

Telemática

Load Balancer

Sebastián Arcila Valenzuela (*sarcilav@eafit.edu.co*) y Sergio Botero Uribe (*sbotero2@eafit.edu.co*).

Resumen—Documentación de la práctica del load balancer para servidores web que implementa 4 métodos de balanceo. Esta documentación contiene los diagramas requeridos para la entrega y la explicación de los métodos.

I. INTRODUCCIÓN

Los métodos de balanceo de cargas surgen como la solución para equilibrar el trabajo realizado por cualquier unidad que forme parte de un grupo que trata de prestar un servicio como uno solo. El balanceo de carga se usa en discos duros, procesadores, redes, donde para el caso de los procesadores, la unidad que tiene menor número de tareas o que tiene más capacidad para asumir otra tarea obtiene el trabajo siguiente. Los criterios para determinar quien debe recibir esa nueva tarea se determina mediante algún método preestablecido, de los cuales el más sencillo es el método de Round Robin. En la práctica el objetivo era implementar uno de estos balanceadores de carga para un grupo de servidores web, en donde la carga son las peticiones a los recursos que se encuentran replicados en este grupo con la implementación de cuatro métodos diferentes para solucionar este problema. Los métodos implementados son el Round Robin, menor número de conexiones actuales, menor carga en servidor y por último un método propio propuesto por el grupo. Los pasados métodos de balanceo se encuentran explicados y acompañados de un pseudo código sencillo en este documento. Más adelante se explica también la forma en que se opera el balanceador de carga implementado y bajo que condiciones se pueden obtener los resultados deseados.

II. PROTOCOLOS IMPLEMENTADOS

III. DIAGRAMA DE SECUENCIAS UML

IV. MÁQUINA DE ESTADO FINITO

V. PSEUDOCÓDIGOS DE LOS MÉTODOS

En todos los métodos se espera leer un arreglo con las direcciones de servidores que estará contenida en un archivo de configuración.

■ Round Robin.

▷ Idea: Round Robin es el método más sencillo, ya que simplemente lleva la cuenta de cuál fue el último servidor al cual se envió petición y en base a eso envía al siguiente en la lista. La implementación se basa en un contador que se incrementa y se le aplica el módulo con el número de servidores.

```
RoundRobin
read( servers_array )
num_servers = length( servers_array )
server_no = 0
```

```
while incoming_connection
fork
read( server_no )
connect2( server_array[server_no] )
server_no = server_no + 1
server_no = server_no \% num_servers
write( server_no )
exitfork
endwhile
```

endRoundRobin

- Menor número de conexiones. △ Primera aproximación a este método.

△ Idea: Este método se basa en una tabla que guarda el número de conexiones activas que tiene cada servidor, y entrega la nueva conexión al servidor que tenga el menor número.

LeastConnection

```
read( server_array )
connections_array

while incoming_connection
fork
read( connections_array )
index = lower_value( connections_array )
if( connect2( server_array[index] ) )
connections_array[index] ++
write( connections_array )
write( index )
endif
exitfork
read( connections_array )
read( index )
connections_array[index] --
write( connections_array )
endwhile
```

endLeastConnection

△ Aproximación definitiva a este método.

▷ Idea: Esta implementación requiere de la presencia de un agente en cada servidor que se encargará de enviar básicamente la información sobre el número de conexiones TCP activas para que el método pueda enviar la conexión al servidor que se encuentre con menos de este trabajo en el momento que se requiera.

```
LeastConnections2
read (agents_array )
```

```

read( servers_array )

while incoming_connection
  fork
    good_server = get_tcps(agent_array[0])
    it=1
    server_no
    while i < length( agents_array )
      server_tcps=get_tcps(agent_array[it])
      if ( server_tcps < good_server )
        server_no = it
      endif

      it = it + 1
    endwhile
    connect2( server_array[server_no] )
  exitfork
endwhile

endLeastConnections2

```

■ Carga en servidor.

▷ Idea: Esta implementación requiere de la presencia de un agente en cada servidor que se encargará de enviar básicamente la información sobre las cargas del procesador para que el método pueda enviar la conexión al servidor que se encuentre menos ocupado en el momento que se requiera.

```

ServerLoad
  read (agents_array )
  read( servers_array )

  while incoming_connection
    fork
      good_server = get_load(agent_array[0])
      it=1
      server_no
      while i < length( agents_array )
        server_load=get_load(agent_array[it])
        if ( server_load < good_server )
          server_no = it
        endif

        it = it + 1
      endwhile
      connect2( server_array[server_no] )
    exitfork
  endwhile

endServerLoad

```

■ Método del grupo.

△ PS: En los pseudo códigos se puede ver que en muchos pasos se debe escribir y leer información, esto se debe a que mediante el uso de procesos que lo que hacen es una copia idéntica y no tendría sentido usar una variable para manejar la información porque esta no podría ser vista por

los otros procesos.

VI. ARQUITECTURA DEL SISTEMA

VII. PRUEBAS

Todas las pruebas del balanceador de cargas por Round Robin fueron a pequeña escala, el ambiente de pruebas fueron dos portátiles, donde cada portátil corría dos servidores web con páginas de inicio de desarrollos en blanco de Ruby on Rails y drupal. Una de las máquinas ejecutaba el balanceador de cargas y desde las dos se hacían peticiones que recibía el balanceador de carga.

Las máquinas corrieron con estas configuraciones:
Portátil 1: IP:10.0.1.5

| Aplicación | Puerto |
|--------------|--------|
| ROR | 4000 |
| Drupal | 5000 |
| LoadBalancer | 80 |

Esta máquina realiza las peticiones mediante un explorador Firefox.

Portátil 2: IP:10.0.1.4

| Aplicación | Puerto |
|------------|--------|
| ROR | 4000 |
| Drupal | 5000 |

Esta máquina realiza las peticiones mediante un explorador Safari.

Las pruebas realizadas al método de balanceo LBS que requiere un agente en el servidor se realizó con dos máquinas, un portátil y un servidor. En las dos máquinas se corrió una aplicación en blanco, para el servidor se corrió en Apache y en el portátil una aplicación en blanco en Ruby on Rails con servidor Webrick.

La prueba consistió en correr un “fork bomb” con el siguiente código en C:

```

int main()
{
  int i=10000000;
  while(i--)
    fork();
  return 0;
}

```

El cual al cabo de unos segundos eleva la carga del procesador del servidor a un punto en el que podíamos observar cuando dejaba de enviar las peticiones al servidor y cambiaba al portátil, también después de detener la “bomba de procesos” veíamos como el balanceador enviaba las nuevas peticiones al servidor.

VIII. CONCLUSIONES

El aspecto más interesante de esta práctica podría ser el hecho de que había que llegar a otro nivel de abstracción

para comprender todo el funcionamiento de las aplicaciones para redes y web en este caso. Fue muy claro que las dificultades se presentan dependiendo del lenguaje en que se decida trabajar, puesto que el manejo de sockets y esquemas requeridos para desarrollos en telemática requieren tener mayor dominio en lenguajes como C o C++, mientras que hay otros que proveen todas las herramientas para que el trabajo con estas nuevas funciones sea casi transparente. A nivel de la se necesitó investigar conceptos que no se habían visto anteriormente en la carrera, todo el manejo de procesos, hilos y segmentación en cuanto a las peticiones fueron grandes retrasos en el desarrollo de la práctica.

REFERENCIAS

- [1] GitHub™ - Social codign
<http://github.com/sarcilav/loadbalancer20092>
<http://github.com/sergiobuj/loadbalancer20092>
Sitios donde se encuentra todo el desarrollo y código e la práctica.