WHITESTEIN
Technologies

Living Systems® Process Suite

Performance Tuning

Living Systems Process Suite Documentation

3.3
Mon Nov 1 2021

Whitestein Technologies AG | Hinterbergstrasse 20 | CH-6330 Cham
Tel +41 44-256-5000 | Fax +41 44-256-5001 | http://www.whitestein.com

Contents

Chapter 1

Performance Tuning

1.1 System Performance

Perform general performance check:

1. Make sure you meet the hardware requirements.
2. Check if your operating system, Java, and browser are supported for your edition.
3. Check your operating system for insufficient resources.
4. Check the JVM memory consumption with Java VisualVM or JConsole.
5. Check compatibility and performance of the application server.
6. Check availability and performance of any external services.
7. Check memory consumption of your browser.
8. Check the database settings and structure, and trace and analyze SQL queries: to display SQL statements in the console set the following:
   - On the application server
     - On JBoss, enable in the profile XML (standalone-full.xml), add spy="true" to the LSPS datasource:

       ```
       <datasource jta="true" jndi-name="java:jboss/datasources/myDS" pool-name="myPool" enabled="true"
              use-java-context="true" spy="true" use-ccm="true">
       ```

       and in <subsystem xmlns="urn:jboss:domain:logging:<VERSION>"> add

       ```
       <logger category="jboss.jdbc.spy">
          <level name="TRACE"/>
       </logger>
       ```
   - On the database
     - For MS SQL, you can do so with *SQL Server Profiler*.

1.2 Performance of LSPS Server

Check your LSPS performance:

1. Check integration code:
   (a) Check integration to external data sources.
   (b) Check the authentication and authorization systems.
2. Check the Java implementation of any custom objects.
3. Check the LSPS Server:
   (a) Make sure there are not too many process and model instances communicating via signals.
   (b) Make sure there are not too many active users.
   (c) Check HealthCheck data of the LSPS Server for suspicious data such as:
      - long model instance wait times (AverageMILockWaitTime and AverageOfLastTenMILockWaitTimes)
      - many loops that exceed the threshold limits for engine loops (ThresholdNumberOfEngineLoops)
      - other statistics provided by the HealthCheck Monitoring
   (d) Check the CREATE_PROCESS_LOG server setting: recommended setting is MODULE. Further information about the setting, is available in the Deployment guide.
   (e) On WildFly and JBoss, configure bean pools: The configuration depends on the expected load as well as other performance parameters. You can change the configuration in the $AS_HOME/standalone/configuration/standalone-full.xml or the respective WildFly or JBoss config.
      - Limit the MDB pool and the default pool used for timers, async, and remoting.
      - Extend the SLSB pool to 10 ∗ (MDB pool size + default pool size) at least.
      - Adapt the acquisition timeouts to reasonable values to prevent long waiting deadlocks in case of lack of resources. Set the data-source connection pool maximum to MDB pool size + the default pool size at least, for example:

```
<bean-instance-pools>
  <strict-max-pool name="slsb-strict-max-pool" max-pool-size="200" instance-acquisitio
  <strict-max-pool name="mdb-strict-max-pool" max-pool-size="20" instance-acquisition-
~
<thread-pools>
  <thread-pool name="default">
    <max-threads count="10"/>
    ...
  </thread-pool>
</thread-pools>
```

1.2.1 Pre-Loading Modules

To compile modules on server restart and prevent their compiling later, you can pre-load the modules: On server start, the server will compile the modules defined by the INITIAL_MODELS_SQL entry in the LSPS_SETTINGS database table.

To enable and define pre-loading of modules when the server is restarted, do the following:

1. Set SQL logging level of Hibernate to ALL.
   For the embedded Wildfly server, add org.hibernate.SQL.level=ALL to <LSPS_WORKSPACE>\.LSPSEmbedded\wildfly<VERSION>\standalone\configuration\logging.properties.

2. Log in the LSPS database as administrator.

   On the default H2 database, the DB URL is `//localhost/h2/h2;MVCC=TRUE;LOCK_TIMEOUT=60000`, the user name and password are `lsps`.

3. Configure which modules should be compiled on server start to prevent compilation on first-user access: In the LSPS_SETTINGS table, add the INITIAL_MODELS_SQL key with the value set to the select statement that returns the MODEL_ID entries you want to pre-load.

```
--Inserts the INITIAL_MODELS_SQL key with the select that returns the IDs
--of the most recently uploaded my-process and my-contracts modules to LSPS_SETTINGS:
INSERT INTO LSPS_SETTINGS (ID, VALUE)
VALUES ('INITIAL_MODELS_SQL',
        'select ID from LSPS_MODULES where
            NAME=''my-process'' and
            UPLOAD_DATE = (SELECT MAX(UPLOAD_DATE) from LSPS_MODULES where NAME=''my-process'
        union
        select ID from LSPS_MODULES where
            NAME=''my-contracts'' and
            UPLOAD_DATE = (SELECT MAX(UPLOAD_DATE) from LSPS_MODULES where NAME=''my-contract
```

1.2.2 Setting Dumping of Model Instances with Exceptions

If a model instantiation fails with an exception (its interpretation fails), the model instance data is lost. This behavior can cause performance issues due to too large model instances with exceptions being stored. Consider enabling storing of the errors with the marshalled model instances in the database.

To enable the setting, perform the following insert on the LSPS database:

```
INSERT into LSPS_SETTINGS(ID,VALUE) values ('DUMP_MODEL_INSTANCE_ON_EXCEPTION','true')
```

1.2.3 Decreasing Frequency of Goal-Condition Checks

If a model contains many goals with conditions, checking of the conditions might cause performance issues since the conditions are checked whenever any of the goals changes its status or a token is moved. This happens when the interpretation strategy is set to `FULL_PARALLEL`. However, the default setting is `BPMN_FIRST`.

You can set the checking to happen only when all transactions finish (all tokens are moved as far as possible).

To do so, issue the following insert statement on the LSPS database:

```
INSERT into lsps_settings (id, value) values('INTERPRETATION_STRATEGY','BPMN_FIRST')
```

1.2.4 Improving Shared Record Search

To improve performance of Shared Record search, define indexes on the Records and their Relationships.

1.2.5 Disabling System Cache Regions

Check the setting of cache regions in the `<YOUR_APP>-ejb/src/main/resources/cache-regions.properties` file of your LSPS application.

1.2.6 Disabling Hibernate Statistics

Make sure that the Hibernate statistics feature is deactivated: you can do so on the MBean tab of JConsole, the node *com.whitestein.lsps* > *Statistics* -> *Attributes* > *StatisticsEnabled*

1.3 Performance of Processes

Search for expression calls with `long execution time` and identify the cause for the long duration.

Performance issues in processes are typically caused by the following:

- **Operations over large in-memory collections**
  - `foreach`: if you are collecting results of iterations in a new collection, consider using functions of the Standard Library, for example, `collect()`, `fold()`, `exist()`, `forAll()`, etc.
- **Inefficient queries**

  Typically, queries or functions that call other queries perform poorly.
- **Assignments in conditions of asynchronous modeling elements**, such as, start conditional event, goal precondition, etc.

  These might result in an infinite loop: make sure such a situation does not occur

  Also consider that the server checks for all changes on such conditions in each transaction: this can mean that the system might perform such extensive checks on every change of a goal status for example. Also consider the `BPMN_FIRST` server setting.
- **Multiple intermediate events, and goals and plans that wait to be activated at the same time**
- **Multiple processes that communicate concurrently with signals or similar mechanisms**

  Avoid too frequent communication between different model instances with signals or shared data, when the signals or data is used in conditions or filters of events. Consider moving the processes into one model, so that the communication takes place within one model instance.
- **Frequent timer checks**

  Consider setting the `TimerInterval` to a non-zero value.
- **Large objects in `Signals` and in properties of model instances**

  (Model instance properties can be set, for example, when the model instance is created with the `createModelInstance(synchronous*: Boolean, model*: Model, properties↩ :Map<String, Object>)` function.)

1.4 Performance of Operations over Data

Search for expression calls with `long execution time with Profiler`.

Performance issues in obtaining persisted data are typically caused by the following:

- **Record fields of complex data types**

  Transform record fields of complex data types to `related records`; complex field values must be serialized and deserialized frequently.

- **Unindexed shared Records**

  Make sure indexes are created for all shared records.

- **Inefficient getting of related records**

  Set lazy or eager loading of related records on data relationships. Also consider using data caching for shared records that are used frequently, e.g., code tables but do not change often.

- **Inefficient getting of large data sets with standard queries**

  - Consider paged queries.
  - Consider using fetch joins.

- **Editing items of records via relationships with List multiplicity**

  Changes on such records are expensive since they require rebuilding of the entire entity and possibly additional SQL statements for shared Records. Therefore, if a List relationship end cannot contain the same item twice, switch it to Set: When you change the Set value, for example, add a new item to the Set, the system checks and updates only the particular item, while for a relationship of the List type, the entire List entry is rebuild.

- **Slow sorting of items using the `sort(list* : List<E>, comparator* : {E, E : Integer}) : List<E>` function** If possible, use the `sortByKey()` function with the parameters which returns a list of elements sorted by the keys produced by the keyExtractor. The function takes

  - the collection of objects to be sorted,
  - the keyExtractor closure that retrieves the keys to use for sorting.

1.5 Performance of UI Forms

Performance issues in obtaining persisted data are typically caused by the following:

- **Table with large data**

  - If you are using a simple table, use lazy or paged tables instead.
  - Consider using of fetch joins for the queries that load table data.
  - Avoid in-memory paging or ordering: use query functionality.
  - Avoid in-memory filtering and sorting.

- **Many components in a single form**

  Divide the form into multiple screens of a screen flows.

- If you are using **complex logic to initialize local variables**, consider moving the initializing to an Init listener: since local variables are initialized before the form tree is assembled, the form would be displayed sooner. However, make sure that the Init listener is then not called repeatedly in the form.

- If you want to exclude a component completely from the form hierarchy use the Expression component with an `if` statement; note that the Conditional component is included in the hierarchy on initialization and initialized, while the Expression component is set when the screen context is created as the returned expression. Note that you cannot recalculate the content of the Expression component once it was initialized.