Smarten
Advanced Data Discovery
Powered by ElegantJ BI

Application Server Optimization Guidelines

Business Intelligence & Advanced Data Discovery

Document Version: 6.0
Product Version: 5.0 and above

Document Information	
Document ID	Smarten-ApplicationServer-Optimization-Guidelines
Document Version	6.0
Product Version	5.0 and above
Date	14-Dec-2018
Recipient	NA
Author	EMTPL

Statement of Confidentiality, Disclaimer and Copyright

This document contains information that is proprietary and confidential to EMTPL, which shall not be disclosed, transmitted, or duplicated, used in whole or in part for any purpose other than its intended purpose. Any use or disclosure in whole or in part of this information without the express written permission of EMTPL is prohibited.

Any other company and product names mentioned are used for identification purpose only, may be trademarks of their respective owners and are duly acknowledged.

Disclaimer

This document is intended to support administrators, technology managers or developers using and implementing Smarten. The business needs of each organization will vary and this document is expected to provide guidelines and not rules for making any decisions related to Smarten. The overall performance of Smarten depends on many factors, including but not limited to hardware configuration and network throughput.

Preface

The Smarten Application Server Optimization Guidelines are part of the documentation set for **Smarten Advanced Data Discovery Version 5.0.0.xxx.**

This document is based on Application Server WildFly v 11 (formerly JBoss Application server), and you are advised to verify suitability of recommendations in this document for the Application Server version you may be using.

Overall, performance tuning is a very important part of creating, maintaining, and deploying a successful Enterprise application.

When organizations select application middleware, performance is always one of their important selection criteria, if not the most important.

To get the most from your investment in middleware, administrators need to know the specific ways through which they can achieve superior performance with the WildFly Application Server. While we would all like to think that an application could perform well straight out of the box, this is usually not the case. Applications can have widely varying characteristics, and while some applications might perform well with default middleware settings, others will not.

If you are new to WildFly or performance tuning, the guidelines provided introduce best practices that can help you avoid common performance pitfalls as you prepare your application for a production environment. If you are an old hand at application performance issues, you may benefit from an update on best practices in WildFly Application Server performance tuning.

Related Documents:

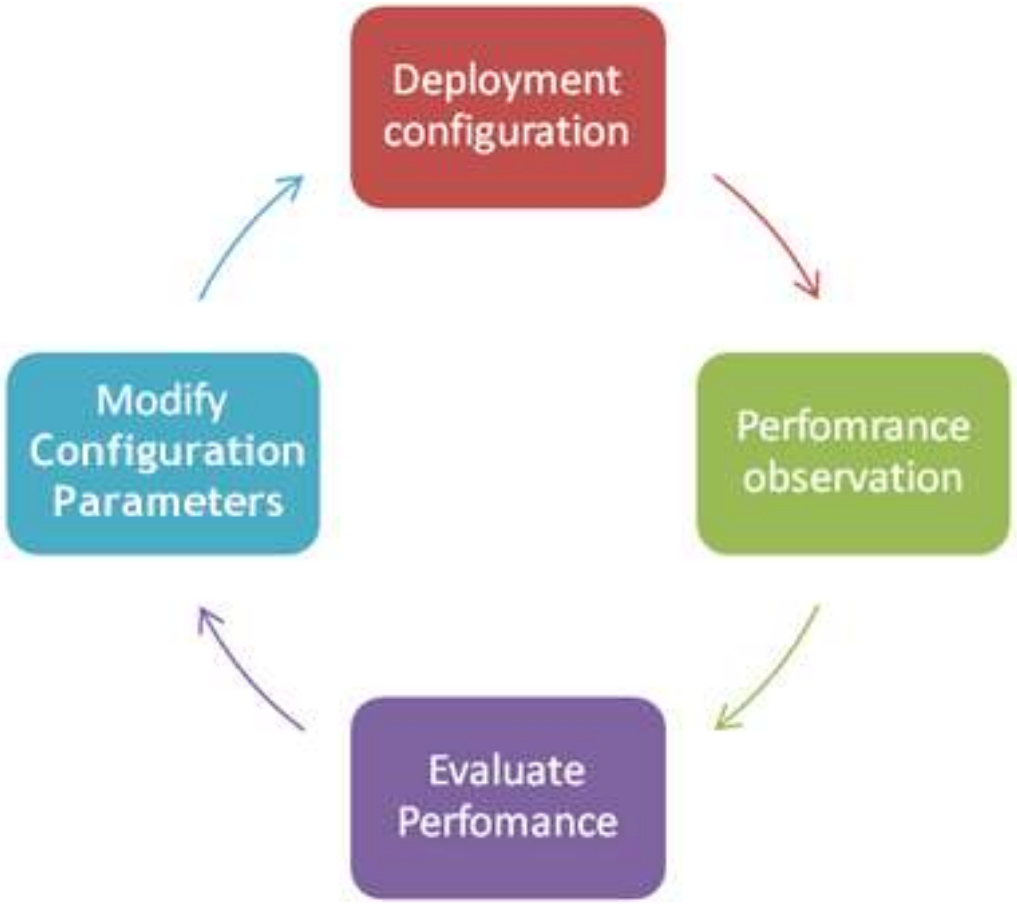
Smarten-Technical-Specifications	Prerequisites and Compatibility of supported software platforms and Client/Server environment for Smarten
Smarten-Installation-Manual	Installation Manual contains information on installing and configuring the Smarten on different application servers with supported operating system platforms

Contents

1 Introduction

Better system performance depends on better design and implementation.
WildFly tuning involves the environment on which WildFly is running, the JVM settings, and OS settings on which WildFly can produce best results.

The performance improvement cycle is shown by the following figure:



2 System Configuration

This requirement is specific to Smarten and may vary for other applications.
For best performance results, make sure that your system specifications meet at least the **recommended system requirements**.

Please refer to the Smarten Technical Specifications guide for details on prerequisites and compatibility with hardware and software platforms for running Smarten.

For specific scenarios, contact the technical support team for hardware sizing and estimation.

Resources required on a particular system depend on the following factors:

- Size of source data
- Complexity of extraction logic, e.g., SQL query, from source data
- Size and design of cube and metadata
- Size of analytical objects
- Intensity of data operations
- Concurrent users
- Other related factors

3 Tune Operating System

For Windows or Linux platforms, the following settings are recommended:

1. Choose 64-Bit OS platform
2. Install WildFly on freshly installed OS
3. Root directory of WildFly should be directly on drive's root and NOT in any other folder

> Note:
> It is recommended that you use the NTFS file system rather than FAT or FAT32 when running the Application Server on any Microsoft Windows platform and Ext3/Ext4 on Linux.

Linux-specific tuning

Since everything is a file in Linux, check the file-max parameter.

Check the value for file-max parameter using the following command:
cat /proc/sys/fs/file-max

In case the parameter is not set, the user can set the value for the memory size.
Path: /etc/
File Name: sysctl.conf

Recommendation

It is recommended that you change the settings below only to fine-tune TCP/IP stack and virtual memory in order to improve system performance.

Example
For 8 GB RAM
set fs.file-max=813916

Increase default socket send/receive buffer

By default, the Linux network stack is not configured for high-speed large file transfer across network links. This is done to save memory resources. You can easily tune the Linux network stack by increasing network buffer size for high-speed networks that connect server systems to handle more network packets.

The default value of rmem_max and wmem_max is about 128KB in most Linux distributions, which may be enough for a low-latency, general-purpose network environment. However, if the latency is large, the default size might be too small.

Please note that the following settings can increase memory usage on your server:

Check the current values for Send/receive buffer:
Path: /proc/sys/net/core

The default and maximum amount for the receive socket memory:
cat rmem_default
cat rmem_max

The default and maximum amount for the send socket memory:
cat wmem_default
cat wmem_max

In case latency is large and the buffer parameters are set to 131072 (128KB), you can modify the max OS send buffer size (wmem) and receive buffer size (rmem) up to 12582912 (12MB) for queues on all protocols.

Path: /etc
File Name: sysctl.conf

Recommendation
It is recommended to expand the buffer size in case network latency is higher.

Example
echo 'net.core.wmem_max=12582912' >> /etc/sysctl.conf
echo 'net.core.wmem_default=12582912' >> /etc/sysctl.conf
echo 'net.core.rmem_max=12582912' >> /etc/sysctl.conf
echo 'net.core.rmem_default=12582912' >> /etc/sysctl.conf

Tune the virtual memory manager

One of the most important aspects of an operating system is the Virtual Memory Management system.

Virtual memory (VM) or virtual memory addressing is a memory management technique used by multitasking computer operating systems wherein noncontiguous memory is presented to software (process) as contiguous memory. This contiguous memory is referred to as the virtual address space.

Virtual memory allows an operating system to perform many of its advanced functions, such as process isolation, file caching, and swapping.

vm.swappiness sets affinity for a process to be swapped to swap space. On setting swappiness to 100, the machine would observe high swap activity. On setting the swappiness to zero, swapping activity would be minimal or nil. For a production web server / Virtual machine host user, swappiness should ideally be set to zero.

In case of memory pressure, the following amendment is recommended to prevent applications from being swapped to disk:

Path: /etc
File Name: sysctl.conf

set vm.swappiness = 0

> Note:
> All the commands mentioned in Linux-specific tuning are specific to RED Hat Linux distribution.

Tune the number of open files parameter

By default, most Linux systems are configured to allow a relatively small number of open files, e.g., 1024.

Certain applications, such as a web server with lots of database connections, require a large number of open files.

When your application or command or script is hitting the max open file limit allowed by Linux, you will get an error message: “Too many open files.”

You need to tune this parameter to handle this error.

Soft limit: This is the minimum number of files that Linux can handle.

To check the current Soft Limit, use the following command:

```
ulimit  -Sn
```

Hard Limit: This is the maximum number of files that Linux can handle.

If the number of opened files crosses the soft limit, then Linux will issue a warning message but will let the system exceed the number of files to Hard limit.

Soft limits could be set by any user, while hard limits are changeable only by the root user.

To check the current Hard Limit, use the following command:

```
ulimit  -Hn
```

To change the soft limit and hard limit for a system:

Path : vi /etc/security/
File Name: limits.conf

And then add the following two lines in the above-mentioned file:

```
UserName soft nofile 4096
UserName hard nofile 20480
```

(UserName= User name of Linux users or to set for all users use '*')

Recommendation

It is recommended to increase these limits as needed rather than setting large limits at random. You need to reboot the Linux machine to apply these settings.

After rebooting, check the current limits again to verify settings.

```
ulimit  -Sn
ulimit –Hn
```

4 Tune Application Server

The server JVM is better suited for applications running constantly over the long run; for example, the Server VM is much faster than the Client VM. To enable it, simply set the server option on the command line.

For Linux

WILDFLY_HOME/bin/standalone.conf

For Windows

WILDFLY_HOME/bin/standalone.conf.bat

Use Server Virtual Memory

```
set JAVA_OPTS=%JAVA_OPTS% -server
```

> **Note**:
> If you do not use Server VM, it will run WildFly under client VM option, which will reduce server performance.

4.1 Tune the heap size

The allocation of memory for the JVM is specified using -X options when starting WildFly (these options may depend upon the JVM that you are using; the examples here are for the Sun JVM).

JVM option	Meaning
-Xms	initial java heap size
-Xmx	maximum java heap size

By default, the virtual machine grows or shrinks the heap at each collection to try to keep the proportion of free space to live objects at each collection within a specific range.

It is good practice with server-side Java applications like WildFly to set the minimum -Xms and maximum -Xmx heap sizes to the same value. This will help WildFly prevent out of memory errors.

Set -Xms and -Xmx to the same value. This increases predictability by removing the most important sizing decision from the virtual machine.

This value should be between 65% and 70% of physical available memory without page fault.
In the examples below, we have provided different memory settings between 65% and 70% verifying page fault.

Windows platforms:
Path: WILDFLY_HOME\bin
File Name: standalone.conf.bat

Linux platforms:
Path: WILDFLY_HOME/bin
File Name: standalone.conf

Example
For 64 GB RAM
java -Xmx1024m –Xms51200m

For 32 GB RAM
java -Xmx1024m –Xms25600m

For 16 GB RAM
java –Xmx2048m –Xms12288m

> Note:
> Do not define the Xms & Xmx settings more than 70%. More than 70% may create a page fault in memory, causing degrading of I/O and performance.
> Do not continue increasing heaps with increasing user base above prescribed limits to avoid page faults.

4.2 Use Large memory pages

For 64-bit systems, use of large memory pages can improve performance significantly. The default memory page size is typically 4KB. When you are addressing large amounts of memory, this quickly adds up to a large number of memory pages—just one gigabyte requires 262,144 memory pages. That is a lot for a system to keep track of, which translates to a lot of system overhead. Aside from helping you avoid the overhead of mapping so many memory pages, large memory pages on Linux cannot be swapped to disk. This is a major advantage because having your heap space swap to disk can wreak havoc on the performance of your application. Large page support begins at 2MB and can

run as high as 256MB on some hardware architectures. These numbers will vary, and you will need to find out the values for your specific server. All the major JVM systems support large memory pages on Linux.

Add the following command in the below-mentioned file:
Path: WILDFLY_HOME /bin
File Name: standalone.conf
-XX:+UseLargePages

4.3 Tune the garbage collector

An object is considered garbage when it can no longer be reached from any pointer in the running program. The most straightforward garbage collection algorithms simply iterate over every reachable object. Any objects left over are then considered garbage. The time this approach takes is proportionate to the number of live objects at that point of time.

> Note:
> It's been demonstrated that an application that spends 10% of its time in garbage collection can lose 75% of its throughput when scaled out to 32 processors.

4.4 Turn off Distributed GC & Turn On Parallel Collector

The RMI (**Java Remote Method Invocation**) system provides a reference counting distributed garbage collection algorithm. This system works by having the server keep track of which clients have requested access to remote

objects running on the server. When a reference is made, the server marks the object as "dirty" and when a client drops the reference; it is marked as being "clean."

This system is quite expensive, however, and by default runs every minute.

Recommendation—set it to run at least every 30-minute interval

The parameter is specified in milliseconds. **Parameter settings in**

Windows platforms:
Path: WILDFLY_HOME \bin
File Name: standalone.conf.bat

Linux platforms:
Path: WILDFLY_HOME /bin
File Name: standalone.conf

Example
set JAVA_OPTS=%JAVA_OPTS% -Dsun.rmi.dgc.client.gcInterval=1800000
-Dsun.rmi.dgc.server.gcInterval=1800000 –XX

- Parallel collector (-XX:+UseParallelGC): It performs GC collections in parallel. This collector is fit for multiprocessor machines and applications requiring high throughput. It is also a suggested choice for applications that produce a fragmented Java heap, allocating large-size objects at different timelines.

- Concurrent collector (-XX:+UseConcMarkSweepGC): It performs most of its work concurrently using a single garbage collector thread that runs with the application threads simultaneously. It is fit for fast processor machines and applications with a strict service-level agreement. It can be the best choice and also is good for applications using a large set of long-lived objects live Http Sessions.

Windows platforms:
Path: WILDFLY_HOME \bin
File Name: standalone.conf.bat

Linux platforms:
Path: WILDFLY_HOME /bin
File Name: standalone.conf

Example
set JAVA_OPTS=%JAVA_OPTS% -XX:+UseParallelGC -XX:+UseConcMarkSweepGC

4.5 Turn on G1GC Collector

One of the major enhancements in Java 7 is the new G1GC ("Garbage first") low-latency garbage collector planned to replace CMS (Concurrent Mark Sweep) in the Hotspot JVM. It is a server-style collector targeted at multiprocessor machines with large amounts of memory.

The G1GC collector is a departure from earlier collectors that had a physical separation between the young and old generations. With G1, even though it is generational, there is no physical separation between the two generations. This collector divides the entire space into regions and allows a set of regions to be collected rather than split the space into an arbitrary young and old generation.

The key features of the G1GC collector:

1. G1 uses parallelism, which is mostly used in hardware today. The main advantage of G1 is that it is designed to make use of all the available CPUs and utilize the processing power of all CPUs as well as increase performance and speed up the garbage collection.
2. The Next feature, which plays a key role in increasing garbage collection, is treating the young objects (newly created) and the old objects (that have lived for some time) differently. G1 mainly focuses on young objects, as they can be reclaimable when traversing the old objects.
3. Heap compaction is done to eliminate fragmentation problems. In essence, because G1 compacts as it proceeds, it copies objects from one area of the heap to the other. Therefore, because of compaction, it will not encounter fragmentation issues that CMS might. There will always be areas of contiguous free space from which to allocate, allowing G1 to have consistent pauses over time.

G1GC tuning parameters:

1. **-XX:MaxGCPauseMillis=n** - Set the target of maximum GC pause time. This is a soft goal and JVM will make best effort to achieve it. The default value is 200 milliseconds. For large heap it is recommended to keep it higher (500 milliseconds).
2. **-XX:InitiatingHeapOccupancyPercent=n** - Sets the Java heap occupancy threshold that triggers a marking cycle. The default occupancy is 45% of the entire Java heap. For memory intensive applications, it is recommended to keep it 25%.

3. **-XX:ParallelGCThreads=n** – This parameter defines maximum number of parallel GC threads. Default value is 8. If there are more than 8 cores available in computer, set the value of n to approximately 5/8th of the total number of cores.

Windows platforms:
Path: WILDFLY_HOME \bin
File Name: standalone.conf.bat

Linux platforms:
Path: WILDFLY_HOME /bin
File Name: standalone.conf

Example
set JAVA_OPTS=%JAVA_OPTS% -XX:+UnlockExperimentalVMOptions -XX:+UseG1GC -XX:+UeG1GC -XX:ConcGCThreads=1 -XX:ParallelGCThreads=2 -XX:MaxGCPauseMillis=500 -XX:InitiatingHeapOccupancyPercent=25

4.6 Tune PermGen

In the JVM, PermGen holds the metadata about classes that have been loaded or created. This information is garbage collected like the other parts of the heap, but some rough edges can prevent this from happening, class loaders in particular. Generally, the amount of PermGen space needed is small in relation to the rest of the heap, and default JVM values should work for you.

Ensure that adequate PermGen space is set. The default is 64MB on the Sun VMs, but for applications like Smarten, the default value of 64MB may not be sufficient, so the recommendation to configure the PermGen is 512MB.

Recommendation
This value should be between 11% and 13% of physical available memory to avoid out of memory issues.

> Note:
> Defining PermGen would help us prevent out of memory issues.

Windows platforms:
Path: WILDFLY_HOME\bin
File Name: standalone.conf.bat

Linux platforms:
Path: WILDFLY_HOME/bin
File Name: standalone.conf

Example
For 16 GB RAM
set JAVA_OPTS=%JAVA_OPTS% -XX:PermSize=5120m -XX:MaxPermSize=5120m

For 32 GB RAM
set JAVA_OPTS=%JAVA_OPTS% -XX:PermSize=10240m -XX:MaxPermSize=10240m

4.7 Turn on aggressive optimizations

The specific optimizations enabled by this option reduced response times on JVM workload by approximately 8%.

Windows platforms:
Path: WILDFLY_HOME \bin
File Name: standalone.conf.bat

Linux platforms:
Path: WILDFLY_HOME /bin
File Name: standalone.conf

Example
set JAVA_OPTS=%JAVA_OPTS% -XX:+AggressiveOpts

5 Tune Apache Spark

You can tune or configure Apache Spark related parameters in spark.config file. It is located in SMARTEN_HOME/conf folder. Following are some important tuning parameters:

Parameter name	Description
spark.master	This parameter specifies the URL for a master node. By default, it runs in a local mode with "local[*]" parameter value. Local[n] - Spark will utilize n number of cores for a particular task. local[*] - Spark will utilize all cores on server. In case of high concurrent usage, instead of local[*], it is recommended to set *n* to less number of cores.
write.partitions	This parameter specifies number of partitions to be created for writing parquet files. Default value is 2. If there is a large data to write and there is low CPU utilization, you can set this number to higher. So, writing process will be completed faster.
spark.shuffle.file.buffer	Size of the in-memory buffer for each shuffle file output stream. These buffers reduce the number of disk seeks and system calls made in creating intermediate shuffle files. Default value is 32K. If you have more memory available and spark.sql.shuffle.partitions parameter is set to a less number, you can keep higher value for this parameter. It will reduce IO operations and will speed up data reading during the shuffle partitioning.
spark.sql.shuffle.partitions	Number of partitions in RDDs returned by transformations like join, reduceByKey, and parallelize. In case of large data and server with more CPU power, you can keep this value higher depending upon number of cores. So, data will be partitioned accordingly and will be processed in parallel.
spark.memory.fraction	Fraction of (heap space - 300MB) used for execution and storage. Default value is 0.6. It means spark will utilize 60% of (Heap memory-300MB) memory for storage and execution. Rest will be used by application server. The lower value of this parameter will lead to more frequent spills and cached data eviction occurs. Value

	of this parameter should be between 0 to 1. If there are more number of cubes or objects placed in memory and evictions are frequent, you can increase this parameter value. Same, in some scenario, if there are more GC events happening, you can decrease this value.

6 Monitoring and Improving Performance

Introduction to Java VisualVM

Java VisualVM is a tool that provides a visual interface for viewing detailed information about Java applications while they are running on a Java Virtual Machine (JVM) and for troubleshooting and profiling these applications. Various optional tools, including Java VisualVM, are provided with Sun's distribution of the Java Development Kit (JDK) for retrieving different types of data about running JVM software instances. For example, most of the previously stand-alone tools JConsole, jstat, jinfo, jstack, and jmap are part of Java VisualVM. Java VisualVM federates these tools to obtain data from the JVM software and then reorganizes and presents the information graphically to enable you to view different data about multiple Java applications uniformly whether they are running locally or on remote machines. Furthermore, developers can extend Java VisualVM to add new functionality by creating and posting plug-ins to the tool's built-in update center.

Java VisualVM can be used by Java application developers to troubleshoot applications and to monitor and improve the applications' performance. Java VisualVM can allow developers to generate and analyze heap dumps, track down memory leaks, browse the platform's MBeans and perform operations on those MBeans, perform and monitor garbage collection, and perform lightweight memory and CPU profiling.

Starting Java VisualVM

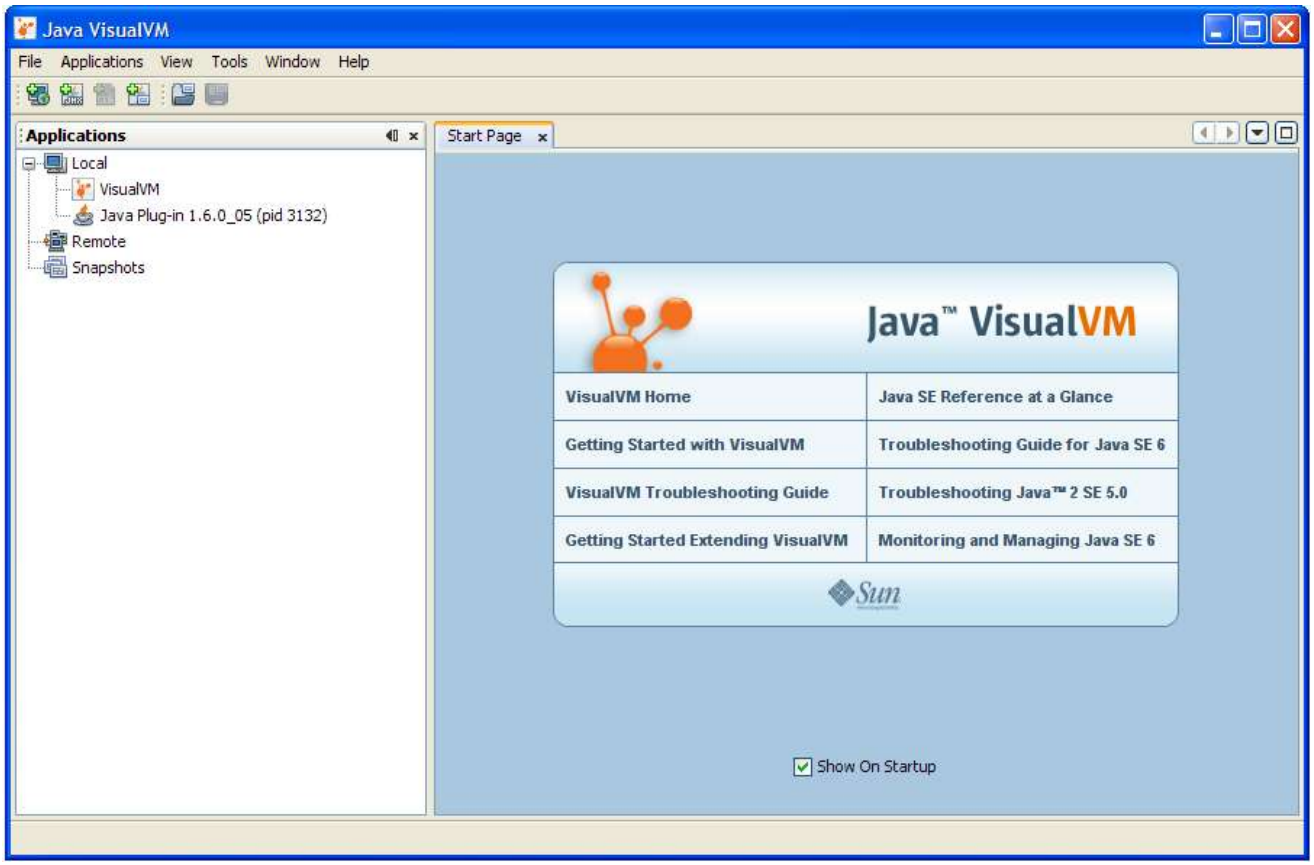
Java VisualVM is bundled with JDK version 6 update 7 or greater. Once you have installed an appropriate JDK version, navigate to your JDK software's bin directory and double-click the Java VisualVM executable. On Windows/Linux, this directory will be the following:

JAVA_HOME\bin

Alternatively, navigate to your JDK software's bin directory and type the following command at the command (shell) prompt:

Jvisualvm

The following window will appear



For advance usage of this 0tool for monitoring please visit
http://docs.oracle.com/javase/6/docs/technotes/guides/visualvm/

7 Product and Support Information

Find more information about ElegantJ BI-Smarten and its features at www.smarten.com
Support: support@smarten.com
Sales: sales@smarten.com
Feedback & Suggestions: support@smarten.com
Support & Knowledgebase Portal: support.smarten.com