ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

COMPUTACIÓN DISTRIBUIDA PARA EL CÁLCULO INTENSIVO DE ISOMORFISMO DE GRAFOS

Realizado por
JAVIER FERNÁNDEZ RODRÍGUEZ

Dirigido por
JOSÉ RAMÓN PORTILLO FERNÁNDEZ

Departamento
MATEMATICA APLICADA I

Sevilla, Junio 2009

Índice de Contenido

1.- Introducción 3
2.- Herramientas, Análisis de antecedentes y Comparación con otras Alternativas 6
2.1.- Estado del arte del Cálculo Distribuido 6
2.2. Estado del Arte en Algoritmia de Grafos 8
2.2.1. Condor 14
2.2.2. Sun Grid Engine (SGE) 17
2.2.3. Nauty 19
2.2.4.- Otras Alternativas Software de generación de grafos 21
2.3.- Formatos de Fichero para Grafos 22
2.3.1. Formato Graph6 22
2.3.2. Formato Sparse6 25
3. Definición de Objetivos 27
3.1. Descripción del problema 27
3.2 Objetivos secundarios 28
4.- Análisis de Requisitos, diseño e implementación 29
4.1.- Diccionario de Requisitos 29
4.2.- Requisitos 29
5.- Análisis temporal y costes de desarrollo 33
6.- Conclusiones y desarrollos futuros 35
7.- Generación de grafos con un subgrafo completo K_n 37
8.- Extensión de nauty 41
Apéndice A: Estadísticas de uso de geng. 44
Apéndice B: Plantillas de CLASSADs Condor del CICA 46
Apéndice C: Instalación de Condor sobre una red de Computación 47
Apéndice D: Instalación de Sun Grid Engine sobre una red de Computación. 50
Apéndice E: Script de ejecución en Sun Grid Engine 52
ANEXO I: Índice de Ilustraciones 53
Bibliografía: 54

1.- Introducción

Durante los últimos años se han producido grandes avances en el ámbito de la computación distribuida. Desde hace mucho tiempo, cuando se formaron las primeras redes de ordenadores, se ha intentado aprovechar los recursos de computación de aquellos ordenadores que permanecían inactivos en la red. En la actualidad la computación distribuida está presente en numerosos proyectos repartidos por todo el mundo. Con la computación distribuida se puede alcanzar una potencia de cálculo muy superior a aquellas supercomputadoras que funcionaban aisladas y completamente dedicadas a un único propósito de cálculo.

Aunque a día de hoy continúa siendo válida la Ley de Moore que nos predice que cada 18 meses se duplica la potencia de los ordenadores, no son capaces de afrontar la demanda actual de procesamiento y almacenamiento requerido por las investigaciones que están siendo llevadas a cabo en la actualidad. A principio de los años 70 se comenzaron a conectar las primeras computadoras, hasta que comenzaron a experimentar con la computación distribuida pasaron algunos años y no fue hasta la llegada de *Arpanet* (predecesor del actual Internet) cuando programas como *Creeper* y *Reaper-ran* explotaron esa faceta de la computación. El siguiente paso en la computación distribuida lo dio *Xerox*, que en su centro de Palo Alto creo la primera red Ethernet sobre la que experimentaron el uso de los recursos conectados inactivos.

Tras la explosión de uso de las nuevas redes de ordenadores comenzaron a encontrarse los problemas derivados de la cantidad de información a manejar por los proyectos que se incrementaba exponencialmente. Los primeros acercamientos a las diferentes soluciones proponían siempre la distribución del cálculo entre los ordenadores de la red, con ello solucionaban numerosos impedimentos entre los que se encuentran:

a) **Almacenamiento**: El volumen de datos del problema y de la solución resulta bastante difícil de manejar por la reducida capacidad tanto de memoria volátil como persistente. La incorporación de redes de computadores permite repartir el almacenamiento entre los diferentes ordenadores que trabajarán con esa fracción de información, agilizando todo el proceso.

b) **Cálculo:** Siempre que el problema permita el cálculo paralelo un supercomputador no podrá competir con una extensa red de computación formada por decenas (o miles)

de ordenadores dedicando todo su tiempo de proceso a resolver el problema propuesto. La potencia de cálculo de una red de computación aumenta con el número de ordenadores conectados a ella, sin embargo necesitan una infraestructura especializada que puede llegar a saturarse si no está a la altura del movimiento de la información requerido.

c) **Tiempo:** Numerosos proyectos por su propia naturaleza requieren calcular todas las posibles combinaciones para obtener una solución. Los cálculos que de otro modo podrían llevar cientos de años, con la computación distribuida se podría resolver en una fracción del tiempo que tardaría la computación tradicional.

A principios de los 90, con la implantación de Internet se globalizó el uso de la computación distribuida. Se realizaron experimentos públicos con los usuarios de Internet sobre computación distribuida hasta que en el año 1997 se estableció el proyecto *dnet* (distributed.net) que llegó a conectar 160000 ordenadores Pentium II para resolver claves criptográficas. Otro proyecto con aún más repercusión se denominó *SETI@home* que envió su último trabajo el 22 de Diciembre del año 2005, aunque se creó un nuevo proyecto *SETI@home* que tiene conectados en la actualidad a más de cinco millones y medio de ordenadores. Anteriormente existieron otros proyectos SETI dedicados a la búsqueda de vida extraterrestre basados en búsqueda de patrones en señales electromagnéticas. *SETI@home* ha inspirado numerosos proyectos de computación distribuida con los ordenadores conectados a internet como recurso de computación, entre ellos están *Genome@home* que pretende entender el funcionamiento del genoma humano, el proyecto *Folding@home* para el estudio de las proteínas y las enfermedades asociadas a ellas y el proyecto *FightAIDS@home* para la búsqueda de un remedio contra el SIDA.

Todos los avances en el campo de la computación distribuida llevan a la denominada computación **GRID**, que es capaz de realizar cálculo intensivo haciendo uso de CPU o espacio de almacenamiento en ordenadores con bajo nivel de acoplamiento sobre una red heterogénea. Los denominados “padres” de la computación GRID *Ian Foster*, *Carl Kesselman* y *Steve Tuecke* lideran el desarrollo del paquete de herramientas *Globus Toolkit*, que implementa además de un sistema de computación distribuida el sistema de almacenamiento compartido, herramientas de seguridad, monitorización y herramientas para el desarrollo de aplicaciones basadas en la infraestructura GRID. En el año 2007 se hizo popular el nombre de *Nube de Computación* para referirse a la Computación GRID.

Las ventajas de la computación GRID sobre el resto de sistemas de computación distribuida radican en su perfecta integración de sistemas y dispositivos heterogéneos. Se trata de una solución altamente escalable, potente y flexible, ya que tiene la capacidad de evitar los problemas relacionados con la falta de recursos (cuellos de botella) debido a la posibilidad de modificar el número y características de sus componentes.

No obstante la computación GRID también dispone de algunos inconvenientes que podrían deteriorar el correcto funcionamiento, que dependiente del problema podría convenir otras soluciones más tradicionales. Estos inconvenientes son:

a) Debe disponer de las herramientas y las características necesarias para manejar los recursos heterogéneos de forma transparente y eficaz, sin importar el sistema o el hardware que se conecte a la red.

b) La red debe ser constantemente monitorizada para poder controlar externamente los procesos que se encuentren en ejecución. Este punto requiere un porcentaje de recursos dedicados al descubrimiento, selección, reserva, asignación, gestión y monitorización de recursos.

c) Los recursos anteriores necesitan un software especializado para el manejo de GRID, con todas las utilidades necesarias para su correcta gestión además del diseño de un modelo de uso eficiente.

d) Las capas de abstracción que se implementan para el funcionamiento del GRID en una red heterogénea pueden provocar deceleraciones en la comunicación.

2.- Herramientas, Análisis de antecedentes y Comparación con otras Alternativas

2.1.- Estado del arte del Cálculo Distribuido

El máximo potencial de la supercomputación sólo se obtiene si disponemos de un software especializado que pueda dividir el problema en sub-problemas que se distribuyan entre los diferentes recursos. Antes que la supercomputación fuese asequible para la comunidad científica media se utilizaba software para trabajar con un solo procesador, por lo que tras la aparición de estos nuevos paradigmas de computación es necesario actualizar el software científico.

Esta adaptación del software ha tomado dos caminos diferenciados: el primero es la adaptación a las nuevas tecnologías de computación a través de la creación de nuevos compiladores especializados en la gestión de sistemas distribuidos (que servirían para crear nuevo software) y una segunda línea que modifica el programa original utilizando herramientas clásicas pero con las nuevas metodologías.

Podemos encontrar algunos ejemplos de estas nuevas herramientas en conocidas aplicaciones de cálculo en:

a) *gridMathematica* (basado en *Mathematica* 6) para el cálculo distribuido y paralelizado.

b) *Mathematica Personal Grid Edition* como solución de cálculo distribuido y paralelizado para máquinas de cuádrupe núcleo.

c) *NAG Fortran SMP Library*, librería basada en Fortran para sistemas *SMP* (*Symmetric Multi-Processor*).

d) *NAG Parallel Library*, librería de funciones de paralelismo para desarrollo de aplicaciones *MPI* (*Message Passing Interface*).

e) Herramientas Cluster de Intel de asistencia a los desarrolladores de aplicaciones y gestores de sistemas distribuidos.

f) Librerías de Rendimiento de Intel para el procesado y cálculo matemático para desarrolladores multimedia y científicos.

GridMathematica es una solución basada en *Mathemática* para resolver problemas en el campo de las ciencias, ingeniería y análisis financiero que requieran grandes volúmenes de datos y cálculos de gran coste computacional. Es un sistema basado en licencias que dispone de un kernel principal (maestro), un gestor de licencias y un kernel de *Mathemática* en cada máquina (esclavos). El maestro gestiona la entrada/salida y la secuenciación de cálculos. *GridMathematica* no acelera todos los cálculos, pues existen algunas operaciones diseñadas para trabajar sobre un único procesador (*Integrate* o *DSolve*), sin embargo es posible ejecutar a la vez estas operaciones si se realizan en diferentes kernels esclavo.

Mathematica Personal Grid Edition es una solución de paralelismo a pequeña escala, para trabajar sobre una única máquina multi-núcleo (específicamente quad-core). Dispone de un kernel principal (maestro) y un kernel (esclavo) para cada uno de los cuatro kernels restantes. Implementa primitivas para la programación en paralelo e incluye comandos de alto nivel para la ejecución en paralelo de operaciones matriciales, representación visual, simulación, búsqueda… Al igual que su versión superior (*GridMathematica*) no puede acelerar todos los cálculos pues algunas operaciones están diseñadas para una única línea de ejecución (un solo procesador).

NAG Fortran SMP Library, dispone de 231 rutinas para permitir hacer uso de la potencia de procesado y paralelismo de memoria compartida de los sistemas *SMP* (*Symmetric Multi-Processor*). La librería está diseñada para ofrecer la misma interfaz que su equivalente implementación monoprocesador, por lo que proporciona una fácil vía de actualización. La distinción se realiza únicamente cuando el código final se enlaza a una librería. No necesita un conocimiento especial de programación paralela para poder aplicar técnicas SMP al código. La ventaja principal de esta librería es que no sufre pérdidas de rendimiento al ser ejecutada la aplicación sobre una única máquina mono-núcleo como ocurre con las demás propuestas de implementación de la computación paralela.

NAG Parallel Library utiliza una rejilla lógica de procesadores que son asignados en los procesadores físicos disponibles, cada llamada a rutina de la librería se ejecutará en cada procesador lógico y cooperan para resolver el problema. El modelo de paralelismo adoptado en la librería es el modelo *SPMD* (*Single Program Multiple Data*) y ofrece mayor velocidad de ejecución sobre programas numéricos secuenciales convencionales y, particularmente, en redes de estaciones de trabajo (permite resolver problemas que estén por encima de la capacidad de memoria de una sola máquina).

2.2. Estado del Arte en Algoritmia de Grafos

Esta documentación no pretende ser un curso de teoría de grafos, pero para su completo entendimiento es importante definir alguna referencia de algunos de los conceptos utilizados:

- **Vértice**: Es la unidad fundamental de un grafo. Se puede utilizar para modelar objetos o situaciones en un grafo que serán relacionados entre ellos por las *Aristas*.

Vértice V

Ilustración 1 - Vértice

- **Arista**: Identifican las relaciones entre dos vértices. Existen dos tipos fundamentales de aristas: Aristas Dirigidas son aquellas que tiene asociada una dirección, por lo que posee un vértice inicial y un vértice final. Las Aristas No Dirigidas no disponen de un orden entre los vértices que conecta.

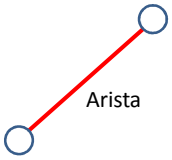


Ilustración 2 - Arista

- **Adyacencia**: Dos vértices son adyacentes si existe una arista entre ellos que los une.
- **Ciclo**: Es un camino que comienza y acaba en el mismo vértice. Los ciclos de longitud 1 se denominan bucles. Un ciclo simple es aquel que tiene como longitud mínima 3 y cada vértice sólo aparece una vez (excepto el inicial que también es final).

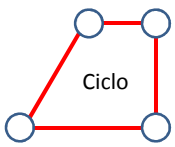
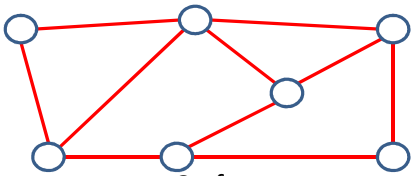


Ilustración 3 - Ciclo

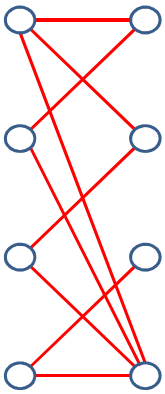
- **Grafo**: Es un conjunto de vértices y aristas que los unen. Formalmente es un par de conjuntos finitos (V, A), donde V es el conjunto de vértices y $A \subseteq V \times V$ es un conjunto de pares de vértices llamados aristas.



Grafo

Ilustración 4 - Grafo

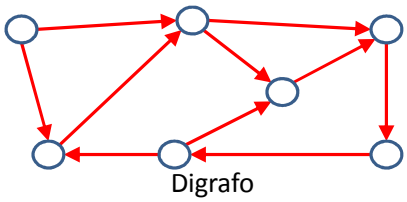
  - **Grafo Acíclico**: Es aquel que no contiene ciclos en su interior
  - **Grafo Bipartito**: Un grafo cuyos vértices pueden ser divididos en dos conjuntos, tal que no hay aristas entre dos vértices del mismo conjunto. Un grafo es bipartito si no hay ciclos de longitud impar.



Grafo Bipartito

Ilustración 5 - Grafo Bipartito

  - **Grafo Conexo**: Se dice que un grafo es conexo si es posible formar un camino desde un vértice cualquiera a otro vértice en el grafo.
  - **Digrafo o Grafo Dirigido** es un grafo en el que cada arista posee un vértice inicial y otro final.



Digrafo

Ilustración 6 - Digrafo

  - **Grafo simple** es un grafo que no posee bucles y no es un multígrafo.
  - **Grafo Vacío** es el grado que no posee vértices o aristas.

- **Multigrafo** es aquel grafo que posee aristas múltiples (o aristas paralelas).Por lo tanto un mismo par de nodos pueden estar relacionados por más de una arista.

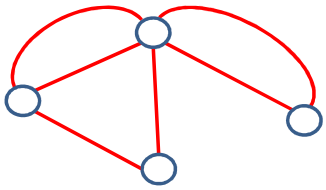


Ilustración 7 - Multigrafo

- **Camino** es una secuencia de vértices de modo que para cada vértice, el siguiente del camino está conectado con el anterior por una arista.
  - **Camino Eucleriano** es el camino que utiliza cada arista una y sólo una vez.
  - **Camino Hamiltoniano** es el camino que utiliza cada vértice una y sólo una vez.
- **Coloreado**: Se denomina coloreado a marcar o asignar diferentes colores a los vértices de un grafo de modo que ningún par de vértices adyacentes compartan la misma marca (o color).
- El **Grado** de un vértice es el número de aristas incidentes a él.
- **Matriz de Adyacencia**: Es una representación de las aristas de un grafo donde la posición *(i,j)* de la matriz indica el número de aristas desde el vértice *i* hasta el vértice *j*. Si el grafo no es dirigido la matriz es simétrica. Es una representación adecuada para grafos densos.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Ilustración 8 - Matriz de adyacencia para grafo con 3 vértices

- Un **Subgrafo** *S* es un grafo cuyo conjunto de vértices es un subconjunto de un grafo *G* y sus aristas son un subconjunto del mismo grafo *G* (el conjunto de aristas que relacionan los vértices del grafo *S*).

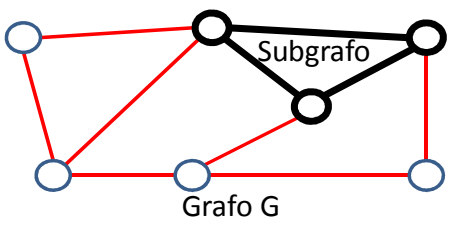


Ilustración 9 - Subgrafo

Los grafos son representados típicamente como un conjunto de vértices unidos por aristas. Se dice que dos vértices están conectados si existe una arista que los une. Plasmar un grafo en papel no debe ser confundido con el grafo en sí, un grafo es una estructura abstracta que puede ser representada gráficamente con diferentes figuras, pero todas ellas representan el mismo grafo. Este fenómeno se denomina Isomorfismo de los Grafos.

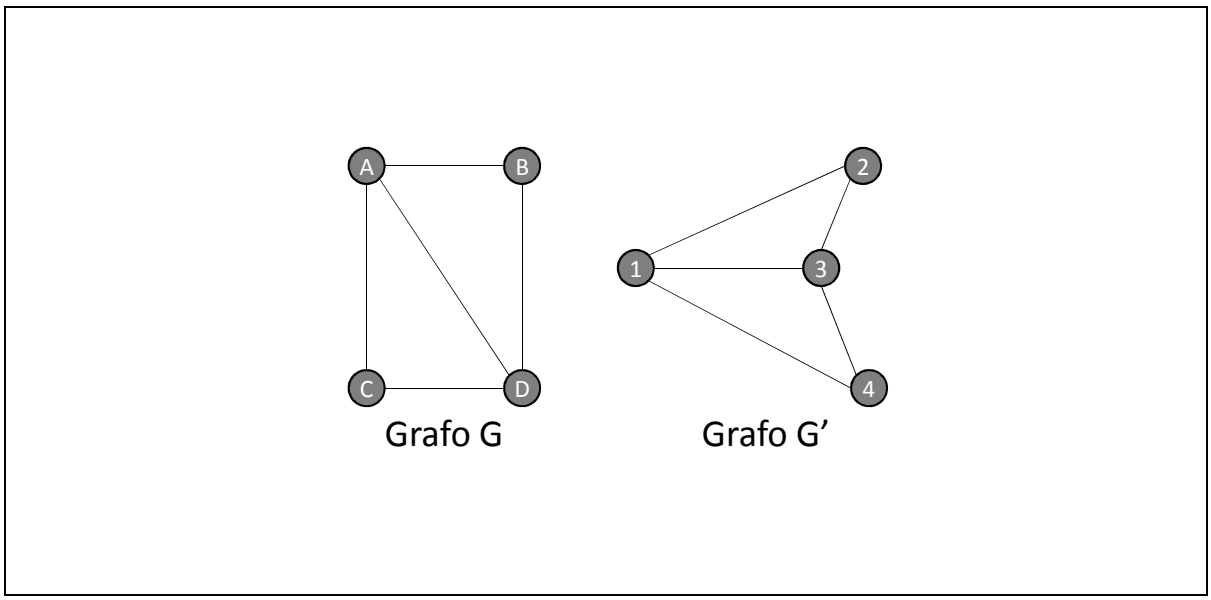


Ilustración 10 - Ejemplo de Grafos Isomorfos.

Conocer si un par de grafos (A y B) son isomorfos es posible si y sólo si existe una matriz P tal que B=PxAxP. La matriz P se denomina Matriz Permutación: una matriz cuadrada con un único 1 en cada par fila-columna. Sabemos que existen para cada n un conjunto finito de matrices de permutación (n! combinaciones) de las cuales n!/2 poseen un determinante positivo (permutación par) y n!/2 determinante negativo (permutación impar), todas ellas ortogonales y su inversa es la traspuesta de la matriz correspondiente.

$$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}_{par}\begin{pmatrix}0&0&1\\1&0&0\\0&1&0\end{pmatrix}_{par}\begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix}_{par}$$
$$\begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix}_{impar}\begin{pmatrix}0&1&0\\1&0&0\\0&0&1\end{pmatrix}_{impar}\begin{pmatrix}1&0&0\\0&0&1\\0&1&0\end{pmatrix}_{impar}$$

Ilustración 11 - Matrices de Permutación de orden 3

Es necesario mentar que no todos los problemas de las matemáticas pueden ser resueltos por una máquina. La clasificación de cada problema se trata en la denominada Teoría de la Computabilidad. La complejidad es la cantidad de recursos que son necesarios para la resolución de un problema, algoritmo o proceso de cálculo. Estos recursos son esencialmente el tiempo y el espacio, la Complejidad Temporal (tiempo de ejecución) y Complejidad Espacial (cantidad de memoria que necesita).

- Problemas **no decidibles** son aquellos que no podemos resolver (o no es factible) mediante un algoritmo.
- Problemas **decidibles** son aquellos que tienen al menos un algoritmo que los resuelve.
    - Problemas **Intratables** son aquellos que no son factibles solucionar por gran necesidad de recursos (temporales o espaciales).
    - Problemas **tratables** pueden ser resueltos.

Generalmente son tratables los algoritmos de complejidad *polinomial* (N^k). Para los algoritmos exponenciales (K^N) suelen ser intratables para valores de N elevados por su elevado tiempo de ejecución. Desde hace algunos años se intentan solucionar los altos requerimientos de los problemas Intratables para convertirlos en Tratables. Estas soluciones pasan desde aumentar la velocidad de las máquinas que calculan los algoritmos, aumentar el número de máquinas modificando apropiadamente el software (y adaptando el algoritmo al paralelismo siempre que sea posible) hasta utilizar modelos de computación no convencionales como la computación cuántica o la celular por membranas.

Todas estas soluciones pasan por el concepto de ***SuperComputación***, que intenta resolver esos problemas de complejidad extrema que no pueden ser resueltos por máquinas o técnicas convencionales. Este modelo de computación se calcula sobre una supercomputadura o más actualmente sobre un cluster de ordenadores o en un grid. La gran diferencia entre un supercomputador y un grid radica en que mientras un grid está formado por ordenadores independientes, un supercomputador dispone de una "*virtualización*" que le hace comportarse como una sóla máquina en cantidad de memoria y disco. Los grandes cálculos que requieran en exceso cantidad de memoria (64Gb) no podrían ser ejecutados en un grid pues ninguna máquina dispone de tanta memoria. A grandes rasgos un Grid es un Cluster de Clusters.

No sólo es necesario un supercomputador o un cluster de ordenadores para resolver un problema de computación extrema, tiene la misma importancia un software que pueda gestionar y utilizar todos los recursos disponibles ejecutándose en el sistema. Es un software especializado en supercomputación o supercomputación distribuida que permite administrar colas de procesos o tareas a ejecutar cuando alguno de los procesadores esté libre. Estos procesos son el resultado de paralelizar los problemas para convertirlos en tratables y deben tener la capacidad de reconstruir una solución conjunta con todas las subsoluciones obtenidas.

Existen entre otros dos paquetes de utilidades que entran dentro del conjunto de supercomputación con software "*Open Source*": **Condor** [1]y **Sun Grid Engine**[2].

[1] http://www.cs.wisc.edu/condor/
[2] http://www.sun.com/software/sge/

2.2.1. Condor

Condor Project es un entorno de trabajo especializado en gestión de cargas de trabajo para procesos de cómputo intensivo. Dispone de utilidades para la gestión de recursos, prioridades, monitorización, etc.

Está desarrollado en su mayor parte en lenguaje C y C++ y dispone de más de 680000 líneas de código programadas por el equipo de Condor. Si se incluyen las aportaciones externas puede llegar a los 9 millones de líneas de código. Comparaciones interesantes:

- Servidor Web Apache ~60000 líneas
- Pila TCP/IP en Linux ~80000 líneas
- Kernel v2.6.0 de Linux ~5'2 millones de líneas
- Windows XP (completo) ~40 millones de líneas

La solución propuesta por el equipo de Condor no requiere un sistema de ficheros compartidos y permite una fácil migración para diferentes tipos de trabajos en serie. Dispone de mecanismos para tolerancia a fallos como paradas del sistema, sobrecargas en la red y otros desastres derivados de tipo software.

Para abastecer de una mayor flexibilidad y un mejor acercamiento a la modificación y ampliación del software, Condor ofrece una serie de API's de desarrollo para Servicios Web SOAP[3], DRMAA[4] (C), empaquetados Perl, GAHP[5] y herramientas en línea de comandos entre otras utilidades.

La ventaja de Condor no sólo radica en su flexibilidad sino que dispone de todo un abanico de plataformas sobre las que se puede desplegar: Linux i386/IA64, Windows 2K/XP, MacOS, Solaris, TRIX, HP-UX, Compaq Tru64 y otros entornos.

En Condor existen una serie de definiciones que debemos conocer antes de ejecutar alguna tarea sobre el sistema: Trabajo y Máquina. En el Trabajo (job) se definen las preferencias que requiere el proceso a ejecutar, como son la plataforma (Linux/x86 por ejemplo), cantidad de memoria requerida, dirección de la cola, etc… Una Máquina es un punto donde poder ejecutar procesos y como los trabajos, dispone de unos requerimientos y preferencias: Ejecutar sólo procesos de un cierto "universo", ejecutar trabajos que no requieran teclado, etc…

Una vez hay máquinas libres y trabajos en la cola, el arbitrador de Condor hace coincidir Trabajos con Máquinas compatibles en requerimientos y preferencias. Llevar a cabo este emparejamiento requiere algo más que las preferencias de un Trabajo o una Máquina, para ello hay que enviar el trabajo rodeado de cierta información:

[3] *Simple Object Access Protocol*
[4] Distributed Resource Management Application API
[5] Grid ASCII Helper Protocol

- El Universo será el entorno de ejecución y dependerá de la naturaleza del proceso o tarea a ejecutar, los posibles valores son*: Vanilla, Standard, Grid, Java, Parallel y VM*. El universo *Vanilla* está dedicado a los procesos en "serie". Dispone de todos los mecanismos de comunicaciones entre memoria y almacenamiento de las máquinas afectadas*. MPI o Parallel* es aquel universo dedicado a los procesos paralelos, estos procesos deben estar específicamente diseñados para ser ejecutados en este entorno. Standard es el nombre del universo que permite parar y retomar un proceso en cualquier instante (necesario para procesos que demoran mucho tiempo en ejecución, días o meses). El universo *Java* se utiliza para aplicaciones Java de cálculo extremo (pasando el fichero JAR como parámetro) y *VM* es exclusivo para ejecución de máquinas virtuales.
- El proceso o tarea debe poder ser ejecutable en segundo plano, no requerir entrada interactiva ni tener interfaz gráfica (GUI[6]), es decir preparada para la ejecución por lotes (batch).

Esta información se envía a través de un fichero denominado *"Submit Description File"* definiendo todos los datos necesarios para la correcta ejecución de la tarea:

```
#Ejemplo de fichero condor_submit
Universe     = Vanilla
Executable   = exec
Output       = output.out
Queue
```

Ilustración 12 - Condor Submit Description File

Al enviar el fichero condor_submit al sistema se crea un "***ClassAD***" que es la representación de datos interna de Condor, definiendo un objeto con sus atributos. Puede contener detalles sobre cómo debería ser la ejecución del proceso, requerimientos y preferencias. Para un condor_submit como el anterior podría generar un ClassAD como:

```
MyType       = "Job"
TargetType   = "Machine"
ClusterID    = 1377
Owner        = "user"
Cmd          = "exec.exe"
Requirements = (ARCH == "INTEL")
     && (OpSys == "LINUX")
     && (Disk >=DiskUsage)
     && ((Memory*1024)>=ImageSize)
```

Ilustración 13 - Condor ClassAD

[6] Graphic User Interface

La información sobre el estado de la tarea es una característica que toma gran importancia en los entornos Condor, y ofrece diferentes mecanismos para notificar cualquier cambio en el estado que pudiera alterar una ejecución o simplemente mostrar una "fotografía" de la situación actual del proceso. Se recogen en ficheros de salida, de error, registros (logs) y notificaciones por correo electrónico.

Entradas en el registro de Condor:

```
000   (0001.000.000)   05/25   19:10:03   Job   submitted   from   host:
<128.105.146.14:1816>
...
001   (0001.000.000)   05/25   19:12:17   Job   executing   on   host:
<128.105.146.14:1026>
...
005 (0001.000.000) 05/25 19:13:06 Job terminated.
   (1) Normal termination (return value 0)
...
```

Ilustración 14 - Extracto del registro de Condor

Todos estos ficheros de registros y de salida deben estar definidos antes de enviar el proceso a ejecución, por lo que han de ser fijados en el fichero condor_submit del siguiente modo:

```
Universe          = vanilla
Executable        = /my_job.condor
Log               = my_job.log
Input             = my_job.in
Output            = my_job.out
Error             = my_job.err
Arguments         = -a1 -a2
InitialDir        = /condor/run
Queue
```

Ilustración 15 - Ejemplo de Condor Submit Description File

2.2.2. Sun Grid Engine (SGE)

Sun Grid Engine es la solución de computación distribuida de alto rendimiento propuesta por la compañía *Sun*. El proyecto Grid Engine forma una comunidad que enfoca todo su esfuerzo en la adopción de soluciones para el cómputo distribuido. Este paquete de software está ampliamente documentado y estructurado para mayor facilidad de aprendizaje de la plataforma. El manual está dividido en cuatro secciones que separan los diferentes niveles de uso: Manual de usuario y administración, Interfaces de Programación, Documentación Técnica y Documentación para el Desarrollador.

Sun ha definido una serie de roles diferenciados para cada máquina que pertenezca a la red de computación. Estos roles concretan las funcionalidades que están asignadas. Existen cuatro roles que son: *Nodo de Envío* (nodo autorizado a enviar trabajos y obtener información sobre su estado), *Nodo de Ejecución* (Nodo con permisos de ejecución de trabajos), *Nodo de Administración* (para tareas administrativas) y *Nodo Master* (Nodo que controla toda la actividad del Sun Grid Engine, información de estado, etc…). Es posible que una misma máquina desempeñe varios roles diferentes sin llegar a ser un conflicto.

Del mismo modo que en otros sistemas de cálculo sobre redes de ordenadores los trabajos a procesar son enviados junto una información que describe las necesidades del proceso y las características de las máquinas para su correcta ejecución:

Como ejemplo de scripts de ejecución para SGE tenemos:

Ejemplo de Trabajo Serie:

```
#/bin/sh
#$ -o $HOME/directorio/ejecutable.out
#$ -N miTrabajo
#$ -M usuario@dominio.es
./etc/profile.sge
cd programas
./myprog
```

Ilustración 16 - Script de ejecución SGE serie

Ejemplo de Trabajo Paralelo:

```
#/bin/sh
#$ -o $HOME/directorio/ejecutable.out
#$ -N miTrabajo
#$ -pe mpi 4-10
#$ -M usuario@dominio.es
./etc/profile.sge
./etc/mpi.setup –e mpi
cd programas
Mpirun –np $NSLOTS ./myprog
```

Ilustración 17 - Script de ejecución SGE paralelo

Las aplicaciones de gestión que ofrece esta herramienta son las típicas de un sistema distribuido basado en colas:

- Obtener el estado de la cola (*qstat*).
- Obtener los anfitriones de ejecución y su configuración (*qhost*).
- Envío de trabajos y scripts a la cola (*qsub*).
- Ejecución de un comando sobre el primer anfitrión libre (*qrsh*).

Con los comandos anteriores podemos obtener una referencia rápida con las acciones más comunes en la administración del sistema:

- Añadir y eliminar privilegios administrativos de un anfitrión:
  - *qconf –ah #* (concede privilegios administrativos al host #)
  - *qconf –df #* (retira privilegios administrativos al host #)
- Crear un nuevo anfitrión de ejecución:
  - *qconf –ah <nombre>* (crea un nuevo host de administración, después es necesario ejecutar "*install_execd*" en el nuevo host).
- Eliminar un nodo de ejecución:
  - Primero es necesario eliminar las colas:
    - *qconf –dq <nombre de las colas>*
  - Eliminamos el host:
    - *qconf –de <nombre del host>*
  - Eliminamos la configuración
    - *qconf –dconf <nombre del host>*
- Crear un nuevo anfitrión emisor de trabajos:
  - *qconf –as <nombre>* (host que podrá emitir trabajos)
  - *qconf –ds <nombre>* (host que no podrá emitir trabajos)
- Mostrar anfitriones de Ejecución/Administrativos/Emisores de trabajos:
  - *qconf –sh* (muestra hosts administrativos)
  - *qconf –ss* (muestra hosts de emisión de trabajos)
  - *qconf –sel* (muestra host de ejecución)
- Manipulación de colas:
  - *qconf –aq <nombre de cola>* (crear nueva cola)
  - *qconf –dq <nombe de cola>* (eliminar una cola)
  - *qconf –mq <nombre de cola>* (modificar una cola)

Como en todos los sistemas de computación distribuida y super-computación encontramos una serie de dificultades que podrían retrasar e incluso abortar las tareas que estemos procesando en el sistema.

2.2.3. Nauty

Nauty[7]es un proyecto desarrollado bajo licencia GNU de fuentes abiertas por Brendan D. McKay[8] del Departamento de Ciencias de la Computación de la Universidad Nacional de Australia. Es un conjunto de procedimientos y herramientas para determinar el grupo de automorfismos de un grafo. Estas herramientas están disponibles en un paquete denominado *gtools* y extienden la funcionalidad de Nauty hacia el procesamiento eficiente de grafos almacenados en ficheros con formato *graph6* o *sparse6* descritos en los siguientes apartados. Aunque el programa Nauty está desarrollado en un subconjunto portable del lenguaje C y puede ser compilado en numerosos sistemas, el paquete *gtools* requiere un entorno Unix, pero puede ser ejecutado en otros sistemas que utilicen el sistema de llamada Unix para funciones con excepción de la utilidad *shortg* que necesita un programa compatible con el comando *sort* de Unix y tuberías para su correcto funcionamiento.

Nauty se distribuye el código fuente, y facilita un conjunto de ficheros de configuración y ayuda para su compilación. Esta compilación se realiza del modo habitual en entornos Unix con los comandos:

```
./configure
make & make install
```

Todo el paquete de utilidades y el propio nauty se generarán con la secuencia de comandos anterior si no hay problemas de dependencias o con el compilador. Cada una de las herramientas está autodocumentada y es posible acceder a la ayuda y funcionalidades con la opción "*-help*" tras el nombre del ejecutable. Las funciones básicas del paquete *gtools* son:

- ***geng***: Generación de grafos pequeños, dispone de una serie de opciones para determinar el número de vértices, restricciones en el grafo (grafos conectados, grafos biconectados, grafos libres de triángulos, libres de C_4…) además de características para modificar el formato de salida y el rendimiento a expensas del uso de menor cantidad de memoria. Más información sobre rendimiento en el apartado Apéndice A: Estadísticas de uso de geng..
- ***geng***: Generación de grafos pequeños bicoloreados. Utiliza la misma filosofía que la utilidad geng para generar los grafos con la restricción que deben ser bicoloreables.
- ***directg***: genera dígrafos (grafos dirigidos).
- ***Multig***: genera multígrafos.
- ***genrang***: crea grafos aleatorios con un número de vértices concreto.
- ***copyg***: convierte el formato de un grafo y selecciona un subconjunto.
  lLabelg: etiquetación canónica de grafos
- ***shortg***: elimina isomorfismos de un fichero de grafos

[7] Nauty (No AUTomorphisms, Yes?) http://cs.anu.edu.au/~bdm/nauty/
[8] bdm@cs.anu.edu.au - http://cs.anu.edu.au/~bdm/index.html

- ***listg***: muestra por pantalla y en diferentes formatos el/los grafos/s pasados por parámetros a esta función.
- ***showg***: versión de listg para línea de comandos.
- ***amtog***: función para leer grafos en forma de matriz de adyacencia.
- ***dretog***: función para leer grafos en formato dreadnaut (Dreadnaut es la denominación del contenedor que se utiliza para ejecutar Nauty).
- ***complg***: complementa un grafo.
- ***catg***: concatena ficheros de grafos.
- ***addedgeg***: añade una arista a un grafo.
- ***deletedgeg***: elimina una arista de un grafo.
- ***countg***: cuenta grafos de acuerdo a un conjunto de propiedades.
  pickg: selecciona grafos de acuerdo a un conjunto de propiedades.

Estas herramientas y otras funcionalidades se encuentran en el paquete de utilidades *gtools* de nauty.

2.2.4.- Otras Alternativas Software de generación de grafos

El programa **Grafos** es un software para la construcción, edición y análisis de grafos. Orientado principalmente a la docencia y aprendizaje de la Teoría de Grafos. Sin embargo el programa *Grafos* puede ser utilizado para numerosas aplicaciones que requieran el cálculo de grafos como puede ser ingeniería de la organización industrial, logística y transporte, invertigación operativa, diseño de redes, etc… Se distribuye bajo licencia CC (*Creative Commons Licence).* Para utilizar Grafos es necesario un sistema operativo basado en Windows (98, NT, Me, 2000, XP, Vista) con el Framework Microsoft .NET. Este Software no está pensado para su utilización en redes de computación o supercomputadores.

GraphThing es una utilidad que permite crear, manipular y estudiar grafos. Es una herramienta gratuita con licencia *GNU GPL*. Su última versión data del día 22 de Diciembre del año 2006 y hasta la fecha no se conocen nuevas actualizaciones. Soporta grafos simples, dígrafos y la creación rápida de grafos comunes (completos, ciclos, nulos, estrellas, etc..), además de poder aplicar algunos algoritmos sobre ellos: camino más corto, conectividad, árbol de recubrimiento, obtención de la matriz de adyacencia, etc… A pesar de ofrecer el código fuente y disponer de una licencia gratuita no dispone actualmente de un diseño inmediato para desplegarlo fácilmente sobre redes de computación.

La aplicación ***aiSee*** de la compañía *AbsInt Angewandte Informatik*[9] permite visualizar grafos en un formato de descripción denominado GDL (*Graph Description Language*). Tiene suficiente potencia para mostrar grafos con miles de vértices y aristas. Dispone de 15 algoritmos diferentes de visualización, posibilidad de animación además de la exportación a formatos gráficos para su impresión.

[9] http://www.absint.com/

2.3.- Formatos de Fichero para Grafos

2.3.1. Formato Graph6

Graph6 es un formato para representación y almacenamiento de grafos ideado por Brendan McKay. Utiliza únicamente los caracteres ASCII imprimibles y tiene la peculiaridad de codificar la matriz de adyacencia en los últimos 6 bits de cada byte. Gracias a la característica imprimible del formato graph6 podemos mostrar por pantalla cada grafo, como por ejemplo grafos de 20 vértices:

```
SsP@PGWD?C?G?G?E?@??H?@??Ag?C_?AS
SsP@PGWC_G?G?G?E?@??H?@??Ag?C_?AS
SsP@PGWC?G_Q?O?G?@??G??g?AO?BG?Ao
SsP@PGWC?G?R?O?G?A??O??g?AO?BG?Ao
SsP@P?WC_I?_?O?I?A??K?@??Ag?AO?@c
SsP@P?OC?O?c@C?_?K?A_?B??I??BG?Ao
```

Ilustración 18 - Ejemplo de grafos de 20 vértices en formato graph6

Este formato está limitado a grafos no dirigidos siendo viable su utilización en grafos pequeños o grafos densos hasta 2^{18}-1 vértices.

Descripción del formato:

En un fichero con grafos en formato graph6 hay únicamente un objeto por línea. Todos los caracteres de la línea (excepto el carácter "fin de línea") todos tienen un valor comprendido entre 63 y 126 (que son los caracteres imprimibles ASCII). Por lo que un fichero de grafos en formato graph6 es un fichero de texto con un grafo por línea.

Para generar un objeto utilizable por graph6 (carácter ASCII imprimible) debemos seguir los siguientes pasos para un vector de bits ("x") de longitud finita ("k") como por ejemplo:

```
1000101100011100
```

1) Hacemos que la longitud de la cadena sea múltiplo de 6 añadiendo tantos 0's a la derecha como sea necesario: 1000101100011100**00**
2) Creamos agrupaciones de 6 bits con el vector: 100010 110001 110000
3) Añadimos 63 a cada grupo considerándolos números binarios bigendian: 97 112 111

Cada valor es almacenado en un byte, por lo que el número de bytes necesarios para una longitud k será k/6.

Por las restricciones propias del formato sólo podrán ser codificados aquellos grafos no dirigidos que tengan un orden comprendido en 0-68719476735.

El fichero de texto dispone de extensión .g6 y con una cabecera (opcional):

```
>>graph6<<
```

Supongamos un grafo G con n vértices, escribiremos el triángulo superior de la matriz de adyacencia de G como un vector de bits de longitud n(n-1)/2, utilizando la ordenación (0,1), (0,2), (1,2), (0,3), (1,3), (2,3), … , (n-1, n).

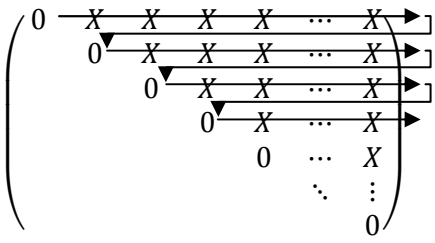


Ilustración 19 - Orden de los bits en la matriz de adyacencia

Cada una de las posiciones del triángulo superior de la matriz de adyacencia se modela como un bit en el vector de n(n-1)/2 posiciones, siendo 1 en caso de adyacencia y 0 en otro caso. Por ejemplo, en un grafo de 7 vértices la matriz de adyacencia:

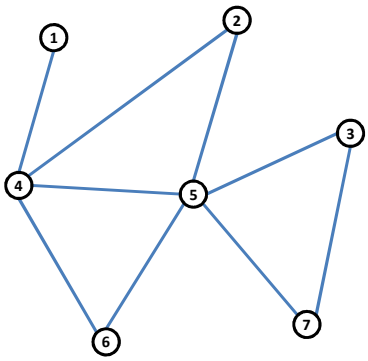


Ilustración 20 - Grafo de 7 vértices

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ & 0 & 0 & 1 & 1 & 0 & 0 \\ & & 0 & 0 & 1 & 0 & 1 \\ & & & 0 & 1 & 1 & 1 \\ & & & & 0 & 1 & 1 \\ & & & & & 0 & 0 \\ & & & & & & 0 \end{pmatrix}$$

De la matriz de adyacencia obtenemos el vector x con los bits en el orden indicado anteriormente:

x: 001000 01100 0101 111 11 0

1) Añadir tantos 0's como sea necesario para hacer la longitud múltiplo de 6:

001000 01100 0101 111 11 0 **000**

2) Creamos grupos de 6 bits

001000 – 011000 – 101111 – 110000 = 8 – 24 – 47 - 48

3) Sumamos 63 a cada grupo

71 – 87 – 110 – 111

Cada grupo se codifica en un byte. Para completar el formato graph6, el vector de bytes obtenido en el paso anterior debe ir precedido del número de vértices del grafo utilizando el mismo algoritmo para convertirlo en carácter ASCII imprimible, esta función será **R(x)** = n+63 siendo n el número de vértices del vector x:

R(x) =7 vértices + 63 = **70**

Por lo que obtenemos que el grafo anterior se codifica en graph6 como:

70 – 71 – 87 – 110 – 111 = **FGWno**

Una versión optimizada de graph6 es el formato sparse6.

NOTA: si el número de vértices es mayor a 64 se utilizan 4 bytes para su codificación, el primer byte será 126 y los otros tres se obtienen de crear grupos de 6 bits y almacenarlos en bytes (8bits) con codificación imprimible del resultado (+63).

Podemos deducir el algoritmo de la función que llamaremos **N(n)**:

- Si $0 \leq n \leq 63$ definimos N(n) como el byte resultado n+63. N(n) = R(x)
- Si $n > 63$, definimos N(n) como el conjunto de 4 bytes “126 R(x)” donde x es la forma binaria (bigendian) de 18 bits del vector n.

Por ejemplo

- N(30) = 30 + 63 = 93
- N(12345) = N(000011 000000 111001) = 126 (3+63) (0+63) (57+63) = 126 66 63 120

2.3.2. Formato Sparse6

Esta ampliación del formato *graph6* permite representar grafos de orden 0-262143, admite grafos con bules y múltiples aristas pero no es posible representar aristas dirigidas con este formato.

Estructura general: cada grafo ocupa una línea de texto y cada byte de la cadena tiene la forma 63+x donde 0<=x<=63 codificados en bloques de 8 bits del mismo modo que en su formato predecesor graph6.

El grafo codificado consiste en:

- El carácter ":" que indica el uso del formato Sparse6.
- El número de vértices.
- Una lista de aristas.
- Fin de línea.

1) El número de vértices se calcula del mismo modo que con la función N(n) de *graph6*.
2) La lista de aristas:

Sea *k* el número de bits necesarios para representar n-1.

La secuencia de bits:

b[0]x[0] b[1]x[1] b[2]x[2] b[3]x[3] … b[m]x[m]

b[i] tiene 1 bit de tamaño, x[i] tiene k bits de tamaño. La secuencia se empaca en grupos de 6 bits con ordenación bigendian y debe ser múltiplo de 6, para ello se añaden los bits necesarios al final del vector con las siguientes condiciones:

- Si (n,k) = (2,1), (4,2), (8,4) o (16,5), y el vértice n-2 tiene una arista pero n-1 no tiene arista y faltan k+1 bits para hacer el tamaño múltiplo de 6 el vector => añadimos un 0 y tantos 1 como hagan falta para hacer el tamaño múltiplo de 6.
- En otro caso añadir tantos 1's al final como sea necesario para hacer el tamaño múltiplo de 6.

Esta regla se aplica para ser compatible con el formato del paquete de utilidades *gtools* y para evitar un bucle extra del programa en algunos casos concretos y poco usuales.

Separamos el tren de bits en bloques de 6 y almacenamos cada uno de los bloques en un byte (8bits).

Si n es el número de vértices, los vértices estarán numerados de 0 a n-1, por lo que codificamos las aristas con el siguiente algoritmo:

```
v=0;
for i from 0 to m do
     if b[i] = 1 then
          v = v+1;
     endif
     if x[i] > v then
          v = x[i];
     else
          output{x[i], v};
     endif
```

Ilustración 21 - Algoritmo de codificación de aristas en sparse6

En decodificación, una pareja (b,x) incompleta al final del vector es descartada.

Un ejemplo de utilización del formato *Sparse6*, tenemos el siguiente grafo:

```
:Fa@x^
```

- El carácter ':' indica que está en formato Sparse6
- Restamos 63 a cada byte y los escribimos en binario (sólo los 6 últimos bits, pues los dos primeros serán 0)

000111	100010	000001	111001	011111

El primer bloque no es 63, por lo que es directamente el número de vértices, en este caso el grafo dispone de 7 vértices (n=7). Podemos codificar n-1 = 6 con 3 bits. El siguiente paso es agrupar el resto de bits (todo el vector eliminando el tamaño) en bloques de 1 y 3 bits respectivamente:

1	000	1	000	0	001	1	110	0	101	1	111

Esta es la secuencia b[i]x[i]: 1,0 - 1,0 - 0,1 - 1,6 - 0,5 - 1,7.

La última pareja b/x 1,7 se ignora pues por el formato sabemos que no existe el vértice 7 (para 7 vértices se identifican desde 0 hasta 6). Del resto se obtienen las aristas 0-1, 0-2, 1-2, 5-6. Una matriz de adyacencia:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ & 0 & 1 & 0 & 0 & 0 & 0 \\ & & 0 & 0 & 0 & 0 & 0 \\ & & & 0 & 0 & 1 & 0 \\ & & & & 0 & 0 & 0 \\ & & & & & 0 & 0 \\ & & & & & & 0 \end{pmatrix}$$

3. Definición de Objetivos

3.1. Descripción del problema

El propósito principal de este proyecto es el estudio de las herramientas actuales de Computación Distribuida para el cálculo intensivo de isomorfismo de grafos. Para este estudio se utiliza como problema de cálculo la generación de grafos con ciertas características y detección de isomorfismos sobre ellos haciendo uso de todas las técnicas y herramientas disponibles (tanto software como hardware), y en caso de no disponer de alguna herramienta software, realizar una implementación de la misma para compensar la necesidad.

Un objetivo importante es el aprendizaje. Este desarrollo no se centra únicamente en el problema a resolver en este proyecto, sino que implica plasmar el conocimiento adquirido en el transcurso de la carrera en las diferentes asignaturas. En este proyecto se han aplicado conocimientos de Matemática Discreta, Teoría de Grafos, Programación, Redes, Arquitectura de Sistemas, Ingeniería del Software, Sistemas Distribuidos, Sistemas Operativos entre otras. Se persigue conocer más a fondo la utilidad y técnicas para el cálculo distribuido, arquitecturas de supercomputación, redes y comunicaciones.

En este proyecto, por tanto, se expondrá mediante la puesta en práctica del mismo, el proceso de generación y cálculo de grafos sobre redes de supercomputación de tal manera que pueda servir como referencia para futuros desarrollos o como manual rápido para las herramientas utilizadas.

3.2 Objetivos secundarios

El problema de cálculo utilizado para este proyecto puede ayudar a encontrar la demostración más corta del teorema de Köchem-Specker de Física Cuántica. El teorema se refiere al número de variables independientes que puede haber en un experimento y la conjetura de Peres (que pretende demostrar) afirma que el número mínimo es 18.

Debido a las iteraciones entre variables, el problema puede reducirse a probar que no existe ningún grafo simple con menos de 18 vértices que cumpla unas determinadas características.

Trabaja en varias dimensiones y, en cada dimensión “d” buscan los grafos:

- Cuyos vértices tienen todos valencia mayor que “d”
- No contiene como subgrafo a ningún grafo que sea K_{d+1}-(2 aristas disjuntas)
- Son coloreables por vértices en verde y rojo de forma que cada K_d contenga al menos un vértice verde, pero no haya dos vértices verdes adyacentes.

Un problema con estas características parece adecuado para el estudio de redes de Supercomputación debido a la gran complejidad del cálculo necesario, siendo poco viable el llevarlo a cabo en un sistema de computación tradicional.

4.- Análisis de Requisitos, diseño e implementación

Se han codificado los requisitos teniendo en cuenta lo siguiente:

- Deben tener una numeración que permita realizar una ordenación.
- Una ver ordenados los requisitos deben estar seguidos los que estén relacionados.
- Debe permitir la inserción de nuevos requisitos sin que ello implique el cambio de codificación de los requisitos ya existentes.

Se ha utilizado el criterio RQ-XYZ siendo XYZ el orden del requisito en árbol. Apareciendo sólo aquellas etiquetas que lo identifiquen: XYZ indica requisito X, subrequisito de primer nivel Y, subrequisito de segundo nivel Z y así sucesivamente.

4.1.- Diccionario de Requisitos

El diccionario de requisitos es un simple resumen nominal de requisitos en el que se relaciona el código con el nombre del requisito.

RQ-001 *Sistema Unix/Software Libre*
RQ-010 *Redes de Ordenadores, Clusters y software de Control de procesos.*
RQ-011 *Arquitectura.*
RQ-012 *Condor.*
RQ-013 *SGE (Sun Grid Engine).*
RQ-020 *Software de generación de grafos Nauty.*
RQ-021 *Adaptar problema al cálculo distribuido con Nauty.*
RQ-030 *Búsqueda de alternativas para generación de grafos.*
RQ-031 *Implementación.*
RQ-040 *Aproximar el tamaño mínimo del problema.*

Ilustración 22 - Diccionario de Requisitos

4.2.- Requisitos

Código:	**RQ-001**
Nombre:	*Sistema Unix/Software Libre.*
Versión:	00
Categoría:	Técnico
Descripción:	Acercamiento al manejo de sistemas Linux/Unix a nivel de usuario avanzado. Como usuario avanzado entendemos a un usuario familiarizado con la configuración del sistema operativo (nivel alto) además del uso básico. Acercamiento a la programación en sistemas Linux/Unix a nivel de programación de comunicación máquina-máquina, cálculo de números grandes (mayores de 32-64 bits), tanto en C/C++ como en Java.

Código:	**RQ-010**
Nombre:	*Redes de Ordenadores, Clusters y software de Control de procesos.*
Versión:	00
Categoría:	Técnico
Descripción:	Estudio de las arquitecturas de las redes de computación, clusters y los gestores que hacen posible la comunicación entre ellos para realizar el cálculo distribuido.

Código:	**RQ-011**
Nombre:	*Arquitectura*
Versión:	00
Categoría:	Técnico
Descripción:	Es necesario estudiar las arquitecturas típicas de las redes de computación, su funcionamiento y estructura como sistema de comunicaciones. Estudio de las topologías típicas en computación distribuida.

Código:	**RQ-012**
Nombre:	*Condor*
Versión:	00
Categoría:	Técnico
Descripción:	Condor es un software de gestión de procesos basado en colas. Estudiar el funcionamiento de Condor y sus aplicaciones en la computación distribuida y más concretamente en la resolución del problema del Isomorfismo de Grafos así como en la prueba del teorema de Köchem-Specker y la conjetura de Peres.

Código:	**RQ-013**
Nombre:	*SGE (Sun Grid Engine)*
Versión:	00
Categoría:	Técnico
Descripción:	SGE es el software de gestión de procesos basado en colas de Sun Microsystems. Estudiar el funcionamiento de SGE y sus aplicaciones en la computación distribuida y más concretamente en la resolución del problema del Isomorfismo de Grafos así como en la prueba del teorema de Köchem-Specker y la conjetura de Peres.

Código:	**RQ-020**
Nombre:	*Software de generación de grafos Nauty*
Versión:	00
Categoría:	Técnico
Descripción:	Nauty es el acrónimo de "**N**ot **AUT**omorphisms, **Y**es?". Es un software de generación de grafos. Estudiar los formatos de archivos más utilizados en el almacenamiento de grafos (y compatibles

Código:	**RQ-021**
Nombre:	*Adaptar problema al cálculo distribuido con Nauty*
Versión:	00
Categoría:	Funcional

Descripción:	Estudiar el uso de Nauty en una red de computación y ampliar los métodos de detección de grafos de Nauty para cumplir los requisitos de la prueba del teorema de Köchem-Specker y la conjetura de Peres.

Código:	**RQ-030**
Nombre:	*Búsqueda de alternativas para generación de grafos*
Versión:	00
Categoría:	Funcional
Descripción:	Nauty no es el único paquete de utilidades para la generación de grafos. Buscar otras alternativas para la generación de grafos y estudiar su adaptación a redes de computación y su posible adaptación a la resolución del problema seleccionado para las pruebas.

Código:	**RQ-031**
Nombre:	*Implementación*
Versión:	00
Categoría:	Funcional
Descripción:	Estudiar una resolución personalizada de generación de grafos para el problema propuesto, compatible con redes de computación y más concretamente con el software de gestión de procesos basado en colas descrito en el proyecto.

Código:	**RQ-040**
Nombre:	*Aproximar el tamaño mínimo del problema*
Versión:	00
Categoría:	Funcional
Descripción:	El uso de redes de computación no es siempre la mejor solución para el cálculo. El uso de un cluster o un GRID de computación será rentable en cuanto al tiempo siempre que el cálculo pueda ser realizado en menor tiempo que un único equipo teniendo en cuenta los requisitos de cálculo y las colas de procesos del cluster, por ello es necesario calcular el tamaño mínimo del problema que hará rentable el uso de una red de computación.

5.- Análisis temporal y costes de desarrollo

- *Investigación y aprendizaje*: es una tarea que se llevará a cabo a lo largo de todo el proyecto. Es la recopilación de la información de los aspectos teóricos de las redes de supercomputación (tipos de redes, clasificación, redes más utilizadas, protocolos de comunicaciones, etc…), aspectos teóricos de la algoritmia y generación de grafos (programas de generación, algoritmos de búsquedas de subgrafos, de isomorfismos, coloreado, etc…), aplicación del teorema de Köchem-Specker y la conjetura de Peres, etc…

- Definición del problema, planteamiento de soluciones, estudio de antecedentes y alternativas de los diferentes programas existentes y esbozo del diseño del software de generación de grafos propio. Esta fase del proyecto se centró en perfilar el problema de la comunicación entre ordenadores desde un punto de vista práctico. Se llevaron a cabo pruebas sobre la red de supercomputación del Centro Informático Científico de Andalucía[10] (CICA) con los gestores de procesos Condor 6.8, Sun Grid Engine (SGE) y el programa Nauty.

- Ejecución de los algoritmos de generación y detección sobre las redes de supercomputación. El coste temporal de ejecución de nauty sobre una red de ordenadores es considerablemente menor que sobre un computador simple. Otro punto importante de este apartado es la implementación de un generador de grafos con ciertas características (útiles para la resolución del problema propuesto). El fichero de salida de este generador será la fuente de datos para que nauty discrimine los isografos.

[10] http://www.cica.es/

- Estudio de los resultados. Debido a los requerimientos de computación y a la complejidad del problema los resultados más sencillos tardan del orden de días y pueden requerir varios Terabytes[11] de almacenamiento. El tiempo de procesado (cuenta y exclusión de grafos con las características del problema) de los grafos generados es insignificante respecto al de generación y detección de isomorfismos.

- Documentación oficial.

Debido al carácter de investigación de este proyecto las fases están altamente entrelazadas, siendo difícil definir un periodo para cada una de ellas en la duración del proyecto, que ha sido desarrollado durante los 9 meses desde la adjudicación del proyecto este curso. La etapa de mayor duración es la de ejecución de algoritmos puesto que cada prueba sobre un cluster (concretamente el cluster CICA) podría tomar desde pocas horas hasta semanas o meses y prestando gran atención a la aparición de errores o problemas durante el procesado. Estos problemas han sido principalmente de espacio de almacenamiento principalmente o de memoria disponible en el menor de los casos e interrumpían o alteraban el procesado de los grafos. Mientras se realizaban las pruebas se procedía a la revisión de algoritmos, investigación sobre nuevas vías de ejecución con otros generadores, implementando las extensiones de nauty y el generador personalizado. La última etapa del proyecto ha sido plasmar la información que ha sido recopilada a lo largo del proyecto en esta documentación con un mes de duración.

[11] Unidad de medida de almacenamiento, símbolo **TB** y equivale a 1024 GB.

6.- Conclusiones y desarrollos futuros

Se han probado en el cluster del CICA los algoritmos de generación con *geng* y detección con los comandos *w4f, w5f y w6f* (detectores de ruedas de 4, 5 y 6 vértices).

Para que el uso de una red de supercomputación tenga sentido es preciso definir un tamaño mínimo del problema que compense la ejecución sobre más de un ordenador. La carga de trabajo de una red de supercomputación eleva el tamaño mínimo debido a los grandes tiempos de espera en las colas de procesos. Si un proceso tarda en ejecutarse 6 horas en un ordenador doméstico, pero en la red hay una espera media de 6 horas no resulta asequible el esfuerzo de paralelizar los algoritmos.

Podemos observar cómo en el cluster CICA donde se han realizado las pruebas ha aumentado considerablemente el uso de CPU a lo largo del tiempo aumentando el tamaño mínimo del problema. Para acercarnos a la resolución del objetivo secundario es necesario ejecutar el algoritmo de detección de ruedas sobre grafos de hasta 18 vértices. Un único ordenador Pentium III a 550Mhz tardaría 10 años en calcular todos los grafos posibles (sin restricciones) de 13 vértices mientras que una red con 64 ordenadores de la misma potencia tardarían aproximadamente 3 meses.

Utilizando un algoritmo personalizado de detección (PRUNE en Nauty) la detección de C_4 en grafos de 18 vértices alcanzaría los 62 años sobre un único nodo de computación. Puede ser reducido a 16 años en un monoprocesador actual, 2200Mhz, y a 3 meses calculando sobre 64 ordenadores a tiempo completo de CPU o a 3 semanas utilizando el cluster del CICA con todos los nodos trabajando al máximo para este único proceso (el cluster CICA dispone actualmente de 256 nodos).

Sin embargo la detección de C_4 no es el problema a resolver sino la detección de ruedas en grafos de 13 a 18 vértices que tiene un gasto considerablemente mayor de cálculo. por lo que podemos llegar a la conclusión que en la actualidad no es viable la generación con detección de ruedas en grafos con hasta 18 vértices.

El segundo gran inconveniente es el espacio de almacenamiento necesario para la tarea, aunque es posible calcular a lo largo del tiempo (el que sea necesario) el problema para 18 vértices, el espacio requerido es extremo. El almacenamiento previo para procesado de los resultados al generar los grafos de 15 vértices llegó a ocupar todo el espacio de almacenamiento disponible para procesos en el CICA, aproximadamente 1TB de información sin poder generar todo el conjunto de grafos posible.

La media de carga del cluster[12] de procesado del CICA, que indica una referencial del uso en la actualidad, se puede observar en el siguiente diagrama:

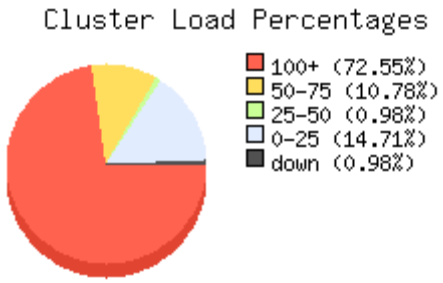


Ilustración 23 - Carga media del cluster CICA

Es posible mejorar los tiempos de cálculo si se realiza una generación de grafos de acuerdo a unas características en lugar de comprobar si cada grafo generado cumple las características. Si generamos todos los grafos posibles con un K_n en una posición concreta del grafo, podemos asegurar que si generamos todos los grafos posibles con un K_m en una posición diferente del grafo (n=m) ambos grupos serán isomorfos. En el apartado 7.- Generación de grafos con un subgrafo completo K_n se encuentra un algoritmo para la generación de grafos con ciertas características y que puede obtener una salida compatible con el formato graph6 o sparse6 utilizado por nauty.

Otra mejora para el cálculo del problema sería no almacenar los resultados, el proceso de generación almacena en disco los resultados para que sean el fichero de entada para otros procesos (exclusión, cuenta, coloreado…). Si se conoce el siguiente ejecutable o se tiene el algoritmo es posible modificar el programa para que no sea necesario el almacenamiento: el grafo generado es pasado como parámetro de entrada directamente al siguiente algoritmo. Soluciona el problema del almacenamiento, pero haría necesario calcular la pérdida de tiempo en: trasladar el grafo entre funciones, llamada al nuevo procedimiento, ejecutar y retornar un resultado, para cada uno de los grafos generados.

[12] Fuente: http://cube.cica.es/

7.- Generación de grafos con un subgrafo completo K_n

Generación de un grafo no dirigido con formato compatible con nauty que contenga un subgrafo completo K_n.

Matriz de Adyacencia de Grafo Completo K_5:

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Ilustración 24 - Matriz de adyacencia de un grafo K_5

Insertando el Grafo Completo K_5 en un grafo con mayor número de vértices:

$$\begin{pmatrix} K_5 & \cdots & X_{(0,n-3)} & X_{(0,n-2)} & X_{(0,n-1)} & X_{(0,n)} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ X_{(0,n-3)} & \cdots & 0 & X_{(n-3,n-2)} & X_{(n-3,n-1)} & X_{(n-3,n)} \\ X_{(0,n-2)} & \cdots & X_{(n-3,n-2)} & 0 & X_{(n-2,n-1)} & X_{(n-2,n)} \\ X_{(0,n-1)} & \cdots & X_{(n-3,n-1)} & X_{(n-2,n-1)} & 0 & X_{(n-1,n)} \\ X_{(0,n)} & \cdots & X_{(n-3,n)} & X_{(n-2,n)} & X_{(n-1,n)} & 0 \end{pmatrix}$$

Ilustración 25 - Grafo K_5 en una Matriz de Adyacencia de orden arbitrario

Sabemos que es una matriz simétrica por lo que sólo tenemos que calcular las posibles combinaciones de uno de los triángulos asegurando que mantenemos fija la figura del Grafo Completo K_5.

El número total de grafos no dirigidos (incluidos los isomorfismos) de 13 vértices a calcular tomarían n·(n-1)/2 posiciones en la matriz de adyacencia:

$$2^{1+2+\cdots+(n-1)} = 2^{78} = 302.231.454.903.657.293.676.544$$

Si forzamos en la matriz de adyacencia un Grafo Completo K_n tendremos que restar en la potencia el número de combinaciones que hemos eliminado al establecer el valor previo:

- $K_3 = 3$
- $K_4 = 6$
- $K_5 = 10$

Podemos obtener la fórmula de número de posiciones ocupadas en la matriz de adyacencia por un Grafo Completo K_n como:

$$a_n = a_{n-1} + (n-1)$$

Siendo n el número de posiciones ocupadas en la matriz por el Grafo Completo y con valor inicial n_2=1. Para un grafo de 13 vértices con un único K_5 tendríamos:

$$a = 10 \text{ posiciones.}$$

$$2^{1+2+\cdots+(n-1)-10} = 2^{68} = 295.147.905.179.352.825.856$$

Al reducir el rango de posibilidades de generación a aquellos grafos que contengan al menos un Grafo Completo K_5 obtenemos una aceleración de 2^{10}=1024. Esta aceleración es constante y únicamente depende del tamaño del Grafo Completo:

- Para un K_4 la aceleración es 2^4=16.
- Para un K_6 la aceleración es 2^{15}=32768
- Para un K_8 la aceleración es 2^{28}=268435456

Podemos obtener la fórmula de la aceleración respecto al cálculo de todos los grafos de cualquier tamaño como:

$$\text{aceleración} = 2^{a_n}$$

Siendo a_n el número de posiciones que ocupa un Grafo Completo K_n en la matriz de adyacencia. En el diagrama podemos ver cómo el problema de la generación de grafos se reduce mientras aumenta el tamaño del Grafo Completo K_n.

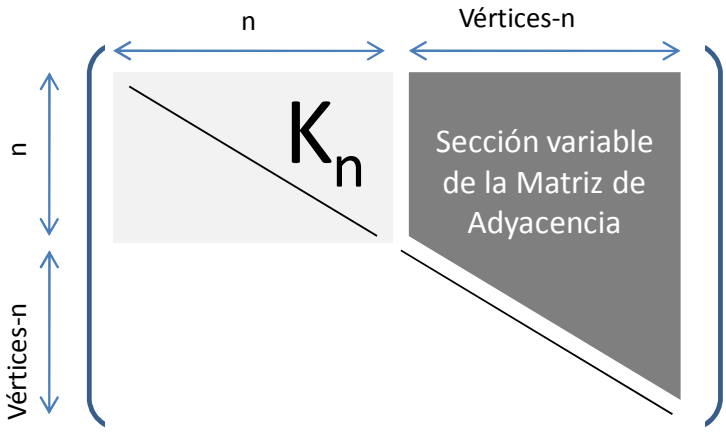


Ilustración 26 - Zona variable de un grafo arbitrario+K_n no dirigido

Es posible reducir aún más el problema si una solución válida incluye dos o más grafos completos **independientes** entre sí:

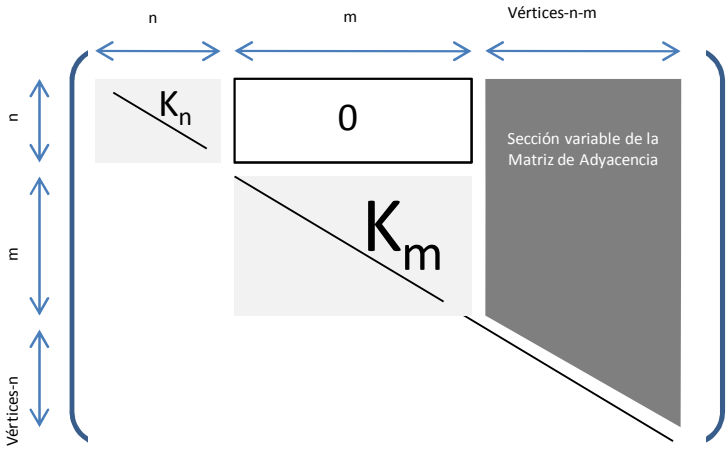


Ilustración 27 - Zona de generación de un grafo arbitrario+K_n+K_m no dirigido

Este método de generación no elimina ni detecta los Grafos Isomorfos del subconjunto que obtiene. Pero gracias a esta característica podemos aislar el bloque de la Matriz de Adyacencia que será modificado y conocer los índices de cada elemento que serán víctimas del generador. Un aspecto importante de la generación es que no es importante la situación del Grafo Completo K_n en la Matriz de Adyacencia, pues cada grafo de los conjuntos generados con dos disposiciones diferentes del Grafo Completo es isomorfo (al menos) con un grafo del conjunto alternativo.

Para 18 vértices tendremos que generar:

$$\frac{18 \cdot 17}{2} = 153 \text{ bits}$$

$$2^{153} = 1{,}14179815416476790484662877555 96 \cdot 10^{46} \text{ grafos}$$

Si generamos los grafos incluyendo un Grafo Completo K_6 tendremos que generar:

$$2^{153-6} = 2^{147} = 1{,}784059615882449851322857461 8119 \cdot 10^{44} \text{ grafos}$$

Proceso de generación con inserción de un Grafo Completo K_n en un grafo de v vértices con v>n.

- Calcular el tamaño en bits del vector a generar: $\frac{v\cdot(v-1)}{2} - n = t$
- Generar todas las posibles combinaciones de t bits (un contador por ejemplo)
  - Para cada vector generado insertamos K_n
  - Formamos agrupaciones de 6 bits y cada una de ellas se convierte a un byte (8bits) por lo que cada byte tendrá un rango de 0x00-0x3F
  - Cada byte hay que incrementarlo 63 (0x3F) para que cumpla el formato del programa nauty. El tamaño de un carácter en C es de 8 bits y sumarle 63 lo convierte en un carácter que puede ser impreso por pantalla.

8.- Extensión de nauty

El problema a resolver por la red de supercomputación requiere la generación de un conjunto de grafos con exclusión de isomorfos. Para ello utilizaremos las posibilidades de generación que ofrece el programa nauty extendiendo convenientemente su funcionalidad.

El conjunto de utilidades gtools de nauty dispone de un generador de grafos denominado geng. Este generador obtiene grafos que cumplen una serie de características cuyos algoritmos vienen preestablecidos en la propia utilidad. El formato del comando geng es:

```
geng [-cCmtfbd#D#] [-uygsnh] [-lvq] [-x#X#] n [mine[:maxe]]
[res/mod] [file]
n     número de vértices
-c    obtiene grafos conectados
-C    obtiene grafos biconectados
-t    genera grafos libres de triángulos
-f    genera grafos libres de 4-ciclos
-b    genera únicamente grafos bipartitos
-m    libera memoria a costa de tiempo
-d#   grado mínimo
-D#   grado máximo
-v    muestra el número de aristas
-l    grafos con etiquetación canónica
-u    no escribir grafos, sólo generar y contar
-g    utilizar formato graph6
-s    utilizar formato sparse6
-y    utilizar formato “y” (en desuso)
-h    escribir cabecera para formatos graph6 y sparse6
-q    suprime salida auxiliar
```

Ilustración 28 - Parámetros de la utilidad geng más comunes

Para una ayuda más completa ejecutar “geng –help”.

Una de las características más importantes de la utilidad geng es que permite extender su funcionalidad permitiendo generar grafos con nuevas características. Esta extensión se realiza programáticamente en lenguaje C y en nauty se denomina como variable PRUNE y PREPRUNE.

La variable de preprocesador PRUNE se define en tiempo de compilación (la definición de PREPRUNE es la misma que la de PRUNE):

```
int PRUNE(graph *g, int n, int maxn);
     g          grafo en formato nauty
     n          número de vértices en g
     maxn       número de vértices para salida (obtenido de los
argumentos de la línea de comandos)
```

Ilustración 29 - Definición de PRUNE

geng construye el grafo comenzando con el vértice 0, y añade los vértices 1, 2, 3… en orden. Cada grafo en la secuencia será introducido como un subgrafo de los demás grafos en la secuencia. Estos grafos (parciales y finales) son pasados por referencia a la función PRUNE que realizará el tratamiento personalizado del grafo.

Este tratamiento del grafo es donde se realiza la exclusión de los grafos que no cumplan con ciertas características. Si PRUNE devuelve un valor distinto de 0 el grafo es rechazado y se pasa a la generación del siguiente grafo. Podemos ver un ejemplo de algoritmo de exclusión en uno de los apartados del problema propuesto con la exclusión de grafos con ruedas de 6 vértices:

```
int PRUNE (graph *g, int n, int maxn)
{
setword escribir[n];
setword common, common2, common3, common1, common4, common5, common6, common7;
      int v,w,u,i,z, indice, d;

      for(i=n-1; i>2; i--){
            for (v = i-1; v >= 0; v--){
                  if (((g[i] & bit[v]) == 0)&&(i!=v)){
                        common = g[i] & g[v];
                        common1=common;
                        while (common){
                              TAKEBIT(w,common);
                              common2=common1 & ~g[w];
                              common2=common2 & ~bit[w];
                              common3=common1 & g[w];
                              while(common2){
                                    TAKEBIT(u, common2);
                                    common5=common3 & g[u];
                                    if(common5){
                                          common4=common5;
                                          while(common4){
                                                TAKEBIT(z, common4);
                                                common6=g[z] & common5;
                                                common7=common6;
                                                while(common7){
                                                      TAKEBIT(d, common7);
                                                      if(common6 & g[d])
                                                            return 1;
```

```
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                return 0;
}
```

Ilustración 30 - Función PRUNE para detección de ruedas de 6 vértices con K_5 en su interior

Podemos ver a simple vista la complejidad del cálculo a realizar para la exclusión en cada uno de los grafos generados por geng que es del orden exponencial. Utilizando para este proyecto detectores de ruedas de tamaño 4, 5 y 6 vértices.

Apéndice A: Estadísticas de uso de geng.

Grafos

V	*Número de grafos*		
1	1		
2	2		
3	4		
4	11		
5	34		
6	156		
7	1044		
8	12346	0,11	segundos
9	274668	1,77	segundos
10	12005168	1,22	minutos
11	1018997864	1,72	horas
12	165091172592	285	horas
13	50502031367952	10	años

Grafos libres de C3 (-t)

V	*Número de grafos*		
1	1		
2	2		
3	3		
4	7		
5	14		
6	38		
7	107		
8	410		
9	1897		
10	12172	0,21	segundos
11	105071	1,49	segundos
12	1262180	15,9	segundos
13	20797002	4,08	minutos
14	467871369	1,5	horas
15	14232552452	45,6	horas
16	581460254001	79	días

Grafos libres de C4 (-f)

V	*Número de grafos*		
1	1		
2	2		
3	4		
4	8		
5	18		
6	44		
7	117		
8	351		
9	1230		
10	5069	0,11	segundos
11	25181	0,48	segundos
12	152045	2,67	segundos
13	1116403	18	segundos
14	9899865	2,5	minutos
15	104980369	25,7	minutos
16	1318017549	5,33	horas
17	19427531763	82,6	horas
18	333964672216	62	años

Grafos libres de C3 y C4 (-tf)

V	*Número de grafos*		
1	1		
2	2		
3	3		
4	6		
5	11		
6	23		
7	48		
8	114		
9	293		
10	869		
11	2963	0,1	segundos
12	12066	0,36	segundos
13	58933	1,5	segundos
14	347498	7,76	segundos
15	2455693	50,9	segundos
16	20592932	6,79	minutos
17	202724920	1,11	horas
18	2322206466	12,7	horas
19	30743624324	168	horas
20	468026657815	110	días

Grafos Bipartitos (-b)

V	*Número de grafos*		
1	1		
2	2		
3	3		
4	7		
5	13		
6	35		
7	88		
8	303		
9	1119		
10	5479	0,11	Segundos
11	32303	0,59	Segundos
12	251135	3,99	Segundos
13	2527712	35,1	Segundos
14	33985853	7,22	Minutos
15	611846940	2,05	Horas
16	14864650924	48,9	Horas
17	488222721992	70	Días

Grafos Bipartitos libres de C4 (-bf)

V	*Número de grafos*		
1	1		
2	2		
3	3		
4	6		
5	10		
6	21		
7	39		
8	86		
9	182		
10	440		
11	1074		
12	2941	0,15	segundos
13	8424	0,43	segundos
14	26720	1,37	segundos
15	90883	4,3	segundos
16	340253	14,9	segundos
17	1384567	57,1	segundos
18	6186907	4,01	minutos
19	30219769	18,4	minutos
20	161763233	1,57	horas
21	946742190	8,85	horas
22	6054606722	56,2	horas
23	42229136988	16,6	días
24	320741332093	121	días

Fuente: Manual de usuario de la utilidad *geng* incluido en la distribución **nauty 2.2 Stable**.

Apéndice B: Plantillas de CLASSADs Condor del CICA

ClassAD para tareas secuenciales:

```
# ClassAD Tareas Secuenciales #

universe          = vanilla
executable        = fichero, binario, ejecutable (puede ser un script)
requirements      = (Arch == "X86_64" && OpSys == "LINUX")
arguments         = ficheros entrada o argumentos

log               = nombre.log
output            = nombre.out
error             = nombre.err

transfer_input_files    = ficheros E/S, argumentos, ficheros separados
                          por coma.
should_transfer_files   = YES
when_to_transfer_output = ON_EXIT_OR_EVICT

queue
```

Ilustración 31 - Plantilla Condor para tareas secuenciales

ClassAD para tareas paralelas (MPI[13]):

```
# ClassAD Tareas Paralelas MPI #

universe          = parallel
executable        = /opt/openmpi/mpirun .... datos de la ejecución MPI
requirements      = (Arch == "X86_64" && OpSys == "LINUX")
arguments         = ficheros de entrada o argumentos

log               = nombre.log
output            = nombre.out
error             = nombre.err

machine_count     = número de máquinas que se usarán en la tarea
Scheduler         = "DedicatedScheduler@condor.cica.es"
transfer_input_files    = ficheros E/S, argumentos, ficheros separados
                          por coma.
should_transfer_files   = YES
when_to_transfer_output = ON_EXIT_OR_EVICT

queue
```

Ilustración 32 - Plantilla Condor para tareas en paralelo

[13] *Message Passing Interface*

Apéndice C: Instalación de Condor sobre una red de Computación

Antes de la instalación es necesario tomar unas decisiones de arquitectura:

¿Qué máquina será el Gestor Central (Central Manager)?
¿Qué máquinas tendrán permiso para enviar trabajos?
¿Ejecutar Condor como "root" o como usuario restringido?
¿Tendrá un usuario denominado "condor" y compartirá su directorio principal?
¿Qué directorios podrá utilizar Condor en cada máquina?
¿Qué parte de Condor será instalado?
    Ficheros de Configuración
    Directorio de resultados, librerías, sistema, etc…
    Documentación
¿Espacio de almacenamiento para Condor?

Una máquina de la red debe ser el Gestor Central, es necesario instalar Condor en esta máquina primero. Esta es el respositorio de información centralizado. Si el Gestor Central deja de responder por algún motivo (fallo del sistema operativo, fallo eléctrico, etc..) cualquier máquina asignada a trabajo podrá continuar su labor, pero no podrá avanzar las colas de procesos, por ello es necesario que el Gestor Central sea reiniciado lo antes posible tras un fallo de este tipo.

El Gestor central gestiona una considerable cantidad de tráfico de red: todos los demonios ("daemons") envían actualizaciones cada 5 minutos (por defecto) a esta máquina. Además del volumen de red debe gestionar una mayor cantidad de memoria dependiendo del número de máquinas en la red. Una red de computación con 100 máquinas requiere aproximadamente 25Mb de memoria en el Gestor Central, sin embargo una red con 1000 máquinas requiere unos 100Mb de memoria.

La velocidad de CPU del Gestor también influye en el tiempo de *matchmaking*[14].

Algunos de los daemons de Condor requieren privilegios de administración, sin esto, Condor no podrá tomar decisiones ni aplicar políticas de seguridad. Es posible ejecutarlo en un entorno de usuario restringido pero conlleva serias consecuencias de seguridad y bajo rendimiento.

Condor necesita una serie de directorios que son individuales para cada máquina de la red: *spool*, *log* y *execute*. *Execute* es el directorio que actúa como directorio de trabajo para cualquier trabajo Condor que ejecute en la máquina. *Spool* contiene la cola de trabajos, el historial de ejecución y los puntos de recuperación para todos los trabajos, este directorio puede requerir mucho espacio de almacenamiento. Cada daemon de Condor escribe su propio registro en el directorio *log*.

[14] Asociación de un trabajo a una máquina en la red

Cada distribución dispone de 5 subdirectorios: *bin*, *etc*, *bin*, *sbin* y *libexec*. Estos subdirectorios se encuentran en el directorio RELEASE_DIR especificado en los ficheros de configuración y están destinados a almacenar los ejecutables y librerías necesarias para la ejecución correcta de Condor en la máquina.

Instalación de Condor en Unix:

Tras la descarga, todos los ficheros están comprimidos en formato *tar*. Necesitan ser descomprimidos con el comando:

```
tar xzf condor.tar.gz
```

Creará un directorio que contiene los scripts *condor_configure* y *condor_install*, así como los directorios *bin, etc, examples, include, lib, libexec, man, sbin, sql y src*.

El script *condor_install* se utilizará para instalar Condor mientras que *condor_configure* es utilizado para modificar la configuración de una instalación Condor existente. Sin embargo ambos scripts son idénticos salvo en los parámetros por defecto: *condor_install* es equivalente a ejecutar el script "*condor_configure –install=.*"

Los archivos anteriores aceptan argumentos de los cuales para la instalación necesitaremos conocer:

--install=/ruta/de/instalación : Especifica la ruta para los directorios Condor.

--install-dir=directorio
--prefix=directorio : Especifican el directorio de instalación

--local-dir=directorio : Especifica la ruta del directorio local

--type=manager,execute,submit : Define el rol de la máquina

Instalar Condor en el Gestor Central:

```
# condor_install –prefix=~condor –local-dir=/programs/condor \
 –type=manager
```

Para actualizar la instalación anterior de Condor y añadir permiso para enviar trabajos:

```
# condor_configure –prefix=~condor –localdir=/programs/condor \
 –type=manager,submit
```

Las opciones de *–type* son *manager, submit, execute*. La opción "–type=manager, submit" sólo es recomendada para redes de computación muy pequeñas.

Tras la instalación del Gestor Central de Condor en una máquina es necesario configurar el resto de las máquinas para ejecución y/o envío de trabajos (dependiendo de la función que se asigne a la máquina la opción type contendrá execute, submit o ambas):

```
# condor_install –prefix=~condor –local-dir=/programs/condor \
 –type=execute,submit
```

El software Condor dispone de una versión para sistemas operativos Windows XP, 2000 o superior.

Apéndice D: Instalación de Sun Grid Engine sobre una red de Computación.

La instalación *de Sun Grid Engine* requiere gran cantidad de información para su funcionamiento. Esta sección pretende ser una guía simplificada para la instalación básica del gestor de procesos basado en colas *Sun Grid Engine*. El primer paso es descargar el fichero de instalación para la plataforma, en este caso será `Linux24_amd64/tar/sge-6_2-bin-linux24-x64.tar.gz`

Creación del directorio de instalación:

```
# mkdir –p <directorio de instalación>
# chown <usuario administrador> <directorio de instalación>
# chmod 755 <directorio de instalación>
```

El sistema *Sun Grid Engine* necesita un puerto TCP para comunicaciones, todos los anfitriones en el cluster deben utilizar el mismo puerto, para ello hay que modificar el fichero `/etc/services` y añadir el servicio "`sge_commd 535/tcp`" en cada máquina. La selección del puerto puede ser un problema si no conocemos los procesos que ejecutará el sistema, por ello se utiliza un puerto privilegiado por debajo de 600 que evita conflictos con aplicaciones que asignen dinámicamente puertos, privilegiados o no.

Desempacamos el contenido del fichero *tar* que contiene los binarios de *Sun Grid Engine* en el directorio seleccionado para la instalación con el usuario que deba ejecutarlo:

```
# cd <directorio de instalación>
# tar –xvpf sge-6_2-bin-linux24-x64.tar.gz
```

Las instalaciones estándares *de Sun Grid Engine* consisten en un anfitrión ***master*** y un número arbitrario de nodos de ***ejecución***. El nodo maestro controla la actividad y carga del cluster mientras que los nodos de ejecución controlan los trabajos que les han sido asignados. Es posible que un nodo maestro sea a su vez un nodo de ejecución.

Una vez seleccionado el nodo maestro, una máquina que no tenga sobrecarga con otras tareas y que disponga de la memoria necesaria para ejecutar SGE (el requerimiento de memoria depende de la cantidad de ordenadores en el cluster y de la carga de trabajo, 10Mb para clusteres con pocas docenas de nodos y 100 trabajos hasta 1Gb con 1000 nodos y 10000 trabajos) procedemos a la instalación del nodo.

La instalación requiere permisos administrativos, por lo que es necesario ejecutar los *scripts* con el usuario *root*:

```
# ./install_qmaster
```

El proceso en los nodos de ejecución es parecida a la del nodo maestro, es necesario ejecutar los scripts con el usuario root para tener acceso a todas las características de Sun Grid Engine. Una vez en el directorio de instalación con el contenido del fichero tar desempacado ejecutamos el comando:

```
# ./install_execd
```

En algún momento de la instalación será necesario introducir la configuración por defecto para el nodo y las máquinas en las que se instalará el *agente de ejecución* de Sun Grid Engine, además de la configuración de las colas iniciales del nodo. Una cola describe la lista de atributos y requerimientos que un proceso debe cumplir para poder ejecutarse en ese nodo en particular. Las colas para los nodos de ejecución disponen:

- Nombre de la cola: <nombre>.q
- Posiciones (trabajos concurrentes): <número de procesadores>
- Asignan recursos ilimitados del sistema (memoria, tiempo de CPU) a los trabajos.
- No fuerzan restricciones para usuarios o grupos de usuarios. Cualquier usuario con una cuenta válida puede ejecutar trabajos en la cola.

Apéndice E: Script de ejecución en Sun Grid Engine

Como en Condor, para el entorno Sun Grid Engine es necesario indicar unas restricciones y características especiales necesarias para la ejecución de la tarea. Este script no es un proceso común en SGE pues para esta tarea en particular necesitamos reservar cierto número de máquinas para conectar con ellas por SSH (Secure Sockets) y ejecutar los trabajos en cada una, para ello tenemos que indicar:

- Shell que ejecutará la tarea: bash
- Nombre del trabajo: W5FJob
- Número de máquinas a reservar a la vez: 4

Para poder conectar a las máquinas reservadas tenemos que conocer sus identificadores asignados en el cluster, haciendo uso del fichero machines que contiene los ordenadores reservados para la tarea. Una vez conocidos los nodos reservados conectamos uno a uno con las credenciales necesarias y ejecutamos las tareas en paralelo. Para SGE es importante definir la ruta absoluta del ejecutable.

```
#$ -S /bin/bash
#$ -N W5FJob
#$ -pe mpi_slots 4
#$ -e w5f$JOB_ID.err
#$ -o w5f$JOB_ID.out
#$ -V
#$ -cwd


## Creamos un fichero temporal con las maquinas que SGE le ha asignado
### Cada linea corresponde con una cpu reservada, a la que nos
conectaremos despues por SSH

cat $TMP/machines > machines$JOB_ID


## Creamos el bucle para conectarnos por ssh a las maquinas anteriores.
El comando que se ejecutara es el que va entre comillas

for i in $(cat machines$JOB_ID)
do

/usr/bin/ssh $i "./w5f -Cd5 15 $SGE_TASK_ID/10 | ./countg –HG"

done
```

Ilustración 33 - Script de ejecución del problema en SGE

En esta configuración realiza una conexión SSH sobre cada ordenador reservado (indicado en el fichero machines$JOB_ID[15]) y ejecuta el comando *./w5f* con 10 particiones que irán secuencialmente sobre cada máquina. El resultado de cada partición será la entrada de *./countg*. El comando *w5f* es un generador de grafos con exclusión de aquellos con ruedas de 5 vértices.

[15] JOB_ID es asignado automáticamente a cada tarea al ser enviada al nodo de ejecución.

ANEXO I: Índice de Ilustraciones

Ilustración 1 - Vértice 8
Ilustración 2 - Arista 8
Ilustración 3 - Ciclo 8
Ilustración 4 - Grafo 9
Ilustración 5 - Grafo Bipartito 9
Ilustración 6 - Digrafo 9
Ilustración 7 - Multigrafo 10
Ilustración 8 - Matriz de adyacencia para grafo con 3 vértices 10
Ilustración 9 - Subgrafo 10
Ilustración 10 - Ejemplo de Grafos Isomorfos 12
Ilustración 11 - Matrices de Permutación de orden 3 12
Ilustración 12 - Condor Submit Description File 15
Ilustración 13 - Condor ClassAD 15
Ilustración 14 - Extracto del registro de Condor 16
Ilustración 15 - Ejemplo de Condor Submit Description File 16
Ilustración 16 - Script de ejecución SGE serie 17
Ilustración 17 - Script de ejecución SGE paralelo 17
Ilustración 18 - Ejemplo de grafos de 20 vértices en formato graph6 22
Ilustración 19 - Orden de los bits en la matriz de adyacencia 23
Ilustración 20 - Grafo de 7 vértices 23
Ilustración 21 - Algoritmo de codificación de aristas en sparse6 26
Ilustración 22 - Diccionario de Requisitos 29
Ilustración 23 - Carga media del cluster CICA 36
Ilustración 24 - Matriz de adyacencia de un grafo K_5 37
Ilustración 25 - Grafo K_5 en una Matriz de Adyacencia de orden arbitrario 37
Ilustración 26 - Zona variable de un grafo arbitrario+K_N no dirigido 39
Ilustración 27 - Zona de generación de un grafo arbitrario+K_N+K_M no dirigido 39
Ilustración 28 - Parámetros de la utilidad geng más comunes 41
Ilustración 29 - Definición de PRUNE 42
Ilustración 30 - Función PRUNE para detección de ruedas de 6 vértices con K_5 en su interior 43
Ilustración 31 - Plantilla Condor para tareas secuenciales 46
Ilustración 32 - Plantilla Condor para tareas en paralelo 46
Ilustración 33 - Script de ejecución del problema en SGE 52

Bibliografía:

Jonathan L. Gross, Jay Yellen.*Handbook of Graph Theory*. CRC Press, 2004. ISBN 158488090

Augusto Cortéz. Teoría de la Complejidad Computacional y Teoría de la computabilidad. Facultad de Ingeniería Informática. Universidad Nacional Mayor de San Marcos, 2004. ISSN: 1815-0268 (impreso).

Brendan D. Mckay, *Practical graph Isomorphism*. Congressus Numerantium, Vol. 30 (1981).
http://cs.anu.edu.au/~bdm/papers/pgi.pdf

Carlos Paredes Egea, Dídac González Sánchez, Josep María Borell Serra, Eva Jurjo Mosoll, Luis Gámiz Fernández. *Análisis del Software Grafos*, Métodos Cuantitativos III, Curso 2007-08, Q2. Universidad Politécnica de Cataluña, ETSEIAT.
http://personales.upv.es/arodrigu/grafos/ayuda/Analisis_Soft_Grafos_UPC_MC3.pdf

Condor Team, University of Wisonsis-Madison, October 16, 2006. *Condor® Versión 6.8.2 Manual.*
http://www.cs.wisc.edu/condor/

Sun Microsystems. *Sun N1 Grid Engine 6.1 User's Guide*. Part No: 820-0699, May 2007.
http://gridengine.sunsource.net/