JBoss Performance Tunning

byJ

목 차

1. Throughput 향상
2. 빠른 응답시간
3. 응답시간의 보장
4. 시스템 성능의 측정

2. JBoss AS 의 성능 Factors

- Application
- 웹 레이어
- EJB 레이어
- Database
- 보안
- Logging
- 클러스터링
- JVM
- Profiling

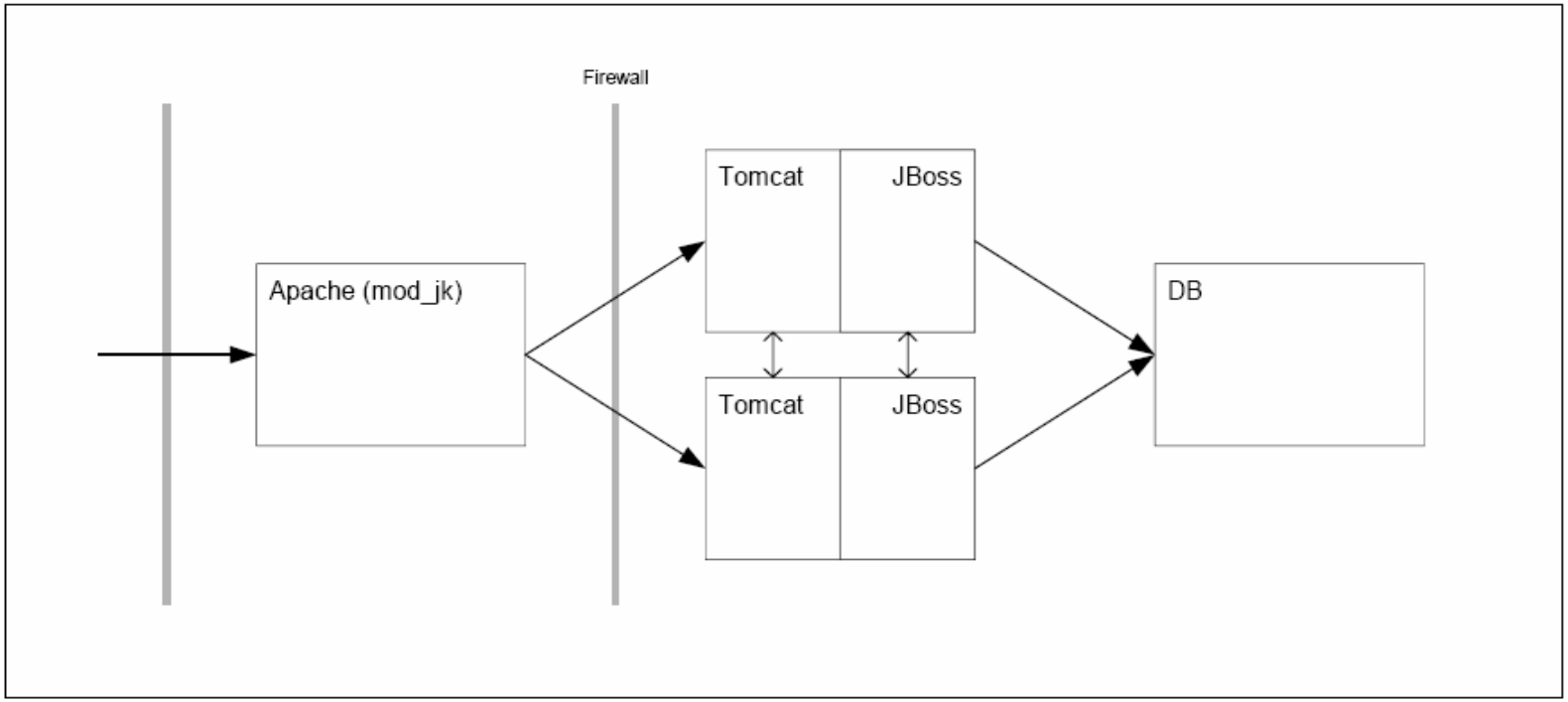
2. JBoss AS 의 성능 Factors

- **Application 은 다른 성능 factor 이상으로 성능을 결정한다.**
- **App Server 나 하드웨어를 튜닝한다고 해도 Application 디자인에서 문제가 있다면 성능이 나올 수 없다.**

- **Tips:**
  - 당신의 application 을 최대한 효과적으로 튜닝하는 것이 제일 좋은 방법이다.
  - 테스트와 최적화 작업을 초기부터 고려하여 같이 진행하는 것이 좋다.

❑ Apache 와 Tomcat

- 로드 밸러싱을 위해 Apache를 이용하여라.(mod_jk)
- mod_jk2를 사용하지 않도록 한다.
- 나중에는 mod_proxy 를 사용하는 것이 좋다.



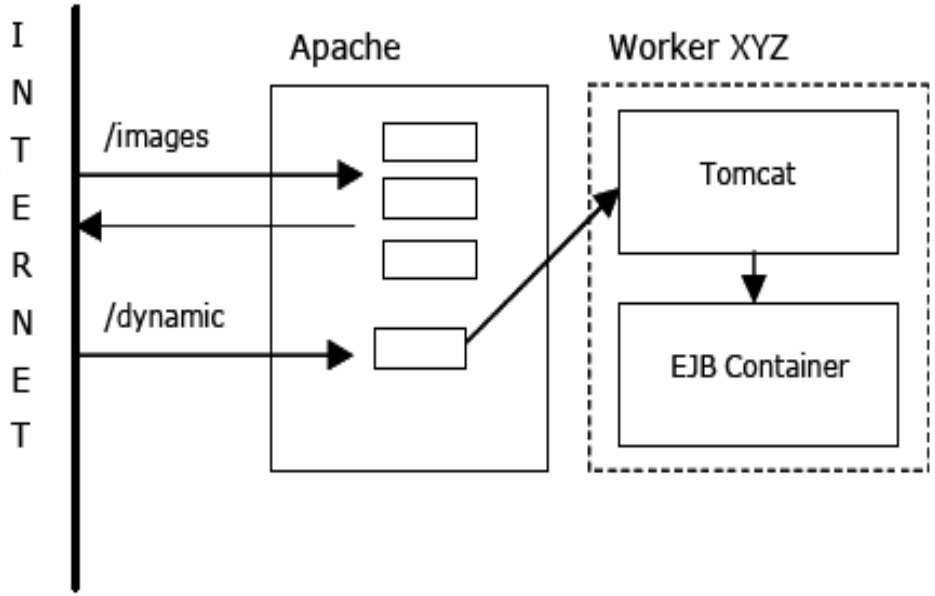
❑ Apache

mod_jk 의 configuration을 확인하라

- mod_jk 문서를 활용하여라
- jboss.com의 Webinar를 활용하여라

– Apache Httpd server를 이용하여 static content 서비스를 하여라

- load를 줄여준다.



– cache 시간을 조절하는 mod_expire를 이용하여라.

- 브라우져와 proxy 서버에 있는 caching 정책을 조절하는 유일한 방법이다.
- 로드를 줄임으로써 엔드유저에게는 성능 향상을 느끼게 해준다.
- 파일의 타입에 따라 caching 정책을 따로 하여 관리한다.

❑ Tomcat

- 일반적으로
  - logging 을 최소화하거나 끈다.
  - Connectors(http+ajp)에 사용되는 쓰레드의 최소/최대 개수를 튜닝한다.
  - AJP Connector를 이용하는 경우 http Connector는 사용하지 않는다.
- JSP 의 최적화
  - Development 모드를 끈다.(각 JSP 의 새버젼을 체크하기때문에)
  - JSPs 를 선 컴파일 해둔다.
  - JSPs 의 debug 정보를 사용하지 않도록 한다.(default:yes, classdebuginfo)
- Tomcat 4.x 보단 5.x 버전을 디볼트로 사용하도록 한다.

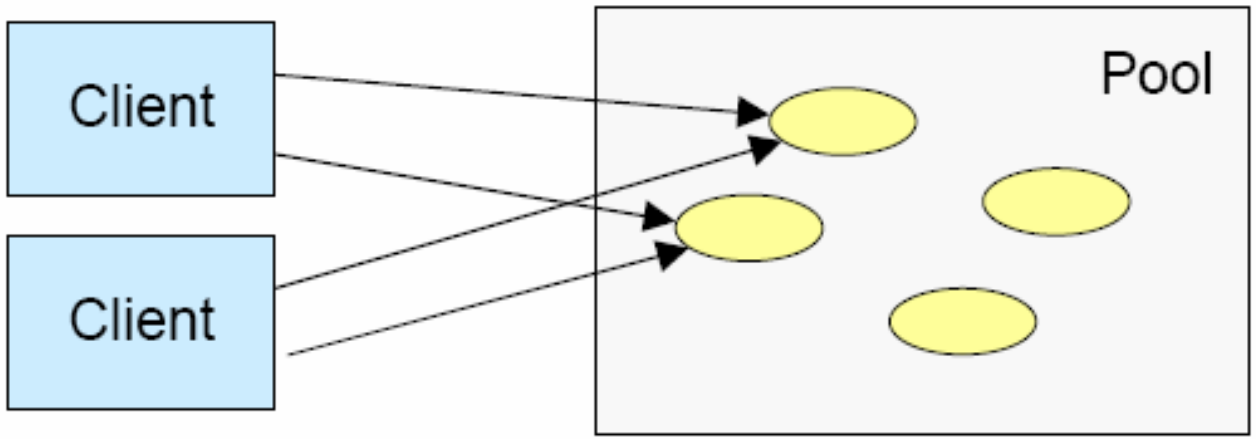
❑ Pooling

Object가 사용되기전에 먼저 생성된 Object 그룹을 제공한다.

– Object의 생성과 초기화에 들어가는 시간을 절약할 수 있다.

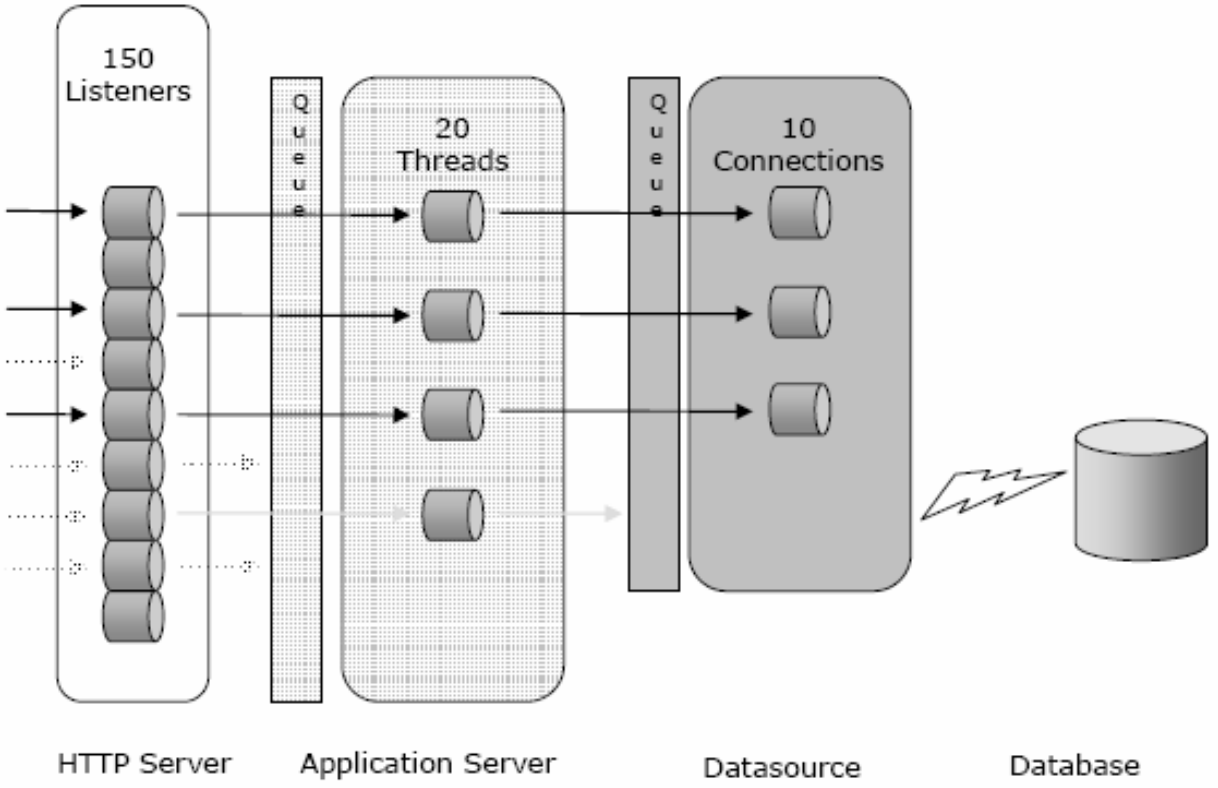
예를 들자면

- http 리스너
- datasources
- EJBs

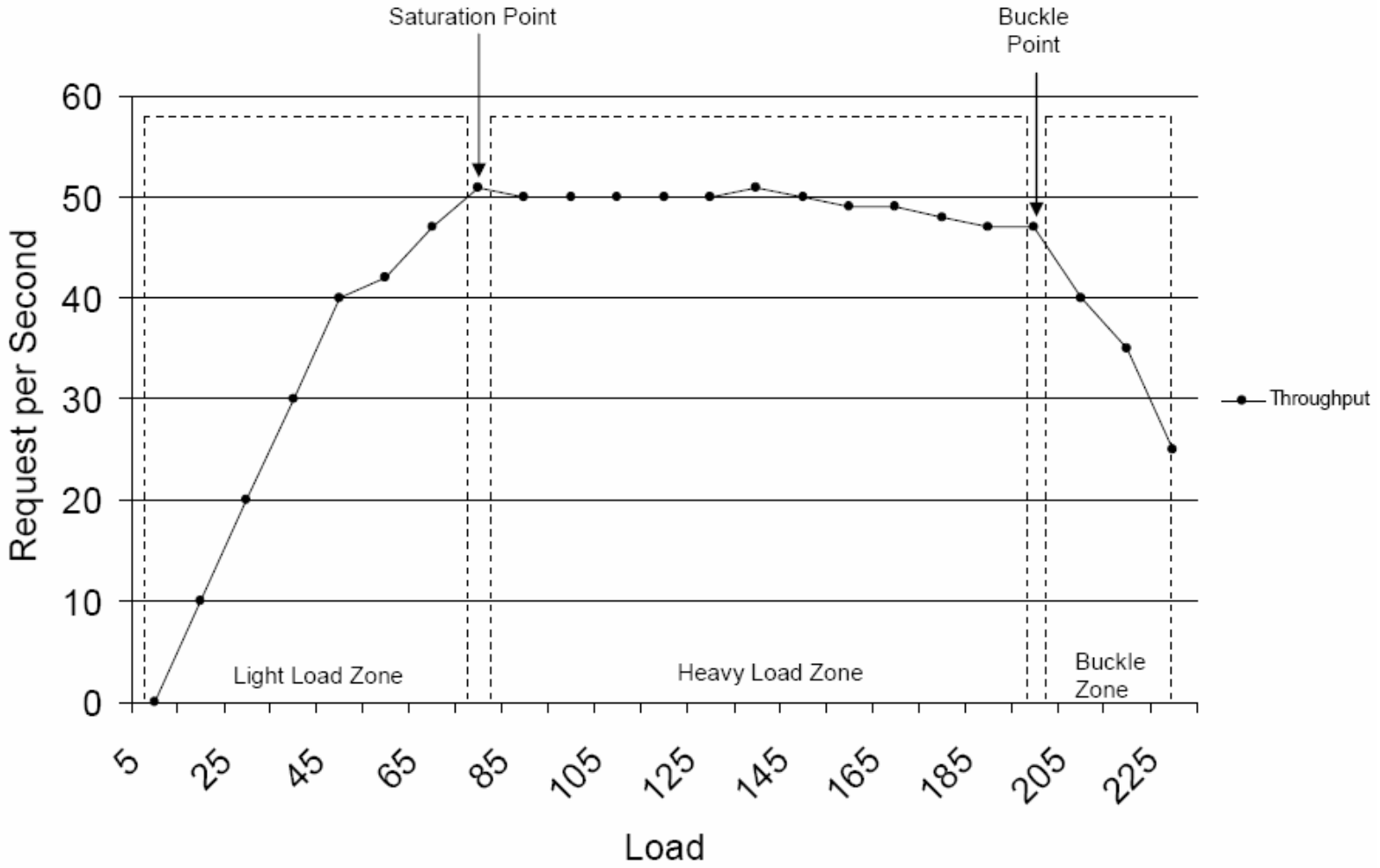


Pooling and Queuing

- Pools have a maximum size
- Pools are a protection
- Pools are a source of queuing, max be a bottleneck
- Need to be tuned
- Very application specific



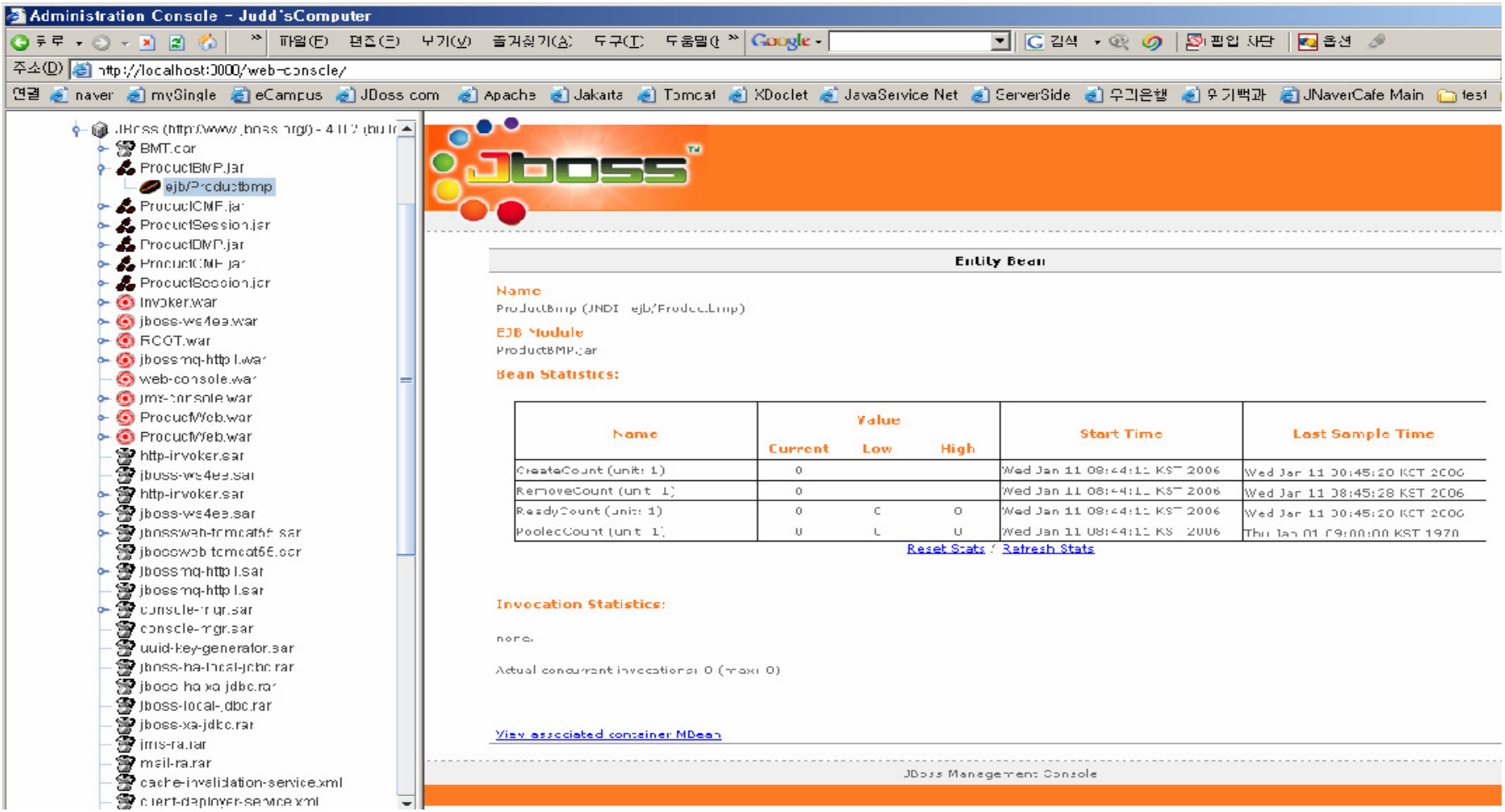
Typical Load Profile



2. JBoss AS 의 성능 Factors

❑ Bean invocation 통계정보

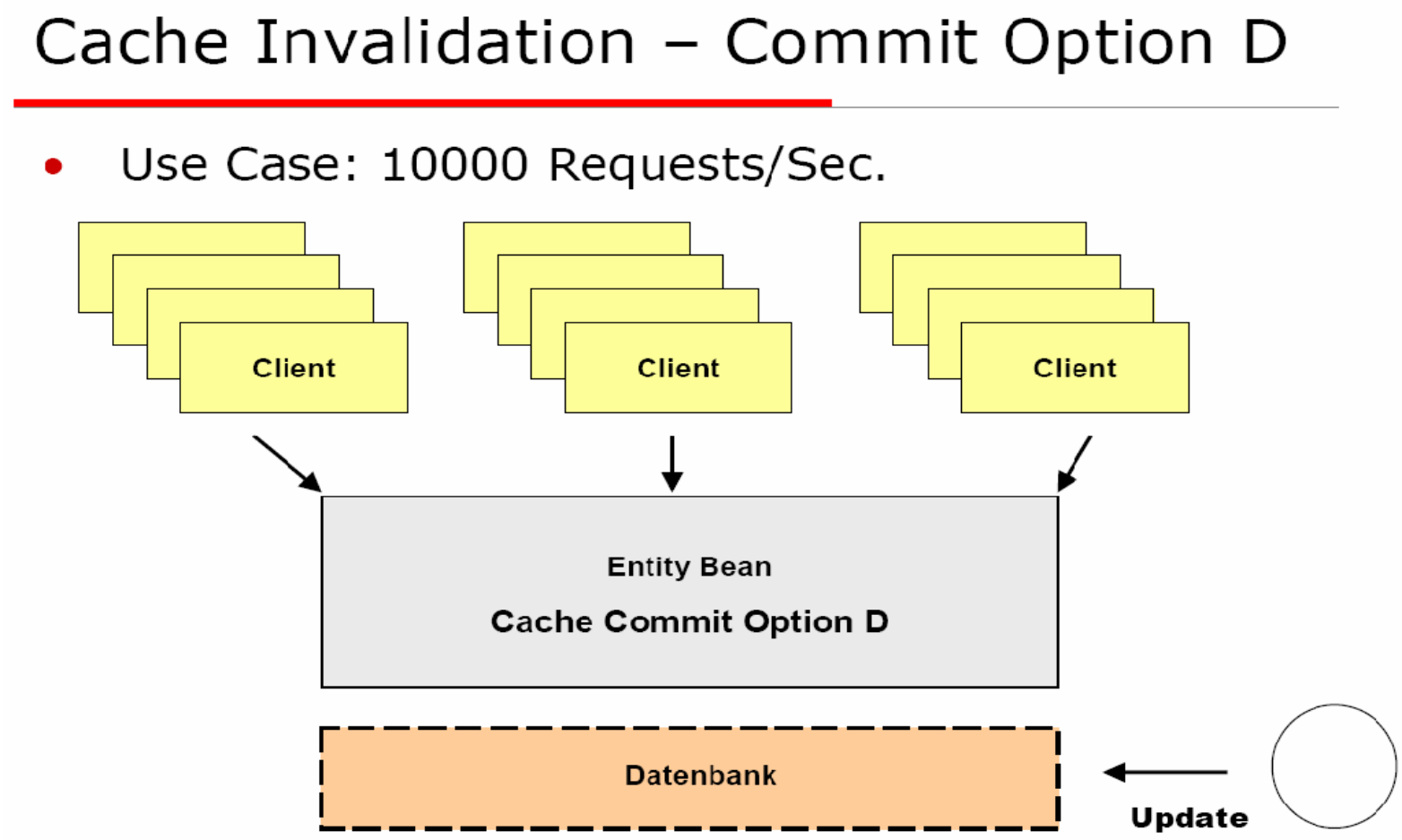
– http://localhost:8080/web-console



❑ Caching

- 가능한 부분에 모두 캐싱하도록 한다. (Cache 하는 것이 가장 좋다.)
- Database의 접근하거나 serialization 하는 것을 줄일 수 있다.
- CMP Caching 은 Commit Options (A/B/C/D) 에 따라 정책을 적용할 수 있다.
  - db와의 데이터 싱크 타이밍을 맞추어주는 역할을 한다.
- Caching 은 성능과 캐싱된 데이터 검증사이에 있다. 따라서 데이터 오프젝트의 검증은 필요하다.
- 클러스터/멀티노드 모드의 Caching은 반드시 cache 된 데이터 체크가 있어야 한다.
  - 그렇다고 caching 모드를 끄는 것은 좋지 않다.
  - CMP 모드를 가진 프레임워크에서는 caching 은 유효하지 않다.
  - HIbernate / JBossCache를 이용하라.

2. JBoss AS 의 성능 Factors



- **Cache는 30초마다 refresh 된다.**

❑ CMP 튜닝

- Caching
- 로드 분산 전략
  - 그룹을 나누어 로드를 준다.
  - DB의 데이터 읽는 타이밍의 전략(싱크)
    1. Find 시점
    2. Load 시점
- CMT 튜닝을 위해서는 JBoss 문서와 튜닝가이드를 참조해서 튜닝하면 된다.

❑ CMP 튜닝

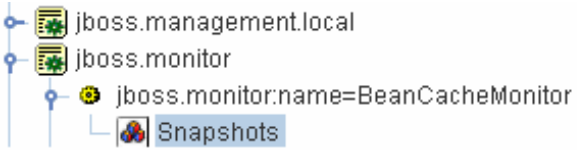
– CMP Cache 사용

- CachaMonitor 를 반드시 활성화 하여야 한다.

jboss-service.xml

```
<mbean code="org.jboss.monitor.BeanCacheMonitor"
       name="jboss.monitor:name=BeanCacheMonitor"/>
```

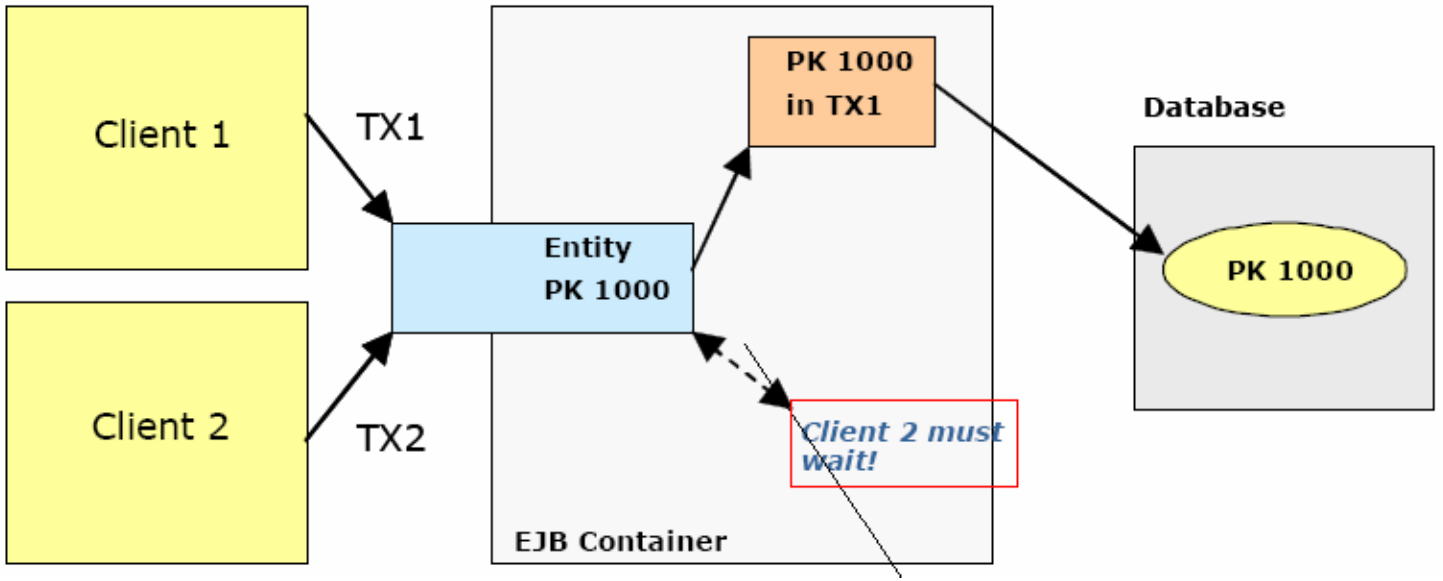
- Cache 사용률(사이즈)를 체크 하여라




```
[Cache Snapshot for application 'null', container for bean 'Counter':
min capacity: 50
max capacity: 1000000
capacity: 1000000
size: 0
number of beans scheduled for passivation: 0, Cache Snapshot for application 'null', container
min capacity: 50
max capacity: 1000000
capacity: 1000000
size: 1
```

❑ 트랜젝션과 동시 사용

- 트렌젝션은 데이터 무결성을 보장되어야 한다.
- 같은 EJB 인스턴스에 멀티 트렌젝션이 발생하는 경우 트랜젝션의 락킹이 발생한다.



- 결론적으로 클라이언트1이 해당 트렌젝션이 끝날때까지 클라이언트2가 기다리게 된다.
  - Thread-Locking 발생

❑ Locking Detection

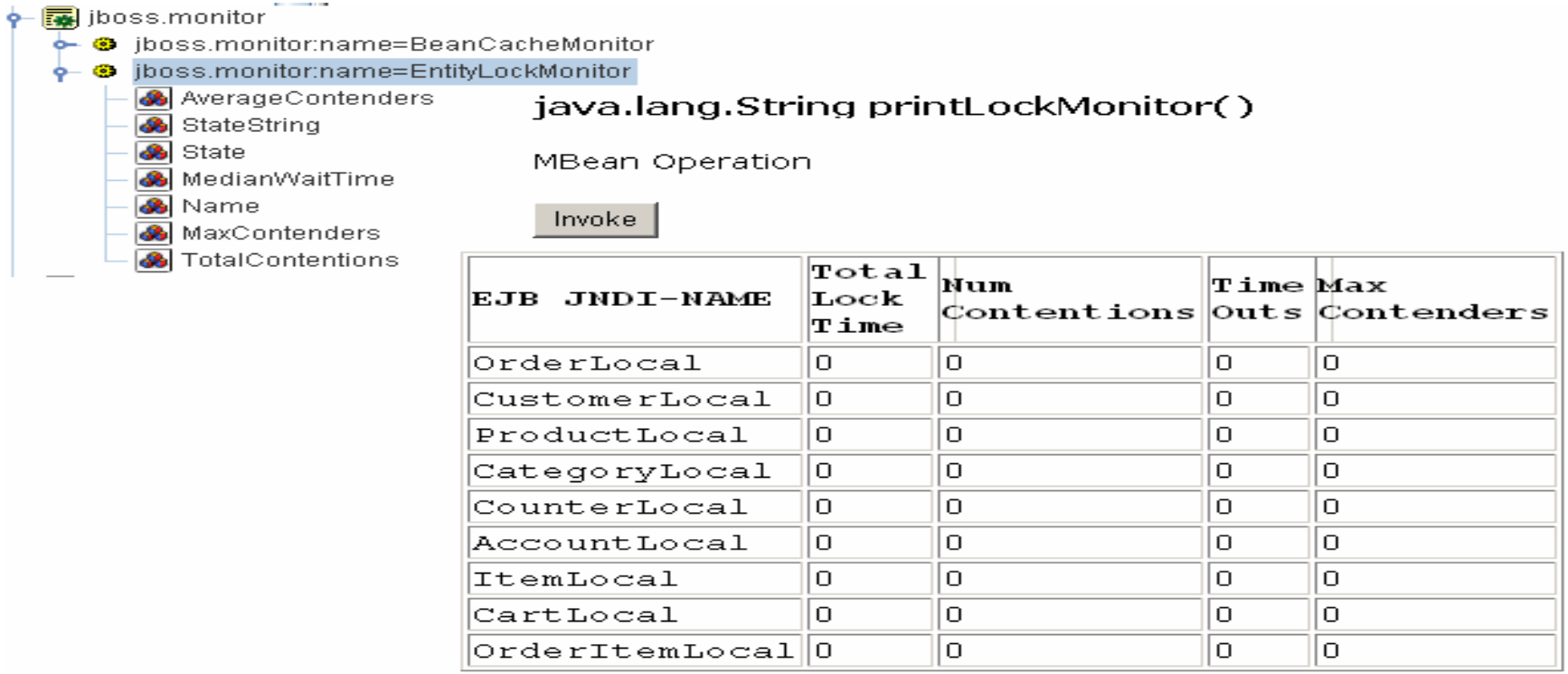
– EntityLockMonitor 는 활성화 되어 있어야 한다.

jboss-service.xml

```
<mbean code="org.jboss.monitor.EntityLockMonitor"
       name="jboss.monitor:name=EntityLockMonitor"/>
```

– Contention -> 다른 트렌젝션이 Locking이 풀리기를 기다리고 있을 경우 발생한다.

• Lock Time



EJB JNDI-NAME	Total Lock Time	Num Contentions	Time Outs	Max Contenders
OrderLocal	0	0	0	0
CustomerLocal	0	0	0	0
ProductLocal	0	0	0	0
CategoryLocal	0	0	0	0
CounterLocal	0	0	0	0
AccountLocal	0	0	0	0
ItemLocal	0	0	0	0
CartLocal	0	0	0	0
OrderItemLocal	0	0	0	0

❑ Locking 최적화

- 항상 트렌젝션 Locking 이 필요한가?
- “read only” 만 하는 메소드나 Bean을 구분한다.
- 되도록이면 트렌젝션의 짧게 하도록 한다.
- Locking 정책을 변경하는 것이 도움이 될 수도 있다.

❑ Locking

- Locking은 데이터의 무결성을 유지 하기위해 있다. standardjboss.xml / jboss.xml 에 있는 각각의 다른 빈별로 다른 locking 정책을 적용이 가능하다.
- Pessimistic Locking 정책 (default)
  - 트랜젝션이 일어나는 동안 관련 리소스는 locking 된다.
  - 동시에 트랜젝션이 접근시 한 개는 기다리게 되고 한 개는 진행하게 된다.(exclusive Lock)
- Optimistic Locking 정책
  - 동시 접근을 허용한다.
  - 최소의 locking 시간을 갖는 정책이다.(database의 synchronization 함으로써 lock 를 관리한다.)
- Locking없는 정책
  - exclusive Lock 을 사용하지 않는다.
  - 멀티 쓰레드/트렌젝션에 의해 동시 접근이 가능하다.
- Read-Write Locking 정책(가장 적절한 정책으로 보인다.)
  - 동시에 여러 개의 Read Locks을 허용한다.
  - 단 Write Lock 시 Read-Locks 이 걸리지 않는다.
- Read-only 전략
  - 데이터의 생성/삭제/수정을 허용하지 않는다.(no ejbStore())
  - 트랜젝션 lock 은 발생하지 않는다.
- 가능하면 Read-only 전략을 사용한다.(공유되는 마스터 데이터의 경우)
- Read-Write Locking 정책을 적용하는 것이 좋다.

2. JBoss AS 의 성능 Factors

❑ Database Connection Pools

```
oracle-ds.xml:

<local-tx-datasource>
<jndi-name>DefaultDS</jndi-name>
<connection-url>jdbc:oracle:thin:@host:1521:sid</connection-url>
<driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
…
<min-pool-size>0</min-pool-size>
<max-pool-size>20</max-pool-size>
…
<valid-connection-checker-class-name>
org.jboss.resource.adapter.jdbc.vendor.OracleValidConnectionChecker
</valid-connection-checker-class-name>
…
<track-statements>true</track-statements>
<prepared-statement-cache-size>100</prepared-statement-cache-size>
```

- 어플리케이션을 위한 poolsize를 튜닝하여라 (database 에서 지원하는 개수를 초과하지 않도록 한다.
- statements 를 트래킹하여라.
  - 테스트시 로그에 남기도록 하고 운영시에는 남기지 않도록 한다.
- prepared-statement 캐시를 이용한다.
  - 성능의 향상을 가져온다. insert 시 60%이상의 성능향상을 가져온다.

❑ JMS 성능 향상

- 확장성을 위해서는 JMS 클러스터링을 이용하라.
  - 클러스터 노드를 여러 개 둠으로써 로드를 분산시킬 수 있다.
  - 버전 3.2.7 이후 에 사용하라.
- 서버의 overload를 피하기 위해서 MDB 인스턴스의 개수를 제안하라
  - 동시에 100개의메세지를 100개의 MDB에서 처리하는 경우 여러가지 문제가 발생한다.
    CPU, Database, Memory, Excution time
  - 합리적인 개수를 정하여 제한을 한다.
- 디볼트로 세팅되어 있는 Hypersonic database 를 이용하지 않도록 한다.

❑ DataBase

- Database 스키마를 체크하라.
- 테이블별 Indexs 를 잘 정비하도록 한다.
- 성능 분석을 위한 Database tools을 이용하라.

❑ Security

– JBoss 에서 제공하는 Security Cache를 이용한다.

- Authentication 과 Authorization 소스에 있어 로드를 줄이는 역할을 한다.
  (LDAP, Database…)
- AuthenticationCache를 이용하는지 확인하라.
- web-console 또는 로그 파일의 MBean을 체크 하여라

❑ Logging

- 개발자를 위한 Application logging 가이드라인을 제시하며 세팅한다.
  - log level 별로 어떤 메시지를 남길지 결정한다.
  - system.out.println()은 쓰지 않도록 한다.
- 불필요한 logging은 끄도록 한다.
  - 예를 들자면, Tomcat의 불필요한 로그를 남기지 않도록 한다.
- load 테스팅과 튜닝하기전에 error 로그가 있는지 확인한다.
- load 테스팅시 남겨지는 로그를 확인하라.
  - 성능적으로 중요한 힌트를 얻을 수 있다.

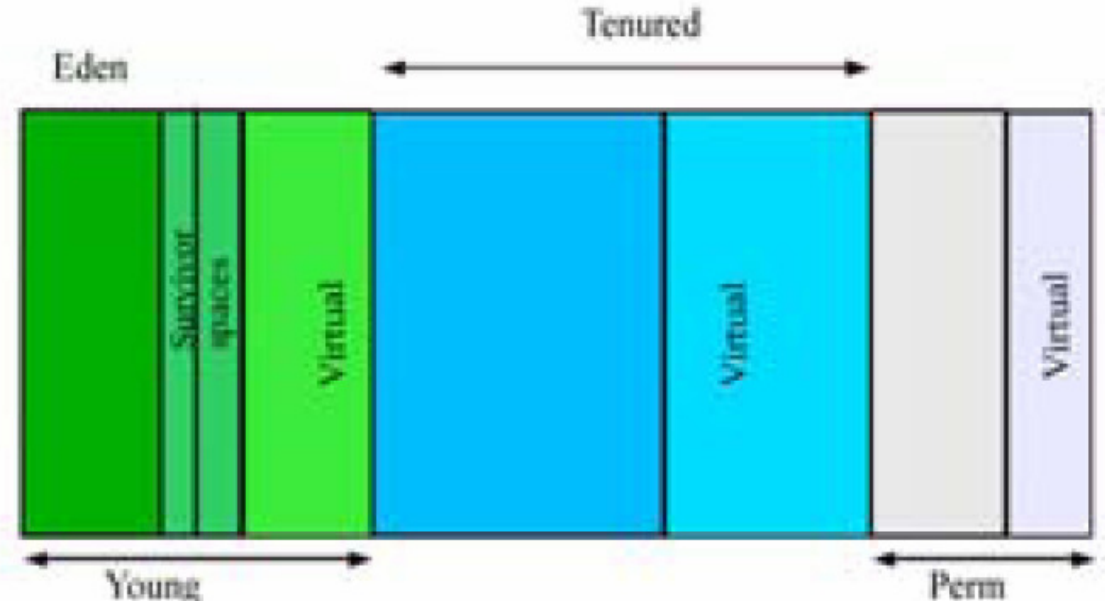
2. JBoss AS 의 성능 Factors

❑ Clustering

- Elements:
  - ✓ Load Balancing
  - ✓ Failover
  - ✓ State Replication
- Aspects:
  - ✓ Scaliability
  - ✓ High Availability
- Tips:
  - ✓ Determine your requirements
  - ✓ Do not cluster if there is no need to
- Be careful:
  - ✓ It is easy to deploy cluster designs that do NOT scale…
  - ✓ Application needs to be checked and prepared for clustering

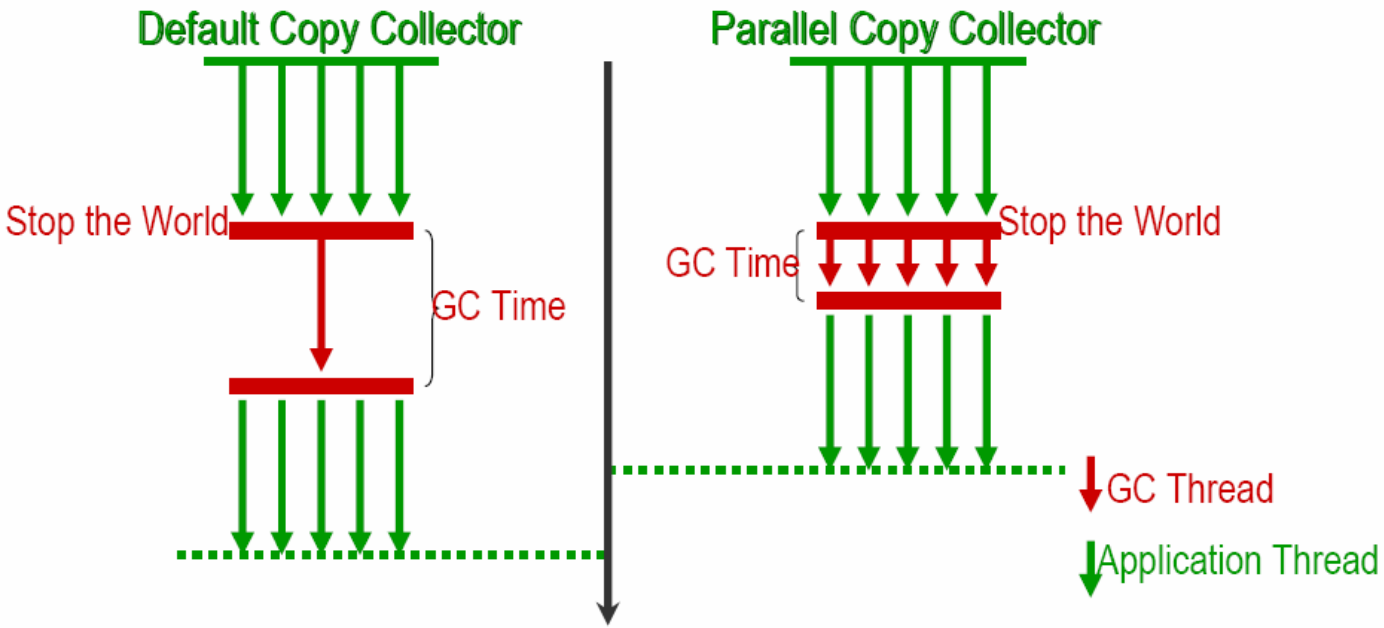
JVM Tuning – Memory Areas

- Separation of memory areas in „young" and „old" generation
  - ✓ Short-living objects in young generation (e.g. local variables)
  - ✓ Long-living objects are moved to old generation
  - ✓ Garbage Collection on a regular basis on young generation (Minor Collection)
  - ✓ Full Garbage Collection in larger intervals (Major Collection)



Parallel Garbage Collection

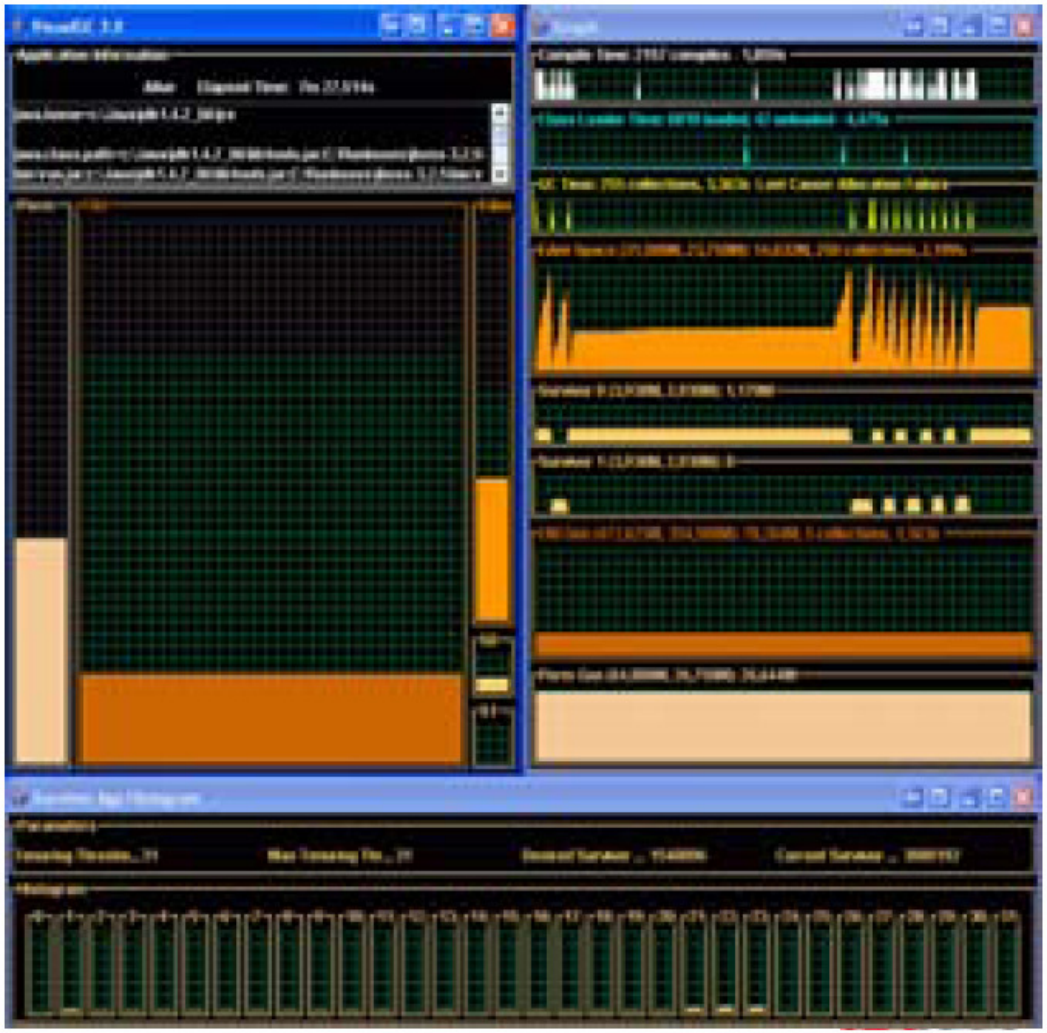
- More Efficiency for Multi-CPU Systems (2+x)
- Threaded Garbage Collection



-XX:+UseParallelGC
-XX:ParallelGCThreads=<n>

Garbage Collection Analysis

- VisualGC, comes with JDK 5, can also monitor JDK 1.4 apps
- Repoting and Analysis of GC activities
- Interpreting output



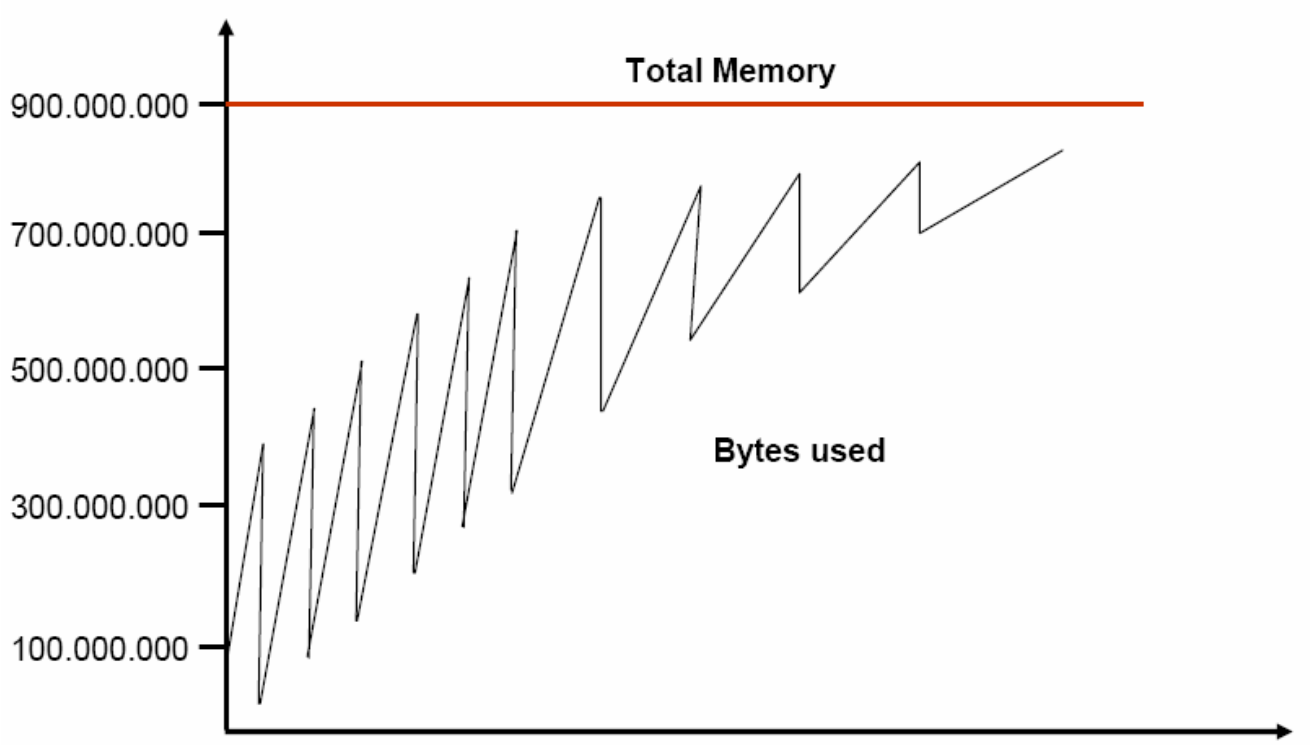
❑ JVM 튜닝 권장사항

- 서버사이드 Application을 위한 디볼트 최대 heap 사이즈가(64MB) 작다
- 멀티 CPU인 경우 Parallel GC 를 이용하여라.
- 로드 테스팅 , 메모리 튜닝시 메모리 Profile을 확인 하라.
- JDK 1.4 보다는 Java 5를 권장한다.
- 가비지컬레션(GC) 최적화는 성능을 향상시킨다.
- 상당히 높은 트렌젝션 시스템에 대한 잠재력에 대해 낮게 평가되고 있다. JVM의 option 활용도를 높여라.

❑ JVM Tunning

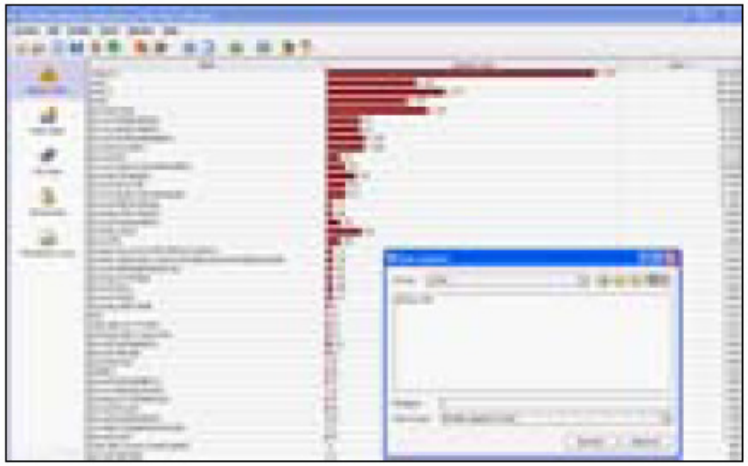
– 메모리 누수 현상 감지하기

- 메모리 관리는 JVM에 의해 관리되지만 개발자에 의해 메모리 누수가 발생할 수 있다.
  - profiling 이 필요하다.



❑ Profiling

- 분석을 통해 좀더 나은 성능을 찾을 수 있다.
- JVM 스냅샷을 비교한다.
- 메모리 누수 현상을 찾을 수 있다.
- 상용 SW : OptimizeIT, JProble, JProfiler
- JBoss Production : JBoss Profiler



3. 로드테스팅 Hints

❑ 로드 테스팅 Hints

- 실제적인 로드테스팅을 위해서는
  - Load VS. Concurrent Load
  - 실질적인 비즈니스 시나리오
  - 시간을 생각하라.
- 테스트중 시스템 모니터링
  - 블랙박스에서 테스트 하지 마라
  - 병목현상이 발생하는 지점을 찾아라.
  - 시스템 구성의 최적화를 하여라.
- 선행 테스트를 위하여 Pools 의 개수를 체크하여라
  - web server pool, DB connection pool, http/ajp Thread pool
  - pool을 사용함으로써 병목현상 해결에 도움이 된다.
  - 시스템의 최적 수치를 찾아라.
- Caching 효과를 이용하라.

4. 튜닝 방법

❑ 튜닝 방법

1. 성능관련한 요구사항을 분석하고 목표를 정한다.
2. 케이스 분석을 한다.
3. Application 분석 : Application 단의 성능저하 요인을 제거한다.
4. 기본적인 환경설정을 체크하고 Pools 을 설정한다.
5. 모니터링을 위한 세팅을 한다.
6. 로드 테스트를 실행한다.
7. 분석 : 병목현상 지점을 찾는다. 로그파일 분석.
8. 최적화 작업 실행 다시 6번(re-run 테스트 실행)
9. 확장성 테스트 실행

5. references 자료

- **JBoss Knowledge Sources**
  - ✓ JBoss Documentation, JBoss Tuning Guide
  - ✓ Wiki
- **JBoss Products to have a look at**
  - ✓ JBoss Cache
  - ✓ Hibernate
  - ✓ JBoss Profiler
  - ✓ JBoss Network Enterprise Manager – Monitoring
- **Links**
  - ✓ Sun VM Garbage Collection Tuning Guide
    - http://java.sun.com/docs/hotspot/
  - ✓ Download this presentation
    - www.jboss.com/events/jbossworld
- **Book**
  - ✓ Performace Analysis for Java Websites (Addison-Wesley)

Java Virtual Machine Tuning

- JVM configuration can also be a key factor for performance
  - ✓ Default values inappropriate for server side J2EE-Applications (esp. Java 1.4)
  - ✓ JVM: abstraction layer between Code and OS
  - ✓ Unreferenced objects get removed by JVM Garbage Collection
  - ✓ Limited influence of the developer, no manual memory allocation and release (like in C / C++)
  - ✓ Garbage Collection is executed in intervals
- Sometimes whole JVM is suspended for garbage collection (→ also your application) for several seconds!
  - ✓ Garabage Collection behaviour can be influenced by JVM parameters
  - ✓ Need to find out own configuration for your specific application

JVM Options – Excerpt

- General Options:
  - ✓ -Xms512m (initial Heap)
  - ✓ -Xmx1024m (max. Heap)
  - ✓ -XX:+DisableExplicitGC (prevent System.gc())
  - ✓ -XX:+UseParNewGC (Multi-CPU-GC!)
  - ✓ -XX:+UseConcMarkSweepGC (Concurrent Collector)
  - ✓ -XX:NewSize=128m (young generation initial)
  - ✓ -XX:MaxNewSize=196m (young generation maximal)
  - ✓ -Dsun.rmi.dgc.server.gcInterval=3600000 (RMI gc. only once an hour)
  - ✓ -XX:+AggressiveHeap

- GC-Output:
  - ✓ -verbosegc
  - ✓ -XX:+PrintGCDetails
  - ✓ -XX:+PrintGCApplicationStoppedTime
  - ✓ -XX:-PrintTenuringDistribution
  - ✓ -XX:+PrintGCTimeStamps
  - ✓ -Xloggc:/foo/bar/gc.log