IBM Tivoli Identity Manager 5.x

API Performance Tuning Guide

Issue Date:
2013 May 16 – Third Edition

Publication Number:
SC23-8711-01

Table of Contents

About this guide

This guide identifies best practices for application developers when using the IBM® Tivoli Identity Manager™ application program interface (API).

Who should use this guide

Use this guide if you are responsible for using the IBM Tivoli Identity Manager system APIs. The following competencies are recommended:

- Familiarity with basic database and directory design principles.
- General knowledge of the Java programming language.

Contributors

- Daniel Nieman
- Tommy Nguyen
- Barry Evans
- Nnaemeka Emejulu
- Gowree Venkatachalam

Introduction

The IBM Tivoli Identity Manager product provides an API that customers often use to integrate Tivoli Identity Manager functionality into existing processes or customer interfaces. As IBM Tivoli Identity Manager deployments grow in size, scalability and performance become more crucial for customer satisfaction. The API allows great flexibility for accessing and manipulating information. This flexibility can often result in developers using the API in ways that yield poor performance or in ways that do not scale. This document describes some best practices on how to use the API while maximizing performance and scalability.

All tips in this guide are applicable to IBM Tivoli Identity Manager servers at V4.5.1 and later unless otherwise specified.

This is a supplement, not a replacement, to the IBM Tivoli Identity Manager API documentation shipped with the product.

This document is a working document. As more information is gathered, settings will be added, removed or changed in future editions. Check the IBM Web site for the most recent version at http://www-1.ibm.com/support/docview.wss?uid=swg27009686.

Local and remote functions

The IBM Tivoli Identity Manger provides two classes of API functions: local and remote. Local API functions are those designed to be used from within the Tivoli Identity Manager product, such as within custom workflow extension or a javascript extension. Remote API functions are designed to be called from an external process, such as a custom interface or abstraction layer.

Tips for remote APIs

The following tips apply only to remote API calls.

Avoid getResults() for large result sets

When using SearchResultsMO (or SearchMO.execute()) for large result sets, avoid using getResults(). Instead, get the results one page at a time using getPage() with a reasonable pageSize on the SearchMO object.

Impact

Returning the results from the SearchMO object in batches will reduce the amount of memory required in the Tivoli Identity Manager JVM thereby preventing an out of memory (OOM) error. Fetching the data one page at a time ensures that manageable chunks of data are transmitted from the server to the client.

Increasing the pageSize in the SearchMO object will improve throughput at the cost of more memory usage on the server side and larger serialized objects.

Example

```
SearchMO searchMO = new SearchMO(platform, subject);
searchMO.setPageSize(100);

SearchResultsMO res = SearchMO.execute();
int pageNumber = 0;
Collection localResults = new ArrayList();
Collection page = res.getPage(++pageNumber);
while(page != null && page.size() == 100) {
   localResults.addAll(page);
   page = res.getPage(++pageNumber);
}

if(page != null && pageNumber == 1) {
   localResults.addAll(page);
}
```

Use getPageCount(int) where possible

The SearchResultsMO.getPageCount() function is commonly used to return the total number of pages in the result set. When possible, callers should use the getPageCount(int lookaheadLimit) instead.

Impact

The getPageCount() function requires loading the entire result set, which takes time and server memory. Often, particularly for user interfaces, it is sufficient to see if there are more than a specific number of pages rather than needing the total number. In this situation, it is preferable to call the getPageCount(int lookaheadLimit) instead. This allows the server to load only the subset of data requested.

Example

```
SearchMO searchMO = new SearchMO(platform, subject);
searchMO.setPageSize(100);

SearchResultsMO res = SearchMO.execute();

// in our UI, we want to show 2 pages and only load subsequent pages
// if requested by the user. To that end we'll request 3 pages, show
```

```
// 2 of them, and only show the 'next' button if getPageCount(3) returns 3
int pageCount = res.getPageCount(3);
```

Don't use 'new Date()' for scheduling arguments

Several AccountMO, PersonMO, and RoleMO functions accept a java.util.Date parameter used to schedule the action for some time in the future. If the desire is to have the operation start immediately, don't pass in a value of new Date(), use null instead.

Impact

Passing in a new Date() value has two downsides. The first is that if the API call is done on a remote system, the current system's date may not match up with the server's date due to being out of sync or timezone differences resulting in undesired behavior. The second, and more important, downside is that specifying a date instead of null results in a message for future activity being added to the SCHEDULED_MESSAGE table creating unnecessary work for ITIM and possibly resulting in lock contention for busy systems.

Example

```
// to have a person update happen immediately, use null as the second argument
personMO.update(person, null)

// ditto for account suspends
accountMO.suspend(null);

// and removing a member from a role
roleMO.removeMember(person, null);
```

Tips for local APIs

The following tips apply only to local API calls.

Avoid toCollection() and size() for large result sets

Avoid calling toCollection() or size() on a SearchResult. Instead, use an iterator to process each result entity.

Impact

Calling toCollection() on a SearchResult will cause the entire result set to reside in memory as a collection. Similarly, calling size() on a SearchResult will force traversal of the entire list.

Instead, it is a better practice to use an iterator to iterate through the results.

Example

```
SearchResults sr = null;
try {
   PersonSearch perSearch = new PersonSearch();
   sr = perSearch.searchByFilter()
   SearchResultsIterator e = sr.iterator();
   while(e.hasNext()) {
      PersonEntity entity = (PersonEntity) e.next();
      // code to process each entity would go here
      e.remove();   // remove it after being used to free up memory
   }
} finally {
   if (sr != null)
      sr.close();
}
```

Tips for both local and remote APIs

The following tips apply to both local and remote API calls.

Avoid using (objectclass=*) where possible

To return all objects from a search, use an empty filter instead of using (objectclass=*).

Impact

Filters used with AccountSearch, SearchMO, and related functions are often modified before being sent to the directory server. Specifying (objectclass=*) for a filter when all results are requested may result in a non-optimal filter being sent to the directory server. Instead, pass an empty string to the API.

Example - Remote API

```
/* Code to set max size, when there –
   A. Is a guarantee that the filter is bound to match a single unique entry
   B. Run an existence search only
*/
SearchMO searchMO = new SearchMO(platform, subject);
// Set sizeLimit to 2 for case A, to ensure no duplicate entities
// Set sizeLimit to 1 for case B, to check for consistence
searchMO.setMaxSize(2);

/* Set return attributes */
String [] retAttrs = new String[2];
retAttrs[0] = "cn";
retAttrs[1] = "title";
searchMO.setAttributes(retAttrs);
```

Example - Local API

```
/* Set search result limit */
SearchParameters params = new SearchParameters();
params.setSizeLimit(2);

/* Set return attributes */
ArrayList retAttrs = new ArrayList();
retAttrs.add("cn");
retAttrs.add("title");
params.setAttributes(retAttrs);
```

Specify return attributes and size limit for searches

When searching for an entity, explicitly specifying the desired attributes, specifying a limit on the number of entities returned, or both will improve performance.

Impact

By specifying a list of desired attributes, the directory server can minimize the amount of data returned decreasing the time required to return the results. Specifying an attribute list can also reduce the memory overhead of the resulting data set.

Setting a limit on the resulting result set can provide information to the directory server to allow it to optimize the search for the data as well as decreasing the memory overhead of undesired results.

Specifying a limit of 1 or 2 is useful for calls designed to check for the existence or uniqueness of an entity.

Example - Remote API

```
/* Code to set max size, when there –
   A. Is a guarantee that the filter is bound to match a single unique entry
   B. Run an existence search only
*/
SearchMO searchMO = new SearchMO(platform, subject);
// Set sizeLimit to 2 for case A, to ensure no duplicate entities
// Set sizeLimit to 1 for case B, to check for consistence
searchMO.setMaxSize(2);

/* Set return attributes */
String [] retAttrs = new String[2];
retAttrs[0] = "cn";
retAttrs[1] = "title";
searchMO.setAttributes(retAttrs);
```

Example - Local API

```
/* Set search result limit */
SearchParameters params = new SearchParameters();
params.setSizeLimit(2);

/* Set return attributes */
ArrayList retAttrs = new ArrayList();
retAttrs.add("cn");
retAttrs.add("title");
params.setAttributes(retAttrs);
```

Optimize SearchMO context

When using the SearchMO object, directory server performance can be improved by specifying an optimal logical context. The logical context, set with the SearchMO.setContext() function, specifies the base within the Organizational Tree. Whenever possible, specify the lowest organizational context that will return the desired result set rather than using the tenant context.

Impact

When a tenant-level logical context is specified and the recycle bin is enabled, the resulting LDAP filter to the directory server will include the filter (!(erIsDeleted=Y)) to remove any entries contained in the recycle bin. This filter results in poor performance of the directory server. By specifying an organizational-level logical context, this filter is not required and can yield better directory server performance.

In addition, by specifying an organizational-level context, the directory server can evaluate a smaller number of possible matches.

The recycle bin can be disabled in IBM Tivoli Identity Manager V4.5.1 IF 83 or higher and V4.6 IF 54 or higher. It is disabled by default in V5.0. Consult the Performance and Tuning Guide for more information on disabling the recycle bin.

Explanation

When a person search is done with a filter and a tenant logical context, the resulting call to the directory server will look something like:

Base1 – <*tenantDN*>

Filter1 – (&(<*user_specified_filter*>)(!(erisDeleted=Y)))

When the same filter is used with the organizational context, the resulting call is:

Base2 – ou=people, <*organizationDN*>

Filter2 – (<*user_specified_filter*>)

Filter2 will yield faster results than Filter1. Also, using the organization DN as the base narrows the search to a smaller set within the directory server.

Example - Remote API

```
// Code to get compound DN object – do this one time
String primaryOrgName  = "ACME";  // Primary Organization Name, usually same as tenantId
ContainerManager cm = new ContainerManager(platform, subject);
Collection containers = cm.getContainers("organization", primaryOrgName, null);
OrganizationalContainerMO orgMO =
   (OrganizationalContainerMO) containers.iterator().next();
// Create a compound DN with both the Tenant DN and the Org Unit DN
CompoundDN cdn = new CompoundDN(cm.getRoot().getDistinguishedName());
cdn.append(orgMo.getDistinguishedName());

// Code to set the context in SearchMO using the compound DN – do this as needed
searchMO searchMO = new SearchMO(platform, subject);
searchMO.setContext(cdn);  // Prevents '((erIsDeleted=y))' condition
searchMO.setScope(SearchMO.SUBTREE_SCOPE);  // Ensures subtree search
```

Example - Local API

```
CompounDN searchCtxt = new CompounDN(tenantDN);
searchCtxt.append(<ORG_DN>);
SearchParameters params = new SearchParameters();
params.setScope(SearchParameters.SUBTREE_SCOPE);
SearchResults searchRes =
   AccountSearch().searchByFilter(searchCtxt, profileName, filter, params);
```

Close SearchResults and SearchResultsMO

When processing search results using SearchResults or SearchResultsMO, always close them when the results are no longer needed, such as after the last page is accessed from a paged search or in a finally block.

Impact

Both functions internally cache the LDAP connection to the directory server. Failing to close the result set can lead to exhausting the available LDAP connections in addition to holding onto memory that would otherwise be freed.

Example - Remote API

```
SearchMO searchMO = new SearchMO(platform, subject);
// code to set up search goes here
SearchResultsMO searchRes = null;
try {
   searchRes = searchMO.execute();
   // code to process search results goes here
} finally {
   // close SearchResultsMO
   if(searchRes != null) {
   try {
      searchRes.close();
```

```
    } catch(Exception e) {}
}
```

Example - Local API

```
SearchResults searchRes = null;
try {
   searchRes =
     new AccountSearch().searchByFilter(searchCtxt, profileName, filter, params);
   // code to process search results goes here
} finally {
   // close SearchResults
   if(searchRes != null) {
   try {
      searchRes.close();
   } catch(Exception e) {}
}
```

Avoid using SearchMO.setSortAttribute() when possible

The SearchMO.setSortAttribute() method can be used to sort the search results by a specific attribute. Sorting the results places additional load on the LDAP server if server-side sorting is enabled or on the IBM Tivoli Identity Manager server if server-side sorting is disabled and should not be used if sorted results are not required.

Impact

If server-side sorting is enabled in the enRole.properties file, sorting even small result sets can result in poor LDAP server performance. If server-side sorting is disabled, sorting large result sets can be CPU and memory intensive on the IBM Tivoli Identity Manager server.

Other resources

You will find the following resources useful for further tuning of IBM Tivoli Identity Manager:

IBM Tivoli Identity Manager 5.x Performance Tuning Guide:

http://www-01.ibm.com/support/docview.wss?uid=swg27011444

IBM Tivoli Identity Manager 5.x Performance Tuning Scripts:

http://www-01.ibm.com/support/docview.wss?uid=swg27013557