**Gestion de memoria en debian mediante openjdk y oracle java**

**Integrantes : Jaime David Mejia Quintero**

**Eduar Cortes Tamayo**

**Sebastian Eusse Bermudez**

**Docente : Sergio Armando Gutierrez Betancur**

**Universidad Nacional de Colombia**

**Sede Medellin**

**2015 2**

**Introduccion**

El proceso de gestion de memoria en linux es practicamente igual para la mayoria de distribuciones de esta, pueden variar pequeñas cosas pero que en terminos relativos no tiene mucha diferencia. Datos que se pueden conseguir con los comandos predeterminados como “vmstat”,”top”,”htop”,”jstat”. Una de las cosas importantes es como evaluar el proceso de gestion de memoria, mediante la configuracion minima de debian (una de las distribuciones ligeras de la rama de linux) se logra que los paquetes externos no interfieran en el proceso de memoria al momento de analizar datos, sin embargo. Luego de que se instale como esta en “version de consola” toca extraer los datos, igualmente para la instalacion del netbeans. Por eso se instala el Gnome Desktop.

El proceso de gestion de memoria en linux se toma por proceso de tablas por niveles, Un directorio de paginas (tamaño de una pagina, es la zona mas externa), el directorio intermedio de paginas(puede tener varias paginas, es el intermediario entre el directorio de paginas y la tabla de paginas), finalmente la tabla de paginas. Esta ultima puede obtener varias paginas.

**Directorio de paginas :** un proceso activo tiene un solo directorio de páginas que es del tamaño de una página. Cada entrada en el directorio de páginas apunta a una

página del directorio intermedio de páginas. Para un proceso activo, el directoriode páginas tiene que estar en la memoria principal.

**directorio intermedio de paginas:** este directorio puede ocupar varias páginas y cada entrada de este directorio apunta a una página de la tabla de página.

**Tabla de páginas:** esta tabla de páginas también puede ocupar varias páginas, y cada entrada de la tabla de página hace referencia a una tabla virtual del proceso.

La direccion virtual de linux ( es la que finalmente decide el movimiento de los procesos que estan siendo utilizados, se tiene en cuenta los tres primeros factores. Tambien se tiene en cuenta un ultimo factor que es el desplazamiento entre cada uno de los valores. Este valor de desplazamiento es diferente para la arquitectura de 64 bits y la arquitectura de 32 bits.

Las paginas se encuentran cargadas constamente en diferentes bloques, aun asi la pagina 1 es diferente a la ultima pagina (tomando por ejemplo de 32 paginas). Para esto se organiza diferentes algoritmos para pasar de una pagina a otra. En este caso debian utiliza el algoritmo de reloj asociando el bit de uso y el bit de modificacion para las cosas antiguas y las cosas nuevas.

**Algoritmo de Garbage Collection :** Antes de hablar de las diferentes estrategias que se pueden tomar para el algoritmo, hablemos de la definicion. Practicamente el termino Garbage(basura) es la defincion de los codigos o segmentos los cuales no estan determinados y por lo tanto pueden ser eliminados. Cosas como variables no inicializadas en ningun momento del codigo o matrices que estan en null, pueden ser eliminadas por el garbage collection.

1) **Cuenta de referencias :** Cualquier objeto con un CR (contador) de cero, es recolectado.

Cada objeto creado se le coloca este contador, hay que tener en cuenta de que existe una raiz (CR = 0 ) a partir de este se empieza a incrementar el contador por cada objeto creado y adjudicado. Cuando se elimina un objeto el contador de los demas se descuenta (los objetos asociados al objeto recolectado). Se tiene en cuenta tambien que si el objeto era la raiz de alguna de las ramas sus hijos tambien son eliminados (por ejemplo instrucciones de codigo inalcanzables).

2) **Barrido de marcas :** Solamente es invocado cuando no hay memoria entonces se quiere tener espacio. No es totalmente practico debido a que detiene la aplicacion (en sistema de tiempo real es impractico).

El barrido de marca se toma en dos partes, primero tiene que recolectar todos los objetos que tienen que ser eliminados, finalmente los objetos son recolectados y los campos de memoria se unifican(Este barrido se logra con compactacion, existe el barrido de marca sin compatacion).

3) **Recogida de copias :** Practicamente se tiene en cuenta que el algoritmo se limita en dos partes. En una de las partes (donde practicamente se trabaja) es el algoritmo donde se empiezan a mirar los objetos alcanzables, si el objeto es alcanzable entonces lo pasa a la siguiente parte, finalmente los objetos inalcanzable se quedan en la primera parte. Luego se intercambian los roles, Una de las ventajas es que la fragmentacion esta dado. Pero como se van probando uno a uno las partes alcanzables o inalcanzables tienen mucha diferencia en el heap cuando es muy grande.

**Dimensionamiento del Heap :** Para el JVM controlar la pila del dimensionamiento del heap es posible. En general con cierto parametros se puede controlar. Estos parametros son :

xms : Tamaño minimo dimensionamiento del heap (dinamico)

xmx : Tamaño maximo dimensionamiento del heap (dinamico)

Los metodos son los siguientes :

XX: MinHeapFreeRatio = máximo

* -XX: MaxHeapFreeRatio = máximo
* -XX: NewRatio = relación
* -XX: NewSize = tamaño
* -XX: MaxNewSize = tamaño
* -XX: + AggressiveHeap

**Informacion del documento**

Para el vmstat y el jstat se empezo a monitorear los procesos constantemente cada 5 seg. Para la configuracion del garbage collection se decidio utilizar tanto el sistema para el “Colector serial” y el “Colector en paralelo”. Para explicar las razones :

**Garbage collection serial :** Utiliza un unico hilo para realizar todo el proceso de recoleccion de basura, se tiene en cuenta que como es solo un hilo es el mas indicado para un unico procesador. Ya que el en trabajo tenemos la especificacion de solamente utilizar un unico procesador sera bastante indicado y posiblemente el mejor.

**Garbage collection paralelo :** Esta si utiliza varios hilos, principalemente para los multicore puede servir, ademas. Grandes informaciones de datos puede servir, hay que tener en cuenta que como se utiliza varios hilos puede haber una sobrecarga de informacion en cada uno de los hilos.

Hay que tener en cuenta las definiciones para el openjdk y el oracle java. Practicamente la diferencia se realiza en que el openjdk es la version open source del oracle java proporcionada por Sun. Algunos componentes pueden variar puesto que ya oracle lo tiene como ambito comercial. Tambien el codigo maquina (el codigo final luego de ser compilado) es diferente.

Para la parte del heap adaptable fue dificil la decision, especialmente porque como uno podra especificar el maximo posible?. Segun la pagina de java para arquitectura de x64 el maximo a utilizar es de 2GB en cambio el de x32 es de 64MB. Aun asi, nuestra maquina virtual solo posee 512 MB de RAM. Aunque en la compilacion se deja utilizar los 2GB de memoria en teoria. En tiempo de ejecuccion si la maquina llenase el espacio saltaria una excepecion de memoria diciendo que no hay espacio disponible. Para evitar este problema fijamos que el minimo Xmin : seria de 128MB y el maximo Xmax : 512 MB. En cuanto a las generaciones con el siguiente codigo *-XX:NewRatio=4.* O modificando los valores xmin y xmax en la relacion. Es decir xmin : 32mb y xmanx : 128mb . Esto nuevamente se hace con el objetivo de que no se salga fuera de memoria.

Los resultados fue que con el heap fijo, la parte de la memoria vieja estaba llena mientras la parte del eden se seguia llenando practicmente de la misma manera que con el heap

adaptable tanto para el oracle java como para el openjdk.

**Resultados**

Para la primera parte se comprobo los resultados entre el GC en serie y el GC en paralelo (con 2 hilos) para el openjdk y el oracle, con los comandos “vmstat” y “jstat”. Para el openjdk el GC en serie mostraba mejora en la administracion de memoria con mucho mayor memoria libre y swpd que el GC en paralelo. Igualmente ocurria en el oracle. Aunque ambas contenian en diferentes tiempos mejoraras una que otra pero en general, el oracle Java lleva una mejora en el manejo del swap y memoria que el openjdk. Probablemente se deba al hecho de como se puede optimizar el vmstat en codigo maquina de cada uno de los componentes ya que se tendra que ejecutar el codigo constantemente.

Para el tema de las recomendaciones realmente son pocas, los parametros que se manejaron son bastante precisos y poco variantes. En el caso del GC tambien se puede intentar probar The Concurrent Mark Sweep (CMS) Collector (orientada a eventos con pausa) o el G1 que es la combinacion de los 3 anteriores. Lo mas probable es que si estos GC fueran implementados y analiados el oracle Java tenga ventaja sobre el openjdk.

En el ambito del sistema operativo, si utilizamos multiples nucleos podria mejorar el caso de exito del GC pararelo para la distribucion adecuada de hilos con respecto a los procesadores emulados por el JVM, el tamaño de la memoria Ram tambien puede ser importante, en general si el proceso esta activo dentro de la ram puede ser llamado mas rapido que si no esta en ella.

La memoria virtual o el swap tambien debe de ser aumentada. Practicamente como no se puede emular todo en memoria real (se puede decir que esta llena) entonces aumentar la swap ayudara a emular la parte real (si poseemos mayor swap entonces procesos mas grandes podran ser emulados).

Para el heap no hay mucho que decir, ya que el heap es adaptado dependiendo de la situacion podra llegar hasta su maximo si es necesario, en cambio el heap fijo si se necesita mas espacio de memoria puede haber problemas(si el GC no encuentra que eliminar podria quedarse sin crear procesos adicionales).

Lo mejor es buscar un rango bastante pequeño entre la nueva generacion y la antigua, mayores espacios entre la brecha de generaciones dara lugar a un mayor gasto de Cpu para la organizacion de los datos.

**Referencias**

**1)10 opciones de la JVM:** <http://jesusramirezguerrero.com/2014/08/05/10-opciones-de-la-jvm-ejemplos-de-uso/>

# **2)Frequently Asked Questions About the Java HotSpot VM :**

[http://www.oracle.com/technetwork/java/hotspotfaq-138619.html#64bit\_heap\_defaults](http://www.oracle.com/technetwork/java/hotspotfaq-138619.html" \l "64bit_heap_defaults)

# **3)**[**Java Memory Configuration and Monitoring (3rd Act)**](https://blog.codecentric.de/en/2011/03/java-memory-configuration-and-monitoring-3rd-act/) **:**

# <https://blog.codecentric.de/en/2011/03/java-memory-configuration-and-monitoring-3rd-act/>

# **4)Instalando por red desde un CD minimo :**

# <https://www.debian.org/CD/netinst/index.es.html>

# **5)Java GC ergonomics :**

# [http://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/ergonomics.html#ergonomics](http://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/ergonomics.html" \l "ergonomics)

# **6)Gestion de memoria de intercambio en linux (swap)** :

# <http://www.alcancelibre.org/staticpages/index.php/como-swap-linux>

# **7)Perfomance Tuning guide :**

# https://docs.oracle.com/cd/E19159-01/819-3681/abeik/index.html