Desarrollo de un *workflow* genérico para el modelado de problemas de barrido paramétrico en sistemas distribuidos

Sebastián Reyes Ávila

Directores:

Rosa M. Badia Sala

Alfonso Niño Ramos

Tesis presentada para obtener el título de Doctor por la

UNIVERSITAT POLITÈCNICA DE CATALUNYA

DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES (DAC)

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

Barcelona, 2012

A María

Abstract

This work presents the development and experimental validation of a generic workflow model applicable to any parameter sweep problem: the Parameter Sweep Scientific Workflow (PSWF) model. As part of it, a model for the monitoring and management of scientific workflows on distributed systems is developed. This model, Star Superscalar Status (SsTAT), is applicable to the StarSs programming model family. PSWF and SsTAT can be used by the scientific community as a reference for solving problems using the parameter sweep strategy.

As an integral part of the work, the treatment of the parameter sweep problem is formalized. This is achieved by developing a general solution based on the PSNSS (*Parameter Sweep Nested Summation Symbol*) algorithm, using both the original sequential and a concurrent approach. Both versions are implemented and validated, showing its applicability to all automatable PSWF lifecycle phases. Load testing shows that large-scale parameter sweep problems can efficiently be addressed with the proposed approach.

In addition, the SsTAT monitoring and management generic model is instantiated for a Grid environment. Thus, an operational implementation of SsTAT based on GRIDSs, GSTAT (GRID Superscalar Status), is developed. A series of tests performed on an actual heterogeneous Grid of computers shows that GSTAT can appropriately develop their functionality even in an environment so demanding as that.

As a practical case, the model proposed here is applied to determining the molecular potential energy hypersurfaces. For this purpose, a specific instance of the workflow, called PSHYP (Parameter Sweep Hypersurfaces), is created.

Resumen

En este trabajo se presenta el desarrollo y validación experimental de un modelo de *workflow* genérico, aplicable a cualquier problema de barrido de parámetros, denominado *Parameter Sweep Scientific Workflow* (PSWF). Asimismo, se diseña y prueba un modelo de monitorización y gestión de *workflows* científicos, en sistemas distribuidos, designado como SsTAT (Star *Superscalar Status*) que es aplicable a la familia de modelos de programación Star *Superscalar* (StarSs). Los modelos PSWF y SsTAT pueden ser utilizados por la comunidad científica como referencia a la hora de resolver problemas mediante la estrategia de barrido de parámetros.

Como parte integral del trabajo se formaliza el tratamiento del problema del barrido de parámetros, desarrollándose una solución general concretada en el algoritmo PSNSS (*Parameter Sweep Nested Summation Symbol*) en su versión secuencial y concurrente. Ambas versiones se implementan y validan, mostrándose su aplicabilidad a todas las fases automatizables del ciclo de vida PSWF. Mediante la realización de varias pruebas de carga se comprueba que el tratamiento de problemas de barrido de parámetros de gran envergadura puede abordarse eficientemente con la aproximación propuesta.

A su vez, el modelo genérico de monitorización y gestión SsTAT se particulariza para un entorno *Grid*, generándose una implementación operativa del mismo, basada en GRIDSs, denominada GSTAT (GRID *Superscalar Status*). La realización de una serie de pruebas sobre un *Grid* real de computadores heterogéneo muestra que GSTAT desarrolla apropiadamente sus funciones incluso en un entorno tan exigente como este.

Como caso práctico, se aplica el modelo aquí propuesto a la obtención de la hipersuperficie de energía potencial molecular generando a tal efecto un *workflow* específico denominado PSHYP (*Parameter Sweep Hypersurfaces*).

Agradecimientos

En primer lugar quiero expresar mi más sincero agradecimiento a mis directores de tesis, Alfonso Niño y Rosa M. Badia, así como a Camelia Muñoz por el esfuerzo y dedicación empleados en mi orientación, supervisión, motivación y formación como investigador.

Quiero también expresar mi agradecimiento a todas aquellas personas e instituciones que han hecho posible que este trabajo de investigación sea una realidad. De manera muy especial, quiero mencionar a todas las personas que han formado parte de mi grupo de investigación SciCom (*Scientific Computing Group*; anterior QCyCAR).

Quiero mostrar mi agradecimiento al *Barcelona Supercomputing Center–Centro Nacional de Supercomputación* (BSC-CNS) por facilitarme la realización de una estancia, en el mes de Julio del año 2006, y por permitirme el acceso y utilización de todos los recursos necesarios para llevar a cabo las pruebas empíricas que dan refrendo a esta investigación. En particular, quiero mostrar mi agradecimiento a Jose María Cela y a Raúl Sirvent por su colaboración y atención en todo momento.

También quiero mostrar mi agradecimiento al Laboratorio de Química Teórica de la Universidad Autónoma de Puebla en México (LQT-BUAP) y, de modo muy especial, a Francisco Meléndez; al departamento de Astrofísica Molecular e Infrarrojo (DAMIR) del Consejo Superior de Investigaciones Científicas (CSIC) y, particularmente, a María Luisa Senent y al Centro de Supercomputación de Galicia (CESGA) y, en particular, a Carlos Fernández por la cesión de recursos que han hecho posible la obtención de algunos de los resultados que se muestran en este trabajo.

Por último, quiero mostrar mi más profundo agradecimiento a mi familia y amigos por su apoyo incondicional a lo largo de la realización de este trabajo y, lo que es más importante, durante toda mi vida.

Índice

Índice de Figuras

Acrónimos

ADL	*Architecture Description Language*
APGAS	*Asynchronous Partitioned Global Address Space Model*
BPEL	*Business Process Execution Language*
BSC-CNS	*Barcelona Supercomputing Center – Centro Nacional de Supercomputación*
CellSs	*Cell Superscalar*
CG	*Coarse Grained*
ClusterSs	*Cluster Superscalar*
COMPSs	*COMP Superscalar*
C2VO	*Computational Chemistry Virtual Organization*
DC	*Data Catalog*
EGEE	*Enabling Grid for E-sciencE*
ESM	*Earth System Models*
FID	*File Identifier*
GAT	*Grid Application Toolkit*
GCM	*Grid Component Model*
GNDL	*Generalized Nested Do Loop*
GPUSs	*GPU Superscalar*
GRIDSs	*GRID Superscalar*
GMA	*Grid Monitoring Architecture*
GridRM	*Resource Monitoring for the Grid*
GSTAT	GRID *Superscalar Status*
IC2D	*Interactive Control and Debugging for Distribution*
LHC	*Large Hadron Collider*
LKD	*Logical Kronecker Delta*
MDS	*Monitoring and Discovery Service*
MPI	*Message Passing Interface*
NetLogger	*Network Application Logger Toolkit*

NSS	*Nested Summation Symbol*
NWS	*Network Weather Service*
OGF	*Open Grid Forum*
PS	*Parameter Sweep*
PSE	*Parameter Sweep Experiments*
PSNSS	*Parameter Sweep Nested Summation Symbol*
PSWF	*Parameter Sweep Scientific Workflow*
RES	*Red Española de Supercomputación*
RTG	*Round Trip Time Grid*
SHIWA	*SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs*
SMP	*Symmetric MultiProcessing*
SMPSs	*SMP Superscalar*
SNMP	*Simple Network Management Protocol*
SOAP	*Simple Object Access Protocol*
SWF	*Scientific Workflow*
SWFMS	*Scientific Workflow Management System*
SWS	*Scientific Workflow System*
StarSs	*Star Superscalar*
SsTAT	*Star Superscalar Status*
WC	*Workflow Collector*
WE	*Workflow Execution*
WF	*Workflow*
WFR	*Repositorio de Workflows*
WfMC	*Workflow Management Coalition*
WG	*Workflow Generator*
WFT	*Workflow Tools*
WS	*Web Services*
WSDL	*Web Services Description Language*
XML	*eXtensible Markup Language*

1 Introducción

En este capítulo se describe la motivación para la realización de este trabajo, los objetivos y contribuciones del mismo, así como su estructura y organización.

1.1 Motivación

En las últimas dos décadas, se ha producido una revolución en el proceso de desarrollo científico y tecnológico. La computación se ha convertido en la "tercera vía" del proceso científico junto con la teórica y la experimental. Un escenario típico en el proceso científico es la transferencia de información a un sistema de computación para su análisis o para la realización de una simulación. En la última década han surgido los sistemas de flujo de trabajo científico, *Scientific Workflow Systems* (SWS) [1], con el objetivo de automatizar y facilitar este proceso. La idea es permitir que los científicos se centren en los detalles de su investigación (teórica y/o experimental) abstrayéndose, en lo posible, de los pormenores computacionales. De esta forma, se acelera el desarrollo de nuevo conocimiento en sus respectivos campos.

Muchos problemas científicos y de ingeniería pueden formularse mediante modelos computacionales con una gran precisión. El cambio de las entradas del problema permite explorar una gran variedad de escenarios de diseño aportando una imagen de cómo se comporta el sistema. Este proceso puede realizarse mediante un barrido sistemático del espacio de definición de los diferentes parámetros que especifican el estado del modelo. A pesar de que el modelo puede ser costoso computacionalmente, la ejecución paralela en un sistema distribuido puede acelerar considerablemente el proceso y, así, permitir el estudio de problemas complejos.

En este contexto, resulta útil pensar en un determinado modelo computacional como una función que acepta un conjunto de parámetros de entrada y produce un conjunto de resultados. Los parámetros de entrada suelen ser números enteros, reales o cadenas de caracteres. Un barrido de parámetros completo implica todas las combinaciones de los parámetros y permite la exploración de todo el espacio de diseño dentro de una resolución finita. Este enfoque se ha utilizado de manera muy eficaz en distintas áreas de la ciencia y hay coincidencia en la comunidad científica en denominarlo Experimento de Barrido de Parámetros (*Parameter Sweep Experiment*, PSE) [2].

A continuación, se presentan algunos ejemplos ilustrativos, ordenados por áreas de aplicación, de la técnica de barrido de parámetros comentada previamente. No se trata de un análisis exhaustivo de todos los casos existentes, sino de los más representativos encontrados en la bibliografía.

- **Bioinformática**

  Búsquedas de secuencias en genomas, proteínas y alineamiento de secuencias múltiples, donde se puede aplicar una aproximación de grano grueso distribuyendo las secuencias buscadas entre los procesadores [3], [4], [5]. Ejemplos adicionales son, la búsqueda de genes ortólogos y la construcción de árboles filogenéticos [5], las simulaciones de plegado de proteínas [6], o el diseño de nuevos fármacos por simulación de acoplamiento molecular usando técnicas de alta productividad [7].

- **Ciencias de la Tierra**

  Un problema típico es la simulación de la evolución del clima de la Tierra a largo plazo. Para ello, se usan los Modelos de Sistema Terrestre (*Earth System Models*, ESM) [8].

- **Física de Altas Energías**

  Un ejemplo actual es el análisis de eventos físicos de altas energías [9], [10]. Esta es una actividad liderada por los proyectos LHC (*Large Hadron Collider*) [10] y EGEE (*Enabling Grid for E-sciencE*) [11].

- **Química Teórica**

  En ciencias moleculares, se dispone de ejemplos que utilizan técnicas híbridas de dinámica molecular y Monte Carlo, lo que implica el afinado de varios parámetros [12], [13], [14], [15]. Otro ejemplo, es la obtención de la hipersuperficie de energía potencial molecular [16]. También se encuentran simulaciones, usando el simulador Monte Carlo de microfisiología celular MCell, que permiten representar reacciones químicas o la difusión de moléculas en espacios 3-D complejos [17].

- **Otros**

  Sirvan como ejemplos, la optimización de parámetros en modelos de programación genética [2]; el análisis de circuitos analógicos en ingeniería eléctrica [18]; el modelado de influencias sociales o cambios culturales basados en agentes [19]; las simulaciones de redes [20], [21]; el análisis para la calibración de equipos de radiación [22]; las búsquedas de inteligencia extraterrestre [23], o las técnicas de tomografía para reconstruir estructuras tridimensionales de objetos a partir de

una serie de proyecciones en dos dimensiones [24]. También, se encuentran problemas como la simulación computacional en dinámica de fluidos, que se ha aplicado en el estudio del comportamiento de una lanzadera reutilizable de la NASA [25].

Del análisis anterior se deduce que el barrido de parámetros es una estrategia ampliamente utilizada, por la comunidad científica, en la resolución de problemas complejos.

1.2 Objetivos

El objetivo principal de este trabajo es el análisis y desarrollo de un modelo genérico de flujo de trabajo, *workflow* (WF), que permita abordar, en cualquier área científico-tecnológica, la resolución de problemas de barrido paramétrico en sistemas distribuidos. Es decir, se trata de desarrollar un *Parameter Sweep Scientific Workflow* (PSWF).

Para alcanzar el objetivo principal se establecen los siguientes objetivos parciales:

a) Desarrollo de un algoritmo que permita, de forma automática, generar el espacio de definición de los diferentes parámetros que especifican el estado de un problema, almacenar los correspondientes valores del conjunto de parámetros, localizando y recolectando los resultados de las simulaciones para cada conjunto de parámetros. La idea es presentar una solución general que no dependa ni del número de parámetros ni del rango de valores de los mismos, y que sea aplicable a cualquier tipo de problema. La siguiente contribución está directamente relacionada con este objetivo:
    1) J. Díaz, S. Reyes, R.M. Badia, A. Niño and C. Muñoz-Caro, "A General Model for the Generation and Scheduling of Parameter Sweep Experiments in Computational Grid Environments", *Procedia Computer Science*, 1, 565–572, 2010. Presentado en ICCS 2010 (Core A).
b) Diseño de la arquitectura propuesta para el PSWF.
c) Validación experimental de la viabilidad del modelo PSWF. A tal efecto, se aplicará el modelo a la obtención automática de hipersuperficies de energía potencial molecular. Este problema es de gran interés tanto para la simulación e interpretación de espectros rovibracionales como para la simulación por dinámica molecular de reacciones químicas. Ambos aspectos son de especial importancia para la identificación y caracterización (el primero) y la evolución (el segundo) de especies de interés astrofísico y astrobiológico. La consecución de este objetivo ha dado lugar a las siguientes contribuciones:
    2) S. Reyes, C. Muñoz-Caro, A. Niño, R. M. Badia, J. M. Cela. "Performance of computationally intensive parameter sweep applications on Internet-based Grids of computers: the mapping of molecular potential energy hypersurfaces". *Concurrency and Computation: Practice and Experience*, 19, 463-481, 2007.

3) S. Reyes, C. Muñoz-Caro, A. Niño, R. M. Badia, and J. M. Cela. "Development and performance of a grid computing approach to the massive exploration of potential energy hypersurfaces for spectroscopic studies". The Nineteenth Colloquium on High Resolution Molecular Spectroscopy, Salamanca, Spain, September 2005. Book of abstracts, pp. 463-464.

d) Diseño del sistema de monitorización y gestión de tareas SsTAT (*Star Superscalar Status*) para sistemas distribuidos aplicado al modelo de programación paralela *Star Superscalar* (StarSs); implementación, integración y validación mediante el entorno de programación GRID *Superscalar* (GRIDSs) [26], miembro de la familia StarSs. A continuación, se incluyen las contribuciones que se corresponden con este objetivo:

4) S. Reyes, C. Muñoz-Caro, A. Niño, R. M. Badia, J. M. Cela. "Performance of computationally intensive parameter sweep applications on Internet-based Grids of computers: the mapping of molecular potential energy hypersurfaces". *Concurrency and Computation: Practice and Experience*, 19, 463-481, 2007.

5) S. Reyes, C. Muñoz-Caro, A. Niño, R. M. Badia, and J. M. Cela. "Development and performance of a grid computing approach to the massive exploration of potential energy hypersurfaces for spectroscopic studies". The Nineteenth Colloquium on High Resolution Molecular Spectroscopy, Salamanca, Spain, September 2005. Book of abstracts, pp. 463-464.

1.3 Impacto del trabajo

Aparte de las cinco contribuciones específicas presentadas en el apartado anterior se han producido una serie de trabajos tangencialmente relacionados con aspectos no originales de los objetivos b) a d).

1) S. Reyes, C. Muñoz-Caro, A. Niño. "Aplicaciones Grid en una Organización Virtual de Química Computacional". Publicación de la Red Nacional de I+D. Boletín de RedIRIS, 80: 47-51, Abril 2007.
2) S. Reyes, J. Díaz, S. Díaz, A. Niño and C. Muñoz-Caro. "Organization of, and Experiences with, a Spain-Mexico Internet-Based Grid of Computers". 1st Iberian Grid Infrastructure Conference (IBERGRID). Santiago de Compostela (Spain), 2007.
3) S. Díaz, S. Reyes, J. Díaz, A. Niño, y C. Muñoz-Caro. "Diseño y comportamiento de un Grid de Computadores basado en Internet". Proceedings of XVIII JORNADAS DE PARALELISMO 529-535 (2007).
4) S. Reyes, C. Muñoz-Caro, A. Niño, "Vantaxes do uso da tecnoloxía Grid no campo Físico-Molecular", Dixitos, Noviembre 2006.
5) S. Reyes, C. Muñoz-Caro, A. Niño. "Desarrollo de aplicaciones Físico-Moleculares en entornos Grid". XVII Jornadas de Paralelismo, Albacete, 2006. Actas de las jornadas, pp. 259-263, 2006.
6) La implementación del monitor SsTAT para GRIDSs denominada GSTAT se incluyó en la versión 1.9 de GRID superscalar para GT4-prews y se encuentra disponible para su descarga en el BSC-CNS [27].

A su vez, los desarrollos aquí realizados se han usado para la realización de estudios concretos en el campo físico-molecular dando lugar a los siguientes trabajos:

1) A. Niño, C. Muñoz-Caro, M.E. Castro, J. Díaz, S. Reyes and M. Mora. "New Computational Approaches to the Treatment of the Rovibrational problem". Advanced Workshop on Theoretical and Computational Methods for Molecular Spectroscopy and Collisions: Application to Astrophysical and Atmospherical Relevant Systems". CSIC, Universidad de Castilla-La Mancha, Universidad de Tunez-El Manar (Túnez), Universidad de Granada y Université Paris-Est Marne la Vallée (Francia).

2) "Nuevas técnicas computacionales aplicadas al estudio estructural y rovibracional molecular". Observatorio Astronómico Nacional de Chile. Ciclo de conferencias del Observatorio Astronómico Nacional de Chile. Santiago de Chile, Chile. 20 de abril de 2006.
3) "Aplicaciones Grid en Computación Científica". Universidad Tecnológica de Panamá (Sede Central). Ciclo de conferencias de la Facultad de Ingeniería de Sistemas Computacionales. Ciudad de Panamá, Panamá. 10 de diciembre de 2007.
4) "Tecnologías Grid". Universidad Tecnológica de Panamá. Centro Regional de Coclé. Ciclo de Videoconferencias. Coclé, Panamá. 12 de diciembre de 2007.

El modelo, propuesto en este trabajo, para la obtención de la hipersuperficie de energía potencial molecular ha sido utilizado por colaboradores en trabajos de investigación que han conducido a las siguientes publicaciones:

1) M. E. Castro, A. Niño, C. Muñoz-Caro. "Structural and vibrational theorical analysis of protonated formaldehyde in its X1A' ground electronic state", *Theoretical Chemistry Accounts*, 119, 343-354, 2008.
2) M.E. Castro, C. Muñoz-Caro and A. Niño. "Analysis of the nitrile and methyl torsional vibrations of n-propyl cyanide in its S0 electronic state". *Int. J. Quantum Chem*, 111, 3681-3694, 2011.
3) F. J. Melendez, C. Muñoz-Caro, A. Niño, J. Sandoval-Lira and A. Rangel-Huerta. "Structural and vibrational analysis of the OH torsional motion in difluorohydroxyborane". *Int. J. Quantum Chem*, 111, 4389-4399, 2011.

1.4 Estructura de la memoria

Este trabajo se organiza en diferentes capítulos, con los contenidos que se detallan a continuación:

- **Capítulo 1. Introducción**
  En el presente capítulo se describe la motivación de este trabajo, los objetivos y su estructura.

- **Capítulo 2. Antecedentes**
  Se presenta la información, encontrada en la revisión y análisis de la bibliografía, relacionada con este trabajo. Se presta especial interés a los aspectos relacionados con: *Scientific Workflow* (SWF), sistemas de monitorización y gestión de tareas en sistemas distribuidos, y entornos de programación de aplicaciones.

- **Capítulo 3. Modelo de Workflow científico para problemas de barrido de parámetros**
  Este capítulo presenta el modelo general propuesto para la resolución de problemas de barrido paramétrico en sistemas de computación distribuidos. Como parte esencial en la definición del modelo, se considera el operador matemático NSS (*Nested Summation Symbol*) [28]. Se presentan los algoritmos necesarios para que pueda ser utilizado en la práctica y como parte del modelo PSWF.

- **Capítulo 4. Validación del Modelo de Tratamiento de PSEs**
  En este capítulo se presenta la validación experimental del modelo PSWF. En particular, nos centraremos en el procesamiento, mediante computación distribuida, del gran número de tareas que se generan en las aplicaciones de barrido paramétrico. En concreto, se propone una solución general para la obtención de la hipersuperficie de energía potencial molecular basada en el modelo PSWF.

- **Capítulo 5. Modelo de monitorización y gestión de *Workflows* científicos**

  Este capítulo presenta el diseño de un sistema de supervisión y gestión de aplicaciones distribuidas aplicadas a la familia de modelos de programación StarSs. La implementación del sistema se integra en el entorno de programación GRIDSs y posteriormente se valida con problemas de alto rendimiento y de alta productividad.

- **Capítulo 6. Conclusiones y Trabajo futuro**

  En este capítulo se recogen las principales conclusiones de esta tesis y se plantean los retos futuros como continuación del trabajo desarrollado.

2 Antecedentes

En este capítulo se presenta el estado del arte relacionado con este trabajo. En particular, se revisa lo concerniente a los conceptos clave del mismo: *workflows*, operador NSS, monitorización y gestión de tareas en sistemas distribuidos. Puede considerarse como antecedente, para el caso del barrido de parámetros, el material incluido en el apartado 1.1. Allí, se incluyeron algunos casos típicos de barrido paramétrico para ilustrar la motivación de este trabajo. Para evitar redundancia no se ha incluido dicho apartado en este tema.

2.1 Workflows

La tecnología basada en *workflows* fue adoptada, en primer lugar, por la comunidad empresarial. Según la *Workflow Management Coalition* (WfMC) [29], un WF es la automatización de un proceso empresarial, en su totalidad o en parte, durante el cual los documentos, información o tareas se pasan de un actor (un recurso; humano o máquina) a otro, según un conjunto de normas de procedimiento. Para el desarrollo de un WF empresarial se utilizan tecnologías basadas en el paradigma de servicios web y existen estándares como XML, WSDL y SOAP que facilitan la interoperación entre servicios. El lenguaje de modelado de WF *Business Process Execution Language* (BPEL) [30] ha sido adoptado como estándar de facto por la comunidad.

Recientemente, han surgido los WF científicos, SWF [31], como un nuevo paradigma que permite formalizar y estructurar tareas complejas, lo que acelera el proceso de descubrimiento científico. El modelado de WF empresariales está más enfocado al uso de lenguajes de programación tradicionales. Este enfoque, en general, no es apropiado para el desarrollo de WF científicos dada la complejidad de los mismos. En su lugar, los científicos necesitan herramientas de un nivel de abstracción superior que puedan interconectar diferentes componentes, de alto nivel, que resuelven distintos problemas. Un SWF intenta capturar la serie de pasos analíticos que describen el proceso de diseño de experimentos computacionales. Los sistemas de WF científicos proporcionan un entorno que facilita el proceso de descubrimiento a través del análisis, combinación, gestión, simulación y visualización de datos científicos. La comunidad científica, de forma análoga a la empresarial, tiene a su disposición diversas herramientas para el modelado de SWF. Algunas de las más significativas son: Kepler [32], Pegasus [33], Taverna [34], o Triana [35].

Si bien el objetivo de los WF empresariales, orientados a flujo de control, es reducir los recursos humanos (y otros gastos), y aumentar los ingresos, el objetivo de los WF científicos, orientados a flujo de datos, es reducir los recursos humanos y computacionales, así como, acelerar el progreso científico.

A continuación, se enumeran algunos de los requerimientos necesarios para el desarrollo de SWF [36], [37], [38], [39], [40], [41]:

1. Proporcionar herramientas de diseño escalables orientadas a usuarios que no son especialistas en computación, fáciles de usar, con interfaces de usuario sencillas y

con las características más complejas bajo responsabilidad de la propia herramienta.

2. Composición jerárquica de componentes y servicios con la finalidad de reducir la complejidad y facilitar la reutilización.
3. Funciones reutilizables, lo suficientemente generales, para que sean aplicables en diferentes campos.
4. Que sean extensibles al usuario experto.
5. Distribuidos y separados del flujo de ejecución.
6. Que dispongan de herramientas de monitorización, recuperación de errores y gestión por parte del usuario.

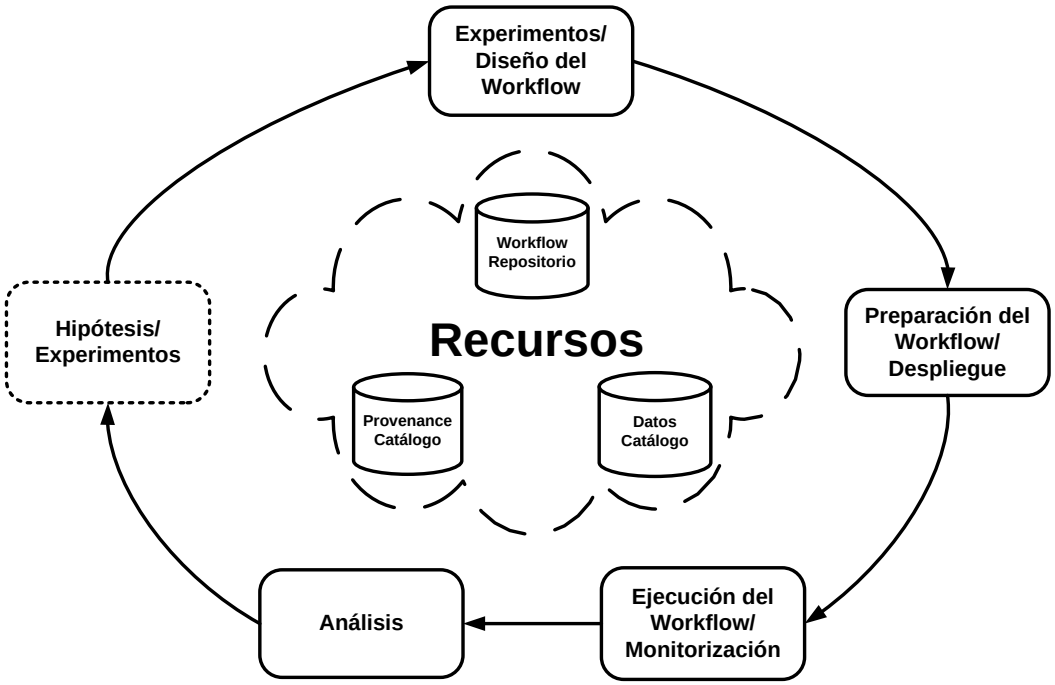

Figura 2.1 Ciclo de vida de *Scientific Workflows*

La Figura 2.1 muestra una visión general del ciclo de vida de un SWF [38], [42], [43], [44]. A partir de unas hipótesis, o de algunos resultados experimentales específicos, se inicia la fase de diseño del SWF. Durante esta fase, los científicos utilizan el repositorio de *Workflows* (WFR), para comprobar si existen plantillas de SWF que se adapten a su problema y puedan reutilizar o modificar. En caso contrario, para el diseño del nuevo SWF pueden utilizar componentes y *sub-workflows* que se encuentren en el repositorio. Los componentes individuales no tienen por qué ser interoperables entre sí, lo que dificulta aún más su reutilización en nuevos SWF. Inevitablemente, será necesario desarrollar aquellas partes del SWF para las que no se encuentren componentes en el

repositorio. Si se trata de un experimento nuevo es muy probable que haya que desarrollarlo en su totalidad.

Durante la preparación del SWF, véase Figura 2.1, el usuario selecciona los datos y establece los requisitos necesarios. El SWF puede requerir la planificación de los recursos de computación de alto rendimiento en sistemas locales o remotos (*cluster*, *grid* o *cloud*).

Durante la ejecución del SWF, véase Figura 2.1, se transfieren y procesan los datos de entrada que conducen a nuevos datos de salida. Para simulaciones a gran escala (que se ejecutan en cientos o miles de nodos, durante horas, días o semanas), el control del tiempo de ejecución es de vital importancia, de ahí la trascendencia de la monitorización y gestión del SWF. Es importante que la monitorización aporte flexibilidad a la hora de interaccionar con el SWF.

Los científicos tienen necesidad de inspeccionar e interpretar los resultados del SWF. En la fase de análisis se evalúan los resultados, se examinan las trazas de ejecución y las posibles dependencias entre los datos, y se llevan a cabo ejecuciones orientadas a tareas de depuración y análisis de rendimiento. Dependiendo de los resultados del análisis, las hipótesis de partida pueden ser revisadas dando lugar a nuevas iteraciones en el diseño y a nuevas ejecuciones del SWF.

A continuación, se enumeran algunas ventajas e inconvenientes de los SWF:

Ventajas

1. Formalizan, y en algunas de sus fases automatizan, el proceso científico.
2. Son fáciles de compartir; de desplegar en diferentes plataformas; de modificar y volver a ejecutar; de ampliar y reutilizar, comparados con la aproximación computacional clásica.
3. Pueden ayudar a gestionar la complejidad y usabilidad del proceso científico.
4. Proporcionan interfaces unificadas capaces de interaccionar con diferentes tecnologías.
5. Pueden ayudar a la supervisión de la ejecución, a la tolerancia a fallos y la repetición de experimentos.

Inconvenientes

1. En general, los SWF dependen de aplicaciones de terceros.
2. El control de versiones es una dificultad añadida al desarrollo de los SWF.
3. La complejidad del ensamblado de componentes sintáctica y semánticamente incompatibles.
4. La necesidad de disponer de equipos multidisciplinares para la utilización, en el proceso científico, de SWF.

Este trabajo se centra en el análisis de aquellos problemas cuya resolución puede abordarse utilizando un enfoque basado en un barrido de parámetros. Se trata de un determinado tipo de problemas cuyo comportamiento puede simularse en base a la modificación de una serie de parámetros (variables) de entrada. Se propone abordar este tipo de problemas por medio de la utilización de SWF.

2.2 Nested Summation Symbol

El operador, o símbolo, de adición anidada (*Nested Summation Symbol*, NSS) fue definido por Carbó y Besalú, véase [28]. Se trata de un operador que representa un número arbitrario de sumatorios anidados.

La notación del operador NSS es la siguiente:

$$\sum_{n}(j = i, f, s, L) \tag{2.1}$$

En la expresión (2.1), *j* es un vector que representa todas las posibles combinaciones de *n* diferentes índices. Los elementos que pertenecen a *j* tienen los siguientes límites:

$$\{i_k \leq j_k \leq f_k,\ si\ s_k \geq 0\}\ o\ \{i_k \geq j_k \geq f_k,\ si\ s_k \leq 0\};\ \forall k = 1,\ n \tag{2.2}$$

Los j_k índices pueden in/decrementarse, respectivamente, en saltos de s_k.

La dimensión del operador NSS viene dada por el índice *n*. De este modo, el operador que se muestra en la ecuación (2.1) representa el siguiente conjunto de *n* sumatorios anidados:

$$\sum_{j_1=i_1}^{f_1}\sum_{j_2=i_2}^{f_2}\cdots\sum_{j_n=i_n}^{f_n} \tag{2.3}$$

El operador realiza todas las sumas involucradas en la generación de todos los posibles valores del índice *j*. En esta formulación, el índice *k* del vector *j* oscila entre los valores que van desde i_k a f_k, según determina el vector $s = (s_1, s_2, \ldots, s_n)$.

El vector *L* es un vector lógico del tipo $\{\delta(L_i)\}$. El símbolo δ se corresponde con la delta de Kronecker lógica, *Logical Kronecker Delta* (LKD). El símbolo LKD es un argumento lógico y en esta función puede tomar dos posibles valores: 1 si el argumento es verdadero y 0 si es falso. Este vector puede utilizarse para excluir ciertos índices del vector *j* lo que implicaría la exclusión de sumatorios de la expresión (2.3). Por defecto, el valor del vector L = 1 = (1, 1, …, 1).

El NSS presenta varias propiedades interesantes. En primer lugar, es un operador lineal y, en segundo lugar, el producto de dos NSS es otro NSS, véase la referencia [45] para más información. Estas propiedades son las que nos llevan a considerar la idoneidad de

la aplicación del NSS en problemas PSE con un gran número de parámetros, y para generalizar la simulación de experimentos que involucren diferentes espacios paramétricos.

El operador NSS fue definido originalmente para hacer frente a los sumatorios anidados que aparecen en el tratamiento de sistemas mecanocuánticos por el método de perturbaciones. Sin embargo, se ha propuesto para diferentes aplicaciones matemáticas como son [45]: la generación de variaciones y combinaciones, la expresión explícita del determinante de una matriz cuadrada o la expansión en serie de Taylor de una función de *n* variables.

Desde el punto de vista de la computación, la implementación de una serie de Taylor requiere la utilización de tantos bucles anidados como sumatorios y en el caso de una *n*-serie de Taylor sería necesario un encadenamiento de bucles anidados con profundidad *n*. Esta es una de las propiedades que destacan Carbó y Besalú en su trabajo [45] como "*generalized nested do loop*". Este comportamiento puede ser adaptado al tipo de problemas considerado en este trabajo.

A continuación, se ilustra con un ejemplo la expresión (2.1), si n=2, i=(1, 2), f=(4, 0), s=(3, -2), entonces:

$$\sum_{n}(j = i, f, s) = (1,2) + (1,0) + (4,2) + (4,0)$$

2.3 Monitorización y gestión de tareas en sistemas distribuidos

La monitorización, desde el punto de vista clásico del término, se define como la recogida, interpretación y presentación de la información relativa a la situación de un sistema hardware o software de interés [46].

El seguimiento dinámico de las tareas en sistemas distribuidos implica la recopilación de información sobre las tareas concurrentes individuales que constituyen los procesos. En este contexto, se han llevado a cabo diversos trabajos desde puntos de vista diferentes. Así, García-Molina y otros [47] describen una metodología para la depuración de los sistemas distribuidos desde una perspectiva de abajo hacia arriba. Este trabajo implica el uso de archivos de seguimiento obtenidos en las tareas de supervisión.

Por otro lado, Snodgrass [48] considera la monitorización como una actividad de procesamiento de información, proponiendo un modelo relacional para la organización de la información generada en los sistemas distribuidos. Por su parte, Harrison [49] propone un método para minimizar el impacto de la supervisión de las tareas. Este enfoque se basa en el uso del análisis del estado de los programas y de las trazas de comportamiento dinámico. Una revisión detallada de los sistemas de control distribuido, en la era pre-*Grid*, es el de Joyce [46].

En el *Grid*, su sistema de información de gestión (GIS) es el encargado de supervisar el estado de los recursos. Sin embargo, el seguimiento también es de interés fundamental para la planificación de tareas, la replicación de datos, la contabilidad, el análisis de datos o para la optimización de las aplicaciones [50]. La monitorización del estado de una aplicación en un *Grid* puede llevarse a cabo, en principio, a partir de la información proporcionada por el sistema de gestión de recursos *Grid*. Sin embargo, de esta forma, sólo se puede obtener información acerca de las tareas individuales. Esto no es un enfoque práctico cuando nuestra aplicación controla decenas o cientos de miles de tareas.

Con respecto a los sistemas de monitorización en *Grid* es necesario referirse a las especificaciones del *Open Grid Forum* (OGF) [51]. En un documento de trabajo [52], perteneciente a este organismo, se establecen un conjunto de quince escenarios donde se justifica la necesidad de la monitorización para identificar, obtener y utilizar datos de rendimiento. En particular, el escenario 6 se corresponde con la monitorización del estado de las tareas.

Además, la OGF ha propuesto un estándar para la arquitectura de monitorización de sistemas *Grid* denominado *Grid Monitoring Architecture* (GMA) [53]. El modelo GMA describe los principales componentes de un sistema de monitorización *Grid* y sus interacciones fundamentales. En este modelo, cualquier recurso (procesadores, medios de almacenamiento, conexiones de red o aplicaciones) se define como una entidad. La arquitectura propuesta se basa en tres tipos de componentes. En primer lugar, el productor que es una entidad que produce eventos. En segundo lugar, el consumidor que es cualquier proceso que recibe eventos. Por último, el servicio de directorio o de registro que permite a los productores la publicación de los eventos que producen. El directorio permite a los consumidores encontrar los eventos de interés. Puede encontrarse más información sobre monitorización de tareas en sistemas *Grid* en [50].

Existen numerosas herramientas para monitorizar recursos computacionales en sistemas distribuidos. Por ejemplo, el servicio de monitorización y descubrimiento (*Monitoring and Discovery Service*, MDS) [54] constituye la infraestructura de información de *Globus Toolkit* [55]. Otra herramienta es Ganglia [56], que es una solución escalable para monitorizar sistemas distribuidos de alto rendimiento como *Clusters* y *Grids*. Por otro lado, se dispone del, así llamado, servicio meteorológico de red (*Network Weather Service*, NWS) [57] que es un sistema portable y no invasivo de vigilancia y predicción en sistemas distribuidos. Otro ejemplo es el *Resource Monitoring for the Grid* (GridRM) [58]. GridRM se basa en una arquitectura de tres niveles y se accede a la información de los recursos a través de otros servicios tales como MDS o el *Simple Network Management Protocol* (SNMP). Esta herramienta utiliza una base de datos relacional para almacenar la información. Por último, GridICE [59] es una herramienta de monitorización distribuida diseñada para sistemas *Grid*. Puede consultarse información adicional sobre servicios de monitorización *Grid* en la referencia [50].

En contraste con las herramientas de monitorización de sistemas, existen pocas herramientas de monitorización del rendimiento y análisis de aplicaciones distribuidas. Por ejemplo, el *Network Application Logger Toolkit* (NetLogger) [60] se utiliza para el análisis de rendimiento en sistemas complejos, tales como aplicaciones cliente–servidor y/o aplicaciones multihilo. OCM-G [61] es una infraestructura para monitorizar aplicaciones *Grid* que utiliza OMIS [62] como una interfaz de comunicación. ProActive [63] es una librería paralela, distribuida y de programación concurrente en Java. Las

aplicaciones desarrolladas con ProActive se monitorizan de forma transparente mediante una aplicación externa: *Interactive Control and Debugging for Distribution* (IC2D) [64]. AutoPilot [65] es un sistema de monitorización y puesta a punto distribuido. Mercury [66] ha sido diseñado para satisfacer requerimientos de monitorización y rendimiento en sistemas *Grid*. Finalmente, el sistema de supervisión de tareas AMon [67] proporciona al usuario información suficiente sobre tareas y recursos. Los datos de monitorización son preanalizados para sugerir al usuario posibles problemas. Los datos se presentan de manera gráfica, lo que permite la interactividad.

Otro enfoque de la monitorización de aplicaciones, de interés para el presente trabajo, se basa en la supervisión de SWF. Informar sobre el estado de un SWF es una tarea compleja y hay pocas herramientas disponibles para ello. Una de ellas es *Pegasus* [68] que realiza un mapeo del SWF sobre el conjunto de recursos *Grid* disponible y el usuario puede monitorizarlo utilizando la herramienta *pegasus-status*. Otra opción es Triana [69]. Triana permite generar SWF a través de un entorno gráfico orientado al análisis de datos e integrado con un *Grid* a través de la herramienta *Grid Application Toolkit* (GAT) [70]. A su vez, ha sido propuesta y aplicada una arquitectura genérica para la supervisión y gestión de aplicaciones en entornos *Grid* (*gridMonSteer* o GMS) [71]. Es posible utilizar GMS para monitorizar SWF desarrollados con Triana.

Las aproximaciones anteriores se centran, fundamentalmente, en el descubrimiento de recursos y, en menor medida, en la supervisión del SWF. En este trabajo se propone un enfoque que facilite al investigador la supervisión y gestión del SWF, así como el seguimiento de la ejecución de las tareas desplegadas en el sistema distribuido y la interacción a nivel de tarea. Otro aspecto importante, que aporta este trabajo, es el análisis *off-line* de los resultados del SWF.

2.4 STAR Superscalar

Star Superscalar (StarSs) es una familia de modelos de programación paralela desarrollada por investigadores del *Barcelona Supercomputing Center* (BSC-CNS) [72]. Cada uno de los miembros de StarSs tiene como objetivo facilitar al usuario la programación para una determinada arquitectura. Por ejemplo, GRIDSs y *COMP Superscalar* (COMPSs) [73] para entornos *Grid* y *Cloud*, *Cluster Superscalar* (ClusterSs) [74] para *Clusters* de computadores, *SMP Superscalar* (SMPSs) [75] para procesadores simétricos, *Cell Superscalar* (CellSs) [76] para el procesador Cell/B.E. o *GPU Superscalar* (GPUSs) [77] para aceleradores gráficos.

Sin embargo, todos ellos comparten la misma filosofía: el usuario debe seleccionar un conjunto de métodos en una aplicación secuencial que se ejecutarán mediante tareas paralelas en los recursos disponibles. Además, el usuario debe especificar los parámetros de cada método, así como, el tipo de cada parámetro y si éste es de entrada, salida o entrada/salida; esta información es utilizada por StarSs para descubrir, en tiempo de ejecución, las dependencias de datos entre tareas. En cuanto al código de la aplicación principal, tiene que ser ligeramente modificado para incluir algunas llamadas a la API o *pragmas*, aunque en algunos casos se puede dejar sin cambios.

StarSs implementa un modelo de flujo de datos: en tiempo de ejecución los métodos seleccionados por el usuario son reemplazados automáticamente por las llamadas al *runtime* que se encarga de crear las tareas. El *runtime* realiza un análisis de las dependencias de datos entre tareas y construye un grafo de dependencias. El paralelismo que exhibe el grafo se explota tanto como sea posible, planificando las tareas libres de dependencias en los recursos disponibles. Este modelo de ejecución ayuda a la reducción del camino crítico en aplicaciones irregulares. El *runtime* también gestiona los datos (realización de copias de datos, transferencias o cambios de nombre, si es necesario) y los controles de finalización de tareas.

Para este trabajo son relevantes los miembros de la familia StarSs diseñados para arquitecturas distribuidas: GRIDSs, COMPSs y ClusterSs. A continuación, se incluye información adicional específica de cada uno de estos modelos.

- **GRID Superscalar**

  GRIDSs [26] es un paradigma para la programación de aplicaciones *Grid*. Con GRIDSs las aplicaciones secuenciales compuestas de tareas con cierta granularidad se transforman, de forma automática, en aplicaciones paralelas donde las tareas se ejecutan en diferentes nodos del *Grid* de computadores. GRIDSs reduce la complejidad en el desarrollo de aplicaciones al mínimo, de tal forma que el esfuerzo invertido en la programación de una aplicación paralela, que se ejecutará en un entorno distribuido como es un *Grid* de computadores, sea equivalente al de su homóloga secuencial.

  GRIDSs está formado por los siguientes componentes: interfaz de usuario, generador automático de código, *runtime*, *Deployment Center* y GRID *superscalar* monitor. El *runtime* dispone de las características siguientes: descubrimiento de recursos, análisis de la dependencia de los datos, planificación de tareas, renombrado de ficheros, gestión compartida de disco y política de transferencia de ficheros, *checkpointing* y tolerancia a fallos.

  Para desarrollar una aplicación con GRIDSs, un programador debe pasar por las etapas siguientes:

  1. Definición de tareas: identificación de funciones en la aplicación que vayan a ser ejecutadas en el *Grid* de computadores.
  2. Definición de los parámetros: identificar qué parámetros son ficheros y cuáles son escalares.
  3. Escribir el programa secuencial: el código principal del programa y de las tareas.

  Para la ejecución de una aplicación se utiliza el *Deployment Center* (realiza la copia y la compilación remota de la aplicación). La ejecución de la aplicación implica llamadas al *runtime* en lugar de ejecutar realmente las funciones. El *runtime* realiza un análisis de dependencias de datos entre las diferentes funciones, basado en parámetros especificados como ficheros, construyendo así un grafo acíclico dirigido que indicará el orden mínimo que se debe respetar en la ejecución de las funciones. De este grafo, las tareas que tengan sus dependencias resueltas serán candidatas a ser ejecutadas. GRIDSs elegirá la tarea más

conveniente y un recurso en el *Grid* para su ejecución remota haciendo uso del *middleware* correspondiente.

- **COMP Superscalar**

  COMPSs [78] es una nueva versión de GRIDSs orientada a Java cuya arquitectura ha sido rediseñada siguiendo los principios de GCM (*Grid Component Model*), un modelo de componentes ideado para el *Grid*. COMPSs ofrece una modelo de programación *Grid* sencillo y transparente para el usuario, que sólo requiere la especificación de las tareas que se ejecutaran en el *Grid*, siendo libre éste de dejar sin cambios el código de su aplicación Java.

  Algunas de las principales propiedades de COMPSs son:

  1. Composición Jerárquica: el *runtime* de COMPSs se define como un conjunto de componentes, cada uno de ellos a cargo de una funcionalidad determinada.
  2. Separación entre interfaces funcionales, que pueden utilizarse para el acceso a la funcionalidad de cada componente, y no funcionales, que proporcionan la funcionalidad de cada controlador.
  3. La comunicación entre componentes puede ser síncrona o asíncrona.
  4. Proporciona una interacción colectiva entre componentes a través de una comunicación *multicast* (uno a muchos).

  El *runtime* de COMPSs lo forman los siguientes componentes, cuyos nombres son autoexplicativos: *Task Analyzer*; *Task Scheduler*; *Job Manager*; *File Manager*.

- **Cluster Superscalar**

  ClusterSs es un miembro de la familia StarSs diseñado para ejecutarse en Clusters o SMPs (*Symmetric MultiProcessing*). Se trata de un modelo de programación basado en tareas asíncronas que se crean y asignan a los recursos disponibles mediante el uso del protocolo de comunicaciones IBM APGAS (*Asynchronous Partitioned Global Address Space Model*) [79].

  El diseño de ClusterSs se basa en el modelo *master-worker* donde el nodo principal (*master*) genera las tareas y el resto de nodos de trabajo (*workers*) las ejecutan. Se trata de un modelo escalable. Los *workers* pueden intercambiar datos sin pasar por el nodo principal lo que ayuda a que no se convierta en un cuello de

botella. Además, cada *worker* mantiene una caché de datos que hace posible la explotación de la localidad de datos y evita transferencias innecesarias. En cuanto a la ubicación de los datos, ClusterSs permite asignar los datos a cada *worker* a través de las tareas, lo que impide que la memoria general esté limitada al nodo principal, y también elimina la necesidad de transferir todos los datos de los *workers* al principio de la aplicación, lo que retrasaría la ejecución.

3 Modelo de *Workflow* científico para problemas de barrido de parámetros

En este capítulo se propone un modelo de *workflow* científico denominado PSWF (*Parameter Sweep Scientific Workflow*), como una aproximación, basada en SWF (*Scientific Workflow*) y en NSS (*Nested Summation Symbol*), para la resolución de problemas de barrido paramétrico.

El análisis llevado a cabo en la motivación de este trabajo, apartado 1.1, permite justificar la necesidad de desarrollar un modelo de referencia para problemas de barrido de parámetros debido a que no se observa, tras el estudio detallado de la bibliografía mencionada, la aplicación de una metodología común.

En la misma línea, el estudio realizado para la elaboración del apartado 2.1, *Workflows*, lleva a concluir que el ciclo de vida propuesto para el desarrollo de SWF no es adecuado para abordar la resolución de problemas de barrido paramétrico debido a que no contempla una formalización para el problema barrido de parámetros. Por estas razones, en este capítulo se propone un modelo para la resolución de este tipo de problemas.

En los dos primeros apartados del capítulo se propone un modelo de ciclo de vida y una arquitectura de niveles que pueden usarse como referencia, por la comunidad científica, para el planteamiento y posterior resolución de este tipo de problemas. En el último apartado se aborda la formalización matemática del problema de barrido de parámetros y se propone una solución computacional general basada en la misma. Dicha solución es integrable en el modelo de ciclo de vida aquí propuesto.

3.1 Ciclo de vida PSWF

El ciclo de vida de los SWF se infiere después de observar cómo se realizan los experimentos y simulaciones científicas. Por lo general, sólo hay un grupo de usuarios, los científicos, asumiendo diferentes roles: diseñador, usuario, administrador y analista. Este aglutinamiento de funciones en un único "actor" da idea de la dificultad del proceso.

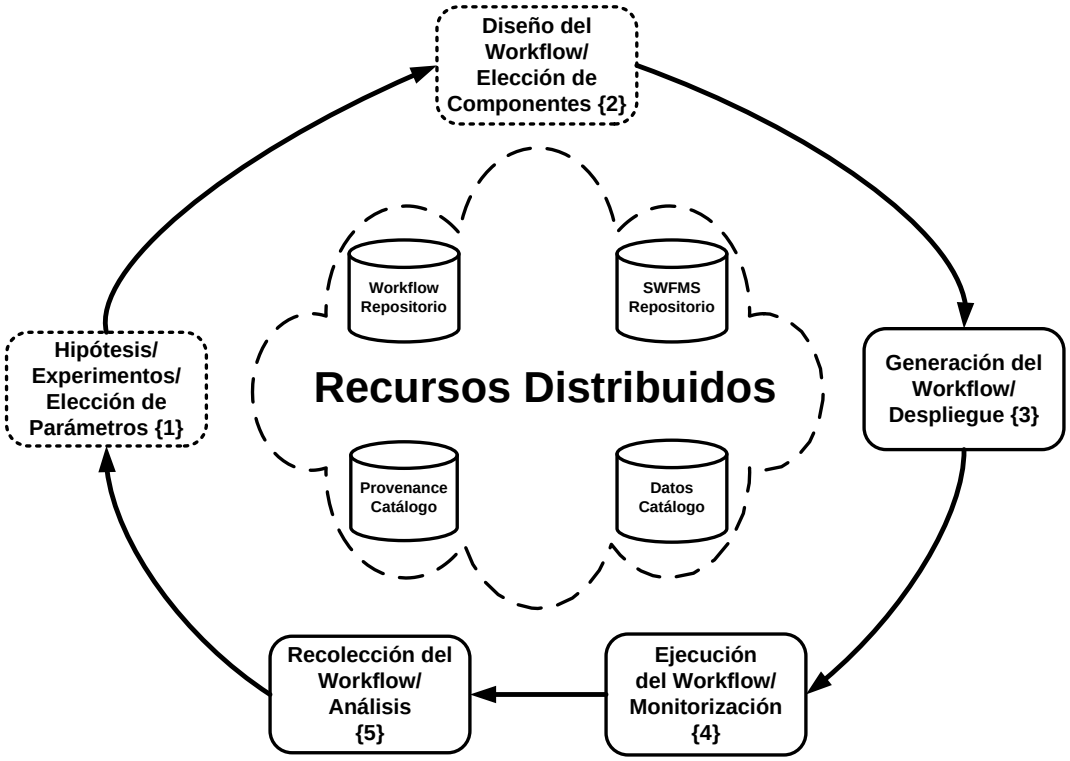

Figura 3.1 Ciclo de Vida PSWF

La Figura 3.1 muestra el ciclo de vida que se propone en este trabajo como modelo a seguir para la resolución de problemas de barrido paramétrico, (*Parameter Sweep*, PS). Se trata de un modelo general de ciclo de vida que puede aplicarse a cualquier tipo de problema de PS. El modelo se denomina PSWF, *Parameter Sweep Sciencitific Workflow*, debido a que está basado en *Scientific Workflows* (SWF).

En el modelo propuesto en la Figura 3.1, las cajas con borde discontinuo ({1} y {2}) representan procesos no automatizables. Aquí, son los expertos en cada campo los encargados de aportar las respuestas a las preguntas planteadas en estos apartados. El resto de fases en el ciclo de vida ({3}, {4} y {5}) sí se pueden realizar de forma automática, como quedará demostrado en el resto del trabajo.

El primer paso del ciclo de vida, véase {1} en la Figura 3.1, trata la fase de *Hipótesis/Experimentos/Elección de Parametros*, donde el equipo de investigadores parte de unas hipótesis y/o de ciertas evidencias experimentales. El equipo conoce el problema en profundidad y afronta su resolución mediante un barrido total o parcial de determinados parámetros. La primera tarea es seleccionar los parámetros adecuados que, posteriormente, se utilizarán en las simulaciones computacionales. Se trata de una tarea específica que no es posible automatizar. Sólo el conocimiento en la materia, por parte de los expertos, puede resolver este primer paso en el ciclo de vida del PSWF.

En cuanto a las características de los parámetros, este modelo de ciclo de vida está diseñado teniendo en cuenta los requerimientos siguientes:

1. El número de parámetros, *n*, deber ser finito aunque no existen umbrales mínimo ni máximo.
2. El rango de cada parámetro debe presentar una cardinalidad finita y tener un inicio, un final y un intervalo o incremento que nos lleve del valor inicial al final o viceversa.
3. Los parámetros pueden pertenecer a cualquier tipo de datos en tanto que se satisfagan los dos requisitos anteriores.

Aunque la elección de los parámetros es una tarea determinante en el ciclo de vida del PSWF hay que tener en cuenta que tras una o varias iteraciones pueden modificarse si el análisis de los resultados así lo requiere.

Una vez seleccionados los parámetros se aborda el *Diseño del Workflow/Elección de Componentes*, véase {2} en la Figura 3.1. En esta etapa, el equipo de investigación se enfrenta a un paso crítico, encontrándose aquí la decisión más importante de todo el proceso desde el punto de vista del diseño del PSWF. En esta fase, es necesario un análisis detallado del repositorio de WF (*Workflow Repository*, WFR) y de los sistemas de gestión de WF, véase Figura 3.1. Por lo general, existe una relación muy directa entre ambos debido a que los SWF se desarrollan mediante los *Scientific Workflow Management System* (SWFMS). Por lo tanto, es conveniente analizar primero el WFR debido a que si se encuentra un SWF que resuelva el problema planteado por el equipo de investigación, de forma indirecta, se habrá elegido el SWFMS.

El repositorio de SWFMS, véase la Figura 3.1, está formado por un catálogo de herramientas para el desarrollo de SWF. Este grupo de herramientas puede dividirse en

dos grandes subconjuntos: *Workflow Tools* (WFT) y entornos de programación de aplicaciones. En general, los WFT incorporan algoritmos utilizados frecuentemente en determinados campos para la resolución de determinados problemas, mientras que los entornos de programación están más enfocados a que el investigador pueda adaptar sus propios algoritmos. También hay que destacar que los WFT, por lo general, no se centran en problemas de barrido de parámetros (como excepción a esta afirmación véase el trabajo desarrollado por *David Abransom* y Colaboradores [80]).

Un factor importante a tener en cuenta en la elección del SWFMS, es la existencia en el repositorio de SWF, véase *Workflow* Repositorio (WFR) en la Figura 3.1, de un SWF concreto desarrollado, previamente, para la resolución de este mismo problema. Si existe el binomio (SWF, SWFMS) para la resolución de un determinado problema, se puede dar por concluida la segunda fase, aunque es importante resaltar que no suele ser lo habitual. En este punto viene a colación mencionar el proyecto europeo SHIWA (*SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs*) [81] cuyo objetivo principal es ofrecer a la comunidad científica internacional un punto de encuentro donde tengan a su disposición las herramientas necesarias para el desarrollo de SWF. En principio, el proyecto estará vigente durante el trienio 2010-2012.

Si no existe la pareja (SWF, SWFMS), que por otra parte es lo más habitual, el procedimiento a seguir, propuesto en este trabajo, es el siguiente:

1. Elección del SWFMS que se va a utilizar para el desarrollo del SWF teniendo en cuenta el campo al que pertenece el problema, determinando si se dispone de algoritmos previos que puedan ser reutilizados para la resolución de *sub-WFs*. En caso afirmativo, la mejor opción es la elección de un entorno de programación de aplicaciones que de soporte al lenguaje de programación usado para la implementación de los algoritmos previos.
2. Diseño del SWF, incidiendo en la reutilización de componentes que puedan encontrarse en el WFR y que, en la medida de lo posible, sean interoperables con el SWFMS seleccionado en el paso 1.
3. Realización del SWF, incluyendo todos sus componentes software, mediante el SWFMS elegido en el punto 1 de este procedimiento.

Como se ha podido observar, la fase {2} es un proceso complejo que requiere de un equipo de trabajo multidisciplinar que se encargue de analizar teórica y empíricamente, de

forma pormenorizada, los repositorios de SWF y SWFMS existentes. En la actualidad, y a falta de más proyectos como SHIWA, esta tarea se lleva a cabo de forma análoga a una revisión bibliográfica sobre un determinado tema.

Es conveniente hacer hincapié en que la fase {2} del ciclo de vida propuesto en este apartado, véase Figura 3.1, es válida para el desarrollo de SWFs genéricos, no sólo para los SWF orientados a barrido de parámetros, PSWF, que son el centro de este capítulo y de este trabajo.

Retomando el ciclo de vida, tenemos en tercer lugar, véase {3} *Generador del Workflow/Mapping* en la Figura 3.1, la generación del barrido paramétrico y posterior planificación, previa al despliegue, sobre los recursos para la ejecución. Como puede observarse en la Figura 3.2, el módulo encargado de realizar el barrido paramétrico es el *Workflow Generator* (WG). Dicho módulo recibe como entradas el WF obtenido en {2} y los parámetros definidos en {1}. Como salida, el módulo proporciona una batería de $n = \prod_{k=0}^{n-1} m_k$ ejemplares de WF que serán almacenadas en el *Catálogo de Datos* (DC), véase Figura 3.1, siendo m_k la cardinalidad de cada conjunto de parámetros. Aunque la Figura 3.2 induzca a pensar que el resultado es una matriz cuadrada en conveniente matizar que no es así, ya que dos parámetros diferentes no tienen por qué disponer del mismo tipo ni rango de valores.

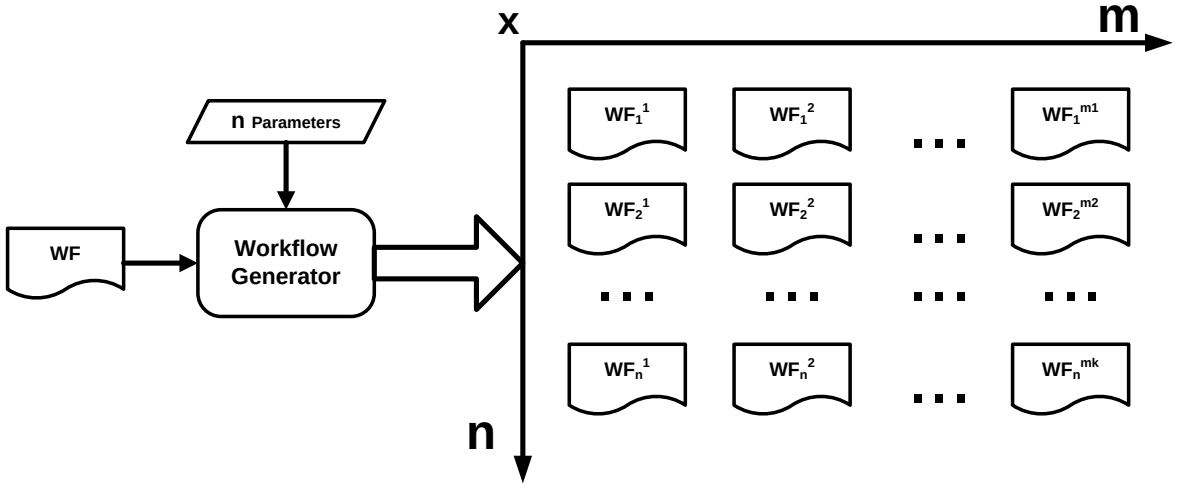

Figura 3.2 Funcionalidad del módulo *Workflow Generator* (WG)

El apartado 3.3 de este capítulo aborda un tratamiento formal del problema de barrido de parámetros que permite una realización general del módulo *Workflow Generator* (WG), véase la Figura 3.2.

El SWFMS, elegido en {2}, asume la responsabilidad del despliegue cuya función es planificar la ejecución de las tareas, asociadas a los *n* ejemplares de WF, generadas por el módulo WG. Recordemos que no existen dependencias de datos (comunicación) entre las *n* tareas obtenidas tras el barrido llevado a cabo por WG y que, por tanto, nos encontramos ante una aproximación de "grano grueso" (*Coarse Grained*, CG) [16].

Como en la etapa anterior del ciclo de vida, la formalización propuesta en el apartado 3.3 permitirá una aproximación genérica para la localización de los *n* ejemplares del WF y posterior planificación y envío a los recursos disponibles.

Continuando con el ciclo de vida, observamos en cuarto lugar, véase {4} en la Figura 3.1, que se lleva a cabo la ejecución de los *n* ejemplares del WF gracias a la funcionalidad aportada, a tal efecto, por el módulo ejecución del *Workflow* (WE), perteneciente al SWFMS elegido en {2}. Las entradas a WE son los *n* archivos (o grupos de archivos) que el módulo WG ha generado y almacenado en el catálogo de datos. Nuevamente, el tratamiento del apartado 3.3. proporcionará una solución genérica al acceso ordenado a los *n* ejemplares del WF.

Durante esta fase se hace imprescindible, para el equipo de investigación, la utilización de una herramienta de monitorización y gestión de tareas que proporcione información *on-line* y *off-line* de todo el proceso de ejecución del PSWF, de cada WF, y de cada tarea individual. Durante la ejecución, el monitor, almacenará toda la información en el catálogo *de* Datos. La información podrá ser requerida en cualquier momento de la ejecución por los usuarios. Una vez finalizada la ejecución, el monitor archivará, en el catálogo de *Provenance,* un registro histórico de la ejecución del PSWF para su ulterior consulta, véase Figura 3.1.

En el capítulo 5, de este trabajo, se presenta el desarrollo de un sistema de monitorización y gestión de SWF, denominado SsTAT (*Star superscalar Status*), que podrá ser utilizado por los investigadores para la supervisión y gestión de PSWFs realizados con alguno de los miembros de la familia StarSs.

En último lugar, véase {5} en la Figura 3.1, el proceso de recolección de *Workflow* (WC) realiza la recogida de los *n* archivos (o grupos de archivos). Aquí, una vez más, la aproximación desarrollada en el apartado 3.3 proporciona una solución genérica a este problema. En esencia, se trata de aplicar una función que depende del problema, para analizar, procesar e interpretar los *n* archivos (o grupos de archivos) unificando la

información resultante en uno o varios ficheros para su posterior análisis por parte del equipo de investigación. Existen SWFMS que aportan determinadas herramientas de análisis aunque, en general, se trata de un aspecto que depende del problema.

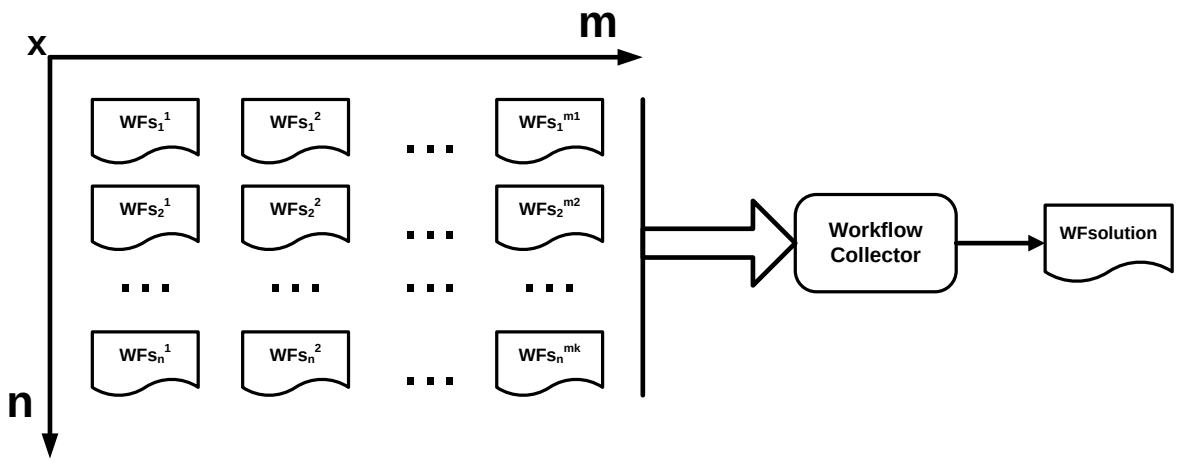

Figura 3.3 Representación gráfica del módulo *Workflow Collector* (WC)

3.2 Arquitectura de referencia

En este apartado se propone una arquitectura de referencia, basada en el modelo PSWF desarrollado en el apartado 3.1 de este capítulo, utilizable por la comunidad científica para afrontar la resolución de problemas de barrido de parámetros. La Figura 3.4 muestra la arquitectura de niveles propuesta para el modelo PSWF. Cada nivel utiliza servicios del nivel inferior y proporciona servicios al nivel superior. A su vez, cada nivel dispone de ciertas restricciones impuestas por el nivel inmediatamente inferior. A continuación, se hace una descripción de abajo a arriba (*botton-up*) de cada uno de los niveles.

- Distributed Resources

  Esta capa se corresponde con la infraestructura física, *hardware*, que conforma el sistema distribuido utilizado. El sistema, para la resolución del problema de barrido paramétrico considerado, puede elegirse entre los siguientes: *Cluster*, *Grid*, *Cloud*, o cualquier otro sistema distribuido existente o que pueda surgir en el futuro. Por otra parte, puede observase en la lista Top 500 [82] que la inmensa mayoría de los supercomputadores siguen una arquitectura de tipo *Cluster* que, a su vez, pueden agruparse para formar un *Grid*. Estos, resultan sistemas idóneos para la ejecución de un mismo algoritmo sobre múltiples datos debido a la ausencia de dependencias o, lo que es lo mismo, por tratarse de un enfoque de grano grueso.

  El criterio de elección debe centrarse en el análisis de tiempos, basado en una muestra de puntos, con el fin de obtener el valor de la desviación típica. Si la desviación es pequeña es recomendable elegir un sistema homogéneo ya que se minimizan los problemas de planificación, equilibrado de la carga y se reduce el tiempo de la simulación. En cambio, si la desviación es grande el sistema heterogéneo puede ser adecuado si se cuenta con una buena estrategia de planificación [83].

- Core Middleware

  Se trata de una capa *software* de abstracción que proporciona una interfaz de programación de aplicaciones distribuidas y servicios a aplicaciones para interactuar o comunicarse con otras aplicaciones, redes, hardware y/o sistemas operativos. El middleware depende del recurso distribuido subyacente. Si como

recurso se dispone de un *Cluster*, el repositorio de middleware disponible es muy amplio. A modo de ejemplo, podemos mencionar herramientas como MPI [84] o APGAS [79]. En cambio, si como recurso se utiliza un *Grid* algunas de las opciones comunes, en cuanto a middleware, son Globus [55], gLite [85] o UNICORE [86].

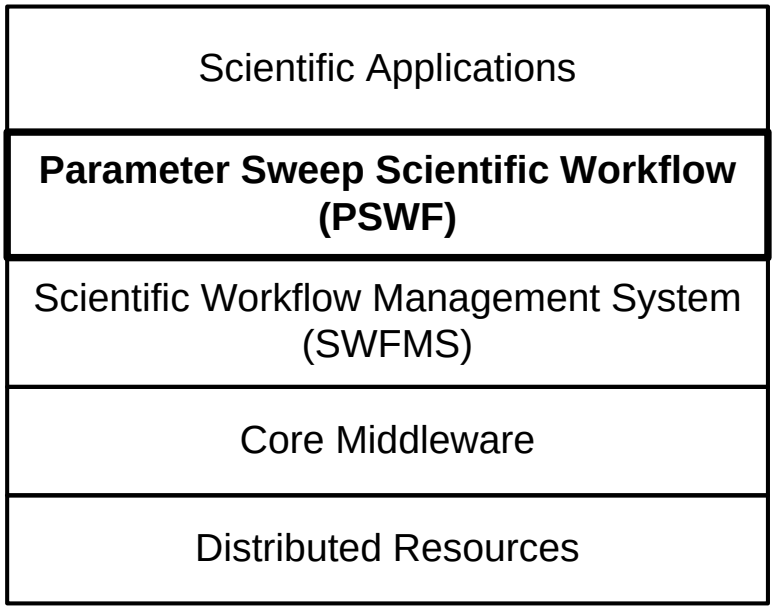

Figura 3.4 Arquitectura PSWF

- **Scientific Workflow Management System (SWFMS)**

  Un SWFMS es un sistema que define, modifica, administra, supervisa y ejecuta SWFs mediante la ejecución de tareas cuyo orden viene determinado por la lógica del WF. Existen dos grandes familias de sistemas de gestión de SWFs: las *Workflow Tools* (como Kepler [87], Pegasus [88], Taverna [89] o Triana [90]) y los entornos de programación de aplicaciones (ClusterSs [74], GRIDSs [91], Ninf-G [92] o ProActive [93]). No todos los ejemplos anteriores disponen de versiones para *Cluster* y *Grid*.

  En este trabajo se recomienda el uso de entornos de programación de aplicaciones, en la resolución de problemas de barrido paramétrico, debido a que al estar basados en lenguajes de programación permiten la reutilización simple de los algoritmos secuenciales que normalmente existen en la resolución clásica de estos problemas.

- **Parameter Sweep Scientific Workflow (PSWF)**

  Este nivel es el que ha sido desarrollado en este capítulo. Aquí, se ha propuesto un modelo general, denominado PSWF, que la comunidad científica puede utilizar para la resolución de problemas de barrido paramétrico. El modelo no depende del tipo de problema ni del número de parámetros.

- **Scientific Applications**

  En este nivel se encuentran los distintos equipos de investigación pertenecientes a áreas científicas tan diversas como: ciencias de la computación, de la tierra, de la vida, astrofísica, física molecular, etc. Estos equipos son los responsables del desarrollo del SWF, utilizando la metodología PSWF, para la resolución-simulación de problemas mediante la estrategia de barrido de parámetros.

3.3 Formalización del problema de barrido de parámetros

Como se ha mostrado en el apartado 3.1, de este capítulo, existen una serie de tareas genéricas que son comunes a cualquier problema de barrido de parámetros. Sería deseable que estas tareas se pudieran tratar de forma genérica, independientemente del problema específico al que correspondan. Para lograr este objetivo debemos abordar un proceso de formalización abstracta del problema de barrido de parámetros. En este apartado se considera este problema usando una aproximación basada en el operador NSS presentado en el apartado 2.2 de este trabajo.

El punto de partida, para el tratamiento genérico del problema de barrido de parámetros, es el desarrollo de una metodología para la generación de los diferentes valores de los parámetros de entrada propuestos por el equipo de investigación en la fase {1} del ciclo de vida, véase Figura 3.1.

Consideremos el problema en detalle. Sea v el valor final de la magnitud en estudio, obtenido para un valor determinado de n parámetros, véase Figura 3.5. v puede entenderse como un valor único o, en general, como un conjunto de valores correspondiente a un tipo abstracto de datos. Todos los posibles valores de v definen el conjunto de valores V.

v	Resultado final de la magnitud en estudio
n	Número de parámetros
$\boldsymbol{p}$	Vector con un valor por parámetro (n)
$\boldsymbol{j}$	Vector con un índice por parámetro (n)

Figura 3.5 Elementos a considerar en la formalización del problema de barrido de parámetros

Si $\boldsymbol{p}$ identifica un vector cuyas componentes son los valores de los n parámetros que producen v, tendremos que v es función de $\boldsymbol{p}$: $v=v(\boldsymbol{p})$. A su vez, un determinado parámetro sólo puede adquirir uno de varios valores. Es decir, cada parámetro se corresponde con una serie de m valores que se pueden considerar identificados por un índice, con 0 para el valor mínimo y $m-1$ para el valor máximo, véase Figura 3.5.

A su vez, si $\boldsymbol{j}$ es un vector que recoge el valor del índice para cada uno de los n parámetros, tendremos que $\boldsymbol{p}$ es función de $\boldsymbol{j}$ ($\boldsymbol{p}=p(\boldsymbol{j})$) y, por tanto, $v= v(p(\boldsymbol{j}))$. Es decir, v viene determinado por el vector de índices, $\boldsymbol{j}$. Este es el punto clave, los valores de los

parámetros pueden ser indexados y se puede generalizar el tratamiento trabajando con los índices de los parámetros en lugar de con sus valores.

Este problema puede abordarse mediante el uso de algunos conceptos de la teoría de conjuntos. Desde este punto de vista, el objetivo perseguido es la generación de todos los elementos, del conjunto de índices del espacio de parámetros de entrada, $J=\{\boldsymbol{j}\}$, para un total de n parámetros diferentes. Teniendo en cuenta que cualquier vector $\boldsymbol{j}$ tiene n componentes (una por cada parámetro) podemos enunciar el problema en los siguientes términos:

¿Cómo pueden generarse todos los grupos diferentes de n elementos (todos los posibles $\boldsymbol{j}$*'s) que puedan formarse con n grupos de elementos (los n conjuntos de índices, uno por cada parámetro) de diferente cardinalidad* (m_k)*, con la restricción de que sólo se pueda utilizar un elemento de cada conjunto?*

Si los n conjuntos de índices, uno por cada parámetro, se etiquetan como J_i, con $0 \leq i \leq n-1$, el número total de grupos de n elementos viene dado por el producto de las cardinalidades:

$$|J| = \prod_{i=0}^{n-1} |J_i| \tag{3.1}$$

La expresión (3.1) muestra que el conjunto J puede generarse a partir de la suma directa de los J_i conjuntos. La forma directa de implementar esta suma, y generar todos los elementos de J, es utilizar una serie de n sumatorios anidados, uno para cada conjunto J_i, como puede observarse en la expresión (3.2).

$$\sum_{i_0}^{f_0} j_0 \cdot \sum_{i_1}^{f_1} j_1 \cdots \sum_{i_{n-1}}^{f_{n-1}} j_{n-1} \tag{3.2}$$

Aquí, las *is* representan valores iniciales y las *fs* valores finales. En este trabajo se propone, como novedad, el tratamiento de la expresión (3.2) mediante el operador NSS [28] introducido en el apartado 2.2. Así, se parte del hecho de que un sumatorio es un NSS de orden 1 para expresar (3.2) como una serie de NSS. A su vez, es posible contraer la expresión obtenida, ya que el producto de dos NSS es otro NSS [28]. De esta forma podemos describir el proceso completo con un único NSS. Así, el conjunto J puede generarse como:

$$\sum_{i_0}^{f_0} j_0 \cdot \sum_{i_1}^{f_1} j_1 \cdots \sum_{i_{n-1}}^{f_{n-1}} j_{n-1} = \sum_{1} (j_0 = i_0, f_0, s_0) \sum_{1} (j_1 = i_1, f_1, s_1) \cdots \sum_{1} (j_{n-1} = i_{n-1}, f_{n-1}, s_{n-1}) = \sum_{n} (j_0 \oplus j_1 \cdots \oplus j_{n-1} = i_0 \oplus i_1 \cdots \oplus i_{n-1}, f_0 \oplus f_1 \cdots \oplus f_{n-1}, s_0 \oplus s_1 \cdots \oplus s_{n-1}) \sum_{n} (\boldsymbol{j} = \boldsymbol{i}, \boldsymbol{f}, \boldsymbol{s}) \quad (3.3)$$

En la expresión (3.3) $\boldsymbol{i}, \boldsymbol{f}, \boldsymbol{s}$ y $\boldsymbol{j}$ son vectores de números enteros representando los valores iniciales, ***i*** *(en principio el vector nulo,* ***0****)*, y finales, ***f***, de los índices, sus incrementos, ***s*** *(en principio el vector unidad,* ***1****)*, y el conjunto de índices, ***j***. La introducción del operador NSS para la generación y tratamiento de problemas de barrido de parámetros proporciona una solución general ya que el número de sumatorios (bucles anidados en la implementación simple) pasa a ser un dato.

Queda por determinar la relación entre un vector ***j*** determinado y el correspondiente vector ***p*** de parámetros. Esta relación puede establecerse de la forma siguiente. Podemos considerar que todos los posibles valores de los parámetros conforman una matriz $\boldsymbol{\mathcal{P}}$ de *n x m,* donde *n* es el número de parámetros y *m* es el número de valores de cada parámetro por cada índice ***j***, véase (3.4).

$$\mathcal{P} = \begin{bmatrix} p_{00} & \cdots & p_{0\,(m-1)} \\ \vdots & \ddots & \vdots \\ p_{(n-1)\,0} & \cdots & p_{(n-1)(m-1)} \end{bmatrix} \quad (3.4)$$

Es importante indicar que *m* no tiene porqué ser igual para cada parámetro. Es decir, en el caso general y desde el punto de vista matemático, $\boldsymbol{\mathcal{P}}$ es una matriz rectangular de *n x m*, donde $m = max\ \{m_k,\ \forall k=0,\ n-1\}$. Lógicamente, las entradas $\boldsymbol{\mathcal{P}}_{ij}$ para $j>m_i$ son nulas. Computacionalmente, $\boldsymbol{\mathcal{P}}$ se puede representar como un vector (*array* monodimensional) de vectores donde el número de elementos de cada uno de estos últimos es independiente del de los otros. El primer vector tiene tantos elementos como parámetros, *n*, y cada uno de los segundos vectores tantos elementos como valores distintos haya del parámetro al que corresponda, m_i. Dada la matriz $\boldsymbol{\mathcal{P}}$, la correspondencia entre un vector ***j*** y el correspondiente vector ***p*** viene dada por,

$$\boldsymbol{p} = \{p_i |\ p_i = \mathcal{P}_{i,j_i}\ \forall\ i = 0, n-1\} \quad (3.5)$$

Dado un vector $\boldsymbol{p}$ el correspondiente valor v se obtiene por medio de la proyección (*mapping*) de $\boldsymbol{p}$ sobre el espacio de v : $v=v(\boldsymbol{p})$. Claramente, esta proyección es función en última instancia de $\boldsymbol{j}$ siendo dependiente del problema estudiado.

En general, en los problemas de barrido de parámetros cada miembro $\boldsymbol{p}$ del conjunto de valores $\boldsymbol{\mathcal{P}}$ se proyecta sobre un ejemplar del SWF. En la práctica, se realiza la proyección sobre uno o varios archivos de código y/o configuración del SWF. En total los ejemplares de SWF generados mediante el barrido de parámetros son: $\prod_{k=0}^{n-1} m_k$.

En determinados casos puede interesar llevar a cabo tratamientos parciales de un problema para lo que resulta necesario poder realizar barridos selectivos. Mediante el operador NSS es posible, utilizando el símbolo $\boldsymbol{L}$ (en esencia una delta de Kronecker lógica o LKD, *Logical Kronecker Delta*), llevar a cabo barridos parciales del espacio de parámetros inicial. $\boldsymbol{L}$ es un vector de dimensión n donde sus elementos simbolizan expresiones lógicas, véase [45]. Si se añade el vector $\boldsymbol{L}$ a la expresión (3.3) obtendríamos:

$$\sum_{n}(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s},\boldsymbol{L}) \tag{3.6}$$

La distinción entre barrido total o parcial se consigue gracias al contenido del vector lógico, $\boldsymbol{L}$, el cual actúa de la forma siguiente:

$$\begin{aligned} &\boldsymbol{if}\ \{\delta(j_k \neq l_k), \forall k = 0, n-1\} \\ &\boldsymbol{then}\ \{\boldsymbol{\mathcal{P}}, Barrido\ total\} \\ &\boldsymbol{else}\ \{subconjunto\ de\ \boldsymbol{\mathcal{P}}, Barrido\ parcial\} \end{aligned} \tag{3.7}$$

Es decir, si $j_k = l_k$ se excluyen de la proyección los $\boldsymbol{p}(j_k)$ correspondientes, lo que genera el barrido parcial.

Como se mencionó previamente, la opción más simple para implementar la expresión (3.3) es la utilización de n bucles anidados, uno por cada sumatorio. Esta es la opción habitual, pero tiene el inconveniente de que es necesario modificar el número de bucles (en la práctica recodificar) cada vez que el número de parámetros, cambia. Además, con el uso directo de la expresión (3.3) el barrido de parámetros es total y no se incluye, en la propia expresión, la posibilidad de llevar a cabo barridos parciales.

A continuación, se propone un algoritmo en pseudocódigo para generar el barrido de parámetros total o parcial basado en las expresiones (3.6) y (3.7):

Algoritmo PSNSS[1]

Algoritmo PSNSS (Secuencial)

```
// Inicialización de n y de los vectores i, f, s y l (dependiente del problema)
   Especificar número de parámetros, n
   Especificar vectores i, f, s y l

// Inicialización del vector j
   for k ← 0 to n − 1
      if (i(k) = l(k)) then i(k) ← i(k) + s(k)
       j(k) ← i(k)
   end_for

// Barrido (opcionalmente selectivo) de parámetros
    k ← n − 1
    while k ≥ 0 do
       if ( (j(k) − f(k)) * s(k) > 0) then
          j(k) ← i(k)
           k ← k − 1
       else
         Ejecutar SubWorkflow (n, j)
         k ← n − 1
       end_if
       if k ≥ 0 then
          j(k) ← j (k) + s(k)
          if (j(k) = l(k)) then j(k) ← j(k) + s(k)
       end_if
    end_while
end_algoritmo PSNSS
```

Figura 3.6 Pseudocódigo del algoritmo de barrido de parámetros basado en el operador NSS.

Es interesante destacar que en un problema de barrido de parámetros la proyección de un vector ***j*** sobre el correspondiente valor *v* es independiente de cualquier otra proyección para un vector ***j*** diferente. Desde el punto de vista concurrente esto define un problema de tipo "hiperparalelo" (*embarrasingly parallel*) que puede abordarse con un patrón de paralelismo de tareas [94]. Teniendo esto en cuenta, se propone a continuación una versión concurrente del algoritmo presentado en la Figura 3.6.

[1] PSNSS: Parameter Sweep Nested Summation Symbol

```
Algoritmo PSNSS (Concurrente)
// Inicialización de n y de los vectores i, f, s y l (dependiente del problema)
   Especificar número de parámetros, n
   Especificar vectores i, f, s y l

// Inicialización del vector j
   for k ← 0 to n − 1
      if (i(k) = l(k)) then i(k) ← i(k) + s(k)
       j(k) ← i(k)
   end_for

// Barrido (opcionalmente selectivo) de parámetros

    k ← n − 1
    Open P líneas de ejecución concurrente
      do en línea de ejecución única
            while k ≥ 0 do
               if ( (j(k) − f(k)) * s(k) > 0) then
                  j(k) ← i(k)
                  k ← k − 1
               else
                 do en línea de ejecución concurrente disponible
                    Ejecutar SubWorkflow (n, j)   // Crea una cola de tareas para cada línea
                  end_do_concurrente
                 k ← n − 1
               end_if
               if k ≥ 0 then
                  j(k) ← j (k) + s(k)
                  if (j(k) = l(k)) then j(k) ← j(k) + s(k)
               end_if
            end_while
      end_do_concurrente
    Close líneas de ejecución concurrente
end_algoritmo PSNSS
```

Figura 3.7 Pseudocódigo concurrente del barrido de parámetros basado en el operador NSS.

Los algoritmos presentados en las Figuras 3.6 y 3.7 están basados en la implementación del "*Generalized Nested Do Loop*" (GNDL) presentado en [28] como aplicación del NSS.

Llegados a este punto es importante considerar otro aspecto del problema que resulta de interés en la realización práctica de PSEs, cuando cada vector de índices ***j*** se corresponda con uno o varios archivos iniciales y uno o varios archivos resultado de la ejecución del PSE para el valor ***p(j)*** correspondiente. Se trata de organizar de forma sistemática estos archivos de manera que dado uno de ellos se pueda determinar el ***j*** correspondiente y viceversa. Este simple objetivo presenta varias ventajas. Primero, simplificaría la interpretación de los PSEs, al poder acceder a la información de forma sencilla, con un mecanismo de indexación. Por otro lado, simplificaría los procesos de recopilación

selectiva de información. Finalmente, pero en la práctica no menos importante, evitaría el uso de un único directorio para almacenar todos los archivos. Esta práctica habitual puede resultar problemática ya que existen límites al número de archivos por directorio en los sistemas de archivos.

Abordemos la solución del problema propuesto considerando la relación entre los diferentes índices de los parámetros y los vectores $\boldsymbol{j}$ generados. Para ello, denotemos como J_k el conjunto de los m_k índices del parámetro p_k:

$$J_k = \{i \mid 0 \leq i \leq m_k - 1\} \tag{3.8}$$

A su vez, denotemos como I_p el conjunto de elementos formado por todas las posibles p –tuplas formadas con los índices de los primeros p parámetros, $0 \leq p \leq n-1$. Así,

$$I_p = \left[(i, j, \ldots, l) \mid i \in J_0, j \in J_1 \ldots l \in J_{p-1}\right] \tag{3.9}$$

Evidentemente con, $I_0 = \emptyset$. Por otro lado, cualquier p-tupla puede obtenerse recursivamente como:

$$\begin{aligned} I_0 &= \emptyset \\ I_{p>0} &= \left\{\left(I_{p-1}, i\right) \mid 0 \leq i \leq m_{p-1}\right\} \end{aligned} \tag{3.10}$$

Lógicamente, el conjunto de n–tuplas, I_n, no es nada más que el conjunto de vectores de índices, J.

Considerando las diferentes tuplas, se construye ahora un nuevo conjunto, T, constituido por todos los elementos de los conjuntos I_0 a I_n. En este contexto, la relación recurrente (3.10) muestra que existe una relación entre parejas de tuplas, lo que define un nuevo conjunto E, esta vez de relaciones entre parejas de miembros del conjunto T. Claramente, los conjuntos T (entidades) y E (relaciones entre las entidades) definen un grafo, G, el cual presenta las siguientes propiedades:

a) **Proposición 1**. El grafo G es conexo.

Justificación: Se trata de comprobar que en el grafo G hay un camino que permite alcanzar cualquier par de elementos de G. La comprobación puede realizarse por inducción. Construyamos el grafo G como una serie de n pasos donde en cada uno de ellos añadimos (suma directa) uno de los conjuntos I_p con $0 \leq p \leq n-1$ y el correspondiente conjunto de aristas entre I_p e I_{p-1}, E_p: $G_p = G_{p-1} \oplus I_p \oplus E_p$. Partimos de la existencia de la 0-tupla la cual define G_0 y, de forma trivial, cumple la proposición 1. Por otro lado, la expresión (3.10) muestra que cualquier

conjunto I_p se obtiene exclusivamente del I_{p-1}. Lo mismo se cumple para los componentes de I_p, cada uno se obtiene exclusivamente a partir de uno de I_{p-1}. Es decir, existe una correspondencia biunívoca entre los elementos de I_p e I_{p-1}. Por lo tanto, si G_{p-1} era conexo el nuevo G_p es conexo pues cualquiera de los nuevos elementos añadidos, los que provienen de I_p, se puede alcanzar desde cualquier otro elemento de I_p atravesando la estructura conexa de G_{p-1}. Aplicando el principio de inducción, si G_0 es conexo también lo es G_1 y, por tanto, también G_2 y así sucesivamente hasta el grafo completo $G_{n-1}=G$.

b) **Proposición 2**. El grafo G es acíclico.

Justificación: La expresión (3.10) muestra que hay un orden en la creación de tuplas, obteniéndose la I_p exclusivamente a partir de la I_{p-1}. Por lo tanto, no hay ninguna conexión entre tuplas I_p e I_q con $|p-q|>1$. A su vez, la relación entre los elementos de I_p e I_{p-1} es biunívoca. Estas dos propiedades implican que no hay caminos que conecten ningún nodo del grafo consigo mismo, en otras palabras el grafo es acíclico.

c) **Proposición 3**. El grafo G es simple.

Justificación: Puesto que la relación entre los elementos de I_p e I_{p-1} es biunívoca no puede haber vértices en el grafo con aristas múltiples y, por tanto, el grafo es simple.

Se trata de un grafo conexo, acíclico y simple o lo que es lo mismo, de un árbol.

Para resolver el problema de la organización de archivos en el PSE, se propone en este trabajo generar en disco una estructura de subdirectorios que se corresponda con el grafo (árbol) definido para el conjunto, T, de tuplas. La relación (3.10) indica que tendríamos un nivel del árbol para cada conjunto de tuplas, I_p, con la Ø-tupla correspondiendo a la raíz del mismo. Las p-tuplas con $0 \leq p \leq n-1$ son nodos internos del árbol y se corresponderían con subdirectorios en el sistema de subdirectorios. Las n-tuplas son nodos externos (hojas) del árbol siendo a su vez los miembros del conjunto de vectores J y, por tanto, correspondiéndose con los archivos generados en el PSWF. Es importante destacar un par de propiedades de la proyección del árbol de tuplas sobre el sistema de subdirectorios:

a) Cada subdirectorio se corresponde biunívocamente con los índices de la tupla que representa.

b) Un vector de índices concreto, ***j***, indica la ruta desde el subdirectorio raíz hasta el archivo correspondiente a dicho vector ***j***.

De esta forma, se ha generado una estructura jerárquica de subdirectorios donde el número de archivos que se almacenan por subdirectorio en el último nivel es igual a la cardinalidad del conjunto J_{n-1} que corresponde a los índices del último parámetro.

Como ejemplo, la Figura 3.8 muestra el árbol de índices generado para el caso $n=m$ usando la terminología del operador NSS. Todos los nodos del árbol son subdirectorios salvo las hojas que son archivos, véase Figura 3.8. El número total de archivos generados en el ejemplo es de $\prod_{k=0}^{n-1} m_k = n^m$. Las etiquetas de los nodos del árbol se corresponden con los elementos del vector $j(n)$ que son los índices necesarios para el acceso a los valores de n.

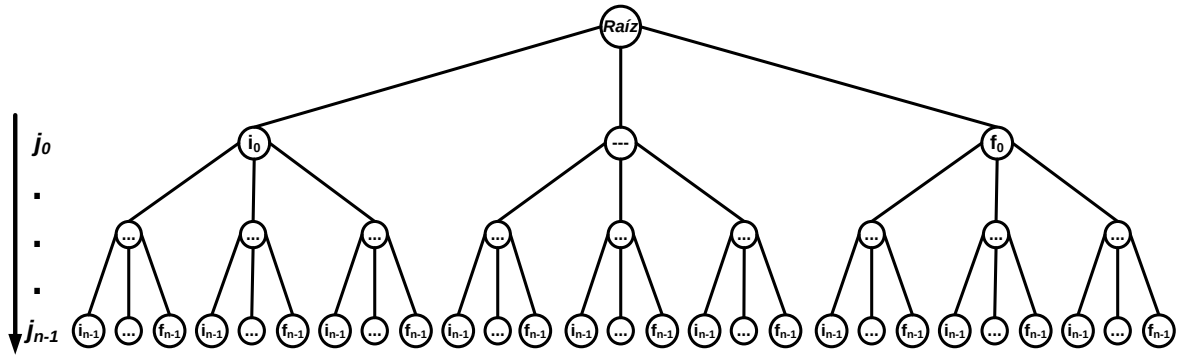

Figura 3.8 Proyección a disco del barrido de parámetros utilizando el operador NSS

Del ejemplo anterior es fácil determinar que, en el caso general, la aproximación propuesta se corresponde con un árbol con las siguientes propiedades:

a) Puesto que los índices de cualquier parámetro aparecen en orden creciente, es un árbol m-ario ordenado donde, como anteriormente, $m = max\{m_k, \forall k=0, n-1\}$.
b) La profundidad del árbol es n.
c) Todos los caminos de la raíz a las hojas tienen la misma longitud y ésta es igual a n.
d) El número total de hojas es: $\prod_{k=0}^{n-1} m_k$.

e) Todas las hojas del árbol tienen la misma profundidad. Es decir, el árbol está equilibrado.

Cada ejemplar del vector $\boldsymbol{j}$ se corresponde con un único camino del árbol, véase Figura 3.8, desde el nodo raíz hasta una de las hojas. En este camino, el primer índice de $\boldsymbol{j}$ identifica la 1-tupla (subdirectorio) de nivel 1 incluida en dicho camino. A su vez, el segundo índice de $\boldsymbol{j}$ identifica el subdirectorio (2-tupla) de nivel 2 incluido en el camino, y así sucesivamente.

Aquí, se propone usar esta serie de índices para etiquetar tanto los subdirectorios, correspondientes a las p-tuplas con $0 \leq p \leq n-1$, como los archivos, correspondientes a las n-tuplas. Así, a cada tupla (nodo en el árbol) se le asocia un identificador de archivo (*File Identifier*, FID) constituido por la secuencia de índices en el vector $\boldsymbol{j}$ correspondiente precedida por 0 (por el nodo raíz). Los distintos valores de los índices se separan por ".". Por ejemplo, para un vector j = (2, 3, 0, 1) el FID sería: 0.2.3.0.1. Hay una ventaja adicional de esta aproximación en el caso de estar usando el operador NSS en la versión concurrente mostrada en la Figura 3.7. La ventaja es que el árbol de subdirectorios resultante es independiente de la naturaleza no determinista del proceso.

A continuación, se ilustra lo anterior con un caso de uso concreto para n=m=3. Es decir, tres parámetros y tres valores por parámetro. Se trata de un caso concreto de la expresión (3.6). Aquí, denotemos por $\mathbf{n_i}$ el conjunto de valores del parámetro i, y como anteriormente, identificaremos como $\boldsymbol{ps}$ las combinaciones concretas de parámetros. Los índices que etiquetan los parámetros y gestionan su manejo seguirán la nomenclatura y significado usados en este apartado: vectores $\boldsymbol{j}$, $\boldsymbol{i}$, $\boldsymbol{f}$ y delta de Kronecker lógica $\boldsymbol{L}$. Así, tendremos las siguientes relaciones:

$\mathbf{n_0}$=(a,b,c); $\mathbf{n_1}$=(d,e,f); $\mathbf{n_2}$=(g,h,i); $i_0=i_1=i_2=0$; $f_0=f_1=f_2=2$; $l_1=l_2=l_3=\alpha$, con $\alpha \neq j_k$

Es decir, tenemos tres conjuntos ($\mathbf{n_0}$, $\mathbf{n_1}$ y $\mathbf{n_2}$) de valores de parámetros cuyos índices comienzan en 0 llegando hasta 2.

En un diagrama,

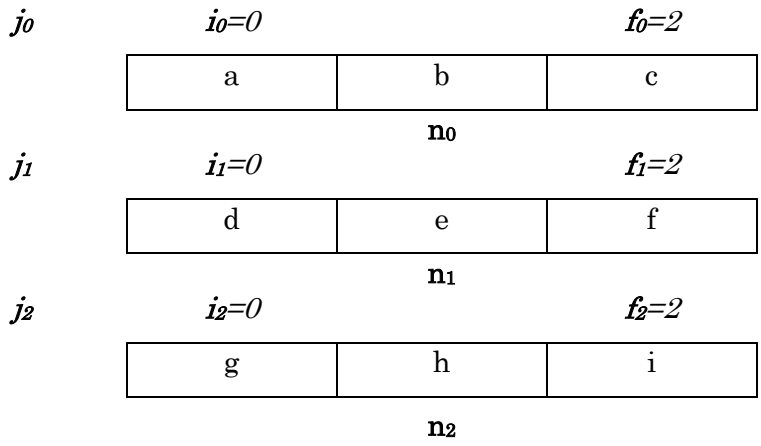

Figura 3.9 Visualización gráfica del ejemplo

Las combinaciones de parámetros específicas, ***p***, son función del vector de índices ***j***, ***p***=*p*(***j***). En este caso tendríamos,

p_0(a,d,g)=p(0,0,0);

p_1(a,d,h)=p(0,0,1);

…

p_8(a,f,i)=p(0,2,2);

p_9(b,d,g)=p(1,0,0);

…

p_{25}(c,f,h)=p(2,2,1);

p_{26}(c,f,i)=p(2,2,2),

que son las 27 posibles combinaciones de índices dadas por $\prod_{k=0}^{n-1} m_k = 3^3 = 27$.

En la Figura 3.10, se muestra el caso de uso anterior, proyectado sobre el árbol generado mediante el algoritmo NSS de la Figura 3.6. Cada miembro ***j*** del conjunto de índices se corresponde con una hoja del árbol, véase Figura 3.10, y da lugar a un archivo en disco cuyo nombre se corresponde con el FID (excluyendo los puntos y el nodo raíz). Por ejemplo, para el caso ***j***=*(0,1,2)* el archivo se nombraría como "012" y su ruta en el árbol de directorios sería "0.0.1.2".

Como puede observarse, tanto el nombre del archivo como su ruta en el árbol son función de ***j***. El operador NSS, a través del vector ***j***, nos permite la generación y recorrido del árbol.

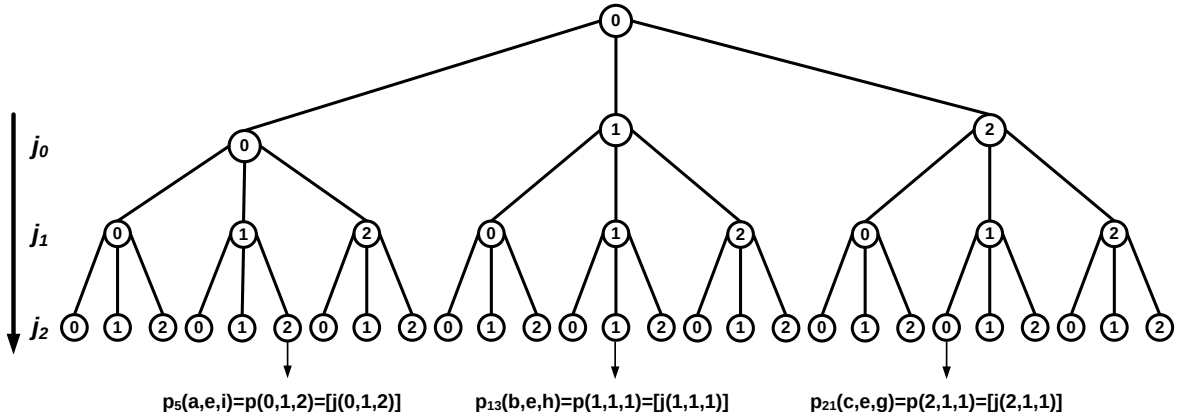

Figura 3.10 Árbol correspondiente al ejemplo con n=m=3 y $l_1=l_2=l_3=\alpha$

Para el caso considerado en la Figura 3.10, se ha utilizado el vector L con $l_1=l_2=l_3=\alpha$ donde α, recordemos, no es coincidente con ningún índice lo que origina un barrido de parámetros total. Si se utiliza ***j*** junto al vector ***L***, con α coincidiendo con algún índice, es posible generar y recorrer el árbol de manera parcial o bien podar algunas de sus ramas. A continuación, se muestra un ejemplo de utilización de ***L*** para excluir una combinación de cada una de los índices del vector ***j***. Por ejemplo, en la Figura 3.11 se muestra el árbol que se obtiene aplicando el algoritmo de la Figura 3.6 al caso actual con $l_0=1$ y $l_1=l_2=\alpha$.

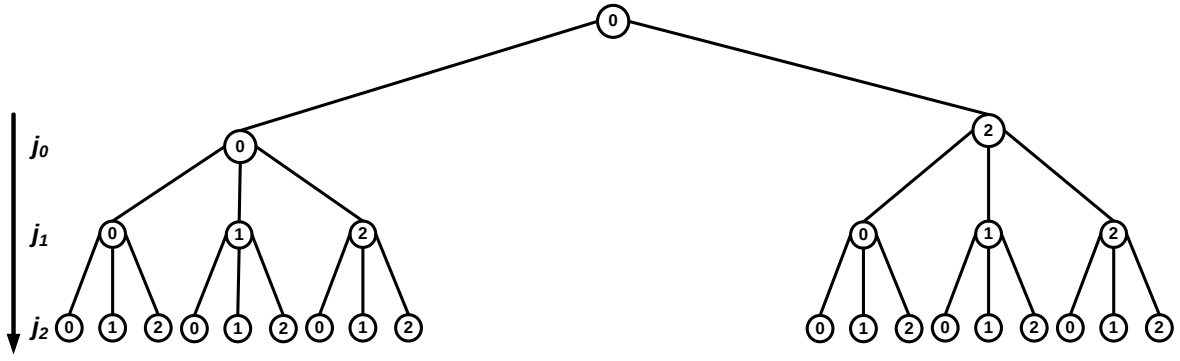

Figura 3.11 Árbol correspondiente al ejemplo con L ≠ α, con $l_0=1$ y $l_1=l_2=\alpha$

La elección de valores de los componentes de ***L*** que muestra la Figura 3.11, provoca la exclusión del barrido (poda en el árbol) de todas las 3-tuplas que incluyan 1 como primer índice, esquemáticamente: p_k(b,?,?)=*[j(1,?,?)]*, donde el símbolo ? corresponde al habitual carácter comodín. En total se excluyen nueve posibilidades. Aquí, hemos provocado la poda de la rama completa que sale de la raíz con etiqueta igual a 1.

La Figura 3.12 muestra otro caso de ejemplo con $l_0=l_2=\alpha$ y $l_1=1$. Es decir, la exclusión va a afectar a los vectores $\boldsymbol{j}$ cuyo segundo índice sea igual a 1. Ahora, se han eliminado todas las ramas que parten del segundo nivel del árbol con índice igual a 1.

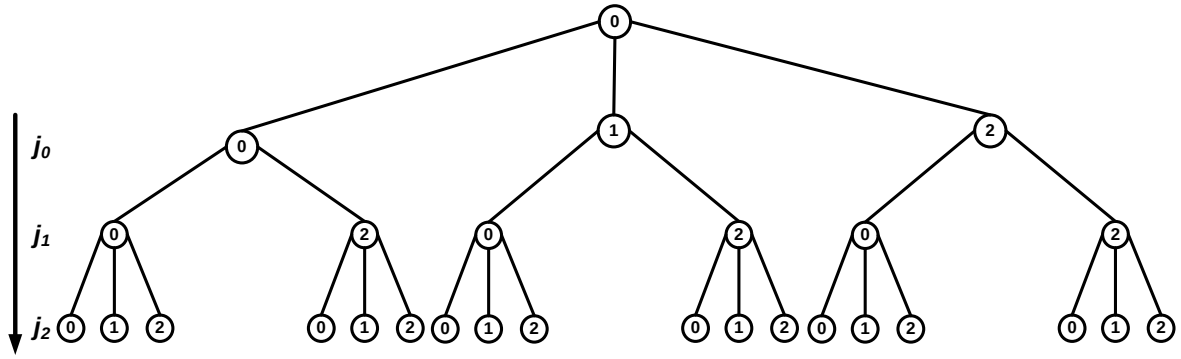

Figura 3.12 Árbol correspondiente al ejemplo con L ≠ α, con $l_0=l_2=\alpha$ y $l_1=1$

Por último, en la Figura 3.13 se muestra un nuevo ejemplo con $l_0=l_1=\alpha$ y $l_2=1$. Aquí, la exclusión afecta a los vectores $\boldsymbol{j}$ cuyo tercer índice sea igual a 1. Se eliminan, por tanto, todas las ramas que parten del tercer nivel del árbol con índice igual a 1.

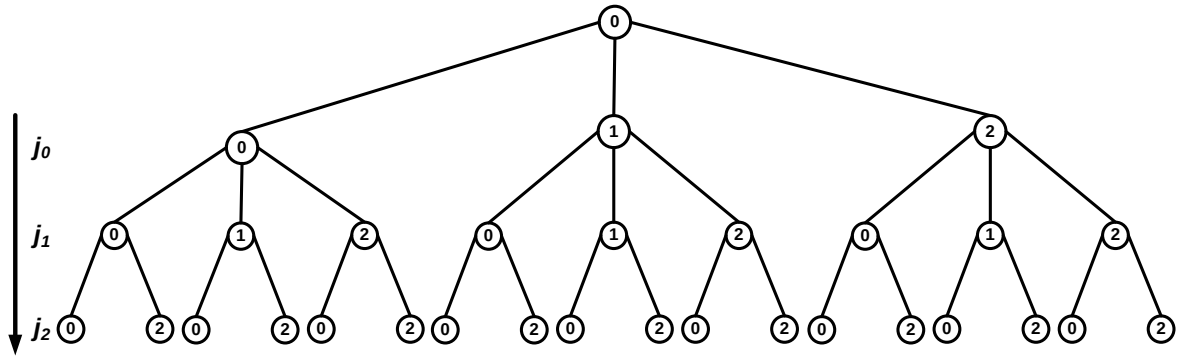

Figura 3.13 Árbol correspondiente al ejemplo con L ≠ α, con $l_0=l_1=\alpha$ y $l_2=1$

Usando el tratamiento presentado en este apartado, es posible llevar a cabo tanto la generación como la recolección de resultados de cualquier barrido de parámetros. A tal efecto, debemos especificar realizaciones concretas del algoritmo genérico recogido en las Figuras 3.6 y 3.7. A continuación se proponen sendos algoritmos para resolver la generación y la recolección del barrido de parámetros.

1. ***Generación*** (*Generator*, G) del barrido total o parcial sobre n parámetros por m_k valores por cada parámetro. A tal efecto, se organizan los archivos en un árbol de directorios con lo que se evitan los problemas que originan los límites que imponen los sistemas de archivos. Esta tarea se consigue sustituyendo la

sentencia *SubWorkflow*(n, ***j***) del algoritmo propuesto en las Figuras 3.6 y 3.7 por el siguiente algoritmo:

```
Algoritmo G(n, j)
  if ∄ dir WFname then      // dir equivale a directorio
    crear_dir WFname
  end_if
  cambiar_a_dir WFName
  for k=0 to n − 1
     if ∄ dir j(k) then
       crear_dir jk
     end_if
    cambiar_a_dir jk
  end_for
  // El FileName sigue la nomenclatura indicada en la página 67, último párrafo
  componer(WFFileName = j(0)j(1), ..., j(n-1) ; WFTemplate; p(j) )
end_Algoritmo_G
```

Figura 3.14 Algoritmo *Generator* (G)

En la Figura 3.14, WF_{Name} se refiere al nombre del proyecto, SWF, asignado por el equipo de investigación (se corresponde con el nodo raíz del árbol de directorios, véase Figura 3.8). A su vez, $WF_{Template}$ se corresponde con los archivos de datos y configuración del problema. Finalmente, el subalgoritmo "componer" implica obtener el $WF_{FileName}$ a partir de la integración de la plantilla $WF_{Template}$ y el vector $p=p(j)$ cuyas componentes son los valores de los n parámetros.

2. ***Recolección*** (*Collector*, **C**) de los archivos iniciales (datos, configuración, …) o finales (resultados), de forma ordenada, para su tratamiento. Esta tarea se consigue sustituyendo la sentencia *SubWorkflow*(n, ***j***) del algoritmo propuesto en las Figuras 3.6 y 3.7 por el siguiente proceso:

```
Algoritmo C(n, j)
  cambiar_a_dir WFName
  for k=0 to n − 1
     cambiar_a_dir jk
  end_for
  procesar WFFileName
end_Algoritmo_C
```

Figura 3.15 Algoritmo *Collector* (C)

En la Figura 3.15, WF_{Name} y $WF_{FileName}$ tienen idéntico significado al indicado en el algoritmo anterior.

La aproximación aquí propuesta para el tratamiento del problema de barrido de parámetros cumple con los requisitos de generalización planteados en el segundo objetivo parcial de este trabajo, véase apartado 1.2.

En primer lugar, el algoritmo propuesto, véanse Figuras 3.6 y 3.7, gestiona un número arbitrario de parámetros y un número arbitrario de valores de los parámetros, ya que son datos de entrada.

En segundo lugar, permite la realización de simulaciones sucesivas sobre distintos subconjuntos del barrido de parámetros global previa modificación del vector lógico ***L***.

En tercer lugar, la etiqueta *SubWorkflow*(n, *j*) del algoritmo es genérica y puede ser sustituida por cualquier otra, u otras, en función de las necesidades del problema. Esto permite el uso del modelo propuesto en cualquier campo de conocimiento.

Por último, se enumeran las ventajas que ofrece la utilización del modelo PSWF frente a una aproximación tradicional:

1. Independencia del modelo frente al número de parámetros, n.
2. Minimización del tiempo de realización de la simulación.
3. Minimización de errores con la aplicación del modelo.
4. Escalabilidad de los problemas. La aplicación del modelo en un sistema más potente reduce el tiempo empleado en la simulación.
5. Aplicación del paralelismo de forma inherente.

Las ventajas anteriores permiten definir una serie de métricas de éxito (en esencia de rendimiento) del modelo como son:

1. Tiempo total invertido en la simulación del PSWF.
2. Número total de tareas finalizadas en el PSWF.
3. Tiempo promedio por tarea.
4. Número total de tareas ejecutadas en cada recurso.
5. Tiempos por tarea: tiempo de ejecución, de CPU y *overhead*.

4 Validación del Modelo de Tratamiento de PSEs

En este capítulo se aborda el estudio de la factibilidad del proceso de modelado de *workflow* presentado en el capítulo anterior para el problema de barrido de parámetros. A tal efecto, usaremos un caso de barrido de parámetros real: la obtención de la hipersuperficie de energía potencial molecular. En concreto, se tratará el problema usando como ejemplo la molécula de acetona, molécula de interés astrofísico. Para ello, aplicaremos las distintas fases del ciclo de vida del PSWF (*Parameter Sweep Scientific Workflow*) introducido en el capítulo 3.

A efectos de presentación este capítulo queda organizado de la forma siguiente. Primero, antes de abordar el desarrollo del *workflow* se introduce el problema que se va a utilizar como caso de prueba. En segundo lugar, se consideran las dos primeras etapas del ciclo de vida: planteamiento del problema y diseño del *workflow*. A continuación, se aborda la viabilidad de la tercera etapa del ciclo de vida como función del número de casos generados en el barrido de parámetros y del número de ficheros creado. Posteriormente, se comprueba la factibilidad de este tipo de estudios sobre una infraestructura distribuida real. A continuación, se considera la etapa de recolección e integración de resultados. Finalmente, se considera el rendimiento de la versión concurrente del algoritmo PSNSS (*Parameter Sweep Nested Summation Symbol*).

4.1 La hipersuperficie de energía potencial molecular

El desarrollo de un hamiltoniano rovibracional anarmónico para moléculas de tamaño arbitrario, que considere múltiples grados internos de libertad, que describa el acoplamiento entre modos de vibración y que considere la interacción rotación-vibración presenta un gran interés desde el punto de vista Físico-Molecular. El elemento clave es la determinación de una función de energía potencial para el movimiento nuclear, así como la variación de la estructura molecular asociada a dicha función. Esta información permite construir un hamiltoniano para el movimiento nuclear. La resolución de la ecuación de Schrödinger correspondiente proporciona la información buscada en este tipo de estudios. Como ejemplo podemos indicar la identificación y caracterización de moléculas de interés astrofísico y astrobiológico a través de su patrón espectroscópico.

La función potencial buscada depende de varios parámetros como son las distancias y los ángulos planos y diedros necesarios para definir la estructura molecular. Esta función potencial es una función multidimensional, por lo que se la denomina habitualmente hipersuperficie de energía potencial molecular. En el marco de la aproximación de Born-Oppenheimer, es posible realizar un mapeo punto a punto de dicha hipersuperficie usando los resultados de energía total de cálculos de estructura electrónica molecular para diferentes configuraciones de la estructura molecular [95]. A tal efecto, sería interesante poder realizar de forma automática una exploración masiva de la hipersuperficie de energía potencial como función de la estructura. En términos computacionales, se trata de trabajar a un nivel de abstracción alto donde los cálculos de estructura electrónica no son el fin, sino el medio para construir nuevos modelos de movimiento nuclear. Dada la independencia de datos entre los cálculos de estructura electrónica para diferentes estructuras, el proceso admite una aproximación de tipo barrido de parámetros. Por tanto, éste es un caso donde es posible aplicar la metodología desarrollada en este trabajo.

4.2 Planteamiento del problema y diseño del *Workflow*

En este apartado se abordan las dos primeras etapas del ciclo de vida presentado en la Figura 3.1: elección de parámetros y diseño del *workflow* propiamente dicho. Como se indicó previamente, se trata de dos etapas no automatizables que dependen de la naturaleza del problema. Consideremos cada una de ellas en orden.

Elección de parámetros

Se trata de la etapa etiquetada como {1} en el ciclo de vida, véase Figura 3.1. En esta etapa, se seleccionan los parámetros estructurales (por ejemplo, distancias, ángulos planos y ángulos diedros) α_1, α_2, …, α_n, de la molécula elegida, sobre los que posteriormente se aplicará el barrido de parámetros. Esta es una tarea dependiente del problema y cada estudio concreto usa una combinación específica de parámetros.

En el caso de prueba usado en este ejemplo, molécula de acetona, la Figura 4.1 muestra las coordenadas internas (parámetros) consideradas en el estudio. En el ejemplo, nos centramos en describir los movimientos de "*bending*" y torsión asociados a los grupos metilo. Para ello se seleccionan como parámetros los dos ángulos planos (*bending*) α, β y los dos diedros (torsión) θ_1 y θ_2.

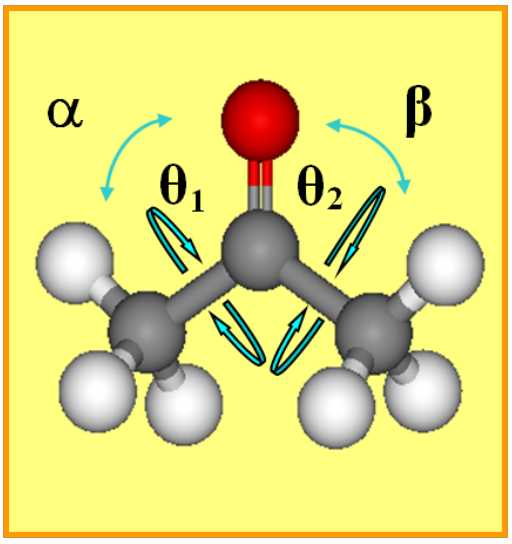

Figura 4.1 Molécula de acetona

Diseño del *workflow*

Se trata de la segunda etapa, etiquetada como {2} en la Figura 3.1, del ciclo de vida. Se aborda aquí el diseño de un *workflow*, denominado PSHYP (*Parameter Sweep Hypersurface*), para la resolución automática del problema del tratamiento de la hipersuperficie de energía potencial molecular. Para ello, en primer lugar se lleva a cabo una revisión del repositorio de *workflows* (*Worflow* Repositorio, véase Figura 3.1). En este caso, no existe ningún *workflow* previo que cumpla los requisitos aquí exigidos. Por lo tanto, se pasa a definir uno que permita el tratamiento del problema considerado. El resultado se presenta en la Figura 4.2 y los diferentes componentes del *workflow* se detallan a continuación.

La Figura 4.2 muestra que la primera tarea es la denominada OPT. Se trata de la determinación de la estructura inicial del sistema sobre la que se aplicará el proceso de barrido de parámetros. En el caso que nos ocupa, hipersuperficie de energía potencial molecular, se trata de una optimización de geometría de la molécula, hasta alcanzar un estado de energía potencial mínima (equilibrio). Este es un proceso estándar de cálculo de estructura electrónica molecular (ab initio). Para la realización de estos cálculos existen diferentes paquetes software de estructura electrónica molecular. Ejemplos concretos, de libre distribución, son GAMESS [96] o NWChem [97]. Como puede verse en la Figura 4.2, el resultado de la optimización molecular se usa para definir una plantilla (*template*, $WF_{Template}$ en la Figura 4.2) sobre la cual se sustituirán los diferentes valores de los parámetros generados en el proceso de barrido.

La siguiente etapa del *workflow* es justamente la generación de las estructuras moleculares (módulo GENHYP en la Figura 4.2) resultado del proceso de barrido de parámetros. Aquí, se obtiene un conjunto de estructuras moleculares que definen el *mapping* deseado de la hipersuperficie de energía potencial de la molécula. Cada una de estas estructuras se corresponde con un archivo de configuración para las posteriores ejecuciones del paquete de estructura electrónica molecular. Dada la falta de un módulo genérico para esta tarea se desarrollará uno aplicando los algoritmos recogidos en la Figuras 3.6 y 3.14 (*Generator*). En esta etapa se implementa, mediante el modelo de programación seleccionado en {2}, el algoritmo PSNSS, propuesto en el capítulo 3, como metodología general para la obtención del barrido total o parcial de parámetros. También, como resultado de esta fase se crea el árbol de directorios producido por el algoritmo PSNSS. Cada hoja del árbol se corresponderá con un archivo fruto de la

integración de la plantilla "$WF_{Template}$", véase Figura 4.2, con un conjunto concreto de los α_i ángulos elegidos como parámetros y como resultado se obtendrán los "mol1.inp, …, molN.inp" mostrados en la Figura 4.2.

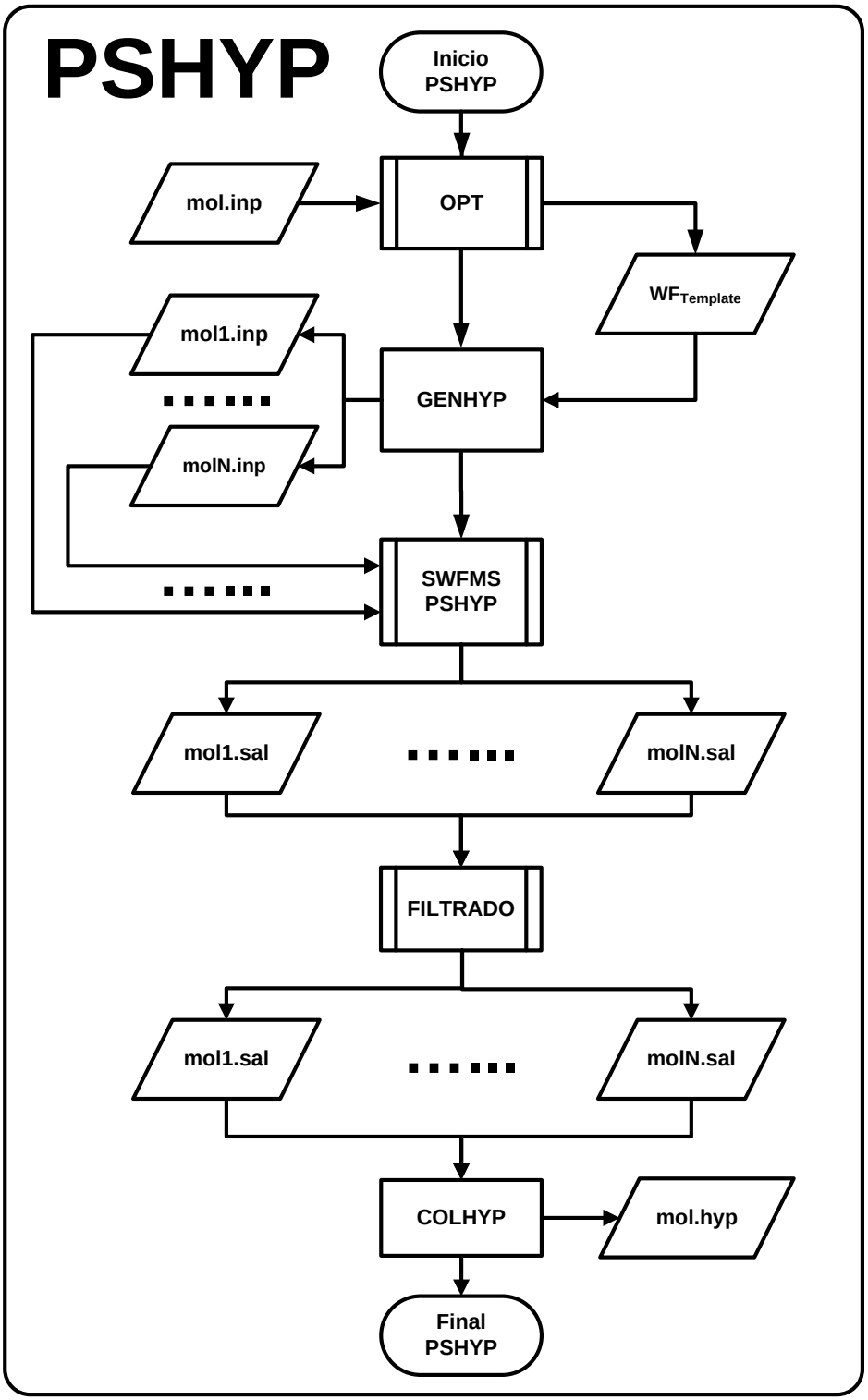

Figura 4.2 Esquematización, en forma de diagrama de flujo, del *workflow* PSHYP.

La siguiente etapa es la ejecución de los cálculos de estructura electrónica (SWFMS PSHYP, véase Figura 4.2) y posterior post-procesamiento (FILTRADO, véase Figura 4.2) de los resultados, uno por cada estructura generada en el proceso previo. En este paso el sistema gestor de *workflows* se encarga del despliegue de las tareas que forman el PSHYP sobre la infraestructura de computación, véase {4} en Figura 3.1. Las diferentes tareas asociadas a esta etapa se realizan concurrentemente sobre los diferentes recursos del sistema distribuido.

La última tarea representada en la Figura 4.2 es la integración (COLHYP) de los datos obtenidos en la etapa previa y se corresponde, en el ciclo de vida de la Figura 3.1, con {5}. El hecho de que cada resultado experimente un post-procesamiento en los recursos locales agiliza el proceso de integración, pues sólo la información necesaria es devuelta desde los recursos locales. Con la información obtenida, el equipo de investigación puede abordar una etapa de análisis, con apoyo o no de herramientas automáticas y, en función del mismo planificar una nueva simulación.

Como parte de esta misma segunda etapa del ciclo de vida PSWF, etiquetada como {2} en la Figura 3.1, en que nos encontramos debemos concretar el modelo de arquitectura que usaremos. Usando como base el modelo PSWF propuesto en la Figura 3.4 del capítulo 3, la Figura 4.3 muestra la arquitectura concreta propuesta en este trabajo para la obtención automática de la hipersuperficie de energía potencial molecular. Considerando las capas de abajo hacia arriba tenemos:

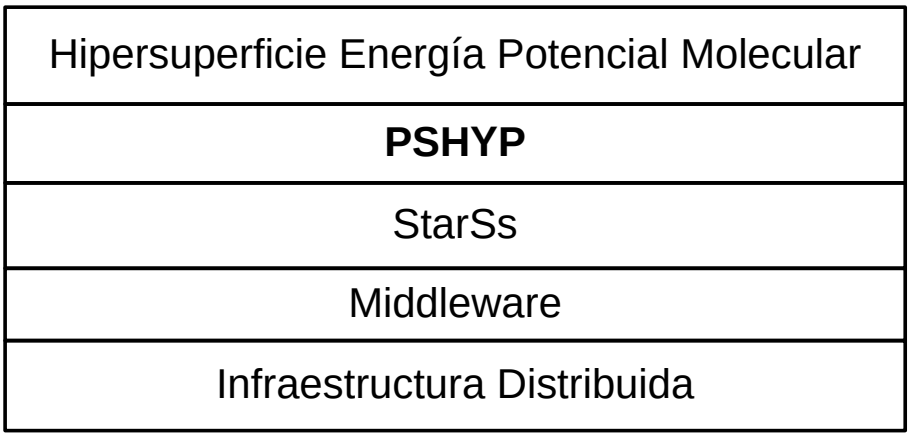

Figura 4.3 Arquitectura PSHYP

- En primer lugar, la infraestructura distribuida usada es un *Grid* de computadores. Se ha seleccionado un *Grid* por ser el sistema donde la heterogeneidad en dispositivos y redes es mayor. Esto permite contrastar la eficiencia del proceso aquí propuesto en condiciones poco favorables. En la Figura 4.4 puede verse la topología del sistema *Grid* construido para la ejecución de la prueba actual. Está formada por cinco nodos. Tres de ellos están geográficamente localizados en Ciudad Real y pertenecen al grupo de Química Computacional y Computación de Alto Rendimiento de la Universidad de Castilla La-Mancha (QCyCAR-UCLM), el cuarto está localizado en Puebla (México) y pertenece al Laboratorio de Química Teórica de la Benemérita Universidad Autónoma de Puebla (LQT-BUAP). El quinto nodo, está localizado geográficamente en Barcelona, en el *Barcelona Supercomputing Center* (Centro Nacional de Supercomputación, BSC-CNS).

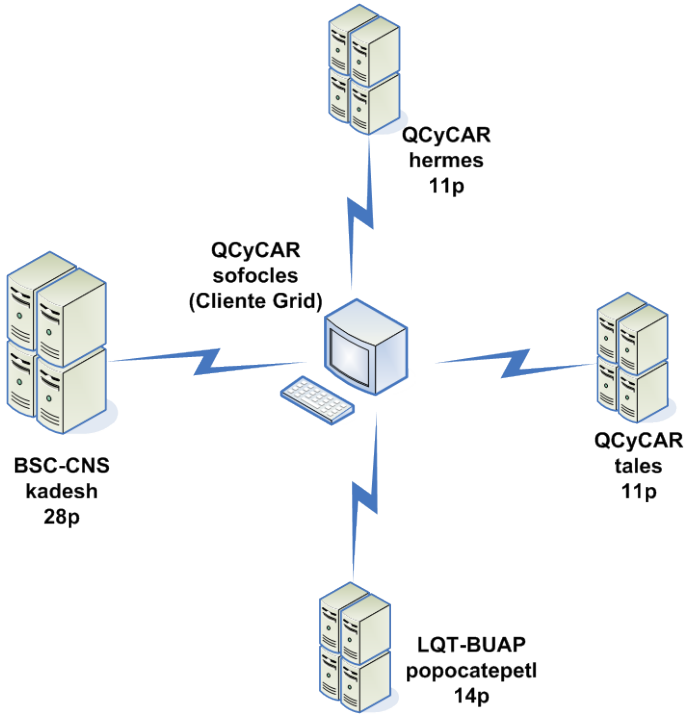

Figura 4.4 Topología *Grid* para el caso de la acetona

- En la segunda capa, como *core middleware* se utilizará *Globus Toolkit* pre-WS de entre los propuestos en el apartado 3.2 de este trabajo.
- En tercer lugar, debemos especificar el sistema de gestión de *workflows* usado. En este caso, se ha elegido la familia de modelos StarSs. La elección responde a los siguientes motivos:

1) La familia StarSs proporciona modelos de programación para los sistemas distribuidos más habituales como son *clusters* (ClusterSs) y *grids* (GRIDSs y COMPSs) de computadores.

2) El modelo de programación no cambia aunque se elija una infraestructura de computación diferente.

3) El modelo facilita la reutilización de algoritmos secuenciales.

4) Este autor, y el equipo de investigación al que pertenece colaboran con el equipo del BSC responsable del desarrollo de la familia StarSs.

Perteneciente a la familia StarSs, se eligió GRIDSs como SWFMS junto con el lenguaje de programación C/C++, en su versión para el *middleware grid Globus Toolkit* pre-WS. Como se trata de una arquitectura de capas, véase la Figura 4.3, la infraestructura, al ser el nivel inferior, condiciona la elección de las distintas entidades en los niveles superiores.

- En la cuarta capa tenemos el PSWF usado. Aquí, se trata del *workflow* PSHYP previamente descrito. Para la realización de todos los cálculos de estructura electrónica molecular se ha utilizado el paquete de estructura electrónica GAMESS [96].
- Finalmente, la última capa define el problema que se pretende abordar. En este caso, la obtención de la hipersuperficie de energía potencial molecular.

4.3 Generación del barrido de parámetros

Abordamos ahora la tercera etapa del ciclo de vida PSWF, ver Figura 3.1. El objetivo aquí es la generación de los ficheros que representan los datos para los distintos ejemplares del problema con valores de parámetros diferentes.

En nuestro caso, el punto de partida es la obtención de la estructura de energía mínima que se usará como referencia para el barrido de parámetros. Dicha estructura se obtiene a partir de una geometría inicial por un proceso de minimización multidimensional al nivel de teoría MP2/6-311G (2d, p) [98]. La estructura más estable se encuentra para valores de los diedros $\theta_1=\theta_2=0^o$. Para los ángulos planos α y β se obtienen valores de 121.8º. Para la generación del barrido de parámetros se han considerado los siguientes incrementos y límites para las cuatro coordenadas consideradas. Primero, los valores de los ángulos α y β se generan con variaciones de 1º entre 120º y 124º. Segundo, los valores de θ_1 y θ_2 se obtienen con incrementos de 20º entre 0º y 120º para θ_1 y, dada la simetría del problema, con similares incrementos para θ_2 con la restricción $\theta_1 \geq \theta_2$ cuando $\alpha=\beta$. De esta forma se obtienen 1120 estructuras diferentes.

A continuación, se muestra la implementación en C/C++ del algoritmo PSNSS (*Parameter Sweep Nested Summation Symbol*) propuesto en el capítulo 3, véase Figura 3.6, y utilizado para la generación del barrido de parámetros de la molécula de acetona.

```
void PSNSS (int n, int *f, void (*function)(int n, int *j, double **param)) {
        int *j;  int k;
        j=new int[n];              // Creación del array de índices
        for (k=0; k<n; k++) {      // Inicialización del array de índices j
           j[k]=i[k];
        }
        k=n-1;
        while (k>=0) {             // Operador NSS
            if (j[k]>f[k]) {
              j[k]=i[k];
              k--;
            }
            else {    (*function)(n, j, param);  // Invocar a G(n, j, param) {*}
              k=n-1;
            }
            if (k>=0) {
               j[k]+=s[k];
            }
        } //Final bucle While
    } //Final función PSNSS
```

Figura 4.5 Implementación en C/C++ del algoritmo PSNSS, véase Figura 3.6, para i=0 y s=1

El puntero a función, véase la línea marcada con {*} en la Figura 4.5, se utiliza para invocar a la función G(n, j, param) que representa la implementación en C/C++ del algoritmo propuesto en la Figura 3.14. Cada llamada a la función G(n, ***j***, param), véase Figura 4.6, genera y/o recorre una rama del árbol de directorios y al llegar al nivel inferior (n) invocar a la función *componerAcetona*(n, j, param), cuya implementación en C/C++ se incluye en el apéndice A de este trabajo, que es la encargada de obtener un archivo a partir del $WF_{Template}$ y un ejemplar de cada uno de los parámetros (param) indexados por el vector de índices (***j***).

```
void G(int n, int *j, double **param ){
   string WF("acetona"); // Creación e inicialización de variables
   string cd, command;
   struct stat st;
   char path[256];

   if (stat(WF.c_str(),&st)!=0){                          //Crear, si no existe, el WF
      getcwd(path, 255);
      command+=path;
      command+="/";
      command+=WF;
      mkdir(command.c_str(),S_IRWXU);
      command.clear();
   }
   getcwd(path, 255);                                     // Acceso al directorio WF
   command+=path; command+="/"; command+=WF;
   chdir(command.c_str());
   getcwd(path, 255);

   for (int i=0; i<n; i++){            // Creación de la rama del árbol correspondiente a j
      cd=IntToString(j[i]);
      if (stat(cd.c_str(),&st)!=0){
         command.clear();
         command+=path; command+="/"; command+=cd;
         mkdir(command.c_str(),S_IRWXU);
      }
      command.clear();
      command+=path; command+="/"; command+=cd;
      chdir(command.c_str());
      cd.clear();
   } // Final del bucle for
   componerAcetona(n, j, param)
}  //Final de la función G(n, j)
```

Figura 4.6 Implementación en C/C++ del algoritmo G(n, j), véase Figura 3.14

La Figura 4.7 muestra una comparativa entre número de archivos generados mediante la función PSNSS, véase Figura 4.5, y el tiempo invertido en la generación dependiendo de los incrementos elegidos para los ángulos θ_1 y θ_2. Como era de esperar, si el barrido de parámetros origina un número mayor de archivos; el tiempo invertido será mayor.

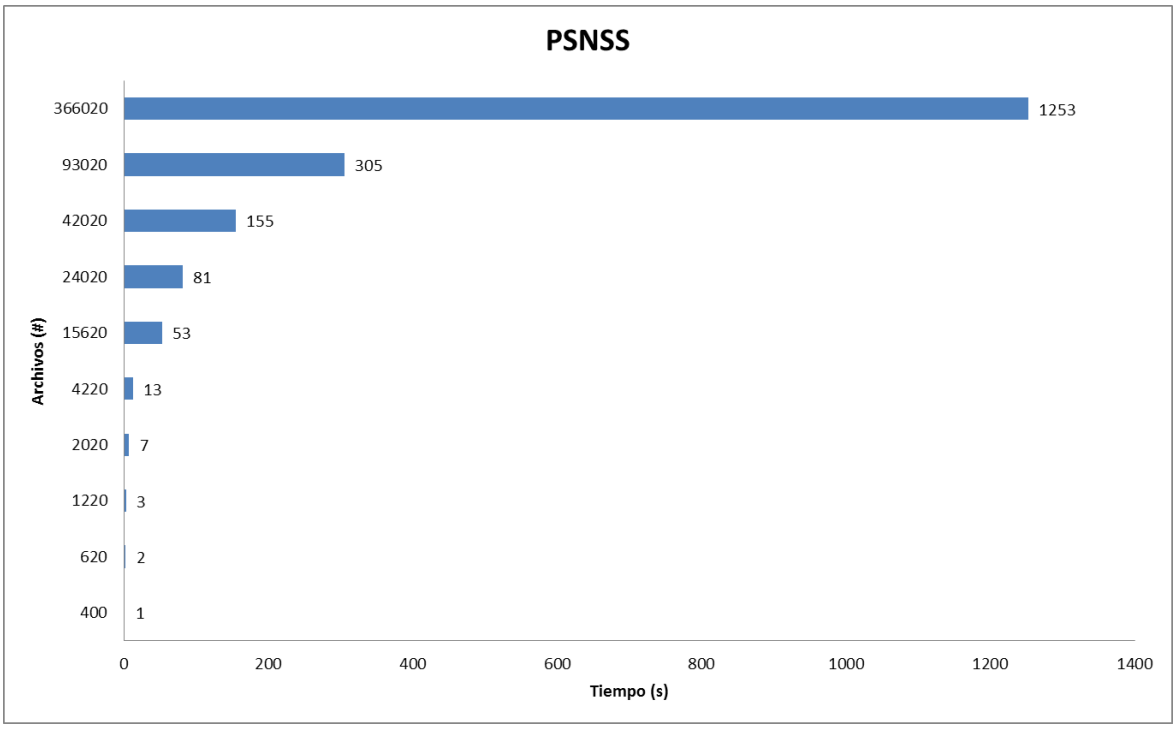

Figura 4.7 Validación del algoritmo PSNSS (encima de cada punto se muestra el tiempo exacto)

En este punto cabe plantearse la utilización de la versión concurrente del algoritmo propuesto en la Figura 3.6, véase Figura 3.7, con el fin de mejorar los tiempos obtenidos cuando el número de archivos, correspondientes al barrido de parámetros, crece. En nuestro caso concreto, obtención de la hipersuperficie de energía potencial, dicha aproximación no ofrece ninguna mejora, en las condiciones en las que se ha aplicado el modelo, con respecto al enfoque secuencial presentado en este apartado.

Para justificar la afirmación anterior, es necesario llevar a cabo un análisis detallado de la función G(n, j, param), véase la Figura 4.6. La función G podemos dividirla en dos fases: en primer lugar, genera y/o recorre una rama del árbol de directorios y en segundo lugar llama a la función *componerAcetona*, que genera y almacena un archivo a partir del $WF_{Template}$ y un ejemplar de param[*j*]; tanto la generación o recorrido de la rama del árbol como la generación y almacenamiento del archivo implican tiempo en operaciones de E/S (acceso a disco); el acceso a disco es una operación secuencial (no es posible la lectura–escritura de más de un archivo a la vez) que imposibilita mejoras aplicando concurrencia. La aproximación concurrente será adecuada en aquellos problemas de barrido de parámetros que impliquen un consumo de tiempo CPU asociado a la función *componer*, véase Figura 4.6, de tal forma que se cumpla la relación $T_{CPU} >>> T_{E/S}$ para la

función *componer* lo que hará que la concurrencia no se invierta exclusivamente en operación de E/S.

4.4 Ejecución del barrido de parámetros

Aquí se considera la cuarta etapa del ciclo de vida PSWF, ver Figura 3.1.

Para la ejecución del barrido de parámetros se han usado dos configuraciones diferentes del sistema *Grid*. En la primera, véase Figura 4.8 Caso a), se utilizan 8 procesadores en cada cluster del *Grid*, por lo tanto son 32 procesadores en total. En la segunda, véase Figura 4.8 Caso b), se usan un total de 64 procesadores distribuidos de la siguiente forma: 11 en Tales, 11 en Hermes, 14 en Popocatepetl y 28 en Kadesh. Los resultados se encuentran recogidos en la Figura 4.8. Los datos de la tabla muestran que incluso con tan solo 32 procesadores el tiempo necesario para el tratamiento de los 1120 casos considerados es más que razonable (31 horas y 15 minutos). Cuando doblamos el número de procesadores, el tiempo se reduce aproximadamente a la mitad (un factor de 0.55 concretamente) a pesar de la heterogeneidad física y lógica del sistema. Este hecho puede explicarse cuantitativamente de la siguiente forma:

	Caso a)	Caso b)
Tales	310	223
Hermes	290	218
Popo	286	262
Kadesh	234	417
Tiempo total (s)	112500	61500

Figura 4.8 Casos de uso para PSHYP

consideremos un sistema distribuido donde uno de los nodos actúa como *master* y el resto actúan como *workers*. El tiempo total, t_n, usado por un proceso concurrente que se ejecute en los n *workers* viene definido por el coste de ejecución más el coste de comunicación. En este contexto, los diferentes factores que afectan el rendimiento en un sistema distribuido han sido incluidos por Le y Rejeb [99] en un modelo de rendimiento LogGP modificado. Con este modelo, t_n puede expresarse como,

$$t_n = o_m + L + G.S + o_w + h_m + h_w + t_{Seq} + t_p \quad (4.1)$$

dónde o_m representa el *overhead* del *master*, o_w la sobrecarga de los *workers* derivada de las actividades de comunicación de los procesadores, L es la latencia de propagación, G es el tiempo de transmisión de una unidad del mensaje y S es la longitud del mensaje.

Ambos, L y G se pueden considerar independientes del número de procesadores, n. *h* es la función introducida por Le y Rejeb para describir los gastos generales asociados al uso de middleware en el proceso de paralelización en los *workers* (h_w) y el *master* (h_m). Finalmente, t_{Seq} y t_p representan el tiempo utilizado por las partes secuenciales y paralelas del código, respectivamente.

En nuestro caso, un total de m tareas independientes se ejecutan en n procesadores, con n<m. En principio, pueden aplicarse diferentes esquemas de planificación para asignar las m tareas a los n procesadores disponibles [100], [101], [102]. Asumimos, por simplificar, que el sistema es homogéneo y que el tiempo por tarea es el mismo, t_1. En estas condiciones, nosotros podemos aplicar el esquema *Chunk Self-Scheduling* (CSS) con el fin de minimizar las sobrecargas debidas a las comunicaciones [100]. De este modo, se asignan un total de m/n tareas a cada uno de los n procesadores. Aplicando estas consideraciones a la expresión (4.1) obtenemos,

$$t_{total} = \sum_{i}^{m/n} t_i = \frac{m}{n}\, t_1 + (o_m + L + G.S + o_w + h_m + h_w) \tag{4.2}$$

En este caso, es útil la definición de un nuevo índice F que muestre la relación entre los costes de comunicación y ejecución:

$$F = n\frac{(o_m + L + G.S + o_w + h_m + h_w)}{mt_1} \tag{4.3}$$

El índice F está formalmente definido en el intervalo (0, ∞). La ecuación (4.3) muestra que si se aumenta el número de tareas, m, se decrementa el peso relativo de los costes de comunicación.

Con respecto a los cálculos de estructura electrónica, el tamaño de los archivos de entrada es pequeño, del orden de unos KB. Por otro lado, y dependiendo del tipo de cálculo, el archivo de resultados puede tener un tamaño del orden de varios cientos de KB, alcanzando incluso varios cientos de MB en el peor de los casos.

Sin embargo, solo se necesita una cantidad mínima de información para la obtención de la hipersuperficie (energía total y coordenadas de vibración, al menos). Teniendo en cuenta que un problema de barrido de parámetros implica miles de puntos, la selección de la información (filtrado) en los *workers*, antes del retorno, reducirá drásticamente los costes de comunicación. Además, si tenemos en cuenta que la duración de los cálculos de

estructura electrónica es del orden de minutos y en muchos casos de horas, podemos asumir que el tiempo de comunicación puede ser despreciable frente al tiempo de ejecución, lo que implica que el índice F de la ecuación (4.3) es muy pequeño. Por tanto, en nuestro caso la ecuación (4.2) quedaría reducida a:

$$t_{total} \cong \frac{m}{n}\, t_1 \tag{4.4}$$

La expresión (4.4) muestra que el tiempo total (t_{total}) depende linealmente del número de tareas, m, y disminuye con el aumento del número de procesadores, n. Por lo tanto, este simple modelo explica los resultados de la Figura 4.8 como consecuencia del mucho mayor peso del coste de computación frente al de comunicación en nuestro problema.

Por otro lado, resulta interesante comparar el coste temporal total del proceso, *makespan*, con el que correspondería a la ejecución del mismo en un único procesador. Teniendo en cuenta que cada cálculo precisa de entre 45 y 60 minutos, el coste total es de entre 3024000 y 4032000 segundos. Estos resultados pueden compararse con los de los casos a y b de la Figura 4.8. El *speedup* en el caso a) está entre los valores 26.9 y 35.8 mientras que en el caso b) obtenemos el intervalo 49.2-65.6. En ambos casos el intervalo acota el número total de procesadores correspondiente (32 y 64, respectivamente). Una vez más, esta concordancia muestra la validez de la ecuación (4), como una consecuencia de la relación entre comunicación y número de trabajos expresada en la ecuación (4.3).

4.5 Recolección de resultados

En este apartado, se afronta la quinta etapa del ciclo de vida PSWF, véase Figura 3.1. El objetivo aquí es la recolección de los ficheros que representan los resultados para los distintos ejemplares del problema y su posterior análisis.

La recolección de resultados, como se explica en el apartado 3.3, se realiza con el algoritmo PSNSS, véase Figura 3.6, sólo que en este caso, a diferencia de la generación, se invoca a la función C(n, j), véase Figura 3.15.

En cuanto a la implementación, se ha utilizado la misma función que para la generación, véase Figura 4.5. La única diferencia estaría en la invocación que pasaría a ser: PSNSS (n, f, C). La implementación del algoritmo C(n, j) se ha realizado en C/C++ y se incluye el código en la Figura 4.9.

```
void C(int n, int *j){
   string WF("acetona"); // Creación e inicialización de variables
   string cd, command;
   struct stat st;
   char path[256];

   getcwd(path, 255);                                    // Acceso al directorio WF
   command+=path; command+="/"; command+=WF;
   chdir(command.c_str());
   getcwd(path, 255);

   for (int i=0; i<n; i++){          // Acceso a la rama del árbol correspondiente a j
      cd=IntToString(j[i]);
      command.clear();
      command+=path; command+="/"; command+=cd;
      chdir(command.c_str());
      cd.clear();
  } // Final del bucle for
  procesarAcetona(n, j);
}  //Final de la función C(n, j)
```

Figura 4.9 Implementación en C/C++ del algoritmo C(n, j), véase Figura 3.15

La función procesarAcetona(n, j) se encarga de integrar los resultados parciales en un único archivo de resultados denominado mol.hyp, véase Figura 4.2.

El resultado final del estudio, el archivo mol.hyp, define una base de datos de energía potencial molecular como función de los ángulos θ_1, θ_2, α y β. Por ejemplo, con la información contenida se puede construir el modelo más sencillo de torsión de los grupos metilos. A tal efecto, se determina la variación de energía potencial con los ángulos de torsión (θ_1, θ_2) para el valor de equilibrio de los ángulos α y β, ver Figura 4.10. Esta

información es fundamental para la interpretación de los datos espectroscópicos de la molécula de acetona observados en el infrarrojo lejano. La Figura 4.10 también permite identificar la posición de equilibrio de la molécula, punto A, y el punto de silla correspondiente a la barrera de rotación de los grupos metilos, punto B.

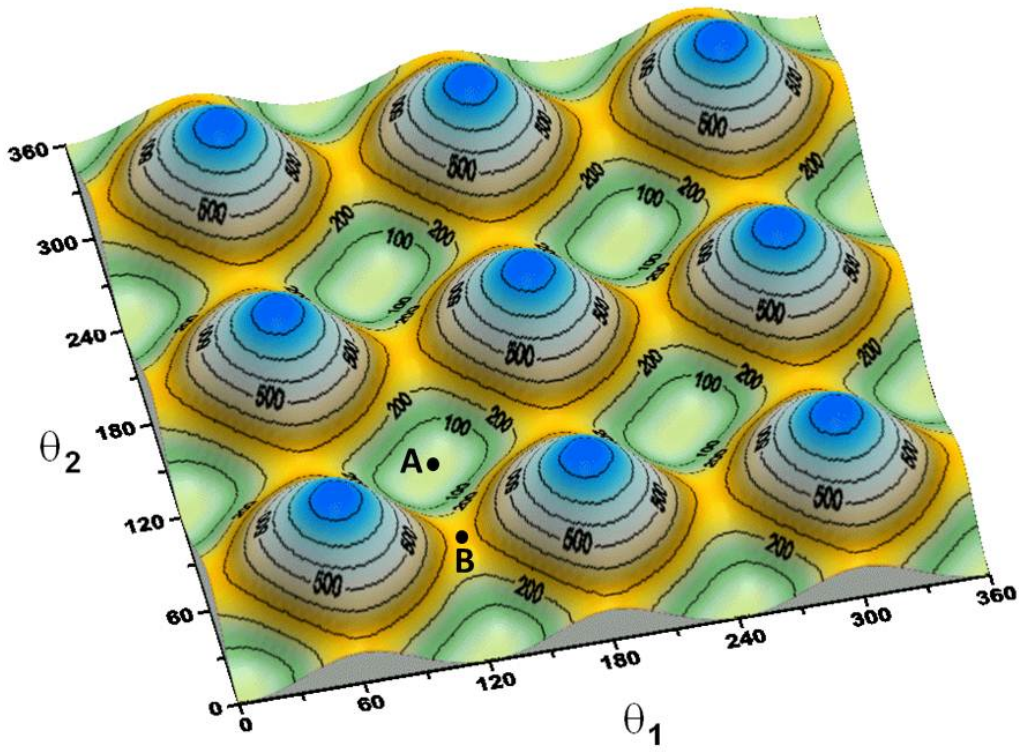

Figura 4.10 Hipersuperficie de energía potencial en 2D de la molécula de acetona.

En relación con las ventajas aportadas por el modelo PSWF, descritas en el apartado 3.3, cabe indicar, en concreto, que el caso presente en la aproximación tradicional hubiera implicado: la generación manual de los ficheros de entrada o su creación usando un programa específico a base de bucles anidados; la remisión y supervisión manual de cada caso considerado. Estas actividades son costosas en tiempo y claramente proclives a error, efectos solventados con el presente modelo. A su vez, la explotación del paralelismo inherente en el problema es altamente ventajosa como se describe en el siguiente apartado.

4.6 Contrastación experimental del algoritmo concurrente PSNSS

Este apartado lo dedicamos al análisis del rendimiento de la versión concurrente del algoritmo PSNSS, recogido en la Figura 3.7. En el apartado anterior se ha justificado la no adecuación de este algoritmo a la generación del árbol de directorios asociado al barrido de parámetros, para la obtención de la hipersuperficie de energía potencial molecular, debido a que el tiempo invertido en el proceso surge de una operación secuencial de escritura en disco.

La aplicación directa de la ley de *Amdahl* [103] muestra que en estas condiciones la concurrencia no aporta ninguna ventaja. Lógicamente, la condición para que la versión concurrente del algoritmo PSNSS aporte alguna mejora es que el tiempo consumido en la ejecución de la función, cuya llamada viene denotada con {*} en la Figura 4.5, cumpla la siguiente relación: $T_{CPU} >>> T_{E/S}$. En este apartado mostraremos la validez de esta afirmación usando como función {*} una multiplicación repetida 1000 veces de dos matrices reales de n x n (10000x10000) en memoria principal.

La Figura 4.11 muestra la implementación en C/C++, usando OpenMP y la directiva *task* disponible en la versión 3.0 [104] o superior, del algoritmo concurrente PSNSS propuesto en la Figura 3.7.

```
void PNSSC (int n, int *f, void (*function)(), int cores) {
 int *j, k;
 j=new int[n];                    // Creación del array de índices
 for (k=0; k<n; k++) {            // Inicialización del array de índices j
   j[k]=0;
 }
 k=n-1;

 omp_set_num_threads(cores);

 #pragma omp parallel          // Operador NSS concurrente
 {
  #pragma omp single
  {
        while (k>=0) {
            if (j[k]>f[k]) {
            j[k]=0;
            k--;
        }
        else {
          #pragma omp task
          {
              (*function)(n, j);  // Invocar a una función dónde TCPU>>>TE/S; {*}
                                  // En este caso se utiliza una multiplicación de matrices
          } // Fin task
```

```
            k=n-1;
          }
        } // Fin while
      } // Fin single
    } // Fin parallel
}//End PSNSSC
```

Figura 4.11 Implementación en C/C++ del algoritmo PSNSS concurrente, Figura 3.7, para i=0 y s=1

La ejecución de la versión concurrente PSNSS que muestra la Figura 4.11, denominada PSNSSC, ha sido realizada en un PC biprocesador con procesadores Intel Xeon 2.4 GHz de 8 núcleos cada uno (16 en total) y 16 GB de memoria RAM. Como índice de rendimiento se usa el *speedup*, definido como el cociente entre el tiempo de ejecución con un número arbitrario de hilos (*threads*) y el caso de un único hilo. En concreto, el tiempo usado por el algoritmo para la ejecución con un único hilo es de 48530 segundos. La Figura 4.12 muestra la evolución del *speedup* en función del número de procesadores (núcleos, *threads*) usados en el proceso concurrente.

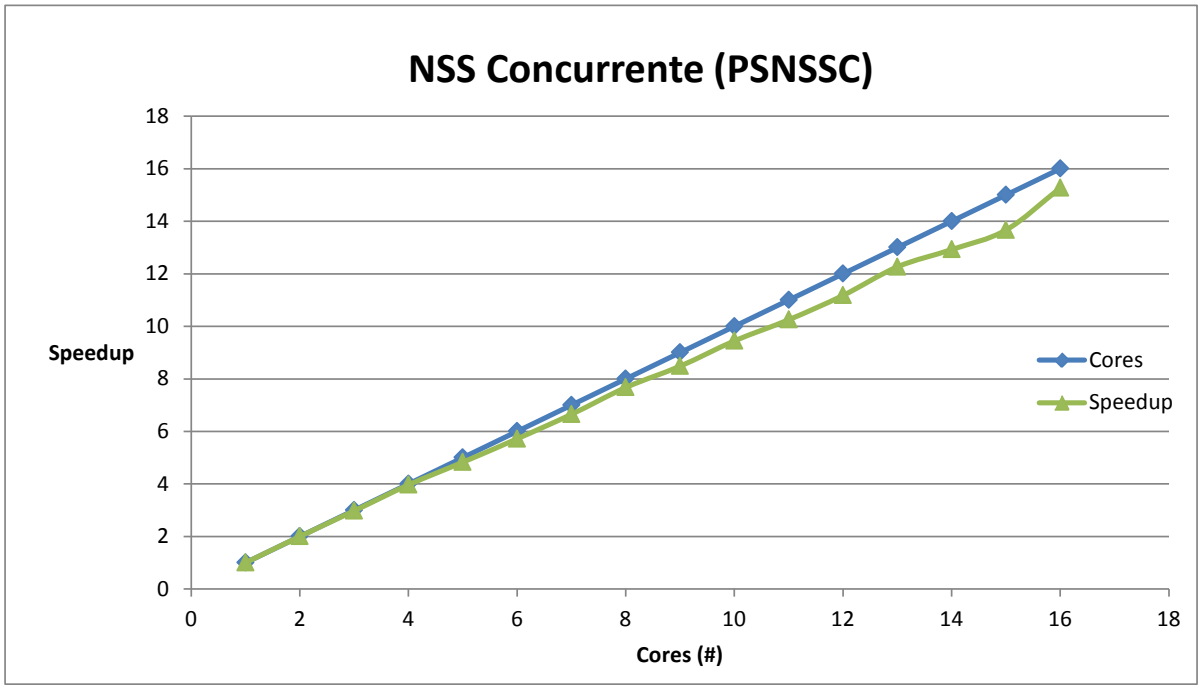

Figura 4.12 Rendimiento del algoritmo PSNSS concurrente. Como referencia, línea azul, la figura incluye la recta correspondiente al escalado ideal (lineal con pendiente uno).

La Figura 4.12 muestra que el comportamiento del algoritmo PSNSS en su versión concurrente hasta 4 núcleos es en la práctica equivalente a la ideal, con la curva de *Speedup* manteniendo una pendiente muy próxima a la unidad. Por encima de 4 núcleos, y hasta los 16, la evolución del rendimiento sigue siendo próxima al caso ideal.

En este punto es importante enfatizar la importancia del resultado anterior ya que el algoritmo PSNSS, propuesto en el apartado 3.3, se aplica en todas las fases ({3}, {4} y {5}) automatizables del ciclo de vida PSWF propuesto en el apartado 3.1. Esto implica que en toda etapa de una aplicación donde se cumpla la relación $T_{CPU} >>> T_{E/S}$, la versión concurrente aporta una clara mejora de rendimiento, véase Figura 4.12.

5 Modelo de monitorización y gestión de *Workflows* científicos

En este capítulo se presenta el desarrollo de un modelo genérico de monitorización y gestión de aplicaciones distribuidas, SsTAT (*Star Superscalar Status*), como es el caso de un SWF (*Scientific Workflow*). La propuesta, aquí recogida, proporciona apoyo en la búsqueda y supervisión de los recursos disponibles, así como en el rastreo del estado de los procesos en ejecución (tareas que forman el SWF). Para abordar estos objetivos, es necesario recoger información sistemática sobre el presente y, a veces, sobre la situación en el pasado de los recursos y tareas. Además, se considera la posibilidad de que el usuario pueda gestionar, de forma dinámica, la ejecución de la aplicación (SWF). Para su realización práctica, el modelo está diseñado para su integración con la familia de modelos de programación StarSs.

Asimismo, SsTAT se presenta como una solución al paso {4} (Monitorización) del ciclo de vida PSWF (*Parameter Sweep Scientific Workflow*), véase la Figura 3.1, y asume la tarea de recolección de la información necesaria para la presentación de las métricas descritas al final del apartado 3.3.

Comencemos presentando el problema que se aborda en este capítulo.

5.1 El problema considerado

La introducción del *Grid* [105] de computadores y de las tecnologías relacionadas, ha permitido que científicos e ingenieros desarrollen aplicaciones cada vez más complejas, basadas en SWF (véase apartado 2.1), para tratar y procesar grandes conjuntos de datos cuya ejecución involucra distintos recursos distribuidos.

Algunos SWF se traducen en una gran cantidad de tareas individuales que son necesarias para obtener la respuesta al problema considerado. Este es el caso de los problemas de barrido de parámetros, objetivo principal de análisis y estudio de este trabajo, que surgen de forma natural en varios campos científicos y de ingeniería (véase apartado 1.1).

En estos problemas, tenemos un sistema definido por un conjunto de parámetros independientes. El comportamiento del sistema se puede determinar mediante la variación de estos parámetros. Esto produce un SWF que se despliega como un conjunto de tareas que se pueden ejecutar de forma independiente. Dependiendo del tamaño del espacio de parámetros, el número de tareas puede ser muy grande. Como se ha presentado en los capítulos anteriores, un enfoque basado en sistemas distribuidos resulta muy eficaz para el tratamiento de este tipo de problemas [16], [106].

El desarrollo de este tipo de aplicaciones científicas o de ingeniería precisa recursos de computación para la composición y ejecución de SWF complejos. En este contexto, con la familia de modelos de programación StarSs se pueden implementar y gestionar SWF de forma automática [107].

Con este enfoque, el usuario debe poder modificar los recursos asociados al SWF, terminar y reiniciar el mismo e incluso cancelar y relanzar tareas individuales. Además, en contraste con las herramientas existentes, también se considera un modo *off-line* de seguimiento del SWF. Este método es especialmente útil cuando la aplicación está compuesta por un gran número de tareas independientes (PSWF) y el usuario tiene que controlar el tiempo de evolución y comportamiento del proceso.

Comencemos con la presentación de la arquitectura propuesta para el sistema de monitorización-gestión.

5.2 Arquitectura

Para el diseño del monitor se han tenido en cuenta las siguientes recomendaciones internacionales (estándares):

- *Job Status and Progress Monitoring* [52].
  Este estándar recomienda una serie de escenarios de monitorización y gestión. El escenario elegido ha sido el número seis; "control del progreso de las tareas en ejecución por el usuario". En este trabajo, se propone que dicho control se centre en dos variables fundamentales: estado y tiempo.
- *Grid Monitoring Architecture* (GMA) [53].
  Como puede observarse en la Figura 5.1, el OGF [108] recomienda, para el diseño de un sistema de monitorización, el uso de tres tipos de componentes: Consumidor, Productor y Servicio de Directorio. Los productores generan los diferentes eventos asociados a cada tarea, de estado y tiempo, que son accesibles a través del servicio de directorio por parte de los consumidores. En el enfoque que aquí se propone, la generación de eventos (por los productores) y su uso (por los consumidores) son procesos asíncronos.

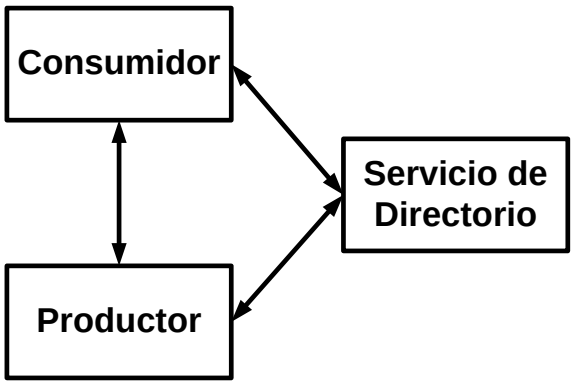

Figura 5.1 OGF *Grid Monitoring Architecture* (GMA)

- Con respecto al estado de las tareas se han seguido las recomendaciones establecidas por *Czajkowski* y colaboradores [109] en cuanto al número y tipo de estados por los que transcurre una tarea durante su ejecución en un sistema distribuido. En la Figura 5.2 puede observarse el diagrama de transición de estados aquí considerado. De esta forma, el monitor debe identificar los posibles estados de cada tarea, a saber:
  - *Start* (S): este estado se corresponde con el inicio de la tarea.

- *Pending* (P): en este estado la tarea está pendiente de ejecutarse; todavía no tiene asignado ningún recurso.
- *Active* (A): en este estado la tarea está en ejecución.
- *Failed* (F): este estado indica que la tarea ha fallado. Como puede observarse en la Figura 5.2, a este estado se llega desde P o desde A y se trata de un estado final.
- *Done* (D): una tarea en este estado indica que ha finalizado su ejecución. Se trata de un estado final.

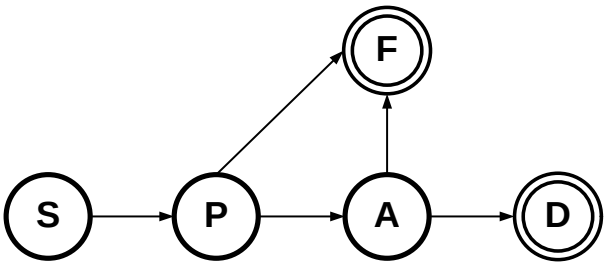

Figura 5.2 Diagrama de transición de estados

Basándose en los estándares anteriores, la Figura 5.3 muestra, en un esquema conceptual, el modelo lógico del sistema de monitorización y gestión de tareas propuesto, SsTAT (*Star superscalar Status*). En este caso se observa que, usando una aproximación *multi-tier*, la interfaz de usuario está formalmente separada de la componente funcional. De esta forma, se permite una adaptación independiente a sistemas de interacción diferentes, es decir, la interfaz de línea de comandos o interfaz gráfica de usuario.

El modelo lógico de SsTAT sigue una organización en niveles, véase Figura 5.3. El primer nivel se corresponde con los subsistemas de monitorización, gestión y ayuda. A su vez, los subsistemas de monitorización y gestión se subdividen en un segundo nivel en los siguientes subsistemas: sistema, SWF y tarea. Se puede observar que la monitorización del sistema permite obtener información sobre el estado del SWF que se ejecuta en el nodo maestro (*master*) y de las tareas en los nodos esclavos (*workers*). También es posible obtener información a nivel de aplicación (SWF). A nivel de tarea es posible obtener información de estado y de tiempo, así como detalles de los recursos que se están utilizando.

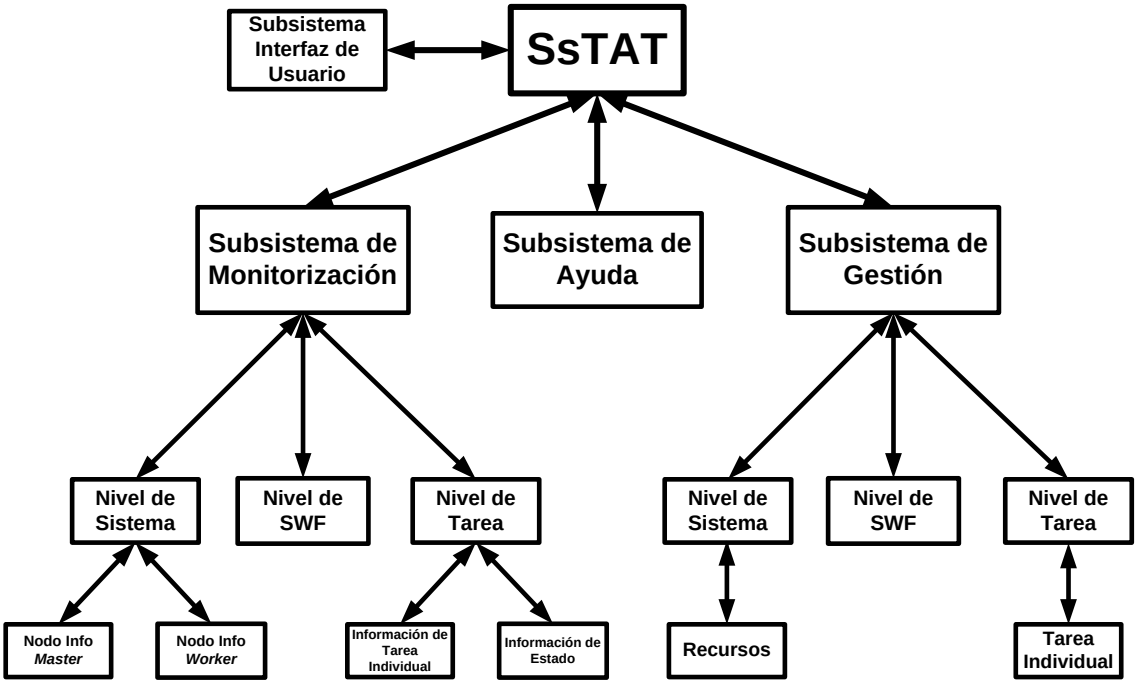

Figura 5.3 SsTAT (*Star Superscalar Status*): modelo lógico

Con respecto al subsistema de gestión, la Figura 5.3 muestra que a nivel de sistema el monitor permite la modificación de los recursos asociados al SWF tales como número de nodos, número de procesadores por cada nodo, etc. Por otro lado, también es posible interaccionar con el SWF. En este nivel se incorpora la posibilidad de detener todas las tareas individuales en la ejecución antes de finalizar el SWF. A continuación, se puede reiniciar el SWF, después de ese punto, utilizando las capacidades de *checkpointig* proporcionadas por la familia de modelos de programación distribuida, StarSs, que suministra servicios a SsTAT. Con respecto a una tarea individual, SsTAT permite su detención y relanzamiento gracias a la capacidad de tolerancia a fallos proporcionada por la familia StarSs al sistema de monitorización−supervisión SsTAT.

Para el diseño se ha tenido en cuenta que la implementación del monitor/gestor debía hacerse para la familia de modelos de programación StarSs aunque sin olvidar la posibilidad de realizar futuras implementaciones adaptadas a otros entornos de programación de aplicaciones distribuidas y sistemas para el desarrollo de *Workflows*. Por lo tanto, se trata de un diseño genérico de monitor/gestor de SWF.

Una vez presentado el modelo lógico del sistema de monitorización y gestión de SWF, SsTAT, se plantea el diseño y análisis de su arquitectura.

La arquitectura del sistema de monitorización−gestión SsTAT se muestra en la Figura 5.4. Como puede observarse se muestran dos visiones del sistema, una conceptual y otra de capas. La Figura 5.4a muestra el modelo conceptual, mientras que la figura 5.4b considera el modelo de capas. Se puede observar que SsTAT se organiza en cuatro módulos distribuidos en el *Master* (recurso cliente; tipo *host*) y en los *Workers* (recursos servidores; tipo *frontend* de un *cluster*), véase Figura 5.4a.

Estos módulos son los siguientes:

- Ssmon

  Es un agente que supervisa la ejecución de la aplicación (SWF). Se inicia con StarSs en el *Master*. Se trata de procesos separados que se ejecutan simultáneamente. Periódicamente, el agente recibe un evento (por tarea) con tres datos procedentes del SWF: el identificador de tarea, el recurso (*Worker*) utilizado para su ejecución y el estado de la tarea. *Ssmon* comienza a contabilizar el tiempo con el primer evento asociado a cada tarea. Los datos se insertan o actualizan en una base de datos, véase la Figura 5.4a. *Ssmon* es un productor de acuerdo a la arquitectura GMA.

- SsmonW

  Se trata de un sensor que observa la ejecución de una única tarea en uno de los recursos (*Worker*). Cada sensor creado por StarSs está asociado a una tarea en un recurso (*Worker*). Habrá tantos sensores (*SsmonW*) como tareas en ejecución (SWFW). Estas dos entidades se han diseñado por separado porque SWFW depende de una aplicación concreta y de esta forma se consigue que *SsmonW* sea independiente de la aplicación. Por lo tanto, el nodo M del *Worker* N ejecuta un ejemplar del SWF, SWFW, y del sensor, *SsmonW*, véase Figura 5.4b.

  El sensor obtiene las métricas de rendimiento (tiempos: CPU, Wall, etc. para SWFW, véase la Figura 5.4b) de cada tarea mediante el acceso al sistema de archivos /proc del recurso. De hecho, lo que se hace es acceder a la información de los procesos individuales, a nivel de sistema operativo, asociados a la tarea. El sensor almacena la información en uno o varios archivos de datos (ArchivosM). De acuerdo con la arquitectura GMA, *SsmonW* es un productor.

- Base de datos

  Los datos generados por *Ssmon* y *SsmonW*, es decir, la información sobre el estado y el tiempo que corresponde a cada tarea, se almacenan y se actualizan en una base de datos (véase Figura 5.4). Esta base de datos forma parte del Catálogo de Datos definido en la Figura 3.1. Se trata de una base de datos relacional que se mantiene en el disco después de la ejecución con el fin de poder analizar la información fuera de línea y contribuir al histórico (*provenance*) del SWF. Este componente es un servicio de directorio de acuerdo a la arquitectura de GMA.

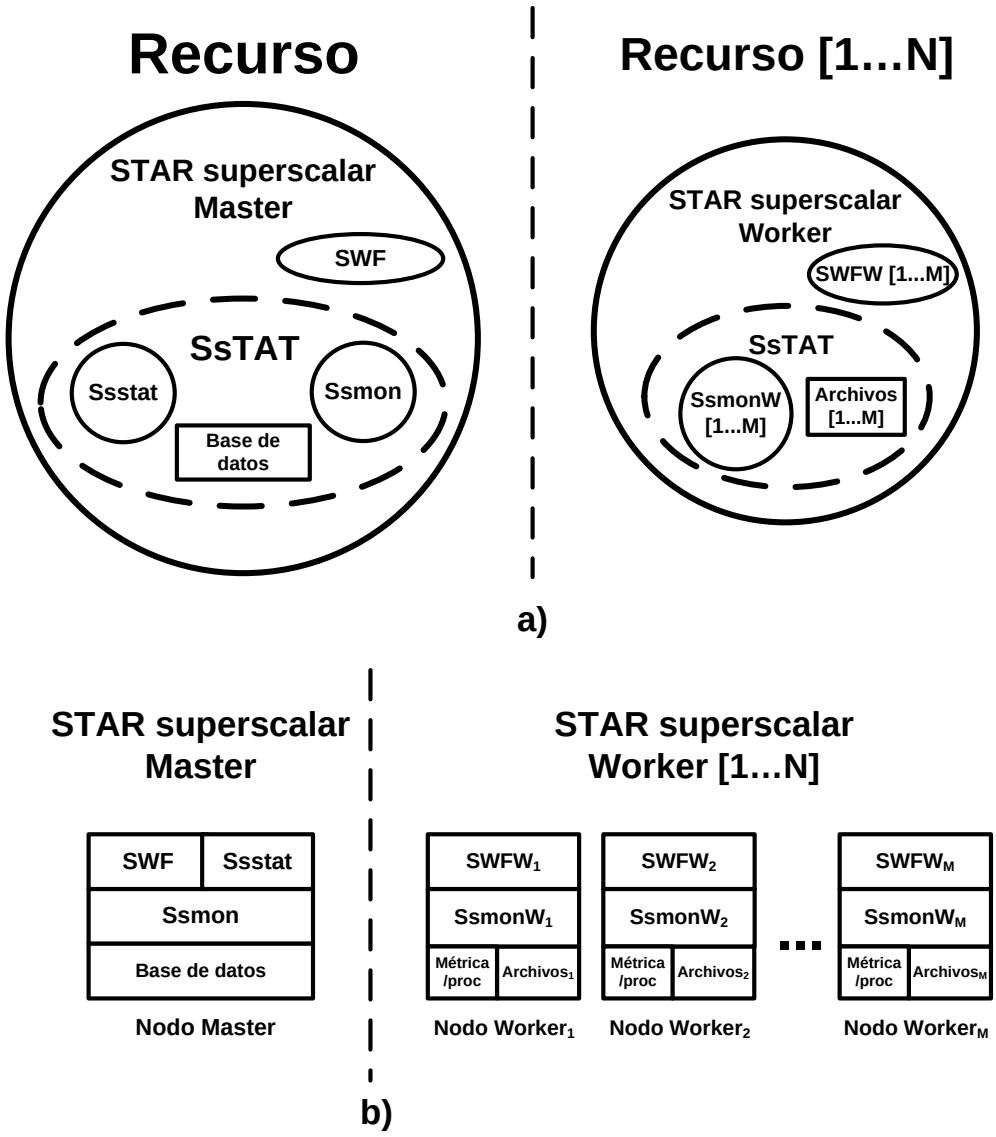

Figura 5.4 Arquitectura SsTAT: a) Modelo conceptual; b) Modelo de capas

- Ssstat

  Es la interfaz de usuario para el sistema SsTAT. Por lo tanto, es el componente encargado de suministrar al usuario, previa solicitud, un informe sobre el estado del SWF y de sus ejemplares en ejecución (tareas), SWFW. Además, permite al usuario modificar las características de los recursos del sistema distribuido asignados y algunas características (finalizar–relanzar el SWF y cancelar tareas) del SWF en ejecución. Este componente es un consumidor de acuerdo con la arquitectura de GMA.

La Figura 5.4b muestra que sólo un ejemplar del agente *Ssmon* se ejecuta en el *Master* asociado al SWF implementado y en ejecución mediante StarSs. Por otro lado, en cada uno de los N *Workers* se ejecutan M ejemplares del sensor *SsmonW*, una por cada tarea SWFW en ejecución.

En la Figura 5.5 se muestra un diagrama de interacción entre los componentes de SsTAT y StarSs. En el *Master*, el usuario ejecuta el SWF. A continuación, StarSs inicia el agente *Ssmon* mediante el *runtime* en el *Master*. Así mismo, el runtime de StarSs en el *Master*, utilizando los servicios subyacentes proporcionados por el Protocolo de Comunicaciones, envía los ejemplares del SWF, $SWFW_i$, para su ejecución en el sistema distribuido. A su vez, el *runtime* de StarSs en el *Worker* ejecuta las instancias $SWFW_i$ y el sensor asociado a las mismas $SsmonW_i$.

El agente *Ssmon* y los sensores, $SsmonW_1$, …, $SsmonW_M$, son síncronos con SWF y $SWFW_1$, …, $SWFW_M$, respectivamente. Por lo tanto, si SWF o algún ejemplar $SWFW_i$ finalizan entonces finaliza también *Ssmon* o el ejemplar correspondiente de *SsmonW*, respectivamente.

Ssstat es la interfaz de usuario y proporciona la información necesaria para la supervisión de eventos, véase la Figura 5.5. De esta forma, a petición del usuario, *Ssstat* envía un evento al agente *Ssmon* solicitando información. Con respecto al comportamiento de los eventos de gestión, en la Figura 5.5 se muestra (mediante la línea discontinua con guiones y puntos) como *Ssstat* permite el envío de eventos al *runtime* de StarSs posibilitando, como consecuencia, acciones específicas tales como la terminación del SWF (terminar) o de alguna tarea específica o SWFW (cancelar). Además, permite la modificación de los recursos asociados a una aplicación (SWF), actualizando la información de configuración de StarSs.

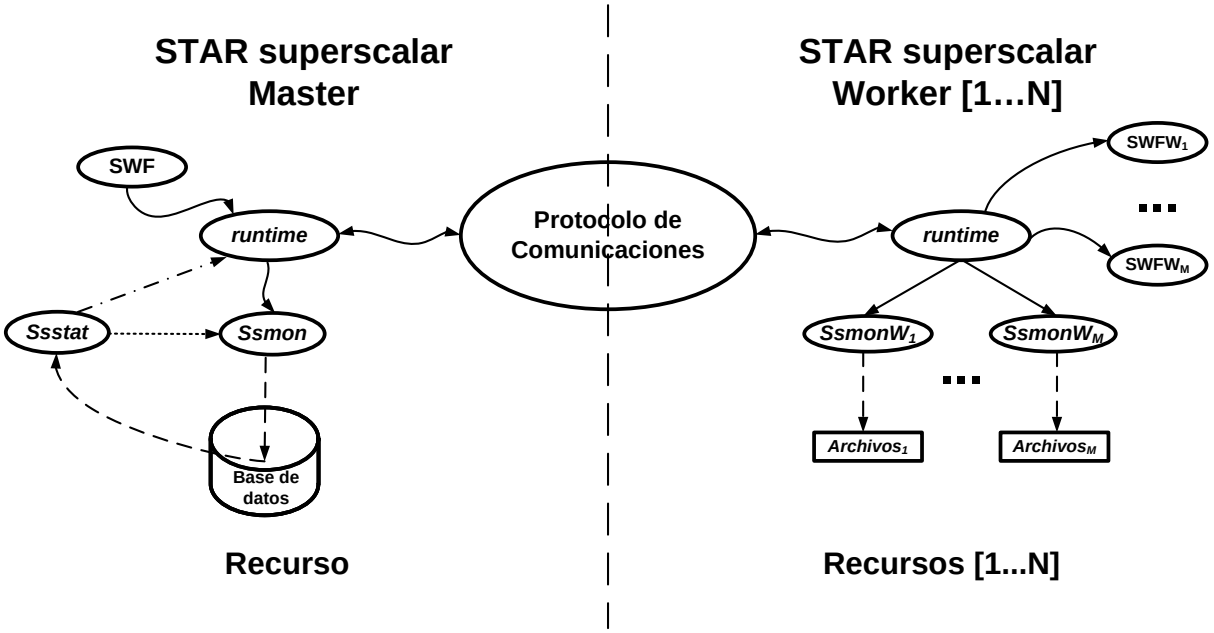

Figura 5.5 Diagrama de interacción entre componentes de StarSs y SsTAT.

En la Figura 5.5 se observa que *Ssmon*, productor según la arquitectura GMA, actualiza la base de datos con la llegada de eventos procedentes del *runtime* de StarSs en el *Master*. Por otro lado, *SsmonW,* como productor en ejecución en un recurso (*Worker),* actualiza uno o varios archivos (*Archivosi*) con información de las métricas correspondientes asociadas al ejemplar de SWFW que supervisa.

Cuando una tarea SWFW finaliza, la información recopilada por *SsmonW* (*Archivosi*), se transfiere hacia el *master* por el *runtime* de StarSs y se actualiza en la base de datos por *Ssmon*. De esta forma, es posible el análisis de la información de monitorización del SWF fuera de línea (*off-line*), una vez que éste haya finalizado.

El diseño desacoplado de SsTAT permite la integración de éste con cualquiera de los miembros de la familia StarSs. De hecho, SsTAT sólo requiere que el entorno de programación soporte las siguientes funcionalidades:

a) Modificación dinámica de recursos
b) *Checkpointing*
c) Tolerancia a fallos.

Por tanto, SsTAT puede ser integrado con cualquier entorno de programación de SWF que satisfaga estos requisitos.

5.3 Presentación de la información

Considérese un sistema distribuido basado en el modelo *master–worker* como es el caso de la familia de modelos de programación StarSs. Cuando se ejecutan SWF con StarSs, un nodo actúa como *Master*, mientras que los demás actúan como *Workers*. Una vez que un SWF se inicia en el nodo *master*, se tiene una serie de tareas que se ejecutan en los procesadores de los nodos individuales de los *Workers*. Estas tareas pueden encontrarse en cualquiera de los diferentes estados identificados por el monitor SsTAT, véase la Figura 5.2. La visualización de la información de monitorización y supervisión puede hacerse de formas diferentes, a saber, en modo texto o mediante una interfaz gráfica de usuario. En esta primera propuesta de SsTAT se ha optado por la línea de comandos como interfaz de usuario.

```
============================================================
        Star superscalar STATUS (SsTAT)
============================================================
USAGE: Ssstat [OPTIONS] [COMMANDS]
OPTIONS:
 -a                      display ACTIVE tasks
 -aw worker              display ACTIVE tasks in worker
 -c n                    CANCEL task number n
 -d                      display DONE tasks
 -dw worker              display DONE tasks in worker
 -e n                    display status of task number n
 -f                      display FAILED tasks
 -fw worker              display FAILED tasks in worker
 -h                      display help
 -k n                    TERMINATE of task number n
 -p                      display PENDING tasks
 -pw worker              display PENDING tasks in worker
 -r                      modify RESOURCES
 -s                      display status of the tasks
 -t n                    display TIMES of task number n
 -w                      display status in all workers
 -w worker               display status in worker
 without argument        equivalent to -s
COMMANDS:
 | more                  display one screen at a time
 | tail                  shows the last lines of the command output
============================================================
```

Figura 5.6 Opciones de monitorización y supervisión disponibles en SsTAT.

La Figura 5.6 muestra cómo acceder a las diferentes capacidades de monitorización y supervisión a través de la interfaz de usuario, véase Figura 5.3. En particular, el sistema de ayuda *on-line* es accesible desde la línea de comandos ejecutando *Ssstat* con la opción *–h*, véase Figura 5.6.

- Capacidades de monitorización
  - *Información sobre el SWF y el recurso Master*. En este caso se accede sólo mediante la invocación del comando *Ssstat*, véase Figura 5.6. La información se presenta en el formato que se muestra en la Figura 5.7 con un ejemplo concreto. En este caso, MASTER, véase la Figura 5.7, identifica el recurso (*hostname*), donde se inició el SWF. La información proporcionada incluye el estado (*Done*, *Active*, *Pending* y *Failed*) de todas las tareas en ejecución pertenecientes al SWF en el momento actual, además del tiempo de ejecución del SWF. Por otro lado, para las tareas finalizadas, "DONE", se muestran las estadísticas de tiempos de ejecución mínimo, medio y máximo. Con esta capacidad de monitorización, los usuarios pueden verificar el estado de sus SWF.

```
==============================================
            Star superscalar STATUS
==============================================
MASTER: hostname
--------------------------------------
Total TIME (dd:hh:mm:ss) = 00:00:31:02
==============================================
DONE: 37 ACTIVE: 3 PENDING: 1 FAILED: 0
==============================================
Statistics for Tasks DONE
-------------------------
Time(Min|Avg|Max)=(00:01:23|00:03:11|00:09:09)
==============================================
            Star superscalar RUNNING
==============================================
```

Figura 5.7 Información, a nivel de aplicación, proporcionada por SsTAT

  - *Información a nivel de recurso (Worker)*. A esta opción se accede mediante la invocación de *Ssstat* con la opción *–w* (*Ssstat –w*). El resultado se muestra en la Figura 5.8. Como puede observarse, se visualiza la información sobre el estado de las tareas que se ejecutan en los *Workers*. Se consideran todos los *Workers* con tareas en ejecución. Si el usuario desea información sobre un *Worker* en particular, puede conseguirla invocando desde la línea de comandos la orden *Ssstat –w worker*.

```
==========================================
Star superscalar Tasks STATUS in WORKERS
==========================================
------------------------------------------
 Worker 1
----------------------
 DONE: 35 ACTIVE: 2 PENDING: 0 FAILED: 0
------------------------------------------
 Worker 2
--------------------------
 DONE: 85 ACTIVE: 2 PENDING: 0 FAILED: 0
------------------------------------------
==========================================
```

Figura 5.8 Información, a nivel de nodo, proporcionada por SsTAT

```
==============================================
            Star superscalar Tasks ACTIVE
==============================================
 Task     Time          Running On
 ------   --------      --------------------------
   30   00:04:02        Worker 1
   32   00:02:20        Worker 2
   33   00:00:38        Worker 1
   34   00:00:35        Worker 2
-----------------------------------------------
 4 Tasks ACTIVE
==============================================
```

a)

```
=======================================================
          Star superscalar STATUS for Task number 30
=======================================================
 Task     Time      Status         Running On
 ------   --------  ----------     --------------------------
   30   00:07:09    DONE           Worker 1
=======================================================
```

b)

```
=======================================================
          Star superscalar TIMES for Task DONE number 1
=======================================================
   StarSs Time       WALL Time        CPU Time         RTS Time
   ---------------   ---------------  ---------------  ---------------
    00:05:05           00:04:51         00:02:03         00:00:14
   ---------------   ---------------  ---------------  ---------------
=======================================================
Executed  in Worker 1    (node-0-2)
=======================================================
```

c)

Figura 5.9 Información, a nivel de tarea, proporcionada por SsTAT

- *Información del estado de las tareas*. A esta opción se accede mediante la invocación de *Ssstat* con las siguientes opciones: *–d* (DONE), *–a* (ACTIVE), *–p* (PENDING) o *–f* (FAILED), véase la Figura 5.6. La información mostrada cambia según la opción elegida, véase Figura 5.9a). Además, se muestra el número de cada tarea, el tiempo entre el envío de la tarea al recurso (*Worker*) y la recepción del resultado, así como, el *Worker* asignado para su ejecución.
- *Información sobre una tarea individual*. Es posible visualizar información relativa a una única tarea, véase la Figura 5.9b). Para tal fin, se invoca *Ssstat –e N*, donde N es el número de la tarea. Además, es posible obtener para una determinada tarea finalizada sus tiempos *StarSs*, *Wall*, *CPU* y el *overhead* (RTS), véase la Figura 5.9c). Aquí, RTS se refiere al tiempo utilizado en la planificación, comunicación y encolado, si se utiliza sistema de colas, de la tarea.

- **Capacidades de gestión (*Steering*)**

  La Figura 5.10 muestra las opciones de intervención, a disposición del usuario, mediante la interfaz de SsTAT, de acuerdo con el diseño en la Figura 5.3.

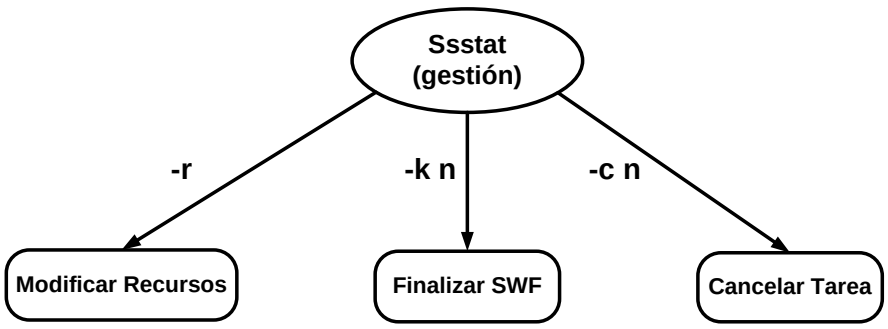

Figura 5.10 Capacidades de intervención incluidas en SsTAT

La Figura 5.10 muestra que con el comando *Ssstat –r* el usuario puede agregar o eliminar recursos y modificar las características de los recursos definidos. Con esta opción, el usuario puede agregar o eliminar *Workers* y/o modificar el número de procesadores asignados a los mismos.

Además, se puede finalizar la aplicación invocando *Ssstat –k N* (identificador de tarea). El *runtime* de StarSs, en el *Master*, prepara la finalización del SWF cancelando las tareas con identificador mayor que N y espera hasta que finalicen aquellas cuyo número de tarea sea inferior o igual a N. Cuando sea necesario, es

posible reiniciar el SWF desde la tarea *N+1*. Finalmente, el usuario puede cancelar una tarea individual invocando *Sstat –c N*, siendo *N* el identificador de tarea.

5.4 Contrastación experimental del modelo

En este capítulo se ha propuesto el diseño de un monitor–gestor de SWF, denominado SsTAT, integrable con la familia de modelos de programación StarSs. Para contrastar la viabilidad del modelo de monitorización y gestión de tareas SsTAT se ha desarrollado *GRID Superscalar Status* (GSTAT). GSTAT es una realización de SsTAT para GRIDSs, uno de los miembros de la familia StarSs. Se ha elegido la versión de GRIDSs que ofrece como interfaz de programación al usuario los lenguajes C/C++ o Perl. Dicha elección establece como protocolo de comunicaciones, véase Figura 5.5, *Globus Toolkit* [55] en alguna de sus versiones pre-Web *Services*. Por lo tanto, utilizaremos el *Grid* como infraestructura distribuida para las pruebas a realizar. En el presente caso, los recursos a los que se hace referencia en la Figura 5.5 serán un PC para el *Master* y uno o varios *Clusters* de computadores para los *Workers*.

Para la implementación de los componentes de GSTAT; *gsstat*, *gsmon*, *gsmonW* y Base de datos, véase Figura 5.5, se ha utilizado Perl [110]. Los componentes *gsmon* y *gsmonW* se ejecutan en segundo plano junto con SWF y SWFW, respectivamente. El módulo *gsstat* se ha implementado como un comando del sistema y puede ser invocado a discreción por el usuario. La separación de la detección y recopilación de información de su interfaz de usuario permite una mayor adaptabilidad del monitor GSTAT.

GSTAT permite analizar el comportamiento de SWF de alto rendimiento y de alta productividad en un *Grid* computacional (PSWF). Por lo tanto, se ha considerado un ejemplo de ambos casos. Para las experiencias se ha utilizado la implementación de GSTAT integrado con GRIDSs.

A continuación se consideran dos casos prácticos para analizar el comportamiento del modelo SsTAT y su realización GSTAT. En primer lugar se presentan las experiencias de monitorización y, posteriormente, las de gestión (intervención). Como primer caso se ha elegido un problema de barrido de parámetros como es la obtención de la hipersuperficie de energía potencial molecular. Aquí, no existe dependencia entre las tareas. Como segundo caso, consideramos la multiplicación en bloques de matrices. Aquí, sí existe dependencia entre las diferentes tareas tratándose de un conocido problema en computación de alto rendimiento.

1) Experiencias de monitorización

Con el fin de llevar a cabo las experiencias, se han utilizado los siguientes recursos: como *Grid Master* (*qcycar*) un PC x86_64 biprocesador con 4GB de RAM, con sistema operativo CentOS [111]. Como *Grid Workers* se han utilizado cuatro *clusters* de computadores: aristoteles, hermes, carrasca y popocatepetl. Aristóteles está formado por 42 procesadores x86_64 con 2GB de RAM. A su vez, hermes consta de 12 procesadores x86 con 1 GB de RAM. Carrasca tiene 8 procesadores x86_64 y 2 GB de RAM. Por último, popocatepetl es un *cluster* con 8 procesadores x86_64 y 1GB de RAM. Todos los *clusters* utilizan sistema operativo CentOS con *Rocks* [112] como *middleware* especializado en gestión y configuración de *clusters*. Aristoteles, carrasca, y popocatepetl utilizan como sistema de colas SGE [113], mientras que hermes usa PBS [114]. Todos los sistemas utilizan *Globus Toolkit* 4 [55] como *middleware Grid*. Físicamente, qcycar, aristoteles y hermes se encuentran en el grupo QCyCAR-ESI de la Universidad de Castilla-La Mancha en Ciudad Real (España), carrasca se encuentra en el Departamento de Astrofísica molecular e infrarrojo (DAMIR) del CSIC en Madrid (España), y popocatepetl se encuentra en la Universidad de Puebla en Puebla (México). Las latencias promedio observadas, han sido de 0,01, 5 y 300 milisegundos entre el *Master* qcycar y aristoteles, hermes (ambos ubicados en Ciudad Real), carrasca (Madrid) y popocatepetl (México), respectivamente. En la experiencia se han utilizado ocho procesadores, dos por cada *Worker*.

Este caso representa un ejemplo de barrido de parámetros en un entorno de computación de alto rendimiento. En particular, se tendrá en cuenta la obtención de la hipersuperficie de energía potencial del formaldehído protonado, una molécula de interés astrofísico [115].

Aquí, se considera una solución parcial de la hipersuperficie de energía potencial del formaldehído protonado. Por lo tanto, se utilizan dos coordenadas internas (ángulos), que definen un problema de dos dimensiones. En este caso es necesario resolver 433 cálculos individuales de estructura electrónica. Cada cálculo corresponde a una tarea en nuestro problema de barrido de parámetros. El SWF GSHYP [16] (se trata de la implementación, utilizando GRIDSs, de modelo PSHYP definido en el capítulo 4) se monitorizó desde qcycar con GSTAT. El tiempo total de ejecución para la aplicación fue de 3 horas y 12 minutos. El

promedio de tiempo invertido en la resolución de cada tarea fue de 3 minutos y 20 segundos. La Figura 5.11 muestra el número de tareas ejecutadas, las latencias promedio, el tiempo *Grid* promedio, el tiempo *Wall* promedio, el tiempo promedio de CPU y la RTG (tiempo *Grid* de ida y vuelta) por tarea en cada *Worker*.

	aristoteles	***carrasca***	***hermes***	***popocatepetl***
Executed Tasks	140	165	65	64
Avg. Latency (ms)	0.01	5	0.01	300
Avg. Grid Time (s)	154	131	341	326
Avg. Wall Time (s)	140	116	333	276
Avg. CPU Time (s)	140	113	150	182
Avg. RTG Time (s)	14	15	8	50

Figura 5.11 Número de tareas ejecutadas y un resumen de los tiempos medios

La Figura 5.11 muestra que en los sistemas con menos recursos de procesador y memoria (hermes y popocatepetl) se ejecutan un menor número de tareas. Este hecho también se refleja en la relación entre el tiempo *Wall* y el tiempo de CPU. De hecho, aristoteles y carrasca no muestran ninguna diferencia en términos prácticos, mientras que hermes y popocatepetl presentan un tiempo *Wall* promedio que casi duplica el promedio de tiempo de CPU. Esto se puede atribuir a la menor cantidad de memoria disponible en las dos últimos: 1 GB de RAM en comparación con 2 GB de aristoteles y carrasca. El cuello de botella en los cálculos de estructura electrónica molecular es la E/S. Por lo tanto, los sistemas con menos memoria necesitan más el acceso a disco y esto se refleja en una mayor diferencia entre el tiempo total (*Wall time*) y el tiempo de CPU.

En el otro extremo del espectro de la computación científica existen aplicaciones de alto rendimiento en las que las dependencias entre tareas son un punto importante a tener en cuenta. Por lo tanto, la necesidad de la sincronización también es un aspecto crucial. Como ejemplo, se tiene la multiplicación de matrices utilizando descomposición en bloques [94]. En este trabajo, se consideran dos matrices de 512 x 512. La aplicación se monitoriza, nuevamente, desde qcycar

con GSTAT. El tiempo total de ejecución para la aplicación fue de 1 hora y 53 minutos para 512 tareas. El promedio de tiempo invertido en la resolución de cada tarea fue de 1 minuto y 24 segundos. La Figura 5.12 muestra el número de tareas ejecutadas, las latencias promedio, el tiempo *Grid* promedio, el tiempo promedio total, el tiempo promedio de CPU y el tiempo de RTG por tarea en cada *Worker*.

	aristoteles	***carrasca***	***hermes***	***popocatepetl***
Executed Tasks	104	232	80	96
Avg. Latency (ms)	0.01	5	0.01	300
Avg. Grid Time (s)	120	46	144	90
Avg. Wall Time (s)	104	32	134	43
Avg. CPU Time (s)	45	31	58	37
Avg. RTG Time (s)	16	14	10	47

Figura 5.12. Número de tareas ejecutadas y resumen de los tiempos promedio de la multiplicación de matrices

La Figura 5.12 muestra que, como en el ejemplo anterior, se asignan más tareas a los nodos *Grid* más potentes: aristoteles y carrasca. Por otro lado, también se observa que aristoteles y hermes exhiben la mayor diferencia entre el tiempo promedio total y el tiempo promedio de CPU. Este comportamiento puede ser explicado en términos de la política de planificación utilizada por GRIDSs. Para explotar la localidad de datos, GRIDSs trata de asignar tareas dependientes a los mismos *Workers*. Por lo tanto, se reduce el *overhead* debido a las comunicaciones. En este caso, aristoteles y hermes se encargan de las tareas dependientes y, en consecuencia, necesitan más tiempo para resolverlas.

Los datos de las Figuras 5.11 y 5.12 también muestran que la mayor diferencia entre el tiempo promedio *Grid* y el tiempo promedio total aparece en el *Worker* con la mayor latencia (popocatepetl). Los datos de GSTAT permiten cuantificar estas diferencias entre 15% y el 52% de aumento para los ejemplos de las Figuras

5.11 y 5.12, respectivamente. Se observa que el tiempo de ida y vuelta *Grid* (*Round Trip Time Grid*, RTG) está muy relacionado con la latencia.

2) Experiencias de gestión (intervención)

Las capacidades de gestión también se pueden ilustrar utilizando el caso de la obtención de la hipersuperficie de energía potencial molecular. Los recursos utilizados han sido los siguientes: *Master* (bscgrid01) un x86 biprocessor PC con 1GB de RAM, con Linux. Físicamente, bscgrid01 se encuentra en el *Barcelona Supercomputing Center* (BSC-CNS). Como *Workers* se usan los siguientes: aristoteles y hermes, que se describieron en la sección anterior. La Figura 5.13 muestra las tareas en estado activo en un momento dado (*gsstat –a*). Aquí, se observa que una de las tareas, la tarea con identificador 49, consume mucho más tiempo (el tiempo promedio es de 3 minutos y 11 segundos, ver Figura 5.7) que las otras. Entre otras causas, esto puede ser debido a problemas en el nodo del *Worker*, donde la tarea se está ejecutando (Hermes). El usuario puede cancelar esta tarea invocando *gsstat –c 49*.

```
================================================
        GRID superscalar Tasks ACTIVE
================================================
 Task     Time          Running On
 ------   --------      --------------------------
   49     00:29:18      hermes.inf-cr.uclm.es
   74     00:04:21      hermes.inf-cr.uclm.es
   76     00:02:10      aristoteles.inf-cr.uclm.es
   77     00:01:01      aristoteles.inf-cr.uclm.es
------------------------------------------------
 4 Tasks ACTIVE
================================================
```

Figura 5.13 Antes de cancelar la tarea 49

Después de eso, la Figura 5.14 muestra que la tarea de 49 ha desaparecido del sistema.

```
==============================================
      GRID superscalar Tasks ACTIVE
==============================================
 Task     Time          Running On
 ------   --------      --------------------------
   76     00:02:29      aristoteles.inf-cr.uclm.es
   77     00:01:20      aristoteles.inf-cr.uclm.es
   78     00:00:01      hermes.inf-cr.uclm.es
----------------------------------------------
 3 Tasks ACTIVE
==============================================
```

Figura 5.14 Después de cancelar la tarea 49

A continuación, el *runtime* de GRIDSs [116] replanifica la tarea en tiempo de ejecución. En primer lugar intenta alojarla en el mismo *Worker* y, si no es posible, la asigna a otro. En el ejemplo, la Figura 5.15 muestra la tarea 49, de nuevo en ejecución, en aristoteles.

```
==============================================
      GRID superscalar Tasks ACTIVE
==============================================
 Task     Time          Running On
 ------   --------      --------------------------
   77     00:01:53      aristoteles.inf-cr.uclm.es
   49     00:00:36      aristoteles.inf-cr.uclm.es
   78     00:00:34      hermes.inf-cr.uclm.es
   79     00:00:21      hermes.inf-cr.uclm.es
----------------------------------------------
 4 Tasks ACTIVE
==============================================
```

Figura 5.15 Después de replanificar la tarea 49

6 Conclusiones y Trabajo futuro

En concordancia con el objetivo principal de este trabajo se ha mostrado la factibilidad del desarrollo de un modelo general de *workflow* para el problema de barrido de parámetros, así como, del modelado de un sistema genérico de monitorización y gestión de tareas, ambos aplicables en sistemas distribuidos. La implementación de los modelos desarrollados ha permitido comprobar que las soluciones aquí propuestas son operativas desde un punto de vista práctico.

A continuación se detallan las conclusiones parciales por capítulos:

En el Capítulo 3,

1) Se puede desarrollar un modelo de *workflow* genérico para problemas de barrido de parámetros de cualquier campo científico-tecnológico.
2) Se puede definir una arquitectura distribuida de referencia utilizable por la comunidad científica para la resolución de problemas de barrido de parámetros.
3) El operador NSS, *Nested Summation Symbol*, permite generalizar formalmente el tratamiento del barrido de parámetros. En concreto,
   a) Existe una solución genérica, basada en NSS, para la realización de barridos de parámetros completos o selectivos.
   b) Se puede definir un algoritmo general para el tratamiento del barrido de parámetros, PSNSS (*Parameter Sweep Nested Summation Symbol*).
   c) La concurrencia implícita del problema permite formular una versión paralela del PSNSS.
   d) La aplicación del NSS permite realizar un *mapping* a disco de la estructura lógica del barrido de parámetros, como un árbol de directorios, facilitando el tratamiento en disco del problema (generación) y la recolección e interpretación de resultados.

En el Capítulo 4,

4) La aplicación práctica del modelo PSWF (*Parameter Sweep Scientific Workflow*), propuesto en el capítulo 3, proporciona una solución eficiente a problemas reales como atestigua el uso de un caso de prueba: la obtención de la hipersuperficie de energía potencial molecular sobre un *Grid* de computadores. En concreto,
   a) El modelo genérico PSWF, aplicado a la obtención de la hipersuperficie de energía potencial molecular, permite generar un *workflow* específico denominado PSHYP, *Parameter Sweep Hypersurface*, que puede aplicarse a cualquier molécula y que resuelve, de forma automática, un problema que hasta el momento sólo tenía una solución manual.
   b) La implementación del algoritmo PSNSS resulta ser una herramienta capaz de manejar cientos de miles de conjuntos de datos con eficiencia. Por tanto, sería aplicable a problemas de *big-data*.
   c) La ejecución de las pruebas sobre el entorno *Grid* muestran una relación de proporcionalidad inversa entre número de procesadores y el tiempo total de ejecución del barrido de parámetros, a pesar de la heterogeneidad del sistema.
   d) La utilización del modelo de rendimiento LogGP modificado de Lee y Rejeb, permite demostrar que la razón del comportamiento anterior radica en el pequeño valor de la relación entre el coste de comunicación y el coste de computación.
   e) La versión concurrente del algoritmo PSNSS presenta un escalado prácticamente lineal en las pruebas realizadas con un máximo de 16 *cores*.

En el Capítulo 5,

5) Se muestra la factibilidad del desarrollo de un modelo genérico de monitorización y gestión de aplicaciones (*workflows*) científicas en entornos distribuidos. El modelo permite la búsqueda y supervisión de los recursos disponibles, así como el rastreo del estado de las tareas en ejecución. Además, es posible gestionar, de forma dinámica, la ejecución de la aplicación. El modelo, denominado SsTAT (*Star Superscalar Status*), es integrable con la familia de modelos de programación StarSs. En concreto,
   a) La aplicación de los estándares internacionales, relativos a la gestión y monitorización de sistemas, permite definir un modelo lógico para el sistema en entornos distribuidos.
   b) El modelo lógico anterior permite abordar una propuesta de arquitectura para el sistema de monitorización y gestión utilizable por la familia de modelos de programación StarSs y basada en cuatro módulos distribuidos entre el sistema *Master* y los *Workers*.
   c) El modelo de monitorización y gestión se puede aplicar en el seguimiento de SWF con y sin dependencias de datos.
   d) Los modelos anteriores permiten el desarrollo de un sistema específico para gestión y monitorización en *Grids* de computadores basado en GRIDSs. Su aplicación práctica, a un caso de barrido de parámetros sobre un *Grid* real, muestra que las capacidades de monitorización, a nivel de *workflow* y de tareas individuales, y las capacidades de intervención (cancelación y replanificación de tareas) se desarrollan eficientemente en el sistema considerado.

Como todo trabajo, el presente es susceptible de ampliación. Algunos de los posibles trabajos futuros que podrían abordarse como continuación del aquí presentado son:

1) La generación y poda dinámicas del árbol de directorios asociado al barrido paramétrico mediante el operador NSS teniendo en cuenta los resultados obtenidos en tiempo de ejecución.
2) Desarrollo de una interfaz gráfica para el sistema de monitorización.
3) La elaboración de una guía de buenas prácticas para la ejecución de SWF basados en el modelo PSWF propuesto en este trabajo.

4) El desarrollo de un sistema de clases para el tratamiento del problema de barrido de parámetros bajo una aproximación orientada a objetos. El sistema incorporaría las versiones secuencial y concurrente del algoritmo PSNSS y, mediante polimorfismo, sería aplicable a distintos tipos de sistemas distribuidos.
5) La implementación del *workflow* PSHYP con ClusterSs y su validación en los recursos del BSC-CNS y la RES (Red Española de Supercomputación).
6) La integración de SsTAT con el resto de modelos de programación pertenecientes a la familia StarSs.

7 Bibliografía

1. **Taylor, I.; Deelman, E.; Gannon, D.; Shields, M.** *Workflows for e-Science.* New York (USA) : Springer, 2007.

2. *Parameter sweeps for exploring GP parameters.* **Samples, M. E.; Daida, J. M.; Byom, M.; Pizzimenti, M.** Washington (USA) : Genetic And Evolutionary Computation Conference, 2005. págs. 212-219.

3. *Exploring the Viability of the Cell Broadband Engine for Bioinformatics Applications.* **Sachdeva, V.; Kistler, M.; Speight, E.; Tzeng, T. H. K.** 11, 2008, Parallel Computing, Vol. 34, págs. 616-626.

4. **Blackstone.** *Post-genomic Challenges Drive Life Sciences to High Throughput.* s.l. : White Paper, 2001.

5. *A Model of Problem Solving Environment for Integrated Bioinformatics Solution on Grid by Using Condor.* **Sun, C.; Kim, B.; Yi, G.; Park, H.** s.l. : Springer-Verlag, 2004. págs. 935-938. LNCS 3251.

6. Folding@Home Project Web Page. [En línea] [Citado el: 4 de 5 de 2012.] http://folding.stanford.edu.

7. **Buyya, R.; Branson, K.; Giddy, J.; Abramson, D.** *The virtual laboratory: Enabling molecular modelling for drug design on the World Wide Grid.* [ed.] Monash University. 2001.

8. **Gulamali, M. Y.; McGough, A. S.; Newhouse, S. J.; Darlington, J.;.** *Using ICENI to Run Parameter Sweep Applications Across Multiple Grid Resources.* Berlin (Germany) : Global Grid Forum (GGF), 2004.

9. **Basney, J.; Livny, M.; Mazzanti, P.;.** *Harnessing the Capacity of Computational Grids for High Energy Physics.* Padova (Italy) : Conf. on Computing in High Energy and Nuclear Physics, 2000.

10. The Large Hadron Collider (LHC) Computing Grid Project for High Energy Physics Data Analysis. [En línea] [Citado el: 4 de 5 de 2012.] http://lcg.web.cern.ch/LCG.

11. EGEE Project Web Page. [En línea] [Citado el: 4 de 5 de 2012.] http://www.eu-egee.org.

12. *GIPSE: Streamlining the Management of Simulation on the Grid.* **Wozniak, J. M.; Striegel, A.; Salyers, D.; Izaguirre, J. A.** San Diego (USA) : Proc. of the Thirty-Eight Ann. Simulation Symp., 2005. págs. 130-137.

13. *Hybrid Monte Carlo.* **Duane, S.; Kennedy, A. D.; Pendleton, B. J.; Roweth, D.;.** 1987, Physics Letters B, Vol. 195, págs. 216-222.

14. **Izaguirre, J. A.; Hampton, S. S.;.** *Shadow Hybrid Monte Carlo: An Efficient Propagator in Phase Space of Macromolecules.* s.l. : Journal of Computational Physics, 2004. págs. 581-604. Vol. 200.

15. *Targeted Mollified Impulse - A Multiscale Stochastic Integrator for Long Molecular Dynamics Simulations.* **Ma, Q.; Izaguirre, J. A.;.** 1, 2003, Multiscale Modeling and Simulation, Vol. 2, págs. 1-21.

16. *Performance of Computationally Intensive Parameter Sweep Applications on Internet-based Grids of Computers: the Mapping of Molecular Potential Energy Hypersurfaces.* **Reyes, S.; Muñoz-Caro, C.; Niño, A.; Badia, R. M.; Cela, J. M.** 4, 2007, Concurrency and Computation: Practice and Experience, Vol. 19, págs. 463-481.

17. *Distributing MCell Simulations on the Grid.* **Casanova, H.; Bartol, T.; Stiles, J.; Berman, F.;.** 3, 2001, Int. Journal High Performance Computing Applications, Vol. 14, págs. 243-257.

18. SpiceWeb Page. [En línea] [Citado el: 4 de 5 de 2012.] http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/.

19. *The Dissemination of Culture: A Model with Local Convergence and Global Polarization.* **Axelrod, R.;.** 2, 1997, Journal of Conflict Resolution, Vol. 41, págs. 203-226.

20. *Stealth multicast: A catalyst for multicast deployment.* **Striegel, A.;.** Athens (Greece) : s.n., 2004. Proc. of IFIP Networking. págs. 817-828.

21. *Simulating Computer Networks Using Clusters of PCs.* **Abramson, D.; Power, K.; Sosic, R.;.** San Diego (USA) : s.n., 1999. HPC-TelePar'99 at the 1999 Advanced Simulation Technologies Conf. (ASTC'99).

22. *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?* **Abramson, D.; Giddy, J.; Kotler, L.;.** Cancun (Mexico) : s.n., 2000. Int. Parallel and Distributed Processing Symp. (IPDPS). págs. 520-528.

23. *A new major SETI project based on Project Serendip data and 100,000 personal computers.* **Sullivan, W. T.; Werthimer, D.; Bowyer, S.; Cobb, J.; Gedye, D.; Anderson, D.** Capri (Italy) : s.n., 1997. Proc. of the Fifth Int. Conf. on Bioastronomy.

24. *Applying Scheduling and Tuning to On-Line Parallel Tomography.* **Smallen, S., Casanova, H. y Berman, F.** Denver (USA) : s.n., 2001. Proc. of the IEEE/ACM Supercomputing (SC 2001) Conf.

25. *Parameter Space Exploration with Gaussian Process Trees.* **Gramacy, R. B.; Lee, H. K. H.; MacReady, W.;.** Alberta (Canada) : s.n., 2004. Proc. of the Twenty-First Int. Conf. on Machine Learning.

26. *Programming Grid Applications with GRID superscalar.* **Badia, R. M.; Labarta, J.; Sirvent, R.; Pérez, J. M.; Cela, J. M.; Grima, R.;.** 2003, Journal of Grid Computing, Vol. 1, págs. 151-170.

27. Barcelona Supercomputing Center - Centro Nacional de Supercomputación. *GRID superscalar Download.* [En línea] [Citado el: 22 de 7 de 2012.] http://www.bsc.es/computer-sciences/grid-computing/grid-superscalar/download.

28. *Nested summation symbols and perturbation theory.* **Carbó, R.; Besalú, E.;.** 1993, Journal of Mathematical Chemistry, Vol. 13, págs. 331-342.

29. **Hollingsworth, D.** *The Workflow Reference Model. Workflow Management Coalition.* Workflow Management Coalition. 1995. Document Number TC00-1003.

30. *Web Services Business Process Execution Language (WSBPEL). Version 2.0.* The OASIS Committee. 2007.

31. **Deelman, E. y Gil, Y.** *Workshop on the Challenges of Scientific Workflows.* Information Sciences Institute, University of Southern California. 2006.

32. *Scientific Workflow Management and the Kepler System. Special Issue: Workflow in Grid Systems.* **Ludäscher, B.; Altintas , I.; Berkley, C.; Higgins, D.; Jaeger-Frank, E.;**

Jones, M.; Lee, E.; Tao, J.; Zhao, Y.;. 2006, Concurrency and Computation: Practice & Experience, Vol. 18(10), págs. 1039-1065.

33. *Pegasus: A framework for Mapping Complex Scientific Workflows onto Distributed Systems.* **Deelman, E.; Singh, G.; Su, M. H.; Blythe, J.; Gil, Y.; Kesselman, C.; Mehta, G.; Vahi, K.; Bruce Berriman, G.; Good, J.; Laity, A.; Jacob, J. C.; Katz, D. S.;.** 2005, Scientific Programming Journal, Vol. 13(3), págs. 219-237.

34. *Taverna: A tool for the composition and enactment of bioinformatics workflows.* **Oinn, T.; Addis, M.; Ferris, J.; Marvin, D.; Senger, M.; Greenwood, M.; Carver, T.; Glover, K.; Pocock, M. R.; Wipat, A.; Li, P.;.** Bioinformatics , Vol. 20(17), págs. 3045–3054.

35. *Triana applications within grid computing and peer to peer environments.* **Taylor, I., y otros.** 2003, Journal of Grid Computing, Vol. 1(2), págs. 199-217.

36. *Towards Automatic Generation of Semantic Types in Scientific Workflows.* **Bowers, S.; Ludäscher, B.;.** 2005. WISE Workshops. págs. 207-216.

37. *Kepler: An Extensible System for Design and Execution of Scientific Workflows.* **Altintas, I.; Berkley, C.; Jaeger, E.; Jones, M.; Ludäscher, B.; Mock, S.;.** 2004. SSDBM'04. págs. 423-424.

38. *Lifecycle of Scientific Workflows and Their Provenance: A Usage Perspective.* **Altintas, I.** SERVICES-I. págs. 474-475.

39. *A Web Service Composition and Deployment Framework for Scientific Workflows.* **Altintas, I., y otros.** 2004. ICWS. págs. 814-816.

40. *Provenance Collection Support in the Kepler Scientific Workflow System.* **Altintas, I.; Barney, O.; Jaeger-Frank, E.;.** 2006. IPAW. págs. 118-132.

41. *A Fault-Tolerance Architecture for Kepler-Based Distributed Scientific Workflows.* **Mouallem, P.; Crawl, D.; Altintas, I.; Vouk, M.; Yildiz, U.;.** [ed.] Springer. 2010. Lecture Notes in Computer Science. Vol. 6187, págs. 452-460.

42. *Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges.* **Deelman, E.; Gil, Y.;.** 2006. IEEE Computer Society. págs. 144-149.

43. *Examining the Challenges of Scientific Workflows.* **Gil, Y.; Deelman, E.; Ellisman, M.; Fahringer, T.; Fox, G.; Gannon, D.; Goble, C.; Livny, M.; Moreau, L.; Myers, J.;.** 2007, IEEE Computer, Vol. 40 (12), págs. 24-32.

44. *Scientific Workflows: Business as Usual?* **Ludäscher, B.; Weske, M.; McPhillips, T.; Bowers, S.** s.l. : Springer, 2009. Lecture Notes in Computer Science. Vol. 5701, págs. 31-47.

45. *Definition and Quantum Chemical Applications of Nested Summation Symbols and Logical Kronecker Deltas.* **Carbó, R.; Besalú, E.;.** 1994, Journal of Mathematical Chemistry, Vol. 15, págs. 397-406.

46. *Monitoring Distributed System.* **Joyce, J.; Lomow, g.; Slind, K.; Unger, B.;.** 2, 1987, ACM Trans. Computer System, Vol. 5, págs. 121-150.

47. *Debugging a Distributed Computing System.* **Garcia-Molina, H.; Germano, F. Jr.; Kohler, W. H.;.** 2, 1984, IEEE Trans. Software Eng., Vol. 10, págs. 210-219.

48. **Snodgrass, R. T.** *Monitoring distributed systems: A relational approach.* Department of Computer Science, Carnegie-Mellon University. Pittsburgh (USA) : s.n., 1982. Ph.D. dissertation.

49. **Harrison, M. D.** *Monitoring a target network to support subsequent host simulation.* Dept. of Computer Science, University of York. Toronto (Canada) : s.n., 1984.

50. *A taxonomy of grid monitoring systems.* **Zanikolas, S.; Sakellariou, R.;.** 2005, Future Generation Computer Systems, Vol. 25, págs. 163-188.

51. Open Grid Forum. [En línea] 15 de 6 de 2011. http://www.ogf.org.

52. **Aydt, R.; Quesnel, D.;.** Performance Data Usage Scenarios (Draft 1). [En línea] 2000. [Citado el: 15 de 6 de 2011.] http://www-didc.lbl.gov/GGF-PERF/GMA-WG/.

53. **Tierney, B.; Aydt, R.; Gunter, D.; Smith, W.; Swany, M.; Taylor, V.; Wolski, R.;.** Open Grid Forum. [En línea] 2002. [Citado el: 4 de 5 de 2012.] http://www.ogf.org/documents/GFD.7.pdf.

54. *A directory service for configuring high performance distributed computations.* **Fitzgerald, S.; Foster, I.; Kesselman, C.; von Laszewski, G.; Smith, W.; Tuecke, S.;.** s.l. :

IEEE Computer Society Press, 1997. Proceedings of the Sixth IEEE Symposium on High Performance Distributed Computing. págs. 365-375.

55. The Globus Alliance. [En línea] [Citado el: 4 de 5 de 2012.] http://www.globus.org/.

56. *Ganglia Distributed Monitoring System: Design, Implementation, and Experience.* **Massie, M. L.; Chun, B. N.; Culler, D. E.;.** 2004, Parallel Computing, Vol. 30, págs. 817–840.

57. *The network weather service: A distributed resource performance forecasting service for metacomputing.* **Wolski, R.; Spring, N.; Hages, J.;.** 1999, Future Generation Computing System, Vol. 15, págs. 745-755.

58. *GridRM: A resource monitoring architecture for the Grid,.* **Baker, M.; Smith, G.;.** 2002. Lecture Notes in Computer Science. Vol. 2536, págs. 268–273.

59. *GridICE: A monitoring service for Grid systems.* **Andreozzi, S.; De Bortoli, N.; Fantinel, S.; Ghiselli, A.; Rubini, G. L.; Tortone, G.; Vistoli, M. C.;.** 2005 : s.n., Future Generation Computer Systems, Vol. 4, págs. 559-571.

60. *NetLogger: a toolkit for distributed system performance tuning and debugging.* **Tierney, B.; Gunter, D.;.** [ed.] G. S. Goldszmidt y J. Schönwälder. s.l. : Kluwer, 2003. Proceedings of the IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003). Vol. 246, págs. 97–100.

61. *Monitoring grid applications with grid-enabled OMIS monitor.* **Balis, B.; Bubak, M.; Wismüller, R.; Radecki, M.;.** Santiago de Compostela (Spain) : LNCS, Springer-Verlag, 2004. Proceedings of the First European Across Grids Conference. Vol. 2970, págs. 230–239.

62. *OMIS 2.0: a universal interface for monitoring systems.* **Ludwig, T.; Wismiiller, R.;.** Cracow (Poland) : LNCS, Springer Verlag, 1997. Proceedings of the Fourth European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. Vol. 1332, págs. 267–276.

63. **Baude, F.; Baduel, L.; Caromel, D.; Contes, A.; Huet, F.; Morel, M.; Quilici, R.;.** *Programming, Composing, Deploying for the Grid in "GRID COMPUTING: Software Environments and Tools".* [ed.] Jose C. Cunha and Omer F. Rana (Eds). s.l. : Springer Verlag, 2006.

64. *IC2D: Interactive Control and Debugging of Distribution.* **Baude, F.; Bergel, A.; Caromel, D.; Huet, F.; Nano, O.; Vayssière, J.;.** 2001. Lecture Notes In Computer Science. Vol. 2179, págs. 193-200.

65. *Autopilot: adaptive control of distributed applications.* **Ribler, R. L., y otros.** 1998. Proceedings of the Seventh IEEE Symposium on High-Performance Distributed Computing. págs. 172–179.

66. *Application monitoring in the grid with GRM and PROVE.* **Balaton, Z.; Kacsuk, P.; Podhorszki, N.;.** San Francisco (USA) : Lecture Notes in Computer Science, 2001. Proceedings of the International Conference on Computational Science (ICCS2001). Vol. 2073, págs. 253-262.

67. *Job monitoring and steering in D-Grid's High Energy Physics Community Grid.* **Lorenz, D.; Borovac, S.; Buchholz, P.; Eichenhardt, H.; Harenberg, T.; Mättig, P.; Mechtel, M.; Müller-Pfefferkorn, R.; Neumann, R.; Reeves, K.; Uebing, Ch.; Walkowiak, W.; William, Th.; Wismüller, R.;.** 2009, Future Generation Computer Systems, Vol. 25(3), págs. 308-314.

68. *Pegasus: Mapping Scientific Workflow onto the Grid.* **Deelman, E.; Blythe, J.; Gil, Y.; Kesselman, C.; Mehta, G.; Patil, S.; Su, M. H.; Vahi, K.; Livny, M.;.** Nicosia (Cyprus) : s.n., 2004. Across Grids Conference.

69. **Taylor, I.; Shields, M.; Wang, I.; Harrison, A.;.** The Triana Workflow Environment: Architecture and Applications. *Workflows for e-Science.* s.l. : Springer, 2007.

70. *The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid.* **Allen, G.; Davis, K.; Goodale, T.; Hutanu, A.; Kaiser, H.; Kielmann, T.; Merzky, R.; van Nieuwpoort, R.; Reinefeld, A.; Schintke, F.; Schütt, T.; Seidel, E.; Ullmer, B.;.** 2005. Proceedings of the IEEE. Vol. 93(3), págs. 534-550.

71. *gridMonSteer: Generic Architecture for Monitoring and Steering Legacy Applications in Grid Environments.* **Wang, I.; Taylor, I.; Goodale, T.; Harrison, A.; Shields, M.;.** 2006. Proceedings of the UK e-Science All Hands Meeting.

72. Barcelona Supercomputing Center. [En línea] [Citado el: 4 de 5 de 2012.] http://www.bsc.es.

73. *COMP Superscalar: Bringing GRID Superscalar and GCM together.* **Tejedor, E.; Badia, R. M.;.** Lyon (France) : s.n., 2008. 8th IEEE International Synposium on Cluster Computing and Grid.

74. *ClusterSs: A Task-Based Programming Model for Clusters.* **Tejedor, E.; Farreras, M.; Grove, D.; Badia, R. M.; Almasi, G.; Labarta, J.;.** s.l. : ACM, 2011. Proceedings of the 20th international symposium on High performance distributed computing (HPDC '11).

75. *A dependency-aware task-based programming environment for multi-core architectures.* **Pérez, J. M.; Badia, R. M.; Labarta, J.;.** Tsukuba (Japan) : s.n., 2008. IEEE International Conference on Cluster Computing (Cluster 2008). págs. 142-151.

76. *CellSs: Making it easier to program the Cell Broadband Engine processor.* **Pérez, J. M., y otros.** 2007, IBM Journal of Research and Development, Vol. 51 (5), págs. 593-604.

77. *An Extension of the StarSs Programming Model for Platforms with Multiple GPUs.* **Ayguadé, E., y otros.** Delft (Netherlands) : s.n., 2009. 15th International Euro-Par Conference. págs. 851-862.

78. *A Component-based Integrated Toolkit.* **Tejedor, E.; Badia, R. M.; Kielmann, T.; Getov, V.;.** Heraklion (Greece) : s.n., 2007. CoreGrid Workshop on Grid Programming Model, Grid and P2P Systems Architecture, Grid Systems, Tools and Environments. págs. 212-215.

79. *The Asynchronous Partitioned Global Address Space Model.* **Saraswat , V.; Almasi , G.; Bikshandi , G.; Cascaval , C.; Cunningham , D.; Grove , D.; Kodali , S.; Peshansky , I.; Tardieu, O.;.** Toronto (Canada) : s.n., 2010. The First Workshop on Advances in Message Passing (co-located with PLDI 2010).

80. *Parameter Space Exploration Using Scientific Workflows.* **Abramson, D.; Bethwaite, B.; Enticott, C.; Garic, S.; Peachey, T.;.** s.l. : Springer-Verlag Berlin Heidelberg, 2009. págs. 104-113. LNCS 5544.

81. SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs. [En línea] [Citado el: 4 de 5 de 2012.] http://www.shiwa-workflow.eu/.

82. Top 500 Supercomputer Sites. [En línea] [Citado el: 4 de 5 de 2012.] http://www.top500.org/.

83. *Derivation of Self-Scheduling Algorithms for Heterogeneous Distributed Computer Systems: Application to Internet-based Grids of Computers.* **Díaz, J.; Reyes, S.; Niño, A.; Muñoz-Caro, C.;.** 2009, Future Generation Computer Systems, Vol. 25 (6), págs. 617-626.

84. The Message Passing Interface (MPI) standard. [En línea] [Citado el: 4 de 5 de 2012.] http://www.mcs.anl.gov/research/projects/mpi/.

85. gLite - Lightweight Middleware for Grid Computing. [En línea] [Citado el: 4 de 5 de 2012.] http://glite.cern.ch.

86. UNICORE (Uniform Interface to Computing Resources). [En línea] [Citado el: 4 de 5 de 2012.] http://www.unicore.eu.

87. The Kepler Project. [En línea] [Citado el: 4 de 5 de 2012.] https://kepler-project.org.

88. Pegasus Home Page. [En línea] [Citado el: 4 de 5 de 2012.] http://pegasus.isi.edu.

89. Taverna Home Page. [En línea] [Citado el: 4 de 5 de 2012.] http://www.taverna.org.uk.

90. Triana Home Page. [En línea] [Citado el: 4 de 5 de 2012.] http://www.trianacode.org.

91. GRIDSs Home Page. [En línea] [Citado el: 4 de 5 de 2012.] http://www.bsc.es/grid/gridsuperscalar.

92. Ninf-G Home Page. [En línea] [Citado el: 4 de 5 de 2012.] http://ninf.apgrid.org.

93. ProActive Parallel Suite. [En línea] [Citado el: 4 de 5 de 2012.] http://proactive.inria.fr.

94. **Mattson, T. G.; Sanders, B. A.; Massingell, B. L.;.** *Patterns for Parallel Programming.* s.l. : Addison-Wesley Professional, 2004.

95. **Jensen, F.** *Introduction to Computational Chemistry.* s.l. : John Wiley&Sons, 1999. ISBN: 0471980554.

96. GAMESS Home Page. [En línea] [Citado el: 4 de 5 de 2012.] http://www.msg.ameslab.gov/gamess/gamess.html.

97. High-Performance Computational Chemistry Software. [En línea] [Citado el: 4 de 5 de 2012.] http://www.nwchem-sw.org.

98. *General atomic and molecular electronic structure system.* **Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S.; Windus, T. L.; Dupuis, M.; Montgomery, J. A.;.** 14, 1993, Journal Of Computational Chemistry, págs. 1347-1363.

99. *A detailed MPI communication model for distributed systems.* **Le, T. T.; Rejeb, J.;.** 3, 2006, Future Generation Computer Systems, Vol. 22, págs. 269-278.

100. *Mach 1000 Fortran Compiler Reference (revision 1.0 edn).* Cambridge, MA : BBN Advanced Computers Inc., 1998.

101. *Guided Self-Scheduling: a Practical Scheduling Scheme for Parallel Supercomputers.* **Polychronopoulos, C. D.; Kuck, D.;.** 1987, IEEE Transactions on Computers, Vol. 36, págs. 1425-1439.

102. *Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers.* **Tzen, T. H.; Ni, L. M.;.** 1993, IEEE Transactions Parallel Distributed System, Vol. 4, págs. 87-98.

103. *Validity of the single processor approach to achieving large scale computing capabilities.* **Amdahl, G. H.** [ed.] Spring Joint Computer Conference. Atlantic City : IEEE Computer Society Press, 1967. Proceedings of the AFIPS Conference. págs. 483-485.

104. The OpenMP® API specification for parallel programming. [En línea] [Citado el: 4 de 5 de 2012.] http://openmp.org.

105. *The Anatomy of the Grid: Enabling scalable virtual organizations.* **Foster, I.; Kesselman, C.; Tuecke, S.;.** 2001, International Journal of High Performance Computing Applications, Vol. 15, págs. 200-222.

106. *Derivation of self-scheduling algorithms for heterogeneous distributed computer systems: Application to internet-based grids of computers.* **Díaz, J.; Reyes, S.; Niño, A.; Muñoz-Caro, C.;.** 2009, Future Generation Computer Systems, Vol. 25(6), págs. 617-626.

107. *Automatic Grid workflow based on imperative programming languages.* **Sirvent, R.; Pérez, J. M.; Badia, R. M.; Labarta, J.;.** 2006, Concurrency and Computation: Practice and Experience, Vol. 18, págs. 1169-1186.

108. Open Grid Forum. [En línea] [Citado el: 4 de 5 de 2012.] http://www.gridforum.org/.

109. *A Resource Management Architecture for Metacomputing Systems.* **Czajkowski, K.; Foster, I.; Karonis, N.; Kesselman, C.; Martin, S.; Smith, W.; Tuecke, S.;.** 1998. Proceedings of the Workshop on Job Scheduling Strategies form Parallel Processing. Vol. 1459, págs. 62–82.

110. The Perl Programming Language. [En línea] [Citado el: 4 de 5 de 2012.] http://www.perl.org.

111. The Community ENTerprise Operating System. [En línea] [Citado el: 4 de 5 de 2012.] http://www.centos.org/.

112. Rocks Clusters. [En línea] [Citado el: 4 de 5 de 2012.] http://www.rocksclusters.org.

113. Oracle Grid Engine. [En línea] [Citado el: 4 de 5 de 2012.] http://www.oracle.com/technetwork/oem/grid-engine-166852.html.

114. OpenPBS. [En línea] [Citado el: 4 de 5 de 2012.] http://www.mcs.anl.gov/research/projects/openpbs/.

115. *Structural and vibrational theorical analysis of protonated formaldehyde in its X1A' ground electronic state.* **Castro, M. E.; Niño, A.; Muñoz-Caro, C.;.** 2008, Theoretical Chemistry Accounts, Vol. 119, págs. 343-354.

116. *Graph-based task replication for workflow applications.* **Sirvent, R.; Badia, R. M.; Labarta, J.;.** Washington, DC, USA : IEEE Computer Society, 2009. In Proceedings of the 2009 11th IEEE International Conference on High Performance Computing and Communications (HPCC '09). págs. 20-28.

8 Apéndice A

A continuación se incluye la implementación de la función *componerAcetona* utilizada en el apartado 4.3 de este manuscrito.

```
void componerAcetona(int n, int *j, double **param){
  string filename;
  ofstream outfile;
  ostringstream buf;

  // Teniendo en cuenta que la simetría es no rígida
  if (param[2][j[2]]==param[3][j[3]] && param[1][j[1]]>param[0][j[0]]) {
     // No hacer nada
  }
  else {
        // Asignar nombre y abrir el fichero
        outfile.open(filename.c_str());

        outfile<<" $CONTRL SCFTYP=RHF RUNTYP=OPTIMIZE MPLEVL=2 COORD=ZMT "<<endl;
        outfile<<" MAXIT=50 NPRINT=-5 NZVAR=24 ICHARG=0 $END            "<<endl;
        outfile<<" $SYSTEM TIMLIM=100000 MWORDS=8 MEMDDI=20             "<<endl;
        outfile<<"         PARALL=.TRUE. $END                           "<<endl;
        outfile<<" $SCF DIIS=.TRUE. ETHRSH=10.0 NPUNCH=0 $END           "<<endl;
        outfile<<" $STATPT IFREEZ(1)=3, 5, 9, 18                        "<<endl;
        outfile<<"         METHOD= RFO NSTEP= 120 $END                  "<<endl;
        outfile<<" $BASIS  GBASIS=N311 NGAUSS=6 NDFUNC=2 NPFUNC=1 $END   "<<endl;
        outfile<<" $DATA                                                "<<endl;
        outfile<<"  Acetona MP2/6-311G(2d,p) fichero "
        <<setprecision(4)<<"  "<<param[2][j[2]]<<"  "<<param[3][j[3]]<<"  "
                  <<param[0][j[0]]<<"  "<<param[1][j[1]]<<endl;
        outfile<<" C1  1                                              "<<endl;
        outfile<<" O                                                 "<<endl;
        outfile<<" C  1  R21                                            "<<endl;
        outfile<<" C  2  R32  1  ALPHA                                       "<<endl;
        outfile<<" C  2  R42  1  BETA  3  D4213                           "<<endl;
        outfile<<" H  4  R54  2  A542  1  TETA1                           "<<endl;
        outfile<<" H  4  R64  2  A642  5  D6425                          "<<endl;
        outfile<<" H  4  R74  2  A742  5  D7425                          "<<endl;
        outfile<<" H  3  R83  2  A832  1  TETA2                           "<<endl;
        outfile<<" H  3  R93  2  A932  8  D9328                          "<<endl;
        outfile<<" H  3  R103 2  A1032 8  D10328                           "<<endl;
        outfile<< endl;
        outfile<<" R21 = 1.217    "<<endl;
        outfile<<" R32 = 1.513    "<<endl;
        outfile<<" R42 = 1.513    "<<endl;
        outfile<<" R54 = 1.088    "<<endl;
        outfile<<" R64 = 1.093    "<<endl;
        outfile<<" R74 = 1.093    "<<endl;
        outfile<<" R83 = 1.088    "<<endl;
        outfile<<" R93 = 1.093    "<<endl;
        outfile<<" R103 = 1.093   "<<endl;
        outfile<<" ALPHA = "<<param[2][j[2]]<<endl;
        outfile<<" BETA = " <<param[3][j[3]]<<endl;
        outfile<<" A542 = 109.7   "<<endl;
        outfile<<" A642 = 110.1   "<<endl;
        outfile<<" A742 = 110.1   "<<endl;
        outfile<<" A832 = 109.7   "<<endl;
        outfile<<" A932 = 110.1   "<<endl;
```

```
        outfile<<" A1032 = 110.1  "<<endl;
        outfile<<" D4213 = 180.0  "<<endl;
        outfile<<" D6425 = 120.0  "<<endl;
        outfile<<" D7425 = -120.0 "<<endl;
        outfile<<" D9328 = 120.0  "<<endl;
        outfile<<" D10328 = -120.0 "<<endl;
        outfile<<" TETA1= "<<param[0][j[0]]<<endl;
        outfile<<" TETA2= "<<param[1][j[1]]<<endl;
        outfile<<" $END       ";
        outfile<<endl<<endl<<endl;

        outfile.close();  // Cerrar outfile
   } // Final if
} // Final componerAcetona
```