Capacity Planning Guide for Adobe® LiveCycle® Data Services 2.6

Create applications that can deliver thousands of messages per second to thousands of end users simultaneously

Table of contents

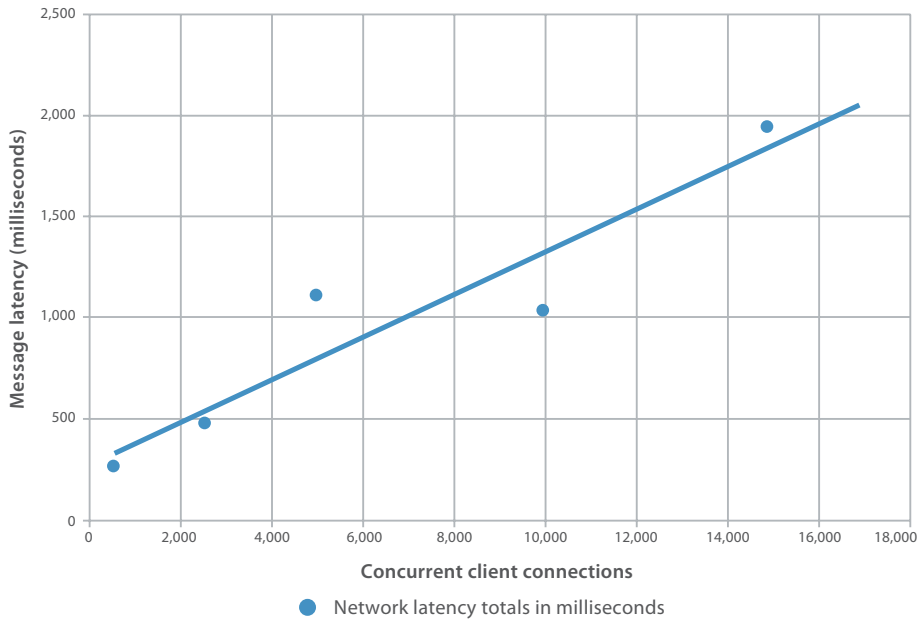
This document presents the results of software benchmark tests performed by Adobe engineers in 2008. These tests show how LiveCycle Data Services 2.6 software can scale and perform under load using various messaging scenarios that represent real-world situations. The goal of this paper is to provide a starting point for those who need to plan a hardware and software infrastructure that can scale to meet peak demand.

LiveCycle Data Services ES provides a framework for developers to quickly create highly scalable applications using Adobe Flex® and Adobe AIR™ technologies. LiveCycle Data Services ES primarily consists of a core Messaging Service, with the higher level Data Management Service and Remoting Service built on top of it. The performance of the Messaging Service determines true performance of the server.

The Messaging Service can communicate with Adobe Flex clients over Real Time Messaging Protocol (RTMP) and HTTP. All performance results in this report are based on clients using RTMP. If your application uses HTTP, we have included results (see page 8) that show the relative performance difference between RTMP and HTTP.

The tests were run scaling up to 30,000 simultaneous consumer subscriptions, each with separate connections to the server via RTMP. The endpoint architecture, based on Java™ New I/O (NIO), provides unprecedented scalability. New streaming channels provide a performance-efficient alternative to RTMP. In addition, using long polling, you can sustain a very high message throughput rate. The tested systems were single- and dual-node configurations made up of four-way HP ProLiant servers running Microsoft® Windows® operating systems and Apache Tomcat application servers. The network connection was Gigabit Ethernet.

Adobe®



Message latency versus client connection count.

Sizing and performance

LiveCycle Data Services ES at its core contains messaging services that the data management service depends on. Rich Internet application (RIA) clients communicate with LiveCycle Data Services over HTTP or RTMP. For applications that require clients to receive real-time or near real-time data, LiveCycle Data Services ES supports several channels. You can find more information about supported channels and endpoints at *http://livedocs.adobe.com/livecycle/8.2/programLC/programmer/lcds/lcconfig_1.html*.

This report focuses on the performance of the applications using the Messaging Service over RTMP. We have also tested the relative performance of RTMP with other HTTP channels that we support. The combined result should provide sufficient insight as you design your application and size your infrastructure.

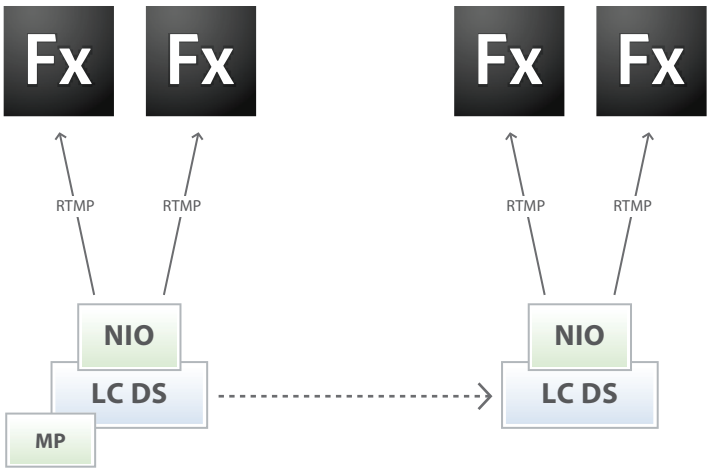
Our test application simulated clients receiving real-time data from one or more LiveCycle Data Services ES (LC DS) servers. The real-time data feed from the Message Producer (MP) systems was available only to one LC DS. In a clustered environment, all messages were replicated to other LC DS servers. You can find more information about LC DS clustering at *http://livedocs.adobe.com/livecycle/8.2/programLC/programmer/lcds/services_clustering_1.html*.

Software support

Adobe LiveCycle Data Services 2.6 is a standalone application that can also be deployed as a solution component of Adobe LiveCycle ES (Enterprise Suite).

Application servers
- JBoss
- Apache Tomcat
- BEA Weblogic
- IBM Websphere

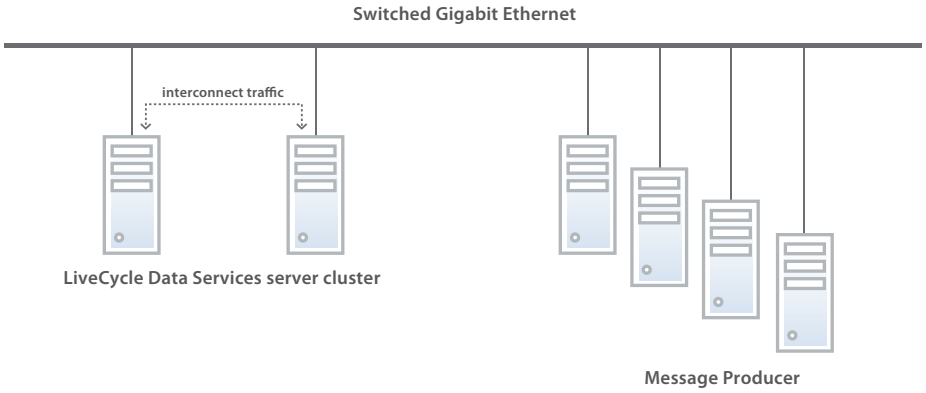
Operating systems
- Red Hat® and SUSE® Linux®
- Sun™ Solaris™
- IBM® AIX®
- Microsoft Windows



Fx: Flex client
RTMP: Real Time Messaging Protocol
LC DS: LiveCycle Data Services ES
NIO: New I/O server built into LC DS
MP: Message Producer

Logical cluster configuration.

Appendixes A and B provide more information about the hardware and software used in our test environment. A private switched network connects the systems and carries no traffic other than the benchmark load. LiveCycle Data Services ES is deployed on both nodes. The two nodes communicate to push transmitted messages from one node to the other, before transmitting them to the clients.



Clustered dual-node configuration for performance benchmarks.

This paper provides general sizing and performance information. Your mileage may vary, depending on your systems, workloads, and environment. Adobe recommends proof-of-concept testing in a non-production environment using the actual target application to gather additional information about real-world performance.

Your specific environment and the choices you make in terms of selecting or using specific technologies can affect how the application performs and your organization's sizing requirements. Application-specific choices include channels, clustering configuration, number of concurrent users, message size, acceptable latency of messages received by clients, and rate of messages to be pushed. Infrastructure-specific choices include the server capacity (number of CPUs, memory size, I/O), number of servers and how they are clustered, network connectivity, operating system, and application server.

Our goal is not to provide a comprehensive list of all factors that affect performance and sizing decisions but rather act as a guide to aid in your decision-making process.

Messaging Service

The messaging performance tests measure how well a LC DS server performs under specific conditions to determine the maximum number of messages that the server can process. Each test involves a single message producer application identified as MP or Message Producer in the setup. MP is a simulation of a data feed; it is not part of LiveCycle Data Services ES.

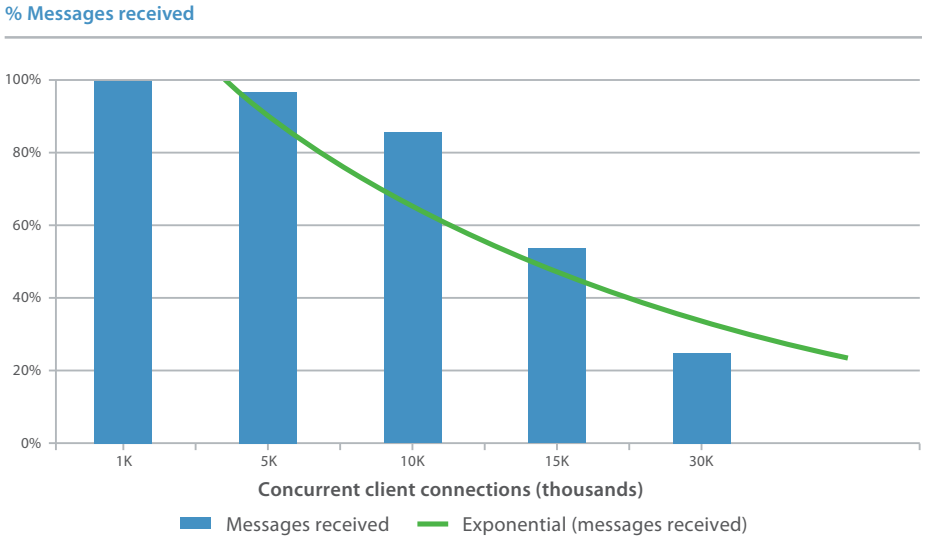
The responsibility of LC DS is to maintain client subscriptions to the data feed and push these feeds to the connected clients. The feed sends a steady stream of messages of a certain data payload size to the server. These messages are then pushed out via the Messaging Service to a fixed number of clients that are subscribed to the same destination as the producer.

Number of clients

The objective of this test is to determine the performance as more clients subscribe to data pushed from LC DS. The test case involves the following:

- Fixed number of clients that subscribe to a Messaging Service destination and wait to receive messages
- Message Producer that generates a data feed of 10 messages per second (mps)

The number of messages that the clients receive in 60 seconds is measured. Measuring in 60-second intervals can reveal how the number of concurrent clients impacts message throughput.



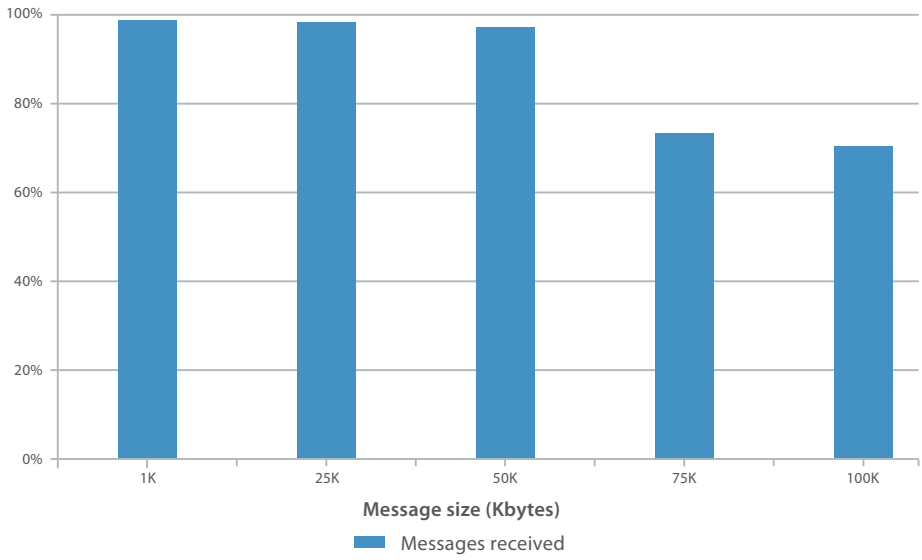
The number of clients impacts message throughput. A single server can support up to 30,000 concurrent users. However, as the number of users increases from 1,000 to 30,000, the number of messages received within 60 seconds drops from 100% to 23%.

If we did not set a time limit, all messages would eventually arrive. So if your application does not require clients to receive data in real time, you can support a high number of concurrent users. If your application requires that data is received by clients in real time or within less than 60 seconds, consider reducing the number of concurrent users.

Message size

We examined what impact message size has on the throughput by calculating the total number of messages received per second by each participant for the 60-second duration. The test has 1,000 concurrent clients sending at a rate of 5 mps. The measurement is taken using successively larger message sizes. The ability of the server to send all the messages in a fixed time reduces as the message size increases.

% Messages received by message size

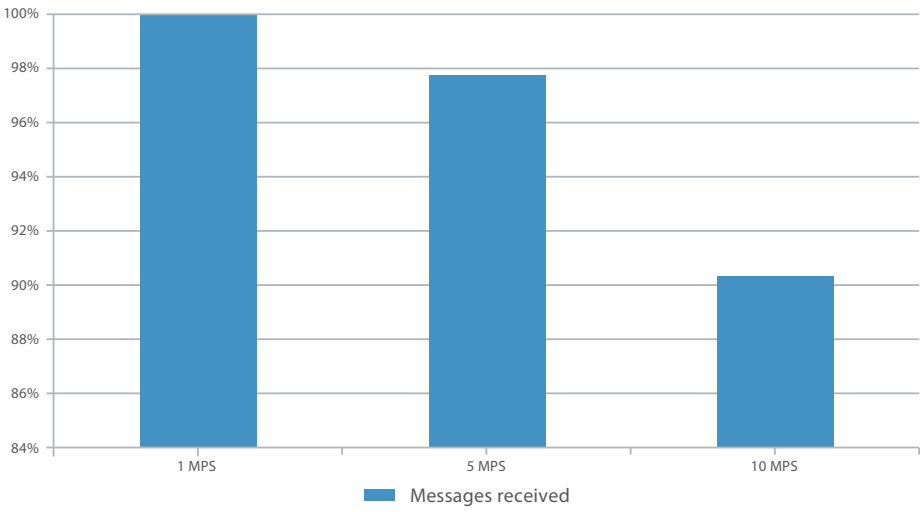


As message size increases, the ability of the server to keep up with bursts of messages declines. If message delivery latency is a critical factor, a smaller message size ensures more rapid delivery.

Message rate

Message rate is calculated by counting the percentage of messages successfully delivered within a fixed time frame. The test uses successively faster send rates, ranging from 1 to 10 mps per client. The ability of the server to send all the messages in a fixed time reduces as the number of messages increases.

% Messages received by 10,000 concurrent clients



Percentage of messages delivered within a fixed time frame with different message send rates.

At 1 MPS, messages are delivered with no measurable backlog. At increased burst rates, a backlog develops and some messages are delivered outside the 60-second test interval. However, even at the maximum test burst rate of 10 mps, over 90% of the messages were received.

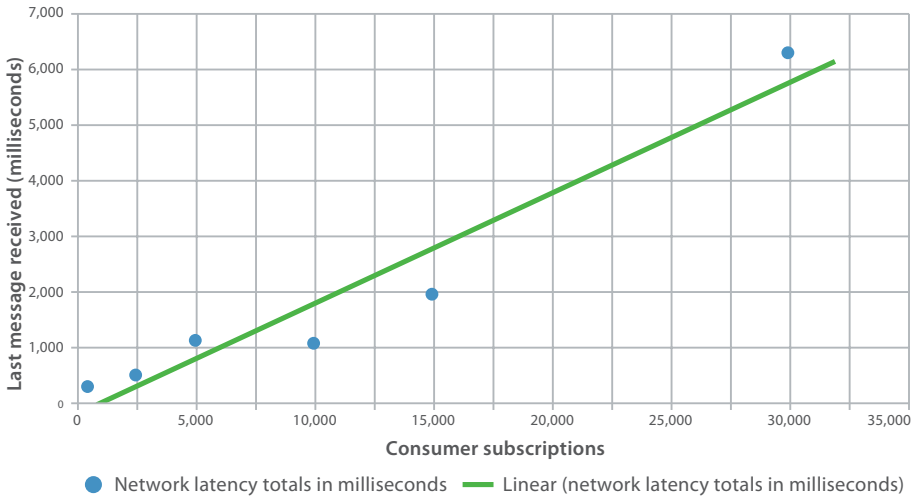
Multiple clients connecting to a single server

This test required that we use our own NIO Load Tool, which uses a Java client library to simulate Flex clients. This tool is capable of making thousands of consumer subscriptions over RTMP channels, simulating thousands of users on a single server. We ran tests beginning with 500 consumer subscriptions and scaled up to 30,000 simultaneous connections. Each subscription uses a separate, dedicated RTMP connection to the server.

After all the consumers are subscribed, a producer publishes a message to the destination. The message is then pushed out to all subscriptions. We marked the time the message is first published by the producer (producer timestamp PT), and the time that it takes for the last consumer to receive the message (consumer timestamp CT). We subtracted the last CT from the PT, which gives us the total network latency: CT - PT = TNL.

The system time of all the machines was synchronized.

RTMP max client connection limits



The total latency to broadcast a message at the server and receive it across all clients increases with the number of connections. As the client count increases, more load is put on the server.

While this test is useful in determining how many open concurrent connections the server can support, it isn't guaranteed to match actual throughput when using a real Adobe Flash® client. Note that for a system to accept tens of thousands of simultaneous connections, you must first increase the number of sockets a process is allowed to manage via the operating system setting.

Data Management Service

Most of the data in this capacity planning guide deals with the Messaging Service. One might ask why the Data Management Service does not have more data. This is because the Data Management Service generally uses the features provided by the Messaging Service internally, so the Messaging Service performance data can be applied to the Data Management Service. The Data Management Service does have some additional overhead compared to using the Messaging Service directly, and some Data Management Service features have more overhead than others.

When the Flex client application adds, updates, or removes data from a collection that is backed by a Data Management Service destination, the Messaging Service is used internally to send these changes to other Flex clients. When the Flex client application performs a fill operation to load data from a Data Management Service destination, a request is made directly to the server. The server then replies with a response containing the requested data, similar to a remote procedure call using the Remoting Service. For Data Management Service fill operations, performance should be similar to Remoting Service operations that return data sets of a similar

size. For Data Management Service create, update, or delete operations, performance should be similar to the Messaging Service, taking into account the additional overhead added by certain Data Management Service features.

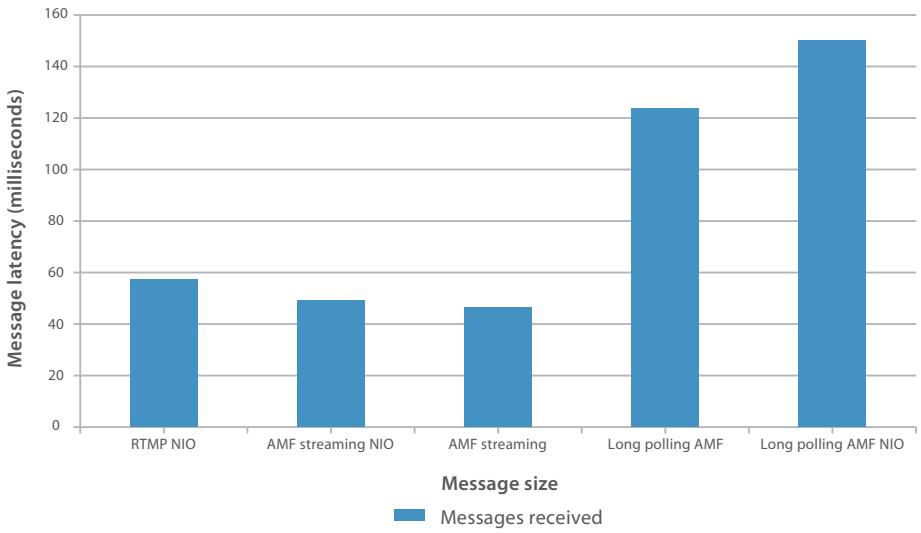
Using different channels

The Messaging Service can communicate with Flex clients over RTMP and HTTP. If you are using HTTP, several channels are available. We tested the performance differences of these channels.

Channels are client-side objects that encapsulate the connection behavior between Flex components and the LC DS. The channel types employ different protocols that have different performance and functional trade-offs. A channel type that can run through a client-side HTTP proxy, for example, might not be the most efficient one to use where firewall rules allow direct network traffic to pass.

The chart below shows a high throughput rate for streaming, and that it is the most efficient real-time channel for performance for both the client receiving the message in real time and the server's ability to process and send the message on a dedicated connection. While servlet-based streaming displays the highest throughput for the real-time channel endpoints, the servlet API requires a thread per connection and cannot scale to thousands of concurrent connections.

Message latency by channel type

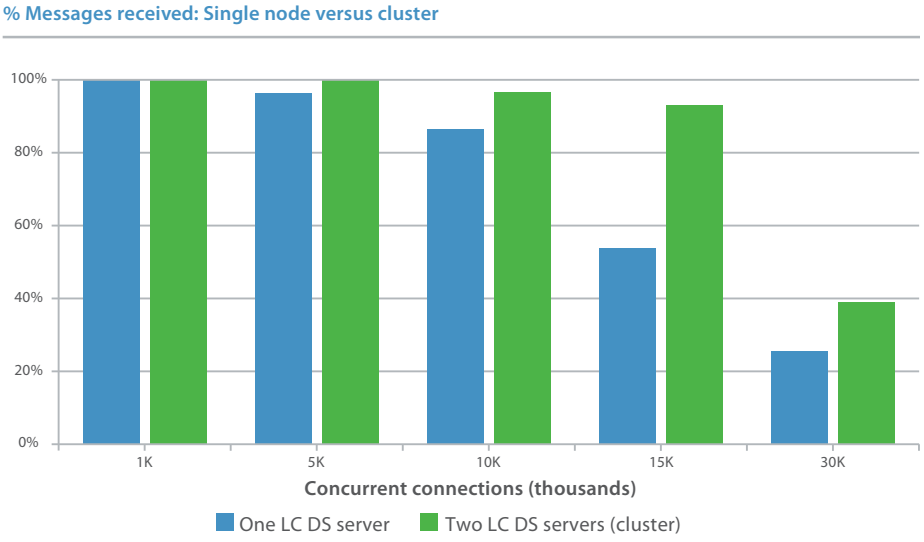


Relative performance of channels and endpoints. Performance for long polling is better for less than a thousand clients because of the message batching that occurs on the server between client poll requests.

The RTMP and action message format (AMF) streaming channel types have similar performance and have the lowest latency compared to other channels. These are the recommended channels when low message latency is a critical requirement. For any high throughput messaging scenarios, long polling could be the best option as long as the network path between the client and server isn't really long. Otherwise, the polling roundtrip would mean more of a hit (such as New York to London). For nonstreaming channels, such as the AMF channel, the polling interval dictates how often the client checks the server for new messages. The longer the polling interval, the longer it is before the client checks for or receives new messages. However, because of the message batching that happens on the server between client poll requests, it is possible to receive many more messages in batch format.

Clustering

We also looked at what effect clusters of LC DS servers have on overall message throughput.



Cluster performance with increasing load.

Results indicate that overall capacity is nearly doubled when a second test node is added. Adding a second node has no impact when there are lower numbers of concurrent users, because the server is not loaded. The results are consistent across channels. As we add another server, we alleviate the stress on the smaller cluster size and increase the cluster's ability to support more traffic, concurrency, and overall message throughput. The resources on the individual server become less strained and more responsive.

Client performance considerations

The threading model for NIO-based endpoints versus servlet-based endpoints differs. For servlet-based endpoints, threading is handled by the servlet container, with the limitation of using or tying up a thread per streaming AMF or HTTP connection, and tying up a thread whenever a long poll request is waiting for data on the server.

The NIO-based endpoints use a dedicated thread to accept connections on each port on which NIO endpoints are exposed, as well as dedicated threads per processor to manage I/O read and write readiness selection and actual writes for these connections. When a connection becomes readable, the actual read, along with message or data processing, is performed by a thread from a thread pool.

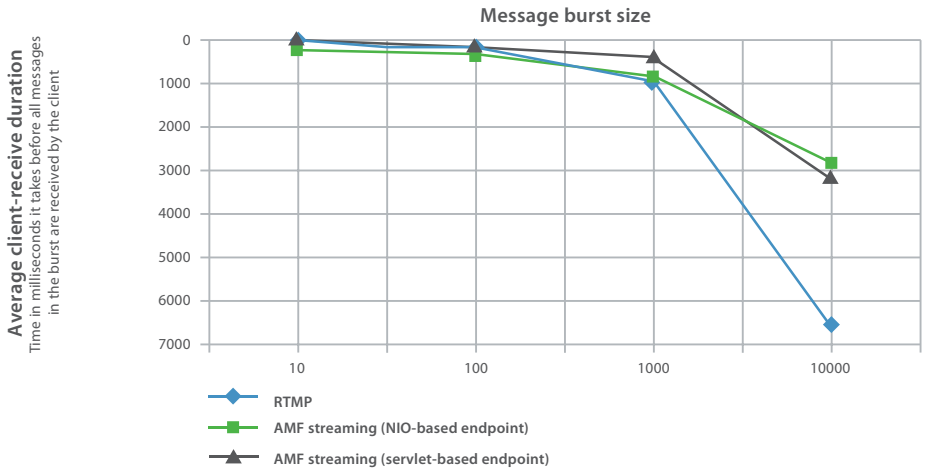
NIO endpoints do not have a dedicated read thread for each connection, which allows the NIO endpoints to support a much higher number of connections. Test results show that modifying the thread pool configuration for the NIO endpoints by imposing a max-worker-threads value lowered total message throughput, but avoided out-of-memory conditions that are caused by too many threads being created.

For applications that want to accept high numbers of mostly quiet connections, test results show that you need to modify system resource limits for TCP connections or the file handles available to the server process. The critical parameters are MaxUserPort on Microsoft Windows and ulimit -n on UNIX® and Linux.

Maximum read test

We used the Message Burst application to generate the data for the read test. In this test, the client calls a remote object, which triggers a burst of messages to be pushed from the server to the client. The burst size is the number of messages sent. The client-receive duration is the amount of time it took the client to receive all the messages, starting from the time that the client received the first message. All duration values are in milliseconds.

Flash Player maximum read by channel



Effect of increasing message burst size on read performance up to 10,000 messages.

From the test results, it is clear that as the number of concurrent messages processed by the client increases, RTMP takes longer to process than AMF. This is because RTMP has an built-in throttling algorithm that controls the rate at which messages are processed. This algorithm ensures that the Adobe ActionScript® library receives messages at a steady rate, thus avoiding having the client hang up because of data overload. With AMF (which is over HTTP), this algorithm does not exist, so it might lock up the Flex client when large streams of data are processed.

Summary

For practical sizing situations, proof-of-concept testing using the actual target application and platform combination is best practice. A proof-of-concept benchmark helps ensure that you have the correct amount and type of hardware and that you are meeting your users' expectations—internal workers are productive, and external customers continue to use your application. This guide shows you a number of characteristics related to the ability of LiveCycle Data Services to scale to support thousands of concurrent users. It provides performance and scalability characteristics of LiveCycle Data Services 2.6 that should help you determine the resource needs of your application. However, it necessarily made a number of assumptions about the application, its users, and their expectations. It is critical that you provide accurate assumptions for your own testing and perform complete tests on your targeted platform.

In situations where a proof-of-concept application is not available or approximate sizing is needed to conduct feasibility analysis prior to investing in a proof of concept, it may be possible to gauge approximate sizing requirements based on these benchmark results through adaptation or extrapolation.

Appendix A: Hardware configurations

This section describes the various hardware platform components used during the benchmarking tests.

For all tests, the client tier consisted of Windows machines running the Data Services Stress Testing tool or the NIO Load Tool. A data tier was not used, because the server-side components used in-memory data objects. The LC DS server was deployed on Windows 2003, along with Tomcat 6 and JDK 1.5. The client browsers were Internet Explorer 7 with the Flash Player plug-in version 9.0.115.

It was important that all systems be synchronized, especially for the maximum client test scenarios. To ensure synchronization, one of the server systems was the Network Time Protocol (NTP) server or authoritative time server and set the Windows Time Service registry entries of the other systems accordingly.

LiveCycle Data Services server hardware platform

Benchmarking tests used the following configuration:

- System model: HP ProLiant DL360 G3
- Processor: four Intel® x86 CPUs at ~2.8GHz
- Total physical memory: 2,047.49MB
- Total virtual memory: 3.86GB

Flash client load-generation hardware platform

The systems running the DSStress and other client load-generator software had a similar configuration as the LC DS server:

- System model: HP ProLiant DL360 G3
- Processor: four Intel x86 CPUs at ~2.8GHz
- Total physical memory: 2,047.49MB

Network hardware platform

All benchmark testing used a switched Gigabit Ethernet network fabric configuration to connect the various hardware components. The test network was isolated from any outside traffic.

Appendix B: Software configurations

This section describes the various software configurations used during the benchmarking tests.

Operating system

Benchmarking tests used the following operating system for the LiveCycle Data Services ES server:

- Microsoft Windows Server 2003, Standard Edition Version 5.2.3790, Service Pack 2 Build 3790

Server socket optimizations

- Windows [HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters]MaxUserPort = 65534
  (default = 5000, max = 65534)
- Linux ulimit -n

Server time synchronization

It was important that the real-time clocks on the server and client systems are closely synchronized. We followed these steps on all but the authoritative server:

- Change Windows to use NTP for time synchronization
  Key: HKLM\SYSTEM\CurrentControlSet\Services\W32Time\Parameters\Type
  Value: Type
  Data: NTP
- Configure the AnnounceFlags value
  Key: HKLM\SYSTEM\CurrentControlSet\Services\W32Time\Config\AnnounceFlags
  Value: AnnounceFlags
  Data: 5
- Enable the NTP server value
  Key: HKLM\SYSTEM\CurrentControlSet\Services\W32Time\TimeProviders\NtpServer
  Value: Enabled
  Data: 1
- Specify the NTP server to use
  Key: HKLM\SYSTEM\CurrentControlSet\Services\W32Time\Parameters\NtpServer
  Value: NtpServer
  Data: 192.168.100.60,0x1
- Select the NTP polling interval
  Key: HKLM\SYSTEM\CurrentControlSet\Services\W32Time\TimeProviders\NtpClient\SpecialPollInterval
  Value: SpecialPollInterval
  Data: 900
- Configure the time correction settings
  Key: HKLM\SYSTEM\CurrentControlSet\Services\W32Time\Config\MaxPosPhaseCorrection
  Value: MaxPosPhaseCorrection
  Radix: Decimal
  Data: 3600
  Key: HKLM\SYSTEM\CurrentControlSet\Services\W32Time\Config\MaxNegPhaseCorrection
  Value: MaxNegPhaseCorrection
  Radix: Decimal
  Data: 3600
- Synchronize the Windows machines by running w32tm /resync

Java application server software

The benchmarks were measured with LiveCycle Data Services ES installed in a Java application server as follows:

- Apache Tomcat server 6.0.14
- JVM version 1.5.0_15

LiveCycle Data Services ES software

RefID	Setting	Channel	Endpoint type relevance	Configuration file
1	<polling-interval>0</polling-interval>	AMFChannel	Servlet, NIO	services-config.xml
2	<wait-interval-millis>60000 </wait-interval-millis>	AMFChannel	Servlet, NIO	services-config.xml
3	<max-waiting-poll-requests>100 </max-waiting-poll-requests>	AMFChannel	Servlet	services-config.xml
4	<client-wait-interval-millis>3000 </client-wait-interval-millis>	AMFChannel	Servlet	services-config.xml
5	<max-worker-threads>UNBOUNDED? </max-worker-threads>	AMFChannel RTMPChannel	NIO	
6	<properties> <http> <session-timeout-minutes>1 </session-timeout-minutes> </http> </properties>	AMFChannel	NIO	
7	<session-config> <session-timeout>1 </session-timeout> </session-config>	AMFChannel	Servlet	web.xml
8	<Executor name="tomcatThreadPool" name-Prefix="catalina-exec-" maxThreads="200" minSpare-Threads="10"/>	AMFChannel	Servlet	server.xml
9	<max-streaming-clients>100 </max-streaming-clients>	Streaming AMFChannel	Servlet	services-config.xml

Load-generation software

Load generation for all reported benchmarks used the following Adobe load-generation tools:

- Data Service Stress Testing Framework—Previously called the DSStress tool, which was revised for this project to accommodate our needs. It is available on Adobe Labs (*http://labs.adobe.com/wiki/index.php/Data_Services_Stress_Testing_Framework*).
- NIO Load Tool—A custom tool that runs as a Java process. Using NIO, it can open large numbers of connections to a LiveCycle Data Services server. However, because it is pure Java, it doesn't include the client-side LiveCycle Data Services APIs that are available in Flash Player. For that reason, it is better suited to simple messaging scenarios that use scripted sets of messages to generate load. The NIO Load Tool is not available outside of Adobe, but plans for an externally available version of this tool are being worked on.
- Messaging Burst Application—A custom tool that is not currently available outside of Adobe.
- Browser: Internet Explorer 7
- Adobe Flash Player: Plug-in version 9.0.115

Appendix C: Benchmark tests

The table below lists the combinations of channel and message sizes that were benchmarked to generate the Messaging Service data.

Test name	Channel name	Data payload
simpleMessagingConsumerTest.mxml	perf-rtmp	5, 25, 50, 75, and 100
simpleMessagingConsumerTest.mxml	perf-nio-amf-stream	5, 25, 50, 75, and 100
simpleMessagingConsumerTest.mxml	perf-polling-amf (3 sec)	5, 25, 50, 75, and 100
simpleMessagingConsumerTest.mxml	perf-nio-amf-poll (3 sec)	5, 25, 50, 75, and 100
simpleMessagingConsumerTest.mxml	perf-streaming-amf	5, 25, 50, 75, and 100

Test 1: Many clients connected to a single server doing pub/sub messaging
We used the NIO Load tool for this test.

Test 2: Size of the message on scalability
Run the simple messaging test described earlier and use the test name and channel name for the URL. Use the data payload sizes to guide you. We performed the test for each channel and each data payload size.

Test 3: Number of clients on throughput
Run the simple messaging test described earlier and use the test name and channel name for the URL. Use the data payload sizes to guide you. We performed the test for each channel and each data payload size.

The following table lists the benchmarks used to provide the channel type comparison data.

Test name	Clients	Channel	Avr. latency	Msgs/sec received
latencyMessagingConsumer 20 msg/sec at 5 passes	100	perf-rtmp	57.0	2793.82
latencyMessagingConsumer 20 msg/sec at 5 passes	100	perf-nio-amf-stream	48.9	2194.93
latencyMessagingConsumer 20 msg/sec at 5 passes	100	perf-long-polling-amf	123.6	2820.87
latencyMessagingConsumer 20 msg/sec at 5 passes	100	perf-long-polling-amf-nio	149.6	2021.58
latencyMessagingConsumer 20 msg/sec at 5 passes	100	perf-streaming-amf	46.4	1899.67

Test 4: Performance of different channels
Results are based on calculating the values from tests 1 through 6. We took the final result from each channel's max clients (100 client instances) and max data payloads (100 record objects). This allowed us to show the relative performance of each one.

Test 5: Comparing different channels with increased clients
Results are based on calculating the values from tests 1 through 6. We took the final result from each channel's client (25, 50, 75, 100) and minimum data payloads (five record objects). This allowed us to show the relative performance of each one while increasing the client instance count.

Test 6: LiveCycle Data Services performance while under stress in a horizontal cluster
Results are based on calculating tests 2 and 3 but while in cluster configurations of 2 and 4. For this test, we used only one data payload size of 5k.

Tests 7 and 8: Max read and max write
These tests were run with the client and server on separate machines

We present these results to show that there is some latency at the client that is not directly due to server performance. The tests used the Message Burst Application.

Adobe

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

95011594 10/08