$, que asigna un flujo de trabajo w calendarizado por f un costo basado en los costos parciales determinados por los servicios utlizados para ejecutar las tareas del flujo.*

El objetivo del problema de la calendarización de flujos de trabajo es encontrar la calendarización f que minimice las funciones de costo total Δ_i, $1 \leq i \leq n$.

4.3.3. Complejidad computacional de la calendarización de flujos de trabajo

Después de haber enunciado varias definiciones para definir el problema de la calendarización de flujos de trabajo, se hará una analogía con el problema básico de calendarización para poder estudiar su complejidad computacional. La analogía es la siguiente:

1. El conjunto de tareas S equivale a las tareas $\mathcal{T}$ descritas por el flujo de trabajo.

2. El ordenamiento parcial $\prec$ está representado tanto por las dependencias de datos $\mathcal{D}$ y las aristas $\mathcal{E}$ que representan el control de flujo que debe respetarse para ejecutar el flujo.

3. Las k computadoras equvivalen al conjunto de servicios $\mathcal{S}$ disponibles para ejecutar el flujo.

4. La función de costo W tiene una equivalencia implícita. El problema básico asume que conocemos apriori el tiempo de ejecución de una tarea. Sin embargo, es muy frecuente que sólo tengamos una estimación del tiempo de ejecución. Por otro lado, podemos establecer una función auxiliar que relacione las tareas con los servicios. Dicha relación es la función de calendarización f. En efecto, ejecutar una tarea en un servicio genera un costo, determinado por las funciones parciales y totales. Por lo pronto, estableceremos una relación proporcional entre costo y tiempo, con el fin de mostrar que existe una equivalencia entre la función de costo W del problema básico y el modelo de costo de un flujo de trabajo.

Con lo anterior, hemos establecido que el problema de la calendarización de flujos de trabajo es equivalente al problema básico de calendarización. Entonces, es natural inferir que el problema de la calendarización de flujos de trabajo pertenece a la categoría de problemas NP-completo.

Capítulo 5

Algoritmos de calendarización de flujos de trabajo

En el capítulo anterior definimos el problema de calendarizar flujos de trabajo. También se vio que dicho problema pertenece a la categoría de los problemas NP-completos, lo cual significa que la complejidad –o tiempo de ejecución– para resolver este problema no está acotada por una función polinomial. Por ello, diversos algoritmos se han propuesto para atacar, ya sea de manera total o parcial, el objetivo principal de la calendarización: optimizar los costos, definidos como tiempo total de ejecución o un presupuesto.

Ahora, con los enfoques de cómputo propuestos en el capítulo 2, se crean diversos algoritmos para diferentes necesidades. Mientras que en los clusters y los grids se asumen que estos recursos son compartidos, los algoritmos de calendarización para estos recursos asumen que los flujos deben ser ejecutados lo más pronto posible, con el fin de hacer que el recurso esté ocupado la mayor parte del tiempo. Por otro lado, el enfoque de nubes permite diseñador de flujos de trabajo elegir entre ejecutar el flujo con todos los recursos a costa de un elevado presupuesto o, minimizar dicho presupuesto tolerando un tiempo de ejecución mayor al mínimo posible.

5.1. Clasificación de los algoritmos de calenzación

Como han aparecido numerosos algoritmos para calendarizar flujos de trabajo, es natural observar que surjan clasificaciones [15] [21] para tener una mejor idea de los avances logrados en este tema y los puntos clave con los que trabajan los algoritmos.

De acuerdo a Yu et al. [21], se pueden clasificar los algoritmos de calendarización los algoritmos de flujos de trabajo ejecutados en grids en dos grandes niveles: los algoritmos de Mejor Eesfuerzo y los algoritmos de Calidad en el Servicio[1]. Al primer grupo pertenecen aquellos algoritmos que tratan de minimizar el tiempo total de ejecución[2], haciendo uso de todos los recursos disponibles. El segundo grupo, los algoritmos tratan de obtener una

[1]En la literatura especializada se conoce a este término como *Quality of Service*

[2]También conocido como *makespan*

calendarización que cumpla las restricciones especificadas como una medida de calidad, con la posibilidad de elegir soluciones que tomen un tiempo de ejecución subóptimo. A su vez, cada grupo tiene ramas de clasificación que esbozaremos a continuación.

A continuación, mostraremos los algoritmos descritos en la clasificación de Yu et al., con los ajustes en notación necesarios para que coincidan con las definiciones de flujo de trabajo establecidas en el capítulo anterior.

5.2. Algoritmos de Mejor Esfuerzo

Este tipo de algoritmos tratan de minimizar algún criterio, que en muchos casos es el tiempo total de ejecución o *makespan*. Cabe aclarar que para los siguientes algoritmos, se asumirá que el grafo del flujo de trabajo tiene una correspondencia biyectiva entre los nodos $\mathcal{V}$ y las tareas $\mathcal{T}$, es decir, cada nodo del grafo representa una tarea del flujo de trabajo. Las aristas del grafo representan *dependencias entre tareas*.

También, los algoritmos de mejor esfuerzo se pueden dividir en algoritmos guiados por heurísticas y en algoritmos guiados por metaheurísticas. El primer subgrupo debe su nombre a que las soluciones están basadas en ideas que no están comprobadas. El segundo subgrupo contiene a los algoritmos que utilizan algún método para elegir o generar calendarizaciones que se acerquen al óptimo.

Los algoritmos heurísticos se pueden dividir en cuatro grandes tipos: (1) algoritmo inmediato, (2) algoritmos basados en lista, (3) clusterización de tareas y (4) duplicación de tareas. El primer grupo corresponde al algoritmo miope, que sólo asigna taras a recursos conforme se va necesitando. Los algoritmos basados en lista ordenan las tareas de acuerdo a algún criterio (fase de priorización) y seleccionan las tareas de acuerdo a cierto criterio (fase de selección). Los algoritmos de clusterización de tareas crean grupos que, al inicio, crean un grupo para cada tarea; después, con un paso de mezcla se van uniendo grupos hasta que quede un número de grupos igual al número de recursos disponibles; finalmente se ordenan las tareas de cada grupo para ejecutarse en cada grupo. Finalmente, los algoritmos de duplicación de tareas calendarizan una tarea en varios recursos con el fin de reducir el costo por comunicación entre recursos; cada uno de estos algoritmos se caracteriza por la manera en que eligen los algoritmos a calendarizar.

Los algoritmos metaheurísticos son clasificados de acuerdo a la metaheurística que utilizan. Para esta clasificación, vamos a describir tres tipos de heurísticas: (1) algoritmos genéticos, (2) búsqueda aleatoria adaptatipa dirigida –GRASP– y (3) recocido simulado. Los algoritmos genéticos simulan el proceso de selección natural con posibles calendarizaciones e introducen mutaciones para evitar enfocarse en una solución local; con la ayuda de una función de evaluación (*fitness*) se mide la calidad de la calendarización y se eligen aquellas soluciones que resulten mejor evaluadas. El algoritmo GRASP[3] genera aleatoriamente soluciones y con un prodecimiento glotón[4] elige las soluciones óptimas locales. El recocido

[3]GRASP son las siglas en inglés de *Greedy Randomized Adaptive Search Procedure*

[4]Greedy

simulado, como su nombre lo indica, emula el proceso de formación de cristales donde en cada iteración se va creando una solución más óptima.

5.3. Algoritmos de Calidad en el Servicio

En estos algoritmos, se definen un conjunto de restricciones que debe respetar la calendarización. Principalmente, estas restricciones tienen que ver con un presupuesto global limitado y un límite en los tiempos de ejecución total del flujo de trabajo. También es posible definir límites de tiempo de ejecución o límites de prespuesto para cada tarea particular del flujo.

De acuerdo a la forma que trabajan estos algoritmos, pueden dividirse en: restringidos por presupuesto o restringidos for fecha límite. Los primeros son algoritmos que utilizan el presupuesto para cumplir con restricciones de calidad del servicio. Los algoritmos restringidos por fecha límite buscan calendarizaciones que cumplan con fechas proporcionadas. Ambos grupos se subdividen en heurísticos y metaheurísticos. A continuación, hablaremos de estas subdivisiones.

Los algoritmos restringidos por presupuesto se dividen en: (1) heurísticos y (2) metaheurísticos. En la primera división se encuentran los algoritmos que trabajan con una idea base que genera soluciones que no están garantizadas que sean óptimas. En este grupo de algoritmos se encuentran un ejemplo importante: el algoritmo LOSS/GAIN, propuesto por XXXX. Los algoritmos metaheurísticos generan soluciones aleatoriamente y utilizan la información de iteraciones pasadas para refinar las soluciones. En el trabajo de Yu et al. [20] proponen un algoritmo genético cuya función de evaluación valora mejor a las calendarizaciones que cumplan con el presupuesto con un tiempo de ejecución mínimo.

Los algoritmos restringidos por fechas límites también se dividen en: (1) heurísticos y (2) metaheurísticos. En el primer grupo se encuentran el algoritmo BackTracking, propuesto por Menascé et al. [12], y el algoritmo de distribución de fechas límite, propuesto por Yu et al. [22]. En el segundo grupo se encuentra un algoritmo genético basado en el algoritmo del grupo antarior [20], cuya función fitness valora mejor a aquellas soluciones que cumplan con la fecha límite, con un presupuesto mínimo.

5.3.1. Estimación del tiempo

En los algoritmos vistos en esta seccion, se asume que se conoce el tiempo de ejecucion de una tarea. Sin embargo, en la etapa de diseño de un flujo de trabajo, es complicado conocer la duracion de las tareas, debido a que son muchos los factores que influyen el tiempo total de ejecucion del flujo, como el rendimiento de la plataforma de computo, la cantidad de datos a procesar, el tiempo de comunicacion entre nodos de computo, entre otros. Para mitigar este problema, se han hecho algoritmos basados en series de tiempo que estiman el tiempo de ejecucion de cada una de las tareas de un flujo de trabajo. Estos algoritmos utilizan informacion historica de ejecuciones de flujos de trabajo para realizar las predicciones [9].

Capítulo 6

Software para la administración y ejecución de flujos de trabajo

En el capitulo anterior se hizo una clasificacion se extendio la clasificacion de algoritmos de calendarizacion, elaborada por Yu et al. [21]. Sin duda, estos algoritmos han sido probados en simulaciones e implementados en sistemas que administran la ejecucion de los flujos de trabajo. De acuerdo a Y el al. [21], un sistema de administracion de flujos de trabajo[1] se encarga de definir, coordinar y ejecutar los flujos de trabajo en los recursos de computo.

Para este trabajo, clasificaremos los sistemas de administracion de flujos de trabajo de acuerdo a su enfoque de computo, enumerando las siguientes caracteristicas: año de aparicion, proyecto, utilizacion, autores y los algoritmos de calendarizacion utilizados en estos sistemas.

6.1. Software orientado a clusters

6.1.1. HTCondor DAGman

URL: `http://research.cs.wisc.edu/htcondor/`

Desarrollado por la Universidad de Wisconsin en Madison, HTCondor es un sistema para administrar trabajos que necesitan ser ejecutados en entornos de computo distribuido. Este tiene un modulo llamado DAGman que se encarga de calendarizar tareas de un flujo de trabajo, expresadas como grafos dirigidos aciclicos. El algoritmo de calendarizacion utilizado por DAGMan es el algoritmo miope. HTCondor presenta al usuario un unico recurso de computo, formado de varias computadoras interconectadas entre si. Esta es la razon por la que este sistema de administracion de flujos de trabajo cae dentro de la clasificacion de software orientado a clusters.

[1] En la literatura academica, estos sistemas son conocidos como *workflow management sofware systems*

6.2. Software orientado a grids

6.2.1. SwinDew-G

URL: http://www.swinflow.org/swindew/grid/ URL: http://link.springer.com/chapter/10.1007%2F978-1-4614-1933-4_5 Swindew-G es un sistema de administracion de flujos de trabajo cuya caracteristica especial es que trabaja con redes de computadoras P2P. En la primera version de SwinDew, el proceso de calendarizacion es estatico [19], i.e., que en la fase de preparacion de la ejecucion se crea el plan de ejecucion del flujo y se aplica sin ninguna modificacion a lo largo del tiempo. Varios esfuerzos se han hecho para mejorar el mecanismo de calendarizacion, como el algoritmo CTC [8] diseñado especialmente para flujos de trabajo que son intensivos en instancias pero que contienen pocas tareas simples, como los flujos de trabajo utilizados en los bancos y empresas.

Tambien, existen versiones especificas de SwinDew para grids y nubes. En este caso, citamos la version para grids llamada SwinDew-G. La primera version de SwinDew-G aparece en publicaciones aceptadas en el 2006. Finalmente, SwinDew y sus variantes fueron desarrollados en la Universidad de Swinburne, en Australia.

6.2.2. Pegasus

URL: https://pegasus.isi.edu/

Pegasus fue desarrollado en la Universidad del Sur de California. Al igual que SwinDew-G, se puede utilizar Pegasus para trabajar con Grids y con Nubes. Pegassus implementa max-min, min-min y Sufragio como algoritmos de calendarización. Las principales aplicaciones de Pegasus son los flujos de trabajo científicos. Los primeros trabajos publicados que reportan el uso de Pegasus datan del 2003. El módulo encargado de calendarizar el flujo de trabajo es el Mapper.

6.3. Software orientado a nubes

6.3.1. ANEKA

URL: http://www.manjrasoft.com/products.html

Aneka utiliza un mecanismo de negociación mediante agentes que buscan *cerrar* los mejores tratos con los proveedores de los servicios de la nube, que mejor satisfagan los requisitos de calidad en el servicio.

6.3.2. ASKALON

URL: http://www.dps.uibk.ac.at/projects/askalon/

Askalon implementa HEFT para calendarizar flujos que requieran reducir tiempo de ejecución y algoritmos genéticos para trabajar flujos de trabajo con restricciones de calidad en el servicio.

Capítulo 7

Conclusiones

En este trabajo hemos revisado el problema de calendarizar flujos de trabajo. Primero se describió el concepto de flujo de trabajo, ilustrándolo con algunos ejemplos de aplicación. También vimos que la descripción de las tareas y las dependencias entre ellas juegan un papel muy importante a la hora de planear y administrar los recursos asigados a cada tarea del flujo de trabajo.

Más adelante, se estableció que los flujos de trabajo son ejecutados en sistemas de cómputo distribuido, los cuales clasificamos en clusters, grids y nubes, de acuerdo a la forma en trabajan para un objetivo común.

Después se estudió la complejidad de calendarizar flujos de trabajo comparando este problema con otro problema de calendarización más general. Como este problema es NP-completo, se han propuesto varias soluciones heurísticas y metaheurísticas que resuelven el problema bajo ciertas circunstancias.

También, con el objetivo de comprender las diferentes heurísticas propuestas para calendarizar flujos de trabajo, se propuso un modelo para describir de la forma más general posible un flujo de trabajo, basados en el trabajo de Wieckzorek et al..

Con el modelo establecido, se realizó una clasificación de los algoritmos de calendarización, la cual está basada en el trabajo de Yu et al.; la aportación de este trabajo fue dar otro punto de visa sobre la forma de clasificar las metaheurísticas, ya que éstas, dependiendo de cómo son programadas, pueden hacer optimizaciones estilo mejor esfuerzo o buscar soluciones que cumplan con restricciones de calidad en el servicio.

Finalmente, se describieron algunos sistemas de administración de flujos de trabajo y los algoritmos de calendarización que utilizan estos sistemas, clasificandolos de acuerdo a la orientación que tienen los algoritmos para calendarizar. Esto es, si la calendarización está pensada para utilizarse en clusters, grids o nubes. Como se puede notar, en estos sistemas puede verse claramente la unión de las dos teorías sobre enfoques de cómputo distribuidos y algoritmos de calendarización.

Apéndice A

Pseudocódigos

A.1. Algoritmos de menor esfuerzo

A.1.1. Miope

Este es el más simple de todos los algoritmos. Lo único que hace es buscar un recurso disponible que pueda ejecutar la tarea y asignarle dicha tarea. No toma en cuenta otra característica a optimizar. Este algoritmo fue propuesto por Ramamritham et al. [14]. El algoritmo [21] peresentado en estre trabajo se encuentra en la sección A.1.1 del Apéndice.

Input: Un grafo de flujo de trabajo $w = (\mathcal{V}, \mathcal{E})$
Output: Una calendarización f

1: **while** $\exists v \in \mathcal{V}$ no completada **do**
2: $t \leftarrow$ Obtener una tarea lista, no calendarizada, con padres calendarizados;
3: $r \leftarrow$ Obtener un recurso que pueda resolver la tarea en el menor tiempo;
4: Calendarizar t en r, i.e., $f(r) = t$;
5: **end while**

A.1.2. Definiciones para los algoritmos Max-Min y Min-min

Los algoritmos min-min y max-min utilizan las siguientes estimaciones:

- $EET(t, r)$ – **Tiempo estimado de ejecución:** Tiempo que el recurso (servicio) r tomará en ejecutar la tarea t, desde que la tarea es ejecutada en el recurso
- $EAT(t, r)$ – **Tiempo estimado de disponibilidad:** Tiempo en el que el recurso r estará disponible para ejecutar la tarea t
- $FAT(t, r)$ – **Tiempo de archivo disponible:** Tiempo más pronto en que todos los archivos requeridos por la tarea t están disponibles en el recurso r
- $ECT(t, r)$ – **Tiempo estimado de terminación:** Tiempo estimado en cual la tarea

t terminará su ejecución en el recurso r:

$$ECT(t,r) = EET(t,r) + max(EAT(t,r), FAT(t,r))$$

- $MCT(t)$ – **Tiempo mínimo estimado de terminación:** ECT mínimo para la tarea t sobre todos los recursos disponibles, es decir:

$$MCT(t) = \min_{r \in \mathcal{S}} ECT(t,r)$$

A.1.3. Min-Min

El algoritmo min-min está basado en la heurística de terminar las tareas más cortas en el menor tiempo posible. Para ello, hace una estimación del tiempo de ejecución tomando en cuenta el tiempo de preparación de las tareas en los servicios –o recursos– para tener el tiempo y los archivos necesarios disponibles para el recurso en cuestión. El algoritmo fue presentado por Maheswaran et al. [10]. El pseudocódigo está descrito en el listado A.1.3.

Input: Un grafo de flujo de trabajo $w = (\mathcal{V}, \mathcal{E})$
Output: Una calendarización f

```
1: while ∃v ∈ V no completada do
2:     t ← Obtener conjunto de tareas listas, no calendarizadas, con padres calendarizados;
3:     SCHED(tasks);
4: end while
5: procedure SCHED(availTasks)
6:     while ∃t ∈ availTasks no calendarizadas do
7:         for t ∈ availTasks do
8:             res ← Obtener recursos disponibles para t;
9:             for r ∈ res do
10:                Calcular ECT(t, r);
11:            end for
12:            R_T ← arg mín_{r∈res} ECT(t, r);
13:        end for
14:        T ← arg mín_{t∈availTasks} ECT(t, R_T);
15:        Calendarizar T en R_T;
16:        Remover T de availTasks;
17:        Actualizar EAT(R_T);
18:    end while
19: end procedure
```

A.1.4. Max-Min

El algoritmo max-min –también propuesto Maheswaran et al. [10]– es muy similar al algoritmo min-min. La diferencia radica en que éste calendariza tareas cuyo tiempo mínimo de ejecución es el mayor, de tal modo que se ejecutan las tares más *largas*.

El único cambio que se necesita hacer al algoritmo A.1.3 es cambiar la siguiente línea (14):

$$T \leftarrow \arg \min_{t \in availTasks} ECT(t, r);$$

por

$$T \leftarrow \arg \max_{t \in availTasks} ECT(t, r);$$

Así, el algoritmo modificado puede verse en el pseudocódigo A.1.4.

Input: Un grafo de flujo de trabajo $w = (\mathcal{V}, \mathcal{E})$
Output: Una calendarización f

```
1: while ∃v ∈ V no completada do
2:     t ← Obtener conjunto de tareas listas, no calendarizadas, con padres calendarizados;
3:     SCHED(tasks);
4: end while
5: procedure SCHED(availTasks)
6:     while ∃t ∈ availTasks no calendarizadas do
7:         for t ∈ availTasks do
8:             res ← Obtener recursos disponibles para t;
9:             for r ∈ res do
10:                Calcular ECT(t, r);
11:            end for
12:            R_T ← arg mín_{r∈res} ECT(t, r);
13:        end for
14:        T ← arg máx_{t∈availTasks} ECT(t, R_T);
15:        Calendarizar T en R_T;
16:        Remover T de availTasks;
17:        Actualizar EAT(R_T);
18:    end while
19: end procedure
```

A.1.5. Sufragio

El algoritmo Sufragio[1] [10] es una variación del algoritmo Min-min, el cual considera el valor del sufragio para hacer la calendarización. Dicho valor es la diferencia entre el menor tiempo de ejecución para una tarea t sobre un conjunto de recursos disponibles y el segundo menor. Se calendariza a la tarea que tenga el valor del sufragio más alto, por el hecho de que las tareas que son muy sensibles a los cambios de los recursos deben ser calendarizadas primero. El algoritmo se encuentra en la sección A.1.5 del Apéndice.

Input: Un grafo de flujo de trabajo $w = (\mathcal{V}, \mathcal{E})$
Output: Una calendarización f

```
1: while ∃v ∈ V no completada do
2:     t ← Obtener conjunto de tareas listas, no calendarizadas, con padres calendarizados;
```

[1] También conocido como *Sufferage*

```
    SCHED(tasks);
end while
procedure SCHED(availTasks)
    while ∃t ∈ availTasks no calendarizadas do
        for t ∈ availTasks do
            res ← Obtener recursos disponibles para t;
            for r ∈ res do
                Calcular ECT(t, r);
            end for
            R_t^1 ← arg mín_{r∈res} ECT(t, r);
            R_t^2 ← arg mín_{r∈res, r≠R_t^1} ECT(t, r);
            suf_t ← ECT(t, R_t^2) − ECT(t, R_t^1);
        end for
        T ← arg máx_{t∈availTasks} suf_t;
        Calendarizar T en R_T^1;
        Remover T de availTasks;
        Actualizar EAT(R_T);
    end while
end procedure
```

A.1.6. HEFT

El algoritmo HEFT[2] fue propuesto por Topcuoglu et al. [15]

Input: Un grafo de flujo de trabajo $w = (\mathcal{V}, \mathcal{E})$
Output: Una calendarización f

```
Calcular Tiempo de Ejecución Promedio (1) para cada tarea v ∈ V;
Calcular Tiempo de Transferencia de Datos Promedio (2) entre tareas y sus sucesores;
Calcular Rango para cada tarea, de acuerdo a (3) (4);
Ordenar las tareas por Rango, en orden decreciente en una lista Q;
while Q ≠ ∅ do
    t ← Remover la primera tarea de Q;
    r ← Encontrar un recurso que pueda ejecutar r en el menor tiempo;
    Calendarizar t en r;
end while
```

A.1.7. Híbrido

Input: Un grafo de flujo de trabajo $w = (\mathcal{V}, \mathcal{E})$
Output: Una calendarización f

```
Calcular Peso de cada tarea y arista, de acuerdo a (1)(2);
Calcular Rango para cada tarea, de acuerdo a (3) (4);
```

[2]Heterogeneous Earliest-Finish Time

3: Ordenar las tareas por *Rango*, en orden decreciente en una lista Q;
4: $i \leftarrow 0$
5: Crear un grupo G_i;
6: **while** $Q \neq \varnothing$ **do**
7: $t \leftarrow$ Remover la primera tarea de Q;
8: **if** t tiene una dependencia con una tarea en G_i **then**
9: $i \leftarrow i + 1$;
10: Crear un grupo G_i;
11: **end if**
12: Agregar t a G_i;
13: **end while**
14: $j \leftarrow 0$;
15: **while** $j \leq i$ **do**
16: Calendarizar tareas en G_i usando un algoritmo *batch*;
17: $j \leftarrow j + 1$;
18: **end while**

A.1.8. TANH

Input: Un grafo de flujo de trabajo $w = (\mathcal{V}, \mathcal{E})$
Output: Una calendarización f
1: Calcular *Parámetros* para cada nodo tarea;
2: Agrupar tareas del flujo de trabajo;
3: **if** Número de clusters $\geq$ Número de recursos disponibles **then**
4: Reducir el Número de clusters al Número de recursos disponibles;
5: **else**
6: Ejecutar duplicación de tareas;
7: **end if**

A.2. Metaheurísticos

A.2.1. GRASP

Input: Un grafo de flujo de trabajo $w = (\mathcal{V}, \mathcal{E})$
Output: Una calendarización f
1: **while** Criterio de terminación no satisfactorio **do**
2: $sched \leftarrow$ createSchedule(w);
3: **if** *sched* es mejor que *bestSched* **then**
4: $bestSched \leftarrow sched$;
5: **end if**
6: **end while**
7: **procedure** createSchedule($workflow$)

```
    solution ← constructSolution(workflow);
    nSolution ← localSearch(solution);
    if nSolution es mejor que solution then
        return nSolution;
    end if
    return solution;
end procedure
procedure constructSolution(workflow)
    while calendarización no completada do
        T ← Obtener tareas listas sin asignar;
        Crear RCL para cada t ∈ T;
        subSolution ← Seleccionar un recurso aleatoriamente para cada t ∈ T de su RCL;
        solution ← solution ∪ subSolution;
        Actualizar información para futuros RCL;
    end while
    return solution;
end procedure
procedure localSearch(solution)
    nSolution ← Encontar una solución local óptima;
    return nSolution;
end procedure
```

Bibliografía

[1] Cluster Híbrido de Supercómputo — Xiuhcoatl — Cinvestav. `http://clusterhibrido.cinvestav.mx/`. Consultado el 10 de noviembre de 2013.

[2] Laboratorio de Supercómputo y Visualización en Paralelo. `http://supercomputo.izt.uam.mx/infraestructura/aitzaloa.php`. Consultado el 10 de noviembre de 2013.

[3] Sebastian Alcerreca Alcocer. Emisión de Comprobantes Fiscales Digitales (CFD) - Desarrollo del sistema de emisión, distribución y almacenamiento de CFD de la corporación Montecito. Tesis de licenciatura, Instituto Tecnológico Autónomo de México, 2013.

[4] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona 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.

[5] Ewa Deelman, Scott Callaghan, Edward Field, Hunter Francoeur, Robert Graves, Nitin Gupta, Vipin Gupta, Thomas H Jordan, Carl Kesselman, Philip Maechling, et al. Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example. In *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, pages 14–14. IEEE, 2006.

[6] J Octavio Gutierrez-Garcia and Kwang Mong Sim. Agent-based cloud workflow execution. *Integrated Computer-Aided Engineering*, 19(1):39–56, 2012.

[7] Charles E Leiserson, Ronald L Rivest, Clifford Stein, and Thomas H Cormen. *Introduction to algorithms*. The MIT press, 2001.

[8] Ke Liu, Hai Jin, Jinjun Chen, Xiao Liu, Dong Yuan, and Yun Yang. A compromised-time-cost scheduling algorithm in swindew-c for instance-intensive cost-constrained workflows on a cloud computing platform. *International Journal of High Performance Computing Applications*, 24(4):445–456, 2010.

[9] Xiao Liu, Zhiwei Ni, Dong Yuan, Yuanchun Jiang, Zhangjun Wu, Jinjun Chen, and Yun Yang. A novel statistical time-series pattern based interval forecasting strategy for activity durations in workflow systems. *Journal of Systems and Software*, 84(3):354–376, 2011.

[10] Muthucumaru Maheswaran, Shoukat Ali, HJ Siegal, Debra Hensgen, and Richard F Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 30–44. IEEE, 1999.

[11] Michael Mair, Jun Qin, Marek Wieczorek, and Thomas Fahringer. Workflow conversion and processing in the ASKALON grid environment. In *2nd Austrian Grid Symposium*, pages 67–80. Citeseer, 2007.

[12] Daniel A Menasce and Emiliano Casalicchio. A framework for resource allocation in grid computing. In *MASCOTS*, pages 259–267, 2004.

[13] Angela O'Brien, Steven Newhouse, and John Darlington. Mapping of scientific workflow within the e-protein project to distributed resources. In *UK e-Science All Hands Meeting*, pages 404–409, 2004.

[14] Krithi Ramamritham, John A. Stankovic, and P-F Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *Parallel and Distributed Systems, IEEE Transactions on*, 1(2):184–194, 1990.

[15] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260–274, 2002.

[16] Jeffrey D. Ullman. NP–complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[17] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.

[18] Marek Wieczorek, Andreas Hoheisel, and Radu Prodan. Taxonomies of the multi-criteria grid workflow scheduling problem. In *Grid Middleware and Services*, pages 237–264. Springer, 2008.

[19] Yun Yang, Ke Liu, Jinjun Chen, Joel Lignier, and Hai Jin. Peer-to-peer based grid workflow runtime environment of swindew-g. In *e-Science and Grid Computing, IEEE International Conference on*, pages 51–58. IEEE, 2007.

[20] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14(3):217–230, 2006.

[21] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. Workflow scheduling algorithms for grid computing. In *Metaheuristics for scheduling in distributed computing environments*, pages 173–214. Springer, 2008.

[22] Jia Yu, Rajkumar Buyya, and Chen Khong Tham. Cost-based scheduling of scientific workflow applications on utility grids. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8–pp. IEEE, 2005.