



# Assessing Performance

© 2009 Active Endpoints Inc. ActiveVOS and ActiveVOS are trademarks of Active Endpoints, Inc. All other company and product names are the property of their respective owners.



2009

## Assessing ActiveVOS Performance

We are often asked, “What is the maximum load that ActiveVOS can handle?” The answer, as is the case in any benchmarking scenario, is “It Depends.” In particular, the number of transactions that ActiveVOS can process in a second depends on a number of factors, including:

- Complexity of the ActiveVOS process
- Is there a process persistence requirement to enable the ability for a process to recover gracefully in case of system failure?
- ActiveVOS Server Deployment Environment
  - Processor power
  - Multiple CPU’s, which helps multi-threading performance
  - Server clustering, which improves scalability and provides fail-over
- Memory
- Database efficiency (if persistence is used)
- Response time of services and systems used in a process

The above is just a partial list of the factors that can influence ActiveVOS performance.

To give you a feel for the throughput that is possible with ActiveVOS, we created the following four tests:

### 1) Echo web service (447 txn / sec)

This does not use BPEL processes at all. It shows the maximum throughput for request-response Web Services in our runtime environment.

### 2) Echo business process (428 txn / sec)

This is basically the same as the Echo Web service, except that it is implemented using a BPEL process with 3 activities: receive request, assign result (4 copies to XPath targets), send reply.

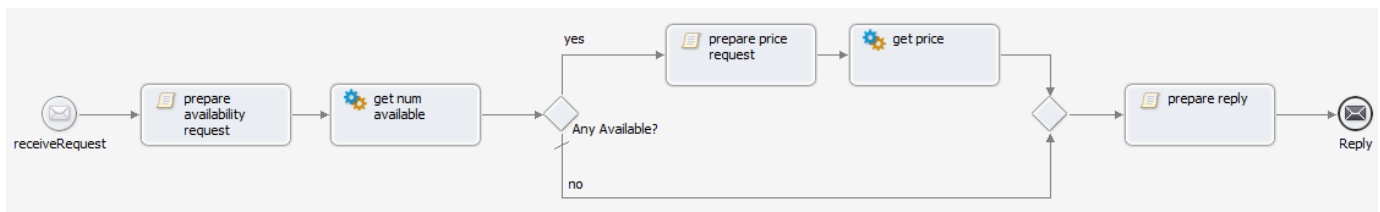
### 3) Availability business process – no persistence (327 txn / sec)

This uses a more complex business process, which is shown below (with a full listing at the end of the document). It includes a Web Service receive, 2 activities implemented in Java, 2 XPath conditional expressions, 3 assign activities and a Web Service reply. This version did not have persistence turned on.

### 4) Availability business process – with persistence (127 txn / sec)

This is the same process as above (and shown below), except that the processes state information are persisted to the database.

The availability process used was the following:

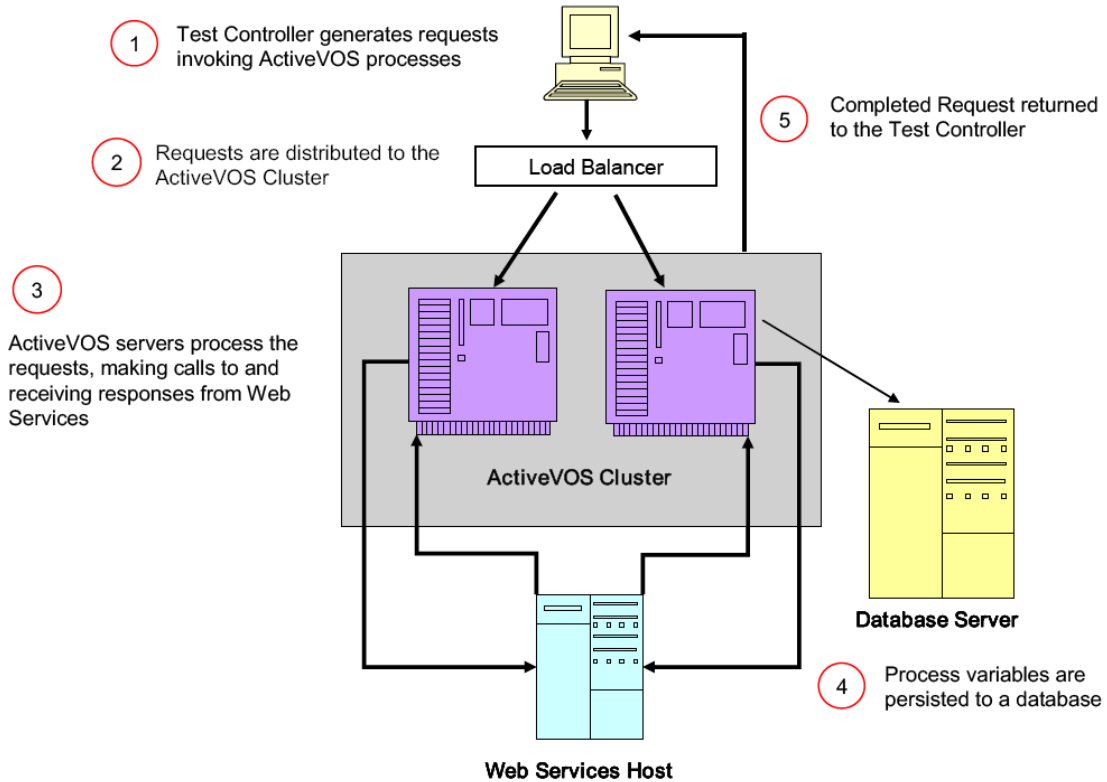


The full BPEL for this process is at the bottom of this document.

The activities in this process were implemented with Java code and invoked through our Java invocation handler. To the extent that your own process uses Web Service calls, the time that it takes for the web service overhead, as represented by the Echo Web Service above, should be factored in. Naturally, you also need to figure in the performance of the actual business logic of our services as well. The test used here just generated random numbers for the inventory and the price.

## Runtime Environment

The environment used to generate the results described above is depicted here.



The software and hardware used in this configuration is as follows:

1. Testing tool - JMeter 232 running on Solaris, 2x1.7ghz SPARC CPUs
2. Load balancer – Apache 2.2 load balancer running with round robin on Solaris, 2x1.7ghz and CPUs
3. Server - ActiveVOS Server on JBoss running on 2 cluster nodes, each running Win2k3, 2x3.0 XEON CPUs
4. Database - SQLServer running on 1 Win2k3 server, 2x3.2 XEON CPUs

The software configuration on the server nodes was the following:

JBoss JVM settings: -Xms256m -Xmx1024m -XX:MaxPermSize=512M  
 JBoss connection pool max = 100 default  
 JBoss Http thread max = 250 default  
 ActiveVOS settings - process count = 150, logging = off, rest are all defaults

The tests were run with a variable request rate from a maximum of 50 simultaneous synthetic users.

## Summary/Conclusion

As we described the maximum load that ActiveVOS can handle varies on a number of factors. This document contrasted the performance of a native Java Web service with a BPEL implementation of an Echo service, as well as contrasting the overhead associated with process persistence.

## About Active Endpoints

Active Endpoints ([www.activevos.com](http://www.activevos.com)) is the leading developer of visual orchestration systems. VOS empowers line of business project teams to create applications using services and industry standards, making their businesses more agile and effective. Active Endpoints' ActiveVOS promotes mass adoption of SOA-enabled applications by focusing on accelerating project delivery time with a standards-based, easy to use system. Active Endpoints is headquartered in Waltham, MA with development facilities in Shelton, CT.

To find out how Active Endpoints can help your business, visit <http://www.activevos.com>, call +1 781 547 2900 and press 1 for Sales, or email us at [info@activevos.com](mailto:info@activevos.com).

## Full BPEL Definition of Availability Process

```

<bpel:process xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
xmlns:ext="http://www.activebpel.org/2006/09/bpel/extension/query_handling"
xmlns:ns1="http://www.activevos.com/wsd1/CheckPriceAndAvailability"
xmlns:nvn="urn:com:acme:services:inventoryservice" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
ext:createTargetXPath="yes" ext:disableSelectionFailure="yes" name="CheckAvailabilityProcess"
suppressJoinFailure="yes" targetNamespace="http://CheckAvailabilityProcess">
  <bpel:extensions>
    <bpel:extension mustUnderstand="yes"
namespace="http://www.activebpel.org/2006/09/bpel/extension/query_handling" />
  </bpel:extensions>
  <bpel:import importType="http://schemas.xmlsoap.org/wsd1/"
location="../wsdl/CheckPriceAndAvailability.wsdl"
namespace="http://www.activevos.com/wsd1/CheckPriceAndAvailability"/>
  <bpel:import importType="http://schemas.xmlsoap.org/wsd1/"
location="../wsdl/InventoryService/inventoryservice.wsdl"
namespace="urn:com:acme:services:inventoryservice"/>
  <bpel:partnerLinks>
    <bpel:partnerLink myRole="inventory" name="client" partnerLinkType="ns1:AvailabilityPLT"/>
    <bpel:partnerLink name="InventoryBackend" partnerLinkType="nvn:InventoryServicePLT"
partnerRole="java"/>
  </bpel:partnerLinks>
  <bpel:variables>
    <bpel:variable messageType="ns1:AvailabilityRequest" name="AvailabilityRequest"/>
    <bpel:variable messageType="ns1:AvailabilityResponse" name="AvailabilityResponse"/>
    <bpel:variable element="nvn:getNumAvailable" name="getNumAvailable"/>
    <bpel:variable element="nvn:getNumAvailableResponse" name="getNumAvailableResponse"/>
    <bpel:variable element="nvn:getPrice" name="getPrice"/>
    <bpel:variable element="nvn:getPriceResponse" name="getPriceResponse">
      <bpel:from>
        <bpel:literal>
          <nvn:getPriceResponse>
            <double>0.0</double>
          </nvn:getPriceResponse>
        </bpel:literal>
      </bpel:from>
    </bpel:variable>
  </bpel:variables>
  <bpel:flow>
    <bpel:links>
      <bpel:link name="L1"/>
      <bpel:link name="L3"/>
      <bpel:link name="L6"/>
      <bpel:link name="L5"/>
      <bpel:link name="L2"/>
      <bpel:link name="L4"/>
      <bpel:link name="L7"/>
    </bpel:links>
    <bpel:receive createInstance="yes" name="receiveRequest" operation="checkAvailability"
partnerLink="client" variable="AvailabilityRequest">
      <bpel:sources>
        <bpel:source linkName="L1"/>
      </bpel:sources>
    </bpel:receive>
    <bpel:reply operation="checkAvailability" partnerLink="client" variable="AvailabilityResponse">
      <bpel:targets>
        <bpel:target linkName="L7"/>
      </bpel:targets>
    </bpel:reply>
    <bpel:invoke inputVariable="getNumAvailable" name="getNumAvailable" operation="getNumAvailable"
outputVariable="getNumAvailableResponse" partnerLink="InventoryBackend">
      <bpel:targets>
        <bpel:target linkName="L2"/>
      </bpel:targets>
    </bpel:invoke>
  </bpel:flow>

```

```

        <bpel:source linkName="L3">
            <bpel:transitionCondition>$getNumAvailableResponse/int &gt; 0</bpel:transitionCondition>
        </bpel:source>
        <bpel:source linkName="L6">
            <bpel:transitionCondition>$getNumAvailableResponse/int = 0</bpel:transitionCondition>
        </bpel:source>
    </bpel:sources>
</bpel:invoke>
<bpel:invoke inputVariable="getPrice" name="getPrice" operation="getPrice"
outputVariable="getPriceResponse" partnerLink="InventoryBackend">
    <bpel:targets>
        <bpel:target linkName="L4" />
    </bpel:targets>
    <bpel:sources>
        <bpel:source linkName="L5" />
    </bpel:sources>
</bpel:invoke>
<bpel:assign name="prepGetNumAvailable" suppressJoinFailure="yes">
    <bpel:targets>
        <bpel:target linkName="L1" />
    </bpel:targets>
    <bpel:sources>
        <bpel:source linkName="L2" />
    </bpel:sources>
    <bpel:copy>
        <bpel:from part="partID" variable="AvailabilityRequest"/>
        <bpel:to variable="getNumAvailable">
            <bpel:query>arg1_string</bpel:query>
        </bpel:to>
    </bpel:copy>
</bpel:assign>
<bpel:assign name="prepGetPrice">
    <bpel:targets>
        <bpel:target linkName="L3" />
    </bpel:targets>
    <bpel:sources>
        <bpel:source linkName="L4" />
    </bpel:sources>
    <bpel:copy>
        <bpel:from part="partID" variable="AvailabilityRequest"/>
        <bpel:to variable="getPrice">
            <bpel:query>arg1_string</bpel:query>
        </bpel:to>
    </bpel:copy>
</bpel:assign>
<bpel:assign name="prepReply">
    <bpel:targets>
        <bpel:target linkName="L6" />
        <bpel:target linkName="L5" />
    </bpel:targets>
    <bpel:sources>
        <bpel:source linkName="L7" />
    </bpel:sources>
    <bpel:copy>
        <bpel:from variable="getPriceResponse">
            <bpel:query>number (double) </bpel:query>
        </bpel:from>
        <bpel:to part="price" variable="AvailabilityResponse"/>
    </bpel:copy>
    <bpel:copy>
        <bpel:from variable="getNumAvailableResponse">
            <bpel:query>int </bpel:query>
        </bpel:from>
        <bpel:to part="numAvailable" variable="AvailabilityResponse" />
    </bpel:copy>
</bpel:assign>
</bpel:flow>
</bpel:process>

```