Universidad Tecnológica Nacional – Facultad Regional Buenos Aires

Ingeniería en Sistemas de Información

sistemas operativos

INSOMNIO

RED DE PROCESAMIENTO CONSTANTE

Definición

Idea y diseño:
Ing. Diego Marafetti

Colaboradores:
Pablo Lorenzatto
Santiago Ciciliani
Ing. Pedro Vazquez

Revisión: 1.0
2° Cuatrimestre 2010

Índice de Contenidos

Prólogo

Bienvenidos al trabajo práctico de Sistemas Operativos. Este documento los guiará a través de los requerimientos necesarios para cumplir los objetivos de dicha práctica. El alumno aprenderá los conceptos prácticos y teóricos sobre los sistemas operativos más populares y podrá relacionarlos con lo aprendido en clase. Este trabajo práctico cubre los siguientes temas:

- ✓ Creación y manipulación de procesos y threads mediante la API del sistema.
- ✓ Sincronización de Procesos, IPC y manejo de señales.
- ✓ Administración de Memoria.
- ✓ Manejo del Filesystem a través de la interfaz de sistema.
- ✓ Introducción a los Sistemas Distribuidos.
- ✓ Arquitecturas basadas en capas físicas, protocolos y mensajería.
- ✓ Diferencias entre las plataformas más utilizadas del mercado.

Desde el punto de vista académico el trabajo está diseñado para que:

- ➢ Los alumnos adquieran los conocimientos prácticos sobre el uso de un conjunto de herramientas que ofrecen los sistemas operativos modernos.
- ➢ Entiendan la importancia de una norma o protocolo estándar en la comunicación entre procesos y diferentes plataformas.
- ➢ Dominen los problemas específicos de este tipo de implementaciones.
- ➢ Apliquen en forma práctica el uso de lenguaje C en implementaciones de bajo nivel.
- ➢ El grupo de alumnos aprenda el trabajo en equipo con las problemáticas y las responsabilidades que eso implica.

Requisitos necesarios antes de empezar

Este documento contiene la definición del trabajo práctico, en otras palabras, contiene los requerimientos funcionales de la aplicación. Acompañado, viene otro documento llamado "Restricciones Técnicas" el cual es un complemento de este y el cual contiene restricciones que el alumno debe tener en cuenta a la hora de implementar una solución. Con estos dos documentos el alumno ya se encuentra preparado para comenzar a trabajar en la construcción de la aplicación.

Mucha Suerte!
El equipo de ayudantes de Sistema Operativos

Introducción

En los laboratorios de la NASA, el Deep Space Department (Departamento de Espacio Profundo) recibió los últimos datos del telescopio espacial Hubble, luego de tres semanas con el lente abierto observando el supercluster de galaxias de Virgo ubicado a 100 millones de años luz.
En total, unos 56 Terabytes de imágenes fueron captados con el nuevo filtro que le permite detectar la Materia Oscura (Dark Matter).

Los científicos están muy excitados por este suceso ya que les va a permitir explicar que forma tiene el universo y qué papel juega la materia oscura en esto. Procesando la información obtenida y corriendo una simulación podrán ver la forma real del universo y cómo influye la fuerza de gravedad de la materia oscura sobre los superclusters de galaxias.

Para lograr esta simulación la NASA ordenó la adquisición de 500 equipos SGI Altix 4700, dotados con 24 procesadores Intel Xeon y con soporte de hasta 1024 núcleos en una sola instancia de Unix, 96 GB de memoria local (creciendo hasta 128 TB global) y I/O escalable dentro de un entorno de memoria compartida (último modelo de la empresa que usan la potente arquitectura NUMA Flex).

Al mismo tiempo, los directores del departamento de Ingeniería de Software llamaron a licitación poniéndose en contacto con las empresas más reconocidas en el ámbito de desarrollo de software para poder construir la plataforma que permitiría correr la simulación y ejecutar procesamientos muy extensos.

Es un proyecto enorme y muy ambicioso ya que la fecha límite en la cual se debe presentar la plataforma es dentro de cuatro meses. Varias compañías rechazaron participar del proyecto por este motivo.
La UTN fue seleccionada para la realización de la plataforma y cientos de estudiantes de Ingeniería en Sistemas de distintos años se ofrecieron para llevar a cabo el desarrollo.

Durante el desarrollo del plan de proyecto se llegó a la conclusión de que los estudiantes de Sistemas Operativos eran los más capacitados para llevar a cabo su implementación, dado sus conocimientos sobre los internos de los sistemas operativos más modernos en el mercado.
La NASA brindó un documento con especificaciones muy estrictas sobre implementación en cada plataforma el cual se expone a continuación.

Esquema general

El proyecto se basa en la implementación del primer prototipo del sistema distribuido que permita ejecutar procesos que requieren gran capacidad de carga de recursos en un Grid de computadoras. De otro modo sería imposible su ejecución en una computadora normal. Esta plataforma será el punto de partida para luego correr simulaciones muy complejas.

En este sistema distribuido entran en juego los siguientes participantes:

- *Jobs*: es el trabajo de procesamiento a realizar que a su vez está compuesto por un conjunto de tareas que dependen entre sí.

- *Distribuidor de Carga*: Determinará quién es el nodo más adecuado para procesar la tarea basada en sus recursos. Analizará cada Job y generará un grafo de dependencias de las tareas para distribuir en el Grid. Habrá un solo distribuidor de carga en cada momento.

- *Nodos*: Son los procesos que ejecutan las tareas que componen un Job. Cualquier nodo puede funcionar como Distribuidor de Carga y la elección del mismo será determinada por todos los nodos en un proceso de elección de coordinador.

- *Web Server*: Será una pequeña aplicación que utilizará el protocolo http para hacer de interfaz entre el Grid y el mundo exterior. Será el medio por el cual el usuario cargue los Jobs a ser ejecutados por el sistema y consulte el estado del Distribuidor de Carga.

Para que el sistema cumpla con las expectativas y sea aceptado por los ingenieros del equipo, todos los componentes deberán cumplir con los requisitos que se especifican en este documento. A continuación se les presenta a los desarrolladores una descripción detallada sobre cada uno de los componentes que forman parte de la plataforma.

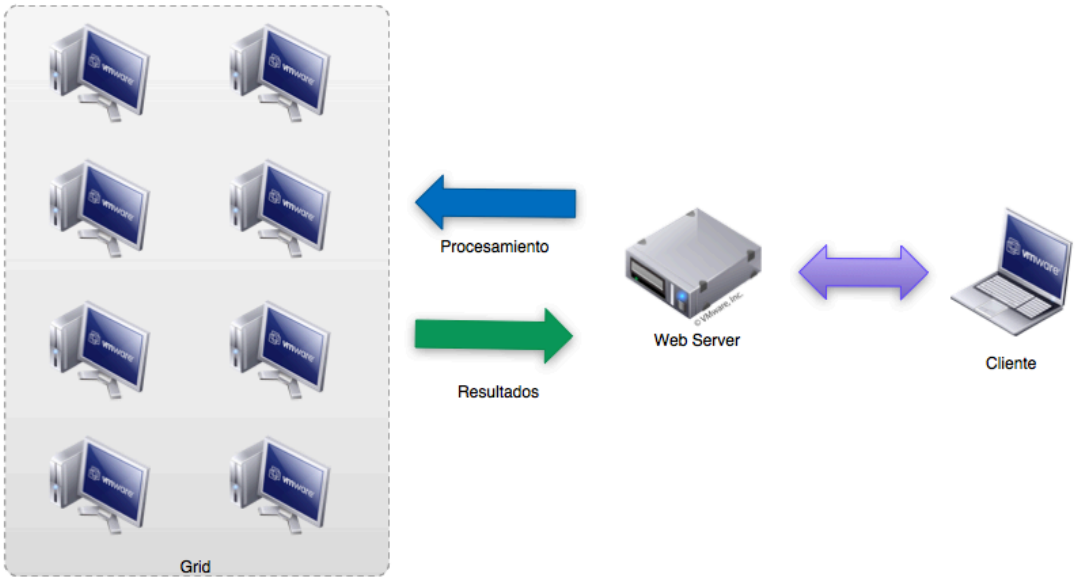


Diagrama de Arquitectura de la Red

Procesamiento

En esta primera etapa del proyecto se utilizará el poder de cómputo de la red para romper claves utilizando un mecanismo de fuerza bruta. Se planea en un futuro cambiar la implementación del Task Executer (explicado más adelante) para que acepte otro tipo de cómputo. La especificación de cómo se van a procesar las claves es la siguiente:

- Solo se aceptarán valores alfanuméricos con una longitud fija. La expresión regular (según el estándar Basic Regular Expression ó BRE) de una clave válida es "`^[a-zA-Z0-9]\{n\}$`". El valor de n es establecido antes del procesamiento.

- Solo se aceptarán claves con hash MD5 aplicado.

Para lograr romper la clave por fuerza bruta el sistema deberá calcular todas las combinaciones hasta encontrar cual es la combinación de caracteres para el cual C = MD5("palabra"), donde C es el hash que se envió al Grid y MD5("palabra") es el resultado de aplicar el algoritmo MD5 a una combinación de letras. Por ejemplo, dado el siguiente hash "6e023d8b8758f8f826152de77a3cc1d9" con una longitud de palabra de 6 caracteres resultaría en las posibles combinaciones:

MD5 ("AAAAAA") = 36d04a9d74392c727b1a9bf97a7bcbac
....
MD5 ("AAABAA") = e25c192a3590650ecd19bc5b38ff77ff
MD5 ("AAABBA") = 4e98618fb6b1af0f13f6cecc4e3465d9
MD5 ("AAABBB") = 6e023d8b8758f8f826152de77a3cc1d9

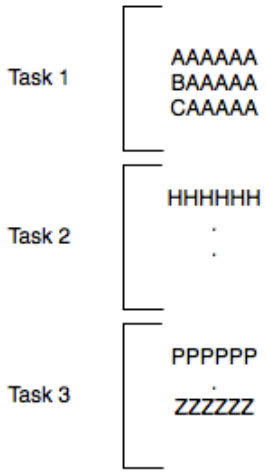
En este caso la palabra buscada coincide con "AAABBB".

Descripción de un Job

Un Job describe el grupo de tareas a ejecutar. Esta definición en XML describe la tarea en términos de usos de recursos y de dependencias.

El significado de tarea ó Task para está implementación del Task Executer será la de un lote de combinaciones a procesar en un nodo especifico. En el ejemplo presentado más adelante, el Distribuidor de Carga que reciba este Job podrá ejecutar cada una de las tareas en tres nodos distintos en forma paralela reduciendo el tiempo en tres.

Será el Distribuidor de Carga el responsable de calcular el rango del lote de acuerdo a la cantidad de tareas existentes y a la longitud de la palabra buscada.



Dependencias

Las tasks tendrán interdependencias entre ellas y esa relación también estará expresada en el XML. Esto significa que puede depender de cero, una o más tareas. Como consecuencia, para comenzar la ejecución de una Task, todas sus dependencias deben haber terminado de ejecutar. Cabe aclarar que esta dependencia es ficticia y no tendría ningún significado válido en la realidad.

Si la solución se encuentra en un lote de datos la ejecución finalizará y si hubiese tareas esperando estas no se ejecutarán. Tampoco serán válidos los lotes que se encuentran en ejecución en forma concurrente.

El XML presentado al final de la sección describe al Job como un conjunto de tareas (tasks). El orden de aparición de un ítem dentro de la colección en el tag "tasks" no garantiza que ese sea el verdadero orden de ejecución. Eso lo determinará el Distribuidor de Carga que interpretará todo el documento. Además, dicho orden es establecido por el tag "dependencies" (en el ejemplo, la *task1* no podrá ejecutarse porque depende de la finalización de las tareas *task2, task6* y *task3* respectivamente).

Uso de recursos

Una Task está definida en términos de uso de recursos del Nodo host en donde se ejecute. Los recursos disponibles son los siguientes:

Memoria real: Cantidad de bytes que la tarea necesita para llevarse a cabo.

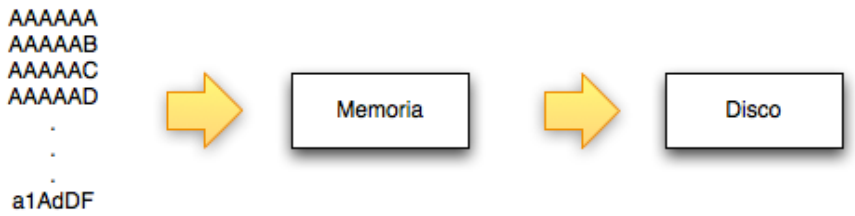
Disco: Cantidad de bytes de uso del dispositivo que requiere la tarea para llevarse a cabo.

Cpu: Tiempo de uso. Es el tiempo asignado de CPU estimado para procesar esa cantidad de memoria y disco en ese nodo.

Todas las unidades que representen bytes se pueden expresar de la siguiente forma (256MB, 5GB, 204523433B). El tiempo se mide en segundos.

En esta implementación del Task Executer (proceso explicado más adelante) el uso de los recursos para la ejecución de las tareas será administrado de la siguiente manera.

Cada combinación MD5 de 32 bits que se genere será almacenada en un buffer en la memoria RAM hasta cubrir el límite del tamaño especificado por la restricción. Cuando dicho límite se alcance, el buffer será volcado a disco y restablecido. Este proceso continuará hasta alcanzar la cantidad máxima de uso de disco especificado en la restricción. En el escenario donde ese máximo es alcanzado, ambos búffers serán vaciados y el proceso de volcado de memoria comenzará nuevamente.
El proceso de uso de memoria y disco será válido dentro del tiempo de uso de CPU asignado al procesamiento de la tarea. Si luego de dicho período no se encontró un resultado válido, el procesamiento será suspendido. Como consecuencia, el Distribuidor de carga reasignará la tarea a otro Nodo de la red para continuar con el procesamiento.



Ejemplo: XML que especifica un trabajo a ser procesado en el Grid.

```
<job id="job1">
   <target>
       <hash>6e023d8b8758f8f826152de77a3cc1d9</hash>
       <method>MD5</method>
       <length>6</length>
   </target>
   <tasks>
       <task>
         <task-id>task1</task-id>
         <task-description>This is the first task</task-description>
         <resources>
             <resource name="disk"   value="512MB" />
             <resource name="memory" value="250MB" />
             <resource name="cpu"    value="300"   />
         </resources>
         <dependencies>
             <dependency ref="task2" />
             <dependency ref="task6" />
             <dependency ref="task3" />
         </dependencies>
       </task>
       .
       .
       .
   </tasks>
</job>
```

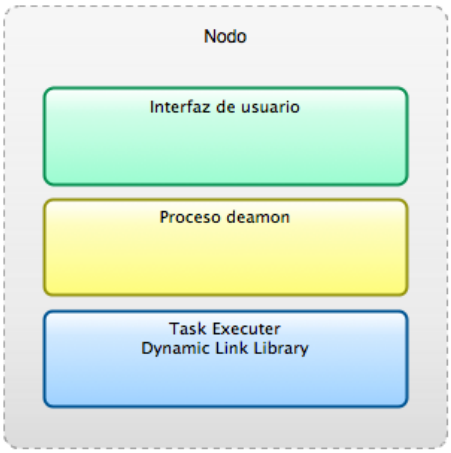
Nodos

Objetivos

Un Nodo será el proceso que forme parte del Grid programado para procesar una tarea maximizando los recursos del host donde se ejecuta. Su objetivo es simplemente registrarse en la red cuando esté operativo y procesar tareas de acuerdo a los recursos de hardware disponibles en el equipo.

Especificaciones técnicas

Básicamente, un Nodo estará conformado por una consola que hará de interfaz con el usuario y por un proceso que se ejecutará en modo Daemon. Esto significa que una vez que el usuario del host ejecute el comando para iniciar el proceso, este quedará corriendo en segundo plano sin utilizar los flujos STDIN, STDOUT y STDERR.



Esta aplicación estará compuesta por dos binarios y una biblioteca de enlace dinámico: la consola, el Daemon y el Task Executer.
El desarrollo de este componente estará orientado a la plataforma UNIX y será un único proyecto con código portable tanto para Linux como para Solaris. En el proceso de compilación y linkeo se detectará la plataforma y se procederá a realizar los enlaces que crea conveniente, sin embargo el código fuente debería ser portable. El lenguaje de programación es C++.

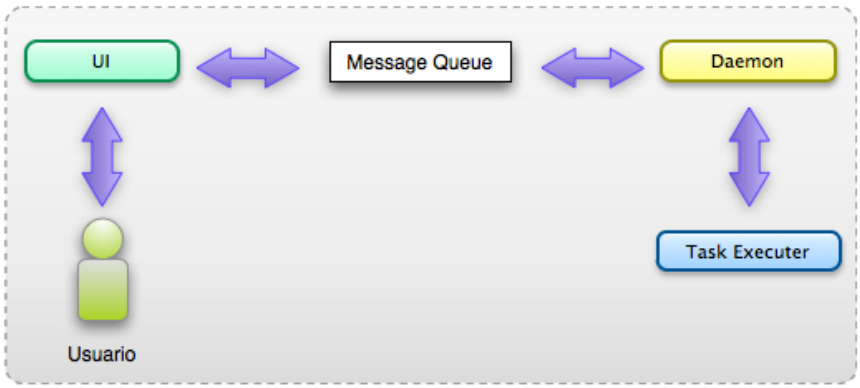
Daemon

Tendrá 2 funciones principales:

1. Establecer un canal de comunicación con otros nodos utilizando el protocolo de descubrimiento de red y mantener una tabla de estado del resto de los nodos.

2. Recibir las tareas enviadas por el Distribuidor de Carga y delegarle el procesamiento al Task Executer.

Para la comunicación con la Interfaz de Usuario, el Daemon estará a la escucha de los comandos a través de una Message Queue en común, como mecanismo de IPC (Inter Process Comunication). Toda su configuración es obtenida de su archivo de configuración en formato XML. Además, si debiera loguear algún tipo de información lo hará en su correspondiente archivo Log ubicado en el directorio de instalación.



Interfaz de usuario

Este proceso será un programa por consola que se encargará de enviar comandos al Daemon así como de recibir información del mismo y presentarla al usuario por pantalla.

El administrador podrá ejecutar los siguientes comandos (ver tabla) para interactuar con el Daemon. Toda salida será presentada por consola en un formato legible.

Comando	Operación
node start	El comando ejecutará el Nodo en modo Daemon, realizará el código de inicialización necesario y quedará a la espera de tareas. No existirán tareas incompletas que el Nodo deba continuar.
node stop	Le indica al Daemon que suspenda el procesamiento, actualizando su estado en el Grid. Las tareas actuales serán asignadas a otro nodo por el DC.
node --display --local	El binario mostrará por consola las características físicas del host donde se encuentra ejecutándose. • Memoria física total y disponible. • Espacio libre en disco.

Resultados

Una Task puede tener tres resultados posibles: *TASK_SUCCESS, TASK_COMPLETE* y *TASK_FAIL*. Dichos resultados serán informados al Distribuidor de Carga por medio de un XML de respuesta utilizando el protocolo IRC/IPC standard. Un ejemplo de dicho mensaje se muestra a continuación:

```
<taskResult task=”taskid” parent=”job1”>
   <result>
     <status>TASK_FAIL</status>
     <value>DDDDDF</value>
   </result>
</taskResult>
```

El tag “status” indica el resultado de la tarea al finalizar la ejecución. El tag “value” depende directamente del estado. Si la tarea finalizó exitosamente (*TASK_SUCCESS*) entonces dicho campo contendrá la contraseña buscada, de otro modo la contraseña no fue encontrada (*TASK_COMPLETE*). En cambio, si el resultado no es satisfactorio (*TASK_FAIL*) el tag contendrá el último valor calculado hasta el momento de la suspensión del procesamiento. Esto permitirá al Distribuidor de Carga reasignar la tarea a un nuevo Nodo y que este pueda continuar con el procesamiento.

Task Executer (Shared Library)

Es un componente interno del Nodo encargado de ejecutar los programas que cargan los recursos de una máquina. Con el objeto de poder ejecutar distintos tipos de tareas se dispuso crear este componente con una interfaz genérica para que luego pueda ser reemplazada por una implementación que procese otro tipo de tareas. Estará programado como una Shared Library de enlace dinámico y cuyo código fuente estará en ANSI C (también portable a las distintas plataformas Unix).

La implementación en esta primer etapa de proyecto será la aceptar los lotes de claves a calcular con el algoritmo MD5. Más abajo se describe la API pública que expone:

```
SHORT taskExecuter_setup(ENVIRONMENT *);
```

A través de esta llamada el componente es inicializado con datos del entorno que se consideren requeridos para poder operar. Aquí se ejecuta el código de inicialización de la biblioteca. Recibe como parámetro la dirección de memoria de una estructura que agrupa dicha configuración.

```
HANDLE taskExecuter_executeTask(TASK *, CALLBACK_FUNCTION *);
```

Se le indica al Task Executer que comience la ejecución de una Task que es enviada por parámetro. Tal estructura contiene la información necesaria para llevar a cabo dicha ejecución. El segundo parámetro es la dirección de la rutina que será llamada cuando la tarea finalice. Retorna el descriptor de la tarea en ejecución.
Esta llamada se ejecutará de forma asíncrona, esto significa que no bloqueará al cliente que la utilice, retornando inmediatamente luego de llamarla. El tipo de dato `CALLBACK_FUNCTION` corresponde con el siguiente alias: `typedef void (* CALLBACK_FUNCTION)(void *);` donde el puntero que se recibe por parámetro corresponde con el resultado de la operación.

```
SHORT taskExecuter_cancelTask(HANDLE);
```

Esta llamada cancela una tarea en ejecución. Se recibe como parámetro el descriptor de dicha tarea. Retorna el resultado de dicha operación.

```
VOID taskExecuter_shutdown(VOID);
```

En una finalización normal, el cliente debe llamar a esta rutina la cual se encarga de liberar los recursos utilizados por el Task Executer.

Descubrimiento de la red

Cada Nodo que ingresa en el Grid deberá conocer quién es el Distribuidor de Carga actual y también deberá conocer el estado del resto de los nodos. Este es un requerimiento muy importante que garantizará la aplicación de un algoritmo de elección de coordinador ante la migración de un DC.
Para el descubrimiento de la red se utilizará un protocolo basado en una serie de mensajes en los cuales se incluye un encabezado con una serie de campos especificados en el Diagrama de "Insomnio Descriptor Header".

El objetivo de esta comunicación es mantener una tabla con el estado de toda la red en cada nodo de la red. Esta información incluye por cada nodo:

- Dirección IP y Puerto para elección de Distribuidor de Carga.
- Booleano que indique si el nodo actúa como Distribuidor de Carga.
- Valor numérico que pondere la cantidad de recursos que posee el nodo.

El proceso deberá repetirse cada cierto intervalo para mantener actualizada la tabla. Dicho tiempo será especificado por archivo de configuración.

Handshake

Estas comunicaciones entre nodos ocurren por arriba del protocolo TCP/IP. Por eso una vez que se haya establecido una conexión TCP/IP el cliente que inicio la conexión enviará el string "INSOMNIO CONNECT/0.1\n\n". El nodo que aceptó la conexión responderá con un mensaje "INSOMNIO OK\n\n" indicando que se ha establecido una conexión válida en la red. Cualquier otro mensaje que no cumpla con este handshake será tomado como un cliente inválido y se procederá a rechazar la conexión.

Reglas de propagación

El proceso de comunicación para el descubrimiento de la red es el siguiente:

- Un Nodo que ingresa a la red (el usuario ejecuta el comando correspondiente) envía un mensaje PING a otro Nodo conocido por él anunciando su existencia. Esto significa que por configuración tendrá la dirección de este y establecerá una conexión en un primer momento.

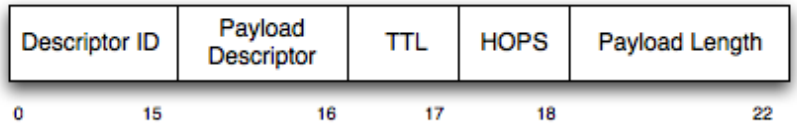
- Al recibir un mensaje PING, se responderá con un mensaje del tipo PONG. En este mensaje se envían los datos correspondientes del nodo.

- Los mensajes PING se propagan a todos los nodos conectados directamente excepto al que envió dicho mensaje. Este nodo está anunciando la existencia de un nuevo nodo a todos sus vecinos. Este patrón continuará recursivamente hasta que el campo TTL expire.

- Los mensajes PONG se propagan a todos los nodos conocidos (broadcast).

- Los mensajes duplicados se desechan.

- El campo HOPS y TTL deben cumplir con la siguiente ecuación: TTL(0) = TTL(I) + HOPS(I). Un nodo decrementará en 1 el campo TTL de un mensaje e incrementará el valor del campo HOPS en 1 antes de propagar dicho mensaje. El mensaje no será propagado cuando el TTL alcance en valor 0.



Insomnio Descriptor Header

Descriptor ID: representa el ID único del mensaje en la red.

Payload Descriptor: Tipo de mensaje.

TTL: Time To Live. Cantidad de saltos máximo que puede realizar el mensaje.

HOPS: Representa la cantidad de saltos que realizó el mensaje.

Payload Length: longitud del cuerpo del mensaje. No incluye el header.

Recomposición de la red

La red de nodos deberá tener cierta inteligencia para evitar que se formen islas (el grafo de interconexión de nodos deberá ser conexo). Para lograr mantenerla así, el nodo deberá hacer una distinción entre las conexiones salientes y las conexiones entrantes y agrupar los nodos alcanzables a través de cada una.

Por ejemplo, a través de la conexión saliente del nodo 3 este puede conoce a los nodos: 4, 5, 6, 7 y 8. En un determinado instante el nodo 4 se cae. El nodo 3 detecta este evento y como es muy probable que se haya generado una isla consultará a los nodos que se podían alcanzar a través de 4. Estos son: 5, 6, 7 y 8. Cualquier conexión hacia uno de ellos le garantizará al nodo 3 que sus nodos dependientes no queden aislados del resto de la red.

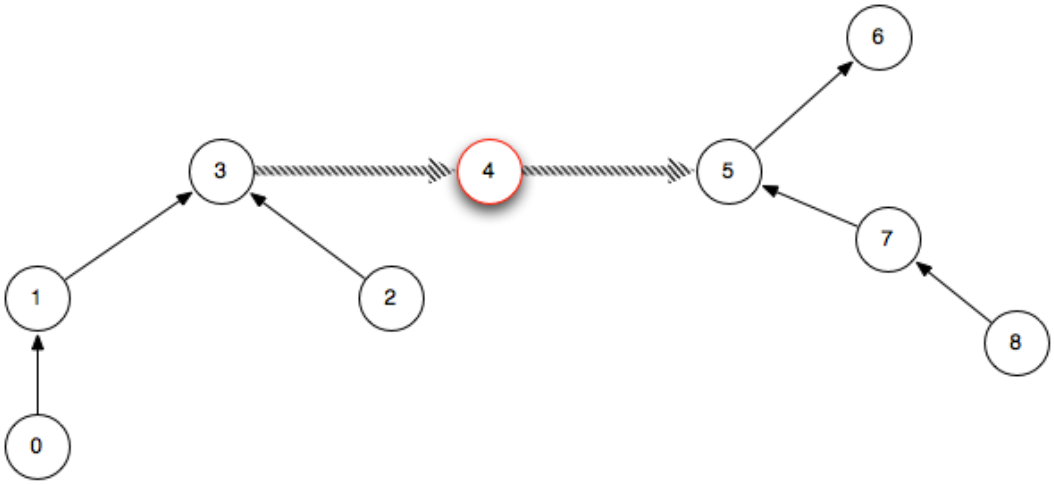


Diagrama de interconexión de nodos

Distribuidor de Carga (DC)

Objetivos

Es el responsable de obtener la descripción de un trabajo en formato XML del Web Server, interpretarlo y ejecutarlo en el Grid. El código fuente estará escrito en ANSI C portable entre los sistemas Unix.
El DC obtendrá el XML con el procesamiento a ejecutar y dado que las tareas que componen un Job pueden tener dependencias entre sí, el proceso deberá poder resolver estas dependencias y lanzar las tareas al Grid (se interpreta como el envío de la Task a un Nodo para que este la ejecute). Todas las comunicaciones hacia el Web Server se realizarán utilizando el protocolo de comunicación HTTP.

Especificaciones técnicas

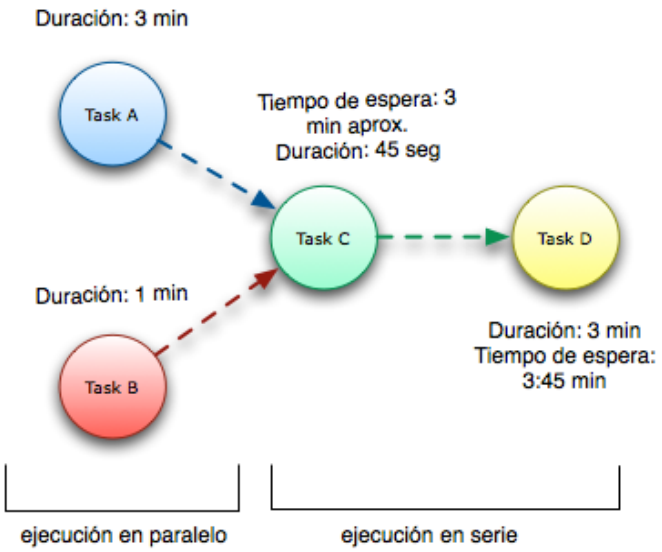
- El DC conocerá el estado de toda la red a través del protocolo de descubrimiento de red (explicado en detalle más adelante). Se requiere un mecanismo IPC con el Nodo local.
- Podrá migrar de nodo host ante la solicitud del administrador. La elección del nuevo nodo será resultado del proceso de elección de coordinador.
- Consultará periódicamente la cola de listos del Web Server para determinar si existe algún Job listo para ejecutar.
- Es un proceso hijo de un Nodo. El primer Nodo que se levante en la red será un Distribuidor de Carga.
- La mensajería que maneja es específica del proceso de elección y de asignación y procesamiento de tareas.
- Mantendrá al Web Server actualizado con su ubicación y con el estado de sus tareas.

Proceso de ejecución de tareas

El proceso estará haciendo *polling* (consultando el estado como una actividad sincrónica) de la cola de trabajos listos para ejecución del Web Server. Al recibir un Job disponible (en formato XML), el Distribuidor de Carga generará un grafo de dependencias de las Tasks que lo componen. Antes de enviar una tarea, consultará su tabla de Nodos disponibles del Grid y seleccionará el que tenga el valor de ponderación de recursos más alto. Esto garantiza que el Nodo efectivamente posea los recursos necesarios para ejecutar el procesamiento requerido. En caso de no haber Nodos disponibles esperará a que uno entre a la red.

Una tarea que depende de otra no podrá ser ejecutada si su dependencia no fue ejecutada. En caso de ser posible, el proceso podrá lanzar en paralelo todas las tareas posibles. Por ejemplo:

Se define un Job = (A, B, C, D) con el siguiente grafo de dependencias entre tareas.

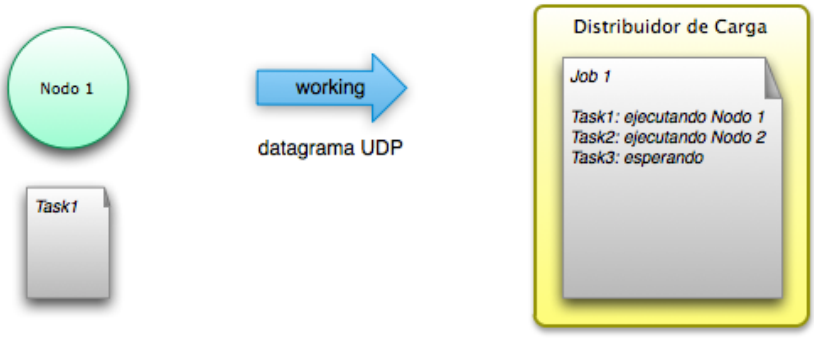


Donde C depende de A y B y la tarea D depende de C. En este caso la ejecución quedaría de la siguiente manera: el DC podrá ejecutar en paralelo A y B esperando a que ambas tareas finalicen. Al hacerlo podrá continuar solamente con C y por último con D finalizando por completo el Job.

Heartbeat DC-Nodo

En el momento que el DC entregó una Task a un Nodo con los recursos para ejecutarla, este quedará a la espera de la finalización de dicha tarea. Durante el tiempo de espera se recibirá, cada cierto intervalo, un mensaje "working" que indica que el Nodo se encuentra ejecutando dicha tarea. Al recibirlo el proceso chequeará su tabla de tareas en ejecución.

El Distribuidor de Carga tendrá un único puerto para recibir dichos mensajes parametrizables por el XML de configuración. El protocolo de comunicación será UDP. A su vez, el Nodo podrá parametrizar el intervalo entre pulso y pulso también por archivo de configuración.



Failover

Esta es una capacidad que tiene el Distribuidor de Carga para mover una tarea cuando el Nodo en el que se ejecutaba tuvo una falla. Su comportamiento es muy similar al de la cancelación de una tarea, excepto que el procesamiento en otro nodo comenzará de nuevo.
Por ejemplo, el nodo se encontraba procesando una tarea cuando en un determinado momento el hardware falla. El DC detecta un corte en el heartbeat del Nodo y elige otro Nodo más adecuado para procesar la tarea siguiendo el proceso de selección de Nodos.

Migración

Para evitar suspender el procesamiento de un determinado Job ante, por ejemplo, una actualización de hardware el Distribuidor de Carga dispondrá de un sistema de migración a otro host en el cual todos los nodos estén de acuerdo.
El procedimiento tendrá los siguientes requerimientos:

1. Se iniciará el procedimiento al recibir la señal SIGUSR1.

2. Se le comunicará a los nodos directamente conectados al nodo donde el DC se ejecuta.

3. Durante el periodo de migración no se aceptarán Jobs.

4. Solo cuando el próximo Nodo electo para tener el DC esté disponible, el ex DC le transferirá su tabla de ejecución al nuevo DC para continuar su ejecución.

Elección de un coordinador

Se utilizará el algoritmo de elección "abusón" (*The Bully Algorithm*) para elegir un único Nodo que juegue el rol de Distribuidor de Carga en un determinado momento. Es importante que todos los Nodos estén de acuerdo en la elección ya que cuando el DC abandona su cargo una nueva elección será requerida.

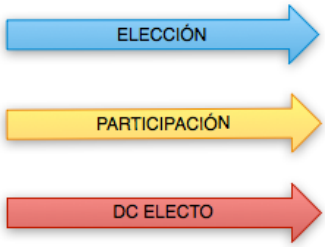
- Un determinado participante solo llama a elección una vez, sin embargo, N participantes podrían llamar a N elecciones concurrentemente (dos nodos pueden llamar a elección si detectan que el DC ha fallado). A pesar de esto, el presidente de la elección será único. Un requisito fundamental es que el Nodo elegido sea aquel que tiene la capacidad de recursos más baja (En esta simulación no habrá dos Nodos con los mismos recursos).

- Cada Nodo tendrá una variable *NODO_DC* que contiene el ID del Nodo que fue elegido presidente y por lo tanto juega el rol de Distribuidor de Carga. Cuando el proceso comienza a participar de una elección establece su variable a 0x00.

- Durante el proceso el Nodo mantendrá en espera todas las tareas que finalicen. Cuando el DC sea electo estos resultados le serán enviados.

- Al ser el DC, también realizará tareas específicas del Nodo (como el procesamiento de Tasks).

- Se definen tres tipos de mensajes en este algoritmo como se muestra en el gráfico de **Tipo de mensajes**.



Tipos de Mensajes

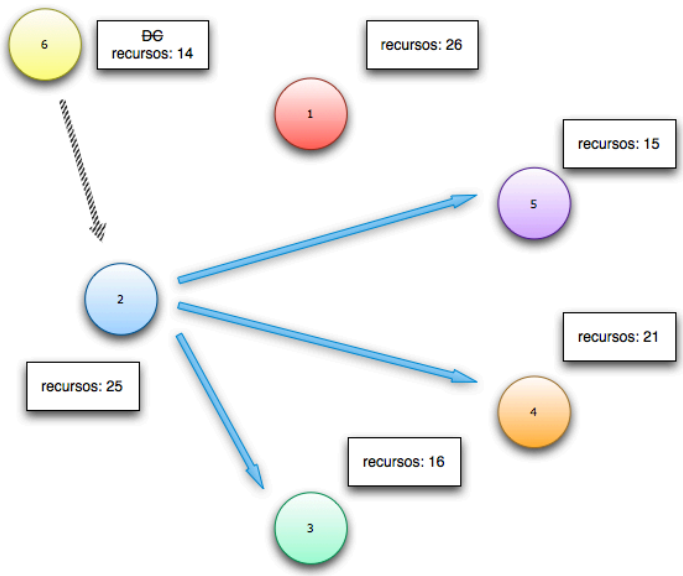
Proceso de elección

Cuando un Nodo vecino detecta el inicio del procedimiento de migración de un DC este inicia el proceso de elección actualizando su *NODO_DC* a 0x00. Este proceso se lleva a cabo de la siguiente manera:

1. Un Nodo P envía un mensaje "elección" a todos los nodos con el valor de recursos mas bajo que él.

2. Si ninguno responde, entonces el Nodo P gana la elección siendo el nuevo DC del grupo.

3. Si en cambio, el Nodo P recibe mensajes de "Participación" de los demás nodos, entonces seleccionará el de menor valor de recursos y enviará a todos los Nodos del grupo el mensaje de "DC electo" luego de esperar un cierto intervalo de tiempo en la espera de recibir otro mensaje de DC electo (otro flujo de elección).



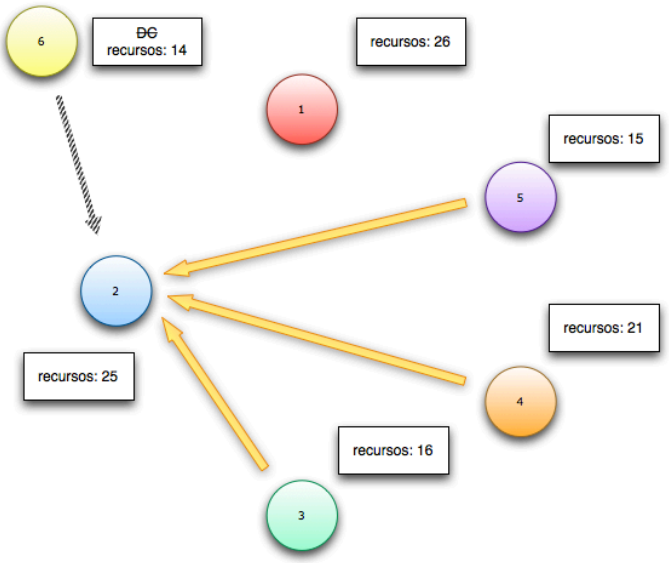
Al recibir un mensaje de "DC electo" el Nodo actualizará su *NODO_DC* con el ID del DC.

En el siguiente ejemplo existen 6 nodos con un valor de recurso asignado. En un determinado momento, el Nodo 6 y actual DC se cae (en este ejemplo solo el DC finaliza; no así el Nodo el cual seguirá funcionando a partir de la próxima elección). El primero que detecta este suceso es el Nodo 2 quien envía el mensaje de elección a todos sus nodos conocidos con menor número de recursos. El nodo perteneciente al DC no participa de la elección. **Ver figura 1**.

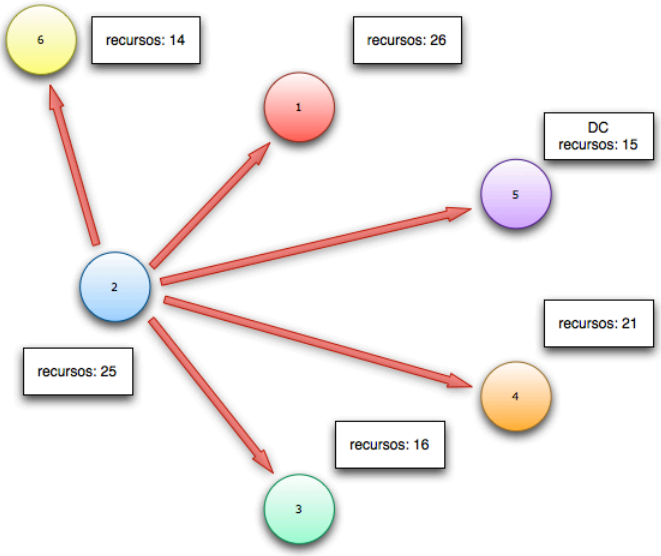
El resto de los nodos también detectarán la caída del DC lo cual generará el mismo comportamiento. Otro flujo alternativo puede ser el siguiente: el Nodo 1 observa en su lista

a los nodos con menor valor de recursos y envía el mensaje de "elección" al 2, 3, 4 y 5. El Nodo 3 solo envía al Nodo 5 y el Nodo 4 al 3 y al 5. El nodo 5 observa que no existe un nodo en este momento que tenga menor nivel de recursos por lo tanto se proclama DC enviando el Mensaje "DC electo a todos los nodos".

La **figura 2** muestra que los nodos 3, 4 y 5 se encuentran habilitados para participar de la elección enviando el mensaje correspondiente. El Nodo 2 detecta que el Nodo 5 es el que participa de la elección con el valor de recurso más bajo.



Seguimos con el primer flujo de mensajes. A continuación envía el mensaje avisando que existe un nuevo DC en el grupo. **Ver Figura 3**. Siendo el Nodo 5 el nuevo DC todos los nodos actualizarán su *NODO_DC* a 5.



Aclaraciones

- Cada nodo conoce el estado de todos los nodos de la red a través del protocolo de descubrimiento de la red.

Web Server

Será una aplicación muy simple programada en C++ que estará ejecutándose sobre la plataforma Windows. Utilizará el protocolo HTTP 1.1 para aceptar como cliente a un browser como Internet Explorer ó Firefox.
El servidor expondrá una serie de *endpoints* (punto de ingreso a un servicio, proceso ó a una cola) a través de los cuales los clientes podrán acceder a los servicios que este brinda. Sus objetivos serán:

- Consultar el estado del Distribuidor de Carga para mantener al usuario informado sobre el estado de las tareas que se encuentran en ejecución en el Grid.
- Brindarle al usuario una interfaz para cargar nuevos trabajos a ser procesados.
- Almacenar los resultados obtenidos en una base de datos interna para que luego un usuario pueda consultar dicha información.

Registro del DC

El servidor deberá conocer el estado del Distribuidor de Carga en todo momento y para lograrlo es indispensable que conozca la dirección IP del Nodo donde se encuentra ejecutando. Por lo tanto es requerido que ante la creación del DC este se registre en el Web Server enviando un mensaje a la siguiente URL:

```
HTTP POST http://direcciónIp:puerto/webserver/register.do?ip=ipdelDC&puerto=puertoDC
```

Donde los parámetros ip y Puerto indican la ubicación actual del DC en el Grid.

Consulta del DC

Al recibir un request a la siguiente URL deberá generarse un nuevo thread de ejecución y comunicarse con el Distribuidor de carga:

```
HTTP GET http://direcciónIp:puerto/webserver/status.do
```

Se deberán responder los siguientes datos en formato HTML para que el cliente los pueda presentar en pantalla:

1. Cantidad de Jobs en ejecución.
2. Cantidad de tareas por Jobs.
3. Cantidad de tareas en espera por Job.
4. Cantidad de tareas ejecutándose en el Grid por Job.

Colas de Jobs y resultados

El Web Server administrará tres colas que estarán expuestas al Distribuidor de Carga para tomar los Jobs que estén en estado listo para ejecutar, para guardar los resultados del procesamiento y para informar errores en el proceso. Las mismas serán accesibles a través del protocolo HTTP.

El proceso estará constantemente monitoreando un directorio local llamado "jobs" en espera de algún cambio. Cuando un nuevo archivo XML con la especificación de un Job es copiado a este directorio, se detectará el evento, el Job será cargado en la cola de listos y el archivo será eliminado del directorio.

La política de short-term scheduling que se seguirá para administrar la cola será HRRN (Highest Response Ratio Next) basado en el tiempo de CPU del Job. Dicho tiempo será la sumatoria de los tiempos de cada tarea, teniendo en cuenta las tareas en paralelo.

El DC consumirá esta cola para obtener los trabajos a procesar.

```
HTTP GET http://direcciónIp:puerto/webserver/readyQueue.do
```

Los resultados obtenidos de la ejecución de Job serán almacenados en la cola de resultados que será accedida en la siguiente URL:

```
HTTP POST http://direcciónIp:puerto/webserver/doneQueue.do
```

Se almacenará en la cola de errores los Jobs que no pudieron ser ejecutados por algún motivo que haya impedido su ejecución. Será accedido en la siguiente URL:

```
HTTP POST http://direcciónIp:puerto/webserver/errorQueue.do
```

El proceso tendrá un thread monitoreando las colas de resultados y de errores cada un cierto intervalo de tiempo. Al detectar un cambio consumirá el registro de la cola y lo almacenará en la base Berkeley DB de la cual tendrá una instancia.
El usuario que envió un trabajo a ser procesado en el Grid podrá consultar los resultados con un browser y la aplicación le responderá en formato HTML.

```
HTTP GET http://direcciónIp:puerto/webserver/result.do?job=idJob
```

Se consultará dicha URL para conocer el estado de un Job en particular, donde "idJob" es el identificador del trabajo que se quiere consultar. Internamente, el proceso consultará la base de datos Berkeley DB en busca del estado final del trabajo. Primero se buscará en la tabla de resultados. Si no existe, entonces es posible que haya habido un error y se chequeará la tabla de errores. Si el mismo no existe en ambas tablas se deberá delegar la consulta al Distribuidor de Carga para consultar el estado de los Jobs en ejecución (se utiliza el protocolo IRC/IPC). En cambio, si el request es enviado sin parámetros el proceso retornará todos los trabajos ejecutados hasta el momento con su correspondiente estado.
Todos los resultados serán comunicados al usuario en formato legible.

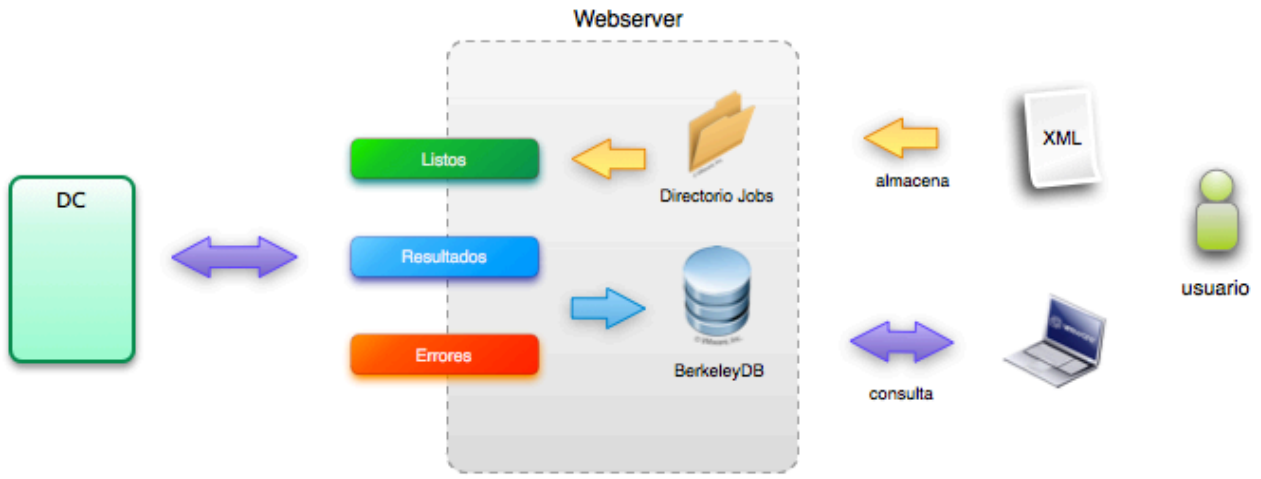


Diagrama de la Arquitectura del Webserver

El desarrollo de la plataforma estará dividido en 5 iteraciones ó sprints con una demo de la aplicación al final de cada una. En algunos casos puede ser opcional y en otros presenciales.

Sprint 1

Requerimientos

Se desarrollará la primera versión de un Nodo y del Web Server. El Nodo no formará parte de ninguna red, sino que se ejecutará en forma local. Tampoco es requerido que sea ejecutado en modo Daemon. Por el momento la lógica correspondiente con la ejecución de la tarea estará en el Nodo y no en el Task Executer.
El Web Server será un proceso simple que tomará la ubicación por línea de comando del XML con un Job de varias tareas y la pondrá en una cola de espera. El Nodo consultará esta cola cada cierto intervalo. Al detectar un Job listo para ejecutar este ejecutará todas las tareas en forma secuencial. En esta entrega no habrá dependencias entre tareas. Si se encuentra la solución la mostrará en pantalla.

Modo de testeo	
• Es recomendable poder testear con lotes de datos reales para poder finalizar la funcionalidad de luego será la del Task Executer. • Controlar el correcto uso de memoria y disco. • Tener presente flujos alternativos de finalización del proceso (terminación anormal).	
Duración (semanas)	2
Tipo	No Obligatoria, no presencial
Fecha	18/09/2010

Sprint 2

Requerimientos

En esta iteración se realizará la primera versión de un Distribuidor de Carga. A partir de ahora será el que recibirá una tarea en formato XML para que genere un plan de ejecución (elección de los nodos con más capacidad de procesamiento primero) y distribuya el procesamiento entre los nodos. Habrá un solo DC en la red.
En esta entrega ya existe la interdependencia entre tareas. Los nodos no estarán en una red sino que el DC tendrá conocimiento sobre la ubicación de cada uno de ellos (sin descubrimiento de red).
El Web Server expondrá los tres endpoints que el Nodo deberá consultar. Hasta el momento no será necesario una base de datos Berkeley DB sino que los resultados serán grabados en archivos en el filesystem.

Modo de testeo	
• Testear el correcto funcionamiento del plan de ejecución.	
Duración (semanas)	3
Tipo	Obligatoria, no presencial
Fecha	9/10/2010

Sprint 3

Requerimientos

A partir de ahora, los nodos que ingresen a la red utilizarán el mecanismo de descubrimiento de red para conocer el estado de todos los nodos. Este proceso se llevará a cabo cada cierto intervalo parametrizable. Por lo tanto el Distribuidor de Carga conocerá los nodos disponibles y el estado de cada uno.
El usuario podrá utilizar la consola para ejecutar un Nodo en modo Daemon y la lógica de procesamiento de tareas será delegada al Task Executer. Se deberá crear dicho componente como una Shared Library.

Modo de testeo
• Controlar las conexiones de procesos ajenos al protocolo.

Duración (semanas)	3
Tipo	Obligatoria, Presencial
Fecha	30/10/2010

Sprint 4

Requerimientos

El objetivo en esta iteración es lograr la elección de un coordinador en una red de Nodos. El Distribuidor de Carga recibirá la señal correspondiente por consola dando como resultado el inicio del proceso de migración a otro nodo portador. Para conseguir a tal nodo, la red llamará a un proceso de elección.
Al mismo tiempo, el Distribuidor de Carga es completamente terminado y tendrá todos los mecanismos de análisis de trabajos y distribución de tareas en el Grid, sistema de failover y migración de nodo host.

Modo de testeo
• Es muy importante el manejo de un conexión. Control sobre la apertura y cierre de conexiones TCP/IP. • Correcto manejo de señales y creación de procesos hijos.

Duración (semanas)	2
Tipo	No Obligatoria, no presencial
Fecha	13/11/2010

Sprint 5 – Entrega Final

Requerimientos

Esta es la iteración final antes de presentar la primer release del sistema. El Web Server ya posee todos los endpoints funcionales los cuales el cliente podrá consultar. Los directorios de resultados dejan de existir y son reemplazados por una base de datos Berkeley DB.

Al mismo tiempo la red de nodos podrá regenerarse ante la formación de una isla, garantizando que el poder de procesamiento de toda la red no se pierda, dando como resultado una red más inteligente.

Modo de testeo
• Comprobar el correcto control de interdependencias entre tareas. • Generar islas intencionales para testear el proceso de regeneración de la red.

Duración (semanas)	2
Tipo	Obligatoria, Presencial
Fecha	27/11/2010