Tuning the server

Java Virtual Machine (JVM) Tuning

Java version
Servoy supports both Java 5 and Java 6 (for the Application Server, the Smart Client and Servoy Developer). As Java 6 has numerous performance improvements, it is preferred over Java 5.

If the hardware in which Servoy is running is 64bit, make sure to also run a 64 bit Java Virtual Machine. When in doubt which version is used, check the Servoy Admin page, under "Servoy Server Home" > "Servoy Server Status" > "JVM Information":

- 64 bit: java.vm.name=Java HotSpot(TM) 64-Bit Server VM
- 32 bit: java.vm.name=Java HotSpot(TM) Client VM

An 32 bit JVM will allow a max memory of 2 Gb in total, In order to run with larger heap sizes (over 2 Gb) you have to use 64 bit JVM on a 64 bit OS

Java Virtual Machine Server mode
Java has 2 distinct operating modes: client mode and server mode. When running in server mode, the startup of the Java process will take longer and initial execution of code will be slower, but after a while the application will perform faster. The server mode is best suited for the Servoy Application Server.

Whether or not a Java Virtual Machine supports client and/or server mode depends on the hardware, see http://download.oracle.com/javase/6/docs/technotes/guides/vm/server-class.html.

If the JVM supports servermode, it will automatically detect if the machine it is running on is a "capable" machine and automatically run in server or client mode depending. The definition of "capable" may differ from Java version to Java version. For Java 5/6 this means a 2CPU, 2Gb memory machine.

Checking whether or not a Servoy Application Server runs in client or server mode can be seen on the Servoy Admin page, under "Servoy Server Home" > "Servoy Server Status" > "JVM Information":

- Server mode: java.vm.name=Java HotSpot(TM) 64-Bit **Server VM**
- Client mode: java.vm.name=Java HotSpot(TM) **Client VM**

It is possible to enforce the Servoy Application Server to run in server mode (assuming the JVM is capable to operate in server mode), an extra startup argument can be added to the startup of the Servoy Application Server.

In servoy_server.sh/bat:

```
java -server -Djava.awt.headless=true .....
```

When using the Service wrapper:

```
# Java Additional Parameters
wrapper.java.additional.1=-Djava.awt.headless=true
wrapper.java.additional.2=-Duser.dir="C:\Servoy\application_server"
wrapper.java.additional.3=-XX:MaxPermSize=128m
wrapper.java.additional.4=-server
```

On Windows 32 bit environments, the Java Runtime Environment does not include support for server mode. In order to take advantage of server mode optimizations on 32 bit Windows systems, it is required to install Java Development Kit (JDK)

Memory

See Memory Management

Database Connection Pool Tuning

See Connection Pooling in the Database Connections chapter

Database Tuning

Architecture

In order to get the best possible performance, the communication between the Servoy Application Server and the database needs to be as fast as possible. The fastest option would be installing the database engine and the Servoy Application Server on the same machine, whether or not this is possible depends on the dimensions of the physical server and the resource requirements of both the database engine and the Servoy Application Server.

When opting for dedicated servers for both the database engine and the Servoy Application Server, the connection between them should be as fast as possible, preferably a direct 1Gb cable connection between the two machines.

Database tuning

When experiencing performance issues the "Performance Data" page of the Servoy Admin page can be analysed to find slow performing SQL statements, or SLQ statements that are fired often.

The page provides an overview of all queries fired at the database. The performance data is sorted by the total time taken to perform the query, being the result of the average time * number of executions.

At any given time regular SQL statements (Type = Relation, Load foundset) should have an average execution time below 25 milliseconds.

Especially for SQL statements that are fired often and take up significant time it should be checked if it's possible to optimize the database, for example by creating the relevant indexes.

Web Client Settings Tuning

servoy.webclient.templates.use_local_ids

For the Web Client, Servoy generates HTML markup. In order for Servoy to dynamically update only parts of the UI through Ajax, each object in the HTML Markup receives a unique ID. This ID is also placed in the editable Web Client templates.

Servoy has 2 modes for generating the ID's:

- Using unique identifiers that are part of the internal design of the Solution created with Servoy. These ID's, as they are part of the solution design, are the same on any Servoy Application Server where the solution is imported.
- Using local identifiers. These identifiers are generated at runtime and thus differ per installation and session.

The advantage of the local identifiers are that they are shorter in length, thus bring down the weight of the generated HTML markup. The advantage of the unique identifiers is that they are the same on any server where the solution is running.

If templates are used to modify the generated HTML markup for Web Clients, then the unique identifiers are to be used, in order to maintain the link between the identifiers in the edited templates and the identifiers used at runtime. Otherwise the local identifiers can be used.

By default a Servoy Application Server uses the unique identifiers. Enabling the use of local identifiers can be done by setting the servoy.webclient.templates.use_local_ids option on the Servoy Admin page to true (located under Servoy Settings > Web Client settings)

Content compression

In order to minimize the size of the content that the browser has to download for the Web Client, it's possible to turn on compression. The Servoy Application Server is currently not configured to automatically compress the content.

Compression can be done by the Apache Tomcat instance that is part of the Servoy Application Server, or can be done by an HTTP server that is in front of the Servoy application Server, like an Apache HTTPD Server.

In order to enable compression in the Tomcat instance that is part of the Servoy Application Server, open the file "server.xml" located in {servoy_install} application_server\server\conf\. and locate the Connector:

```
<Connector port="8080"
  protocol="HTTP/1.1"
  maxThreads="500" connectionTimeout="60000"
  redirectPort="8443" useBodyEncodingForURI="true" />
```

and add the following atttributes:

- compressableMimeType="text/html,text/xml,text/css,text/javascript,text/plain"
- compression="on"
- compressionMinSize="2048"

so the connector is configured like this:

```
<Connector port="8080"
  protocol="HTTP/1.1"
  maxThreads="500" connectionTimeout="60000"
  redirectPort="8443" useBodyEncodingForURI="true"
  compressableMimeType="text/html,text/xml,text/css,text/javascript,text/plain" compression="on"
  compressionMinSize="2048"/>
```

Save the edits and restart the Servoy Application Server to let the configuration changes take effect.

As of Servoy 6.0, the Tomcat compression configuration will be enabled by default

It is also possible to handle the compression outside of the Servoy Application Server, for example in an Apache HTTPD Server placed between the Servoy Application Server and the outside world. The advantage of this is that Tomcat's compression options are limited, so an external HTTP server offers more options.

When it comes to Apache HTTPD server, there are two options:

1. Compression using mod_deflate
2. Compression using gzip

When it comes to compression, there are two factors that come into play, the time it takes to do the actual compression and the rate of compression achieved, resulting in quicker downloads to the browser due to smaller size. This is primary difference between mod_deflate and gzip in Apache HTTPD: while gzip gets a better compression rate than mod_deflate, it takes more time and CPU to perform the compression. In the case of Servoy Web Clients, that, due to it's Ajax driven design, performs a lot (albeit small) requests to the server, the overall performance will be better using mod_deflate.

Details on setting up mod_deflate on an Apache HTTPD server can be found here: http://httpd.apache.org/docs/2.0/mod/mod_deflate.html

Servoy Application Server log

At all times try to keep the Server log free from javascript application errors/excpetions, all application exceptions point to possible problems inside the application.

- Out of memory errors means the heap size is configured to small.
- Nullpointer errors indicate Servoy codebase problems, incase these are noticed file a support case containing the log entry.
- Network error are only listed as non critical (orange) which may point to (possible) networks (config) problems, when many errors are continuous listed these should definitely be investigated.