UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

Exposición: Sistemas basados en Grid

Sistemas Operativos

Grupo: 06

Profesor: Gunnar Wolf

Alumnos: Cabrera Pérez Oswaldo
Ceres Martínez Hanna Sophia

1.Introducción

1.1.- ¿Qué es el Grid computing?

El Grid Computing es una infraestructura de recursos que se virtualiza, estos son: mayor poder en el CPU, aplicaciones, datos, almacenamiento y recursos de red. Está para que el cliente la utilice cuando sea necesario, sin preocuparse de dónde se localiza, en qué sistema operativo residen los datos y demás recursos, etc. [1]. Se dice que es una infraestructura porque conecta diversos recursos usando hardware y software que lo coordina y administra. Permite la computación distribuida mediante una red de recursos.

1.2.- Características de un Grid

- Los recursos no comparten un reloj físico
- Los recursos no comparten memoria
- Los recursos están geográficamente separados
- Autonomía y heterogeneidad en los recursos
- Coordina recursos que no están sujetos a un control centralizado: Los grids computacionales deben conectar recursos en diferentes dominios administrativos mediante políticas comunes.
- Uso de protocolos e interfaces abiertos, estándares y de uso general: Es importante el uso de protocolos e interfaces para definir el mecanismo básico de cómo se establecen, negocian y manejan las relaciones entre los usuarios y los recursos.
- Calidad del Servicio: Se debe llegar a un acuerdo respecto al nivel de calidad o Quality of Service (QoS).

1.3.- Tipos de Grid

- Grid computacional

Este tipo es el más común, está conformado por los ciclos de cómputo proporcionados por los procesadores de las máquinas. Estos procesadores pueden ser distintos uno del otro en su velocidad, arquitectura, memoria, almacenamiento, etc (heterogeneidad).

- Grid de datos

Este tipo de grid se utiliza para transferir, procesar y almacenar grandes cantidades de datos. Cada ordenador en el grid normalmente reserva alguna cantidad de almacenamiento para ser usado por el grid. Este almacenamiento puede ser registros o almacenamiento secundario.

- Grid de Servicios

Este grid consiste en diferentes recursos o servicios que otras máquinas no posean y sean necesarias para completar en conjunto algún trabajo como podría ser alguna licencia o software.

1.4.- Arquitectura del Grid

La arquitectura Open Grid Services Architecture (OGSA) es la responsable de ejecutar y definir diferentes servicios y protocolos web, los cuales se encargan de la autentificación, control de acceso, monitorear los recursos, etc**.** Este acceso se logra mediante el uso de un middleware y Organizaciones Virtuales (VO).

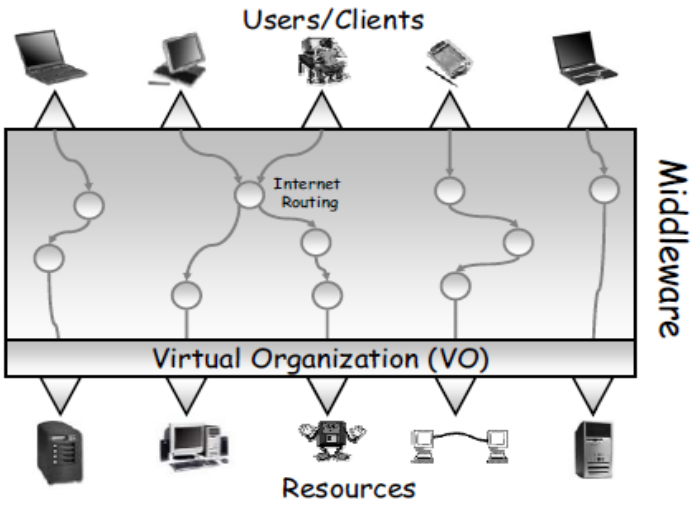

Figura 1.- Arquitectura general de un sistema Grid (Ian. J Taylor)

2.Desarrollo

2.1.- Seguridad

La arquitectura Open Grid Services Architecture (OGSA) ejecuta protocolos web, los cuales se encargan de la autentificación. En estos protocolos se implementa criptografía de llave pública y privada, para hacer esto usamos una Autoridad de Registro (RA por sus siglas en inglés).

Una RA se encarga de registrar nuevos nodos que deseen acceder al grid, esto mediante la verificación de credenciales. RA usa algoritmos tanto para el lado del usuario como para el lado del proveedor.

Algoritmos:

- **Extremo Servidor - Usuario**

Se generará en primer lugar la solicitud para establecer el enlace de comunicación, después se proporcionan las credenciales de identidad que son evaluadas por RA. Si la identidad es verificada correctamente, se da el acceso al Grid, si no, el acceso al Grid es denegado.

```
Proceso ServiceUserEnd
    GenerarSolicitud() //Generar la solicitud para establecer
    //el enlace de comunicación con RA via OGSA
    ProporcionarCredenciales()
    respuesta = EsperarRespuestaRA()
    Si respuesta = "Acceso autorizado" Entonces
        Escribir "Acceso autorizado"
        UtilizarGrid()
    SiNo
        Escribir "Acceso no autorizado"
    Fin Si
FinProceso
```

- **Extremo Servidor - Proveedor**

  Se revisa si hay solicitudes para establecer una comunicación, si hay solicitudes se reciben las credenciales de identificación para la incorporación de un nuevo nodo y se procede a la verificación de estas. Si la verificación es realizada correctamente se da acceso al Grid, si no, el acceso es denegado. En el caso donde no hay solicitudes para establecer una comunicación, se espera hasta encontrar alguna y empezar el proceso ya mencionado.

```
Proceso ServiceProviderEnd
    Si HaySolicitud Entonces
        IrPaso7
    SiNo
        IrPaso2
    Fin Si
Credenciales = RecibirCredenciales //identity cre
Si VerificarCredenciales(Credenciales) Entonces
    Escribir "Acceso Autorizado"
SiNo
    Escribir "Acceso no Autorizado"
Fin Si
FinProceso
```

2.2.- Comunicación: Broadcast y Multicast

Una infraestructura Grid se modela como un grafo, donde cada recurso es un nodo y las aristas representan la red por la cual están conectados (regularmente es mediante Internet). Cada nodo que forma parte de ella se comunica con otro procesador enviando mensajes mediante un canal de comunicación que es bidireccional. Dependiendo del tipo de

tarea a realizar o el tipo de mensaje, se puede hacer un Broadcast (enviar un mensaje a todos los nodos de la red) o un multicast (enviar un mensaje a un ciertos nodos).

Protocolo Broadcast:

- Procesador P transmite un mensaje

-El mensaje del procesador P es recibido sin corromperse por el procesador Q

-Procesador Q incluye un reconocimiento positivo para los mensajes de P en los siguientes mensajes de Q

-Procesador R recibe los mensajes de Q y es consciente de que los mensajes de P han sido reconocidos y no hay necesidad de que R los reconozca en el siguiente mensaje, R reconoce el mensaje de Q

-Si el procesador R no ha recibido el mensaje de P, el mensaje de Q alerta a R de esta pérdida, por lo siguiente R incluye un reconocimiento negativo para los mensajes de P en el siguiente mensaje de R

2.3.- Exclusión Mutua

El problema de exclusión mutua trata sobre un grupo de procesadores que necesitan acceder a algún recurso compartido que no puede ser usado o modificado a la misma vez por más de un procesador.

Cada procesador necesita ejecutar una sección de código llamada sección crítica y más de un procesador no puede estar en esta sección. En general, cada programa que realiza un procesador está dividido en las siguientes secciones:

Entrada: El código que se ejecuta antes y prepara al procesador para entrar a la sección crítica.

Crítica: El código que se ejecuta utilizando los recursos y puede modificarlos. Se necesita proteger esta sección durante la ejecución.

Salida: El código es ejecutado cuando se deja la sección crítica.

Resto: El resto del código.

Algoritmo Bakery (Leslie Lamport):

Este algoritmo hace una analogía con el proceso de compra de una panadería. Existe una cola de procesadores, cada procesador tiene un identificador único y cada uno adquirirá turno (ticket) diferente de cero que le permitirá entrar a la sección crítica.

Cada vez que un procesador intente entrar a la sección crítica, pondrá Seleccionando en verdadero y escogerá un número de ticket tomando el valor de ticket más alto más 1 y cambiará Seleccionado a Falso.

Se empieza a recorrer los procesadores que ya estaban en la cola, se tiene que esperar a que el procesador siguiente en la cola ya haya escogido su turno y que indique que ha salido de la sección crítica cambiando su variable ticket a 0 o saber que el valor de ticket es mayor al ticket[i]. Se hace la comparación de tickets tomando en cuanta también los números o identificadores de procesador, ya que puede ocurrir que dos procesadores lean el mismo ticket máximo de la cola y obtengan el mismo número de ticket, en ese caso lo que desempata es el número de procesador. Cuando se sale de la sección crítica se cambia el valor del ticket a 0.

```
Proceso BakeryAlgorithm
    Dimension  Seleccionando[100]
    Dimension Ticket[100]
    Ticket[i] = 0
    Seleccionando[i] = Falso

    //Entry section
    Seleccionando[i] = Verdadero
    Ticket[i] = Maximo(Tickets) + 1
    Seleccionando[i] = Falso
    Para j=0 Hasta n-1 Hacer //n = total de procesadores excepto el que esté ejecutando las instrucciones
        Mientras !Seleccionando[j] = Falso Hacer
        FinMientras
        Mientras Ticket[j]=0 o dupla(Ticket[j] ,j) > dupla(Ticket[i],i) Hacer
        Fin Mientras
    Fin Para
    //Se entra a la Sección Crítica
    //Se sale de la zona Crítica
    Ticket[i]=0
FinProceso
```

Código para el procesador p_i, para 0 < i < n-1, n=total de procesadores

2.4.- Sincronización de Relojes:

En el Grid Computing los recursos se encuentran situados geográficamente en lugares diferentes, lo que significa también que cada uno cuento con un con un reloj propio. La ausencia de un reloj global hace que los mensajes intercambiados entre ellas mientras trabajan en la resolución de un problema pueden sufrir retrasos, adelantos o perderse.

El tiempo es un tema importante en los sistemas distribuidos, el transcurso de él en un ordenador es dirigido por una señal de reloj propia que pudiera desincronizarse con respecto a los otros relojes. Para sincronizar el reloj de cada recurso necesitamos una fuente de referencia externa, esta fuente puede ser una máquina o varias que intercambian mensajes pidiendo la hora. Estos toman en cuenta el tiempo de ida y vuelta del mensaje, calculan y guardan el ajuste en una variable para ajustan el reloj local acelerándolo o ralentizándolo hasta que el ajuste se haya realizado.

Existen diferentes protocolos y algoritmos para llevar a cabo la sincronización, a continuación se expone uno de ellos:

Algoritmo:

En un arreglo llamado diff guardaremos la diferencia de tiempo que hay entre un procesador p_i con los demás procesadores con los que se tenga comunicación. Haremos un broadcast a todos los procesadores enviando el tiempo que marque nuestro reloj y a su vez recibiremos un mensaje con la hora que marque el reloj de los demás procesadores. Cada vez que recibamos un mensaje con el tiempo de otro procesador (p_j) guardaremos la diferencia de tiempo que hay entre ambos relojes usando la fórmula:

$$\text{diff}[j] := T + d - u/2 - HC$$

donde:

T: tiempo del procesador p_j (donde $0 < j < n - 1$, $j \neq i$, n = total de procesadores)

d: retraso que tiene un mensaje (tiempo que tarda en llegar el mensaje)

u: sesgo que existe
HC (Hardware clock): valor actual que tiene el reloj de cada procesador
Una vez que se tiene todas las diferencias calculadas, se hace el promedio de las diferencias y se tiene el nuevo reloj ajustado.

```
1   Proceso ClackSynchronization
2       Dimension diff[i]=0
3
4       EnviarHC
5
6       Mientras MensajeRecibido Hacer
7           diff[j] = T + d - u/2 - HC
8           Si MensajesCompletos Entonces
9               adj = Promedio_diff
10          Fin Si
11      Fin Mientras
12  FinProceso
```

Código para el procesador p_i, para 0 < i < n-1, n=total de procesadores

2.5.- Selección

Cuando se trabaja en equipo o en conjunto para resolver algún problema en particular, uno de los aspectos clave para lograrlo es la elección de un líder para poder coordinar y asignar las tareas a los demás participantes. Un buen algoritmo de elección deber ser capaz de escoger al mejor líder de entre todos los candidatos.

Algoritmo:

En este algoritmo cualquier nodo puede iniciar una elección de líder ya sea por diferentes razones como que el nodo líder dejó de responder, está a punto de quedarse sin batería, etc.

Cuando un nodo empieza la elección de líder envía un mensaje de elección a sus nodos vecinos. Cuando los nodos vecinos reciben este mensaje, marcan al nodo emisor como su nodo Padre y envían un mensaje de elección a sus nodos vecinos a excepción del nodo padre y se repite el proceso. Si un nodo que ya ha enviado un mensaje de elección recibe un mensaje de elección simplemente confirma la recepción del mensaje y que ya está en proceso de elección.

Una vez que ya no hay más nodos a los cuales enviar mensajes de elección, cada nodo hijo deberá de enviar información acerca de sus características y recursos para que el nodo padre pueda compararlos y seleccionar al mejor nodo y enviarlo a su vez al padre de este. Así, el nodo que empezó la elección, recibirá un grupo de los mejores nodos candidatos, los comparará y escogerá al mejor nodo para ser el líder.

```
Funcion eleccionDeLider
    nodoLider=enviarMensajeEleccion(nodosVecinos)
    Broadcast(nodoLider)
Fin Funcion

Algoritmo seleccionLider

    participacion = Falso
    mensaje = Falso
    Si RecibiMensaje y !participacion y !mensaje Entonces
        mensaje = Verdadero
        participacion=Verdadero
        nodoPadre = mensajeTrasmisor
        mejorNodo = enviarMensajeEleccion(nodosVecinos - nodoPadre)
    SiNo
        confirmarRecepcion
        Mientras RecibirConfirmaciones Hacer
        Fin Mientras
        enviarConfirmacionPadre
        enviarInfoRecursos
        enviarMejorNodo
    Fin Si

FinAlgoritmo
```

2.6.- Consenso

El problema del consenso en un sistema consiste en que todos los procesadores coincidan en un mismo valor o en una decisión. El problema de los Generales Bizantinos aborda el tema del consenso y trata de lo siguiente: Existe un grupo de generales de un ejército bizantino que rodean una ciudad enemiga, cada general está al cargo de una tropa existen solo dos órdenes: atacar o retirarse. Los comandantes se comunican mediante mensajes, pero existe la sospecha de que algunos de ellos sean traidores, alguno podría enviar un mensaje diciendo que su tropa atacará cuando realmente se retirará, poniendo en desventaja a las demás tropas y aumentando sus posibilidades de fracaso.

De este planteamiento nacen lo que son las fallas bizantinas las cuales están divididas en dos categorías. Una es “fallar y parar” en donde un nodo falla y deja de funcionar, y la otra categoría es falla arbitraria del nodo, las fallas arbitrarias más comunes son:

- Fallar al momento de mandar un resultado
- Responder con un resultado incorrecto
- Responder con un resultado incorrecto a propósito

En general, se dice que puede haber un consenso si en el sistema se detectan los procesadores que pueden llegar a fallar o a mentir y estos no son mayor a la tercera parte del total de procesadores.

El algoritmo de Tolerancia práctica a fallas bizantinas (pBFT por sus siglas en inglés) fue creado para resolver este tipo de fallas y es el siguiente:

- El cliente envía su petición al nodo líder
- El nodo líder hace un broadcast de la petición a los nodos secundarios (o de respaldo)
- Todos los nodos (líder y de respaldo) realizan la tarea o servicio y envían su resultado al cliente
- El servicio es realizado exitosamente cuando el cliente recibe ‘m + 1’ respuestas de diferentes nodos con el mismo resultado, donde m es igual al número máximo de nodos que pueden fallar.

3.Conclusión

Cada día los problemas del mundo actual son más complejos y se necesita más poder computacional para poder resolverlos. El coste de construir una máquina con N veces la potencia de una sola puede costar N veces más o puede ser imposible de construirla.

La meta del Grid Computing es reunir la mayor cantidad de recursos computacionales para ponerlos a disposición de quien los necesite a un bajo coste. Las ventajas que ofrece el Grid han hecho que su popularidad aumente y sea implementado en proyectos y áreas de todo tipo.

Garantizar el correcto funcionamiento de un Grid no es una tarea fácil, es por eso que se han creado (y se siguen creando) diferentes algoritmos y protocolos que permitan una buena administración, coordinación, seguridad, comunicación, tolerancia a fallos, etc.

4.Bibliografía

1. Aguilar, C. (2005). *Grid Computing*. Universidad Nacional del Nordeste, Argentina. Recuperado de http://exa.unne.edu.ar/informatica/SO/Gridmonogcarol.pdf
2. Kshemkalyani, A. & Singhal, M.. (2005). *Distributed Computing: Principles, Algorithms and Systems*. United States.
3. Juhász, Z., Kacsuk, P., & Kranzlmuller, D. (Eds.). (2004). Distributed and Parallel Systems: cluster and grid computing. Springer Science & Business Media.
4. Attiya, H. & Welch J.. (2004). *Distributed Computing Fundamentals, Simulations and Advanced Topics* . United States: John Wiley & Sons, Inc.
5. Kumar, M. & Yadav, P. (2017). *Security of Grid Computing: A Cryptographic Approach*. International Journal of Computer Applications. Recuperado de https://www.ijcaonline.org/archives/volume164/number6/singh-2017-ijca-913646.pdf
6. Navarro, L. & Marqués Joan.(s.f).*Sincronización, tolerancia a fallos y replicación.* Universitat Oberta de Catalunya.Recuperado de https://www.academia.edu/34717226/Sincronizaci%C3%B3n_tolerancia_a_fallos_y_replicaci%C3%B3n
7. Autor desconocido. (2019, 12 de diciembre). practical Byzantine Fault Tolerance(pBFT). *GeeksforGeeks.* https://www.geeksforgeeks.org/practical-byzantine-fault-tolerancepbft/
8. Taylor, I. J., & Harrison, A. (2006). From P2P to Web services and grids: peers in a client/server world. Springer Science & Business Media.
9. Abbas, A. (2004). *Grid Computing: A Practical Guide to Technology and Applications.*
10. Coulouris, G. F., Dollimore, J., & Kindberg, T. (2005). Distributed systems: concepts and design. pearson education.
11. Agrawala, V., Moser, L. & Melliar.Smith, P. (1990). *Broadcast Protocols for Distributes Systems.* IEEE.