Gardner Buchanan, Solution Architect, Strategic Customer Programs

Adobe LiveCycle® ES2 Asynchronous Process Benchmarking

Examining the performance of a non-interactive application involving the asynchronous invocation of a LiveCycle process.

Overview

In this guide we examine the performance of a non-interactive application involving the asynchronous invocation of a LiveCycle process performing a straight-through process. The standard sizing models and guidelines for LiveCycle Process Management ES2 do not apply in this instance and so a customized benchmark is required to assess the performance of LiveCycle for this type of application.

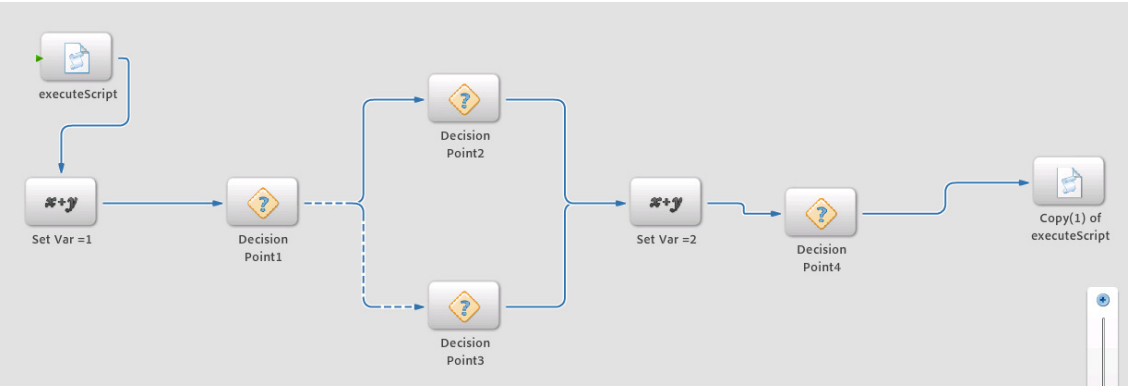
Because invocation is asynchronous, it is not possible to simply examine the response time of the invocation in order to see how LiveCycle keeps up with the load. This guide discusses other methods that can be used to assess the throughput and efficiency of process execution in this scenario.

Benchmark

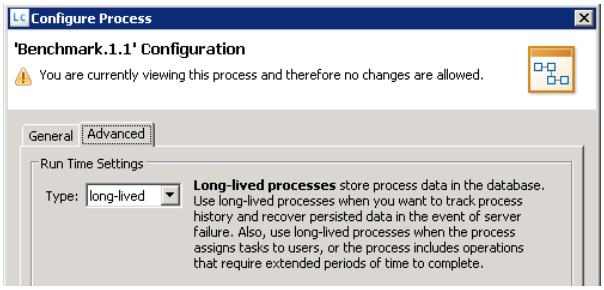
The following sections describe the details of the benchmark used. The overall approach is to establish a test process definition, generate a known test load against the server using this process, and monitor the execution of the test load to understand how the system behaves under load and how readily different load levels are sustained.

Process definition

The process map is shown below. It consists of 8 actions, four of which are decision points, two are set values, and two are script execute. The script execution steps are included to instrument the process so that performance data may be collected, however their presence also slightly increases the overall complexity of the process.



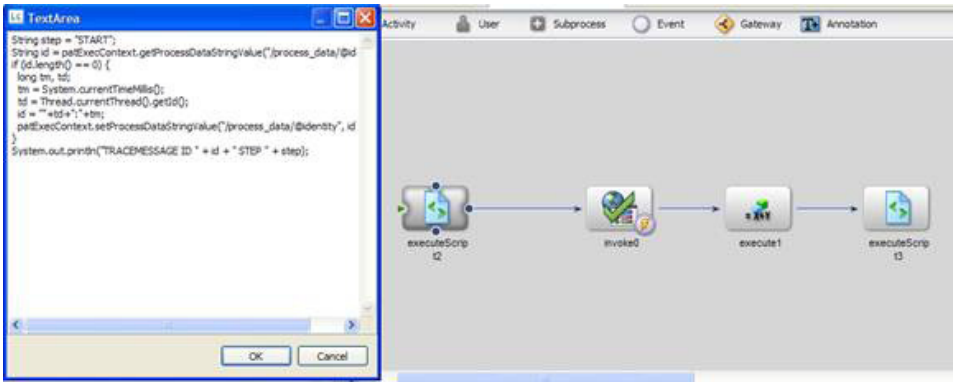
The process in the previous example is configured as long-lived. Within LiveCycle ES2 it is possible to make an asynchronous invocation against a short lived process, and the execution of the short lived process is expected to be faster than an equivalent long lived one. The overall methodology for instrumentation and benchmarking are the same in either case.



Process map instrumentation

The suggestion here is to instrument a LiveCycle process map with checkpoints at which performance data is recorded. Using the performance data you can then assess how often the process is invoked, how long it takes to execute, or how long it takes to reach various points in its execution.

The orchestration sample below is bracketed by two script execution actions whose purpose is to generate a timestamp in the application server log. By means of the resulting log messages, the execution time for the orchestration within LiveCycle can be measured. If multiple activities occur within the process map that you want independent timing information for, it is possible in insert instrumentation steps at multiple points.



If needed, the same script can be inserted at multiple points and the various points would just be given different step names ("START" shown in red below). In this example the step names are "START" and "END".

```
String step = "START";
String id = patExecContext.getProcessDataStringValue("/process_data/@identity");
if (id.length() == 0) {
  long tm, td;
  tm = System.currentTimeMillis();
  td = Thread.currentThread().getId();
  id = ""+td+":"+tm;
  patExecContext.setProcessDataStringValue("/process_data/@identity", id);
}
System.out.println("TRACEMESSAGE ID " + id + " STEP " + step);
```

LiveCycle is likely to execute more than one process instance concurrently. When reviewing the logs, it may be necessary to be able to distinguish one transaction from another one so unique transaction IDs are assigned.

In the case that a transaction ID is not set, the script constructs a unique transaction ID using the current time and the current thread ID. This ID is then placed in the string process variable "identity". The script allows the process to use its own suitable unique transaction ID and in this benchmark we pass this value into the invocation from JMeter.

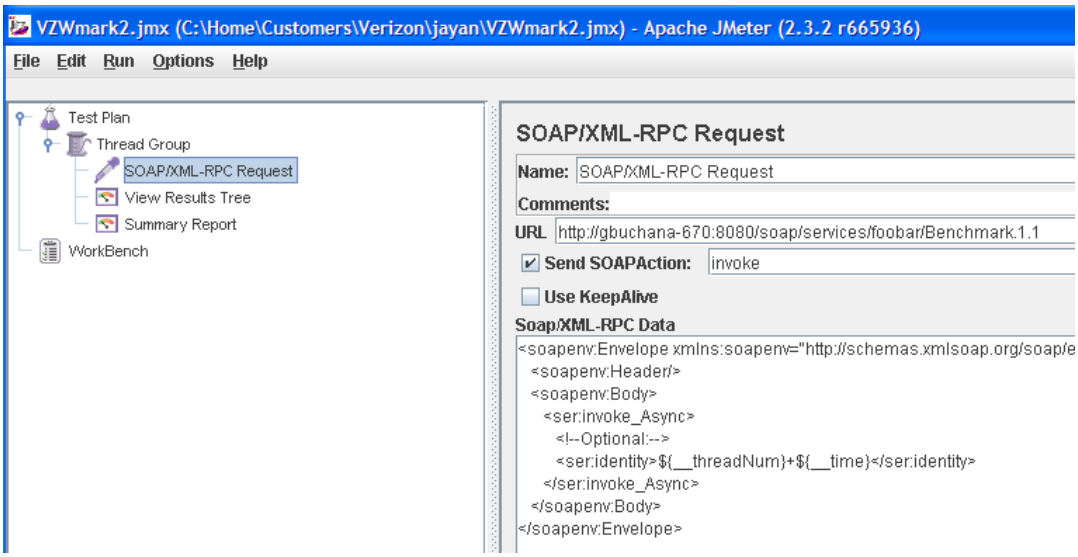
Analysis of the timestamps allows the overall LiveCycle server execution time to be calculated. This sample log is generated by WebSphere.

```
2010-02-12 16:37:38,901 INFO  [STDOUT] TRACEMESSAGE ID 3+1266010413786 STEP START
2010-02-12 16:37:39,041 INFO  [STDOUT] TRACEMESSAGE ID 18+1266010413301 STEP START
2010-02-12 16:37:39,057 INFO  [STDOUT] TRACEMESSAGE ID 12+1266010413630 STEP END
2010-02-12 16:37:39,088 INFO  [STDOUT] TRACEMESSAGE ID 4+1266010413301 STEP END
```

In this example a transaction ID value is shown as "3+1266010413786". This value is generated by the load generation tool and passed into the invocation. It is composed of the thread ID (3) and the timestamp in milliseconds (1266010413786). By capturing the submission time in these log messages it is possible to obtain a full analysis of the latency, time over which the process executed, and transaction rates over time. A simple script can be used to extract this data. This data can then be transformed to generate charts or other useful analysis.

Load generation

The process is configured so it may be invoked asynchronously and the bulk invocations required are accomplished through an Apache JMeter load script. The following image shows the outline of the test plan as well as the key SOAP request step that performs the invocation.



Within this test plan the Thread Group settings specify the number of concurrent requests that will be performed (20) and the number of sequential requests by each thread (100) for a total of 2,000 requests in a single burst.

The SOAP request being passed in is a simple asynchronous invocation composed as follows:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://adobe.com/idp/services">
  <soapenv:Header/>
  <soapenv:Body>
    <ser:invoke_Async>
      <!--Optional:-->
      <ser:identity>${__threadNum}+${__time}</ser:identity>
    </ser:invoke_Async>
  </soapenv:Body>
</soapenv:Envelope>
```

In this XML message the ${__threadNum} and ${__time} strings are variables that are inserted by JMeter at run time. *${__threadNum}* is replaced by the JMeter thread number and *${__time}* is replaced by the timestamp in milliseconds. **Note that properly interpreting comparisons between the original invocation time generated by JMeter and execution times generated on the server requires that the clocks on the two systems be carefully synchronized, such as by NTP**.

Database monitoring

LiveCycle ES2 uses a database to store data having to do with long lived process execution and asynchronous invocations. This is used to permit durable storage of partial results and resumption of jobs across system restarts. It is possible to monitor the content of certain tables to ascertain the level of backlog of asynchronous jobs within LiveCycle ES2. The simplest measure is to count the pending jobs in the job instance table using the following query:

```
select count(*) from tb_job_instance
  where status = 1;
```

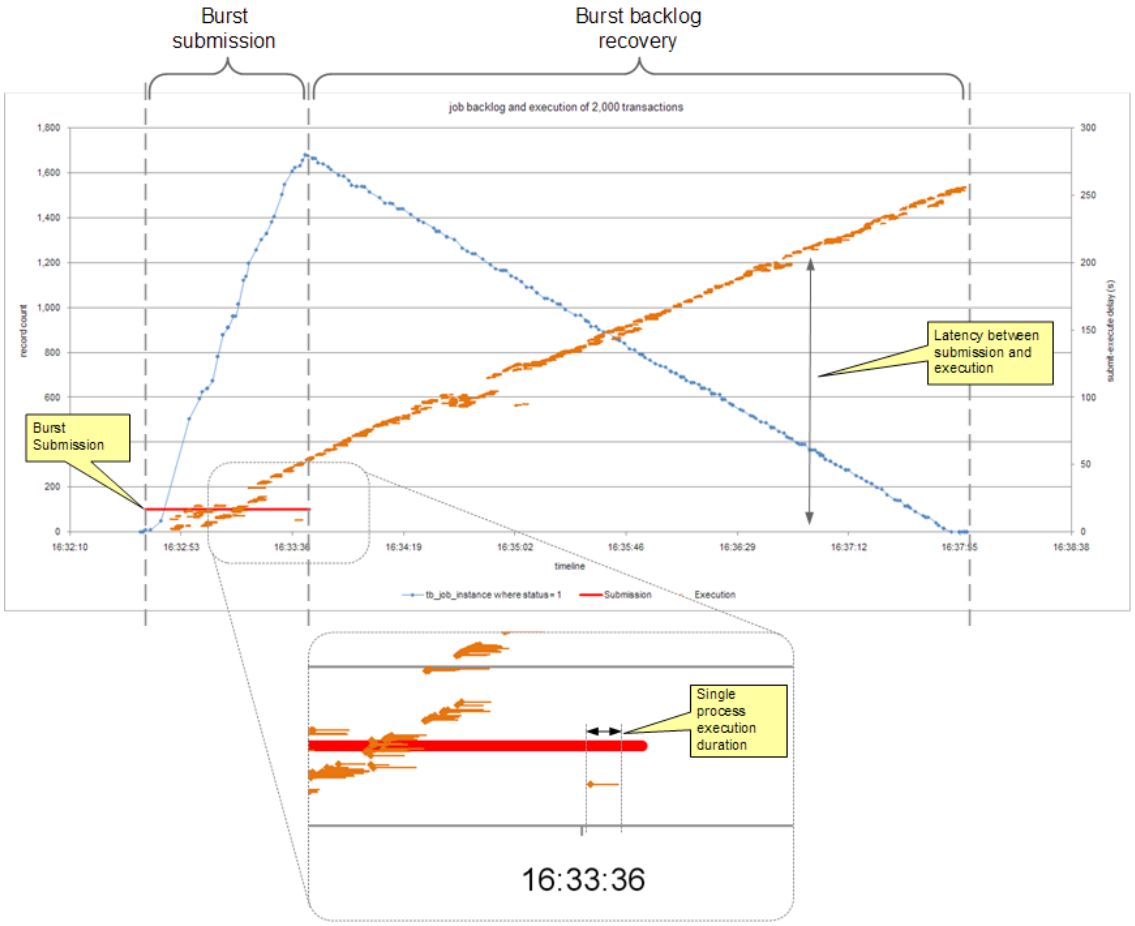
This is a reasonably efficient query that can be performed at intervals during a test to monitor backlog and execution progress.

Interpretation of results

Log metrics

The results of a benchmark run will reveal several useful metrics about the ability of any given LiveCycle ES2 configuration to accept and execute the asynchronous invocations being created. The graph below illustrates the various performance metrics of interest and combines these into a single view. The metrics of interest include:

- What is the submission rate for a burst of invocations? Ex.: How long does it take to submit 2,000 invocations in rapid succession. This may vary with the complexity of the XML arguments passed in, overall system performance, database performance, and number of requesting threads employed.
- After submission, what is the typical latency before the processes begins to execute? In a large burst the latency will be higher for invocations received later in the burst. This may vary with the overall system performance and database performance.
- How long does each job instance take to execute? What is the variability in this execution time? This will be effected by the complexity and length of the process map, overall system performance and the tendency of the process to rely on external resources. The process may be short or long-lived and this choice will affect process execution and overall performance.
- How long does it take to complete the execution of the backlog of invocations in the burst? This will vary with the overall system performance, resources, and time required to execute the individual process instances.
- What is the overall throughput of processes executed over time? Ex.: The throughput in jobs per hour. This will vary with the overall system performance, resources, and time required to execute the individual process instances.



System metrics

In addition to the internal LiveCycle ES2 execution metrics, it will also be useful to capture system level performance data. Useful information may include:

- CPU utilization on the LiveCycle ES2 application server. Both overall and use by the application server JVM.
- Memory utilization, including JVM physical memory and Java garbage collector activity.
- Network utilization between the application server and database, between the load generator and application server, and between the application server and remote services used by the processes such as web services.
- CPU utilization and disk utilization on the database server.

These items can be monitored using common system utilities such as Windows Perfmon, UNIX VMStat, Sun JConsole, and similar utilities.

References

- Apache JMeter may be used to test performance under different load types
  http://jakarta.apache.org/jmeter/
- Adobe LiveCycle ES2 full documentation set
  http://help.adobe.com/en_US/livecycle/9.0/lc_doclist.html
- A useful tutorial on charts in Microsoft Excel published by Peltier Technologies
  http://peltiertech.com/Excel/ChartsHowTo/index.html

We welcome your ideas and comments. Please send any feedback on this technical guide or suggestions for other topics to: LCES-Feedback@adobe.com

For more information and additional product details:
http://www.adobe.com/devnet/livecycle/

Adobe

Adobe Systems Incorporated
345 Park Avenue
San Jose, CA 95110-2704
USA
www.adobe.com