TES 6x Performance Tuning Whitepaper (9-2012)

Overview

Due to the black-box nature of software components and the complexities involved in their interactions, performance tuning can sometimes appear to be more art than science. Our goal in this document is to provide you with tips and guidelines to hopefully make it more science. The document is divided into three main sections: logging, JConsole and performance tuning. The first two sections cover how to determine what needs to be tuned, and the third section covers how to tune it.

Terminology

This document will refer to several terms and abbreviations that are defined here:
Master - the server that handles all of TES business logic.
Client Manager (CM) - the server that interacts with all the TES web and REST-based API clients.
Java Virtual Machine (JVM) - the Java application that runs TES software.
Data Source Provider (DSP) - the module that resides in the Client Manager that contains the actual "TES client" software that interacts with the TES web clients.
Cache or Data Cache - the database that the Client Manger uses to store read-only data from the Master. This is an exact copy of the data from the Master database.

Logging

Configuring the correct log settings can help capture the right information and right amount of information for a given situation. The following log settings should be configured in the DSP file.

Logging is divided into the following categories.

DspLog - core system logging
CacheLog - data cache related logging
RequestLog - logging for high-level communication between CM and Master
RpcLog - logging for low-level communication between CM and Master
ClientLog - logging for client-related operations
DebugLog - miscellaneous debug logging
CommonsLog - basic logging from common system libraries

Each category's verbosity can be set with the following values, in order of increasing verbosity.

SEVERE
WARNING
AUDIT
INFO
CONFIG
FINE
FINER
FINEST

FINE is enough verbosity for most diagnostics and debugging. This is the recommended setting for normal operation. FINEST is necessary only for capturing specific information for a short duration of time.

Startup Sequence

There are several steps in the CM's startup sequence, including the primary sync, loading adapters and the secondary sync. Knowing what each step does and how it is logged is important in diagnosing performance problems. The clientmgr.out and DSP log files will contain the relevant logging.

Primary Sync

The purpose of the primary sync is for the DSP to fetch definition data (schedules, jobs, users, events, etc.) from the Master. The DSP does need to re-fetch data that it has already fetched. Therefore, the first primary sync with an empty or deleted cache will usually take the longest.

A normal primary sync should look like following.

ClientNode: Client starting ...
ClientNode: Connecting to [tcp://localhost:6215].
ClientNode: Connected to [tcp://localhost:6215].

CacheSynchronizer: Synchronizing primary objects ...
CacheSynchronizer: [Users] Sync Range: 0 => 84.
CacheSynchronizer: [Users] Sync Range: 85 => 415.
CacheSynchronizer: [Users] Last sync time = 12/31/1969 16:00:00.
CacheSynchronizer: [Users] Cached 6 object(s).
CacheSynchronizer: [Users] Cached 6 object(s) total.

CacheSynchronizer: Primary objects synchronized in 141 seconds.

Loading Adapters

After the primary sync, the DSP will load and install the adapters. You should see the following logs during this step.

DataSourceProviderImpl: Initializing adapters ...

AdapterPluginManager: Installing adapter plugins.

AdapterPluginManager: Installing [WebService].
AdapterPluginManager: [WebService] version = W.X.Y.Z.
AdapterPluginManager: [WebService] Deployed new JAR.
AdapterPluginManager: [WebService] Deployed new WAR.
AdapterPluginManager: Loading [WebService].

DataSourceProviderImpl: Adapters initialized: 37 seconds.
DataSourceProviderImpl: Client initialized.

Note: after this step, users can point their web clients to the CM. However, due to the performance impact of the secondary sync, we recommend waiting until after the secondary sync completes.

Secondary Sync

The purpose of the secondary sync is for the DSP to fetch all runtime data (job history, event history, message logs, etc.) from the Master. Like the primary sync, the DSP will not re-fetch any data that it has already fetched in the secondary sync. Thus, the first time the secondary sync runs will usually be the longest. In addition, due to the huge size difference between runtime data vs. definition data, the secondary sync may take many times longer than the primary sync.

You should see the following logging during the secondary sync.

CacheSynchronizer: Synchronizing secondary objects ...

CacheSynchronizer: [MessageLog] Sync Range: 363229 => 363233.
CacheSynchronizer: [MessageLog] Last sync time = 12/31/1969 16:00:00.
CacheSynchronizer: [MessageLog] Last sync time = 12/31/1969 16:00:00.
CacheSynchronizer: [MessageLog] Cached 5 object(s).
CacheSynchronizer: [MessageLog] Cached 5 object(s) total.

CacheSynchronizer: Secondary objects synchronized in 1 seconds.

After the secondary sync, it is perfectly ok to start pointing web clients to the CM.

JConsole

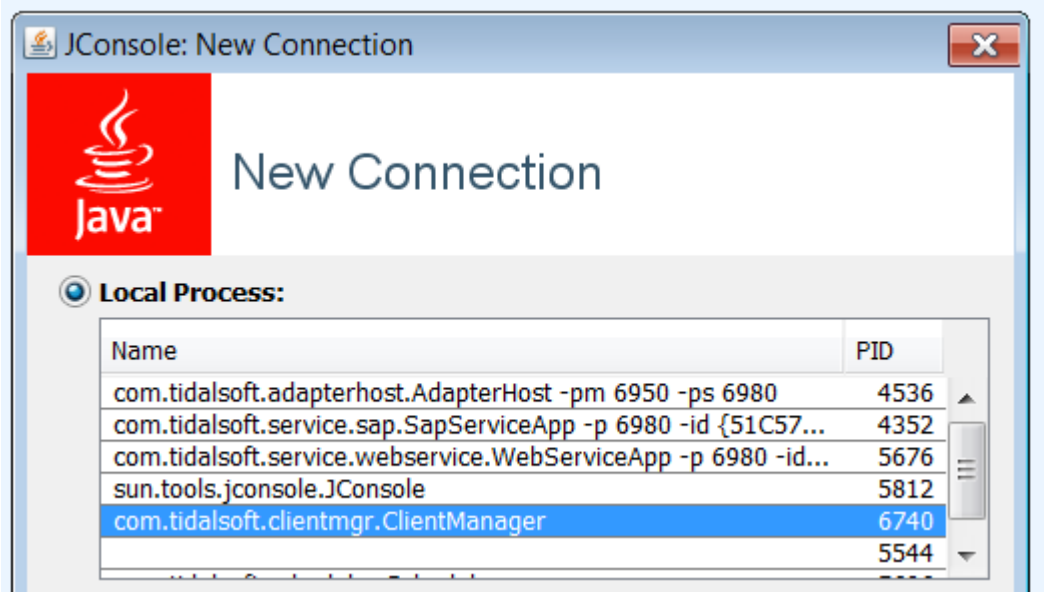
JConsole is a powerful tool that can connect to any JVM (including the CM and Master JVMs) and provide useful diagnostic information. It is free and comes with the Java Development Kit (JDK).

Connecting JConsole

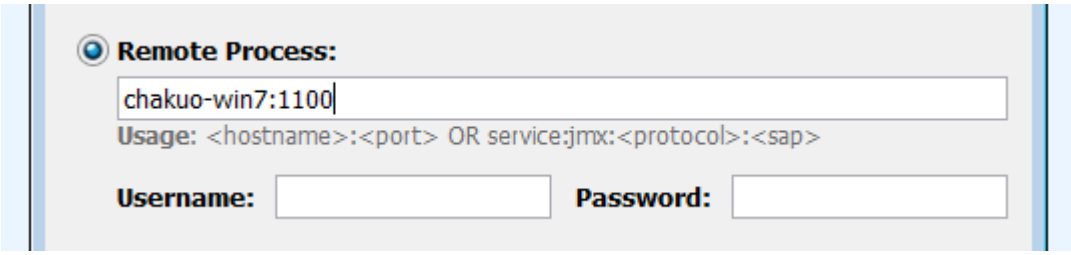
Before connecting JConsole to either the CM or Master, make sure the following property is set in clientmgr.props or master.props.

JmxOn=Y

If you are running JConsole on the same machine as the JVM you are connecting to, the JVM will be listed in JConsole.



If you are connecting JConsole to a JVM running on a remote machine, type in the remote JVM's machine host name and port (the default CM port is 1100).

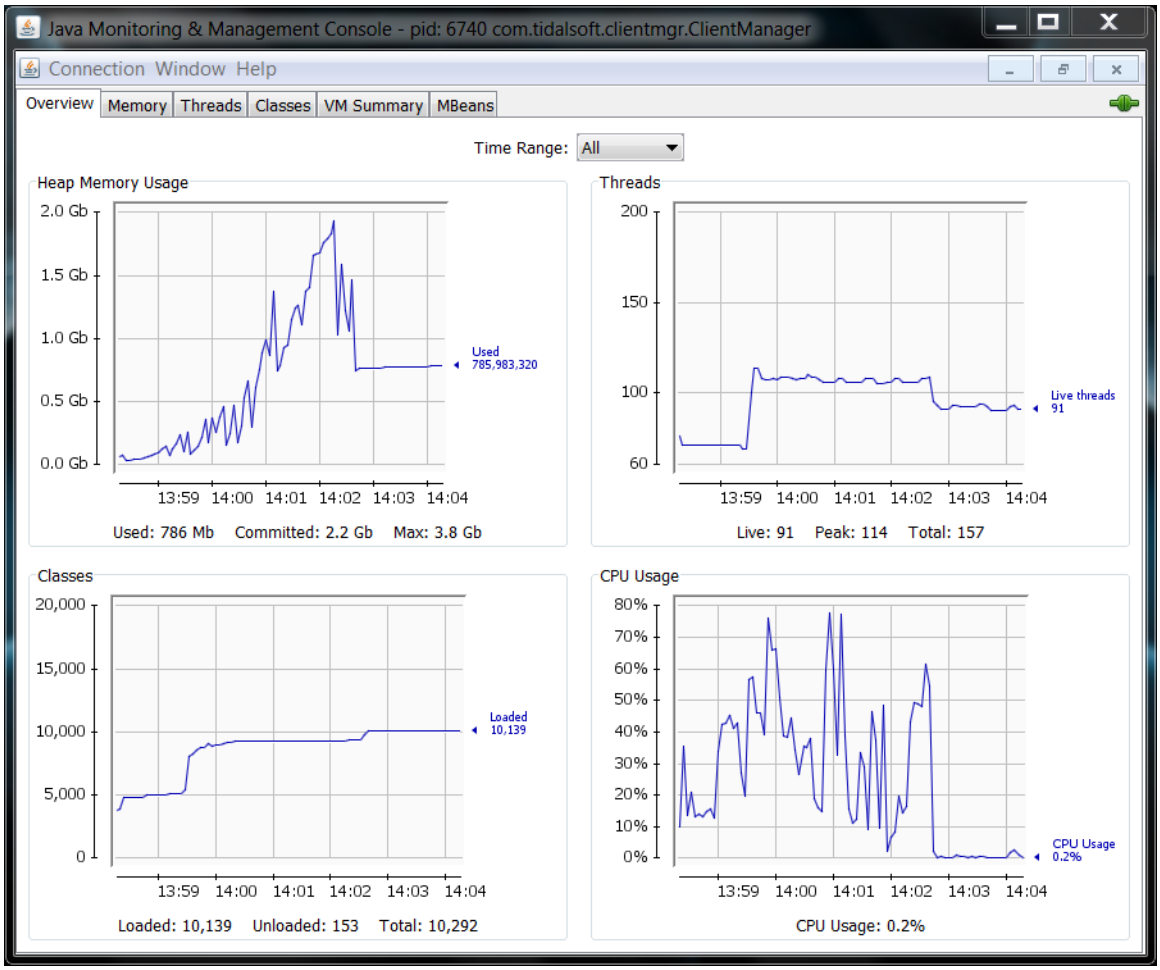


Note: you can change the default port for the JVM by setting the following property in clientmgr.props or master.props.

JmxRmiPort=1200

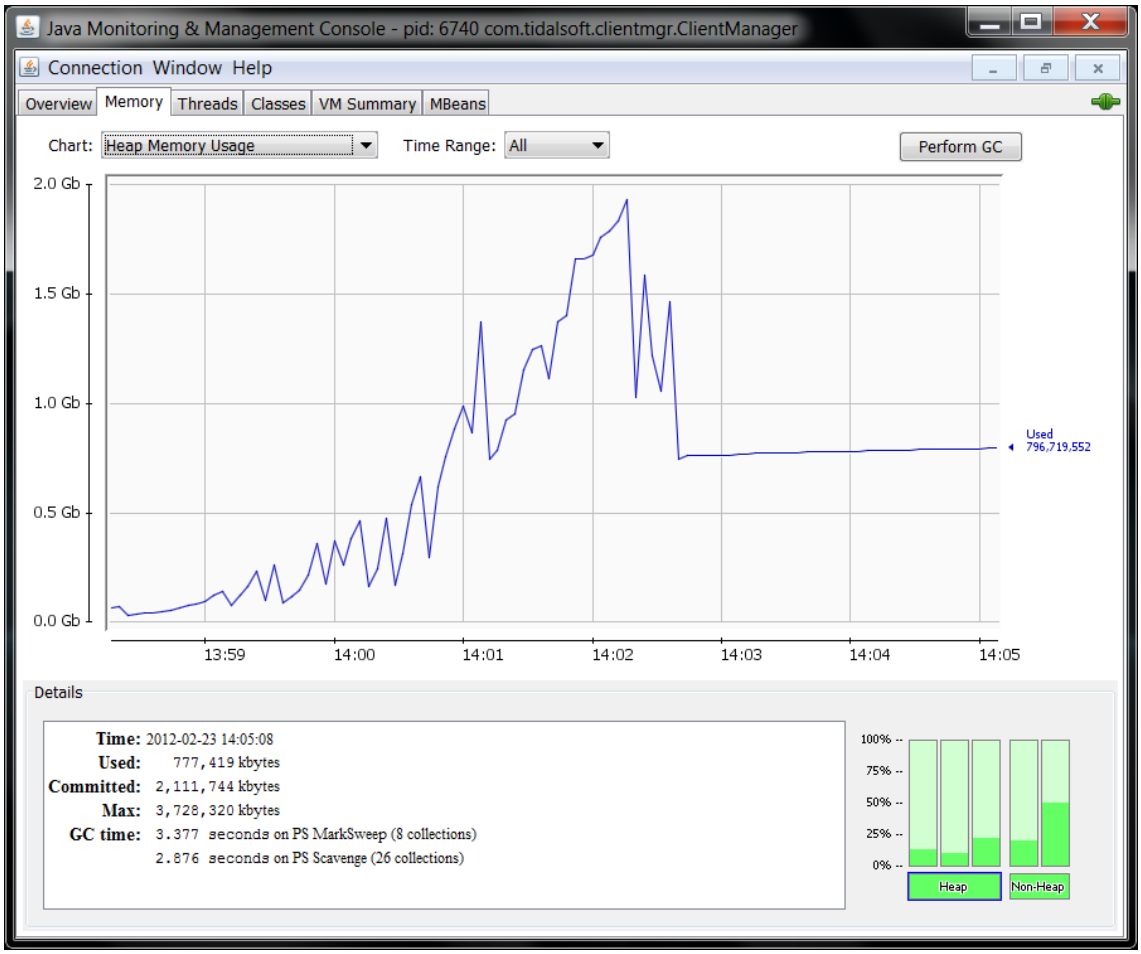
Overview Screen

The first screen of JConsole shows an overview of the JVM's memory, threads and CPU usage. This screen is very helpful to get a snapshot of the JVM's performance.



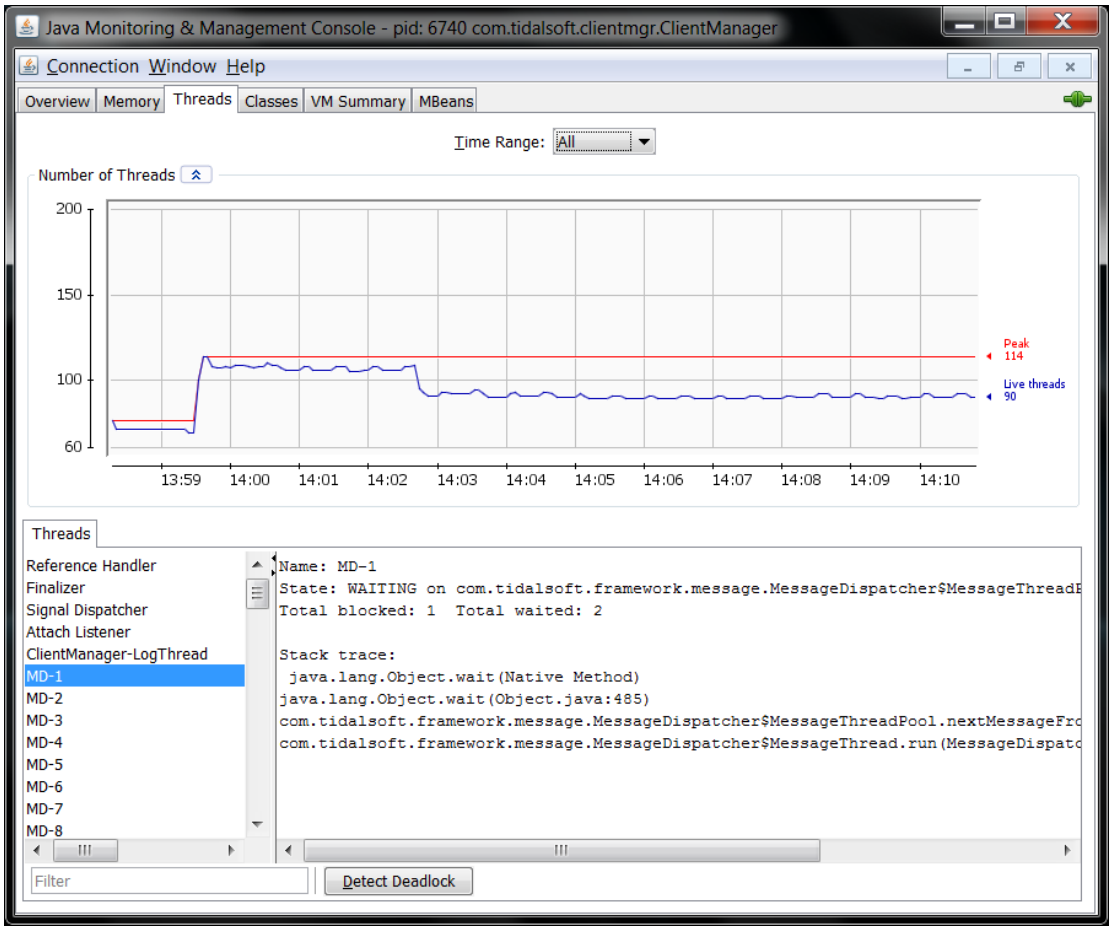
Memory Screen

The memory screen shows more detailed information about the JVM's memory use. For a normal running JVM, you should see memory use increase and decrease in the short term, but it should not increase in the long term. If it is increasing in the long term, it may indicate a memory that will eventually result in an out of memory termination. You can also use this screen to determine if the JVM does not have enough memory for the application that is running.



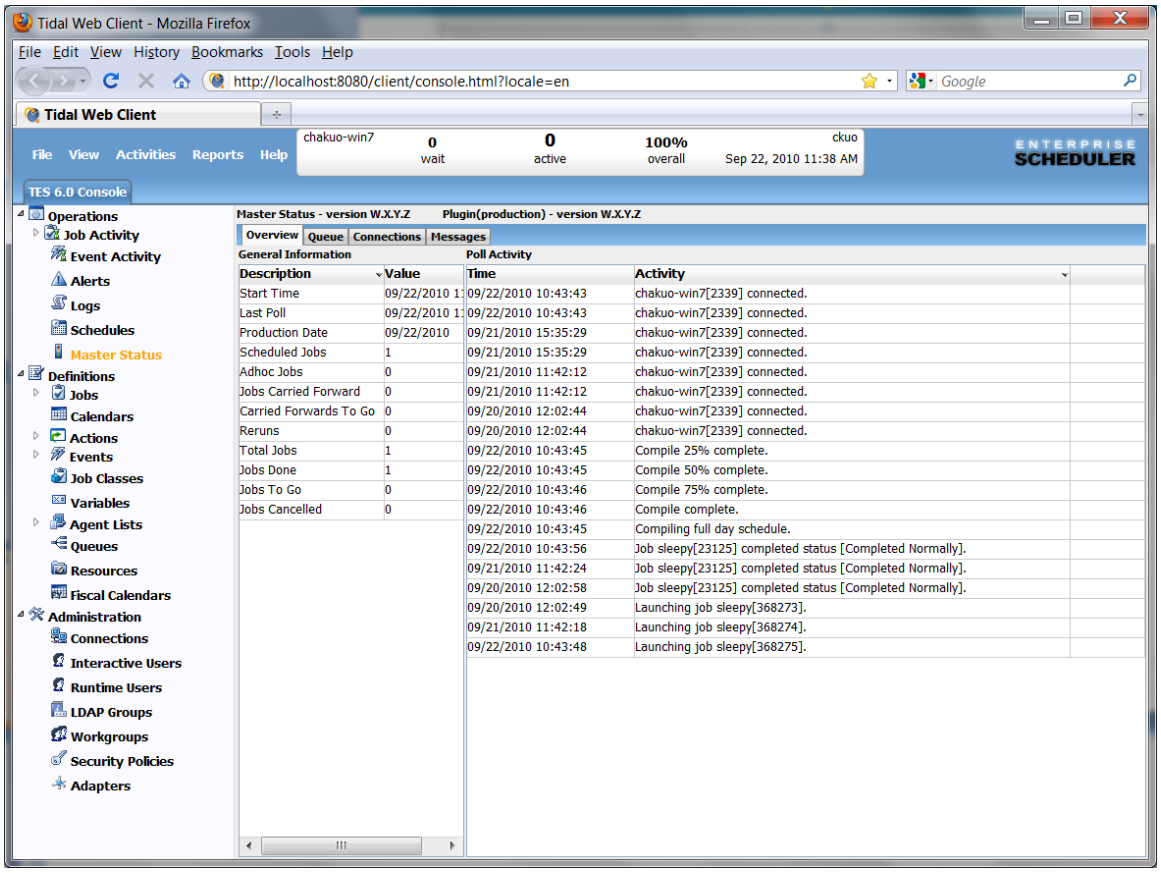
Thread Screen

This screen provides stack traces for every thread in the JVM. A stack trace will show exactly what a thread is doing at the time of the trace. This screen also has a function to automatically detect thread deadlocks.



CM and Master Versions

After applying a Master or CM patch, you usually check the corresponding log (master.log or DSP log) to make sure the right versions are applied. You can also check the versions in the web client as shown. While its ok to have separate versions of the Master and CM, it's advised to have the latest versions of each installed.



Installing Client Patch

Patching the client involves the following steps.

1. Stop ClientManager service.
2. Copy **tes-6.0.0.0.jar** and **tes-6.0.0.0.war** to **cache/tes-6.0.0.0**.
3. Copy **client.war** to **webapps**.
4. Delete contents of **webapps** except for **client.war**.
5. Delete contents of **plugins** if you need to destroy the cache.
6. Restart ClientManager service.
7. Delete the browser cache.
8. Connect browser to CM once CM completes secondary sync.

Performance Tuning

In order to fully utilize the hardware (primarily CPU and memory) available to the CM and Master, both need to be tuned with the following parameters.

tes-6.0.0.0.dsp

CacheSynchronizer.NumThreads - number of threads used to synchronize the cache on startup. These threads run in parallel so having more threads usually

results in faster sync times. You should normally set the number of threads to the number of CPUs that are available on the CM machine.
DataCache.ReadConnectionsMin - minimum number of stored JDBC connections used to read from the cache. Having a minimum number of connections available saves the DSP from having to re-create a connection every time it needs to read from the cache.
DataCache.ReadConnectionsMax - maximum number of stored connections used to read from the cache.
DataCache.WriteConnectionsMin - minimum number of stored connections used to write to the cache. Having a minimum number of connections available saves the DSP from having to re-create a connection every time it needs to write to the cache.
DataCache.WriteConnectionsMax - maximum number of stored JDBC connections used to write to the cache.
DataCache.PageCacheSize - number of data cache pages (usually 4KB chunks of data) that the DSP stores in memory. Having more stored pages increases the read performance of the cache at the expense of memory usage.
DataCache.ConnectionPoolMinSize - minimum number of stored JDBC connections used for both reading and writing the cache. Note: this is different from the other min/max connection settings above.
DataCache.ConnectionPoolMaxSize - maximum number of stored JDBC connections used for both reading and writing the cache. Note: this is different from the other min/max connection settings above.
DataCache.StatementCacheSize - number of JDBC statements the DSP stores for querying the cache. A larger statement cache provides better performance but will increase the permanent generation size for the CM process (MaxPermSize JVM setting).
ClientNode.MinSessionPoolSize - minimum number of stored communication sessions the DSP uses to communicate with the Master. Having a minimum number of sessions available saves the DSP from having to re-create a session every time it needs to communicate with the Master.
ClientNode.MaxSessionPoolSize - maximum number of stored communication sessions the DSP uses to communicate with the Master.
ClientNode.MaxConcurrentMessages - maximum number of concurrent Master communication messages each DSP session can store before another session is allocated. Having more sessions increases communication throughput at the expense of memory and CPU usage.

clientmgr.props
JVMARGS - parameters passed directly to the JVM
ClientSession.MinSessionPoolSize - minimum number of stored communication sessions the CM uses to communicate with web clients. Having a minimum number of sessions available saves the CM from having to re-create a session every time it needs to communicate with a web client.
ClientSession.MaxSessionPoolSize - maximum number of stored communication sessions the CM uses to communicate with web clients.

ClientSession.MaxConcurrentMessages - maximum number of concurrent web client communication messages each CM session can store before another session is allocated. Having more sessions increases communication throughput at the expense of memory and CPU usage.
DataSource.MinSessionPoolSize - minimum number of stored communication sessions the CM uses to communicate with DSPs. Having a minimum number of sessions available saves the CM from having to re-create a session every time it needs to communicate with a DSP.
DataSource.MaxSessionPoolSize - maximum number of stored communication sessions the CM uses to communicate with DSPs.
DataSource.MaxConcurrentMessages - maximum number of concurrent DSP communication messages each CM session can store before another session is allocated. Having more sessions increases communication throughput at the expense of memory and CPU usage.

master.props
MessageBroker.MemoryLimit - how much memory (in MB) the Master allocates for communication with CMs. Note: since the Master hosts the communication "broker" this only needs to be set on the Master.
MessageBroker.StoreLimit - how much disk space (in MB) the Master allocates for storing unprocessed communication with CMs. Note: since the Master hosts the communication "broker" this only needs to be set on the Master.
ClientConnection.MinSessionPoolSize - minimum number of stored communication sessions the Master uses to communicate with CMs. Having a minimum number of sessions available saves the Master from having to re-create a session every time it needs to communicate with a CM.
ClientConnection.MaxSessionPoolSize - maximum number of stored communication sessions the Master uses to communicate with CMs.
ClientConnection.MaxConcurrentMessages - maximum number of concurrent CM communication messages each Master session can store before another session is allocated. Having more sessions increases communication throughput at the expense of memory and CPU usage.

The following are a set of recommended settings for small, medium and large server configurations.

Small Configuration
8GB Memory, 2 Core, 5-10 concurrent users, < 4GB database

tes-6.0.0.0.dsp
CacheSynchronizer.NumThreads=2
DataCache.ReadConnectionsMin=5
DataCache.ReadConnectionsMax=10
DataCache.WriteConnectionsMin=5
DataCache.WriteConnectionsMax=10
DataCache.PageCacheSize=16384

DataCache.ConnectionPoolMinSize=5
DataCache.ConnectionPoolMaxSize=10
DataCache.StatementCacheSize=750
ClientNode.MinSessionPoolSize=5
ClientNode.MaxSessionPoolSize=10
ClientNode.MaxConcurrentMessages=10

clientmgr.props
JVMARGS=-Xms4096m -Xmx4096m -XX:PermSize=128m -XX:MaxPermSize=128m
ClientSession.MinSessionPoolSize=5
ClientSession.MaxSessionPoolSize=10
ClientSession.MaxConcurrentMessages=10
DataSource.MinSessionPoolSize=5
DataSource.MaxSessionPoolSize=10
DataSource.MaxConcurrentMessages=10

master.props
MessageBroker.MemoryLimit=256
MessageBroker.StoreLimit=4096
ClientConnection.MinSessionPoolSize=5
ClientConnection.MaxSessionPoolSize=10
ClientConnection.MaxConcurrentMessages=10

Medium Configuration

16GB Memory, 4 Core, 10-20 concurrent users, < 16GB database

tes-6.0.0.0.dsp
CacheSynchronizer.NumThreads=4
DataCache.ReadConnectionsMin=10
DataCache.ReadConnectionsMax=20
DataCache.WriteConnectionsMin=10
DataCache.WriteConnectionsMax=20
DataCache.PageCacheSize=131072
DataCache.ConnectionPoolMinSize=10
DataCache.ConnectionPoolMaxSize=20
DataCache.StatementCacheSize=1500
ClientNode.MinSessionPoolSize=10
ClientNode.MaxSessionPoolSize=20
ClientNode.MaxConcurrentMessages=10

clientmgr.props
JVMARGS=-Xms8192m -Xmx8192m -XX:PermSize=256m -XX:MaxPermSize=256m
ClientSession.MinSessionPoolSize=10
ClientSession.MaxSessionPoolSize=20
ClientSession.MaxConcurrentMessages=10
DataSource.MinSessionPoolSize=10

DataSource.MaxSessionPoolSize=20
DataSource.MaxConcurrentMessages=10

master.props
MessageBroker.MemoryLimit=512
MessageBroker.StoreLimit=16384
ClientConnection.MinSessionPoolSize=10
ClientConnection.MaxSessionPoolSize=20
ClientConnection.MaxConcurrentMessages=10

Large Deployment

32GB Memory, 8 Core, 50-100 concurrent users, > 32GB database

tes-6.0.0.0.dsp
CacheSynchronizer.NumThreads=8
DataCache.ReadConnectionsMin=50
DataCache.ReadConnectionsMax=100
DataCache.WriteConnectionsMin=50
DataCache.WriteConnectionsMax=100
DataCache.PageCacheSize=1048576
DataCache.ConnectionPoolMinSize=20
DataCache.ConnectionPoolMaxSize=40
DataCache.StatementCacheSize=7500
ClientNode.MinSessionPoolSize=50
ClientNode.MaxSessionPoolSize=100
ClientNode.MaxConcurrentMessages=10

clientmgr.props
JVMARGS=-Xms24576m -Xmx24576m -XX:PermSize=512m -
XX:MaxPermSize=512m
ClientSession.MinSessionPoolSize=50
ClientSession.MaxSessionPoolSize=100
ClientSession.MaxConcurrentMessages=10
DataSource.MinSessionPoolSize=50
DataSource.MaxSessionPoolSize=100
DataSource.MaxConcurrentMessages=10

master.props
MessageBroker.MemoryLimit=1024
MessageBroker.StoreLimit=65536
ClientConnection.MinSessionPoolSize=50
ClientConnection.MaxSessionPoolSize=100
ClientConnection.MaxConcurrentMessages=10

Other Settings

There are several other settings that may be useful in debug situations. The following settings are configured in the DSP props file.

SyncCache

This setting controls whether the DSP will synchronize the data cache on startup. It may be set to "N" if either the primary or secondary sync is causing a major performance problem. Note: this setting should normally be set to "Y" otherwise the data cache will not have the latest data from the Master.

CacheSynchronizer.Purge

This setting controls whether phantom records are purged from the cache during the primary and secondary syncs. Phantom records are left in cache if corresponding records re deleted on the Master while the Master and CM are disconnected. This setting should be turned on if the purge is causing a performance problem. Note: this setting should normally be removed or set to "Y" or else users may see phantom data that no longer exists on the Master.

CacheSynchronizer.StreamCommitSize

When the DSP is synchronizing data from the Master, it writes the data to the cache in batches. This setting controls how many records are in one batch. Writing many records in a batch is more efficient than writing a record at a time. Note, there is a tradeoff between memory usage and efficiency when increasing the batch size. The default batch size is 1000.