Oracle® Enterprise Manager

Application Configuration Console Performance and Tuning Guide

Release 5.3.2

E14655-02

September 2009

This document offers advice from a number of perspectives on tuning Application Configuration Console and third-party software to improve performance and efficiency. The advice runs the gamut from adjusting configuration settings to introducing more hardware.

This document contains the following sections:

1 Introduction

Application Configuration Console operates in a multi-tiered architecture, which means there are lots of potential flash points where performance can be impacted. Understanding this architecture can facilitate "knowing where to look" in terms of troubleshooting problems and resolving bottlenecks.

End users access Application Configuration Console through a rich, Eclipse-based client or a Web-based reporting and dashboard client. These clients communicate with the Application Configuration Console Server through remote procedure calls (RPC) over HTTPS (XML-RPC). The Application Configuration Console Client is installed on each user's machine.

With the reporting and dashboard client, users can access from a Web browser real-time information about managed configuration information and activity in Application Configuration Console.

The Application Configuration Console server runs on a Tomcat server to handle client requests and relies on two repositories for managing the data: a relational database to manage the Application Configuration Console metadata and a version control repository to manage the versioning history of the configurations managed by Application Configuration Console.

2 Tracking and Comparisons

Tracking and compare operations, by their very nature, place a high demand on the system. Careful planning attendant to the setup, configuration, and execution of these tasks can greatly reduce the demand and go a long way toward preventing server overload.

This section outlines several strategies to consider in the use of tracking and compare operations.

2.1 What's the Right Data Model?

If you tend to run tracking and compare activities at the asset level (the default Resource View), then tracking or comparing assets with large numbers of configurations will stress the system. The solution is to break assets down into logical subsets of configuration files by type, for example, operating system, application, Web tier, so that you wind up with a greater number of smaller assets on which you perform these activities.

On a per-configuration basis, other factors besides the type of operation itself can drive up the cost. The size of the files matters as does the type of file involved in the operation. For example, comparing two 100K files consumes more memory and CPU time than comparing two 1K files. Or, comparing XML-mapped files takes more resources than comparing Java property-mapped files, all else being equal. The objective, then, is to carefully consider which configurations to track or compare, especially when they are large and complex.

2.2 Refine the Configuration List

To optimize tracking and compare operations, determine precisely which configurations need to be involved and then build a resource specification around this set of files. Avoid use of wildcards with directories unless you determine that all configurations are deemed important.

If your particular environment doesn't lend itself to this approach, use asset views to cull subsets of configurations within assets that form logical groupings to be tracked or compared. An asset view is just that, a way to look at a subset of configurations within an asset that are somehow related or of particular interest to specific groups of users.

By example, consider the case where you have two assets, A and B, to be compared. Each asset contains 500 configurations, out of which there are 30 configurations that are significant to the comparison. For each asset you create an asset view, A1 and B1 respectively, that identifies the 30 configurations of interest. You then set up the comparison to run at the asset view level, comparing A1 to B1.

To learn more about asset views, refer to the *Application Configuration Console Client Online Help*.

2.3 Schedule Tracking and Compare Operations Off-Peak

The heading pretty much says it all. By running tracking jobs overnight, you can free up resources during normal business hours to perform day-to-day tasks such as find and replace and asset loading more efficiently.

Scheduling comparisons off-peak, at least those known to take a long time, has the same net effect: improved performance during the workday. And the resulting reports will be there for review the next day.

If you are scheduling multiple events off-peak, you may want to stagger them to diffuse the impact on resources.

2.4 Use Comparison Keys and mvPath Evaluation

Application Configuration Console performs comparisons based first on name and then on position of elements within containers. When identical names are used within configurations and the positional alignment is consistent, comparisons can do what they are supposed to do, which is to detect differences in property values. But we don't live in a perfect world, so names and positions often differ across configurations, which can result in "false positives" in detecting differences. WebLogic and WebSphere configurations, in particular, are susceptible to "noise" that gets in the way of meaningful comparisons.

To get around this issue, you can set metadata in the form of comparison keys, which afford a way to bridge naming and positional differences for the purposes of comparing values. To learn more about comparison keys and how they are used in this regard, refer to the *Application Configuration Console Client Online Help*, in particular, see the Use Alternate Comparison Strategies topic.

Useful as comparison keys are, setting metadata on configurations can be laborious. The metadata is also ephemeral, as it is overwritten on an update operation. To automate the process of setting metadata, as when applying comparison keys, you can use a save specification (saveSpecEntry). A saveSpecEntry defines a series of rules to be applied when an asset is loaded or updated. In this case, the saveSpecEntry sets metadata on the configurations to a type of comparison key that derives its value by evaluating an mvPath expression. To learn more about save specifications and how they are used in this regard, refer to the *Application Configuration Console Client Online Help*, in particular, see the Comparison Keys and the Save Specification Registry: a Use Case topic.

Note, however, that mvPath expression evaluation can be time-consuming if the construct is too general. Consider the following two expressions:Note, however, that mvPath expression evaluation can be time-consuming if the construct is too general. Consider the following two expressions:

- Resolve a single known configuration (`config.xml`) for a particular property name (`ID`) located under an element of type `Server`.
- Resolve all known configurations within Application Configuration Console, and all of their corresponding elements, for a particular property name (`ID`).

The first expression will quite obviously resolve much faster. So the rule of thumb is to be as specific as possible when using mvPath expressions in comparisons.

2.5 Beware Timeouts During Interactive Compares

The Application Configuration Console Client has a hardcoded limit of 20 minutes to execute and complete an interactive compare operation, that is, one that is not scheduled to execute at a later time. If the operation times out, the results are lost. If you follow the strategies enumerated in this chapter, you should see a noticeable reduction in execution time. Any comparisons that exceed a few minutes of execution time should be re-evaluated for efficiency.

Note: Scheduled comparisons (and tracking events) do not time out; they will run to completion, barring any processing abnormality.

2.6 Consider Tracking Server Redeployment

An Application Configuration Console installation creates separate deployments of its Server, the Web Reports server, and a tracking server. This facilitates server redeployment to another instance of a Tomcat JVM.

If scheduling tracking operations during off-peak hours is not an option, an alternative remedy is to redeploy the tracking server to another JVM instance. If your site makes heavy use of Web Reports, overall performance can benefit from redeploying the Web Reports server as well.

See the *Application Configuration Console Installation Guide* for information on redeploying the tracking and Web Reports servers.

3 Load Balancing

Organizations that use Application Configuration Console extensively may want to distribute the workload across multiple host machines. With this in mind, the software is packaged in such a way that you can effectively accomplish functional scaling by moving (redeploying) certain components to discrete JVM instances on separate host machines.

An Application Configuration Console installation creates separate server deployments as follows:

- Core Server – Performs activities such as asset loading, comparison generation, and response to Client requests. This is the primary server, the core of Application Configuration Console.
- Tracking server – Executes tracking operations against external resources based on job schedules. This is a secondary server.
- Web Reports server – Provides Web-based report generation against Application Configuration Console data. This is a secondary server.

The deployments are set up as separate directories on a single Tomcat instance in the following location:

```
$OACC_INSTALL/appserver/tomcat/webapps/mvserver|mvtrack|mvwebreports
```

There is no particular necessity for server redeployment, but you may be a candidate if some or all of the following conditions apply at your site:

- Tracking is a high-volume activity.
- Users actively generate and view reports.
- There's an overall high level of activity on Application Configuration Console's Core Server.

See the *Application Configuration Console Installation Guide* for information on server redeployment.

4 Database Tuning

The Oracle RDBMS is inherently complex, and Application Configuration Console is a database-intensive application. This combination demands that your DBA proactively monitor and analyze all aspects of database performance. There are literally hundreds of potential causes of performance problems with the Oracle database. Oracle has identified several areas that should be examined and addressed, as necessary. Perform these tasks in the order of appearance within this chapter. First and foremost, though, Oracle recommends that Application Configuration Console's Core Server and the Oracle database reside on separate hosts.

4.1 Generate Statistics on a Regular Basis

Generate statistics periodically, at least weekly. This not only provides usage information for analysis, but also enables the database to optimize access plans, which can speed up operations considerably.

To generate statistics:

1. Connect to the database as the oaccuser and execute the following command:

```
exec dbms_stats.gather_schema_stats(ownname => 'OACCUSER', options =>
'GATHER AUTO');
```

2. Stop and restart the Core Server.

4.2 Identify and Address Resource Bottlenecks

Resource bottlenecks such as CPU overload, excessive memory swapping, and disk I/O, can often be reduced or eliminated by introducing additional or more powerful hardware. Take, for example, disk I/O, which, incidentally, is the single largest aspect of Oracle response time. If this is identified as a bottleneck, adding a separate drive dedicated to the redo logs will have a significant positive impact on performance.

4.3 Tune the Database Instance

Take these steps to tune the database instance:

- Set the SGA size (on 32-bit Oracle instances) to the maximum configurable value of 1.7GB.
- Increase the JServer (Oracle JVM) pool size to at least 256MB, and preferably to 512MB. For example:

  ```
  alter system set JAVA_POOL_SIZE=512M
  ```

  Note that an out-of-the-box 10g instance has an unacceptably low (32MB) pool size.
- Create indexes as necessary on specific tables.

5 Tomcat/JVM Tuning

Application Configuration Console is a Java-based application that is deployed to an Apache Tomcat instance and executed within a virtual machine (JVM). This chapter covers some general tuning considerations both for Tomcat and for the JVM. A wealth of information about these technologies exists on the Web, a Google search away. Or simply go to the Sun Microsystems Java website (`http://java.sun.com`), where you can find documentation such as the following:

```
http://java.sun.com/performance/reference/whitepapers/tuning.html
```

5.1 Tomcat Tuning Recommendations

There are two areas where you can tweak settings to improve performance on Tomcat:

- JVM threads to support concurrent activities
- Connection pools to support database access

5.1.1 JVM Threads

Think of threads as workers within the JVM. The more threads available, the more work can be done simultaneously. Ensure that there are enough threads available to handle incoming requests. Depending on your findings, you may want to increase the thread settings in server.xml in the following directory:

```
$OACC_INSTALL/appserver/tomcat/conf/
```

Specifically, locate the Connector objects for port 9980 and for port 9943 and change the values of the maxThreads, minSpareThreads, and maxSpareThreads properties. Also, consider changing the values of the like-named properties for the HTTP or HTTPS Connector in the same file.

Note: Remember, this is a tuning exercise; do not increase the values indiscriminately. Threads consume memory and CPU. Too many idle threads within a JVM can actually degrade performance.

5.1.2 Connection Pools

Tomcat creates and maintains a pool of connections to the database for each server deployment; that is, for the Core Server, the tracking server, and the Web Reports server. You can modify connection pool size for any deployment by changing the value of the maxActive property in the respective configuration file (mvserver, mvtrack, mvwebreports) located in the following directory:

```
$OACC_INSTALL/appserver/tomcat/conf/Catalina/localhost
```

Note that you should change this property value only if your analysis substantiates a deficiency in connection pool availability.

5.2 JVM Tuning Recommendations

Heap refers to the amount of memory available to the JVM. Generally speaking, you can never have too much heap. This section describes how to increase the heap setting for Application Configuration Console's Core Server and for the Client.

5.2.1 Application Configuration Console Core Server Heap

Application Configuration Console's Core Server is initially configured to use 1024MB of heap, which is sufficient for handling relatively small amounts of data. Oracle recommends, though, that this be doubled to 2048MB, provided the operating system has the additional memory available. To increase the heap size:

1. Navigate to the following directory:

   ```
   $OACC_INSTALL/appserver/tomcat/bin
   ```

2. Open the `setenv.bat` or setenv.sh file for editing, as appropriate to your platform.
3. Set these Java options as follows:

   ```
   JAVA_OPTS=-Xms2048m -Xmx2048m
   ```

4. Save your edits.

The next time you restart the Core Server the increased heap size will be in effect.

5.2.2 Application Configuration Console Client Heap

An Application Configuration Console Client installation is configured to run with a heap size of 512MB, which is sufficient for handling relatively small amounts of data. Oracle recommends, though, that this be doubled to 1024MB, provided the operating system has the additional memory available. You will definitely want to do this if the amount of data to be loaded exceeds 10,000 configurations. To increase the heap size:

1. Navigate to the following directory:

   ```
   $OACC_INSTALL/eclipse
   ```

2. Open the start.bat or start.sh file for editing, as appropriate to your platform.
3. Set these Java options as follows:

   ```
   -vmargs -Xms1024m -Xmx1024m
   ```

4. Save your edits.

The next time you restart the Client the increased heap size will be in effect.

6 LDAP

Application Configuration Console issues search filters and bind operations against a V3-compliant LDAP to authenticate users. In cases where authentication seems to take an unusually long time, investigate the following:

- Check for precision in Group and User DN values. Imprecision can lead to unnecessary searches down leaves of the LDAP tree.
- Ensure that the LDAP server has sufficient resources. Have an LDAP administrator check server load, examine logs, and so forth.

7 System Maintenance

This section discusses various processes for maintaining Application Configuration Console.

7.1 Reducing the Size of the SVN Repository

Subversion (SVN), the version control system that Application Configuration Console uses, can grow exponentially over time, given the precision that Application Configuration Console applies to operational backups. While a fresh installation of Oracle requires approximately 400MB of disk space, it's not unusual for an active, high-volume environment to grow to 50GB or more. With such an imposing foot print, Oracle recognizes the need for a recommended method of pruning back the SVN repository to a manageable size, for performance as well as storage reasons. The method can't simply remove old versions to conserve disk space, however; it also must seamlessly maintain the integrity of the database by resetting the version numbers of the retained history.

The process detailed in this section has the following objectives:

- Provide a convenient way of removing versions from the repository that are deemed expendable.
- Rebuild the repository, based on the version history that remains.
- Update the database to reflect the reconstituted version numbers.

7.1.1 Preparation

Take synchronized backups of the Application Configuration Console installation and the Oracle database.

1. Shut down Application Configuration Console. Oracle and the SVN service can continue to run.
2. Make a backup copy of the Application Configuration Console file system. For example:

   ```
   /opt/oracle/oacc/server/*
   ```

   The pruning script preserves a copy of the repository (`/opt/oracle/oacc/svn`), which you can delete, once completing the entire process (see "Update Version Numbers in the Database" on page -10).
3. Have the DBA back up the database according to the protocol observed at your site.

Examining the historical record that constitutes the repository, establish a cutoff point at which you want to purge all versions older than this marker. For example, if your site's policy is to maintain a six-month version history, find a version that is approximately six months old to be the new start of your version history.

Do a list of the repository's version history directory (default: `$OACC_INSTALL/svn/db/db/revs`). This will show versions starting at 0 and ranging up to *n*, where *n* is the latest version number. Locate the version number with a date that corresponds to the date you have chosen to be the new start of your version history. Make a note of this version number and also of the latest version number. You will provide these version numbers as arguments to the pruning script.

Note: The removal process requires scratch disk space approximately equal to the size of the existing repository. If your repository is 20GB, you will need to allocate temporarily 20GB of disk space for SVN pruning.

7.1.2 Prune the Repository

Before proceeding, note the following as regards the process described in this section:

- The instructions in this section observe the following convention: `$OACC_INSTALL` refers to the Core Server installed location, for example, `C:\Program Files\Oracle\oacc\server` on Windows or `opt/oracle/oacc/server` on Linux or UNIX.
- If the SVN binary directory location is other than the default (`$OACC_INSTALL/svn/bin`), you must update the `SVNLOCATION` variable in the `MVRemoveVersions` script accordingly.
- If the SVN repository location is other than the default (`$OACC_INSTALL/svn/db`), you must update the `DB_LOCATION` variable in the `MVRemoveVersions` script accordingly.

To prune the SVN repository:

1. Shut down the Core Server and any secondary servers (tracking and Web Reports), if applicable.
2. Open a shell and traverse to the following directory:

   ```
   $OACC_INSTALL/appserver/tomcat/shared/scripts
   ```

3. Execute the version removal script as appropriate to your platform. The script takes two arguments (the start and end range of versions to retain, where the end version must be the latest in the repository). For example:

   ```
   ./MVRemoveVersions.sh 100 200
   ./MVRemoveVersions.bat 100 200
   ```

   On UNIX systems, you may have to grant execute permissions on the script, for example:

   ```
   > chmod +x MVRemoveVersions.sh
   ```

   Note: Script execution takes approximately an hour for each 2000 versions retained. There is no progress indicator during script execution.

When you see the following message:

```
Complete. If there were no errors encountered in the execution, you should now
execute remove_versions.sql as the mValent Oracle user (default=OACCUSER).
```

Before updating the database, check the log file (located at: `$OACC_INSTALL/appserver/shared/scripts/MVRemoveVersions.log`) to ensure that there were no errors.

7.1.3 Update Version Numbers in the Database

After pruning the repository, update the database so it reflects the new version numbering.

1. Copy the `remove_versions.sql` script to a temporary directory on the Oracle server. (The script must be executed on the host where the database exists.)
2. Using the `su` command, change the login to the Oracle user. For example:

   ```
   su oracle
   ```

3. Using the Oracle username and password for the Application Configuration Console user (default: `OACCUSER`), open a SQL*Plus session and log in to the database. This step validates the username and password. For example:

   ```
   > sqlplus OACCUSER/OACCUSER@OACCSERV
   > exit
   ```

4. Open a shell and traverse to the temporary directory where you copied the script in Step 1.
5. Execute the remove_versions.sql script, using the same version range specified in the version removal script. For example:

   ```
   > sqlplus OACCUSER/OACCUSER@OACCSERV @remove_versions.sql 100 200
   ```

6. The script prompts for the Oracle SID and the Oracle username and password for the Application Configuration Console user (default: `OACCUSER`). Enter appropriate values.

When the script completes, you should see a reduction in the size of the repository (`$OACC_INSTALL/svn/db`) commensurate with the version history pruned.

Verify that you can start Application Configuration Console and that you can access version history in the Client. If everything appears to be working as expected, you can delete the dated copy of the original repository that the script has preserved at `$OACC_INSTALL/svn/db_$DATE`, where `$DATE` is a numerical datestamp string equivalent to the date and time of script execution.

8 Additional Considerations

This section contains a number of miscellaneous recommendations that would expectedly contribute to improved performance of Application Configuration Console.

8.1 Take the Upgrade Path

Always upgrade to the latest release of Application Configuration Console, as it contains the latest in performance enhancements. For example, the 5.3.2 release is certified on JDK 1.6, which has measurable performance improvements, especially in the area of garbage collection and memory management, as compared to JDK 1.5. Additionally, the 5.3.2 release of Application Configuration Console includes several specific code-based performance enhancements.

8.2 2 bits? 4 bits? No, Make It 64 bits

Oracle recommends that you run on 64-bit hardware, operating system, and JVM when possible, to optimize performance and overall throughput. Perhaps even more important is that you host the database on a 64-bit machine, as that is where you are likely to encounter bottlenecks.

8.3 Beyond Server Redeployment

Section 3 talks about server redeployment as an antidote to busy multiuser environments with large amounts of data, and high demand on tracking, compare, and reporting operations. If the sky's the limit on resources, you might want to consider as an alternative, the ultimate in load balancing: separate Application Configuration Console instances that access unique database instances. For example, one approach might be a division of configurations as follows:

- Application Configuration Console Instance A – UNIX/Linux operating system configurations
- Application Configuration Console Instance B – Application configurations
- Application Configuration Console Instance C – Windows operating system configurations

If you go this route, it's important that you understand the tradeoffs. Yes, you gain the obvious advantages that this sort of load balancing affords. But understand that these are truly separate and distinct instances. This means you can't do comparisons across instances or generate reports against multiple instances. In short, you cannot merge data from two or more instances of Application Configuration Console. If these terms are acceptable to you, then separate instances might be something you want to consider.

8.4 If It's Not Being Used, Get Rid Of It

There's a certain overhead to maintaining everything that has ever been created in Application Configuration Console. It's advisable, therefore, to periodically do some housecleaning to eliminate things that are no longer valid or in use.

- Delete assets, hosts, authentication packs, and resource specifications that fall in to this category.
- Remove any asset views that are not in use. In particular, research default asset views created through the WebSphere or WebLogic Automation Modules, as they sometimes redundantly define configuration data. Any asset views that fall in to this category should be removed.

8.5 When It Comes to Assets, Smaller Is Better

Section 2 suggests strategies for optimizing performance in tracking and compare operations. For example, break large assets into logical subsets of configurations by type; use asset views to form logical groupings of configurations within assets to be tracked or compared. Beyond guidelines such as these, Oracle recommends as best practice that smaller assets are better, and that, in any case, the upper range should not exceed 500 configurations. This should serve as a hedge against performance degradation, not only during Application Configuration Console operations (compare, update, synchronize), but at asset load time as well.

9 Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html, and a list of phone numbers is available at http://www.fcc.gov/cgb/dro/trsphonebk.html.

Application Configuration Console Performance and Tuning Guide, Release 5.3.2
E14655-02