Departamento de Arquitectura de Computadores y Sistemas Operativos (CAOS/DACSO)

Escuela Técnica Superior de Ingeniería (ETSE)
Universidad Autónoma de Barcelona

Master Computación en Altas Prestaciones

Módulo: Inicio a la investigación y trabajo fin de master.

Curso 2008/2009

Memoria de investigación:

GAPS: Generador de aplicaciones sintéticas

Autor: Marcela Castro León

Director: Eduardo César Galobardes

Barcelona, Julio 2009.

Indice General

Indice de Figuras

Indice de Cuadros

1. Introducción

1.1. Resumen

El presente trabajo describe el desarrollo de un generador de aplicaciones paralelas sintéticas de estructura compuesta del tipo Master/Worker donde estos últimos funcionan como pipeline.

Dentro del grupo de investigación en Herramientas de Análisis y Sintonización de prestaciones, se desarrolla en estos momentos un modelo de rendimiento para este tipo de aplicaciones.

Dicha investigación se basa en que es posible generar modelos de rendimiento asociados a la estructura de la aplicación para automatizar las tareas de monitorización, análisis y sintonización durante la ejecución de la aplicación, utilizando técnicas de sintonización dinámica.

Cabe recordar que el objetivo que persiguen estos estudios es minimizar el tiempo de ejecución de las aplicaciones, haciendo un uso eficiente de los recursos.

Para que la herramienta de sintonización dinámica pueda funcionar, se necesita, como primer paso, conocer el modelo de rendimiento asociado a la estructura. Para facilitar el estudio, se utilizan patrones de aplicaciones como Master/Worker, Pipeline, SPMD, Divide & Conquer, etc. Para cada caso, se desarrollan las funciones de performance que describen el comportamiento en términos de tiempos de ejecución, y se definen los puntos de medición y las acciones de cambio de parámetros que tiene que tomar la herramienta de tuning para resolver el problema de rendimiento detectado [12].

Para comprobar las conclusiones de estos modelos, que se llevan a cabo en forma analítica, se utilizan normalmente aplicaciones sintéticas para poner a prueba las hipótesis del investigador. Esta técnica permite además, mostrar los resultados en un escenario real donde se pueden estudiar las variaciones de las variables consideradas.

Por ejemplo, una de las variables que se sintonizan en el modelo para aplicaciones Master/Worker es la del número de workers.

Para tomar una buena decisión acerca de la cantidad óptima, es necesario definir indicadores como la disminución del tiempo de ejecución por cada uno de los workers agregados o la relación del speedup observado con el ideal.

Estos indicadores, que normalmente se definen por umbrales inferior y superior, son los que permitirán tomar una buena decisión al fijar el valor de esta variable durante la sintonización dinámica.

Para establecer los umbrales estos indicadores, se requiere un conocimiento profundo de la aplicación en su ejecución. Al trabajar con patrones genéricos, se puede realizar una aplicación sintética que emule al conjunto de aplicaciones originales que representa y estudiar como se comporta el modelo con cada valor de parámetros durante las ejecuciones.

El presente trabajo explica el desarrollo, la implementación y pruebas de un generador de aplicaciones sintéticas que denominamos ***Gaps***. Esta herramienta permite generar una aplicación paralela sintética que emula una de estructura compuesta del tipo Master/Worker dónde estos últimos funcionan como pipelines.

En el ámbito de la línea de investigación de análisis de rendimiento y sintonización de aplicaciones paralelas del departamento, se han desarrollado modelos de rendimiento para aplicaciones Master/Worker [4] y por otro lado para pipeline [7]. En este momento, se esta investigando un modelo compuesto para lo que se utilizará **Gaps**.

El código generado, en C con paso de mensajes a través de MPI (Message Passing Interface), se basa en los parámetros de entrada que definen el modelo de aplicación a generar, como la cantidad de procesos workers, el tamaño de tareas, el número de etapas de los pipelines, y atributos de cada una de estas: tiempo de ejecución, varianza del tiempo de ejecución, si esta agrupada o replicada, etc.

La herramienta permite modelar aplicaciones de estructura compuesta, a través de la definición de los principales parámetros de la estructura y de funcionamiento que el investigador necesitará ir cambiando en sus sucesivas pruebas durante la investigación.

El director de la tesina es Eduardo César, miembro del grupo que desarrolla la línea de investigación de análisis de rendimiento y sintonización de aplicaciones paralelas y distribuidas del departamento de Arquitectura de Computadores y Sistemas Operativos (CAOS), a quién agradezco su brillante y generosa dedicación y colaboración.

1.2. Motivación

La línea de investigación de análisis de rendimiento y sintonización de aplicaciones paralelas y distribuidas del departamento de Arquitectura de Computadores y Sistemas Operativos (CAOS), desarrolla un modelo de

sintonización dinámica para aplicaciones de estructura compuesta Master/Worker de pipelines.

Aunque esta línea de investigación posee en su haber un conjunto robusto de trabajos anteriores bien consolidados como la herramienta MATE (Monitoring, Analysis and Tuning Environment), para realizar la sintonización dinámica de aplicaciones paralelas [12], y el desarrollo del framework Poetries (Performance Oriented Environment for Transparent Resource-Management, Implementing End-user parallel aplications) [10], y el trabajo doctoral de Eduardo Cesar del 2006, "Definition of Framework-based Performance Models for Dynamic Performance Tuning" [11] y, más avances como "Modeling master/worker applications for automatic performance tuning" [4], "Dynamic Pipeline Mapping (DPM)" [7], "Automatic tuning of data distribution using factoring in master/worker application" [3] y "*Modeling pipeline applications in POETRIES*" [2], constituye un reto en sí concluir un modelo robusto para la estructura compuesta de Master/Worker de pipelines.

El modelo de rendimiendo para un patrón de aplicación compuesta Master/Worker de pipelines, persigue un objetivo general de conseguir un mejor tiempo de ejecución, haciendo un uso eficiente de recursos.

Para conseguirlo, tiene que tomar las decisiones adecuadas acerca de la distribución de tareas entre los miembros, la cantidad de procesos por procesador, la cantidad de procesos y/o procesadores realizando una tarea, el tamaño de la tarea, etc.

El camino elegido para obtener conclusiones sólidas y de amplio alcance es someterlas a prueba en un conjunto de aplicaciones sintéticas que representen una parte significativa del universo que se estudia.

Para la definición y el desarrollo de los modelos mencionados en los trabajos anteriores se han sido utilizados simuladores y generadores de aplicaciones sintéticas con el objetivo de validar los resultados obtenidos.

Es lógico pensar en utilizar la misma vía para la validación de modelos compuestos, pero el tipo de modelos es en este caso, en sí mismo, una tarea compleja. Esta es la principal razón de haber propuesto este desarrollo como un proyecto de investigación independiente.

1.3. Estado del Arte

Hay muchos grupos de investigación que realizan estudios de rendimiento y sintonización de aplicaciones paralelas, y, que en lugar de tomar aplicaciones reales, construyen aplicaciones sintéticas parametrizables que puedan controlarse mejor, pero normalmente no se publican y se tienen para uso interno.

El grupo de investigación INRIA, dirigido por el Dr. Thierry Priol, en Francia, utiliza este tipo de herramienta para hacer pruebas de sus modelos de Master/Worker con componentes.

También el grupo de la Universidad de La Laguna utiliza un generador de aplicaciones para experimentación.

Revisando en las publicaciones de ACM, podemos destacar trabajos relacionados al presente en el objetivo final:

[5] Aurora: An Approach to High Throughput Parallel Simulation

[1] ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics.

Hay que tener en cuenta que para las pruebas de hipótesis de modelos analíticos se utiliza:

(1) **Simuladores:** En este caso se toma en cuenta los parámetros del modelo y se pueden validar las predicciones pero no es útil para medir o detectar la influencia de elementos externos. En este tipo desarrollos, cabe destacar Simba [6], de la Universidad del Texas, relacionado con el presente trabajo en que sirve para predecir el comportamiento de rendimiento, pero no esta orientado a aplicaciones paralelas.

(2) **Aplicaciones sintéticas:** Se comprueba en entornos reales y, aunque se controlan todos los parámetros del modelo, si hay algún otro factor externo que influye en el comportamiento de la aplicación aquí saldrá, y además, en un entorno real se ve como va variando la latencia o el ancho de banda, e incluso el tiempo de ejecución. Se observa cómo afectan al modelo estas variaciones.

(3) **Benchmarks, programas conocidos y muy analizados**.

(4) **Aplicaciones reales**, como prueba final.

1.4. Objetivo

El objetivo es desarrollar una herramienta que permita generar aplicaciones sintéticas de estructura compuesta del tipo Master/Worker de pipelines.

El uso principal, al menos el primero, será comprobar la estrategia de sintonización automática de aplicaciones de estructura compuesta que se esta desarrollando.

En la Figura 1 se representa un modelo de aplicación Master/Worker de pipelines. Tiene un número N de workers, y a su vez, cada uno de estos esta formado por un número diferente de etapas o tareas secuenciales (pipeline).

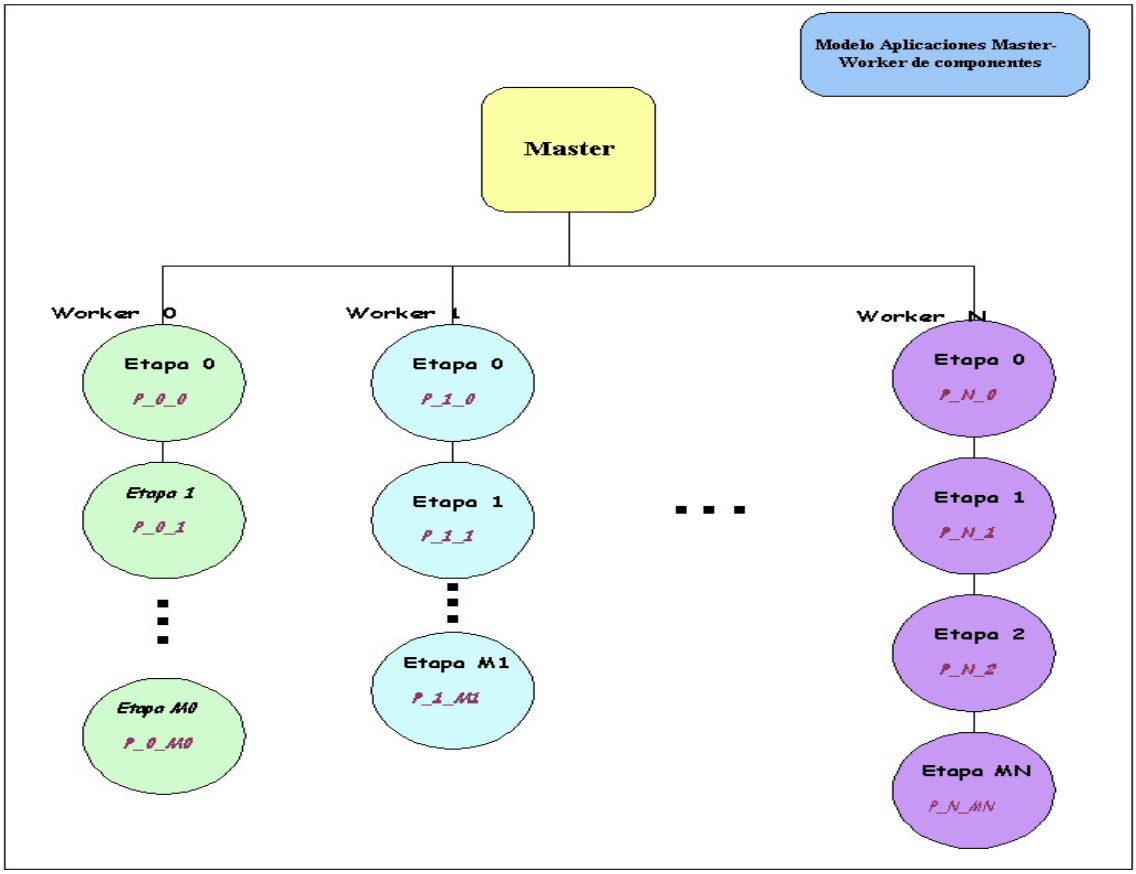

Figura 1. Esquema de una aplicación Master/Worker de pipelines

Teniendo en cuenta que el investigador utilizará el modelo de la aplicación sintética para comprobar y cuantificar en tiempo las acciones de optimización del modelo de rendimiento variando los parámetros, la herramienta tiene que tener las siguientes opciones de diseño:

- Permitir modificar el número de workers de la aplicación.
- Permitir modificar el tamaño de tareas a computar y comunicar entre Master/Worker y etapas del pipeline.
- Permitir definir el número de tareas iniciales de los workers, que permitirán un uso eficiente de las secuencia de etapas del pipeline,

estando ocupados todos la mayor parte del tiempo en la medida que el trabajo inicial tenga longitud suficiente.

- Permitir definir qué etapas del pipeline se ejecutan en un mismo procesador. A este mecanismo se lo conoce como agrupamiento, y se realiza para optimizar el pipe y el uso de recursos en aquellos casos de secuencias de etapas rápidas en procesadores con capacidad ociosa.

- Permitir definir qué etapas se replican en procesadores diferentes, y con cuantas copias. Esto se aplica para mejorar el tiempo total de la secuencia, que esta limitado por la duración de la etapa más lenta. Al ejecutarse en forma replicada, mejorará el tiempo, reduciéndose, en el mejor de los casos, en forma proporcional al número de copias.

- Permitir definir el tiempo medio de duración de cada etapa. Para lograr un comportamiento más real incluirá una varianza del mismo.

1.5. Organización del trabajo

En la introducción del presente trabajo se expone el contexto en que será utilizado este desarrollo, se explican los motivos que lo impulsaron, el objetivo a conseguir, y se mencionan algunos trabajos relacionados dentro del ámbito de la investigación de rendimiento de aplicaciones.

Luego, siguiendo el orden de las fases del ciclo de vida de desarrollo de software, se explica el análisis, el diseño, la implementación y las pruebas del generador Gaps.

Para terminar, se propone una serie de mejoras a la herramienta en trabajos futuros y las conclusiones de todo el desarrollo del trabajo de investigación.

2. Análisis

2.1 Resumen

El presente capítulo explica los pasos de análisis del problema que se ha realizado hasta llegar a una solución que cumpla con los requisitos y los objetivos trazados.

Durante el análisis se detectan y explican en detalle los requerimientos que tiene que cubrir la solución. Se hace hincapié en los puntos más críticos y/o complejos, que en este caso, corresponde a los casos de agrupamientos y réplicas de procesos, que es fundamental que se resuelvan correctamente.

Se plantean también los problemas y a afrontar con las herramientas de desarrollo seleccionadas, en este caso C con MPI, observando la forma de distribución de procesos en las diferentes implantaciones, de modo que la aplicación pueda resolver la mayoría de los casos.

El capítulo finaliza con una propuesta de solución por fases, dejando en forma detallada los pasos a realizar en cada una para tener la aplicación sintética que se pretende.

2.2 Requerimientos

La idea principal del uso del generador es desarrollar el modelo de rendimiento de la estructura compuesta Master/Worker de pipelines, siguiendo la misma metodología que se utilizó para las aplicaciones Master/Worker [4] y para las de estructura pipeline [7].

Dicha metodología se basa en definir un modelo de rendimiento que permita implementar en forma simple el proceso de sintonización dinámica.

Un modelo de rendimiento se define como una expresión matemática que representa aspectos específicos claves dentro de un sistema de computación. Permite que se pueda analizar el comportamiento de la aplicación durante su ejecución de modo de interferir para conseguir el objetivo de disminuir el tiempo de ejecución haciendo un uso eficiente de recursos.

El modelo tiene que incluir:

- Funciones de rendimiento para detectar los puntos críticos (performance functions).
- Puntos de medición para evaluar el modelo (measure points).
- Parámetros de sintonización (tuning points).

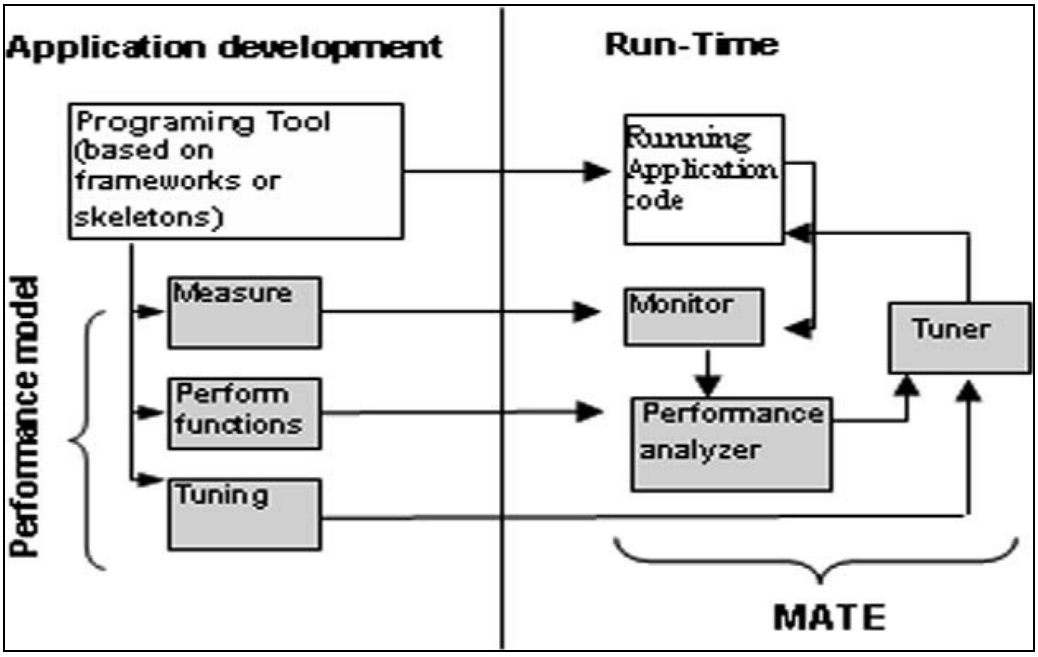

Figura 2. Esquema de Modelo de Rendimiento para sintonización dinámica

Una vez definido el modelo de rendimiento, en tiempo de ejecución se lo utiliza para monitorizar los puntos de medida y evaluarlos (performance analyzer) y basado en los resultados se realizan los acciones requeridas para mejorar el rendimiento (tuner). La herramienta que permite realizar este proceso se llama MATE: Monitoring, Analysis and Tuning Environment). [8] [12].

La Figura 2 muestra el proceso completo del desarrollo del modelo de rendimiento en tiempo de desarrollo y la ejecución con MATE.

En el caso de aplicaciones Master/Worker, el master envía tareas a cada uno de los workers, que a su vez realizan un proceso computacional durante un tiempo y le envían el resultado al master. Dependiendo de la naturaleza del problema, el Master deberá esperar una respuesta de **todos** antes de enviar una nueva tarea, por lo que podemos considerar que se divide en "iteraciones". Este es el caso de dependencias de datos entre tareas, "separable dependencies or geometric decomposition patterns", según terminología definida en [9]. El otro caso es en que el

master espera a que **cada** worker termine para enviarle la siguiente tarea, y se lo conoce como "tareas independientes" o "embarrassingly parallel pattern" [9].

El modelo de rendimiento del Master/Worker [4] consiste en una combinación de la estrategia de conseguir un ***balanceo de carga entre los workers*** y la de ***definir un número adecuado*** de estos.

Para conseguir un balanceo de carga adecuada se seleccionaron tres estrategias de distribución [4] [11]:

- **Fixed Size Chunking (FSC):** Se divide la tarea en porciones iguales. En este caso, se ha de decidir la cantidad de partes para conseguir un buen balanceo.

- **Dynamic Predictive Factoring (DPF):** Consiste en aplicar un mismo factor de partición en cada iteración, de modo que el tamaño de las tareas va decreciendo en cada una. Se trata de encontrar el mejor factor para determinar la porción de tarea restante que será distribuida la siguiente iteración.

- **Dynamic Adjusting Factoring (DAF):** Es como el anterior pero que se evalúa en cada iteración de acuerdo al balanceo de carga de ese momento.

Se ha de decidir la política de distribución de tareas entre workers para conseguir el balance adecuado para mejorar la performance, controlando la relación entre tiempo de procesamiento y el de comunicaciones. Las tareas más grandes minimizan los tiempos de comunicaciones y aumentan el procesamiento y las tareas pequeñas, a la inversa.

El generador tiene que permitir variar el tamaño y la cantidad de tarea que el master envía a los workers y la cantidad de iteraciones que realiza el proceso para que el investigador pueda experimentar con estas estrategias de distribución y balanceo de carga.

El número de workers se necesita adaptar porque impacta directamente en el tiempo total de ejecución, que es lo que busca aumentar la función de rendimiento del modelo. En forma teórica, asumiendo que no hay tiempo de comunicaciones y suponiendo un balance de carga perfecto y que la aplicación escala en forma ideal, el tiempo total de ejecución se calcula como la duración de la aplicación en modo secuencial dividido el número de workers.

Sin embargo, en la realidad, el speedup de la aplicación decrece a medida que se asignan más recursos, indicando una pérdida de eficiencia. Esto ocurre debido al deterioro de la relación computo/comunicación y a la saturación del Master. El modelo de rendimiento tiene que tener en

cuenta estos factores para una buena decisión en el uso eficiente de recursos.

Por lo tanto, el generador de aplicaciones sintéticas tiene que permitir variar el número de workers para que el investigador pueda experimentar, observar y analizar, sobre todo, el tiempo de ejecución con cada valor.

Por otro lado, en [7] y [11] se estudió y se estableció el modelo de rendimiento para aplicaciones del tipo pipeline.

En este caso el tiempo total de ejecución está limitado por la duración de la etapa más lenta. La propuesta para realizar la sintonización dinámica para este tipo de aplicaciones, se basa en:

- Agrupar las etapas consecutivas más rápidas en un único procesador (se aplica a etapas consecutivas para tener menos costo de comunicación).

- Replicar las etapas más lentas en varios procesadores para incrementar la cantidad de tareas procesadas en todo el pipeline.

Es por esto que el generador tiene que permitir definir que etapas consecutivas se agrupan y cuáles se replican para que el investigador pueda probar estas soluciones en el modelo compuesto.

2.3 Especificación

El generador de aplicaciones sintéticas ***Gaps*** tiene que permitir modelar una variedad importante de estructuras combinadas de Master/Worker de pipelines y definir las propiedades de comportamiento de cada proceso.

Las características que tiene que tener para cumplir los requerimientos son:

- El aplicativo esta escrito en lenguaje C (Gnu) para permitir portabilidad a distintos ambientes.

- Los procesos paralelos se comunican con paso de mensajes MPI. (Message Passing Interface).

- Permite generar aplicativos paralelos de un gran número de workers compuestos por múltiples etapas.

- Cada worker puede tener un número diferente de etapas que el resto.
- Admite definir etapas replicadas en diferentes procesadores.
- Permite agrupar etapas consecutivas en un mismo procesador.
- El master enviará a los workers varias tareas de tamaño fijo.
- Las últimas etapas de los workers envían al master un mensaje de finalización de tamaño fijo.

2.4 Agrupamiento

Las etapas se agrupan en una cantidad "K", cuando estos "K" procesos consecutivos se ejecutan en un mismo procesador. Se aplica en los procesos que son rápidos en finalizar sus tareas para aprovechar recursos ociosos en el procesador.

En la Figura 3 se muestra un esquema con dos agrupamientos, uno en el worker 1 de tres etapas y otro en el worker 2 de dos etapas. *Sólo se agrupan etapas consecutivas del pipeline*. Este mecanismo permite disminuir costos en la comunicación y, sobre todo, liberar procesadores que se pueden utilizar para réplicas de las etapas más lentas.

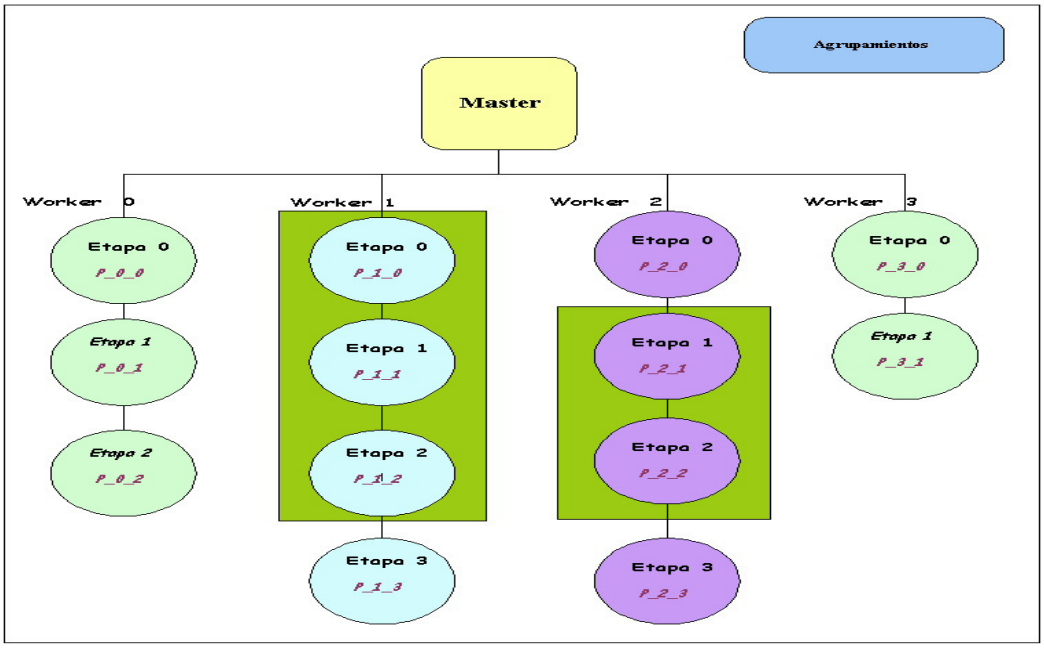

Figura 3. Esquema de aplicación Master/Worker con etapas agrupadas

2.5 Réplicas

Las etapas se replican en una cantidad "Q", cuando una misma etapa se ejecuta "Q" veces en "Q" procesadores diferentes. Se aplica en los procesos de mayor carga de procesamiento y/o de mayor duración, con el propósito de dividir la carga en "Q" procesadores y así mejorar el rendimiento general.

En la Figura 4 se muestra un esquema de una etapa con dos replicas: La etapa 3 del worker 1 se ejecutará en dos procesadores, y la etapa 3 del worker 2 en tres.

Al tener más de una copia de cada etapa, es necesario realizar una gestión de reparto entre las "Q" réplicas de los mensajes que van dirigidos a esta etapa y una gestión de unión con los mensajes que las "Q" réplicas envían a la siguiente etapa.

Por esta razón, añadimos un nuevo proceso que se encargará de gestionar las comunicaciones desde y hacia las réplicas que denominamos "Comunication Manager (CM)".

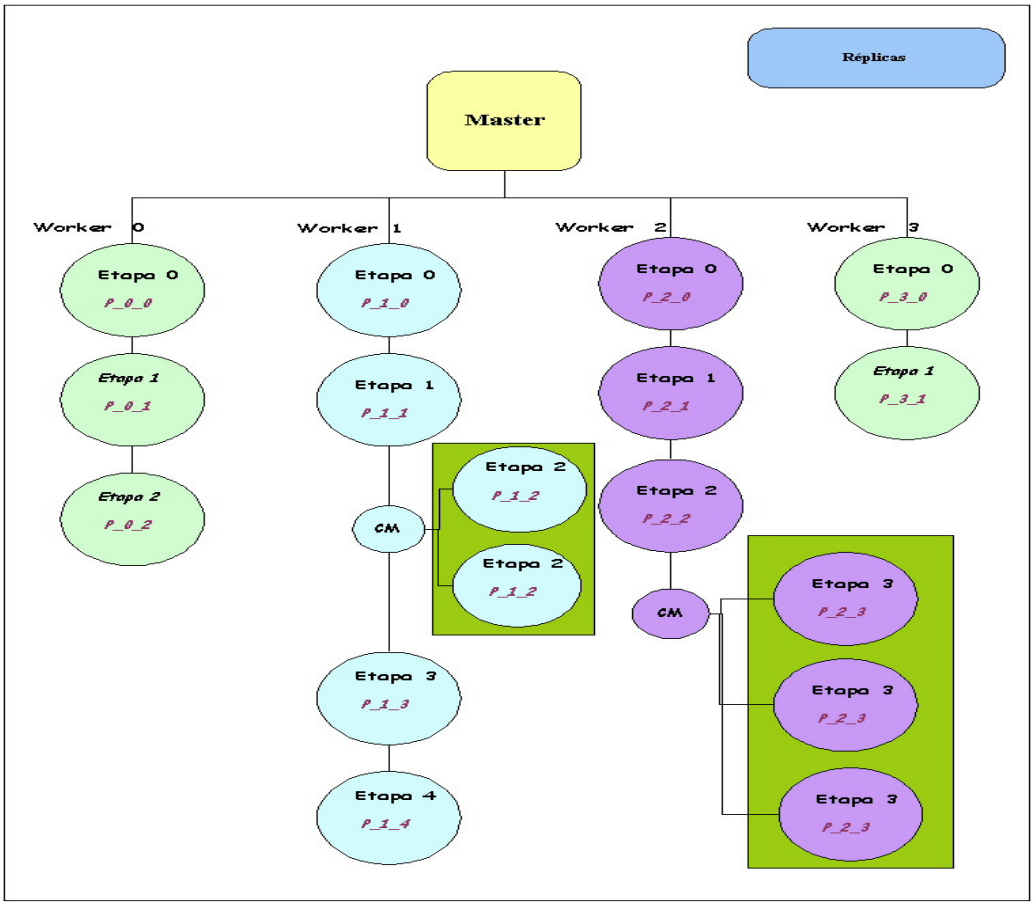

Figura 4. Esquema de aplicación Master/Worker con etapas replicadas

2.6 Gestor de comunicaciones "Comunication Manager"

Por cada réplica que se defina, el sistema genera un proceso adicional que gestiona los mensajes que recibe desde el proceso origen de la etapa y lo envía a una réplica libre. Cada réplica que finaliza, envía el mensaje al gestor en lugar de enviarlo a la etapa siguiente, y es este el que se encarga del reenvío.

La carga de procesamiento esperada de este proceso es baja porque sólo recibe y envía mensajes.

La siguiente Figura 5 muestra un esquema de las operaciones del CM: recibe mensajes desde dos canales diferentes, de la etapa anterior o del master en caso de ser primera etapa y desde las réplicas que gestiona, y envía mensajes a dos canales diferentes: a los procesos réplica que gestiona y a la etapa siguiente el mensaje procesado.

El CM no debe alterar el comportamiento del pipeline, así que no puede recibir un mensaje desde la etapa anterior si no existe al menos una réplica libre.

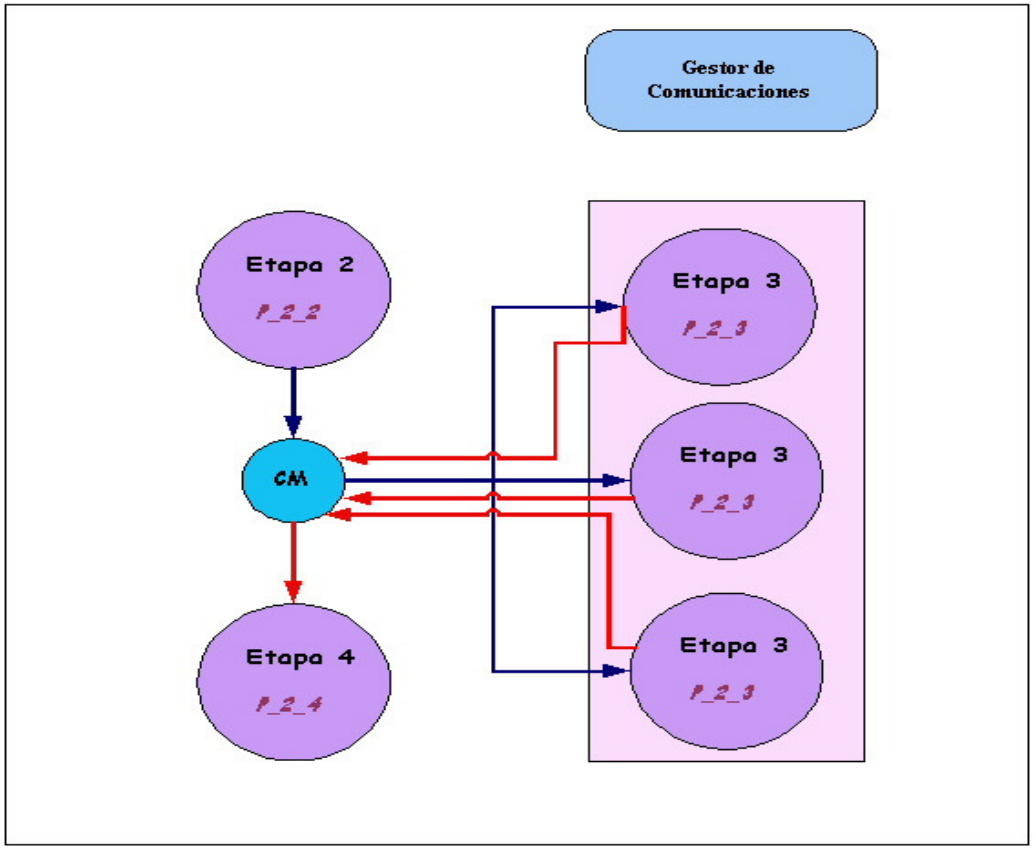

Figura 5. Operaciones del proceso Comunication Manager (CM)

2.7 Problemas con la librería de comunicaciones

El generador de la aplicación sintética necesita conocer la cantidad de procesos paralelos que tiene que generar y la función de cada uno de estos: master, etapa normal, etapa replicada o agrupada.

El usuario conoce el modelo de aplicación que quiere generar, pero:

¿Cómo lograr que mpi deje los procesos agrupados en un mismo nodo?

¿Cómo lograr que los procesos replicados estén en nodos diferentes?

El mecanismo que mpi utiliza para generar procesos en los nodos disponibles es que reparte en forma uniforme y secuencial la cantidad de procesos en los procesadores disponibles. Así, si tiene 7 procesos en 4 nodos:

Nodo 1	**Proceso 0**	**Proceso 4**
Nodo 2	**Proceso 1**	**Proceso 5**
Nodo 3	**Proceso 2**	**Proceso 6**
Nodo 4	**Proceso 3**	

Cuadro 1: Distribución de procesos por nodos de MPI (Standard)

Si necesitamos un agrupamiento de 3 procesos en un nodo, debemos generar:

Cantidad Procesos = 3x nro. Procesadores

Es decir, **todos** los procesadores tendrán 3 procesos, pero sólo uno de ellos es el que los utilizará en el agrupamiento. Por lo tanto, se generarán más procesos que los necesarios, pero como no tendrán tareas a procesar, finalizarán inmediatamente, sin generar sobrecarga de procesamiento en el sistema.

Si se requiere una réplica de procesos, habrá un proceso adicional que funcione como gestor de réplicas o "comunication manager", que se

agrupa con una de las réplicas, ya que se espera que tenga poca carga de trabajo.

Se tiene que calcular la cantidad de procesos necesarios de modo que se tengan suficientes para la configuración pedida por el usuario.

Existen instalaciones de MPI en clusters dónde un gestor de colas se encarga de asignar los procesos paralelos a cada máquina.

En estos casos, aunque la cantidad de procesos por nodo será tal como se prevé, no se puede saber cuál es el número de proceso que estará en ejecución en cada nodo.

Es decir, por ejemplo, en la mayoría de los casos, si tenemos 4 procesadores y ejecutamos 7 procesos, sabemos que el proceso 3 estará en ejecución en el nodo 3, y que el proceso 6 estará en nodo 2 (Ver Cuadro 1).

Pero si hay un gestor de colas, esto puede no cumplirse, aunque sí que habrá 3 procesadores con dos procesos y uno en el restante, puede que el proceso 6 este en un nodo que no sea el 2.

Para evitar problemas de este tipo, el generador de aplicaciones utilizará el nombre del nodo o procesador que alberga al proceso paralelo (host) para asignar los procesos agrupados que comparten el mismo procesador.

2.8 Solución Propuesta

El usuario del **Gaps** necesita utilizar la herramienta como parte de su proceso de investigación del modelo de rendimiento y sintonización de modelos compuestos de Master/Worker de pipelines.

Experimenta con esquemas de aplicaciones paralelas que ya conoce, a las que aplica modelos de rendimiento y sintonización para comprobar, constatar resultados, y estudiar el comportamiento de algunos parámetros en diferentes ejecuciones.

Por lo tanto, el usuario indica al sistema la cantidad de workers que lo componen, las etapas de cada uno con su duración, y cuáles deben ejecutarse agrupadas, replicadas o en forma individual en cada ejecución.

Para generar la aplicación sintética que resuelva el modelo, se precisa calcular cuantos son los procesos paralelos que se distribuirán a lo largo

de los procesadores disponibles para configurar el esquema de aplicación que necesita el investigador.

Este cálculo, por su naturaleza, no puede ser parte de un proceso paralelo.

Además, una aplicación paralela tiene que conocer de antemano cuantos procesos ejecutar.

Distinguimos entonces dos fases de ejecución para solucionar el problema, que se muestran gráficamente en la Figura 6.

Fase I:

La aplicación a generar se define con los siguientes parámetros:

- Cantidad de workers.
- Tamaños de mensajes.
- Cantidad de etapas.
- Etapas agrupadas.
- Etapas replicadas.
- Duración de cada etapa.
- Varianza del tiempo de cada etapa.

En esta fase, y con esta configuración de la aplicación paralela definida a alto nivel, se calcula el número de procesos necesarios y se realiza una especificación detallada de cada proceso, indicando cuáles se han de agrupar, replicar o ejecutar en forma individual o desaparecer.

Se establece el número de procesadores necesarios para cumplir con la distribución requerida.

El proceso produce la definición detallada que necesita el generador para hacer la aplicación paralela sintética que requiere el usuario en la siguiente fase.

Fase II:

Generación y ejecución de la aplicación paralela sintética con la especificación de procesos detallados en la Fase I.

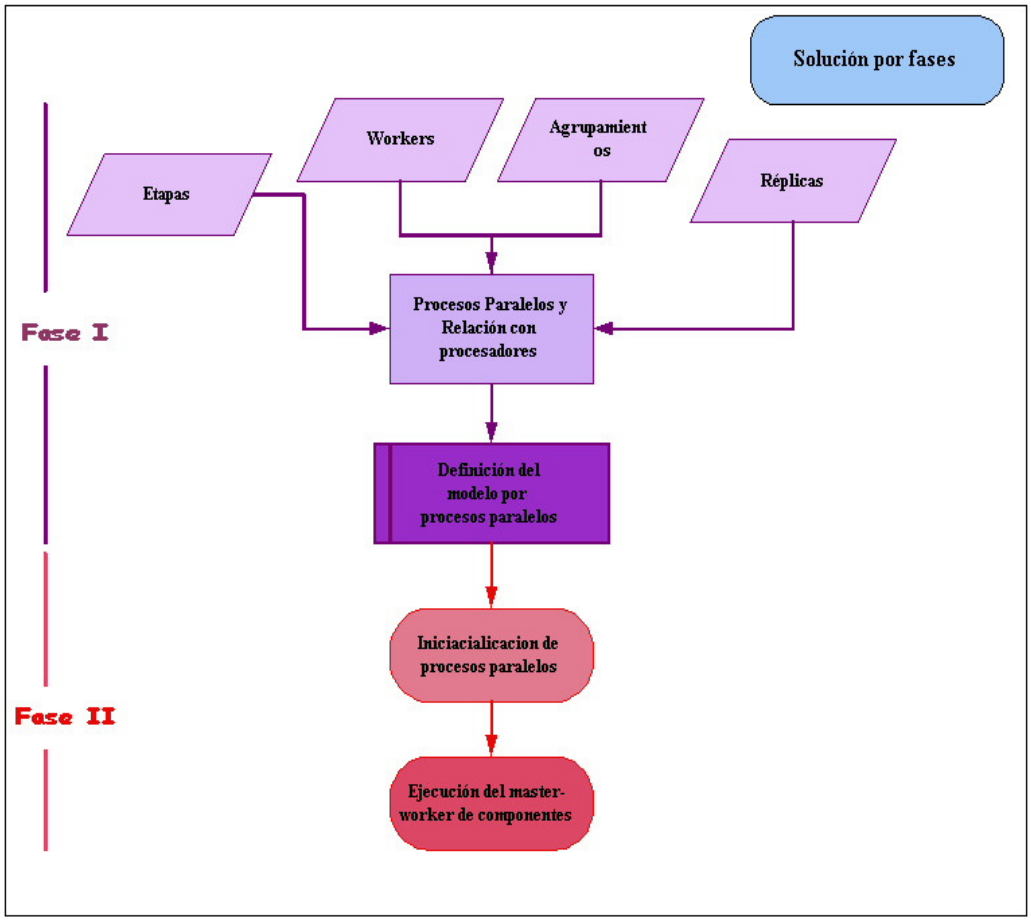

Figura 6. Fases de la solución propuesta

Esta solución por fases ofrece la ventaja al usuario de definir el modelo a alto nivel, con los parámetros que ya conoce, sin preocuparse por los detalles de implementación de la aplicación paralela.

3. Diseño

3.1 Resumen

En este capítulo, se detalla el diseño de las aplicaciones que resuelven las fases I y II de la solución propuesta en el análisis.

En el primer punto se explica como se consigue la meta principal de la primera fase, que es componer la relación de procesos que se ejecutarán en los procesadores disponibles para conseguir el esquema definido por el usuario. Se detalla el proceso de cálculo a partir del fichero de entrada en que el usuario define los parámetros a alto nivel de la aplicación a generar.

Luego se detalla el fichero de salida que es la base del vínculo entre ambas fases, ya que la salida de la primer parte es la entrada de la segunda.

En el punto siguiente, se explica el diseño del generador de la aplicación paralela sintética en sí. En base a los datos obtenidos en la fase anterior, se conoce cuántos procesadores se necesitan, cuantos procesos y que función específica tiene cada uno.

El diseño de Gaps indica la forma en que se asignan las funciones a cada proceso, y la ejecución de los mismos en jerarquía de Master/Worker de pipelines, con los casos especiales de replicación con la necesidad del comunication manager y los agrupamientos.

3.2 Relación de procesos y procesadores

En primer lugar se define el primer proceso que resuelve la Fase I descrita en el análisis. Tiene como objetivo calcular la cantidad de procesos paralelos y la función de cada uno de ellos para conseguir el modelo especificado por el usuario. A este proceso lo denominamos **GASPAR.**

Es un programa que recibe los datos de la aplicación a modelar en un fichero de texto con la definición de los parámetros.

A continuación se describe en el Cuadro 2, cada uno a uno los parámetros de entrada de **GASPAR**:

Parámetro	Nombre	Tipo	Observaciones
Tamaño de la tarea.	qtarea	Entero	*Tamaño del mensaje en bytes que se enviarán entre etapas cuando finalicen.* *El master enviará a la primera etapa un tamaño de tarea de qtarea*qcantmens.*
Cantidad de mensajes a enviar a los workers	qcantmens	Entero	*El master envía esta cantidad de tareas a la etapa 1 por cada iteración.*
Cantidad de iteraciones.	qiteraciones	Entero	*Iteraciones del proceso completo que realiza el master.*
Cantidad de workers	qworker	Entero	Cantidad de workers de la aplicación. Cada uno de estos es un pipeline de etapas.
Cantidad de etapas por defecto	Qetapasxdefecto	Entero	*Los workers se construirán por defecto con esta cantidad de etapas, pero puede especificarse otra cantidad para un worker determinado en otro parámetro.*
Tamaño del mensaje de respuesta	qmsgresp	Entero	*Las últimas etapas enviarán un mensaje de este tamaño al master al finalizar cada la tarea.*

Parámetro	Nombre	Tipo	Observaciones
Tiempo medio en segundos que dura cada etapa	ftmedioxdefecto	Punto flotante	*Todas las etapas duraran por defecto este tiempo. Puede especificarse otra cantidad para una etapa determinada en otro parámetro.*
Varianza de la duración media.	fvarianzaxdefecto	Punto flotante	*Todas las etapas tendrán indicadas la misma varianza. Puede especificarse otra cantidad para una etapa determinada en otro parámetro.*
Cantidad de etapas de un worker.	qetapas_<j>	Entero	*Cantidad de etapas del worker número <j>.*
Tipo de etapa	detapa_<i>_<j>	[AGR_ / REP_] + Entero	*Define si la etapa número <j> del worker <i> está agrupada o replicada. El número entero indica el grado de agrupamiento o replicación. Aquellas etapas de las que no se indica este parámetro, asume que son del tipo "individual", y que se ejecutan solas en un procesador.*
Tiempo medio de una etapa	ftiempomedio_<i>_<j>	Punto flotante	*Define el tiempo medio en segundos que dura esta etapa número <j> del worker <i>.*

Parámetro	Nombre	Tipo	Observaciones
Varianza del tiempo medio de una etapa	fvarianza_<i>_<j>	Punto flotante	*Define la varianza del tiempo medio en segundos que dura esta etapa número <j> del worker <i>.*

Cuadro 2: Parámetros de entrada de Gaspar

Con estos parámetros, **GASPAR** debe ser capaz de interpretar la estructura de la aplicación paralela que se pretende generar, y modelarla en un nuevo fichero de parámetros de salida.

GASPAR tiene que realizar las siguientes tareas para cumplir con su cometido:

- Determinar la cantidad de procesadores que necesita el sistema para ejecutarse, considerando uno por cada uno de los siguientes casos:
    - Master
    - Etapas individuales
    - Etapas agrupadas
    - Etapas replicadas
    - Gestor de replicas ("Comunication manager") agrupado con el primer proceso de replica.

- Generar la estructura de procesos Master/Worker de componentes pipeline

- Determinar el número de procesos mpi que necesita para procesar la aplicación sintética.

- Determinar las propiedades de cada etapa: individual, replicada, agrupada, tiempo medio y varianza, y de que proceso recibe y envía.

- Generar un fichero de salida con todos los datos procesados que sea interpretable por el generador de aplicaciones sintéticas (GAPS).

Para determinar la cantidad de procesadores, Gaspar procederá de la siguiente manera:

- La cantidad de procesadores del sistema es igual a los que necesita el modelo. Por ejemplo, si se tiene que ejecutar un master con 3 workers de 2 etapas cada uno, se necesitan de 7 procesadores, uno para cada proceso individual.
- Resta un procesador por cada agrupamiento. Es decir, siguiendo el ejemplo anterior, si dos etapas están agrupadas, se necesitan 6 procesadores.
- Añade un procesador adicional por cada réplica. Siguiendo el ejemplo anterior, si hay una etapa con tres (3) réplicas, se necesitaran 8 procesadores (2 adicionales)
- El comunication manager no ocupa un procesador adicional porque se agrupa con una réplica.

El fichero de salida que genera **GASPAR**, que es el la definición de entrada del generador **GAPS**, tiene dos partes bien diferenciadas:

1. Parámetros generales. Se detallan en el Cuadro 3.
2. Propiedades de cada una de las etapas. Se detallan en el Cuadro 4.

Los parámetros generales son:

Parámetro	Nombre	Tipo	Observaciones
Cantidad de procesos.	qprocesos	Entero	*Son los procesos de la aplicación sintética. (parámetro –np de mpirun)*
Cantidad de procesadores	qprocesadores	Entero	*Cantidad exacta de procesadores que necesita el generador para ejecutar el modelo. (parámetro –mpihost)*

Parámetro	Nombre	Tipo	Observaciones
Tamaño de la tarea.	Qtarea	Entero	*Tamaño del mensaje en bytes que enviará el master a los workers, y entre etapas.*
Cantidad de mensajes a enviar a los workers	Qcantmens	Entero	*El master envía esta cantidad de mensajes a la etapa 1 por cada iteración.*
Cantidad de iteraciones.	qiteraciones	Entero	*Iteraciones del proceso completo que realiza el master.*

Cuadro 3: Parámetros generales de salida de Gaspar y de entrada de Gaps.

Las propiedades que se definen para cada una de las etapas comienzan con una línea que indica **[p <i>_<j>],** dónde **<i>** es el número de worker y **<j>** el número de etapa.

Parámetro	Nombre	Tipo	Observaciones
Tiempo medio de una etapa	Ftiempomedio _<i>_<j>	Punto flotante	*Define el tiempo medio en segundos que dura esta etapa número <j> del worker <i>.*
Varianza del tiempo medio de una etapa	fvarianza_<i> _<j>	Punto flotante	*Define la varianza del tiempo medio en segundos que dura esta etapa número <j> del worker <i>.*
Tamaño del mensaje de respuesta	qmsgresp	Entero	*Las últimas etapas enviarán un mensaje de este tamaño al*

Parámetro	Nombre	Tipo	Observaciones
			master al finalizar toda la tarea. Se define en todas las etapas, pero el generador solo lo utilizará en caso de que se sea la última. Por defecto enviará mensajes del tamaño indicado en qtarea.
Tipo de etapa	etapa_XXX>_ <k>	XXX= [AGR / REP/ IND] k Entero	*Define si la etapa número es agrupada, replicada o individual. El número k es 1 si es individual o indica el grado de agrupamiento o replicación.*

Cuadro 4: Parámetros por etapa de salida de Gaspar y de entrada de Gaps

3.3 Generador de aplicaciones sintéticas.

El generador de aplicaciones **Gaps** es una aplicación paralela con intercambio de mensajes a través de MPI.

Como se ha dicho, la aplicación es de estructura Master/Worker de componentes pipeline y la estructura y el comportamiento en ejecución estará determinado por el fichero de parámetros que generó Gaspar.

Al inicio, el master, en base a estos datos de entrada, realiza las siguientes acciones para preparar el sistema a ejecutar:

- Validará que los procesos paralelos sean iguales a los que necesita (parámetro ***qprocesos*** del fichero de entrada).

- Validará que el generador tenga disponibles ***exactamente*** la cantidad de procesadores necesarios **(qprocesadores),** para asegurar que los procesos se agruparán en los nodos en la forma deseada. Es decir, si se calcula que se necesitan 14 procesos en 7 procesadores, es porque se busca un agrupamiento de dos

procesos en algunos de los procesadores. Si en el momento de ejecutar la aplicación paralela, hay 10 procesadores, mpi generará los 14 procesos y quedarán sólo 4 con dos procesadores y esto no es lo que se pretendía. Entonces, hay que asegurar que haya 7, ni más ni menos.

- Asignará a cada proceso paralelo el número de etapa que ejecuta para identificarla.
- Se asegurará que las etapas agrupadas lo hagan en el mismo procesador comparando el nombre del procesador de cada una de estas.
- Agrupará al comunication manager con la primero de los réplicas que ejecuta.
- Determinará la función de cada proceso. Estos pueden ser:
    - Primera etapa del pipeline.
    - Etapa normal.
    - Ultima etapa del pipeline.
    - Gestor de réplicas (comunication manager).
    - Desocupado.
- Configurará cada proceso con las propiedades básicas para su funcionamiento: tiempo medio, varianza, de que proceso recibe, a quien envía los datos y el tamaño de la tarea o mensaje a transmitir entre procesos.
- Indicará los procesos que maneja el gestor de réplicas.
- Enviará un mensaje de inicio a cada proceso con su función y propiedades.

Los procesos se inicializaran con el mensaje enviado por el master y, luego de una sincronización, comenzaran a ejecutar sus tareas según la función que cumplen.

Los procesos que se generaron por exceso para los casos de replicación o agrupaciones finalizarán una vez reciban el mensaje del master que les indica que no tienen tareas a realizar.

Una vez que cada proceso conoce su función, comienza la ejecución de la aplicación sintética. Los siguientes puntos describen el funcionamiento esperado:

- El master realiza una cantidad de ciclos iguales determinado por el parámetro "***qiteraciones***".
- En cada una de las iteraciones, envía un mensaje con a las primeras etapas de los workers del tamaño de la tarea por la cantidad de mensajes iniciales (***qtarea*qcantmens***).
- Las primeras procesan dicha cantidad de tareas (***qcantmens***), y al finalizar cada una reenvían a la siguiente etapa.
- Las primeras procesan dicha cantidad de tareas (***qcantmens***), y al finalizar cada una reenvían a la siguiente etapa un mensaje de tamaño ***qtarea***.
- El pipeline se activa al recibir un flujo secuencial de una cantidad de tareas (qcantmens), logrando un solapamiento de las tareas una vez que todas las etapas reciben su tarea a procesar. En principio, el tiempo medio que tarda en realizarse esta cantidad de tareas estará determinado por la duración de la etapa más lenta.
- Las últimas etapas de los workers envían un mensaje al master, a modo de señal de fin de tarea. El tamaño es ***qmsgresp***.
- Si una etapa está replicada, la etapa anterior envía los mensajes al proceso comunication manager.
- El comunication manager recibe cada tarea de la etapa anterior y se la envía a una de las réplicas que este desocupada.
- Las replicas procesan sus tareas y al finalizar envían un mensaje al CM de tamaño **qtarea** que a su vez CM envía a la etapa siguiente.
- Las etapas agrupadas reciben y envían mensajes según su función (primera etapa, etapa intermedia o última etapa) pero residen todas en el mismo procesador.
- Cada etapa, al procesar cada tarea de tamaño ***qtarea*** que recibe, demora un tiempo medio indicado en **f*tiempomedio*** de la etapa.
- Al tiempo medio se le aplica la varianza indicada en la etapa ***fvarianza.***

La Figura 7 muestra un diagrama del diseño de Gaps, mostrando el tamaño de mensaje que se envían entre los distintos procesos según su función:

- Primera etapa del pipeline.
- Etapa normal.
- Ultima etapa del pipeline.
- Gestor de réplicas (comunication manager).

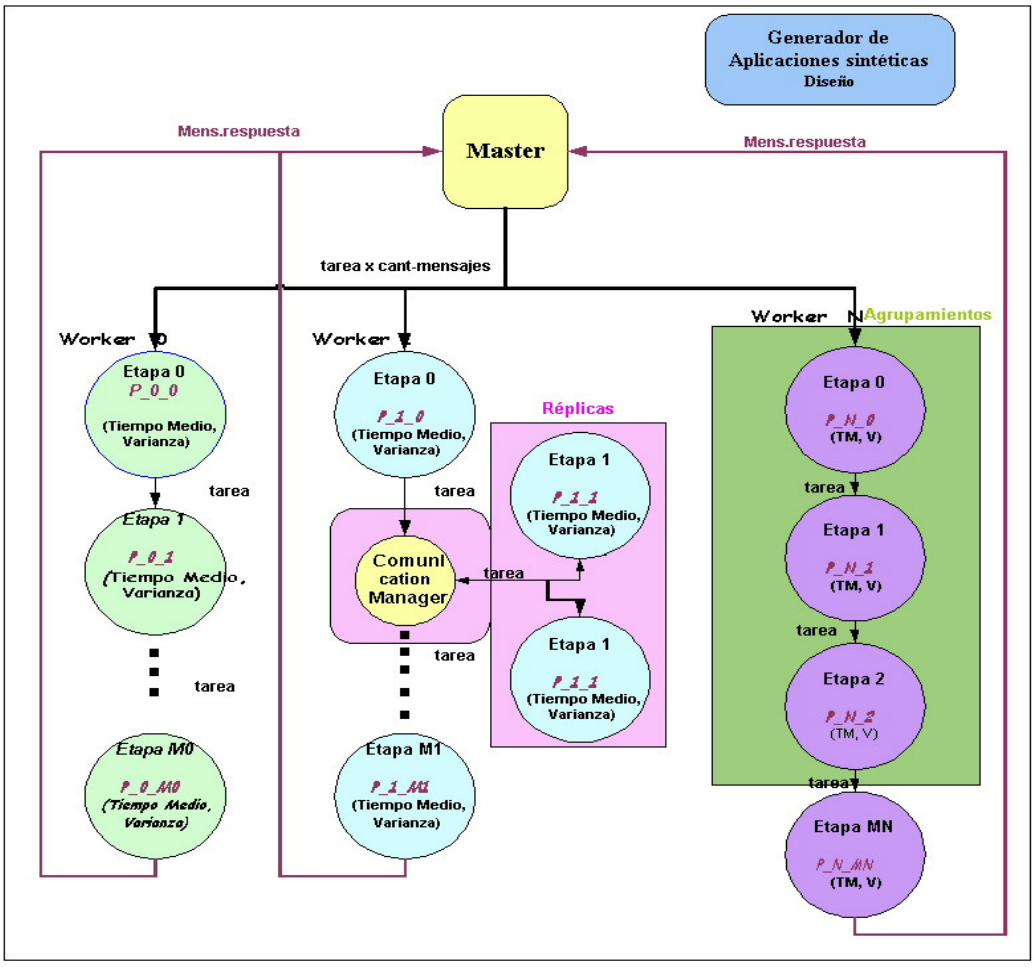

Figura 7. Diseño del generador de aplicaciones sintéticas

4. Implementación

4.1 Resumen

En este capítulo, se explica como se implementan las dos aplicaciones que resuelven las fases I y II de la solución propuesta en el análisis, y que durante el diseño denominamos Gaspar y Gaps respectivamente.

Aunque estos procesos se comunican a través del fichero de parámetros que produce Gaspar y que toma como entrada Gaps, en la implementación comparten una misma estructura de datos denominada "`master_worker`", que mantiene la relación de procesos organizado en la forma jerárquica que plantea la configuración de alto nivel de la aplicación paralela que el usuario quiere generar.

Gaspar es el encargado de generar la información de esta estructura y la vuelca en el fichero de parámetros, y Gaps, la vuelve a dejar en memoria a partir de la lectura del fichero de entrada.

Gaps es una aplicación paralela e inicializará cada proceso con la función y propiedades establecidas en el fichero de parámetros, asegurando que se cumple la distribución por procesadores en los casos de agrupamientos y réplicas.

En el primer apartado se detallan las tareas que realiza Gaspar para cumplir su cometido, y en la segunda parte, como Gaps genera y ejecuta la aplicación sintética, utilizando los cálculos y datos de Gaspar.

4.2 Generador de parámetros: Gaspar

El proceso que genera los parámetros de Gaps, Gaspar, es un programa en C, de estructura y funcionamiento simple.

En base a la configuración de la aplicación indicada en el fichero de entrada, y algunos procesos de cálculo, Gaspar completa una instancia de una estructura jerárquica `master_worker`.

Dicha estructura esta formada por dos subestructuras asociadas de "`worker`" y "`etapa`", y tienen todos los datos que se utilizan para generar la salida del proceso, que, a su vez, constituye la entrada del generador.

El siguiente Cuadro 5 detalla las estructuras de datos mencionadas.

```
typedef struct master_worker
{
          int qprocesadores;
          int qprocesos;
          int qtarea;
          int qiteraciones;
          int qcantmens;
          int qworkers;
          worker workers[MAXWORKERS];
          int modo;
} master_worker;


typedef struct worker
{
      int idworker;
      int qetapas;
      int qmsgresp;
      float ftiempomedio;
      float fvarianza;
      etapa etapas[MAXETAPAS];
} worker;


typedef struct etapa
{
     int idetapa;
     float ftiempomedio;
     float fvarianza;
     char tipo;
     int qetapastipo;
} etapa;
```

Cuadro 5: Estructura Master_Worker / Worker y Etapa

Cabe aclarar que en la implementación, la cantidad máxima de workers que permite el master esta limitada por la directiva `MAXWORKERS` que se define en tiempo de compilación.

Para las pruebas se definió en 24.

Asimismo, la cantidad máxima de etapas de cada worker esta limitada por la directiva `MAXETAPAS` que se define en tiempo de compilación.

Para las pruebas se definió en 64.

El esquema del proceso general de **Gaspar** se muestra en la Figura 8:

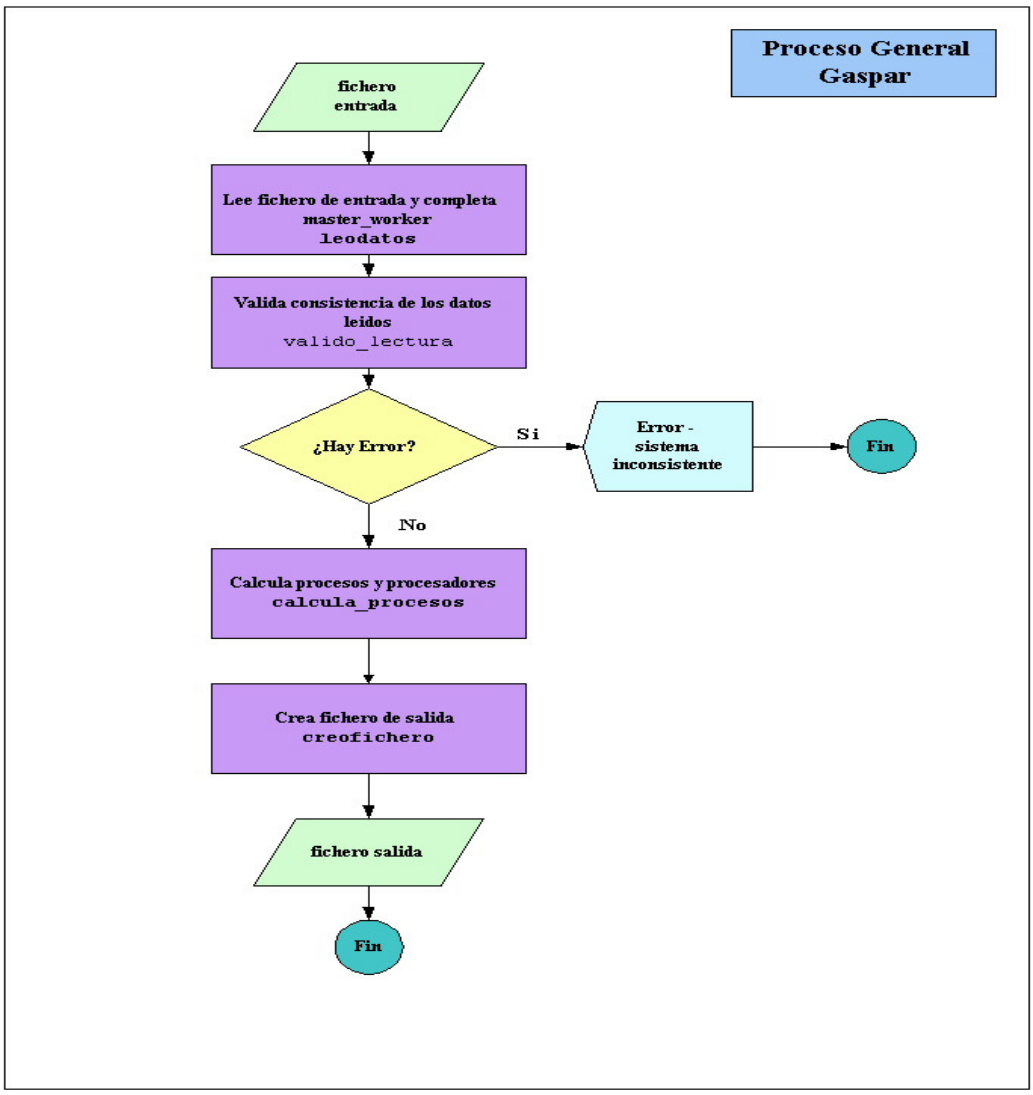

Figura 8. Proceso general del aplicativo GASPAR

En la validación de consistencia de los datos leídos, Gaspar realiza una serie de comprobaciones que se detallan en el Cuadro 6, junto al número de error de referencia que indica en caso de no cumplirse la condición deseada.

Validación	Número de Error
Cantidad de workers mayor que cero.	1
Tamaño de tarea mayor que cero.	2
Cantidad de mensajes mayor que cero.	3
Cantidad de etapas de todos los workers mayor que cero.	4
Tamaño del mensaje de respuesta de todos los workers mayor que cero	5
Tiempo medio de duración de todas las etapas mayor que cero.	6
Varianza del tiempo medio de duración de todas las etapas mayor que cero.	7
Todas las etapas son del tipo I (individual), A(Agrupada) o R(Replicada)	8
El grado de agrupamiento de una etapa tipo A es mayor que cero.	9
El grado de agrupamiento de una etapa tipo A es consistente con el número de etapas del worker.	10

Cuadro 6: Relación de validaciones y números de errores de Gaspar

En caso de no cumplir una de las condiciones mencionadas, **Gaspar** no genera el fichero de salida y, por pantalla muestra un mensaje del tipo:

"***Error=X - Inconsistencia en el sistema master_worker leído***"

Sin duda, una de las funciones principales de **Gaspar**, es calcular el número de procesos y el número de procesadores que requiere el generador de la aplicación sintética para ejecutar el modelo en forma adecuada.

El número de procesadores está determinado por:

- Un procesador para el proceso master.
- Se añade uno por cada etapa individual.
- Se añade uno por cada etapa replicada.
- Se resta uno por cada etapa agrupada.

Una vez que se calculó exactamente el número de procesadores que se necesita para ejecutar el modelo tal como está definido, se calcula el número de procesos de la siguiente forma:

- Se determina el máximo nivel de agrupamiento de procesos en un mismo procesador que necesita el sistema. Para este cálculo se considera el grado de agrupamiento de las etapas y en caso de réplicas, un agrupamiento de nivel 2 porque el gestor de réplicas comparte procesador con la primera de las etapas.

- El número de procesos es igual al máximo entre el número de procesadores por el máximo nivel de agrupamiento y el número total de procesos necesarios incluidas las réplicas.

Cant. Procesos = Máximo (Procesadores*Max.Nivel Agrupamiento**;** Procesos-Agrupaciones+Réplicas)

4.3 Generador de aplicaciones paralelas: Gaps

4.3.1 Estructuras de datos

El generador de aplicaciones sintéticas es el programa paralelo que ejecutará el sistema configurado en base a los parámetros indicados por el usuario y a los calculados por Gaspar, agrupados en un fichero de entrada.

La cantidad de procesos está indicada en ese fichero bajo el parámetro **qprocesos**. El usuario deberá procurar que los procesadores disponibles para mpi en el momento de ejecución coincidan exactamente con lo establecido en la variable **qprocesadores** para asegurar que los procesos se ejecutan en forma independiente o compartiendo procesador como lo requiere el modelo a generar. **Gaps** dará un error en caso de no cumplirse esta condición.

Al comienzo el proceso principal o master de **Gaps** interpreta el fichero de entrada, y completa una estructura de datos `master_worker` igual que la utilizada por **Gaspar**. Con ello dispone de los datos de todos los workers con sus etapas y atributos.

Completa un vector de datos denominada **procesos** con un elemento por cada uno de los procesos mpi que están en ejecución en el sistema, incluyendo el nombre del host donde se está ejecutando.

Los atributos de cada elemento de esta estructura se detallan en el Cuadro 7:

Atributo	Nombre	Tipo	Observaciones
Nro. De Worker	idworker	Entero	*Identificador del worker. Será un número entre 0 y el parámetro qworkers-1.*
Nro. de etapa	idetapa	Entero	*Identificador de la etapa dentro del worker. Será un número entre 0 y el parámetro qetapasxdefecto o qetapas_<j>.*
Nombre del procesador o nodo.	nomnodo	Entero	*Nombre del procesador o nodo que ejecuta el proceso. El master pregunta este dato a cada proceso al inicio, antes de distribuir los procesos del modelo.*
Tiempo medio de ejecución	Ftiempomedio	Float	*Tiempo medio que tarda esta etapa en realizar una tarea.*

Atributo	Nombre	Tipo	Observaciones
Varianza del tiempo medio de ejecución	fvarianza	Float	*Varianza del tiempo medio que tarda esta etapa en realizar una tarea.*
Tamaño de la tarea.	qtarea	Entero	*Tamaño en bytes de la tarea que tendrá que procesar.*
Cantidad de mensajes	qcantmens	Entero	*Cantidad de mensajes que recibirá para procesar.*
Tamaño del mensaje de respuesta	qmsgresp	Entero	*Tamaño del mensaje que enviará la última etapa cuando se finalice una tarea.*
Nro. de proceso de quien recibe.	recibe_de	Entero	*Identificador del proceso de quien recibe mensajes.*
Nro. de proceso a quien envía	envia_a	Entero	*Identificador del proceso de quien recibe mensajes*
Identificador de tipo de proceso a ejecutar	Tipoproceso	Entero	*Puede ser* *0=normal* *1=ultima etapa* *2=com.manager*
Cantidad de procesos que gestiona el com. Manager.	comman_qproc	Entero	*Para procesos del tipo com. Manager, se indica la cantidad de réplicas que gestiona.*
Vector de identificadores de procesos gestionados por el com.manager	comman_id[MAX_COMPROCESS];	Entero	*La cantidad MAX_COMPROCESS de definió en 16 para esta compilación.*

Cuadro 7: Estructura de datos “Procesos”

El master completa la información y envía a un mensaje a cada proceso con esta información y así se inicializan.

Una vez que recibe este mensaje, el proceso sabe que tareas y como ejecutarlas para comportarse tal como se requiere: primer etapa, intermedia, última, comunication manager o réplica, tal como requiere la definición del usuario.

Los procesos que han de desaparecer porque no tienen tareas a realizar, reciben una identificación de worker igual a -1. Con dicha señal, el proceso finaliza, liberando los recursos asignados.

4.3.2 Proceso General

El diagrama general del proceso paralelo ***Gaps*** se muestra en la Figura 9.

El proceso master completa los datos de los procesos y realiza el mapeo con los procesadores para asegurarse que comparten procesador según la posición en la configuración que se está evaluando:

- Etapa individual en un único procesador
- Las agrupadas lo comparten
- Las réplicas en procesadores diferentes.
- El comunication manager que se agrupa con la primera de las réplicas.

Una vez dispone de la información, el master envía un mensaje a cada proceso con la estructura de datos “Proceso” que se detalló en el apartado anterior.

Con esa información, cada proceso paralelo sabe si tiene que ejecutar el procedimiento de etapa normal, de comunication manager o bien, finalizar porque no tiene ninguna tarea a realizar.

Durante el mapeo de procesos, ***Gaps*** une a las etapas agrupadas y al comunication manager con su primera réplica, asignándoles procesos con igual nombre de host.

Esto asegura que se cumpla con las agrupaciones en procesadores tal como está definido por el usuario, aún en aquellos casos en que la

implementación de MPI no siga una distribución de procesos en forma uniforme en los procesadores disponibles.

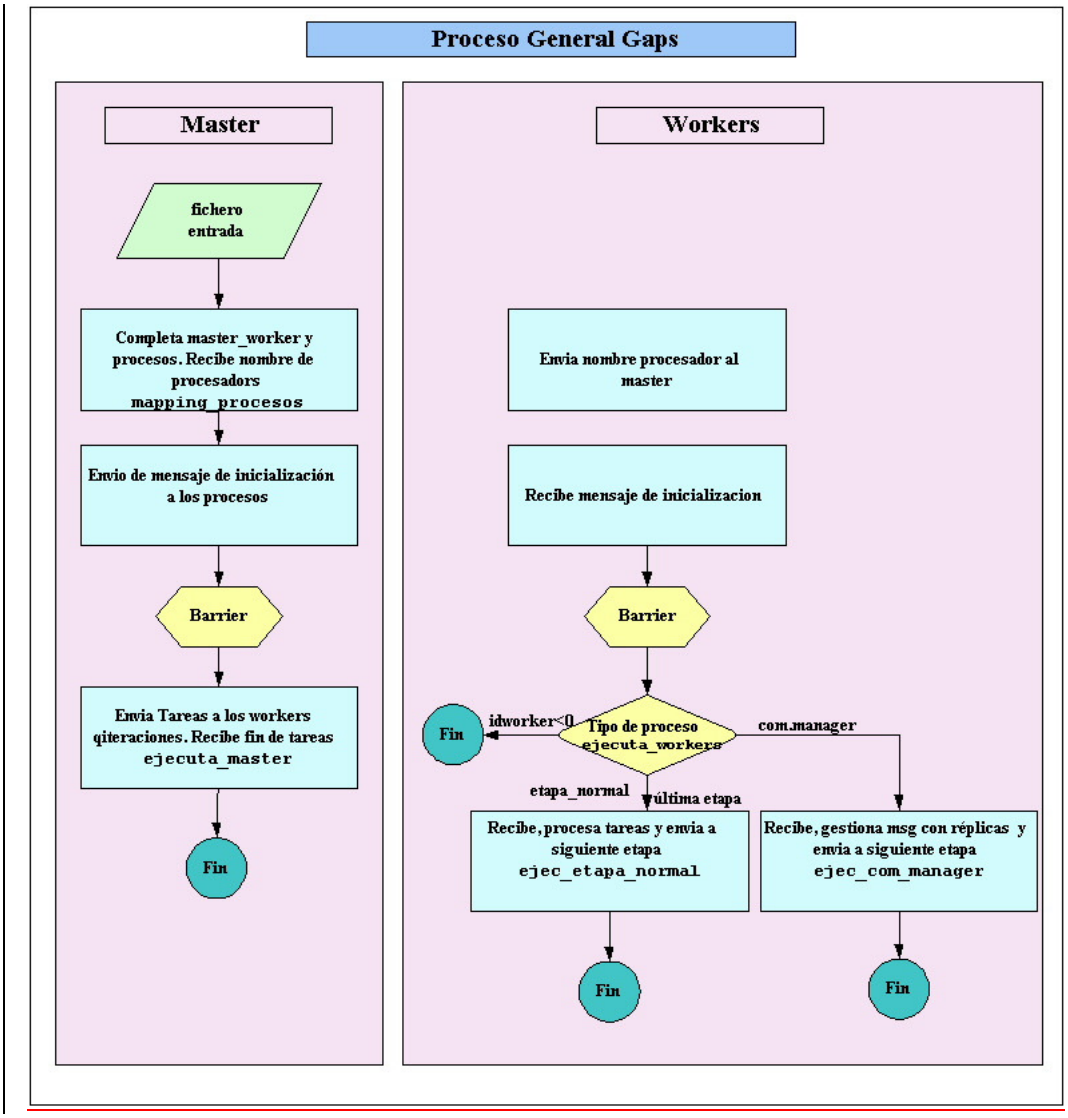

Figura 9. Proceso general del aplicativo GAPS

Una etapa normal es aquella que recibe una tarea de un proceso previo, la ejecuta en un tiempo medio con una varianza definida y al finalizar, envía la tarea a realizar a la siguiente etapa.

En este tipo de etapas se distingue a la primera etapa porque esta puede recibir una cantidad de tareas iniciales distinta de uno, ya que este es el comportamiento típico de una estructura pipeline.

A su vez, la última etapa de esta cadena, también tiene un comportamiento particular, ya que cuando finaliza, debe avisar al master, enviando un mensaje de un tamaño diferente que la tarea.

El procedimiento que ejecutan todas las etapas para recibir y enviar a la siguiente es el mismo, y se denomina **ejec_etapa_normal**. La diferencia del comportamiento que mencionamos entre primera, última y del medio, la consigue el master al introducir los datos en la estructura proceso que envía a cada uno.

Por ejemplo, en el caso de procesos que sean primera etapa del worker, el master completa el atributo de cantidad de mensajes (qcantmens), de modo que el proceso prevé recibir el tamaño de la tarea por este factor.

Y para las últimas etapas, el master completa el atributo de tamaño del mensaje de respuesta (qmsgresp) con el tamaño de mensaje que tiene que enviar.

La siguiente Figura 10 ilustra como se comporta una etapa normal, indicando el tamaño de mensajes que envía y recibe.

El master envia la cantidad inicial de qcantmens a las primeras etapas con el propósito de activar el funcionamiento de pipeline, que consigue la simultaneidad de tareas cuando todas las etapas estan procesando.

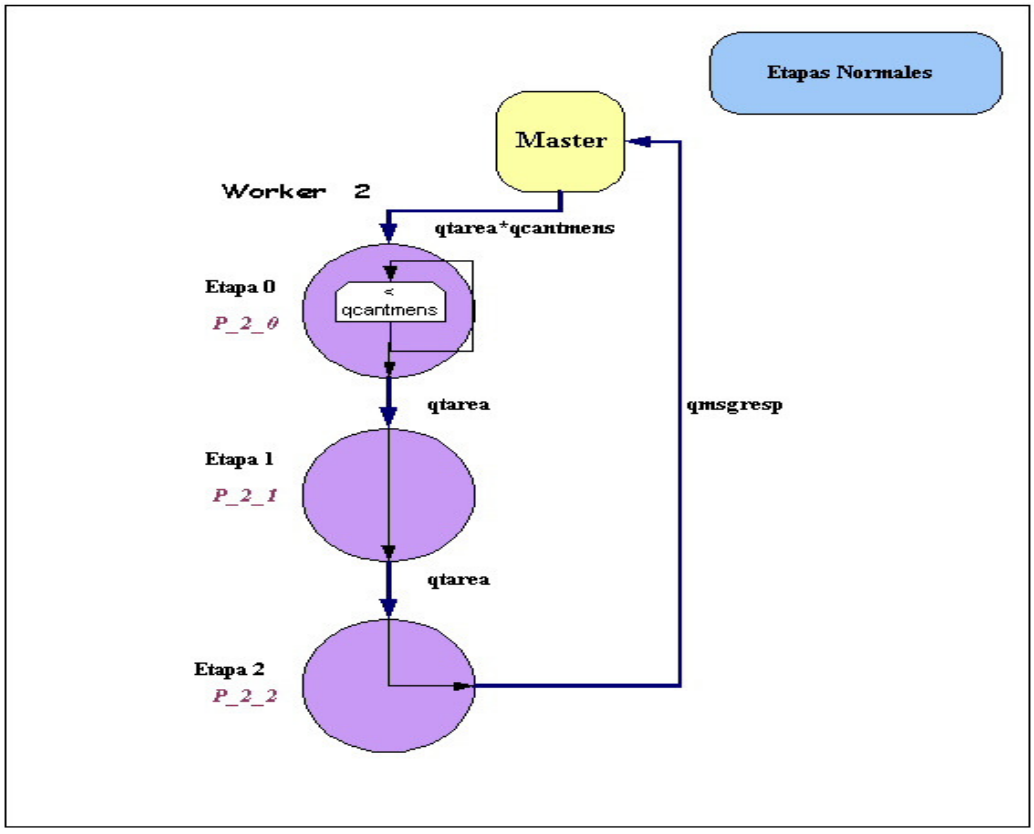

Figura 10. Comportamiento de una etapa normal

Las etapas intermedias reciben una tarea por vez, de tamaño qtarea. Una vez que la procesan en un tiempo medio establecido, envían a la etapa siguiente un mensaje del mismo tamaño.

Sólo cuando es última tarea, envia un mensaje de tamaño diferente, qmsgresp, y en este caso, al master.

En cambio, el procedimiento del comunication manager (**ejec_com_manager**) es muy diferente. Este proceso controla a un conjunto de etapas, que a su vez se comportan como normales, con la diferencia que reciben y envían tareas de/a este gestor en vez de a una etapa del pipeline.

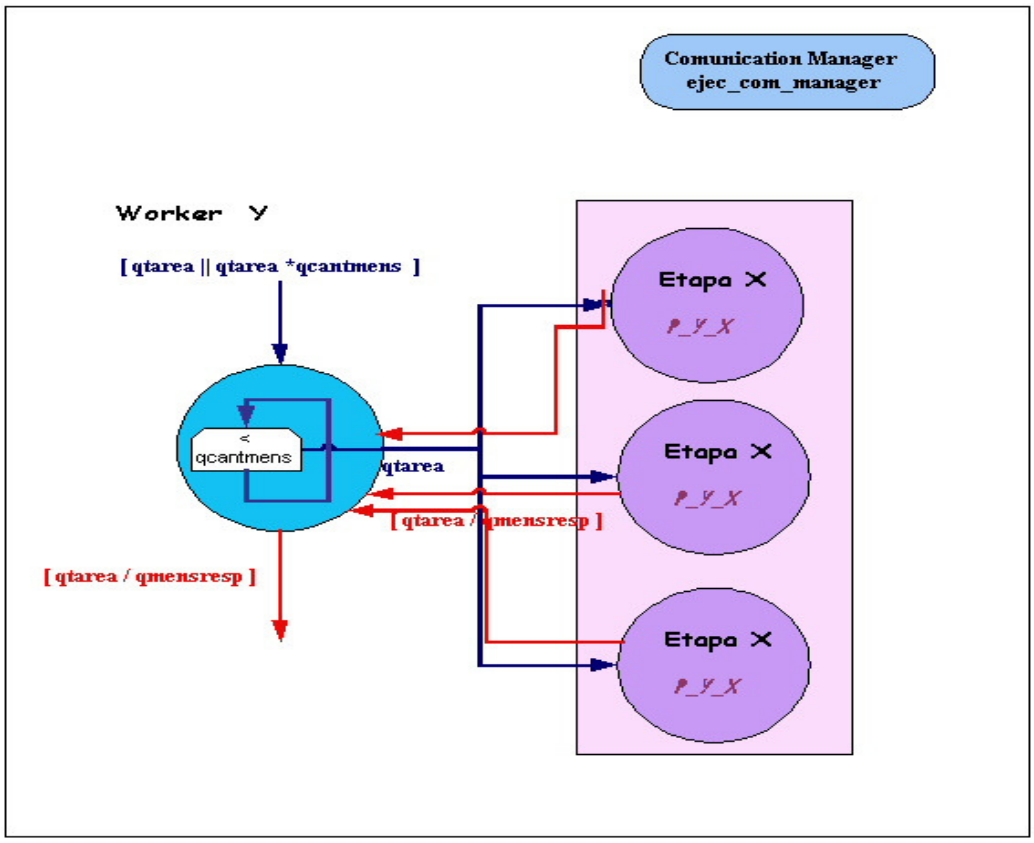

Figura 11. Comportamiento del CM

El proceso del comunication manager recibe una tarea de tamaño **qtarea** de la etapa anterior o varias del master (cantidad indicada en ***qcantmens***), en caso de ser primera etapa. Inmediatamente busca una réplica desocupada y le envía la tarea a ejecutar. Cuando la réplica termina, le envía la tarea siguiente al gestor, que a su vez, re-enviará a la etapa siguiente. Si son réplicas de la última etapa, el mensaje será para enviar al master (***qmensresp***).

El comunication manager recibe mensajes desde dos vías simultáneamente, desde la etapa anterior o master, y desde las réplicas y envía hacia dos destinos diferentes, la etapa siguiente o master si es una réplica de la última etapa, y hacia las réplicas.

En la implementación se ha utilizado el mecanismo de MPI_Probe que permite detectar si se recibió algo, para evitar bloqueos por recibir de dos canales diferentes. El gestor sólo recibirá nuevos mensajes a procesar si tiene réplicas libres, porque tiene que emular el funcionamiento del sistema original.

La figura 11 muestra como se comporta el comunication manager, indicando el tamaño de mensajes que envía y recibe y distinguiendo los dos canales de comunicaciones de entrada y salida en colores rojo y azul.

4.3.3 Pseudo código de procesos principales

Master

Este es el pseudo-código que ejecuta el master luego de haber enviado a cada proceso el mensaje inicial a cada proceso worker.

Master

```
Mientras qiteraciones hacer
   Para J ←1 hasta workers
         ENVIAR qtarea*qcantmens A Proceso PJO
   Fin para J.
   Para J ←1 hasta workers
         RECIBIR qmsgresp de Proceso PJk //k es la última etapa del
worker j
   Fin para J.
Fin mientras.
Para J ←1 hasta workers
   Enviar "Señal-fin" a Proceso PJO
Fin para J.
```

Primera etapa del pipeline

Este es el pseudo-código del proceso que se ejecuta cuando se recibe un mensaje de inicio desde el master que indica que se recibe del master (id=0).

Primera etapa

Mientras no "señal-fin" hacer
RECIBIR ***qtarea*qcantmens*** *de Proceso master (****recibe_de****=0)*
Para I ←1 hasta ***cantmens***
<Proceso_tarea durante tiempo (ftiempomedio,fvarianza)
ENVIO ***qtarea*** *a proceso* ***envia_a***
Fin para I
Fin Mientras
Enviar-señal-fin a proceso ***envia_a***

Última etapa del pipe

Este es el pseudo-código del proceso que se ejecuta cuando se recibe un mensaje de inicio desde el master que indica que se envía al master (id=0) y el tipo de proceso es igual a última etapa. (Tipoproceso=1).

Última etapa

Mientras no "señal-fin" hacer
RECIBIR ***qtarea*** *de proceso* ***recibe_de***
SI NO "senal-fin"
<Proceso_tarea durante tiempo (ftiempomedio,fvarianza)
ENVIO ***qmsgresp*** *a proceso* ***envia_a***
Fin para I
Fin Mientras

Comunication manager

Este es el pseudo-código del proceso que se ejecuta cuando se recibe un mensaje de inicio desde el master que indica que el tipo de proceso es igual a última comunication manager (Tipoproceso=2).

Cabe puntualizar en este las siguientes consideraciones para comprender mejor el siguiente pseudocódigo, aunque varias se han mencionado antes:

- Este proceso recibe dos datos adicionales en la estructura de proceso que envía el master al inicio.

- Cantidad de procesos que gestiona el com. Manager: **comman_qproc.**

- Vector de identificadores de procesos gestionados por com.manager **comman_id[**MAX_COMPROCESS].

- Cuando se reciba del proceso indicado en **recibe_de** del mensaje de inicialización, deberá re-enviarse a una de las replicas que este libre.

- Cuando recibe un mensaje de uno de los procesos que gestiona, significa que ha terminado y lo deja como libre, y re-envía dicho mensaje a **envia_a.**

- Necesitará mantener un vector de estado por cada proceso de réplica para saber si esta LIBRE o OCUPADO

Comunication Manager

Mientras no "señal-fin"
Libre <- comman_id[id] // **Selecciono un proceso libre**
<probe de línea de comunicaciones> // **Pruebo si hay mensaje pendiente libre**
Si hay mensaje de ***<recibir_de>*** *y tengo proceso libre*
RECIBIR ***qtarea*qcantmens*** *de proceso* ***recibe_de***
ENVIO ***qtarea*qcantmens*** *a proceso libre*
Marco OCUPADO a comman_estado[<libre>]
Libre <- siguiente_libre(comman_estado); // **Selec. otro proceso libre**
Fin si
Si hay mensaje de un proceso replica [comman_id]
RECIBIR ***[qtarea | qmsgresp]*** *de [comman_id]*
ENVIO ***[qtarea | qmsgresp]*** *a* ***enviar_a***
Marco LIBRE a comman_estado[<libre>]
Fin si.
Fin mientras
Mientras hay algún comman-id ocupado ***//antes de fin. espero terminen las tareas***
<probe de linea de comunicaciones>
Si hay algo de algún comman_id
RECIBIR ***[qtarea | qmsgresp]*** *de comman_id*
ENVIO ***[qtarea | qmsgresp]*** *a Proceso enviar_a*
Marco LIBRE a comman_estado[<libre>]
Fin mientras.
Para J <- 1 hasta comman_qproc
Enviar-señal-fin a comman_id[j]
Fin Para
Si ***enviar_a != 0*** *Enviar-señal-fin a* ***enviar_a***

Réplicas

Este es el pseudo-código del proceso que se ejecuta cuando se recibe un mensaje de inicio desde el master que indica que en los atributos **envia_a** y **recibe_de** al mismo número de proceso que a su vez corresponde a uno del tipo comunication manager.

Las réplicas se comportan como una etapa normal, pudiendo ser la primera, intermedia o última. El master indicará en los atributos **envia_a** y **recibe_de** al número del proceso comunication manager para que sea el quién gestione las comunicaciones desde y hacia estos procesos.

El atributo de cantidad de mensajes qcantmens, sólo será distinto de uno cuando corresponda a la primera etapa de un worker.

Replicas

Mientras no "señal-fin" hacer
RECIBIR ***qtarea*qcantmens*** *de proceso* ***recibe_de***
Si no "señal-fin"
<Proceso_tarea tiempo (ftiempomedio,fvarianza)
ENVIO ***[qtarea | qmsgresp]*** *a* ***enviar_a***
Fin mientras
Enviar-señal-fin a proceso ***envia_a***

4.4 Salida

Gaps tiene tres tipos de salida diferentes, que se selecciona configurando la variable de precompilación MODODEBUG con los valores 0,1, y 2.

En todos los casos se mostrarán los casos de error y las estadísticas finales de cada proceso paralelo: el tiempo total, tiempo medio, porcentaje de desvío estándar, cantidad de mensajes recibidos y enviados

El funcionamiento para cada caso se explica en el siguiente Cuadro 8:

MODODEBUG	Salida
0	Nivel bajo. La salida de los procesos es por pantalla (stdout), y solo se imprimen mensajes de inicio de procesos y casos de error.
1	Nivel alto por pantalla. La salida de los procesos es por pantalla (stdout), incluyendo un trazado completo de la actividad que realiza.
2	Nivel alto por fichero. La salida de los procesos es por fichero *gaps[nro-proceso].log*. Incluye un trazado completo de la actividad que realiza.

Cuadro 8: Tipos de mensajes de salida de Gaps

5. Pruebas

5.1 Resumen

En este capítulo, se explican los casos de prueba que se seleccionaron para comprobar que las aplicaciones Gaspar y Gaps realizan la tarea que se espera, resolviendo el esquema definido por el usuario.

Las pruebas se realizaron en el cluster de desarrollo del laboratorio del departamento de Caos que dispone de un total de 8 nodos.

Se distinguen dos tipos de pruebas:

1. **Funcionales:** Pretenden probar que Gaps genera y ejecuta aplicaciones sintéticas compuestas Master/Worker de pipelines de diferentes tipos:
    - Con etapas individuales: Se prueba 1 caso.
    - Con etapas agrupadas: Se prueban 2 casos.
    - Con etapas replicadas: Se prueban 2 casos.
    - Con etapas agrupadas y replicadas: Se prueban 3 casos.

2. **Ejecución:** Pretenden probar que Gaps permite al investigador desarrollar modelos de rendimiento para aplicaciones compuestas Master / Worker de pipelines. Se comprueban las posibilidades de la herramienta para agrupar etapas rápidas y replicar las lentas, y obtener los resultados para el análisis de comportamiento. Se prueban dos aplicaciones, con dos ejecuciones para cada una.

Se clasifican las pruebas de esta forma para comprobar en forma separada la funcionalidad de la aplicación y el grado de utilidad para la que fue concebida.

Para las pruebas funcionales, se seleccionaron ocho casos de prueba que cubren todas las posibilidades expuestas. Se detalla y se grafica el resultado de la ejecución de cada caso, mostrando los procesos que se ejecutan en cada nodos, la función que cumplen y los tiempos de cada uno. Sólo en el primer caso se incluye un detalle del fichero de entrada de Gaspar y el fichero de salida que constituye la entrada de Gaps a modo de ejemplo y se detallan los comandos para ejecutar cada prueba. En todos los diagramas los procesos que se ejecutan en un mismo nodo

tienen el mismo color y se identifican con P<nro.proceso>. Si se indica una X, significa que el proceso finaliza en cuanto recibe el mensaje inicial del master. (Función Desocupado).

Para las pruebas de ejecución, se seleccionaron dos aplicaciones. Para cada una se muestra el resultado de la ejecución con Gaps en su configuración original y en la sintonizada, esto es, luego de aplicar las fórmulas que corresponden a un modelo de rendimiento de pipeline, que indican cuando agrupar y replicar etapas.

Los ficheros de las ejecuciones se anexarán en formato digital con el presente trabajo. En el título de los diagramas se indica el nombre del fichero de parámetros, para una rápida localización.

5.2 Pruebas Funcionales

Caso 1: Etapas Individuales

El modelo que se quiere generar es un caso básico que no tenga réplicas ni agrupamiento, por lo cual todas son etapas individuales, en el sentido que se ejecutan cada una en un único procesador.

La Figura 12 muestra el diagrama de la ejecución.

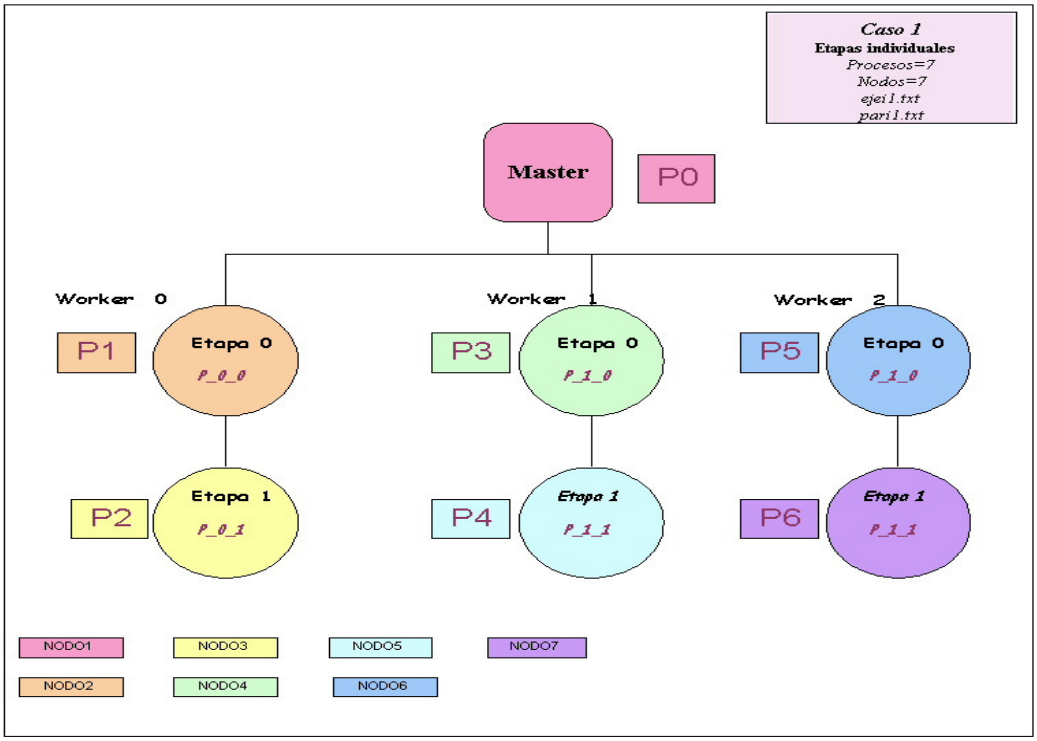

Figura 12. Caso de prueba 1. Etapas individuales.

El fichero de entrada que describe el caso 1, se llama "ejei1.txt" y es:

```
#qtarea: tamaño de la tarea en bytes
qtarea=1000
#qcantmens: cant. mensajes a enviar a los workers.
qcantmens=5
#qiteraciones: cant. iteraciones a realizar
qiteraciones=10
#qworkers: cant. de workers
qworkers=3
#qetapas: cant.de etapas de cada worker por defecto.
qetapasxdefecto=2
#qmsgresp: tamaño del mensdefaje de respuesta
qmsgresp=10
#ftiempomedio de cada etapa
ftmedioxdefecto=3.5
fvarianzaxdefecto=2.35
```

ejei1.txt

Se ejecuta Gaspar para obtener el fichero "pari1.txt".

```
./Gaspar ejei1.txt pari1.txt
```

```
qprocesos=7
qprocesadores=7
qtarea=1000
qcantmens=5
qiteraciones=10
[p 0_ 0]=
ftiempomedio=3.500000
fvarianza=2.350000
qmsgresp=10
etapa_ 0_ 0=IND_ 1
[p 0_ 1]=
ftiempomedio=3.500000
fvarianza=2.350000
qmsgresp=10
etapa_ 0_ 1=IND_ 1
[p 1_ 0]=
ftiempomedio=3.500000
fvarianza=2.350000
qmsgresp=10
etapa_ 1_ 0=IND_ 1
[p 1_ 1]=
ftiempomedio=3.500000
fvarianza=2.350000
qmsgresp=10
etapa_ 1_ 1=IND_ 1
[p 2_ 0]=
ftiempomedio=3.500000
fvarianza=2.350000
qmsgresp=10
etapa_ 2_ 0=IND_ 1
[p 2_ 1]=
ftiempomedio=3.500000
fvarianza=2.350000
qmsgresp=10
etapa_ 2_ 1=IND_ 1
```

pari1.txt

Al ser procesos individuales, el número de procesadores y procesos coincidirá. En este ejemplo, son 7.

Luego ejecutamos el generador de aplicaciones, con el número de procesos indicado en la variable qprocesos, con una lista de procesadores exactamente igual a la indicada en la variable qprocesadores y con el fichero obtenido con Gaspar.

```
mpirun –np=7 –machinefile=rhosts gaps pari1.txt
```

El generador en su máximo nivel de debug, genera 7 ficheros de salida, uno por cada proceso, donde indica todas las propiedades del mismo: (idworker, idmaster, nomnodo, etc), el flujo de envío y recepción de mensajes, el tiempo total de cada iteración, tiempo total, tiempo medio, porcentaje de desvío estándar, cantidad de mensajes recibidos y enviados.

Caso 2: Etapas agrupadas.

Este es un caso de prueba de agrupamiento de las últimas dos etapas del worker 0. La Figura 13 muestra el modelo y con colores se muestra el resultado correcto de la distribución de nodos obtenido en la prueba.

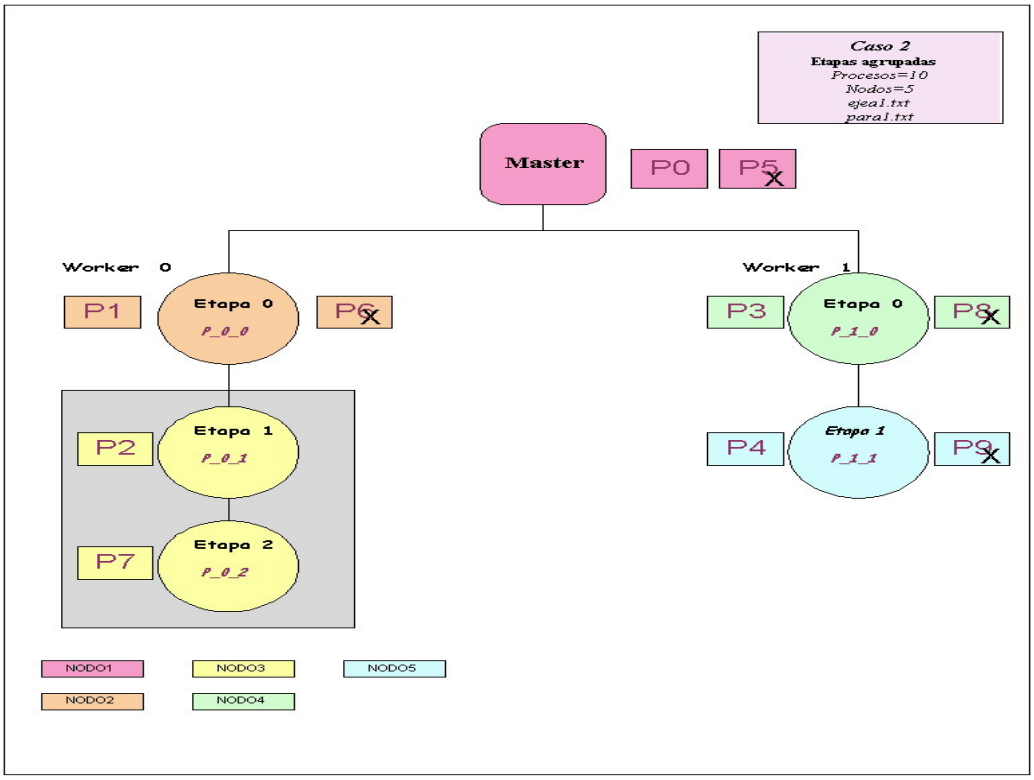

Figura 13. Caso de prueba 2. Etapas agrupadas.

Caso 3: Etapas agrupadas

En el siguiente caso, Figura 14, se agrupan las dos primeras etapas del worker 0.

Esto obliga a que el número de procesos se duplique ya que se tienen que generar dos en cada procesador, y luego se eliminan los que sobran.

En el diagrama se observan los procesos que se generan y cancelan con una X.

Este caso muestra que el sistema funciona con agrupamientos, incluso siendo una primer etapa del pipeline.

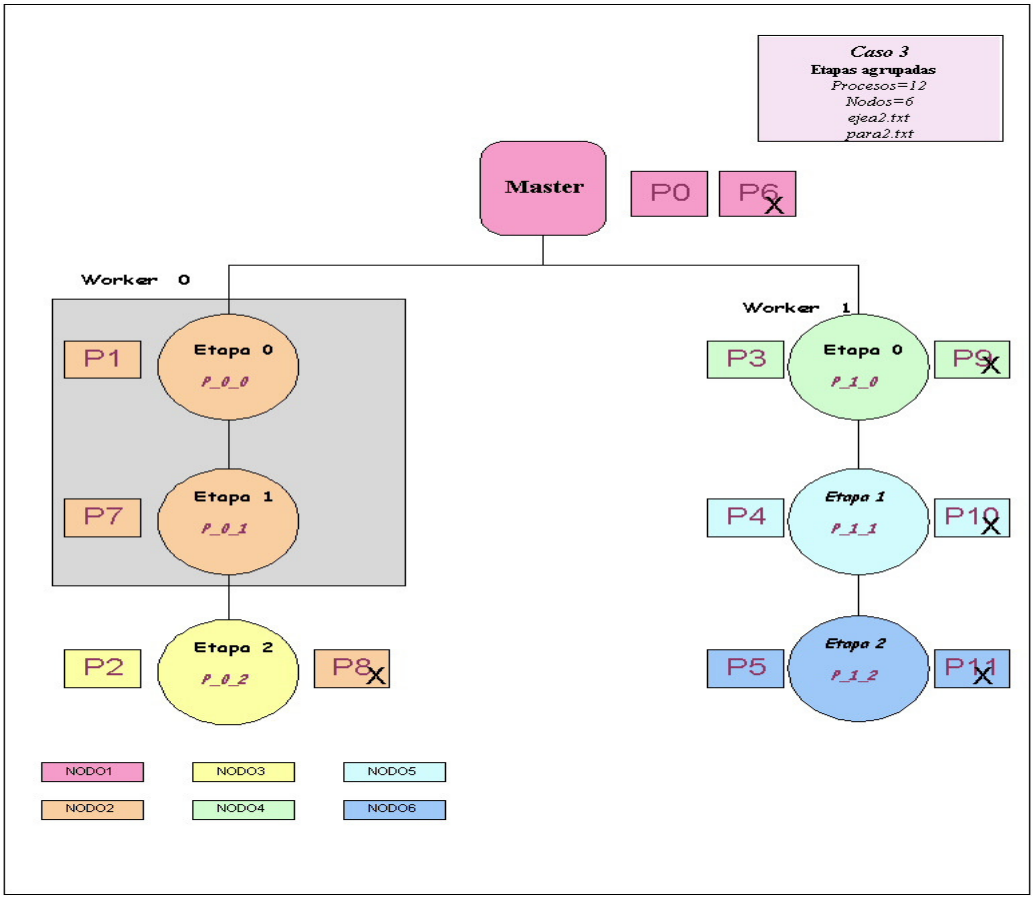

Figura 14. Caso de prueba 3. Etapas agrupadas.

Caso 4: Etapas replicadas

En este caso, representado en la Figura 15, se replica la primera etapa del worker 1.

Esta réplica hace que se necesite el proceso gestor, "comunication manager CM" que se ubica en el mismo procesador que una de ellas, ya que en principio, la sobrecarga de procesamiento es mínima, ya que solo se dedica a concentrar los mensajes que van destinados o salen de la etapa replicada que gestiona.

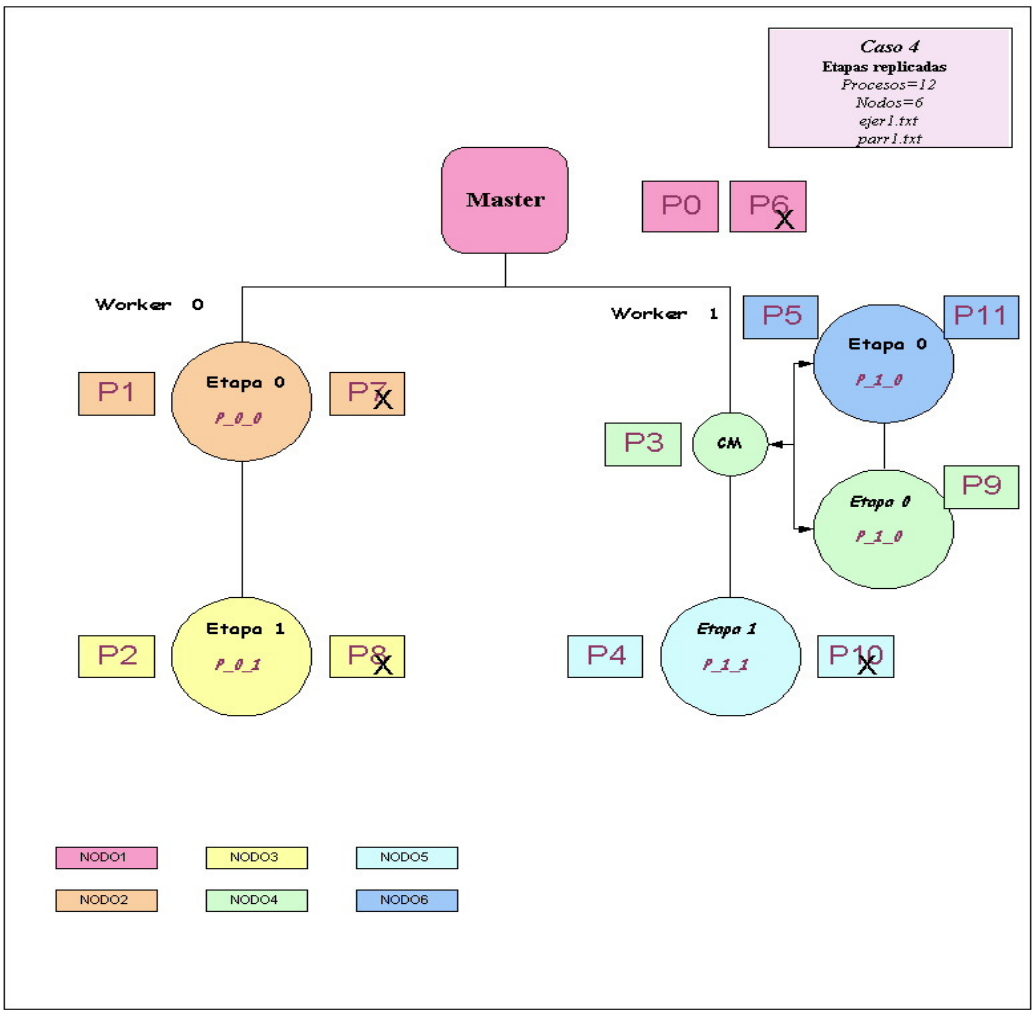

Figura 15. Caso de prueba 4. Etapas replicadas.

Caso 5: Etapas replicadas

En el siguiente caso, representado en la Figura 16, se replican 3 veces la primer etapa del worker 0.

Al igual que el caso anterior, aparece el proceso CM, que se localizará en el mismo procesador que una de ellas.

La diferencia con el ejemplo anterior es que el número de réplicas es tres.

Al haber un agrupamiento implícito de dos procesos en este caso, se duplica de 8 a 16 el número de procesos necesarios.

Se observa que en en esta caso también funciona tal como se espera, los procesos se ubican correctamente en los procesadores y se establece el flujo de mensajes entre etapas en forma correcta.

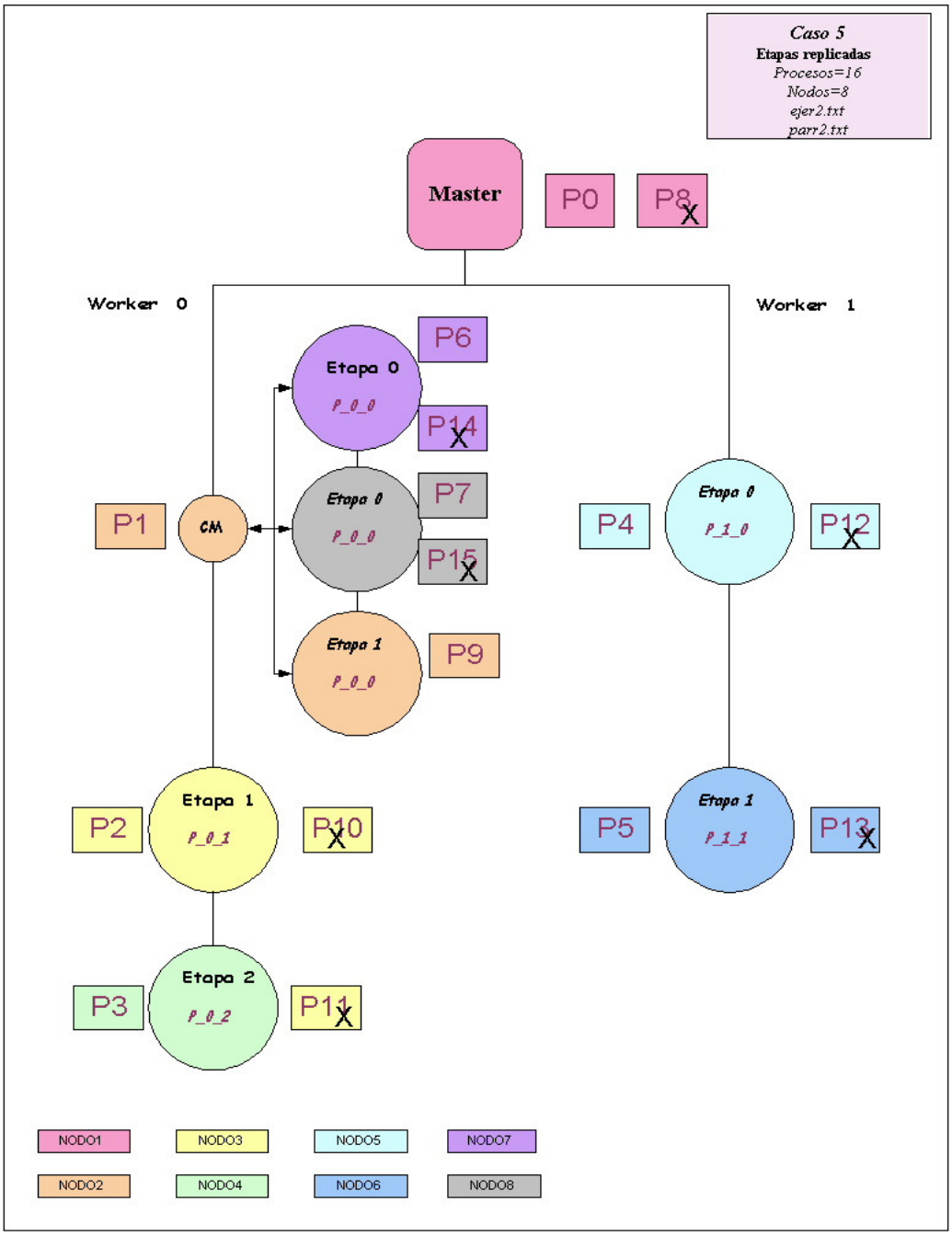

Figura 16. Caso de prueba 5. Etapas replicadas.

Caso 6: Etapas agrupadas y replicadas

La figura 17 presenta el caso 6, con la etapa 1 del worker 0 replicada en dos procesadores y con la etapa 0 del worker 1 agrupada en un mismo procesador con la siguiente.

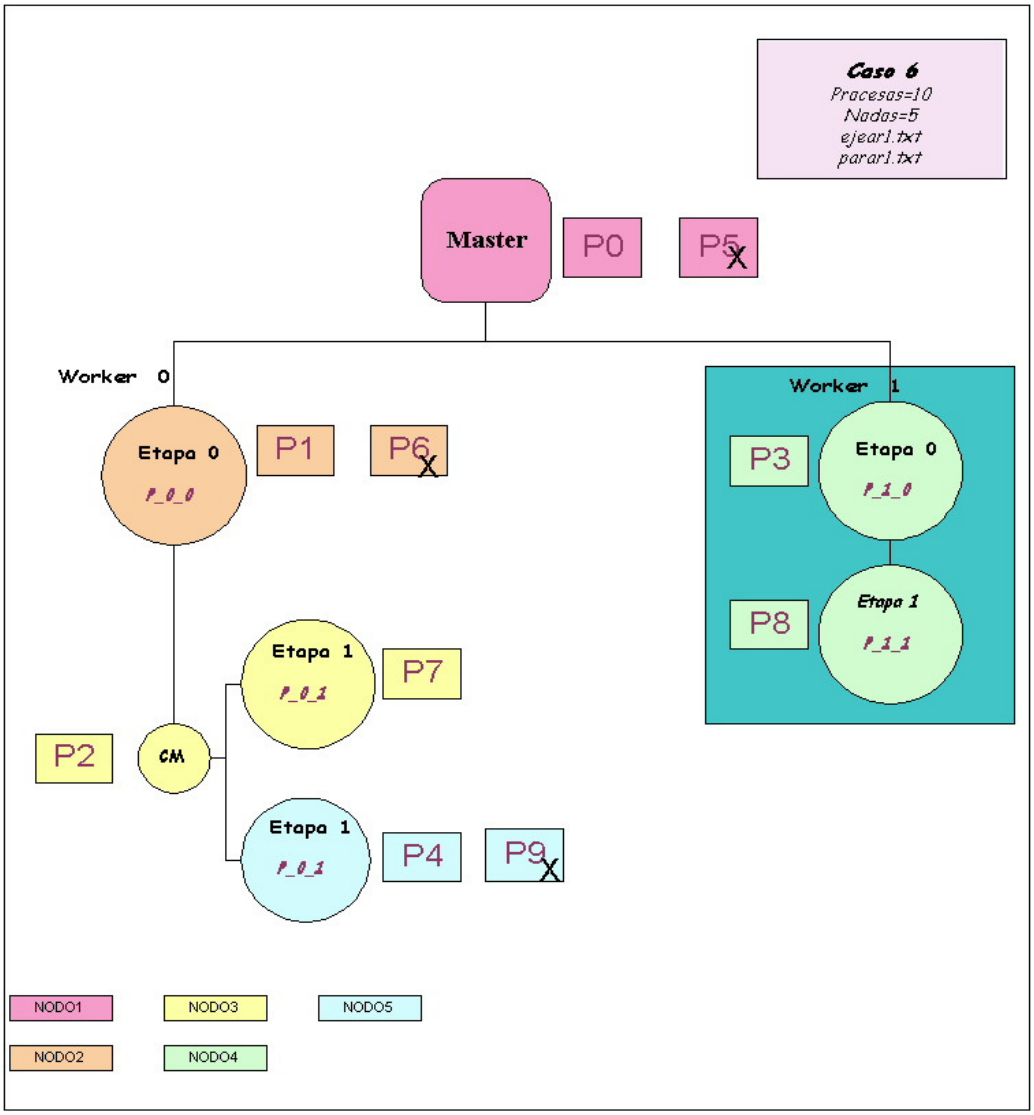

Figura 17. Caso de prueba 6. Etapas agrupadas y replicadas.

Caso 7: Etapas agrupadas y replicadas

En este caso se prueban dos réplicas diferentes: la etapas 0 del worker 0 en dos procesadores y la etapa 2 del mismo worker en tres procesadores. Se agrupan las dos etapas del worker 1.

Se representa con la Figura 18.

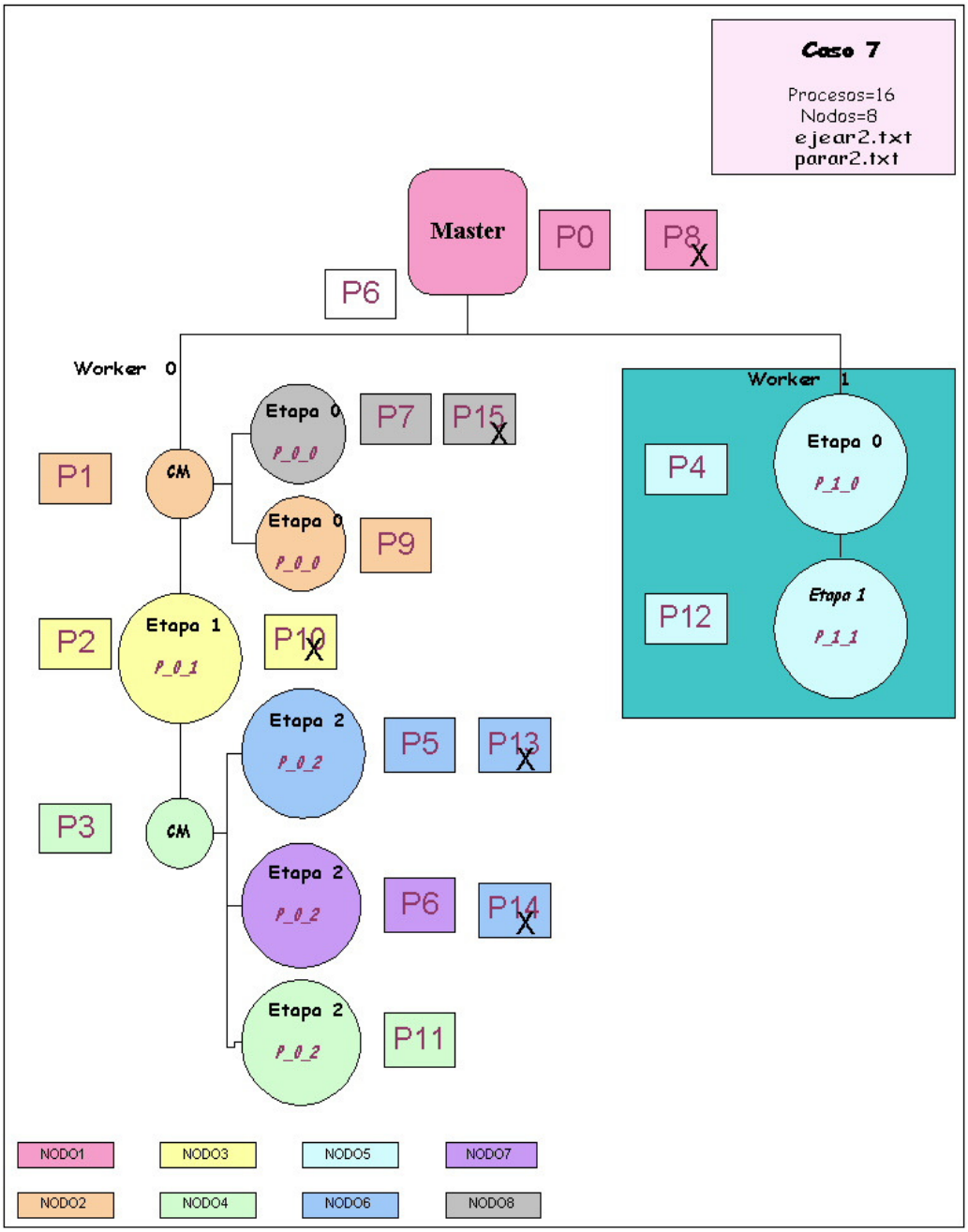

Figura 18. Caso de prueba 7. Etapas agrupadas y replicadas.

Caso 8: Etapas agrupadas y replicadas

En este caso, Figura 19, se prueban dos réplicas también en las etapas 0 y 1 del worker 0 (en dos procesadores) y se agrupan las etapas 1 y 2 del worker 1.

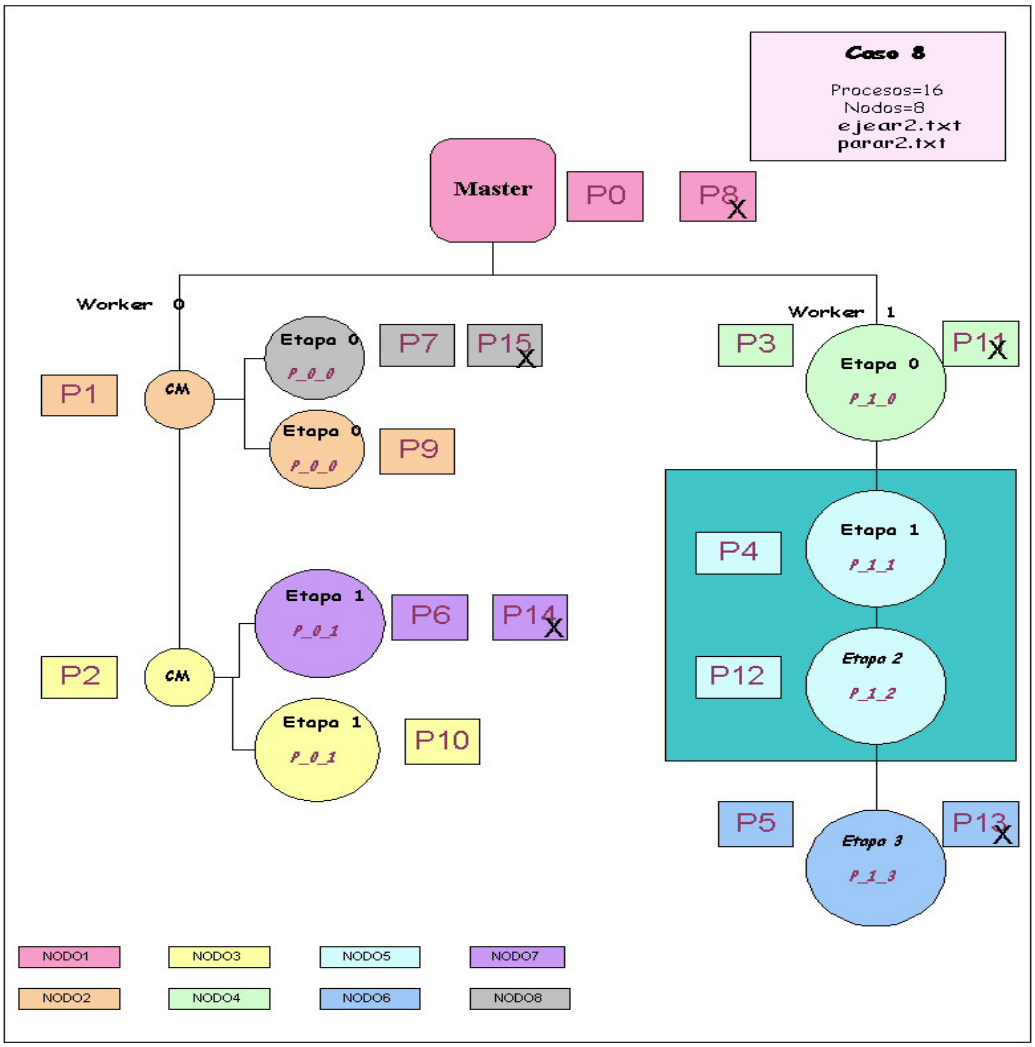

Figura 19. Caso de prueba 8. Etapas agrupadas y replicadas.

5.3 Pruebas de Ejecución

Se prueban dos casos de aplicaciones. En cada uno de estos primero se ejecuta Gaps sin agrupaciones ni réplicas emulando la situación original y luego se ejecutan otra vez, aplicacando las formulas de sintonización.

Caso 1: Master con 2 Workers.

En este caso se utilizaron mensajes de 1000 bytes, con 10 mensajes iniciales y 5 iteraciones en total.

Por tanto, el master envia 50 mensajes en total a cada worker.

La Figura 20 muestra el resultado de ambas ejecuciones, inicialmente sin agrupaciones ni réplicas y luego y sintonizado.

Se utilizan diferentes colores para identificar el nodo donde se ejecuta cada proceso.

Cada etapa muestra la duración en segundos en su interior.

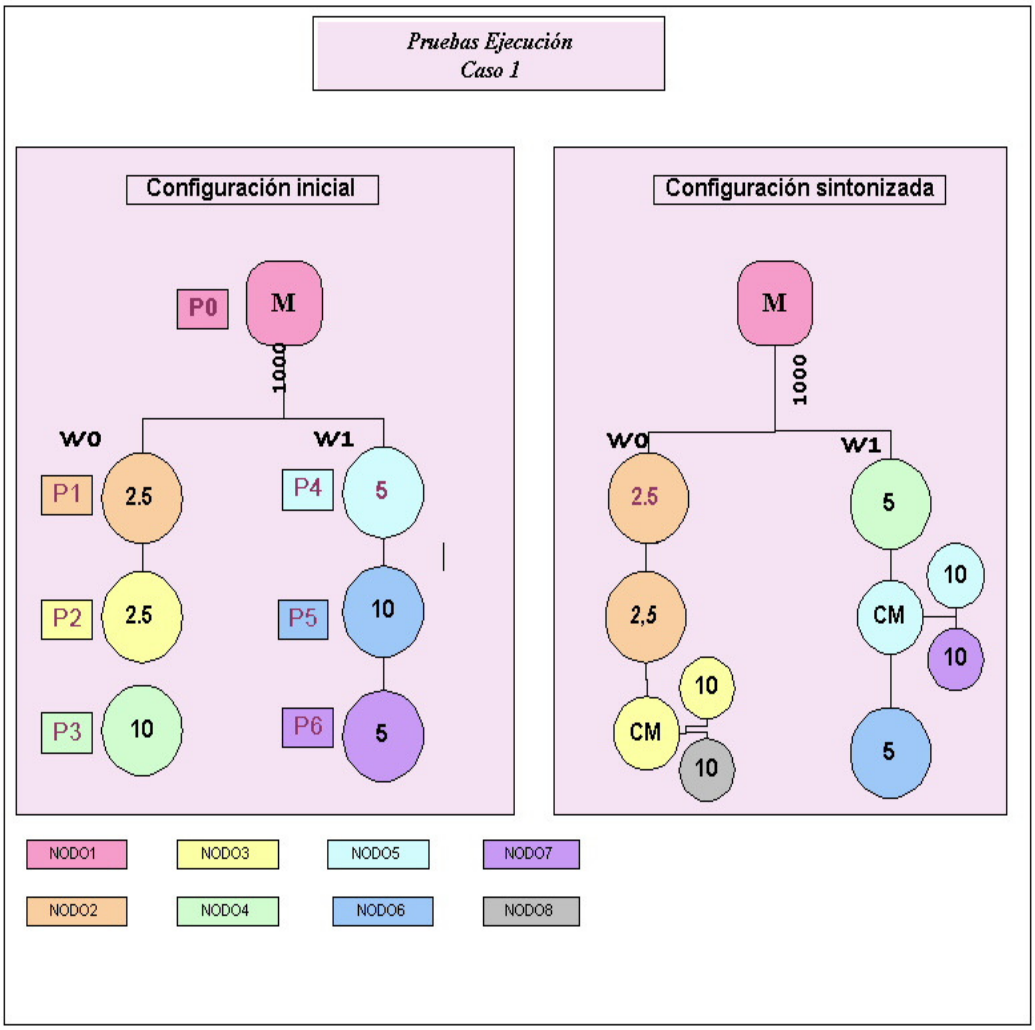

Figura 20. Pruebas de ejecuciones. Caso 1.

Los resultados obtenidos se indican en el Cuadro 8.

Configuración Inicial							
Proc	**nodo**	**Total Seg**	**T.Medio**	**T/tarea**	**Desvio**	**Msg.Env.**	**Msg.Rec.**
0	1	**524.97**	**108.54**	**5.43**	**0.54**	**12**	100
1	2	464.89	92.97	9.30	35.26	**51**	6
2	3	467.89	9.35	9.35	21.49	51	51
3	4	539.97	10.79	**10.79**	2.44	50	51
4	5	483.97	96.79	9.68	26.17	51	6
5	6	538.97	10.77	**10.77**	2.40	51	51
6	7	542.97	10.85	10.85	2.68	50	51
Totales		**548.37**	**109.67**	**5.43**	**0.73**	**316**	**316**

Configuración sintonizada							
Proc	**Nodo**	**Total Seg**	**T.Medio**	**T/tarea**	**Desvio**	**Msg.Env.**	**Msg.Rec.**
0	1	320.17	**64.03**	**3.20**	**0.65**	**12**	100
1	2	286.24	57.25	5.72	15.66	**51**	6
2	3	315.07	3.15	3.15	2.99	103	101
3	4	306.14	61.23	6.12	6.77	51	6
4	5	314.97	3.15	3.15	2.51	103	101
5	6	320.13	6.40	6.40	4.19	50	51
6	7	315.07	**12.60**	**12.60**	**5.33**	26	26
7	8	315.13	**12.61**	**12.61**	**5.44**	16	16
8	1						
9	2	289.23	5.78	5.78	9.38	51	51
10	3	311.07	12.44	12.44	5.32	36	36
11	4						
12	5	310.97	12.44	12.44	5.32	26	26
13	6						
14	8						
15	9						
Totales		**320.17**	**64.03**	**3.20**	**0.01**	525	520

Relación		**58.39%**	**58.39%**	**58.96%**		**166.14%**	**164.56%**

Cuadro 8: Caso de prueba 1: Tiempos de ejecución

Se obtiene una mejora en el tiempo de ejecución del 41,61%, con un overhead de mensajes enviados y recibidos del 66,14% y 64,56%, que se produce por incorporar al comunication manager.

Caso 2: Master con 3 workers.

En este caso también se utilizaron mensajes de 1000 bytes, con 10 mensajes iniciales y 5 iteraciones en total.

Por tanto, el master envía 50 mensajes en total a cada worker.

La Figura 21 muestra el resultado de ambas ejecuciones, en su modo original y sintonizado. Se utilizan diferentes colores para identificar el nodo donde se ejecuta cada proceso.

Cada etapa muestra la duración en segundos en su interior.

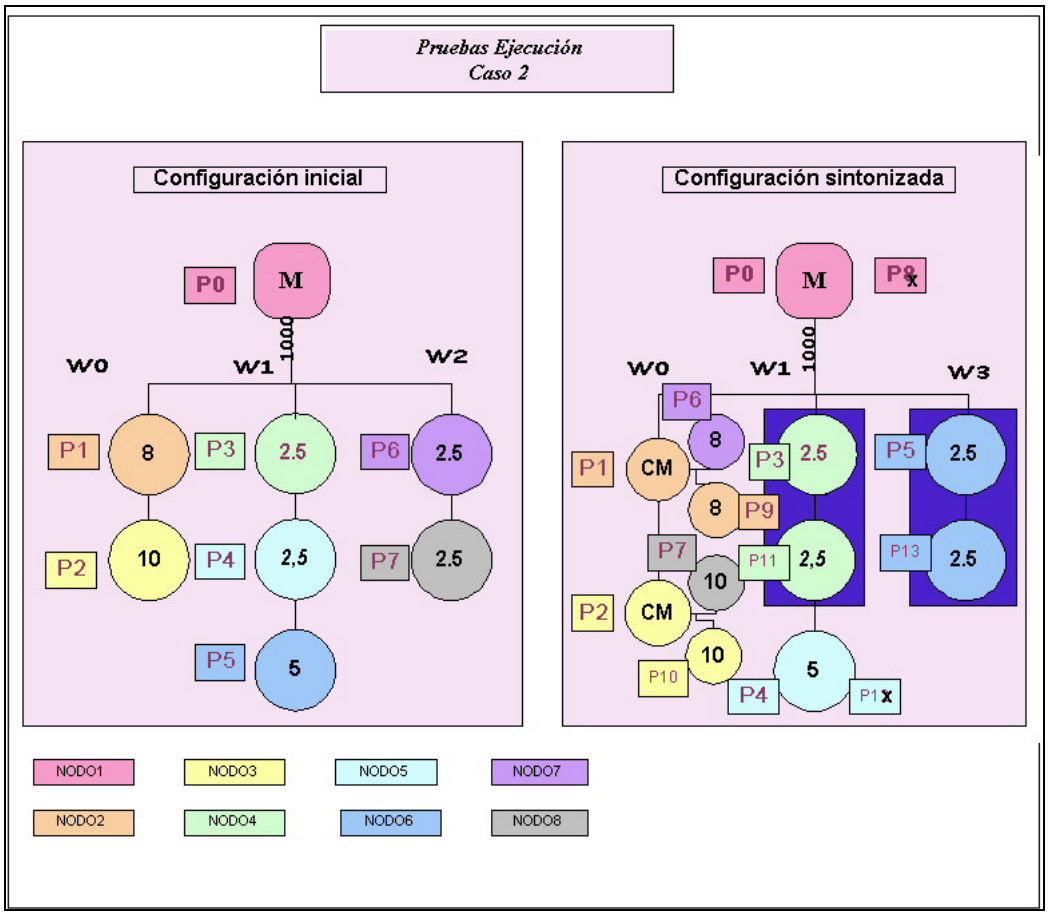

Figura 21. Pruebas de ejecuciones. Caso 2.

Los resultados obtenidos en las dos ejecuciones se indican en el Cuadro 9, con la relación de aos resultados comparativos.

Configuración Inicial							
Proceso	**nodo**	**Total Seg**	**T.Medio**	**T/tarea**	**Desvio**	**Msg.Env.**	**Msg.Rec.**
0	1	**545.44**	**109.08**	**3.64**	**0.19**	18	150
1	2	536.44	107.28	**10.73**	4.23	51	6
2	3	545.44	10.99	**10.99**	2.87	50	51
3	4	466.44	93.28	9.33	36.07	51	6
4	5	468.44	9.36	9.36	21.76	51	51

Configuración Inicial							
Proceso	**nodo**	**Total Seg**	**T.Medio**	**T/tarea**	**Desvio**	**Msg.Env.**	**Msg.Rec.**
5	6	491.43	9.82	9.82	16.42	50	51
6	7	466.44	93.20	9.32	36.08	51	6
7	8	468.44	9.36	9.36	21.90	50	51
Totales		**545.44**	**109.08**	**3.64**		372	372

Configuración sintonizada							
Proceso	**nodo**	**Total Seg**	**T.Medio**	**T/tarea**	**Desvio**	**Msg.Env.**	**Msg.Rec.**
0	1	**374.95**	74.99	**2.50**	0.02	18	150
1	2	365.84	6.65	0.67	1.20	103	56
2	3	374.85	3.75	3.75	1.30	103	101
3	4	329.85	65.97	6.60	20.19	51	6
4	5	355.84	7.12	7.12	6.87	50	51
5	6	329.75	65.95	6.59	20.24	51	6
6	7	374.85	14.99	**14.99**	4.07	26	26
7	8	365.84	14.63	14.63	3.73	26	26
8	1						
9	2	358.85	4.04	4.04	4.04	26	26
10	3	368.85	3.75	**3.75**	3.75	26	26
11	4	332.85	12.32	12.32	12.32	51	51
12	5						
13	6	332.75	12.32		12.32	50	51
14	7						
15	8						
Totales		**374.95**	**74.99**	**2.50**	**0.08**	581	576

Relación		**68.74%**	**68.75%**	**68.75%**		**156.18%**	**154.84%**

Cuadro 9: Caso de prueba 2: Tiempos de ejecución

Se obtiene una mejora en el tiempo de ejecución del 31,26%, con un overhead de mensajes enviados y recibidos del 56,18% y 54,84%, que produce el hecho de incorporar a los CM.

6. Conclusiones

La herramienta **Gaps** desarrollada en el presente trabajo ofrece un elemento fundamental para el trabajo de la investigación del modelo de rendimiento y sintonización para aplicaciones paralelas compuestas del tipo Master/Worker s de pipelines.

El uso de una aplicación sintética es un paso necesario en el desarrollo de modelos de rendimiento de aplicaciones. Sin esta herramienta, el investigador debía desviarse por un momento del objeto principal de su investigación y desarrollar una aplicación sintética a medida para experimentar con las formulas analíticas de su modelo.

En cambio, a partir de ahora, el investigador podrá comprobar las funciones de rendimiento y los puntos de medida que decida para aplicaciones de este modelo y podrá estudiar la variación de parámetros dejando otros fijos, utilizando una herramienta ya preparada y probada para todos estos casos, que se irá consolidando a medida que se la utilice, pero en forma independiente y siguiendo objetivos propios, que son diferentes y, sobre todo, transcienden a los de una sola investigación.

Gaps constituye una nueva herramienta para la línea de investigación de rendimiento y sintonización de aplicaciones paralelas. Ayudará a generar nuevos modelos y mejorar los existentes que ya se encuentran desarrollados e integrados con el resto de herramientas (MATE y Poetries por ejemplo).

Gaps nace para crear aplicaciones del modelo compuesto de Master/Worker de pipeline, pero esto no impide que se incorporen en el futuro otros patrones de aplicaciones paralelas en la herramienta.

7. Trabajos futuros

El generador Gaps permitirá un conjunto más amplio de experimentos con el patrón de aplicaciones Master/Worker de pipelines (que abarca a ambos en forma separada también) con las siguientes extensiones que se proponen realizar en un futuro:

- Permitir una distribución de tamaño diferente de tareas en las diferentes iteraciones, para permitir al investigador estudiar la estrategia de Dynamic Adjunsting Factoring (DAF).

- Permitir que el proceso del comunication manager se ejecute en un procesador diferente de la replica. En la actual implementación, el investigador no puede independizar estos procesos, limitando sus experimentos.

- Gaps actualmente sigue un modelo de Master/Worker de tareas dependientes, porque el master espera a que todos los workers finalicen para comenzar la siguiente iteración. Se propone incluir en la configuración un comportamiento de tareas independientes (embarrassingly parallel) para aumentar la variedad de modelos a generar.

Además Gaps puede extenderse a otros patrones de aplicaciones como Divide and Conquer, SPMD, MDMD, realizando combinaciones de procesos más generales que los desarrollados hasta el momento.

8. Referencias

1. Cahon, S.; Melab, N. y Talbi, E.-G.- ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics - J.Heuristics, 2004, 10, 3, 357-380, Kluwer Academic Publishers, Hingham, MA, USA

2. Cesar, E.; Sorribes, J. y Luque, *E.* - Modeling pipeline applications in POETRIES – 2005, 83-92, Springer-Verlag, Berlin, Germany

3. Morajko, A., Caymes, P., Margalef, T. and Luque, E. - Automatic tuning of data distribution using factoring in master/worker applications - 2005, SPRINGER-VERLAG BERLIN}, HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY

4. Cesar, E.; Moreno, ,A.; Sorribes, J. y Luque, E. - Modeling master/worker applications for automatic performance tuning - Parallel Comput., 2006, 32, 7, 568-589, Elsevier Science Publishers B. V, Amsterdam, The Netherlands, The Netherlands

5. Park, ,Alfred, Fujimoto, Richard M. - Aurora: An Approach to High Throughput Parallel Simulation - 2006, 3-10, IEEE Computer Society, Washington, DC, USA

6. Taufer, ,Michela, Kerstens, Andre, Estrada, ,Trilce, Flores, ,David and Teller, ,Patricia J. - SimBA: A Discrete Event Simulator for Performance Prediction of Volunteer Computing Projects - 2007, 189-197, IEEE Computer Society, Washington, DC, USA

7. Moreno, A., Cesar, E., Guevara, A., Sorribes, J., Margalef, T. y Luque, E. - Dynamic Pipeline Mapping (DPM) - 2008, 5168, 295-304, SPRINGER-VERLAG BERLIN}, HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY

8. Morajko A., Morajko O., Jorba J., Margalef T., Luque E., - Automatic performance analysis and dynamic tuning of distributed applications Parallel Processing Letters, World Scientific, 2003, pp. 169–187.

9. Massingill B.L. , Mattson T.G., Sanders B.A., A Pattern Language for Parallel Application Programs. LNCS, vol. 1900, Springer Verlag, 2000 (Euro-Par 2000), pp. 678–681.

10. Cesar E., Mesa J.G., Sorribes J., Luque E., POETRIES: performance oriented environment for transparent resource-management, implementing end-user parallel/distributed applications LNCS, vol. 2790, Springer-Verlag, 2003, pp. 141–146.

11. Cesar E., "Definition of framework-Based Performance Models for Dynamic Performance Tunning" Ph.d thesis, University Autonomous of ercelona, Belaterra, Barceloan, Spain, 2006.

12. Morajko A., "Dinamic Tuning of Parallel/Distributed Applications" " Ph.d thesis, University Autonomous of Barcelona, Belaterra, Barcelona, Spain, 2004.