튜닝 및 모니터링

HP JVM 튜닝 옵션

2013. 11. 01

TmaxSoft

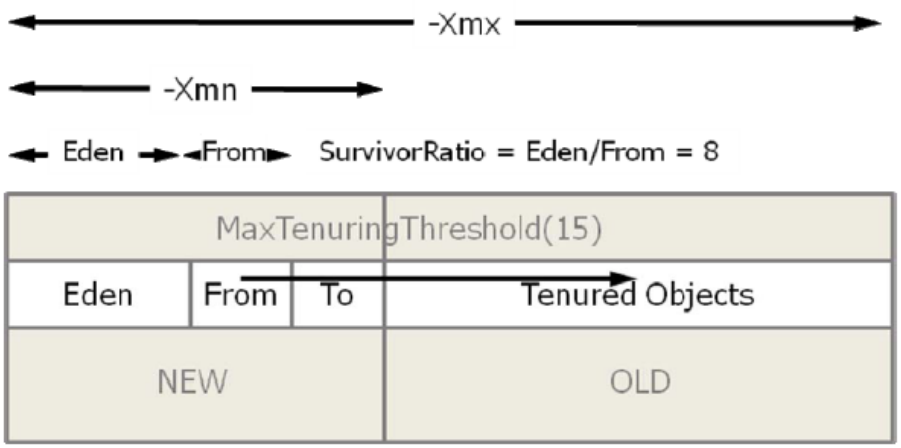
목차

1. 개요

HP JVM 의 특징을 살펴보고, TroubleShooting 방법과, 실제 Site 튜닝사례를 살펴보도록 한다.

2. JVM 특징 소개

JVM 메모리영역



영역	Garbage Collection 특성
New Space	Scavenge garbage collection Copying garbage collector Fast
Old Space	Full garbage collection Mark-Compact copying garbage collector Not so fast

3. JVM 주요 옵션 소개

GC command-line options

-Xmn<size>

new generaton heap 사이즈 결정. (-XX:NewSize=N 설정을 대신하는 옵션.)

사용 예: -Xmn64m

-Xms<size>

memory allocation pool 의 초기 사이즈 결정, 단위 byte. 1024의 배수로서 1MB 보다 커야 한다.

사용 예: -Xms6291456

-Xms6144k
-Xms6m
기본 값: -Xms2m

-Xmx<size>
memory allocation pool 의 최대 사이즈, 단윈 byte. 1024의 배수로서 2MB 보다 커야 한다.
사용 예: -Xmx83886080
-Xmx81920k
-Xmx80m
기본 값: -Xmx64m

-XX:SurvivorRatio=<size>
eden/survivor space 사이즈의 비율.
기본값은 8로써 enden 이 from 과 to의 8배를 의미한다.
Xmn / (SurvivorRatio + 2) = size of from and to, each
(Xmn / (SurvivorRatio + 2)) * SurvivorRatio = eden size

-XX:NewSize=<size>
new generation 의 기본 size(단위 byte).
(For 1.2, specify KB only). Not supported in 1.3.

-XX:MaxNewSize=<size>
new generation 의 최대 size 설정(단위 byte).
1.2 에서는 KB 로만 설정 가능, 1.3 에서는 지원치 않음.

-XX:MaxPermSize=<size>
permanent generation 의 최대 size 설정(단위 byte).
SDK 1.2.2 에서는 kbyte 단위의 integer 값만 유효 함.
기본 값: 64MB

-XX:PermSize=<size>
permanent space 의 초기 size 설정(단위 byte).
1MB 보다 큰 1024 의 배수로 설정해야 한다.
사용 예: -XX:PermSize=6291456
-XX:PermSize=6144k
-XX:PermSize=6m
기본 값: -XX:PermSize=16m (1.4 and later)

-verbosegc

garbage collection 결과를 stdout steam 에 출력.

출력 형태 :

[%T %B->%A(%C), %D]

항 목	출력내용
%T	"GC: " scavenge 일 경우 "Full GC:" full garbage collection 일 경우
%B	garbage collection 수행 전 Java heap 사용량, 단위 KB
%A	garbage collection 수행 후 size, 단위 KB
%C	현재 Java heap 의 전체 용량, 단위 KB
%D	collection 수행 시간, 단위 seconds.

-Xverbosegc<options>

garbage collection 전, 후 Java Heap 내 공간의 세부적인 정보를 출력하는데 사용된다.
1.3.1.14 와 1.4.2.05 이후에서는 정의된 파일명에 process id 가 자동으로 추가되어진다.

옵션 사용 문법:

-Xverbosegc[:help]|[0|1][:file=[stdout|stderr|<filename>]]

Java 5.0 –Xverbosegc:help 내용

```
-Xverbosegc<options>
The syntax is


-Xverbosegc usage: -Xverbosegc[:help]|[0|1][:file=[stdout|stderr|<filename>]]

    :help prints this message.

    0|1 controls the printing of heap information:
        0   Print after every Old Generation GC or Full GC
        1   (default) Print after every Scavenge and Old Generation GC
            or Full GC


    :file=[stdout|stderr|<filename>] specifies output file
        stderr    (default) directs output to standard error stream
        stdout     directs output to standard output stream
        <filename> file to which the output will be written
```

```
At every garbage collection, the following 20 fields are printed:
<GC: %1 %2 %3 %4 %5 %6 %7 %8 %9 %10 %11 %12 %13 %14 %15 %16 %17 %18 %19 %20>

 %1:  Indicates the type of the garbage collection.
      1: represents a Scavenge (GC of New Generation only)
         %2: indicates if this is a parallel scavenge.
             0: non-parallel scavenge
             n(>0): parallel scavenge, n represents the number of parallel GC threads

      2: represents an Old Generation GC or a Full GC
         %2: indicates the GC reason:
             1: Allocation failure, followed by a failed scavenge, leading to a Full
GC
             2: Call to System.gc
             3: Tenured Generation full
             4: Permanent Generation full
             5: Scavenge followed by a Train collection
             6: Concurrent-Mark-Sweep (CMS)eneration full
             7: Old generation expanded on last scavenge
             8: Old generation too full to scavenge
             9: FullGCAlot
            10: Allocation profiler triggered
            11: JVMTI force GC
            12: Adaptive Size Policy
            13: Last ditch collection

      3: represents a complete background CMS GC
         %2:  indicates the GC reason:
             1: Occupancy > initiatingOccupancy
             2: Expanded recently
             3: Incremental collection will fail
             4: Linear allocation will fail
             5: Anticipated promotion

      4: represents an incomplete background CMS GC
           (exited after yielding to foreground GC)
         %2:  n.m
             n indicates the GC reason:
               1: Occupancy > initiatingOccupancy
               2: Expanded recently
               3: Incremental collection will fail
               4: Linear allocation will fail
               5: Anticipated promotion
               6: Incremental CMS
             m indicates the background CMS state when yielding:
               0: Resetting
               1: Idling
```

```
                2: InitialMarking
                3: Marking
                4: FinalMarking
                5: Precleaning
                6: Sweeping
                7: AbortablePreclean

  %3:  Program time at the beginning of the collection, in seconds

  %4:  Garbage collection invocation. Counts of background CMS GCs
       and other GCs are maintained separately

  %5:  Size of the object allocation request that forced the GC,
       in bytes

  %6:  Tenuring threshold - determines how long the new born object
       remains in the New Generation

  The report includes the size of each space:
     Occupied before garbage collection (Before)
     Occupied after garbage collection (After)
     Current capacity (Capacity)
  All values are in bytes

  Eden Sub-space (within the New Generation)
    %7:  Before
    %8:  After
    %9:  Capacity

  Survivor Sub-space (within the New Generation)
    %10:  Before
    %11:  After
    %12:  Capacity

  Old Generation
    %13:  Before
    %14:  After
    %15:  Capacity

  Permanent Generation (Storage of Reflective Objects)
    %16:  Before
    %17:  After
    %18:  Capacity

    %19:  The total stop-the-world duration, in seconds.

    %20:  The total time used in collection, in seconds.

Could not create the Java virtual machine.
```

Java 1.4.1, 1.4.2 –Xverbosegc:help 내용

```
-Xverbosegc<options>
The syntax is


-Xverbosegc usage: -Xverbosegc[:help]|[0|1][:file=[stdout|stderr|<filename>]]
```

```
    :help prints this message.

    0|1 controls the printing of heap information:
        0   Print after every Old Generation GC or Full GC
        1   (default) Print after every Scavenge and Old Generation GC
            or Full GC


    :file=[stdout|stderr|<filename>] specifies output file
        stderr    (default) directs output to standard error stream
        stdout     directs output to standard output stream
        <filename> file to which the output will be written

At every garbage collection, the following 20 fields are printed:
<GC: %1 %2 %3 %4 %5 %6 %7 %8 %9 %10 %11 %12 %13 %14 %15 %16 %17 %18 %19 %20>

 %1:  Indicates the type of the garbage collection.
       1: represents a Scavenge (GC of New Generation only)
          %2: indicates if this is a parallel scavenge.
               0: non-parallel scavenge
               n(>0): parallel scavenge, n represents the number of
                      parallel GC threads

       2: represents an Old Generation GC or a Full GC
          %2: indicates the GC reason:
               1: Allocation failure
               2: Call to System.gc
               3: Tenured Generation full
               4: Permanent Generation full
               5: Train Generation full
               6: Concurrent-Mark-Sweep (CMS) Generation full
               7: Old generation expanded on last scavenge
               8: Old generation too full to scavenge
               9: FullGCAlot
              10: Allocation profiler triggered
              11: Last ditch collection
                  If the heap area holding the reflection objects (representing classes
and methods) is
                  full, VM first invokes permanent generation collection. If that fails,
then it tries to
                  expand permanent generation.
       If that also fails, it invokes last ditch collection, to reclaim as much space as
possible.

              12: Heap dump triggered
              13: gcLocker triggered
              14: No cause specified

              (Number 11: "Last ditch collection" is added since 1.4.2.03. Number 12, 13
and 14 are added since 1.4.2.10).

       3: represents a complete background CMS GC
          %2:  indicates the GC reason:
               1: Occupancy > initiatingOccupancy
               2: Expanded recently
               3: Incremental collection will fail
               4: Linear allocation will fail
```

```
            5: Anticipated promotion

      4: represents an incomplete background CMS GC
           (exited after yielding to foreground GC)
        %2:  n.m
             n indicates the GC reason:
               1: Occupancy > initiatingOccupancy
               2: Expanded recently
               3: Incremental collection will fail
               4: Linear allocation will fail
               5: Anticipated promotion
             m indicates the background CMS state when yielding:
               0: Resetting
               1: Idling
               2: InitialMarking
               3: Marking
               4: FinalMarking
               5: Precleaning
               6: Sweeping

%3:  Program time at the beginning of the collection, in seconds

%4:  Garbage collection invocation. Counts of background CMS GCs
     and other GCs are maintained separately

%5:  Size of the object allocation request that forced the GC,
     in bytes

%6:  Tenuring threshold - determines how long the new born object
     remains in the New Generation

The report includes the size of each space:
   Occupied before garbage collection (Before)
   Occupied after garbage collection (After)
   Current capacity (Capacity)
All values are in bytes

Eden Sub-space (within the New Generation)
  %7:  Before
  %8:  After
  %9:  Capacity

Survivor Sub-space (within the New Generation)
  %10:  Before
  %11:  After
  %12:  Capacity

Old Generation
  %13:  Before
  %14:  After
  %15:  Capacity

Permanent Generation (Storage of Reflective Objects)
  %16:  Before
  %17:  After
  %18:  Capacity

  %19:  The total stop-the-world duration, in seconds.
```

%20: The total time used in collection, in seconds.

Java 1.4.0 –Xverbosegc:help 내용

:help prints this message.

0|1 controls the printing of heap information:
0 Print only after each full GC
1 (default) Print after every Scavenge and Full GC

:file=[stdout|stderr|<filename>] specifies output file
stderr (default) directs output to standard error stream
stdout directs output to standard output stream
<filename> file to which the output will be written

At every garbage collection, the following 18 fields are printed:
<GC: %1 %2 %3 %4 %5 %6 %7 %8 %9 %10 %11 %12 %13 %14 %15 %16 %17 %18 >

1: Indicates the cause of the garbage collection.
-1: indicates a scavenge (during a scavenge only objects from the New space are collected)
0-8: indicates a full garbage collection (during a full garbage collection objects from all areas are collected)

The code indicates the reason for the full garbage collection as follows:

Reason:

0: Allocation failure

1: Call to System.gc()
The call was made by the application.

2: Old Generation full
An object was to be allocated in the Old Generation, but there was no room there.

3: Permanent Generation full

The heap area holding the reflection objects (representing Java classes and methods) was full.

4: Train Generation full
Reserved for Java Virtual Machine developers

5: Concurrent-Mark-Sweep Generation full

6: Old generation expanded on last scavenge
Due to implementation reasons, if the Old Generation expanded on last scavenge, no more scavenges can be performed reliably before the next full garbage collection.

7: Old generation too full to scavenge
An object was to be allocated in the New Generation, but there was no room there. However, the VM has determined that the Old Generation was likely too full for a scavenge to compete without expanding the Old Generation. Therefore a full garbage collection
was performed rather than a scavenge.

8: FullGCAlot
Reserved for Java Virtual Machine developers

%2: The time elapsed between the Java program start and the start of this garbage collection event.

%3: Garbage collection invocation. This is the sequential number (count) of the garbage collection event. Counts of Scavenge and Full GCs are maintained separately.

%4: Size of the object allocation request that forced the GC, in bytes.

%5: Tenuring threshold - determines how long the new born object remains in the New Generation.

The report includes the size of each space:
Occupied before garbage collection (Before)

```
        Occupied after garbage collection (After)
        Current capacity (Capacity)

All are in bytes.

Eden sub-space (within the New Generation)
%6:  Before
%7:  After
%8:  Capacity

Survivor sub-space (within the New Generation)
%9:  Before
%10: After
%11: Capacity

Old Generation
%12: Before
%13: After
%14: Capacity

Permanent Generation (Storage of Reflective Objects)
%15: Before
%16: After
%17: Capacity

%18: Duration of the garbage collection in seconds.
```

Java 1.2, 1.3 –Xverbosegc:help 내용

```
0|1 controls the printing of heap information:

0 Print only after each full GC
1 (default) Print after every Scavenge and Full GC
:file=[stdout|stderr|<filename>] specifies output file

stderr (default) directs output to standard error stream
stdout directs output to standard output stream
<filename> file to which the output will be written
At every garbage collection, the following 18 fields are printed:
<GC: %1 %2 %3 %4 %5 %6 %7 %8 %9 %10 %11 %12 %13 %14 %15 %16 %17 %18 >

1: Indicates the cause of the garbage collection.
    -1: indicates a scavenge (during a scavenge only objects from
        the New space are collected)
```

```
    0-6: indicates a full garbage collection (during a full garbage
         collection objects from all areas are collected)

The code indicates the reason for the full garbage collection as follows:

    Reason:
    0:  Call to System.gc()
        The call was made by the application.

    1:  Old Generation full
        An object was to be allocated in the Old Generation, but there
        was no room there.

    2:  Permanent Generation full
        The heap area holding the reflection objects (representing Java
        classes and methods) was full.

    3:  Train Generation full
        Reserved for Java Virtual Machine developers

    4:  Old generation expanded on last scavenge
        Due to implementation reasons, if the Old Generation expanded
        on last scavenge, no more scavenges can be performed reliably
        before the next full garbage collection.

    5:  Old generation too full to scavenge
        An object was to be allocated in the New Generation, but there
        was no room there.  However, the VM has determined that the Old
        Generation was likely too full for a scavenge to compete without
        expanding the Old Generation.  Therefore a full garbage collection
        was performed rather than a scavenge.

    6:  FullGCAlot
        Reserved for Java Virtual Machine developers.

    7:  Last ditch collection
        If the heap area holding the reflection objects (representing
        classes and methods) is full, VM first invokes permanent generation
        collection. If that fails, then it tries to expand permanent generation.
        If that also fails, it invokes last ditch collection, to reclaim as much
        space as possible.

  (From 1.3.1.14 to 1.3.1.17, GC cause code number 6 represents Last ditch collection and
GC cause code
  number 7 represents FullGCAlot)

%2: The time elapsed between the Java program start and the start of this
    garbage collection event.

%3: Garbage collection invocation. This is the sequential number (count)
    of the garbage collection event. Counts of Scavenge and
    Full GCs are maintained separately.

%4: Size of the object allocation request that forced the GC, in bytes.

%5: Tenuring threshold - determines how long the new born object
    remains in the New Generation.
```

```
    The report includes the size of each space:
      Occupied before garbage collection (Before)
      Occupied after garbage collection (After)
      Current capacity (Capacity)

All are in bytes.

Eden sub-space (within the New Generation)
%6:  Before
%7:  After
%8:  Capacity

Survivor sub-space (within the New Generation)
%9:  Before
%10: After
%11: Capacity

Old Generation
%12: Before
%13: After
%14: Capacity

Permanent Generation (Storage of Reflective Objects)
%15: Before
%16: After
%17: Capacity

%18: Duration of the garbage collection in seconds.
```

HP-UX 상에서의 garbage collection 분석툴로는 HPjtune 이 있다.
참고 사이트 : http:/www.hp.com/go/java

-XX:+DisableExplicitGC

System.gc() 에 의한 명시적인 garbage collection 을 무효화 시킨다.

-Xnoclassgc

class 의 garbage collection 금지 옵션.

-Xoptgc

optimistic garbage collection 플래그. short-lived objects 를 사용하는 응용프로그램에서 garbage collection 성능을 증대시킨다.
사용상 주의가 요구되며, short-lived object를 많이 생성하지 않는 경우에는 권장하지 않는다.

기타 옵션

-Xshare:on, -Xshare:off, -Xshared:auto

UserSharedSpaces 는 HP-UX 상에서는 지원되지 않는다.

-Xss<size>

(excerpt below from http://java.sun.com/j2se/1.3/docs/tooldocs/solaris/java.html#options)
Java thread 는 Java code 와 C code 를 위한 두 개의 stack 을 갖는다.

native stack size 의 최대값 결정(하나의 thread 내의 C code가 사용하게 되는 stack size), 단위 bytes.
sizes 는 반드시 1000 byte 보다 커야 한다.
기본 값: -Xss512k (Java 1.3, 1.4, and 5.0 32-bit mode)
-Xss1m (Java 1.4 and 5.0 64-bit mode)

HeapDump 관련 option

-XX:+HeapDump
SDK 1.4.2.10, JDK 1.5.0.03 이상부터 사용 가능.
해당 프로세스에 SIGQUIT 시그널을 보낼 때 마다 JVM 이 Java Heap 의 dump 를 생성하게 된다.
설정방법은 JEUSMain.xml <command-option> 에 –XX:+HeapDump 또는 환경변수에 _JAVA_HEAPDUMP=1 로 설정할 수 있다.
생성되는 파일의 포맷은 hprof 포맷이며, 생성되는 파일명의 포맷을 아래와 같다.
java_<pid>_<date>_<time>_heapDump.hprof.txt

-XX:+HeapDumpOnly 와 _JAVA_HEAPDUMP_ONLY
SDK 1.4.2.11 이상부터 사용 가능.
설정방법은 JEUSMain.xml <command-option> 에 –XX:+HeapDumpOnly 또는 환경변수에 _JAVA_HEAPDUMP_ONLY 로 설정할 수 있다.
SIGVTALARM signal(signal 20)을 이용하여 HP Heap Dump 할 수 있다.
생성되는 파일명의 포맷을 아래와 같다.
java_<pid>_<date>_<time>_heapDump.hprof.txt
생성되는 출력의 기본 포맷은 ASCII 이며, hprof binary 포맷으로 변경키 위해서는 환경변수에 _JAVA_BINARY_HEAPDUMP 환경 변수를 지정하면 된다.
기본적으로 –XX:+HeapDump 와 –XX:+HeapDumpOnly 의 dump 정보는 ASCII 포맷이다.

-XX:+HeapDumpOnCtrlBreak
SDK 1.4.2.11, JDK 5.0.05 이상부터 사용 가능.
JEUSMain.xml <command-option> 에 –XX:+HeapDumpOnCtrlBreak 로 설정 가능.
-XX:+HeapDump 와 비슷하나 binary 포맷이다.
생성되는 파일명의 포맷을 아래와 같다.
java_<pid>_heapDump.hprof.<millitime>

-XX:+HeapDumpOnOutOfMemoryError

SDK 1.4.2.11, JDK 5.0.04 이상부터 사용 가능하며, low-pause collector(-XX:+UseConcMarkSweepGC)에서는 작동하지 않는다.

Out Of Memory error 발생시 Heap dump 발생하며, hprof binary 포맷으로서 working directory 상에 java_pid<pid>.hprof 의 파일명으로 작성된다.

-XX:HeapDumpPath=<file> 와 같이 사용하여 디렉토리 또는 파일명을 결정할 수 있다.

환경변수와 **HeapDump option** 의 조합

format	_JAVA_BINARY_HEAPDUMP	_JAVA_HEAPDUMP_ONLY	_JAVA_HEAPDUMP	<command-option>	trigger
binary ascii			○	-XX:+HeapDumpOnCtrl	SIGQUIT
binary	○		×	-XX:+HeapDumpOnCtrl	SIGQUIT
binary ascii	○			-Xrunhprof:heap=dump	
binary	○				SIGVTALRM
binary	○	○			SIGVTALRM

4. 분석 기술

-Xverbosegc 로그 출력과 이해 (Java 5.0)

-Xverbosegc 옵션을 통해 Java Heap 의 garbage collection 전, 후의 자세한 내용을 출력 할 수 있다.

-Xverbosege:help 을 통해 출력 내용의 포맷을 확인 할 수 있다.

출력되는 결과는 아래와 같은 형태를 취하고 있다(Java 5.0, 20 개 field).

<GC: 2 1 772.061586 4 1313992 1 382231568 1313992 382926848 1345536 0 65536 145011680
33541424 149946368 61176016 59468800 88080384 0.754580 0.754580 >

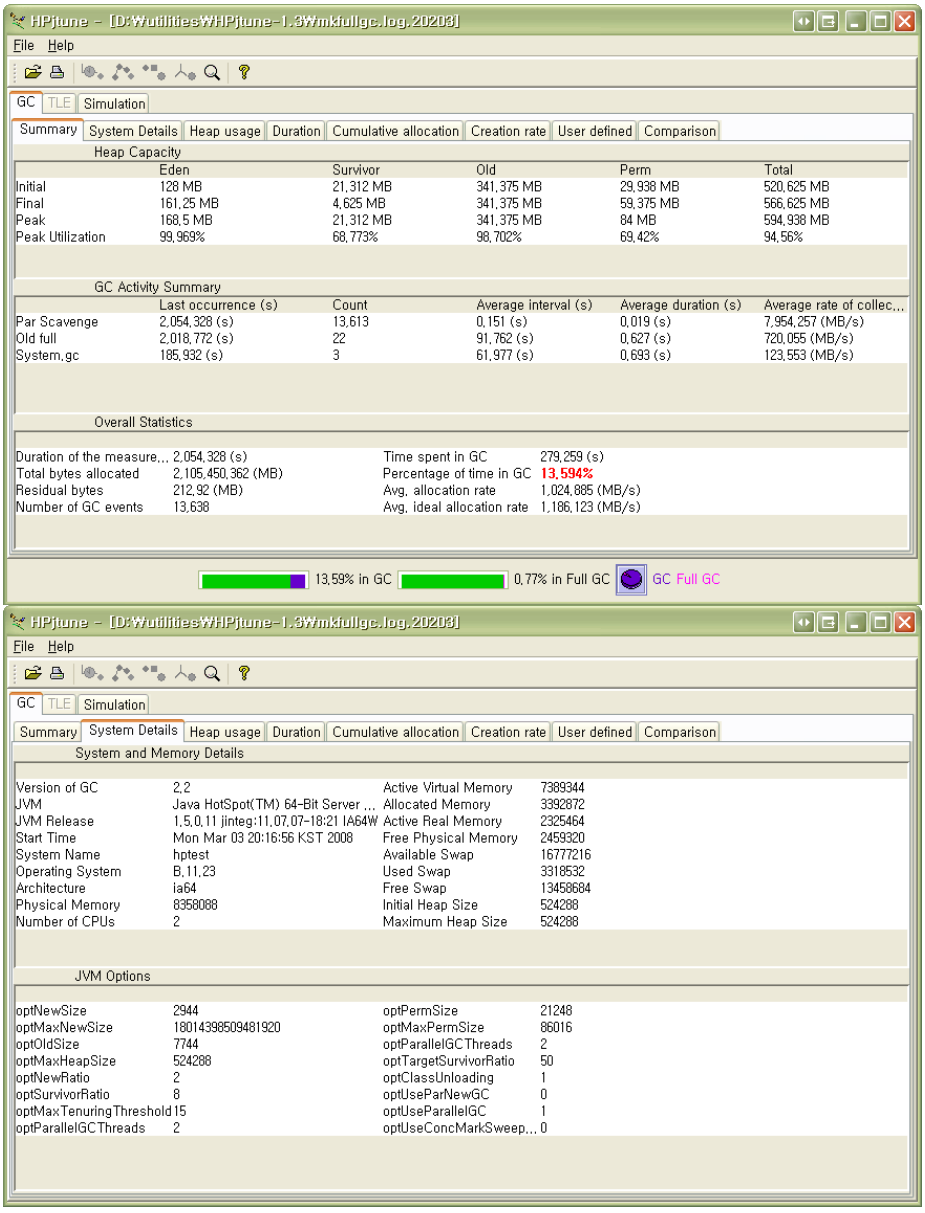
garbage collection 의 종류로 2일 경우에는 Full GC, 1일 경우에는 Scavenge를 나타낸다.
GC 가 발생한 사유로 1은 scavenge 실패에 따른 할당 실패로서, Full GC 를 수반한다.
collection 시작 시점의 프로그램 시간, 단위 초
GC 종류별 garbage collection count. 예에서는 Full GC의 4번째 수행.
GC 를 발생케 한 객체 할당 사이즈, 단위 byte.
Tenuring threshold
Eden 의 garbage collection 전 사이즈, 단위 byte.
Eden 의 garbage collection 후 사이즈, 단위 byte.
Eden 의 현재 용량, 단위 byte.
Survivor 의 garbage collection 전 사이즈, 단위 byte.
Survivor 의 garbage collection 후 사이즈, 단위 byte.
Survivor 의 현재 용량, 단위 byte.
Old generation 의 garbage collection 전 사이즈, 단위 byte.
Old generation 의 garbage collection 후 사이즈, 단위 byte.
Old generation 의 현재 용량, 단위 byte.
Permanent generation 의 garbage collection 전 사이즈, 단위 byte.
Permanent generation 의 garbage collection 후 사이즈, 단위 byte.
Permanent generation 의 현재 용량, 단위 byte.
stop-the-world 기간, 단위 second.
collection 수행 시간, 단위 second.

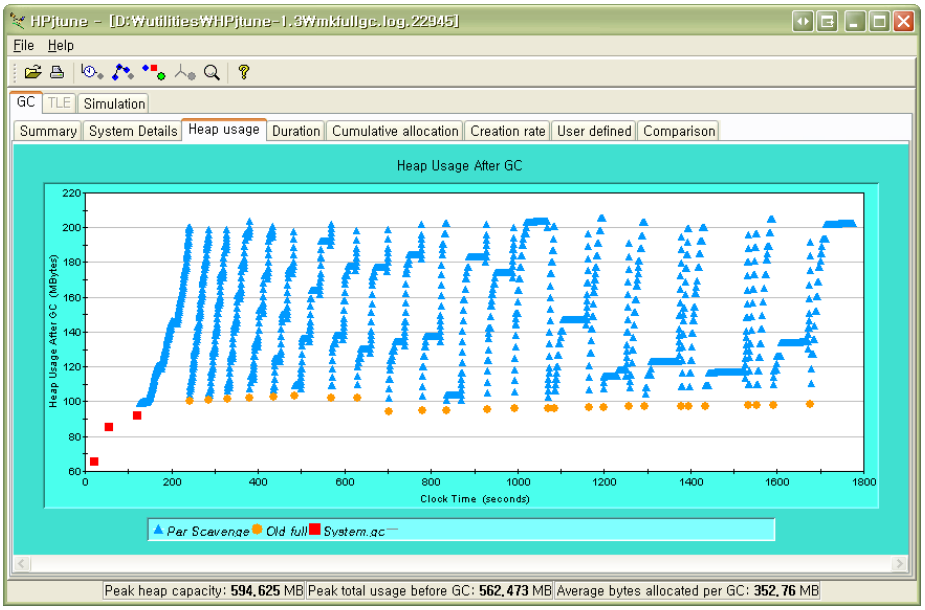
상기의 로그 포맷은 HPjtune 을 이용하여 graphical 하게 분석할 수 있다.

HPjtune 을 이용한 **–Xverbosegc** 분석

1. 다운로드 : http://www.hp.com/products1/unix/java/java2/hpjtune/index.html

2. 실행 화면





3. 설명

실제 –Xverbosegc 의 로그를 분석한 화면으로써, 전체 기간동안의 Java Heap 의 현황과 사용된 JVM option 및 Java Heap 의 Garbage Collection 수행 내역의 정보에 쉽게 접근이 가능하다.

Out Of Memory Error 을 발생시킬 수 있는 몇 가지 사유

Virtual address space 의 **size** 제약
적절치 못한 **Java Heap** 사이즈 설정
낮은 값의 **kernel parameter** 설정
max_thread_proc : 프로세스 당 thread 개수
nkthread : 총 thread 개수
maxdsiz : Data region size
nfiles : open file 개수
maxfiles : 프로세스 당 open file 의 soft limit
maxfiles_lim : 프로세스 당 open file 의 hard limit

Trademarks
Tmax, WebtoB, WebT, JEUS, ProFrame, SysMaster and OpenFrame are registered trademarks of TmaxSoft Co., Ltd. Other products, titles or services may be registered trademarks of their respective companies.

Contact Information
TmaxSoft can be contacted at the following addresses to arrange for a consulting team to visit your company and discuss your options for legacy modernization.

Korea – TmaxSoft Co., Ltd.
Corporate Headquarters
272-6 Seohyeon-dong, Bundang-gu, Seongnam-si, South Korea, 463-824
Tel : (+82) 31-8018-1708 Fax : (+82) 31-8018-1710
Website : http://tmaxsoft.com

U.S.A. – TmaxSoft Inc.
560 Sylvan Avenue Englewood Cliffs, NJ 07632, USA
Tel : (+1) 201-567-8266 Fax : (+1) 201-567-7339
Website : http://us.tmaxsoft.com

Japan – TmaxSoft Japan Co., Ltd.
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073 Japan
Tel : (+81) 3-5765-2550 Fax: (+81) 3-5765-2567
Website : http://jp.tmaxsoft.com

China – TmaxSoft China Co., Ltd.
Room 1101, Building B, Recreo International Center, East Road Wang Jing, Chaoyang District, Beijing, 100102, P.R.C
Tel : (+86) 10-5783-9188 Fax: (+86) 10-5783-9188(#800)
Website : http://cn.tmaxsoft.com

China(JV) – Upright(Beijing) Software Technology Co., Ltd
Room 1102, Building B, Recreo International Center, East Road Wang Jing, Chaoyang District, Beijing, 100102, P.R.C
Tel : (+86) 10-5783-9188 Fax: (+86) 10-5783-9188(#800)
Website : www.uprightsoft.com

TD-JSTN-C1101017