ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

COMPUTACIÓN DISTRIBUIDA Y PARALELA PARA CÁLCULO INTENSIVO DE ISOMORFISMOS DE GRAFOS

Realizado por

MARIO DOMÍNGUEZ HERNÁNDEZ

Para la obtención del título de

INGENIERO TÉCNICO EN INFORMÁTICA DE SISTEMAS

Dirigido por

JOSÉ RAMÓN PORTILLO FERNÁNDEZ

Realizado en el departamento de

MATEMÁTICA APLICADA I

SEVILLA, SEPTIEMBRE 2011

ÍNDICE

1.- Definición objetivos **4**

2.- Análisis de antecedentes y aportación realizada **6**

3.- Análisis: **14**

A) Requisitos 14

B) Diseño 14

Cluster 15
Beowulf 16
GPU 16
GRID 18
Boinc 18
Folding 19
Ibercivis 20

C) Implementación 22

4.- Manual de usuario **26**

SSH 26
PuTTY 28
SGE (GRID ENGINE) 29
Ganglia 32
CICA 32
RICA 34

5.- Pruebas **35**

Prueba 1 35
Prueba 2 37
Prueba 3 40
Prueba 4 42
Prueba 5 45
Prueba 6 47
Prueba 7 48
Prueba 8 50

6.- Comparación con alternativas **54**

Clúster Beowulf 54
Clúster SUN 54
Clúster CICA 54
MPI 55
Ibercivis 57
Condor 57

7.- Conclusiones y desarrollos futuros **59**

Distribución del tiempo 61

Bibliografía **63**

1.- DEFINICIÓN DE OBJETIVOS

Para tratar un problema tal que al elevar el número de incógnitas o el tamaño del problema, su complejidad se dispara exponencialmente, necesitamos recurrir a técnicas para reducir dicha complejidad. Una de ellas es el uso de la computación distribuida o paralela (las diferencias entre ambas se tratarán en un capítulo posterior).

Si encima lo que intentamos resolver son problemas basados en la Teoría de Grafos, lo más idóneo seria apostar por la computación distribuida y un lenguaje de programación estructurada.

Nuestro objetivo principal será generar todos los grafos posibles con N vértices, y cribarlos bajo algún tipo de cualidad (regularidad, ausencia de triángulos, etc.), para así poder llegar a estudiar el isomorfismo entre grafos.

El problema que debemos resolver es básicamente la imposibilidad de resolver un problema tan voluminoso y extenso para un N lo suficientemente grande.

Por ejemplo, si intentamos generar todos los grafos posibles de 12 vértices, resolverlo en una única máquina de modo secuencial, nos llevaría muchísimo tiempo (del orden de semanas) y tendríamos que disponer de gran cantidad de memoria (aproximadamente 2,4 TB). Como esto es inviable, tendremos que distribuir el volumen de cómputo en varias maquinas y diseñar un algoritmo que sea inherentemente paralelo.

Es bastante difícil imaginarse la cantidad de tiempo y de recursos que consumiría el generar todos los posibles grafos de 15 vértices en una única máquina, posiblemente se vería desbordada o daría error. Uno de los objetivos de este proyecto es poder llegar a distribuir este problema.

Usaremos las infraestructuras que nos prestan en el CICA, usaremos el gestor de colas SGE e implementaremos un script para resolver el problema.

El tratamiento de problemas basado en la Teoría de Grafos es idóneo para tratarse de modo distribuido y paralelizado. Si tenemos en cuenta que los problemas modelados con grafos se prestan a hacer llamas recursivas, ha invocar a un problema similar al anterior pero de menor tamaño, o incluso a calcularse muchas veces un mismo problema pero con distintos parámetros iniciales, entenderemos como la computación masiva, la fuerza bruta y la distribución del sistema son básicos para atacar un problema basado en la Teoría de Grafos lo suficientemente grande.

El uso de las GPU y su predisposición por las tareas de fuerza bruta, las hacen francamente practicas para este tipo de acometidos.

Pero como ya hemos visto en algún ejemplo, da igual que tipo de maquina poseas ni lo potente que sea, que si el problema a tratar es lo suficientemente grande, será imposible de abarcar si no se trata de una forma distribuida u/o paralela.

2.- ANÁLISIS DE ANTECENTES Y APORTACIÓN REALIZADA

La Computación Distribuida nace como ya hemos dicho por la necesidad de dar solución a problemas escalables e intratables a partir de cierto tamaño. Desde ya hace varias décadas se vienen pensando estructuras paralelas en la arquitectura de los computadores, pero el mero hecho de duplicar o triplicar un componente no significa que pueda duplicar o triplicar su rendimiento.

A la hora de multiplicar CPU´s (2, 4, 8 cores, etc.) en los ordenadores actuales, no supone un aumento considerable en el rendimiento de las aplicaciones que en ellos se corren. La aplicación aprovecha las CPUs que puede, pero las aplicaciones no suelen estar pensadas para distribuirse entre cores, estas (y la gran mayoría) de las aplicaciones están escritas secuencialmente, lo que quiere decir que están pensadas para correrse sobre un único core.

El esfuerzo humano a la hora de portar las aplicaciones es vital pero no trivial, cada problema necesita un tratamiento diferente el cual debe estudiarse antes de modelar la aplicación. Al ver la programación desde otro punto de vista se le empieza a llamar, Programación Paralela, la cual trata la algoritmia sin olvidar que deberá estar distribuido entre nodos más o menos cercanos y que debe haber una comunicación entre ellos antes de recopilar toda la información .En los casos reales un gran problema se trocea en cientos de miles o incluso en millones de subproblemas para ser enviados a los nodos del clúster o a los ordenadores de una grid distribuidos por todo el globo. Después de calcular cada pequeño problema se reúnen todas las subsoluciones y se unifican para conseguir la solución real al problema propuesto.

En la actualidad podríamos ver dos grandes grupos de intereses en esto de la Computación Distribuida.

Uno seria la carrera internacional a la hora de poseer los clústeres más veloces y potentes, con los cuales poder llegar a conclusiones o simulaciones con mayor velocidad que con uno menos potente. La gran mayoría tiene objetivos industriales, de investigación o gestión gubernamental, en menor medida se utilizan con fines militares o con actividad clasificada. Los de TOP500 se reúnen 2 veces al año para configurar una lista mundial con los clústeres de mayor rendimiento, a los cuales se les hace pasar por varias pruebas de rendimiento y de potencia de cómputo, para así poder clasificarlos lo más fidedignamente posible.

El otro grupo sería la proliferación y consolidación de GRIDs voluntarias, en las que participan personas repartidas por todos los continentes a través de internet. Donan ciclos de computación, para poder entre muchos llegar a resolver cálculos complejos para científicos ,físicos , astrónomos, etc. Es lo que se llama computación voluntaria, los voluntarios donan su hardware a

los investigadores que ellos elijan, para así abaratarles los costes de realización de dicha investigación. En este campo podremos encontrar desde proyectos para dar soporte al LHC (Gran Colisionador de Handrones) de Ginebra hasta otros como Folding para el cálculo de plegamiento de proteínas.

Desde que el hombre descubre las matemáticas siempre quiso encontrar soluciones a problemas cada vez más grandes. Con el gran avance que han supuesto las computadoras modernas hemos podido resolver problemas antes inviables.

Probablemente el primer dispositivo mecánico de cálculo fue la sumadora de Pascal (1642). Junto con la máquina multiplicadora de Leibniz (1673), estos dispositivos permitían realizar cálculos con mayor velocidad.

En 1966 Flynn nos dejo su clasificación:
SISD - único flujo de instrucciones aplicado a un único flujo de datos
SIMD - único flujo de instrucciones aplicado a múltiples flujos de datos
MIMD - múltiples flujos de instrucciones aplicadas a múltiples flujos de datos

La supercomputación se asocia principalmente a los MIMD, donde se encuentran los clústeres y multiprocesadores.

En la primera mitad del siglo XX, la aparición de componentes eléctricos y electrónicos dio lugar a grandes avances en el campo de la automatización de cálculos. La Segunda Guerra Mundial puso a prueba a las mentes del momento y aceleró el desarrollo de las primeras computadoras digitales. Alan Turing ya introdujo el concepto de procesamiento simbólico, y la idea de una máquina universal (máquina de Turing) capaz de ejecutar cualquier algoritmo que pudiera describirse.

Desde el ENIAC en el 1946 comienza una carrera por la perfección de las partes de un computador cada vez más barato y pequeño. En 1964 IBM crea el "Sistema 360", formado por seis procesadores y podía llegar a soportar 40 periféricos.

En 1967, Gene Amdahl publicó su famosa Ley de Amdahl, la cual describe matemáticamente el aceleramiento que se puede esperar paralelizando cualquier tipo de tarea dentro de una arquitectura paralela. En otro capítulo veremos con más detenimiento esta ley.

En la década de los 80 el software paralelo se centró en proveer herramientas de comunicación entre procesadores. Parallel Virtual Machine (PVM),
Message Passing Interface (MPI), High Performance Fortran (HPF), y OpenMP fueron desarrollados para soportar las comunicaciones de aplicaciones escalables.

Para muchos científicos, la calidad de sus investigaciones depende mucho

de la computación a gran escala. No es raro encontrar problemas que requieran semanas o meses de cálculo para obtener una solución. Los científicos involucrados en este tipo de investigaciones necesitan un entorno de computación que proporcione una gran cantidad de potencia de procesamiento durante un amplio período de tiempo.

Estos entornos se denominan entornos HTC (High-Throughput Computing). Por contra, los entornos HPC (High-Performance Computing) proporcionan una gran potencia o cantidad de recursos durante un corto período de tiempo. Los entornos HPC se miden frecuentemente en términos de operaciones en coma flotante por segundo (FLOPS). Mientras que los entornos HTC se suelen medir en operaciones en coma flotante por mes o por año. Los entornos HTC están enfocados a completar el mayor número de trabajos en un largo período de tiempo.

La clave en los entornos HTC es el uso eficiente de los recursos disponibles. Hace años, para poder hacer frente a los problemas que requerían una gran cantidad de recursos durante mucho tiempo, se recurría a los mainframes o supercomputadores. Estos superordenadores eran capaces de afrontar dichos problemas con su gran potencia de cálculo. El problema era el alto precio de estos sistemas, relegando su uso a entidades elitistas (como gobiernos o universidades con alto presupuesto). Además, estos superordenadores tenían un inconveniente, su uso era exclusivo para la tarea o trabajo que procesaba.

En la actualidad, podemos utilizar un número determinado de nodos heterogéneos e interconectarlos para que se distribuyan las tareas a ejecutar. Dichos nodos representan recursos, que pueden ser simples ordenadores personales. Por tanto, en sustitución de los superordenadores, podemos utilizar múltiples ordenadores convencionales (mucho más baratos), interconectarlos a través de una red e instalar un software que se encargue de repartir el trabajo entre los distintos ordenadores, de tal manera, que se aprovechan aquellos que presentan una escasa actividad de procesamiento.

Hoy día, la mayoría de las grandes y medianas empresas poseen algún tipo de infraestructura paralela o distribuida, para dar servicio por internet, o para realizar cálculos extensos.

La mayoría de estos usan Linux como sistema operativo (alrededor del 82%), ya que así evitas el pago de numerosas licencias de algún otro sistema operativo. Teniendo en cuenta que te pedirían abonar una licencia por cada nodo de la red de computación, al contar con que estos pueden ser decenas, cientos o miles de nodos, el gasto sería enorme e inútil.

También existen empresas que lo único que hacen es aportar el volumen de cómputo necesario para otras, y así servirles a las empresas clientes como un centro de cálculo en el que correr todo tipo de procesos o aplicaciones.

Los mayores volúmenes de cómputo de los clústeres existentes se van

reflejando en las reuniones del TOP500 que se celebran un par de veces al año. En ella se pasan a los nuevos (o modificados) clústeres a diferentes benchmarking o pruebas de rendimiento. De ellos el más famoso es el LINPACK, unas pruebas de rendimiento para el cálculo de sistemas lineares de alta densidad.

El uso del benchmarking de LINPACK fue introducido en 1979 por Jack Dongarra, investigador de la Universidad de Tennessee en Knoxville, como parte del paquete LINPACK, un juego de bibliotecas matemáticas en FORTRAN para solución de sistemas de ecuaciones lineales.

El seguimiento por parte del TOP500 se viene haciendo desde el 1993 para poder llegar ahora a conclusiones que antes no podíamos, saber las tendencias de los clúster, que arquitecturas consiguen mayor rendimiento, que microprocesadores destacan sobre otros, que sistemas operativos son más usados o para tener un mínimo control de la finalidad de estos clústeres o de sus propietarios.

Con el actual número 1 de la lista (jun2011) se llega a un rendimiento de 8 billones de operaciones por segundo (petaflop/s), el clúster japonés que se encuentra en la ciudad de Kobe, se llama K Computer. Su nombre procede de la palabra nipona "Kei" por 10^16, el cual representa el rendimiento ideal de la maquina, 10 PetaFLOPS. Posee 68.544 SPARC64 VIIIfx CPUs, cada uno con 8 cores, con un total de 548.352.

Por primera vez en la historia, de entre todas las maquinas de la lista, las diez primeras son las únicas que desarrollan una potencia superior al petaflop/s. De estas maquinas, cinco pertenecen a EEUU, dos a Japón y a China, y una a Francia.

Después de la compra de SUN por parte de Oracle, el software antes conocido como SGE (SUN Grid Engine) ha pasado a llamarse OGE (Oracle Grid Engine). Oracle ofrece una versión del software en su página web, que al igual que sus futuras actualizaciones, todas serán de uso comercial.

Cuando la versión 6.2u6 de Grid Engine fue lanzada el pasado año, no ofrecía el código fuente. En consecuencia a ello, la comunidad que respaldaba el uso libre de la herramienta ha emprendido su propio proyecto, Open Grid Scheduler, el cual tiene como objetivo seguir manteniendo la versión abierta y gratuita de Grid Engine. Siguiendo así con la filosofía de comunidad con la que empezaron a colaborar en el proyecto SUN Grid Engine desde el 2001, año en el que SUN libero el código fuente. La empresa Univa también posee su propia versión forkeada de Grid Engine para uso comercial, después de anunciar a principio del 2011 que había contratado a ingenieros del antiguo grupo de trabajo de SGE -SUN.

Debemos hacer una distinción entre ambos términos. Ambos métodos no son adecuados para los mismos problemas, dependiendo del problema, o más bien sobre su enfoque, favorecerá a la solución un camino u otro. La computación paralela divide una aplicación en tareas que son ejecutadas al

mismo tiempo, en cambio la computación distribuida divide una aplicación en tareas que son ejecutadas en diferentes ubicaciones usando diferentes recursos.

La computación distribuida está más enfocada a problemas en los que el aumento de incógnitas o del tamaño del problema se va haciendo cada vez más complicado y debemos distribuir la carga entre mas máquinas, son problemas escalables que aumentan de tamaño y de complejidad muy rápidamente. Se consigue su máximo potencial en la GRID, donde se reparten subproblemas entre maquinas heterogéneas a traves de Internet.

La computación paralela está más enfocada a problemas cuyo espacio de soluciones sea fácilmente acotable y en donde el gran problema se divide en infinidad de problemas más pequeños y fáciles de resolver. Usan mucho el concepto de granularidad fina o gruesa para referirse al tamaño de esos trabajos. Es posible el uso de cualquier tipo de programación, aunque algunas aplicaciones requieren trabajar con lenguajes de alto nivel. Gracias a las GPUs (Graphics Processing Unit) la computación paralela ha podido exprimir todo su potencial, ya que su arquitectura es inherentemente paralela.

Nuestro problema consta de dos grandes partes las cuales pasaré a explicar: la generación y la criba. Haremos uso de dos comandos de la librería nauty: geng y pickg. El primero es una instrucción para generar grafos no isomorfos, y el segundo para cribarlos dependiendo de nuestras necesidades.
Si lo que queremos es trabajar con todos los grafos posibles de N vértices, lo que haremos será invocar al geng sin parámetros.

$ geng N

ejemplo:

$ ./geng 2
>A ./geng -d0D1 n=2 e=0-1
A?
A_
>Z 2 graphs generated in 0.00 sec

Los datos de los grafos están en formato graph6 o en sparse6
Hay gran cantidad de parámetros para generar esos grafos bajo precondiciones. Eso sería un pretratamiento de la solución muy útil, pero de la que nosotros prescindiremos.

Cuidando con poner algún parámetro lo suficientemente alto, estas son las pruebas realizadas secuencialmente en un 4cores de SUN

$ geng 9 274668 grafos generados en 0,81 segundos
$ geng 10 12005168 grafos generados en 28,59 segundos
$ geng 11 1018997864 grafos generados en 2447,93 segundos = 40,73 minutos

Vemos claramente cómo el crecimiento del tiempo de ejecución es exponencial, haciendo intratable valores mayores.

Nuestras pruebas han sido distribuyendo versiones del problema con valores 12 y 15. Entonces supe que no me habría dado tiempo físico razonable para acabar las pruebas si hubiese elegido construir y usar un clúster beowulf. Los ficheros intermedios pueden pesar los 400 GB y eso que no es el proceso entero, sino sólo una ínfima parte de él. Las pruebas las he realizado con 200, 2000 y 8000 particiones.

El uso del geng es este

> Usage: geng [-cCmtfbd#D#] [-uygsnh] [-lvq] [-x#X#] n [mine[:maxe]] [res/mod] [file]

Las restricciones del geng son estas

n: número de vértices (1...32)
min: maxe: rango para el numero de vértices
(#:0 significa ' # o más ' excepto en el caso 0:0)
res/mod: sólo genera 'res' subconjuntos del rango 0... mod -1
-c : solo escribe grafos contados
-C: solo escribe grafos biconectados
-t: solo genera grafos libres de triángulos
-f: solo genera grafos libres de ciclos de tamaño cuatro
-b: solo genera grafos bipartidos
(-t, -f y -b pueden ser usados en cualquier combinación)
-m: salva memoria en las esperas de tiempo
(sólo marca la diferencia en la ausencia de -b, -t, -f y n <= 28).
-d#: una poda mínima para un mínimo grado
-D#: una poda máxima para un máximo grado
-v: muestra la cuenta del número de grafos
-l: grafos con etiquetas canónicas
-u: no vuelques ningún grafo por la salida, genéralos y cuéntalos
-g: usa el formato graph6 (por defecto)
-s: usa el formato sparse6
-y: usa el formato obsoleto Y-format en vez del graph6
-h: para el formato graph6 o sparse6, añadir también un encabezado
-q: suprime salida auxiliar (excepto para -v)

En vez de geng podríamos utilizar cualquier otro programa del paquete nauty que generase grafos, como el w3f, w4f y el w5f los cuales generan grafos libres de ruedas "wheel-3-free" grafos libres de ruedas de tamaño 3.

Invocar a pickg sin parámetros es lo mismo que no hacer nada con el parámetro de entrada y devolverlo tal cual en el parámetro de salida.

$ pickg

Nosotros invocaremos el pickg detrás de la tubería para tomar como parámetro de entrada la salida del geng.

$ geng 12 | pickg -r

Este nos servirá para cribar de entre todos los posibles grafos de 12 vértices los que cumplan la cualidad de ser regulares.

Restricciones posibles (# es un valor, pero también puede ser un rango #:#, #:# o negado ~#)

-n# numero de vértices
-d# grado mínimo
-r regulare
-z# radio
-g# girth (0=acíclico)
-E Euleriano (todos los grados son par, la conectividad no es necesaria)
-a# tamaño del grupo
-c# conectividad
-e# numero de aristas
-D# grado máximo
-b bipartido
-Z# diámetro
-Y# número total de ciclos
-o# órbitas
-F# fixed points
-t vértice transitivo
-i# mínimo común nbrs de vértices adyacentes
-j# mínimo común nbrs de vértices no adyacentes
-l# máximo común nbrs de vértices adyacentes
-J# máximo común nbrs de vértices no adyacentes
Como ya hemos dicho desestimamos la opción de usar memoria distribuida ya que podríamos haber saturado las redes del clúster innecesariamente. No es que nuestro programa necesite mucha comunicación, de hecho es sólo en momentos puntuales, pero si el envío de datos es muy voluminoso puede haber problemas.

Si hubiésemos tenido que trabajar con más de 60 o 100 tareas simultáneas, el uso de la memoria distribuida estaría justificado, pero como no es imprescindible tener tantas tareas simultáneas usaremos la memoria compartida.

La distribución del problema se hará a dos niveles: a la hora de generar los subespacios del problema a tratar, y a la hora de distribuirlo por las máquinas.

Para generar los subespacios del problema usaremos la opción de los script de Grid Engine -t
y la partición del comando generador en sí.

Por ejemplo si queremos dividir por 10 la carga de trabajo del geng 12 lo que tendremos que hacer será añadir en las opciones del script.

#$ -t 1-10

La opción -t asigna a cada tarea del array un valor en la variable $SGE_TASK_ID entre 1 y 10.

$ geng 12 $SGE_TASK_ID/10

Las particiones del geng no poseen el mismo rango que los índices del array de tareas, por lo que deberemos ajustar los dos valores. El rango de geng va de 0 a rango-1, y el rango de los índices de las tareas, o lo que es lo mismo, la variable $SGE_TASK_ID va de 1 a 10.

Para ello deberemos crear una variable auxiliar a la que poder volcarle el valor del índice y restarle uno.

$ export SGE_TASK_ID_AUX = ` expr $SGE_TASK_ID - 1 `
$ geng 12 $SGE_TASK_ID_AUX/10

Todo esto genera el problema geng 12 subdividido en 10 partes, las cuales se irán computando pero no necesariamente resolviendo en orden. Cuando una máquina esté ociosa, cogerá la subtarea a realizar del array con menor índice.

Las tareas se enlazan a la cola @smnodes y ésta reparte por los cores ociosos. Cuando las tareas terminan, se devuelven los datos y empiezan nuevas tareas.

3. ANÁLISIS

3.A) REQUISITOS

Los requisitos para la realización de mi proyecto son básicamente tres:

1- Conocimiento previo de las herramientas y lenguajes
2- Hardware para realizar las pruebas
3- Un tutor que me guíe en su realización

1.-Engloba todos los conocimientos necesarios para llevar a cabo el proyecto

-Isomorfismo de grafos
-Sistemas Unix
-Lenguaje C
-SGE (GRID ENGINE)

2.-Cuando hablo de hardware me refiero al clúster sobre el que vamos a realizar nuestras pruebas de isomorfismo de grafos. Para ello barajábamos varias alternativas que se comentarán más ampliamente en un capítulo posterior. Descartamos montar nuestro propio sistema Beowulf, desestimamos el uso de los servidores de Murillo, y tampoco llegamos a usar el clúster de SUN de la universidad. Al final decidimos hacer uso del centro de cálculo del CICA y de su gran infraestructura.

3.-La elección de un tutor suele ser problemática, aunque no fuese así en mi caso. Como ya antes comenté José Ramón Portillo fue quien me metió el gusanillo de la computación masiva, ha sido uno de mis mejores profesores y siempre me ha echado un cable cuando lo he necesitado.

3.B) DISEÑO

Las soluciones son varias pero a grandes rasgos podremos distinguirlas entre centralizada y distribuida.

Todo depende del tipo de problema a tratar y del enfoque que se le quiera dar.

Usaremos la solución centralizada si disponemos de los medios necesarios, ya sean un clúster privado, un clúster beowulf o una GPU, y usaremos las soluciones distribuidas si no disponemos de los medios necesarios, ya sea montando un proyecto propio en BOINC, o a traves de Ibercivis.

El tamaño del problema también es decisivo, ya que no vale la pena gastar tiempo en montar un proyecto sobre BOINC si la aplicación tiene una

duración estimada de pocos meses. Tampoco vamos a poder usar indefinidamente un clúster privado, para problemas que necesiten años en calcularse. El tratamiento que hace de la información las GPU no es aplicable a todos los problemas, de hecho solo es aconsejable para problemas altamente paralelos.

Cada problema y caso tiene su solución particular, aquí comentaremos algunos ejemplos.

Clúster

El término clúster es ampliamente usado para referirse a la interconexión de ordenadores independientes, con el objetivo de usarlos como si de una única maquina se tratase.

Llevándolo al extremo también podremos decir que la interconexión de dos o más computadoras para la resolución conjunta de un problema, también es considerado un clúster.

Los clúster pueden dividirse en tres grandes grupos:

HPC (High Performance Computing Clusters: clústeres de alto rendimiento).

Son clústeres en los cuales se ejecutan tareas que requieren de gran capacidad computacional, grandes cantidades de memoria, o ambos a la vez. El llevar a cabo estas tareas puede comprometer los recursos del clúster por largos periodos de tiempo.

HT o HTC (High Throughput Computing Clusters: clústeres de alta eficiencia).

Son clústeres cuyo objetivo de diseño es el ejecutar la mayor cantidad de tareas en el menor tiempo posible. Existe independencia de datos entre las tareas individuales. El retardo entre los nodos del clúster no es considerado un gran problema.

HA o HAC (High Availability Computing Clusters: clústeres de alta disponibilidad).

Son clústeres cuyo objetivo de diseño es el de proveer disponibilidad y confiabilidad. Estos clústeres tratan de brindar la máxima disponibilidad de los servicios que ofrecen. La confiabilidad se provee mediante software que detecta fallos y permite recuperarse frente a los mismos, mientras que en hardware se evita tener un único punto de fallos.

Estos grupos no son del todo excluyentes y por lo tanto puede existir un clúster de un tipo con características de los otros. En la actualidad la mayoría de los clúster suelen ser mixtos.

Los clústeres también pueden dividirse dependiendo de su objetivo: científico o empresarial.
Científico
-Sus aplicaciones suelen ser computacionalmente intensivas
-Necesidad de recursos muy importantes en almacenamiento y memoria
-Necesidad de nodos dedicados
-Las tareas se gestionan mediante un sistema de colas
-Está relacionado con los entornos HPC y HTC
Empresarial
-Sus aplicaciones demandan alta disponibilidad y respuesta inmediata
-Los servicios se están ejecutando continuamente
-No controlados por un sistema de colas.
-Está relacionado con el entorno HAC

Beowulf

Los clústeres Beowulf son clústeres escalables basados en equipos domésticos conectados mediante una red local, utilizando software de código abierto. El diseñador puede mejorar el rendimiento del clúster Beowulf simplemente añadiendo nuevas máquinas. El hardware puede estar compuesto por cualquier número de equipos domésticos, desde dos, interconectados mediante una LAN, ambos corriendo sistemas Linux y compartiendo el sistema de archivos, hasta 1024 nodos con comunicación de alta velocidad y baja latencia.

Su nombre se refiere principalmente a la máquina construida en 1994 por Thomas Sterling y Donald Becker, los cuales buscaban una alternativa más económica a los supercomputadores de la NASA. Su prototipo inicial fueron 16 procesadores DX4 conectados mediante Ethernet.

No hay ningún componente en particular que haga a un clúster Beowulf, pero si una serie de características. Usan sistemas operativos basados en Unix, como BSD, Linux o Solaris, normalmente construidos con código libre. Usan librerías paralelas como MPI y PVM. Estos permiten escribir programas (en C o en Fortran) basados en paso de mensajes. Hoy día los clústeres que cumplen estas particularidades suelen ser conocidos como clústeres Beowulf.

Más información, en su sitio web:
http://www.beowulf.org/

GPU

Las GPU (Graphics Processing Unit) o unidades de procesamiento gráfico, situadas comúnmente en las tarjetas graficas o de video han conseguido que los ordenadores convencionales puedan llegar a ser maquinas de cálculo muy potentes.

Las GPU fueron concebidas para resolver cálculos sobre vectores y matrices, preprocesamiento de gráficos, operaciones en coma flotante, renderizado, etc. para liberar a la CPU de todas estas tareas relacionadas con la interfaz grafica de usuario (GUI).

Con el éxito que han tenido los videojuegos en la ultima década y por el afán de que cada vez estos sean más realistas han conseguido ir mejorando y optimizando la arquitectura de las GPU. La arquitectura interna de las tarjetas graficas no ha variado mucho la última década, pero si la tecnología de las GPU. Aun teniendo una frecuencia de reloj muy inferior a las CPU actuales, 600-800 MHz frente a 3,8-4 GHz, estas poseen una arquitectura inherentemente paralela y es idónea para cálculos matemáticos o simplemente para usar fuerza bruta.

Las GPU modernas están pensadas para trabajar con vértices y pixeles, y suelen estar altamente segmentadas. Un gran número de pequeños cores completamente funcionales con sus respectivas cachés la conforman y permiten el tratamiento en paralelo de vectores y matrices.

Lo más atractivo de las GPU actuales es su capacidad para soportar lenguajes de alto nivel. La primera API usada ampliamente fue el estándar abierto OpenGL (Open Graphics Lenguage), gracias al cual Microsoft desarrollo DirectX. Las posteriores APIs han intentado acercar y facilitar la programación, siendo esta cada vez más natural, como son GLSL (OpenGL Shading Lenguage), Cg (C for graphics) o HLSL (High Level Shading Lenguage).

Aprovechar las capacidades de computo de las GPU con propósito general, o sea, no solo para el procesamiento gráfico, es denominado GPGPU o General-Purpose Computing on Graphics Processing Units.

La empresa Nvidia creó hace relativamente poco tiempo CUDA (Compute Unified Device Architecture), el cual hace referencia a un compilador y a un conjunto de herramientas de desarrollo que permiten a los programadores usar una variación del lenguaje de programación C para codificar algoritmos en GPUs del fabricante. Mediante wrappers
podremos también programar en Python, Fortran y Java.

Actualmente una gran parte de los proyectos pertenecientes a BOINC, disponen de tareas pensadas para resolverse en la CPU, pero también tareas para correrse en las GPU. El proyecto Folding@home también dispone de tareas expresamente para computarlas en tarjetas graficas que soporten CUDA. La GPU de la PS3 (PlayStation 3) también sirve para donar cálculo a Folding@home.

Algunos estudios aseguran que el rendimiento de las GPU frente a las CPU (para resolver determinados problemas) puede ser 50 veces mayor, también dependiendo de la GPU y CPU que estemos comparando. Por todo ello las GPU han encontrado su sitio dentro de la computación paralela y masiva.

GRID

La GRID o computación grid es un sistema de computación distribuido que permite compartir recursos no centralizados geográficamente. Es un nuevo concepto, usando los recursos (de computo y almacenamiento) de maquinas heterogéneas (clústeres, supercomputadores, ordenadores convencionales con diferentes arquitecturas) conectadas a través de Internet para resolver problemas de grandes dimensiones.

Es una solución muy versátil, altamente escalable, potente y flexible. El uso de los recursos a través de la red obedece a unas pautas de seguridad y políticas de gestión para evitar un uso malintencionado de estos. Los recursos se compartirán de manera uniforme, eficiente, fiable y transparente.

Las GRIDs y Peer-to-peer (P2P) tienen en común la idea básica de la compartición de recursos a través de la red. Las P2P son de un carácter más anónimo y poseen pocos mecanismos de seguridad y control de la información, frente a las GRIDs que son de carácter más jerarquizado y la información es bien controlada.

Entre sus ventajas encontramos su alta disponibilidad, su reducción de costes, su capacidad de balanceo, la posibilidad de ejecutar aplicaciones a gran escala, facilitar el acceso a recursos distribuidos, etc.

La GRID supone un gran avance respecto a la World Wide Web: mientras el segundo proporciona un acceso transparente a la información almacenada en otra parte del mundo, la GRID proporciona acceso transparente a potencia de cálculo y capacidad de almacenamiento distribuida por todo el globo.

Todos estos conceptos y beneficios son englobados en la idea de 'e-Ciencia', la cual ayudara al conocimiento en campos tan dispares como la medicina, la predicción meteorológica, la ingeniería, la nanotecnología, la genómica o la proteómica

La computación grid también presenta grandes desventajas, la comunicación es lenta y no uniforme, necesidad de desarrollo de aplicaciones para manejar el grid, organizar los modelos de explotación, las políticas de seguridad, los costes, etc.

BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) es una infraestructura para la computación distribuida, desarrollada inicialmente para el proyecto SETI@home, en la que actualmente se basan cientos de proyectos, de muy distinta índole.

En BOINC participan más de 2, 245,714 voluntarios con un total de 6, 512,032 maquinas distribuidas por 272 países. El promedio de operaciones en coma flotante por segundo aproximado es 4938 TeraFLOPS.

BOINC lo conforman todos sus proyectos, entre los cuales podemos encontrar campos como la biología, ciencias de la tierra, la criptografía, física, astronomía o las matemáticas. Pasare a explicar brevemente los proyectos más conocidos:

Rosetta@home
Ibercivis
World Community Grid
Docking@home
Malaria Control
GPUGRID.net
POEM@home
SIMAP
Climateprediction.net
Enigma@home
Einstein@home
LHC@home
Milkyway@home
QMC@home
SETI@home
Astropulse
Spinhenge@home
uFluids@home
ABC@home
PrimeGrid
SZTAKI Desktop Grid
Rectilinear Crossing Number

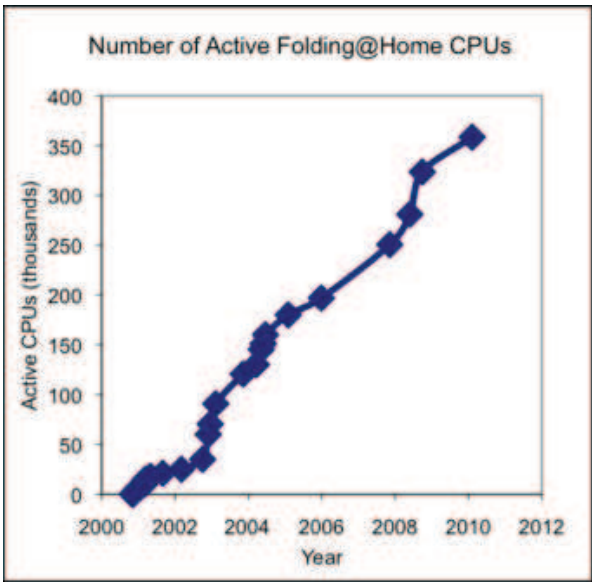
Folding

Folding@home es un gran proyecto de computación distribuida y voluntaria para realizar simulaciones de plegamiento de proteínas, usando principalmente la técnica de dinámica molecular. Actualmente es dirigido por Vijay S. Pande y su equipo de investigadores en el departamento de Química de la Universidad de Standford.

Hay que aclarar que Folding no tiene nada que ver con BOINC, utilizan un software distinto. Últimamente en Folding están trabajando para lanzar un proyecto basado en BOINC para la captación de voluntarios.

Estas simulaciones permiten a la comunidad científica entender más y mejor el desarrollo de enfermedades como el Alzheimer, el cáncer o la fibrosis quística. Con el trabajo realizado desde que el proyecto comenzó en 2000, se han podido crear más de 100 publicaciones científicas.

Folding@home es multiplataforma disponible para Windows, Linux, Mac OS e incluso para PS3. En 2006 Sony firma un acuerdo con la universidad de Stamford para incluir su software en las consolas PS3.



Ibercivis

Ibercivis es un multiproyecto basado en Boinc que da soporte a investigaciones principalmente hipanolusas. Arranco en 2008 con cuatro proyectos, y hoy da cabida a más de diez proyectos diferentes, los cuales expondré brevemente.

-Fusión
Simulaciones de plasmas que serán utilizados en el proyecto ITER (Reactor Termonuclear Experimental Internacional) para poder controlar la fusión por confinamiento magnético, la que podría ser la fuente de energía del futuro.

-Docking
Búsqueda sistemática de sustancias con efectos terapéuticos. Se centra en buscar fármacos para el tratamiento de enfermedades como el cáncer entre otros.

-Materiales
Analizan como las impurezas en materiales magnéticos modifican las propiedades del material.

-Amiloide
Se centra en la búsqueda de compuestos que puedan interferir en la formación de agregados y fibras amiloideas en las enfermedades neurodegenerativas, más concretamente la Polineuropatía Amiloide Familiar (PAF) y la enfermedad de Alzheimer.

-Neurosim
Se analizan las propiedades estructurales de los aminoácidos y de pequeños péptidos que actúan en el cerebro y en el sistema nervioso.

-Nanoluz
Investigación sobre el comportamiento de la luz en sistemas estructurados a escala nanométrica con el fin de aplicar dichos sistemas a comunicaciones y computación óptica así como a análisis biológicos basados en nanosensores.

-Adsorción
Estudian las propiedades de adsorción de las denominadas arcillas PILCS, las cuales tienen gran importancia industrial como catalizadores, materiales que almacenan gases y materiales utilizados en procesos de reparación.

-Cuanticables
Investigación sobre cómo y en qué grado afectan los defectos e impurezas que contienen los materiales en la capacidad que tiene un cable cuántico para producir la corriente.

-Sanidad
Se utilizan técnicas de simulación numérica Monte Carlo para mejorar el conocimiento, las aplicaciones y la eficiencia del uso seguro de las radiaciones ionizantes en par el diagnostico o el tratamiento en la sanidad.

-Criticalidad
Estudio de las propiedades de los sistemas desordenados y los efectos de la fractalidad de sistemas en la transición metal-aislante en los semiconductores.

Todos ellos son proyectos independiente y el voluntario integrante en Ibercivis elegirá en cuáles de ellos quiere donar potencia de computo, por ello es denominado multiproyecto.

Ibercivis también tiene como objetivo la divulgación científica entre la ciudadanía hispana, concienciándola de que también pueden ayudar a la mejoría de la calidad de nuestras investigaciones y al abaratamiento de estas.

Ibercivis fue constituido en 2008 gracias a la cooperación del Instituto de Biocomputación y Física de Sistemas Complejos de la universidad de Zaragoza (BIFI), CIEMAT, CSIC Y RedIris.

En 2009, Ibercivis se expandió por toda la península gracias a los acuerdos luso-hispano que afilian a varias instituciones portuguesas , entre ellas el Ministerio de Ciencia, Tecnología y Enseñanza Superior, el Centro de Neurociencia y Biología Celular de la Universidad de Coímbra y el Laboratorio de Instrumentación y Física Experimental de Partículas.

Actualmente lo forman 17455 voluntarios de 123 países distintos, de los cuales solo 3020 son activos .Un voluntario es denominado activo cuando han donado algo de cálculo al menos en los últimos 30 días.

La plataforma desempeña un promedio de operaciones en coma flotante por segundo de 2,371 TeraFLOPS

Para más información consultar su página web:
http://www.ibercivis.es/

3.C) IMPLEMENTACIÓN

Aquí nombraremos los conceptos básicos de la Teoría de Grafos, sin pasar a explicarlo con detalle:

Vértice: Es la unidad fundamental de los grafos.

Grado: numero de aristas incidentes

Arista: Es un par ordenado de vértices, sirven para denotar un relación entre vértices. El par ordenado indica el sentido de la arista en los grafos dirigidos, pero no en los grafos no dirigidos.

Múltiples: cuando varias aristas tienen el mismo par de vértices

Grafos: Entidades solo formadas por aristas y vértices.

Simple: no posee lazos ni aristas múltiples
Multígrafo: posee aristas múltiples.
Acíclico: No contiene ciclos

Bipartito: Sus vértices pueden ser divididos en dos conjuntos tal que no hay aristas entre vértices del mismo grupo.
Conexo: Existe al menos un camino desde cualquier par de vértices.
Dirigido/ Dígrafo: Las aristas del grafo tienen un sentido, con un vértice inicial y otro final.
Subgrafo: sus aristas son un subconjunto de aristas y los vértices son un subconjunto de vértices del grafo del que proviene.

Adyacencia: Dos vértices son adyacentes si existe una arista que los una.

Matriz de Adyacencia: Representación de las aristas de un grafo en una matriz, donde la posición (i, j) indica el número de aristas entre el vértice i y el j.

Camino: Secuencia de vértices adyacentes

Euleriano: Pasa una única vez por cada una de las aristas del grafo.

Hamiltoniano: Pasa una única vez por cada uno de los vértices del grafo.

Ciclo: Camino cerrado, que comienza y termina en el mismo vértice.

Lazos: Ciclos de longitud 1

Triángulos: Ciclo simple mínimo de longitud 3

La representación en papel de un grafo no puede ser confundido con la idea abstracta del grafo, ya que hay muchas formas de representación para un único grafo. A este fenómeno se le denomina Isomorfismo de Grafos.

Dos grafos son isomorfos si poseen exactamente la misma estructura interna.

Con el uso de nauty utilizaremos un formato de ficheros llamado Graph6. Graph6 es un formato para la representación y almacenamiento de grafos. Utiliza únicamente los caracteres ASCII imprimibles y codifica la matriz de adyacencia en los últimos 6 bits de cada byte. Su uso está limitado a grafos no dirigidos.

En un fichero con datos de grafos codificados en Graph6 hay únicamente un grafo por líneas.

La problemática de abarcar un problema grande es que aun distribuyendo el trabajo pueden quedar tareas ingentes también irresolubles por su gran

tamaño o complejidad.

Los dos grandes problemas que pueden aparecer a la hora de atacar uno de estos problemas son el tiempo y el espacio.

Si el tiempo necesario es lo suficientemente dilatado, no sabremos si tardara mucho en parar de calcular, o si le faltara poco. Es como el típico problema del programa HALT, si es la primera vez que corres un programa, no tienes ninguna referencia para poder saber cuánto duraran sus tareas. Esto obliga a acotar bien el espacio de soluciones y no intentar conseguir abarcar problemas lo suficientemente grandes.

El espacio es crítico a la hora de llevar a cabo un cálculo grande, si no dispones de la memoria suficiente o no troceas lo suficiente el problema, la memoria se verá desbordada y el resultado será erróneo.

Cuando decimos que el tiempo es una problemática, no es algo trivial, aparte de que se extienda mucho o no el cálculo, existen problemas en los que los parámetros de entrada se van actualizando constantemente, como por ejemplo lo son algunos modelos meteorológicos que toman por valores las mediciones tomadas en tiempo real, para así aportar simulaciones de lo que puede acontecer con la climatología.

Si el tiempo de cálculo es elevado y el programa posee gran parte del código paralelizable, una buena opción es distribuirlo. Si el código posee muy pocas partes paralelizables, no aumentaría mucho su rendimiento intentar paralelizarlo y distribuirlo, como bien nos aconseja la Ley de Amdahl.

También pensar en el tiempo de vida humano. Algunos problemas que se han podido solucionar mediante la GRID habrían podido durar más de 100 años corriéndose en la maquina más potente posible. En apenas un tres de años se ha llegaron a soluciones con una ejecución total de 3748 años, este caso es del proyecto concluso Genome Comparison de WCG.

WCG (World Community Grid) es un multiproyecto patrocinado por IBM que viene dando soporte desde el 2004 a investigaciones relacionadas con algún tipo de beneficio para la humanidad, o al menos eso aseguran en su web. Proyectos como Human Proteome Folding Fase1 el cual ayudo a predecir las estructuras de las proteínas de los humanos o Influenza Antiviral Drug Search para intentar conseguir algún compuesto que inocua o destruya el virus de la gripe. WCG tiene acumulado un volumen de ejecución total (entre todos sus proyectos) equivalente a 499.705 años, de los cuales 7822 han sido donados por voluntarios adscritos como españoles.

La ejecución de nuestras pruebas es algo más discretas y manejables. Nuestros cálculos podrían correrse durante semanas o incluso meses en el CICA, pero si no dispusiéramos de esa capacidad de cómputo y tuviéramos

que correrlo de modo secuencial tardaríamos años en simplemente saber si el programa ha terminado o no.

La ejecución del geng 12 en una sola maquina puede durar varios días, y una prueba con un tamaño mayor sería inviable. Un ordenador personal si tiene que estar varias semanas calculando puede llegar a sobrecalentarse y reiniciarse, o simplemente que se vaya la luz para que el cálculo se interrumpa o se corrompa. Por ello también es necesario cuidar los niveles de temperatura y humedad de las maquinas cuando vamos a hacer un uso intensivo de ellas. Haciendo uso del clúster del CICA nos olvidamos de esa problemática.
El espacio es fundamental para algunos problemas, como el nuestro que necesita generar todo el abanico de posibilidades de grafos con N vértices. Si hablamos de archivos que pesan cientos de terabytes (TB) la cosa se complica.

Hace falta distinguir que el espacio está distribuido en varios niveles, el nivel de caché, el nivel de memoria principal y la memoria en disco. También señalar que entre los niveles de antes para las computadoras actuales también utilizan varios niveles de caché, y comparten la memoria principal de otros nodos.

Nuestro proyecto hace un uso masivo de la memoria en disco, por lo que de los otros niveles nos despreocuparemos.

Las pruebas realizadas con el tamaño del problema 12, generan un archivo que pesa 2,54TB. Si hubiéramos querido correrlo en una única maquina de modo secuencial tendríamos serios problemas para almacenar tal cantidad de datos.

Cuando hablamos después de las pruebas de tamaño 15 podemos ver como incluso las subtareas llegan a generar ficheros de 500GB. Con 4 subtareas simultáneas serian archivos de 2TB y así sucesivamente.

Los clústeres suelen tener del orden de varios cientos de TB, como el Tianhe- 1A de China que tiene 224 TB de memoria.

Si hablamos de la cantidad de memoria principal esta depende también de la cantidad de cores de los que disponga el clúster y de su tecnología, normalmente suele ser 2 o 4 GB por core.

La caché viene totalmente ligada con la tecnología empleada en cada microprocesador, y hay que conocer en profundidad la arquitectura para poder sacarle todo su potencial.

4.- MANUAL DE USUARIO

SSH

Secure SHell o SSH es un protocolo de red que permite el intercambio de archivos entre dos dispositivos, de forma segura y encriptada. Usada principalmente en entornos Linux y Unix, fue diseñado para sustituir a Telnet y otras comunicaciones inseguras, las cuales usan texto plano para comunicarse, y por tanto son susceptibles de ser interceptadas y analizadas.

Para invocarlo:

$ssh [user@]hostname [comando]

El parámetro opcional usuario sirve para indicar la cuenta de usuario a la que se quiere conectar en el equipo remoto, de no especificarse ningún usuario se empleará el que se está usando para invocar al cliente de ssh.

El uso del comando también es opcional, sin añadir ninguno, lo que recibiremos es el shell de la máquina accedida.

Vamos a ver los mecanismos y fundamentos básicos usados en una comunicación ssh.

1.-El usuario (cliente) hace una primera conexión TCP y manda su "username" a la maquina destino (servidor).

[cliente]-----(username)----->[servidor]

2.-El demonio del ssh en el servidor ("sshd") responde al usuario con una petición de password.

[cliente]<-----(¿password?)-----[servidor]

3.-El ssh cliente pide al usuario que introduzca su password, la cual es enviada encriptada al servidor (donde se comparan)

[cliente]-----(password)----->[servidor]

4.-Si la password del usuario coincide, el acceso al sistema está garantizado , de forma bidireccional, y mediante un canal seguro (usando para ello un socket dedicado a dicha comunicación)

[cliente]<-----(canal seguro)----->[servidor]

Es necesario mencionar que las comunicaciones ssh están basadas en el principio de autentificación "desafío-respuesta" (Challenge-response authentication). Para que la transmisión de información no sea interceptada tan fácilmente, ésta se manda encriptada (normalmente mediante RSA) usando un sistema de autentificación de clave publica/privada.

Ahora veremos el mismo esquema antes explicado, pero más cercano a la realidad, de la comunicación ssh:

1.-El cliente hace una petición de desafío al servidor(mandándole su "username").

[cliente]-----(username)----->[servidor]

2.-El servidor desafía al cliente a partir de su clave pública

[cliente]<-----(desafío-reto)-----[servidor]

3.-El cliente mediante su clave privada y su password será capaz de resolver el desafío, mandando su respuesta ya encriptada(*)

[cliente]-----(respuesta desafío-reto)----->[servidor]

4.-El servidor recibe la respuesta del desafío, y si corresponde, crea un socket dedicado para dicha comunicación, llamado SSH_AUTH_SOCK

[cliente]<-----(canal seguro)----->[servidor]

Cabe mencionar que la clave privada NUNCA se manda, lo que se manda es una respuesta al desafío a partir de dicha clave.

Una de las ventajas de usar claves públicas frente al uso de contraseñas de usuario es que no tenemos que recordar nada más que una única frase, la frase con la que hemos cifrado nuestra clave privada. Aun así, si tenemos que logearnos en distintas máquinas o cada poco tiempo, esta tarea se convierte en tediosa. Por ello se creó el agente ssh(ssh-agent). Dicho agente gestiona los sucesivos accesos ssh a los host destino, teniendo que logearse sólo la primera vez.

Al ejecutar el ssh-agent, este crea un socket y establece la variable de entorno SSH_AUTH_SOCK con el nombre del socket. Por razones de seguridad los permisos del socket son ajustados para que tan sólo el usuario actual pueda acceder al socket. Además, el agente también crea la variable de entorno SSH_AGENT_PID y establece su valor con su PID.

Para añadir claves usamos el comando

$ssh-add

Si se ejecuta sin argumentos intentará añadir los archivos ~/.ssh/id_rsa, ~/.ssh/id_dsa y ~/.ssh/identity

Para usar este tipo de comunicación debemos efectuar 2 pasos:

1.-Generar el par de claves (público/privada)

$ssh-keygen -t rsa

(nos pedirá el nombre de la clave, el cual podremos dejar en blanco para poder acceder por defecto sin añadir clave)
(genera en ~/.ssh/id_rsa.pub la clave pública y en ~/.ssh/id_rsa la clave privada)
2.-Instalar las claves

La clave privada debe estar en la máquina cliente (~/.ssh/id_rsa), y la pública en el servidor (~/.ssh/authorized_keys). Este archivo puede tener más de una clave pública, de esta manera, muchos usuarios (cada uno con su clave privada) podrán ingresar con el mismo usuario en el servidor.

$cat ~/.ssh/id_rsa.pub | ssh user@server 'cat - >> ~/.ssh/authorized_keys

PuTTY

El uso de comunicaciones SSH tiene la ventaja de conectar dos maquinas aunque estas no posean el mismo Sistema Operativo. Por ello , para administrar un clúster no hace falta personarse en la infraestructura, solo necesitas una conexión a internet y un sistema Unix, o en su defecto, conexión a internet , una máquina con Windows y el programa PuTTY. Ahora hablaremos de este último software, PuTTY.

Es un cliente SSH, Telnet, Rlogin y TCPraw de código libre. Al comienzo solo fue disponible para los Windows y hoy en día ya lo podemos encontrar para plataformas como MacOS o incluso Symbian.

El nombre de dicho software proviene de la conjunción de dos términos, Pu: Port unique y TTY: Terminal Type. En español seria, "Puerto único para tipos de terminal"

Su última versión estable (PuTTY 0.60) consta de un ejecutable bastante ligero descargable desde

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Aunque tiene muchas cualidades y características destacables, nosotros solo nos centraremos en su uso para establecer comunicaciones SSH. Su manejo es sencillo: se lanza el ejecutable descargado en el link antes mencionado, en la pestaña Session abierta por defecto la primera, nos encontramos con un recuadro para especificar el hostname o la IP de la maquina a la que queremos conectarnos , el tipo de comunicación a establecer, y el puerto al que conectarse (por defecto el puerto 22 para SSH). También tendremos en la misma ventana un recuadro donde recordar y guardar las conexiones vayamos a usar en un futuro.

Probando el programa he tenido problemas al conectarme directamente a mi cuenta en pool.cica.es, ya que no puedes conectarte desde una maquina con IP dinámica. Para conectarme primero tengo que tunelar hacia un nodo puente, sesamo.cica.es , el cual me sirve como máquina con IP fija y así poder tunelar hasta pool.cica.es. El acceso remoto era factible pese a que la conexión se resintiera al ser una doble conexión SSH.

Tampoco he conseguido activar la interfaz gráfica (modo -X del ssh) al tunelar, por lo que he tenido que desarrollar el proyecto mediante consola.

Para más información consultar la web:
http://www.chiark.greenend.org.uk/~sgtatham/putty/

SGE (GRID ENGINE)

SGE (Sun Grid Engine) es un gestor de colas de código abierto desarrollado por SUN. Está concebido para manipular procesos distribuidos y es multiplataforma. SGE se encarga del envío y gestión de la ejecución remota y distribuida de gran número de tareas ,estas tareas pueden ser secuenciales, paralelas o interactivas.

Este es el software que utilizan en el CICA, por lo tanto queda desestimada la alternativa de usar Condor como posible gestor de colas.

Esta es la estructura de computación del cica:

devel.cica.es

Está destinada a la compilación de las aplicaciones que el usuario
quiera tener en su /home, dispone de las librerías necesarias y compiladores
adecuados para este propósito.

pool.cica.es

Actúa como cabecera del gestor de colas del clúster Sun Grid

Engine (SGE). En esta máquina únicamente está permitido lanzar tareas al propio de gestor de colas, así como su monitorización. La ejecución de cualquier otra aplicación, compilación o tarea quedará terminantemente prohibida por motivos de seguridad e integridad de la máquina.

Actualmente disponen de 90 máquinas dual core, junto con varios servidores de memoria compartida y un clúster Bull con Infiniband como dispositivo de comunicación entre los nodos.

En total se dispone de 512 cores y aproximadamente 700 GB de memoria RAM disponible. Cada una de las máquinas está definida en una cola de ejecución de SGE.

Hay configuradas 4 colas principales:

- eca_short: 30 servidores dual-core destinados a trabajos de corta duración.

- eca: es la cola por defecto del CICA y dispone de 156 procesadores para tareas paralelas o de larga duración.

- smnodes: servidores de memoria compartida, 2 servidores SUN x4600 (16 y 24 cores) y otros 2 servidores de 8 cores. En total dispone de 256GB de memoria RAM para tareas que requieran gran cantidad de memoria.

- ibnodes: es la cola de ejecución del clúster BULL (el cual posee para su comunicación una conexión Infiniband).

Para la ejecución con SGE se necesita un script escrito en lenguaje Shell (bash, csh, etc..). La única diferencia es una serie de opciones que se añaden en el mismo script, todas comenzando con los caracteres “#$” .

Para su uso viene bien recordar las instrucciones básicas :

$qmon	-despliega el entorno gráfico de la aplicación
$qsub tarea.sh	-lanza una tarea
$qstat	-muestra el estado de las colas
$qhost	-despliega información del nodo
$qdel	-cancelar una tarea

Un ejemplo para lanzar una aplicación sería:

$qsub [-cwd] [-v VAR] [-o path] [-e path] [-M correo] [-l recursos] tarea.sh

Para que sea más cómodo se añaden esas opciones en el script:

$qsub tarea.sh

Siempre es posible llamar a las páginas del manual para ver una descripción más detallada del comando.

$man qsub

A partir de ahora podrá ver el estado de la tarea:

$qstat -u usuario -observar el estado de la tarea

$qstat -f -despliega información más detallada de tareas y colas

$qstat -g c -observar el estado de las colas de ejecución

$qdel id_tarea -eliminar la tarea seleccionada

Lustre es un sistema de ficheros compartido, propiedad de SUN, que proporciona una gran escalabilidad y alto rendimiento. Está especialmente optimizado para sistemas que manejen un gran volumen de datos. En la actualidad disponen de 9.8 TB de almacenamiento, siendo visible por todas las máquinas del clúster.

Para acceder a los recursos de supercomputación debe realizarse una conexión ssh a la máquina correspondiente (pool.cica.es o devel.cica.es, dependiendo del objetivo).
En los sistemas Linux, hay un cliente ssh integrado. Si desea realizar una conexión desde un sistema Windows, recomendamos el uso de PuTTY.

Para realizar una conexión:

$ssh usuario@pool.cica.es [devel.cica.es]

Actualmente el antes conocido como Sun Grid Engine ahora es Oracle Grid Engine, a causa de la compra de SUN por parte de Oracle.

En 2010, Oracle ya no adjuntaba el código fuente en sus actualizaciones del software Grid Engine, y por ello la comunidad Grid Engine comenzó con un nuevo proyecto, el Open Grid Scheluler, para continuar desarrollando y manteniendo una implementación libre de Grid Engine.

En Enero de 2011, la compañía Univa anunció la contratación de varios ingenieros del antiguo equipo de Sun Grid Engine, con el objetivo de

desarrollar su versión forkeada de Grid Engine, Univa Grid Engine. Ésta incluirá soporte técnico y comercial, con la que Univa espera competir con la versión oficial de Oracle Grid Engine.

Para más información:

Open Source Grid Engine Community:
http://gridengine.org/blog/

Oracle Grid Engine
http://www.oracle.com/technetwork/oem/grid-engine-166852.html

Ganglia

Una vez que la tarea esté en ejecución, disponemos de varios mecanismos para monitorizar el estado de la tarea (memoria consumida, posibles errores, carga de CPU utilizada..).

Aparte de las ya mencionadas, para poder observar el estado general del clúster, está a disposición un sistema de monitorización por Ganglia, dentro de la página:

http://cube.cica.es

En dicha página podemos ver el estado de todas las máquinas del clúster, así como el uso que se hace de cada una de forma individual, pinchando en el nodo correspondiente.

Ganglia es un software distribuido para monitorizar sistemas de computación de alto rendimiento. Permite al usuario ver de forma remota estadísticas en tiempo real como carga de trabajo, uso de CPUs, procesos corriéndose, etc.

Su última versión estable, la 3.1.7, se lanzó en marzo del 2010 y está bajo licencia BSD. Es multiplataforma y está escrito en C, Perl, PHP y Python.

WWW.GANGLIA.INFO

CICA

El Centro Informático Científico de Andalucía, CICA, se creó en el 1989. Actualmente depende de la Dirección General de Investigación, Tecnología y Empresa de la Junta de Andalucía, el cual da servicio a la comunidad investigadora de Andalucía.

Las actividades que se realizan en el CICA persiguen principalmente tres objetivos:

1)Potenciar y proporcionar herramientas para la investigación
2)El fomento de la e-Ciencia
3)Tratar de acercar estas tecnologías al mayor número posible de usuarios, tanto en lo relativo a los recursos físicos como en servicios y actividades formativas o de divulgación

Su centro de cálculo, dispone de sendos clústeres de memoria distribuida y compartida.

Actualmente, las actividades principales que realiza el Centro son las siguientes:

-La coordinación y mantenimiento de la red RICA.

-Dentro del apoyo al software libre y de fuentes abiertas: instalación, mantenimiento y administración de la Forja de Software Libre del proyecto nacional encargado por la CRUE-TIC SL a RedIRIS.

-Implantación en la red RICA de los siguientes nuevos servicios: Acceso WiFi en el marco europeo universitario (Eduroam), IPv6, Sistema de Videoconferencia, conexión a RedIRIS con 10 Gbps y escritorio remoto (FreeNX).

-Desarrollo de una PKI (Infraestructura de Clave Pública) para IrisGRID.

-Explotación y mantenimiento de la aplicación “Servicio de Información Científica de Andalucía” (SICA) de la Secretaría General de Universidades, Investigación y Tecnología.

-Colaboración en el desarrollo del DataWarehouse de la Secretaría General de Universidades, Investigación y Tecnología.

-Explotación y mantenimiento de la aplicación “Distrito Único Universitario” de la Secretaría General de Universidades, Investigación y Tecnología.

Despegue de la tecnología GRID para el cálculo masivo usando la red como medio de transporte de los flujos de ejecución de las partes en las que un mismo programa es paralelizado, y la necesidad de impulsar esta tecnología en los distintos centros de investigación ubicados en Andalucía, así como colaborar en estos temas con las Universidades Andaluzas, ha hecho que en el centro se estén desarrollando los conocimientos necesarios para una implantación y coordinación de estos temas a nivel Andaluz, gestionándose actualmente la pertenencia al proyecto europeo gestionado por el CERN EGEE (Enabled Grid for E-SciencE) y a IrisGRID.

-Colaboración con el IAA (Instituto de Astrofísica de Andalucía) en los cálculos asociados al proyecto del satélite Corot de astrosismología y búsqueda de planetas extrasolares.

-Administración del Centro de Excelencia JAVA de la Junta de Andalucía.

-Mantenimiento y gestión de las aplicaciones del servicio de RED.

-La gestión y mantenimiento de las bases de datos referenciales de información científica, y su enlace con las revistas electrónicas adquiridas por las universidades y con los catálogos de las bibliotecas de las universidades andaluzas.

-Instalación y mantenimiento del sistema informático que alberga el Catálogo Colectivo del Consorcio de Bibliotecas Universitarias de Andalucía.

-Gestión de incidentes de seguridad en la Red RICA.

-Participación en proyectos con diferentes Consejerías de la Junta de Andalucía y del Ministerio de Educación y Ciencia.

La Red RICA (Red Informática Científica de Andalucía)

Los Centros de Investigación de Andalucía están conectados entre sí y con el exterior mediante una red homogénea de comunicaciones de alta tecnología: la red RICA (Red Informática Científica Andaluza). Integrada dentro de la red académica española RedIRIS (Interconexión Recursos InformáticoS), RICA forma parte de Internet y ofrece a sus usuarios acceso a servicios distribuidos a nivel global.

A lo largo del territorio de Andalucía, RICA ofrece una serie de puntos de acceso que permiten la conexión de cualquier centro a la red; siendo responsabilidad del centro en cuestión el transporte de la señal desde su localización al punto de acceso a la red. En general, los puntos de acceso están ubicados en las Universidades de la Comunidad Autónoma.

El CICA (Centro de Informática Científica de Andalucía) gestiona y coordina esta red, así como los servicios que a través de ella se prestan. Es, además, el nodo de la RedIRIS en Andalucía.

Para más información consultar su sitio web:
http://www.cica.es/
http://eciencia.cica.es/

5.- PRUEBAS

No todo el camino se plasmará en el presente proyecto, pero aún así aquí lo mencionaré de pasada.

En un comienzo mi proyecto iba a ser la transcripción y testeo de un algoritmo secuencial en uno distribuido haciendo uso de la infraestructura de la aula SUN de la escuela. El algoritmo a distribuir era el de Tutte, un famoso problema de Teoría de Grafos que no pasaré a explicar.

Estuve un tiempo intentando hacer uso del clúster del aula de SUN , pero nunca conseguí llegar a correr ningún trabajo. Terminamos por desestimar el uso de un clúster que daba más problemas de los que resolvía. Siempre nos quedaba la alternativa de usar las infraestructuras del CICA.

Con el código que iba a paralelizar, necesitaba trabajar sobre el código secuencial y funcional, digo funcional porque el proyecto sobre el que iba a trabajar era un supuesto proyecto terminado, subido a REDiris, el cual estaba cojo. Todas las rutas especificadas en el Make eran rutas relativas, por lo tanto solo útiles en el ordenador del alumno del proyecto. El proyecto está incompleto por eso desestimamos trabajar sobre él y elegir otro proyecto más viable, en este caso elegimos un proyecto sobre cálculo intensivo de isomorfismo de grafos.

Prueba 1

Nuestra primera prueba ha sido realizando la versión del problema de tamaño 12 subdividido en 200 tareas. A continuación mostraremos el script usado para ello:

```
#$ -S /bin/bash
#$ -N GENG12
#$ -e GENG12.err
#$ -o GENG12.out
#$ -q smnodes
#$ -V
#$ -cwd
#$ -t 1-200

##accedemos al directorio correspondiente

cd /home/josera/resultados_geng12/

##restamos uno a la variable SGE_TASK_ID de rango (1:200) para que la variable auxiliar tenga el rango (0:199)
```

```
export SGE_TASK_ID_AUX=`expr $SGE_TASK_ID - 1`

##se calcula cada trozo del geng y lo vuelca en un archivo temporal, también manda el resumen del resultado a .subres

/home/josera/nauty22/geng 12 $SGE_TASK_ID_AUX/200 geng12_$SGE_TASK_ID.tmp &> geng12_$SGE_TASK_ID.subres

##concatenamos los resúmenes de los resultados parciales en .finalres

cat geng12_$SGE_TASK_ID.subres >> geng12.finalres

##cribamos el fichero temporal parcial con pickg y lo vamos concatenando en el fichero final .pickg

/home/josera/nauty22/pickg -r geng12_$SGE_TASK_ID.tmp >> geng12.pickg

##borramos los archivos generados intermedios

rm geng12_$SGE_TASK_ID.tmp geng12_$SGE_TASK_ID.subres
```

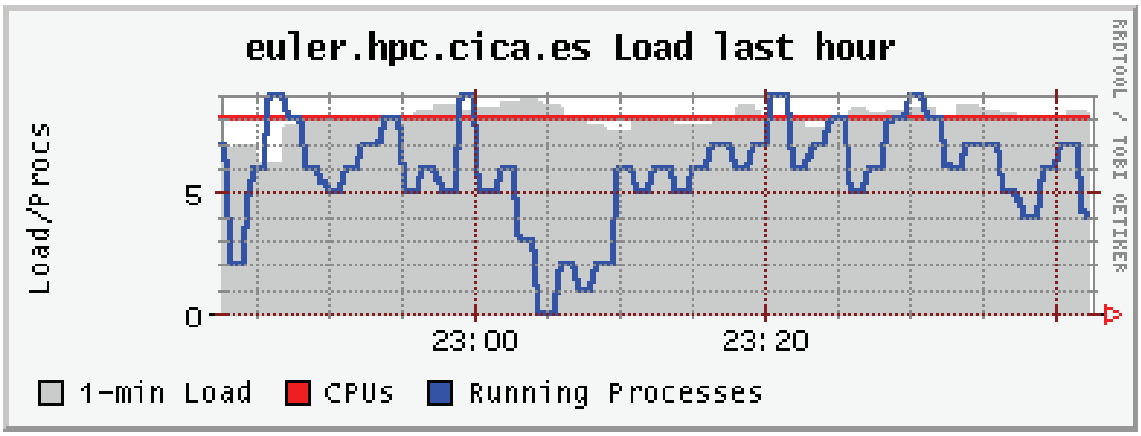
Hemos tenido que restarle una unidad a la variable $SGE_TASK_ID y guardarlo en $SGE_TASK_ID_AUX para poder incluirla dentro de la instrucción geng, ya que en esta van de 0 a rango-1 y las variables $SGE_TASK_ID son asignadas desde 1 hasta 200 como se ve en la opciones del script.

```
#$ -t 1-200
```

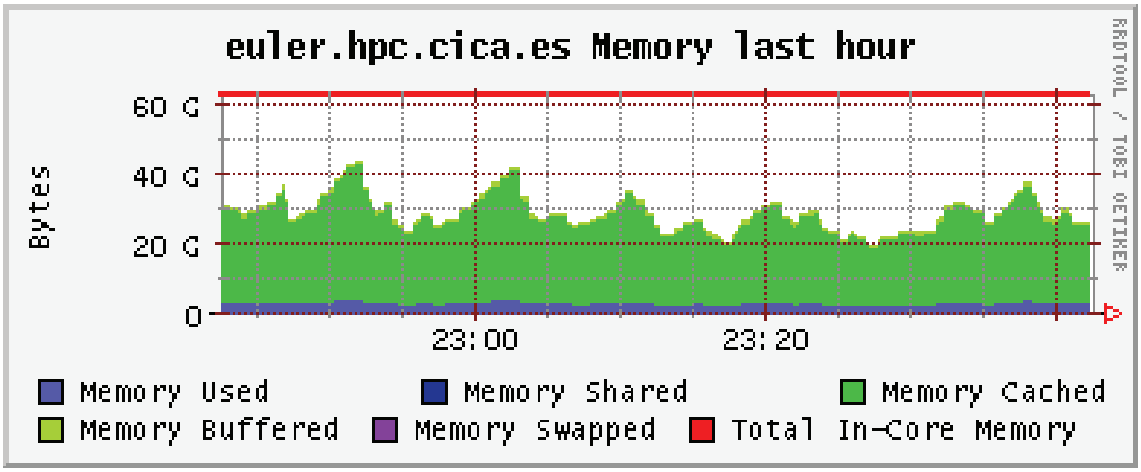
En esta prueba se han calculado las subtareas en grupos de 8, y sus correspondientes archivos intermedios '.tmp' han pesado del orden de 9 a 13 GB, ocupando no más de 110GB en los momentos de más alta capacidad. Hay que tener en cuenta que si hubiésemos intentado resolverlo de modo secuencial ,se habría generado un archivo intermedio de aproximadamente 2,5TB. Se han tenido que calcular 200/8= 25 remesas de 8 subproblemas cada una.

La ejecución total del problema nos ha llevado 20 horas conseguirla, por lo que la estimación del tiempo necesario para correrlo de modo secuencial (en una máquina con las mismas cualidades) sería de alrededor de unos 7 días.

Estas son varias gráficas sacadas con el sistema de monitorización Ganglia



Vemos cómo en algunos momentos los procesos ejecutándose descienden hasta 0, un problema que deberemos subsanar en un futuro, ya que deberíamos aprovechar el máximo tiempo ocioso de los cores.



Después de correr la prueba hemos ejecutado la siguiente instrucción para ver cuántos grafos de todo el conjunto de posibilidades, pasan la criba

$cat -n geng12.pickg

La solución final consta de 19002 grafos

Prueba 2

En nuestra segunda prueba hemos realizando la versión del problema de tamaño 12 subdividido en 2000 tareas. El script escrito para ese fin es este:

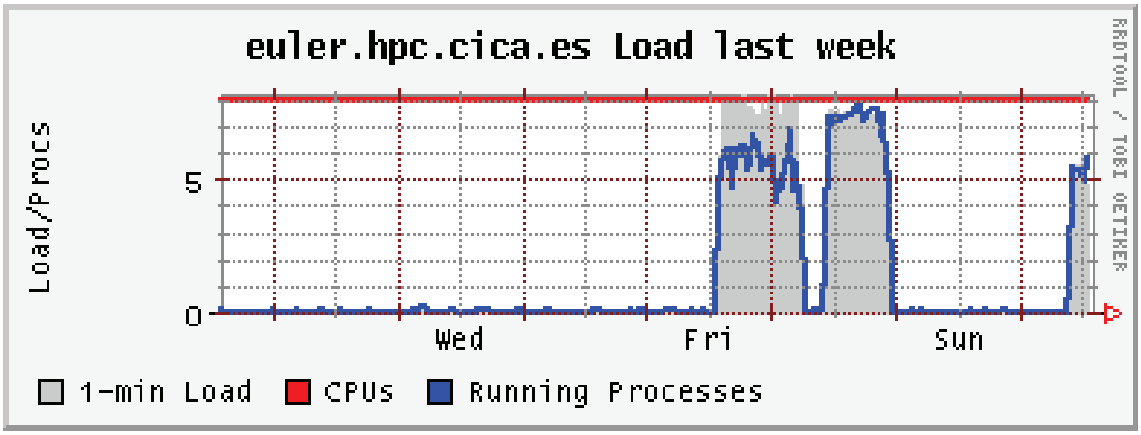
```
#$ -S /bin/bash
#$ -N GENG12.2
#$ -e GENG12.2000.err
#$ -o GENG12.2000.out
#$ -q smnodes
#$ -V
```

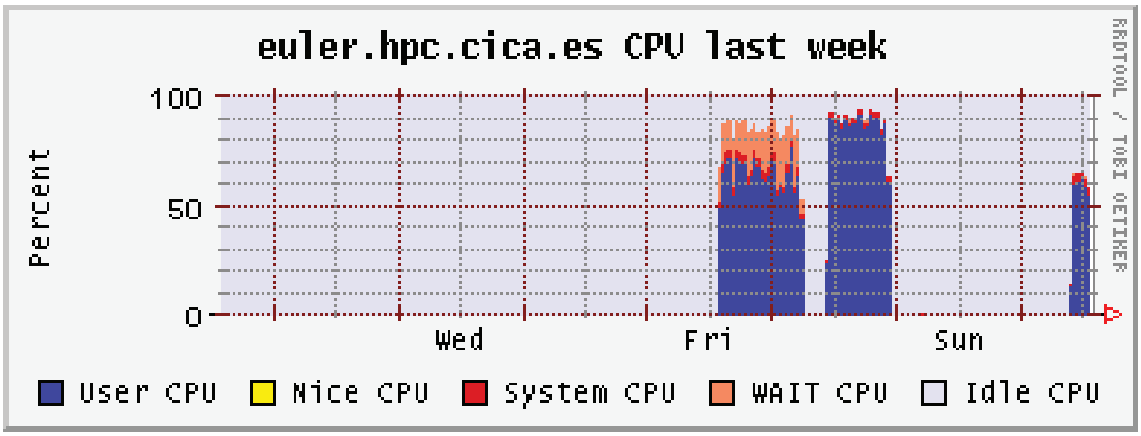
```
#$ -cwd
#$ -t 1-2000

##accedemos al directorio correspondiente

cd /home/josera/resultados_geng12.2000/

##restamos uno a la variable SGE_TASK_ID de rango (1:2000) para que la variable auxiliar tenga el rango (0:1999)

export SGE_TASK_ID_AUX=`expr $SGE_TASK_ID - 1`

##se calcula cada trozo del geng y lo vuelca en un archivo temporal, también manda el resumen del resultado a .subres

/home/josera/nauty22/geng 12 $SGE_TASK_ID_AUX/2000 geng12_$SGE_TASK_ID.tmp &> geng12_$SGE_TASK_ID.subres

##concatenamos los resúmenes de los resultados parciales en .finalres

cat geng12_$SGE_TASK_ID.subres >> geng12.finalres

##borramos los subresúmenes

rm geng12_$SGE_TASK_ID.subres

##cribamos el fichero temporal parcial con pickg y lo vamos concatenando en el fichero final .pickg

/home/josera/nauty22/pickg -r geng12_$SGE_TASK_ID.tmp >> geng12.pickg

##borramos el archivo temporal intermedio

rm geng12_$SGE_TASK_ID.tmp
```

En esta prueba hemos aumentado el número de particiones para el mismo problema planteado en la primera prueba, de 200 a 2000 particiones.

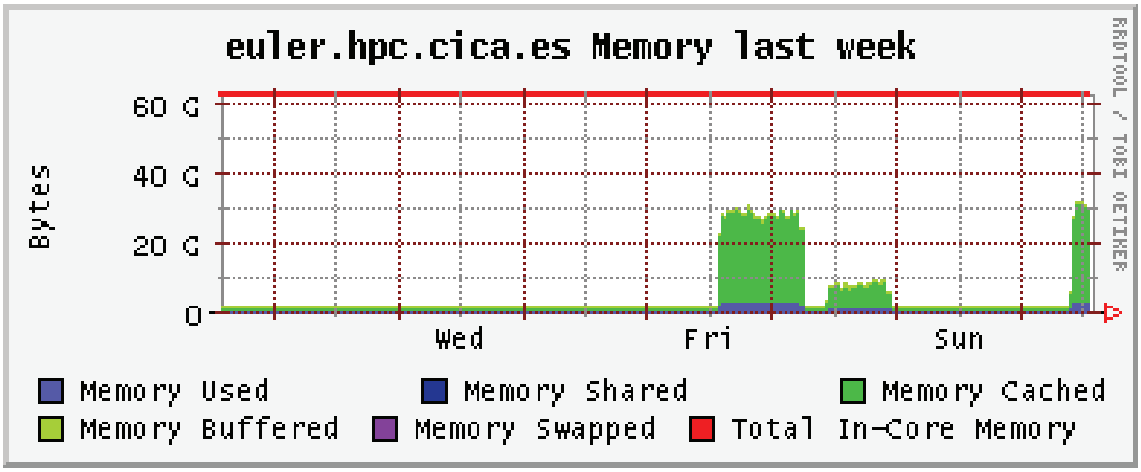
El tamaño de los ficheros intermedios (.tmp) generados son 10 veces más pequeños, en correspondencia con el nuevo particionamiento. Los ficheros pesan de de 800MB a 1200MB.
Haciendo así que la memoria usada en los momentos de más alta capacidad (exactamente justo después de generar todo el espacio de posibles grafos) no es superior a 10GB.

En esta segunda prueba también hemos hecho uso de 8 cores, por lo que se han tenido que calcular 2000/8= 250 remesas de 8 subproblemas cada uno. Pero aun siendo 10 veces más el número de conjuntos de subtareas a solucionar, la ejecución total ha durado menos tiempo llevarla a cabo que en la primera prueba, tal y como se aprecia en la siguiente imagen . Los dos primeros grandes eventos corresponden con la primera y segunda prueba respectivamente.

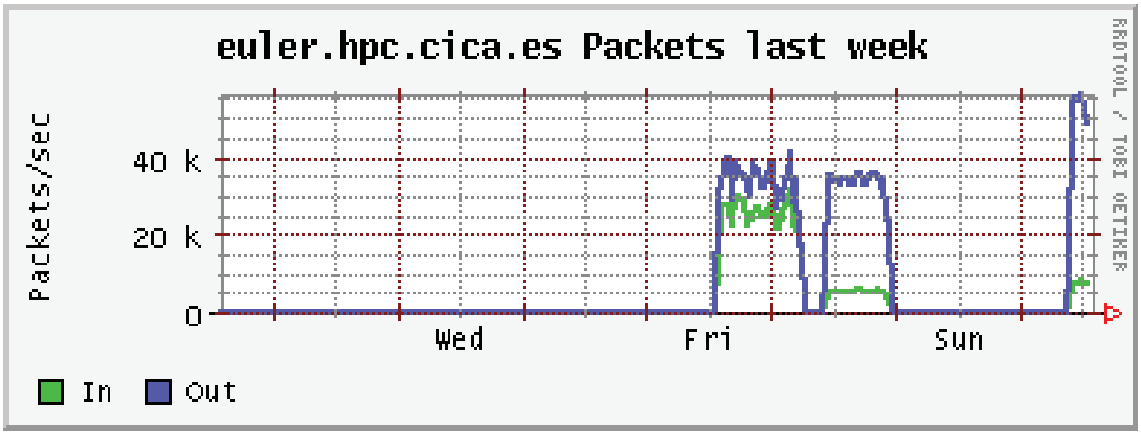




Aquí podremos comparar las dos pruebas, donde vemos que esta segunda se ajusta mucho mejor al problema dado y a los cores usados. Vemos cómo se hace un mayor aprovechamiento de los procesadores, y se aprovecha mejor el tiempo de uso de las máquinas, acortando así la ejecución total.



También al hacer 10 veces más particiones, vemos cómo la memoria caché usada es mucho menor. Hace menor y mejor uso de la memoria esta prueba en comparación con la primera.
El tiempo total de ejecución ha sido de 8 horas , en comparación con las 10 horas necesarias para completar la primera prueba.



Hace un uso menor y más uniforme del ancho de banda, sobrecargando las comunicaciones en mucha menor medida.

Prueba 3

La presente tercera prueba ha sido la versión del problema de tamaño 15 subdividido en 2000 tareas. Este es el script usado para ello:

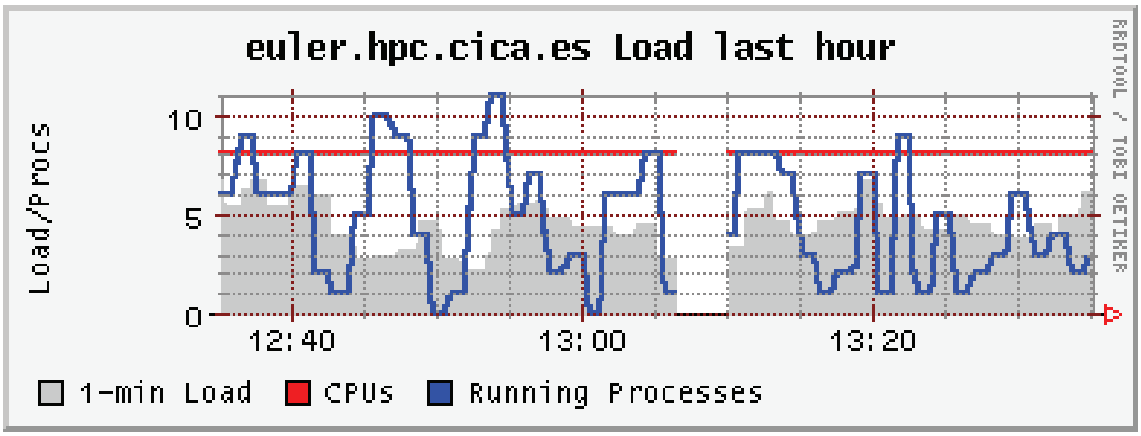
```
#$ -S /bin/bash
#$ -N GENG15
#$ -e GENG15.2000.err
#$ -o GENG15.2000.out
#$ -q smnodes
#$ -V
#$ -cwd
#$ -t 1-2000
```

```
##accedemos al directorio correspondiente

cd /home/josera/resultados_geng15.2000/


#calculamos la variable auxiliar

export SGE_TASK_ID_AUX=`expr $SGE_TASK_ID - 1`

##se calcula cada trozo del geng y lo vuelca en un archivo temporal. También manda el resumen del resultado a .subres.

/home/josera/nauty22/geng 15 $SGE_TASK_ID_AUX/20000 geng15_$SGE_TASK_ID.tmp &> geng15_$SGE_TASK_ID.subres

##concatenamos los resúmenes de los resultados parciales en .finalres

cat geng15_$SGE_TASK_ID.subres >> geng15.finalres

##borramos los subresúmenes

rm geng15_$SGE_TASK_ID.subres

##cribamos el fichero temporal parcial con pickg y lo vamos concatenando en el fichero final .pickg

/home/josera/nauty22/pickg -r geng15_$SGE_TASK_ID.tmp >> geng15.pickg

##borramos el archivo temporal intermedio

rm geng15_$SGE_TASK_ID.tmp
```

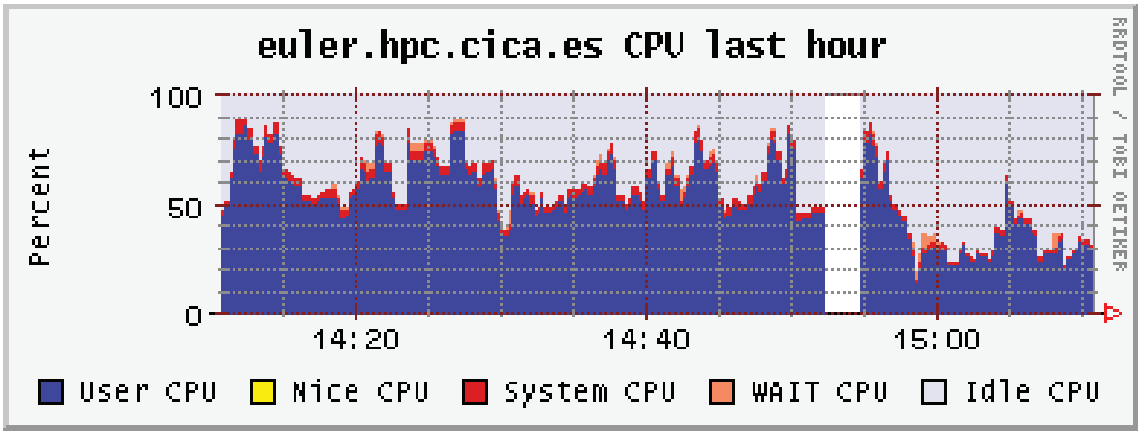
En esta prueba nos hemos lanzado a intentar calcular un espacio de soluciones mucho mayor. Esta vez los grafos generados serán todos los posibles con 15 vértices.

En esta prueba hemos tenido serios problemas a la hora de almacenar tal ingente cantidad de datos, de hecho hemos tenido que parar la ejecución del programa cuando la los archivos temporales pesaban de 300 a 500GB, y la memoria total usada ya superaba los 3 TB.

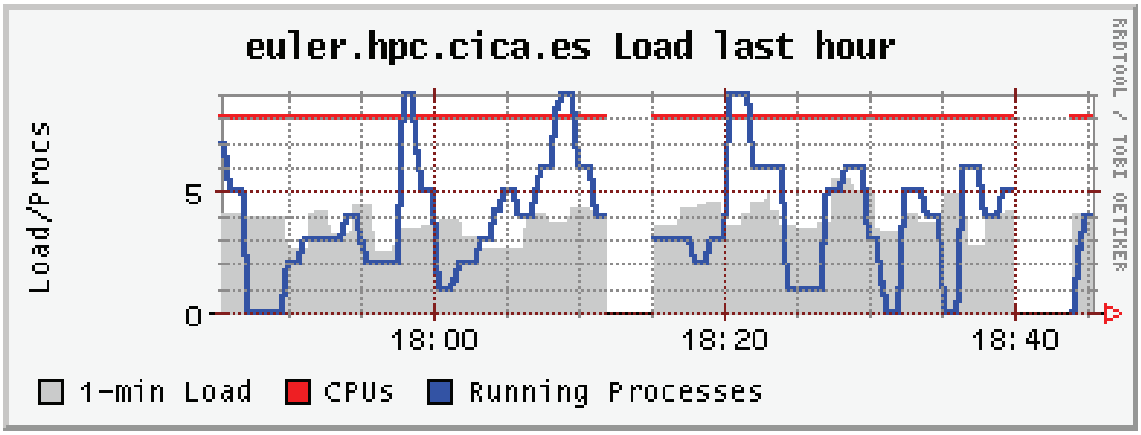
Incluso hemos podido constatar varios malfuncionamientos a la hora de monitorizarlo, donde apreciamos la falta de datos.



No ha sido un comportamiento aislado, sino todo lo contrario.



La sobrecarga del sistema es la causa evidente en todos estos malfuncionamientos.



Vemos cómo para tratar un problema de tal envergadura no es posible atacarlo simplemente subdividiendo el espacio de soluciones en 2000. Necesitaremos aumentar la granularidad del problema si queremos llegar a completar el cálculo de la solución.

También tendremos que tener cuidado con no sobrecargar las memorias de las máquinas, ya que trabajar con ficheros que pesan 3TB es lo sufientemente ineficiente como para no seguir con esta línea de actuación.

Prueba 4

Siendo conscientes de que la problemática de la anterior prueba era el uso abusivo de la capacidad de las máquinas, vamos a hacer uso de un dispositivo de almacenamiento de memoria masivo. Esta memoria de disco está configurada en RAID0 y se encuentra en /scratch/ , es exclusivamente utilizada para programas que requieran mucho espacio en disco.

Para hacer menor uso de la memoria de los ficheros temporales aumentaremos la partición de las subtareas. El rango máximo que puede alcanzar el array de subproblemas está limitado a 20000, al menos en la configuración actual del clúster del CICA.

```
#$ -t 1-20000
```

Esta vez al comenzar la ejecución, hemos podido observar cómo el fichero de salida estándar (.out) se iba llenando de información, supuestamente procedente de la ejecución de un geng. Una salida errática que hacía ir aumentando de tamaño el fichero hasta alcanzar 100GB en apenas 40 minutos. Paramos la ejecución del problema y revisamos el script.

```
#$ -S /bin/bash
#$ -N GENG15
#$ -e GENG15.20000.err
#$ -o GENG15.20000.out
#$ -q smnodes
#$ -V
#$ -cwd
#$ -t 1-20000

##accedemos al directorio correspondiente

cd /home/josera/resultados_geng15.20000/


#calculamos la variable auxiliar

export SGE_TASK_ID_AUX=`expr $SGE_TASK_ID - 1`

##se calcula cada trozo del geng y lo vuelca en un archivo temporal, también manda el resumen del resultado a .subres.

/home/josera/nauty22/geng 15 $SGE_TASK_ID_AUX/20000
```

```
/scratch/geng15_$SGE_TASK_ID.tmp &> geng15_$SGE_TASK_ID.subres

##concatenamos los resúmenes de los resultados parciales en .finalres

cat geng15_$SGE_TASK_ID.subres >> geng15.finalres

##borramos los subresúmenes

rm geng15_$SGE_TASK_ID.subres

##cribamos el fichero temporal parcial con pickg y lo vamos concatenando en el fichero final .pickg

/home/josera/nauty22/pickg -r /scratch/geng15_$SGE_TASK_ID.tmp >> geng15.pickg

##borramos el archivo temporal intermedio

rm /scratch/geng15_$SGE_TASK_ID.tmp
```

El error del script fue un salto de línea que se encontraba como parte de la invocación del geng.

Hemos dicho que el tamaño era un problema, pero el tiempo también lo es. En esta prueba la remesa de 4 tareas han tardado más de 40 horas en ejecutarse, por lo que haciendo cuentas, en el mejor de los casos, la ejecución total del programa podría llegar a durar 23 años:

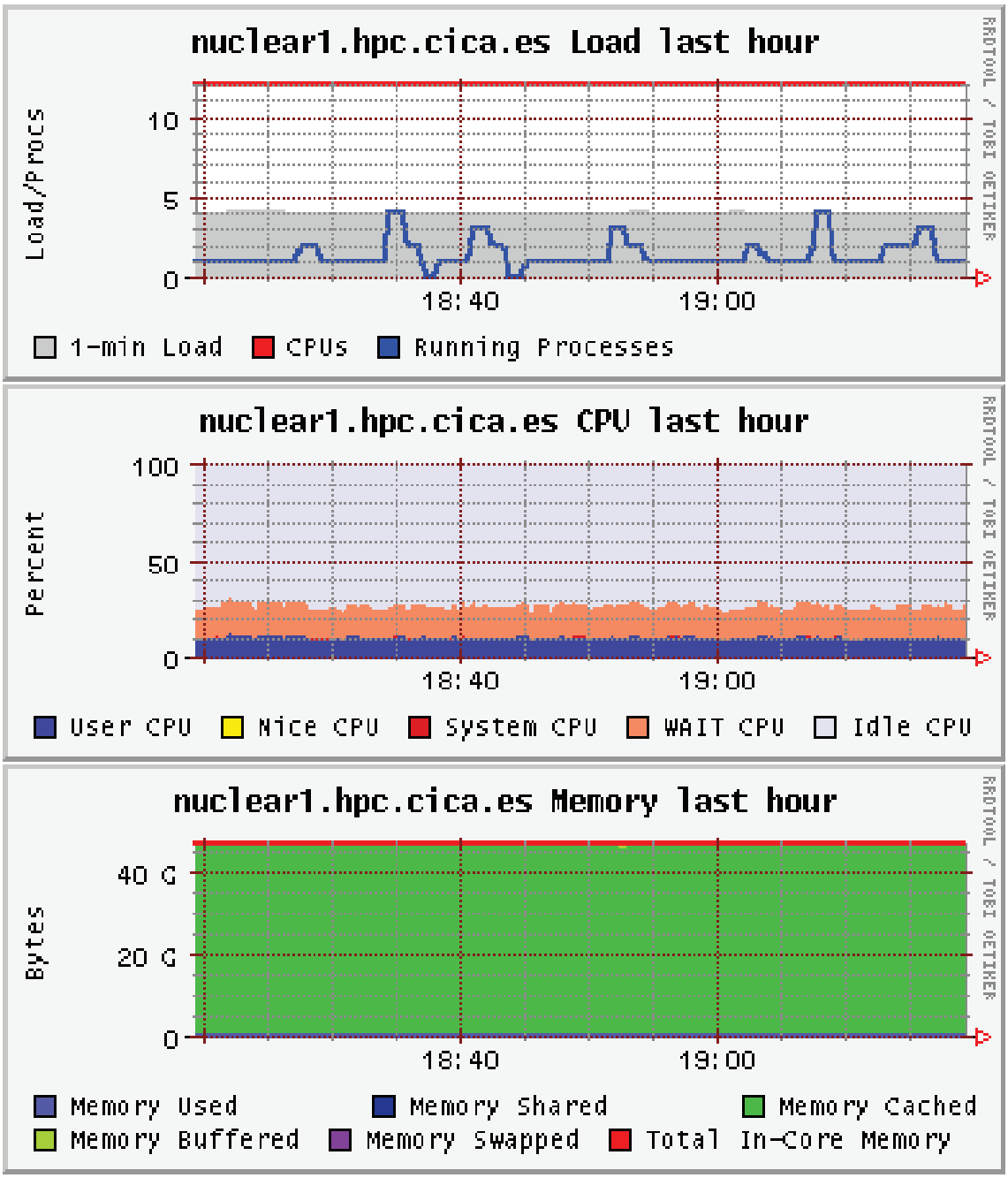
20000particiones/4remesa=5000*40horas/24h=8333dias=23 años

Ni siquiera aumentando el número de ejecuciones simultáneas de tareas a 50, podríamos esperar hasta casi dos años.

20000/50*40/24/365=1,8 años

Los problemas de tiempo y espacio son evidentes, así que tomaremos a partir de ahora otra línea de acción.

Estas son algunas gráficas recogidas durante la ejecución del programa.



Prueba 5

Ahora , en esta nuestra 5ª prueba, disminuiremos el tamaño del problema y no usaremos archivos temporales. Haremos uso de las tuberías para pasar directamente la salida del geng a la entrada del pickg.

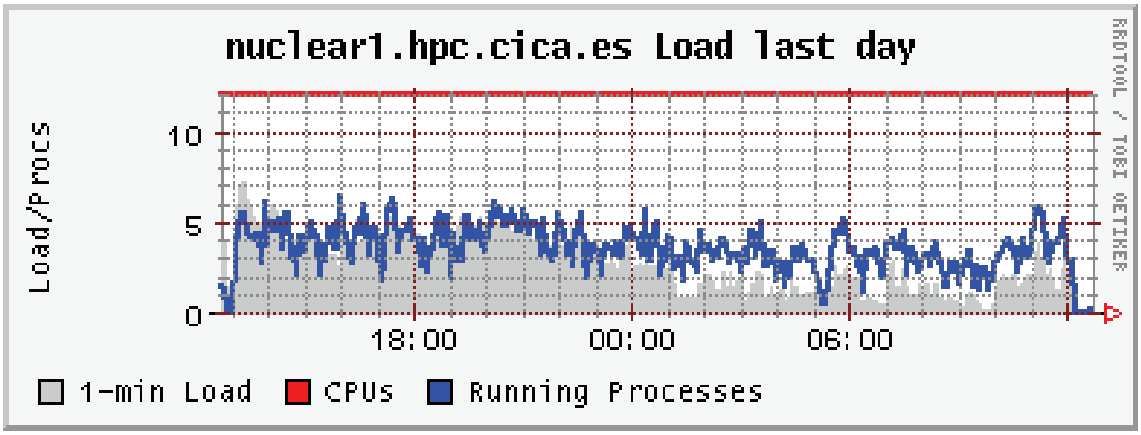
El código del script es este:

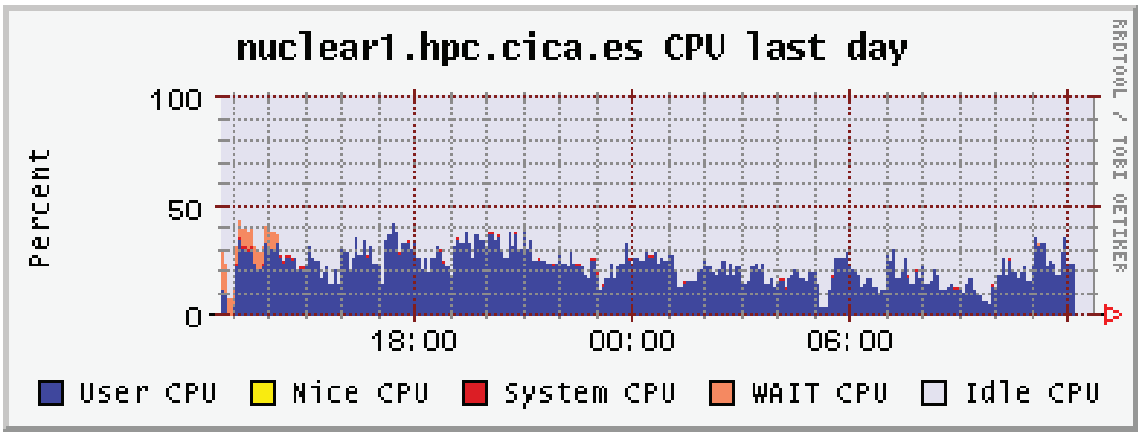
```
#$ -S /bin/bash
#$ -N GENG13A
#$ -e GENG13A.20000.err
#$ -o GENG13A.20000.out
```

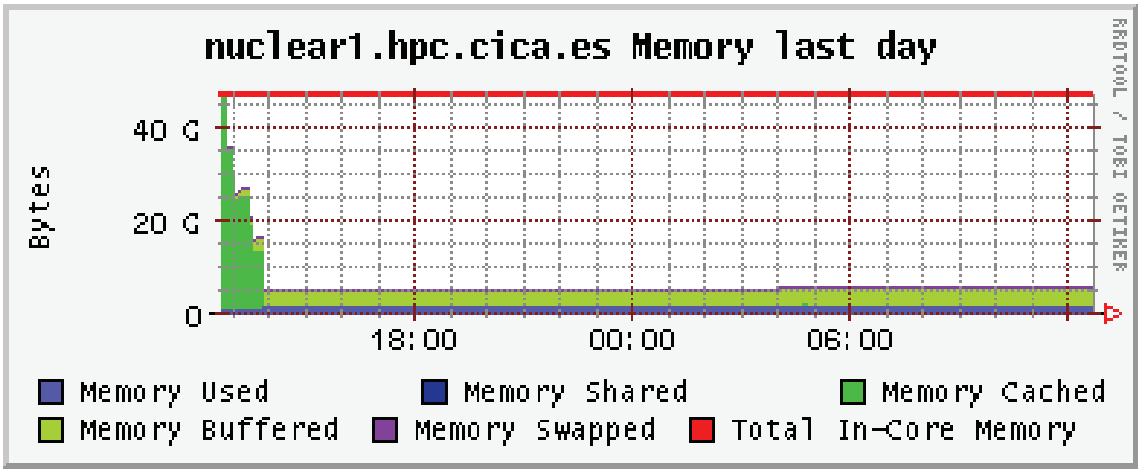
```
#$ -q smnodes
#$ -V
#$ -cwd
#$ -t 1-20000

##accedemos al directorio correspondiente

cd /home/josera/resultados_geng13A.20000/


#calculamos la variable auxiliar

export SGE_TASK_ID_AUX=`expr $SGE_TASK_ID - 1`

##se calcula cada trozo del geng (genera el subespacio de posibles soluciones) y lo pasa mediante una tubería al pickg (criba el subespacio de la subtarea)

/home/josera/nauty22/geng -t 13 $SGE_TASK_ID_AUX/20000 | /home/josera/nauty22/pickg -E >> geng13A.pickg
```

Esta vez trataremos con un problema de tamaño 13, bastante más manejable que uno de mayor tamaño. Los grafos generados por geng estarán libres de ciclos de grado 3 (también conocidos como triángulos) y pickg elegirá de ellos sólo los que sean Eulerianos.

Para este problema se han utilizado 4 cores, por lo que podía haber de 0 a 4 tareas corriéndose simultáneamente.







Comprobamos que el uso de las tuberías y no tener que utilizar la memoria en disco ha favorecido a la ejecución del problema y nos disponemos a ir aumentando progresivamente el tamaño del problema.

Hay que tener en cuenta que si nos saltamos el paso intermedio de escribir en disco toda la salida del geng, y simplemente se almacena en memoria principal , evitaremos 20000 creaciones de los ficheros .subres y .tmp , sus extensas escrituras, sus posteriores lecturas ,para la entrada del pickg y sus correspondientes borrados . Todo eso si lo hacemos en disco, mucho más lento que la memoria principal, la mejor opción sería ahorrarse los pasos intermedios y usar las tuberías a partir de esta prueba.

Prueba 6

Ahora aumentaremos el tamaño del problema para ver si sigue tardando un tiempo razonable,

```
#$ -S /bin/bash
#$ -N GENG14A
#$ -e GENG14A.20000.err
#$ -o GENG14A.20000.out
#$ -q smnodes
```

```
#$ -V
#$ -cwd
#$ -t 1-20000

##accedemos al directorio correspondiente

cd /home/josera/resultados_geng14A.20000/


#calculamos la variable auxiliar

export SGE_TASK_ID_AUX=`expr $SGE_TASK_ID - 1`

##se calcula cada trozo del geng (genera el subespacio de posibles soluciones) y lo pasa mediante una tubería al pickg (criba el subespacio de la subtarea)

/home/josera/nauty22/geng -t 14 $SGE_TASK_ID_AUX/20000 |/home/josera/nauty22/pickg -E >> geng14A.pickg
```

La ejecución del problema entero ha sido de 22 horas, siendo posible plantearse el aumento del tamaño del problema.
La ejecución de los subproblemas se resolvía en muy poco tiempo. Estaban más tiempo en la etapa de distribución de las tareas que calculándolas, así que para la siguiente prueba usaremos grafos de 15 vértices.

Prueba 7

Ahora aumentaremos el tamaño del problema a 15, para ver si esta vez puede resolverse en un tiempo razonable.

El código del script utilizado es este:

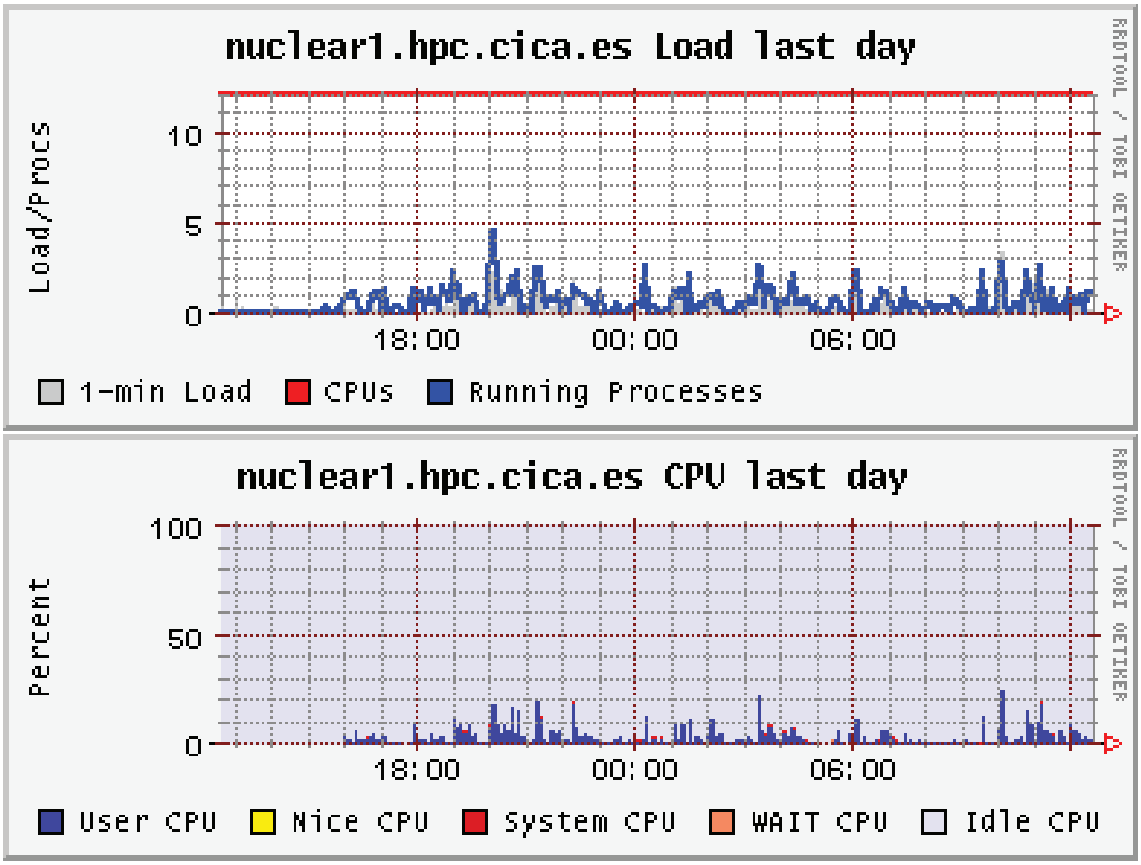
```
#$ -S /bin/bash
#$ -N GENG15A
#$ -e GENG15A.20000.err
#$ -o GENG15A.20000.out
#$ -q smnodes
#$ -V
#$ -cwd
#$ -t 1-20000

##accedemos al directorio correspondiente

cd /home/josera/resultados_geng15A.20000/
```

#calculamos la variable auxiliar

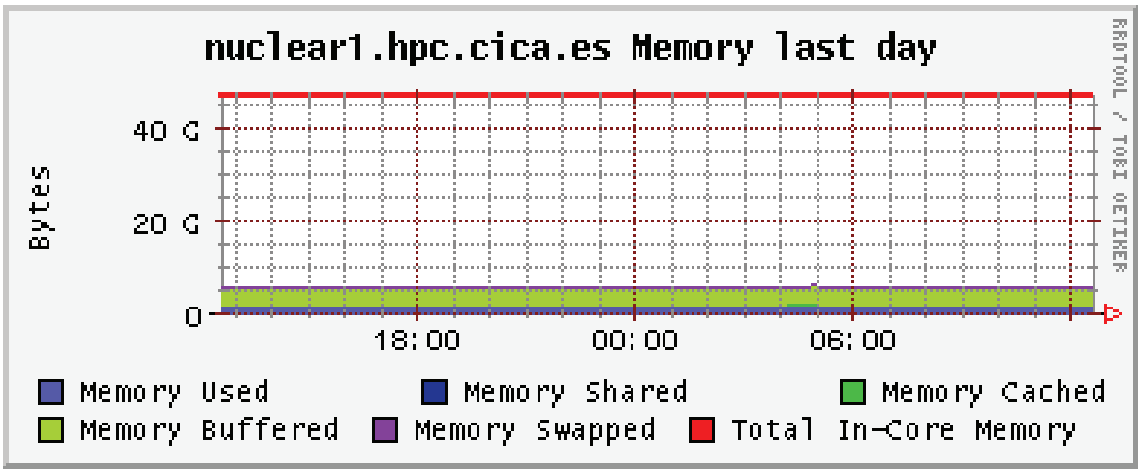
export SGE_TASK_ID_AUX=`expr $SGE_TASK_ID - 1`

##se calcula cada trozo del geng (genera el subespacio de posibles soluciones) y lo pasa mediante una tubería al pickg (criba el subespacio de la subtarea)

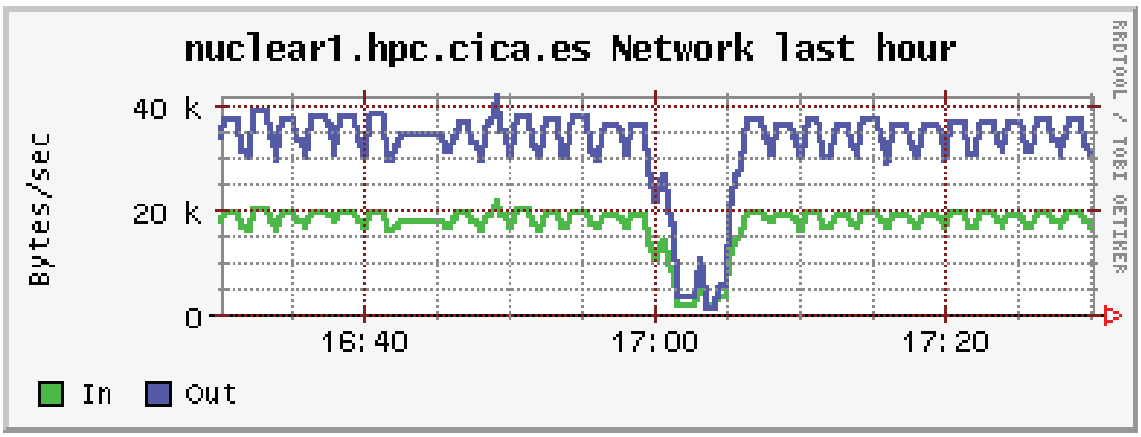
/home/josera/nauty22/geng -t 15 $SGE_TASK_ID_AUX/20000 |/home/josera/nauty22/pickg -E >> geng15A.pickg



Vemos cómo los procesos son más bien esporádicos y poco intensos. Por lo que podremos proceder en la siguiente prueba a aumentar el uso de cpus, que en estas pruebas ha estado limitado a 4.



El uso de la memoria no ha supuesto ningún problema



Un uso de la red relativamente alto.

La ejecución del problema también ha durado 22 horas, al igual que en la prueba anterior, por lo que tendremos que cargar un poco más la necesidad de computo de cada subtarea. Para la siguiente prueba aumentaremos el tamaño del problema.

Prueba 8

En esta prueba nos adentraremos en grafos de tamaño 16, este es el código del script:

```
#$ -S /bin/bash
#$ -N GENG16A
#$ -e GENG16A.20000.err
#$ -o GENG16A.20000.out
#$ -q smnodes
#$ -V
#$ -cwd
#$ -t 1-20000
```

##accedemos al directorio correspondiente

cd /home/josera/resultados_geng16A.20000/

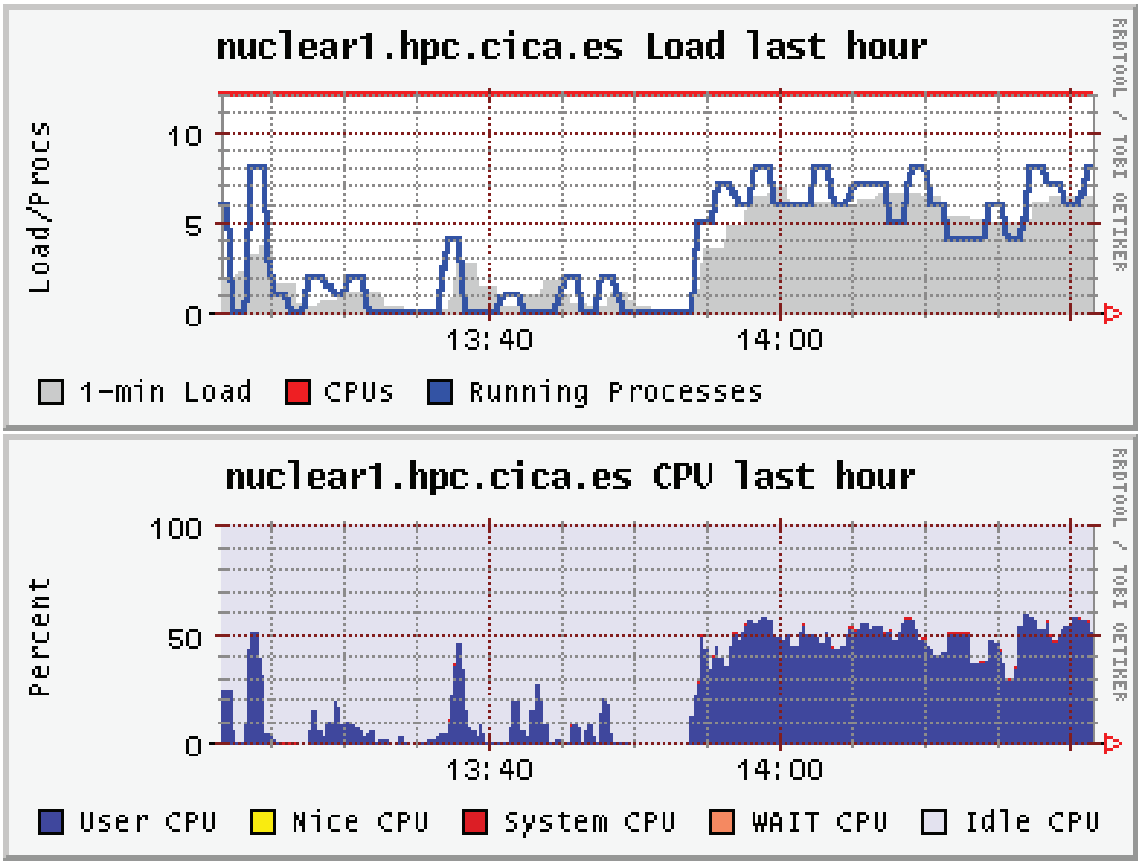
#calculamos la variable auxiliar

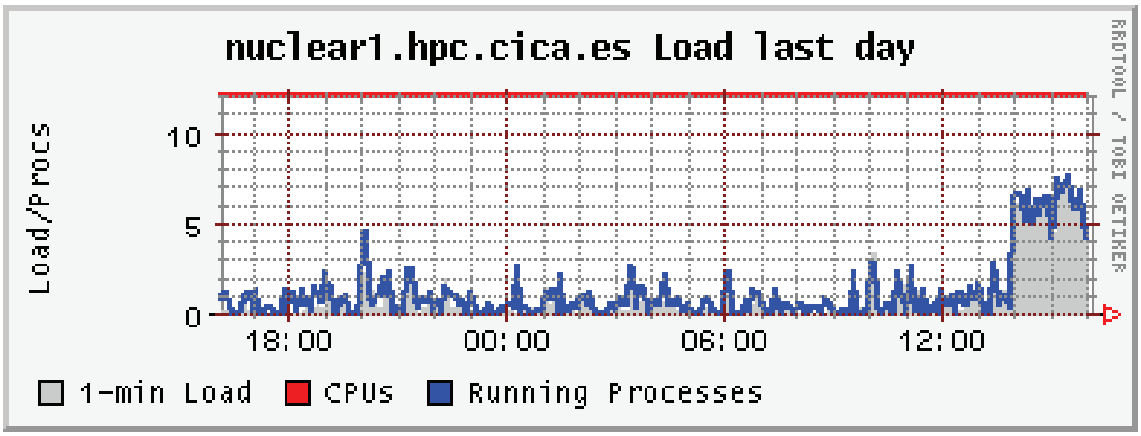
export SGE_TASK_ID_AUX=`expr $SGE_TASK_ID - 1`

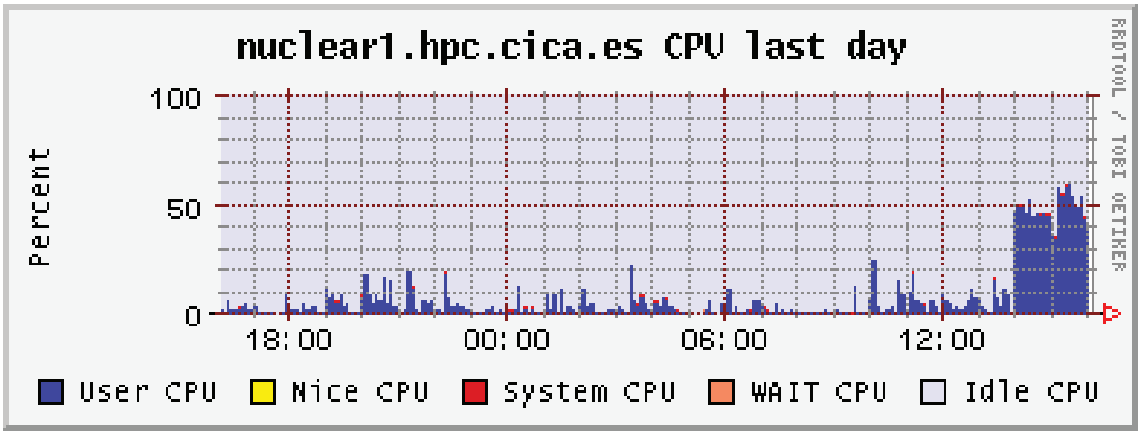
##se calcula cada trozo del geng (genera el subespacio de posibles soluciones) y lo pasa mediante una tubería al pickg (criba el subespacio de la subtarea)

/home/josera/nauty22/geng -t 16 $SGE_TASK_ID_AUX/20000 |/home/josera/nauty22/pickg -E >> geng16A.pickg

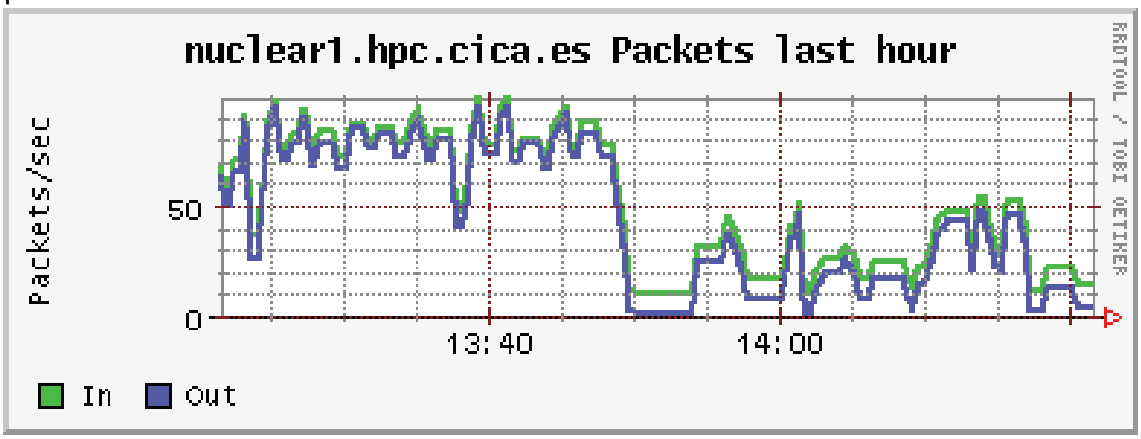
En las siguientes gráficas vemos cómo hacemos uso de un mayor número de procesadores, respecto de la anterior prueba.







En estas otras gráficas veremos cómo el uso de la red es menor y más pausado



El tamaño de las subtareas no es siempre el mismo. Ha habido subtareas que han llegado a durar 23 minutos en computarse. Si todas las tareas fuesen así, el tiempo de ejecución total sería de 80 días.

La primera aproximación del tiempo de ejecución total es de 3,7 días. Ésta se ha estimado una hora después de haber empezado a correrse la aplicación. A las 2 horas del comienzo se ha vuelto a hacer el cálculo de la estimación del tiempo total que ahora es 2,13 días.

Vemos claramente cómo con un pequeño porcentaje de la información total no podemos llegar a inferir ningún valor, ya que puede ser muy dispar en comparación con el valor real.

6.- COMPARACIÓN DE ALTERNATIVAS

Las alternativas son infinitas, dependiendo del tiempo y de los medios que dispongamos elegiremos una opción u otra. Aquí sólo comentaremos algunas de ellas, pero el abanico de posibilidades es enorme.
Se barajaron cuatro alternativas para obtener el volumen de cómputo necesario para poder atacar nuestro problema:

Clúster Beowulf

Fue la primera alternativa y siempre ha estado ahí como posibilidad el crear nuestro propio clúster Beowulf. Reuniendo máquinas en desuso de terceros podríamos haber construido un clúster casero para hacer las pertinentes pruebas.
Aun así nos hubiera quedado bastante pequeño para el objetivo final de este proyecto: los archivos intermedios generados por nuestro algoritmo llegan a pesar cientos (incluso miles) de GB. La potencia de cómputo también es crítica. Corriendo el mismo algoritmo en un clúster como en el del CICA, aun usando el mismo número de cores, reduciríamos muchísimo el tiempo de cálculo, que en nuestro caso puede durar incluso meses.

Clúster SUN

El clúster que tenemos en la universidad ,la sala SUN, pudo también haber sido nuestra herramienta de cálculo, y de hecho lo fue durante unos meses. El principal problema de dicho clúster es su mantenimiento y administración, prácticamente nunca pude mandar a correr en el clúster ninguna aplicación. Eso aunado a que en la universidad no se imparte materias de supercomputación, hizo que el desinterés de los alumnos y profesores ayudase al desuso de este infrautilizado clúster.
Cabe mencionar que allí fue donde tuve mi primer contacto con el gestor de colas SGE, aunque no fuese del todo funcional.

Proyecto en Ibercivis

Podríamos habernos planteado la construcción de un proyecto para lanzarlo mediante el multiproyecto Ibercivis. El hardware en este caso serían las máquinas de todos los voluntarios que me donasen cómputo. El problema tendría que estar planteado para resolverse en un tiempo estimado de pocos años, pero no valdría la pena planteárselo si nuestro problema no fuese lo suficientemente grande.

Clúster CICA

Finalmente ha sido la alternativa escogida, gracias a las comodidades que ofrece: alta disponibilidad , condiciones ambientales idóneas para el

hardware (temperatura , humedad, etc.), gran cantidad de recursos y gran número de profesionales dispuestos a ayudarte a la hora de usar el clúster.

El lenguaje de programación a utilizar debe depender de nuestras necesidades y del enfoque del problema, siempre intentando usar los menores recursos posibles. Si el problema es fácilmente tratable mediante programación estructurada, esta será la opción a elegir. Si la resolución de nuestro problema necesita del uso de clases y objetos, lo más coherente sería utilizar un lenguaje de programación orientado a objetos.

Como el paquete nauty está escrito en C y los script de Grid Engine se escriben con comandos UNIX, la opción de la programación estructurada será la elegida.

La elección de un lenguaje de programación de más alto nivel u orientado a objetos ,sería una mala opción, ya que complicaría la ejecución de la aplicación y desperdiciaría casi todo el potencial que este tipo de programación nos aporta.

MPI

MPI (Message Passing Interface) es un protocolo de programación para comunicación mediante paso de mensajes. Fue diseñado por un comité de académicos e industriales con el fin expreso de convertirse en un estándar de programación con paso de mensajes bien definido buscando que la interfaz establezca un estándar práctico, portable, eficiente y flexible.

La especificación definitiva de MPI 1.0 se completó en 1994, y se ha revisado y actualizado constantemente. Actualmente la especificación más reciente es MPI-2.2 (SEP2009) Su antecesor fue PVM, mencionado en otros capítulos.

Es aplicable desde C o Fortran.

Esta librería de funciones tiene como parte de su funcionalidad comunicar a los procesadores que trabajen conjuntamente mediante paso de mensajes. Aunque posee más de 125 funciones, es posible trabajar paralelamente únicamente con 6 primitivas básicas (escritas en C):

MPI_INIT(&argc, &argv) // int argc, char **argv

Sirve para inicializar el ambiente paralelo

MPI_COMM_SIZE(comm, size) // int size

Regresa el número de procesadores así como el comunicador, por defecto comm=MPI_WORLD_COMM. MPI_WORLD_COMM indicar el conjunto de todos los procesadores asignados a la ejecución del programa. El usuario también puede designar sus propios comunicadores para formar subconjuntos de procesadores.

MPI_COMM_RANK(comm, rank) // int rank

Regresa el número lógico que corresponde a cada procesador. Este valor siempre empieza en cero y alcanza un valor máximo igual al número de procesadores menos uno.

MPI_Send(&buf, count,datatype,dest,tag,comm) // void *buf, int count,dest,tag

Envía un mensaje a otro procesador. Es una instrucción bloqueante, por lo tanto el procesador origen espera que el procesador destinatario haya recibido el mensaje antes de continuar trabajando.

MPI_Recv(&buf, count, datatype, source, tag, comm, &status)
// void *buf
// int count, source, tag
// MPI_Status status

Se dispone a recibir un mensaje de parte de otro procesador. También es una instrucción bloqueante, por lo tanto el procesador destinatario no puede continuar su trabajo hasta haber recibido dicho mensaje.

MPI_Finalize()

Cierra el ambiente de trabajo en paralelo una vez finalizado el trabajo

Es necesario mencionar que necesitamos incluir la librería en la cabecera para
poder hacer uso de sus funciones

#include <mpi.h>

MPI se aplica primordialmente en computadoras paralelas de memoria distribuida, aunque también se puede utilizar en computadoras paralelas de memoria compartida.

Todos los procesadores reciben la misma copia del programa a ejecutar. El usuario es responsable de incluir instrucciones de comunicación entre procesadores y de sincronización de trabajo entre los mismos.

Existen muchas implementaciones de MPI, pero posiblemente la más usada y robusta sea MPICH. Está diseñado por capas, permitiendo gran portabilidad sin sacrificar el rendimiento. MPICH también proporciona los medios para iniciar una aplicación en MPI. En el caso de MPICH, se proporciona un comando $mpirun , al que se le puede especificar el número de procesos a iniciar. Dicho comando encapsula todo el proceso necesario para determinar la arquitectura del equipo en que se está ejecutando, preparar la interfaz de comunicaciones, y lanzar los procesos que componen la aplicación.
Compilación en C

mpicc -o <exec filename> <filename.c> -lm

//lm indica que se utiliza la librería de funciones matemáticas

Ejecución en C

mpirun -np n ./<exec filename>

//n asigna el número de procesadores a utilizar

Ibercivis

Una alternativa para distribuir nuestro problema sería consolidar un proyecto dentro de Ibercivis. El problema a resolver debe ser lo suficientemente grande y largo como para invertir unos meses en poner a punto el proyecto y las unidades de trabajo.

La experiencia personal del investigador que lanzó el proyecto de Sanidad está siendo muy positiva, y nos anima a que planteemos un proyecto si el problema lo requiere.

En Ibercivis, cuando le propones un proyecto, si pasa un gabinete de evaluación para ver si es apto o no, te asesoran a la hora de plantear y distribuir el problema.

Esta opción está desestimada ya que sería resolver problemas más grandes de los que voy a tratar en el presente proyecto y durante un periodo de tiempo más largo a un curso lectivo.

Condor

Una de las alternativas a Grid Engine, es Condor. Aunque esta alternativa

quede desestimada al no usarse este software en el CICA, es conveniente ver sus características, por si en algún momento vamos a usarlo.

Algunas de ellas son:

-Soporta PVM y MPI

-Permite computación Grid

-Checkpoint y migración

Para asegurar la integridad de los trabajos, si se modifica el entorno o la máquina deja de estar disponible, se realiza un checkpoint y se migra a otra máquina que esté disponible.

-Sistema de llamadas remotas

El programa se comporta como si se estuviera ejecutando en la máquina originaria del trabajo, despreocupando al usuario sobre los sistemas de ficheros.

-No es necesario modificar el código fuente de las aplicaciones

Los checkpoints, las migraciones y las llamadas remotas son automáticas y transparentes al usuario.

-Los trabajos se pueden ordenar o priorizar

El orden de ejecución de los trabajos por motivos de dependencias entre ellos se puede manejar fácilmente.

-ClassAds

El mecanismo ClassAd de Condor permite un framework muy flexible para resolver peticiones de recursos.

Condor se encuentra en desarrollo constante, y su última versión estable es la 7.6.2 (Julio2011). Fue desarrollado en la Universidad de Wisconsin-Madison y se encuentra bajo la licencia Apache License 2.0. Puede correrse sobre sistemas tipo Linux, Windows o Mac OS X.
Para más información visitar su sitio web:

http://www.cs.wisc.edu/condor/

7.- CONCLUSIONES Y DESARROLLOS FUTUROS

Hace ya bastante, al comenzar esta carrera, tuve una maravillosa clase de Matemática Discreta impartida por JoseRa, en la que nos habló sobre el proyecto SETI y de la capacidad de cómputo que podríamos conseguir con la computación distribuida en un futuro, resolviendo problemas inabarcables por su tamaño o complejidad.

Esa primera idea me atrajo lo suficiente como para empezar a buscar información. Entonces encontré una cosa llamada BOINC , en la cual me decían que la computación distribuida también podría ser computación voluntaria si los integrantes eran personas anónimas cediendo parte del potencial de cómputo de sus ordenadores personales para resolver problemas de meteorología, proteómica, etc. Estas personas estaban repartidas por todo el mundo y cada una se encargaba de resolver un subproblema y transcurridas 4, 6, 100 horas de cómputo, las que fueran, mandaban la subsolución e iban construyendo, con los cachitos de miles de personas, un problema que no hubiese podido resolverse en un clúster normal.

Cuando pensé en dejar de ser un mero observador o voluntario y enfocar lo que me quedaba de carrera hacia la computación paralela o distribuida, no encontré ningún temario ni ninguna asignatura que me ofertase la universidad y entonces fui a pedir consejo al antes mencionado profesor y mi actual tutor, José Ramón Portillo

Su cara lo transmitía todo, no había nada docente en las dos carreras, la licenciatura y la diplomatura, que versase mínimamente sobre la Computación Distribuida. Me aconsejó que orientase lo que pudiese, el proyecto final de carrera y las prácticas en empresa, y eso hice. A día de hoy espero que me asignen las prácticas de empresa en el departamento de supercomputación del CICA.

Después de llevar algunas pruebas paralelas a cabo vemos como debemos ser lo suficientemente cautelosos a la hora de elegir el tamaño del problema a tratar, y no olvidar el probar los problemas de tamaño poco menor, para poder estimar cuanto más o menos va a poder tardar en calcularse por completo.

El uso de la memoria en disco queda exclusivamente para aquellos casos en los que no nos sea suficiente la memoria principal, intentando exprimir el potencial de nuestro hardware y aumentando lo máximo posible el rendimiento del programa.

El tiempo requerido de ejecución es a veces exagerado y obliga a tratar el problema desde otro enfoque. Cuando después de la primera prueba con

un tamaño del problema de 15, vimos que la ejecución podría durar 23 años, tuvimos que abortar su ejecución y pensar en alternativas.

Al principio intentamos hacer uso del clúster de SUN de la universidad, pero no conseguimos llegar a computar ninguna tarea. Como solución decidimos usar el centro de cálculo del CICA.

Los problemas que tuvimos antes de poder correr ningún script fueron a la hora de escribirlo. Para solucionarlo pedimos a los técnicos del CICA que nos facilitasen las plantillas de un script modelo, ya que la documentación aportada en su web estaba algo desactualizada y no conseguíamos hacer correr ninguna aplicación.

La problemática del espacio la pudimos resolver en un principio usando discos de capacidad masiva y posteriormente ahorrándonos el escribir en disco los archivos temporales mediante el uso de tuberías.

En primer lugar una de nuestras problemáticas fue la escritura del primer script, ya que no teníamos conocimiento de las variables que debíamos usar o la forma de distribuir el problema.

Nos aconsejaron en el CICA que hiciésemos las pruebas con memoria compartida pero con cálculo distribuido, ya que si las pruebas no necesitaban más de 4 u 8 subtareas simultáneas ,el uso de librerías de paso de mensajes supondría una carga. Podría aprovechar mejor el uso de la memoria compartida, en vez de estar haciendo uso de la red para tan pocas tareas simultáneas. En el supuesto en el que pudiese lanzar 50 tareas simultáneas no significaría que pudiese resolver los problemas que me planteo en un tiempo razonable. En las pruebas hemos visto como de 23 años de ejecución habríamos pasado a casi 2 años, mejorando de 4 tareas simultáneas a 50. Necesitaríamos el uso exclusivo de 50 cores del CICA durante 2años.

Hemos llegado con algunas pruebas a llenar 3,4 TB de información y descubrimos como la memoria en disco por muy grande que sea , no siempre es suficiente. Tenemos que tener mucho tacto a la hora de tratar con problemas de gran tamaño. En nuestras últimas pruebas hemos hecho uso de la memoria principal que a diferencia de la de disco esta es más rápida de acceder y modificar.

El tiempo de ejecución es más complicado evitar que se dispare para problemas lo suficientemente grandes. Subdividir el problema en mas subtareas es útil pero tiene sus límites, si la subdivisión es lo suficientemente pequeña (granularidad gruesa) los subproblemas a resolver serian también muy grandes y por lo tanto intratables, y si la subdivisión es lo suficientemente grande (granularidad fina) las tareas usaran un tiempo mayor en comunicaciones entre nodos que en calcularse.

El parámetro de tamaño del array de tareas 't' es máximo 20000, tal y como está ahora configurado el CICA. Si quisiésemos usar una partición mayor tendríamos que pensar en distribuir el problema usando memoria distribuida, pero como ya vimos, este esfuerzo es inútil si no vamos a poder tener más de 50 o 100 tareas simultaneas corriéndose.

Este proyecto puede ser el germen de algún problema aun mayor, como en Física Cuántica, con la resolución de la paradoja del Teorema de Kochen-

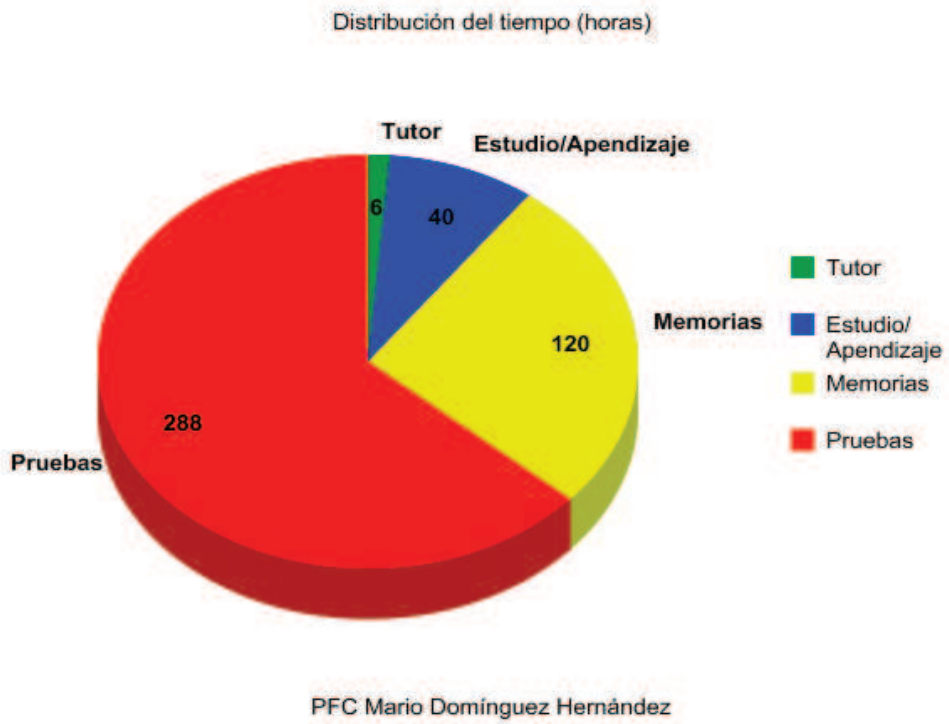
Specker y sus variables ocultas. También podríamos optimizar los resultados para un TSP (Problema del Viajante) de muchas variables, o incluso atacar un problema de simulación de redes neuronales. Las posibilidades son enormes, simplemente deberemos modelar el problema con respecto a nuestros objetivos , requisitos y preferencias.

Otro posible proyecto habría podido ser ver como explotar el potencial de una GPU para correr problemas basados en grafos. Las ultimas GPU pueden llegar a tener 256 o más mini-cores, en forma de malla, con sus pequeñas alu y su propia cache, una estructura inherentemente paralela, por lo que las hacen una herramienta muy útil para este tipo de cálculos masivos.

Distribución del Tiempo

La distribución del tiempo empleado para realizar este proyecto es la siguiente

-Horas con el tutor	6h
-Tiempo de estudio y aprendizaje de las herramientas	40h
-Estructuración y realización de las memorias y la presentación	120h
-Tiempo de ejecución y modificación de las pruebas	288h
Total:	454h



BIBLIOGRAFÍA

Título: ” Introduction to Distributed Algorithms”
Autor: Gerard Tel ISBN: 9780521794831
Publicación: Cambridge, UK (Cambridge University Press)

Título: ” nauty User´s Guide (Version 2.4)”
Autor: Brendan D. McKay Computer Science Department Australian National University
Web:
http://cs.anu.edu.au/~bdm/nauty/nug.pdf

Título:”Sun N1 Grid Engine 6.1 User´s Guide”
Autor: SUN Microsystems, Inc.
Web:
http://www.csb.yale.edu/userguides/sysresource/batch/doc/UserGuide_6.1.pdf

Título: “Student Guide for UNIX & LINUX Fundamentals for HPC ”
Autor: Texas Advanced Computing Center
Web:
http://cms.tacc.utexas.edu/fileadmin/class_materials/10-26_Linux_Unix/Student_Guide_for_UNIX___LINUX_Fundamentals_for_HPC.pdf

Página Web Ibercivis: http://www.ibercivis.es/
Consultada: Agosto 2011

Página Web CICA: http://www.cica.es/
Consultada: Agosto 2011

Página Web E-Ciencia: https://eciencia.cica.es/
Consultada: Agosto 2011

Página Web Folding: http://folding.stanford.edu/
Consultada: Agosto 2011

Página Web Beowulf: http://www.beowulf.org/
Consultada: Agosto 2011

Página Web Top500: http://www.top500.org/
Consultada: Agosto 2011

Página Web Boinc: http://boinc.berkeley.edu/
Consultada: Agosto 2011

Página Web BoincStats: http://es.boincstats.com/
Consultada: Agosto 2011