



**Java™
Italian
Portal**

www.javaportal.it



Tuning, Performances & Problem Solving sugli Application Server

Linee guida generali, controlli a 360 gradi dal sistema operativo al codice.

Spesso quando un Application Server risponde lentamente si dà subito la colpa al produttore, ma ci sono molte cose che devono essere controllate e fatte prima durante e dopo l'installazione, vediamole e commentiamole insieme



Fabrizio Marini

Nel 1996 è stato uno dei due fondatori della K-Tech di cui è stato socio fino al 2003. In K-Tech era il responsabile della formazione e dei clienti strategici dell' "Area Java". E' l'ideatore di JIP – Java Italian Portal (www.javaportal.it).

Ha iniziato a programmare in Java nel 1996 e da allora si è specializzato in questo settore, fornendo consulenze alle più prestigiose ditte del mercato italiano.

Tra il 2000 ed il 2001 ha tenuto numerosi corsi per la Selfin SpA (Centro di Istruzione IBM) inerenti la programmazione e l'analisi ad oggetti, Java, WebSphere ed UML.

Da Gennaio 2001 ha iniziato ad insegnare per BEA Systems Italia specializzandosi sulla piattaforma WebLogic secondo lo standard J2EE1.2 & 1.3, in particolare ha tenuto corsi su WebLogic, EJB & JMS, Personalization & Commerce Server, Business Process Management - WebLogic Integration.

A Febbraio 2002 in collaborazione con SilverStream ha progettato ed aperto il primo UDDI in Italia. (www.webservicesportal.it)

Il 10.02.2003, lascia la K-Tech ed entra nei Professional Services di Bea Italia in qualità di Principal Consultant.



- Sistema Operativo
- Patch Sistema Operativo



- Quale JDK ?
- Patch per Java per specifico JDK
- Patch per per Java per specifico Sistema Operativo
- Riconoscere livello di PATCH per JAVA



- Configurazione TCP/IP
- Close Wait
- Connessioni “appese”
- Configurazione File Descrittori



- Installazione Application Server
- Livelli di Patch per Application Server



- Production Mode
- Classpath
 - ordine elementi
 - patch
 - Riconoscere le librerie di default
 - Attenzione ai driver jdbc



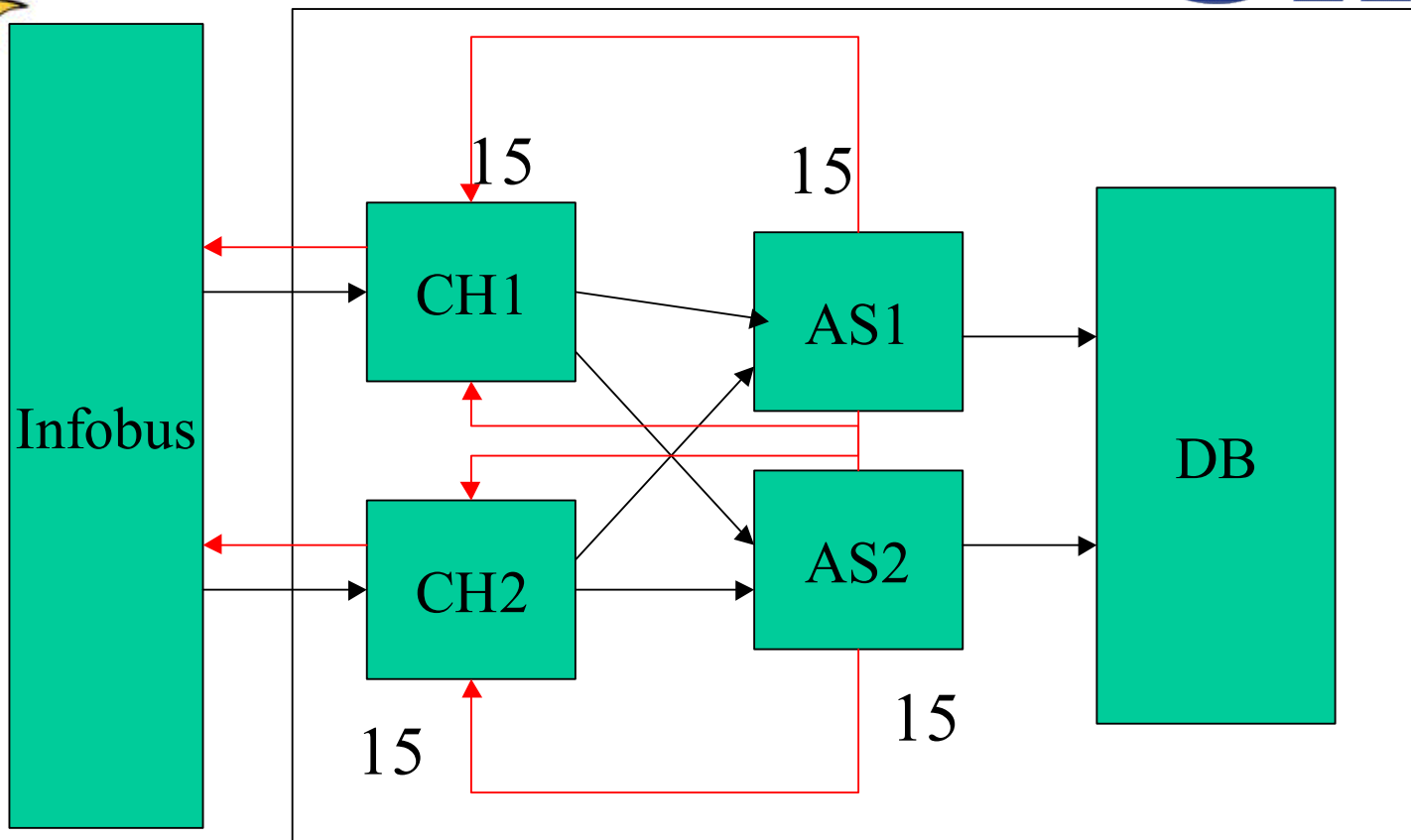
- Driver per DB
 - leggere il manifest del classes12.zip
- Dimensionamento Connection Pool
 - Dimensione rispetto ai Thread di Esecuzione
- XA Driver
 - SetTimeout



- Dimensionamento Pool EJB
 - Evitare Pool che crescono a dismisura
 - MAX Bean in cache
 - MDB se non settati assumono come numero MAX il numero degli Execute Thread



- Execute Thread Application Server
 - Dimensioni
 - Suddivisione per tipologia html, rmi
 - Kill -3



Per il funzionamento dei processi, i CH chiamano gli AS che possono, a loro volta, chiamare di nuovo i CH per uscire su Infobus. Ma se per qualsiasi ragione i thread dei CH sono esauriti in quanto stanno soddisfacendo le richieste che arrivano dall'esterno, non saranno neanche in grado di esaudire le richieste che arrivano loro dagli AS, ingenerando il loop.



- Dimensioni File System
- Spazio per core dump
- Directory dedicate per LOG e Code persistenti
- Livello LOG
 - System.out()
 - log4j



- JMS: code persistenti su DB o File System?
- Messaggi JMS e Properties
 - Tempo di vita
 - Destinazione errori
 - redelivery



- Cluster
 - Multicast 237.0.0.1
 - Round Robin
 - Fail Over
- Load Balancer
- HTTP Plug-in



- Configurazione Memoria JVM

-jdk131 & jdk141

-Grosse quantità di memoria



Tuning Garbage Collection with the 1.4.2 Java™ Virtual Machine

The *throughput* collector: this collector uses a parallel version of the *young* generation collector. It is used if the `-XX:+UseParallelGC` option is passed on the command line. The *tenured* generation collector is the same as the default collector.

The *concurrent* low pause collector: this collector is used if the `-XX:+UseConcMarkSweepGC` is passed on the command line. The concurrent collector is used to collect the *tenured* generation and does most of the collection concurrently with the execution of the application. The application is paused for short periods during the collection. A parallel version of the *young* generation copying collector is used with the concurrent collector if the combination `-XX:+UseConcMarkSweepGC -XX:+UseParNewGC` is passed on the command line.

The *incremental* (sometimes called *train*) low pause collector: this collector is used only if `-Xincgc` is passed on the command line. By careful bookkeeping, the incremental garbage collector collects just a portion of the *tenured* generation at each minor collection, trying to spread the large pause of a major collection over many minor collections. However, it is even slower than the default *tenured* generation collector when considering overall throughput.



-Parametri da passare alla JVM

-Leggere le release notes

-Attenzione a JNI

-Loop?

-Xss<val>K

-XX:ThreadStackSize=<val>k



-Codice:

- Web Application, JSP/Servlet e variabili di classe
- Web App & Multi Threading
- Singleton
- Variabili Statiche
- Ejb e gestione Transazioni
- Sessioni, oggetti serializzabili
- Evitare Stringhe scolpite e/o file properties
- XML



-Tool

-Top ☺

-wlshe11, JMX (eventi)

-jvmstat



Feature	1.3 option - Solaris 8	1. Tempi
Many-to-Many thread based synchronization	default	698 - 658
Many-to-Many lwp based synchronization	-XX:+UseLWPSynchronization	1712 - 1734
One-to-One via bound threads	-XX:+UseBoundThreads	686 - 652 626 - 671
One-to-One via Alternate Libthread	export LD_LIBRARY_PATH=/usr/lib/lwp	580 - 629
Use native thread priorities	-XX:+UseThreadPriorities	1650 - 1740
Thread local portions of the heap used in the young generation	-XX:-UseTLE	5812 - 6001



1) <http://java.sun.com/docs/books/vmspec/2nd-edition/html/Concepts.doc.html#33308>

"If two or more concurrent threads act on a shared variable, there is a possibility that the actions on the variable will produce timing-dependent results."

2) <http://java.sun.com/docs/books/vmspec/2nd-edition/html/Threads.doc.html#21294>

"

8.9 Discussion

Any association between locks and variables is purely conventional. Locking any lock conceptually flushes all variables from a thread's working memory, and unlocking any lock forces the writing out to main memory of all variables that the thread has assigned. That a lock may be associated with a particular object or a class is purely a convention. For example, in some applications it may be appropriate always to lock an object before accessing any of its instance variables; synchronized methods are a convenient way to follow this convention. In other applications, it may suffice to use a single lock to synchronize access to a large collection of objects.

If a thread uses a particular shared variable only after locking a particular lock and before the corresponding unlocking of that same lock, then the thread will read the shared value of that variable from main memory after the lock operation, if necessary, and will copy back to main memory the value most recently assigned to that variable before the unlock operation. This, in conjunction with the mutual exclusion rules for locks, suffices to guarantee that values are correctly transmitted from one thread to another through shared variables.

The rules for volatile variables effectively require that main memory be touched exactly once for each use or assign of a volatile variable by a thread, and that main memory be touched in exactly the order dictated by the thread execution semantics. However, such memory operations are not ordered with respect to read and write operations on nonvolatile variables.

"



Ringraziamenti

KeyTECH

SOLUZIONI INFORMATICHE