Technical white paper

Adobe® Flex™ Data Services 2: Capacity Planning

Adobe Flex Data Services 2 : Capacity Planning

Introduction

The purpose of this document is to help the reader better estimate what the hardware requirements for an Adobe Flex application using Flex Data Services 2 might be in a production environment.

The document gives an overview of capacity planning, then addresses capacity planning issues specific to Flex. The document then walks you through the load testing process using a baseline Flex application and presents performance test results for this application.

By comparing your application to the baseline application presented in this document and by using our performance test results as a guideline, you should be able to better estimate your hardware requirement needs.

It's important to keep in mind that every application differs in complexity and that the demands placed upon an application by its users will not be consistent from one application to the next. Application performance will undoubtedly vary across different application servers. For these reasons it is important to verify estimates with adequate performance testing using the hardware and software configuration you have chosen. It is important to do this testing prior to production release.

Capacity planning overview

Determining what hardware configuration will adequately meet the needs of your application is the process known as capacity planning. When an application is released into production, it is important to be confident that the hardware configuration that's chosen will allow it to function properly and meet the performance requirement goals that were agreed upon. These performance requirement goals are often called service goals or service level agreements. While these are usually defined by management or the business owners of a specific application, they will most likely reflect the end-users' expectations. The following are three simple examples of performance requirements:

- No single request can take more than 8 seconds to process
- The application must support up to 30,000 requests per day
- The application must support 1,000 concurrent users

Gathering accurate performance requirements is an important part of the capacity planning process. If you underestimate the production-level demands for your application, it may not perform or even function properly. This translates to unhappy users.

Application usage can vary over time—even hourly. If you don't accurately predict these fluctuations, your particular hardware configuration may not scale sufficiently to meet the new demands. Overestimating the demands placed on an application is also undesirable. Most organizations do not want to purchase more hardware than necessary.

Consider, for example, an application deployed to serve 60,000 users. Is it reasonable to expect that it would need to handle 60,000 simultaneous requests? The short answer is "probably not." The challenge then is to determine the number of simultaneous requests that reflects a realistic use case. That is, how many simultaneous requests will the application need to handle?

For the purposes of this document, let's assume that the application is intended to replace an existing one that there is accurate usage information for. Reviewing this information, it's determined that 10% of the users (on average) utilize the application daily. This translates to 6,000 users per day as compared with the total.

On further analysis, it's discovered that there are four hours during the day (peak hours) when the application is most often used. It's during these peak hours that the engineer needs to be certain that capacity can be met. Even if in the unlikely event that the application's users could manage to make one request every second during these peak hours, it's plain to see that 6,000 reqs/sec is an order of magnitude lower than the maximum of 60,0000 reqs/sec based on the total number of users.

Applications using real-time data have unique requirements compared with traditional web applications' HTTP request/response paradigm to request and deliver data. Flex Data Services 2 provides a robust messaging system that can support real-time data streaming and in-context collaboration of applications. An application that supports 6,000 users consuming a 100KB stream of data will require additional server capacity compared with an application that supports 6,000 users consuming 10KB streams of data. The performance profile of streaming data and collaborative applications is usually based on total data transferred rather than number of concurrent user connections.

It is also important to keep in mind that there are external factors that may impact the performance of your application. A high-performance application server along with a well-tuned database can dramatically boost the performance of your application. The same can be said for the application itself. A poorly written application will undoubtedly have a negative impact on the amount of hardware necessary to support it. Security is another factor to consider. Many applications use SSL (Secure Sockets Layer) for secure communications between the client and server. While often necessary, particularly for financial applications, it's important to keep in mind that it adds additional overhead. Some application server vendors report up to a 50% reduction in capacity when using SSL. Using a hardware accelerator (which offloads SSL processing to a component on the machine or to another machine entirely) can help improve SSL performance significantly.

To summarize, capacity planning requires an accurate definition of the performance requirements for your application. These requirements should include factors such as user count as well as expected response times for single or multiple requests (transactions). Once these are defined, the planning will then need to focus on finding a hardware configuration that will meet the application's needs. The last step in the capacity planning process involves performance testing of the application with the chosen hardware configuration for verification.

Flex architectural overview

The Flex Data Services 2 provides infrastructure for delivering and managing data delivered to rich clients developed with Flex Builder™ 2 or the Flex 2 SDK. Flex is deployed as a standard J2EE web application and is supported on a number of popular J2EE application servers.

Creating rich Internet applications (RIAs) requires a level of data management that goes beyond the traditional request/response model. Providing a richer, more expressive experience often requires more data-intensive interaction and introduces new challenges in managing data between the client and server tiers. RIAs enable users to work more independently using their own copy of data managed locally. This provides more intelligence and faster response times for the user. This model requires data synchronization between the client-side version of the data and the middle-tier data, which also needs to be synchronized with the database management system (DBMS).

Flex Data Services manages this important data synchronization process. It removes the complexity and error potential by providing a robust, high-performance data synchronization engine between client and server. It also can easily integrate with existing persistence and solutions to provide an end-to-end solution. The Flex Message Service inside Flex Data Services enables new categories of innovative applications to be delivered in the browser in a reliable and scalable manner while preserving the benefits of the traditional web deployment model. Flex Data Services provides publish/subscribe messaging infrastructure, enabling messages to be exchanged in real time between thin clients and servers. It allows thin clients to publish and subscribe to message topics with the same reliability, scalability, and overall quality of service as traditional thick-client applications.

Adding the Flex server to your production environment often does not require more hardware capacity. This is especially true for applications that do not deliver large amounts of data in real time through Flex Data Services messaging service. Performance analysis has shown that network bandwidth usage and server memory consumption for a Flex application and a similarly authored JSP application are identical where CPU usage is significantly reduced, as more processing—such as field validation, data formatting, and so on—can be offloaded to the client. However, applications with a high level of user interaction will see larger overall client performance gains with Flex compared with an HTML interface, as well as increased bandwidth and server memory consumption gains over time. The Flex application has a higher up-front cost in regard to network bandwidth because the entire application is downloaded at once, but it saves on data transfer through the network because the Flex application requires fewer requests to the server, such as navigating to a different page in the application or downloading redundant graphics for each page request.

Of course, applications that leverage real-time messaging have a different performance profile than traditional HTML-based web applications. For applications using real-time data delivery, often the gating performance factor is total network bandwidth usage or network card performance.

Flex Data Services provides a number of data channels for client-to-server communication. Each data channel is optimized for a specific kind of data transfer. The following is a description of each data channel that Flex Data Services 2 supports.

HTTP service (HTTP)

HTTP service allows data to be sent between server and client using the standard HTTP protocols. Data sent via HTTP will be encoded as plain text and included in the body of an HTTP request. HTTP is most useful for transporting data objects that are static (or mostly static) or rely on the browser's built-in HTTP communication capabilities. While there are ways to trick a web server and browser to communicate asynchronously (using polling or HTTP and so on) to simulate data push, the HTTP protocol is designed for request/response communication—which means that the HTTP data channel isn't well suited to asynchronous (or data push) data transfer.

Web service (SOAP/ HTTP)

Web services are the standard for SOAP-based data transfer. Web services transfer data as an XML/SOAP object using HTTP. Web services can only support request/response communication, and data payloads are sent and received as plain text XML objects. This can lead to performance problems when interacting with large data sets that require frequent updates.

Remote objects (AMF)

Action Message Format (AMF) is a communication protocol that serializes server-side Java™ objects into a compressed-binary form and transfers these objects via HTTP to a Flash® or Flex client. Once received, the serialized objects are deserialized into an ActionScript representation of the Java object. AMF is a highly efficient protocol for data communication because it requires less bandwidth than plain text protocols (like HTTP or SOAP).

Messaging (RTMP)

Messaging support is new in Flex Data Services 2. It is different from the above integration services because it is asynchronous and does not require the request of a client for a server message to be sent from the server to the client. Instead, a permanent connection is established, which is then used to stream data between server and client. Clients can subscribe to a certain information channel, and the server will send new information to all clients that have subscribed to the channel. Testing messaging applications that use messaging services is usually difficult because of their asynchronous nature and new data may or may not be available during a load test.

Application server tuning

Flex Data Services can be deployed on a number of J2EE application servers or servlet containers. It is recommended that you follow the application server vendor's tuning recommendations. An application server that is not tuned can cause a significant decrease in performance.

Application servers have different hardware requirements. Some are built to run large, enterprise-class applications, while other application servers are more suited to smaller intranet-type applications. Most application server vendors provide capacity planning guides. It is a good idea to review and follow their recommendations.

Flex scalability

In most cases, application servers will be configured to work with a load balancer, configured as a cluster, or some combination of the two. Since the Flex Data Services server is a J2EE WAR file, it will participate nicely in these environments. The baseline performance numbers shown later in this document, reveal that Flex will scale along with the underlying application server. There are areas where special attention can be paid that will reap additional performance gains. These are covered in the next section. The Flex Data Services messaging features will scale along with the external messaging system integrated with it.

MXML compilation

MXML compilation is CPU intensive. In most cases this will not be a factor because an MXML page will be compiled once by the server the first time it is requested. On subsequent requests the MXML page will be served from a cache. Developers may elect to precompile the application using Flex Builder 2 or the command-line utility that is included with Flex SDK 2.

Forcing the server to recompile an MXML page for every request is not recommended because this has a significant impact on the performance of your application. If there is a requirement to dynamically generate MXML, you should be careful that the number of unique versions of the application is kept to a minimum. Rather than dynamically generating MXML for each user, it is better to generate the MXML based on a user's role. That way, if it takes ten seconds to compile an application, the first user to request the application will experience a delay of ten seconds, but the next user will be served that application from cache. If you want to serve a different application for employees, managers, and executives, only the first request from a user in each of the three groups will experience a compilation delay; if you compiled a separate application for each user, each user would experience the delay.

Proxy performance tuning

One of the key features of the Flex Presentation Server is its ability to "proxy" service requests on behalf of the client. This proxy is similar in nature to an HTTP web proxy and is used to handle web service and HTTP service calls made by Flex clients. The Flex proxy accepts an HTTP service or web service request from the client application. It then forwards the request to the intended service, waits for a response, and returns the results.

While it is possible to bypass the Flex proxy, there are a number of benefits to using the proxy. The Flex proxy allows your application to access URLs that reside on different domains, adds an additional layer of security, and provides support for stateful services.

There is some overhead associated with using the Flex proxy. Requests that go through the proxy may take slightly longer than direct requests. Also, using the proxy will result in additional TCP/IP sockets opened on the application server machine.

This is to be expected since the Flex server is acting as a web client to the HTTP service or web service. This is true even if the target service resides on the same machine as the Flex server. It is always recommended that TCP/IP settings be optimized for the server machine running Flex.

Scalability test methodology description

Flex Data Services load testing was preformed using a series of test suites written by the Flex server performance team. The goals of these applications are to:

- Show round-trip data processing performance for small, medium, and large data payloads
- Show that Flex scales well under load for each of the data channels that Flex Data Services supports

These performance test suites test all of the Flex Data Services data access features including RPC services (web service, HTTP service, RemoteObject service), messaging service (using JMS), and data management service. Load testing of each of the Flex Data Services data channels is done in two ways. First, we measure the round-trip performance of each data service at various data payload sizes. Second, we measure the number of concurrent requests processed by a data service at increasing data user loads. The size of data payloads used in these tests are 24KB, 100KB, and 200KB.

Test tools used by the Flex performance team included the Microsoft® Web Application Stress Tool and the Flex Data Service Load Test Tool. Both of these tools are freely available and allow individuals to test their applications using the same methods and techniques used to create this report.

The Flex Data Services RPC services were load tested using the Microsoft Web Application Stress Tool. The tool was used to generate load against the server, and its record feature was used to create a test script. The record feature launches a browser window into which the application is loaded. Once the application is loaded into the browser, the user (tester) may begin to interact with the application, usually following a sequence of previously defined test steps. All network requests made to the server by the client are recorded by the stress test tool. When the test script is later executed, or "played back," the tool will resend these network requests to the server.

The messaging service and the data service were load tested using the Flex Data Services load-testing tool. This tool was written by the Flex performance testing team to allow for load testing using the RTMP protocol. RTMP is a custom protocol supported by Adobe Flash Player and does not go over HTTP, so it cannot be load tested using traditional web application stress test tools (like the Microsoft Web Application Stress Tool). The Flex Data Services load-testing tool works by running a small socket server, called the browser server, on a number of client machines. The browser server is responsible for launching web browsers on the client machine. Each web browser will run an MXML test application that then makes requests to the browser server. To increase load, the browser server can be used to launch more client browser instances. When the resources of the client machines running the browser server are fully utilized, more machines running the browser server can be added to increase the scalability.

Both of these load-testing applications are executed with no built-in "think time" for each client or virtual user. This means that one client thread or virtual user does not directly translate to a single user interacting with the application. The Flex Performance team estimates that a client thread represents approximately 10 actual users, so 10 client threads or virtual users would represent around 1,000 users interacting with the application simultaneously. This method of ignoring think time is preferred because accurately calculating think time is difficult and in most cases doesn't provide any greater level of accuracy.

For each server configuration, a test script was run to measure the following:

- The performance of a specific data service (RPC, data management service, messaging) for a given amount of data transferred
- The total number of concurrent requests that can be handled by the server with a specific amount of data throughput

Test system configurations

For all tests, the client-tier consisted of one Windows® machine running the MS Web Application Stress Tool or the Flex Data Services Load Testing tool. No data tier was used as the server-side components used in memory data objects. The Flex Data Services server was deployed along with an Adobe JRun 4 J2EE application server.

Server configuration

The server machine that hosted the Flex Data Services applications had the following specifications:

- HP® ProLiant DL380
- Microsoft Windows Server 2003, Enterprise Edition
- Dual-processor Intel® Xeon 3.06Ghz CPU
- 2GB of RAM

Client configuration

The client machines had different specifications, since they required only enough memory to open 20 browser sessions. Here is a typical specification:

- Microsoft Windows Server 2003, Standard Edition
- Dual-processor Intel Xeon 2.08GHz CPU
- 2GB of RAM
- Internet Explorer v6.0.3790.1830

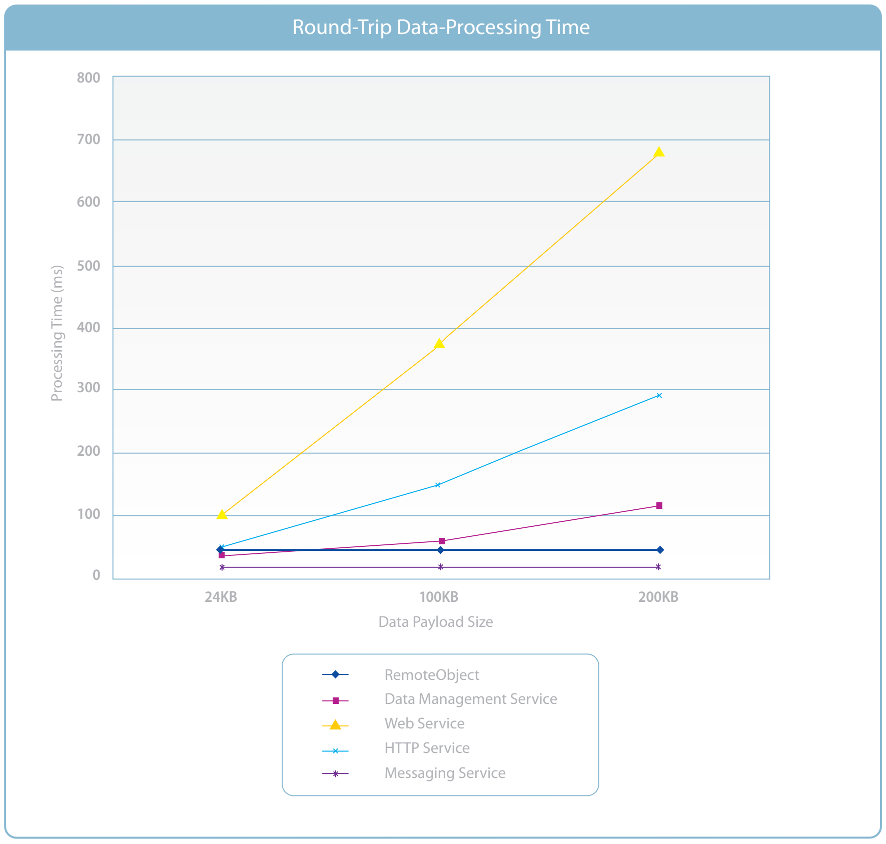
Network configuration

A 100 Base-T (100Mbps) network was used.

Baseline application performance results

Data service performance by channel

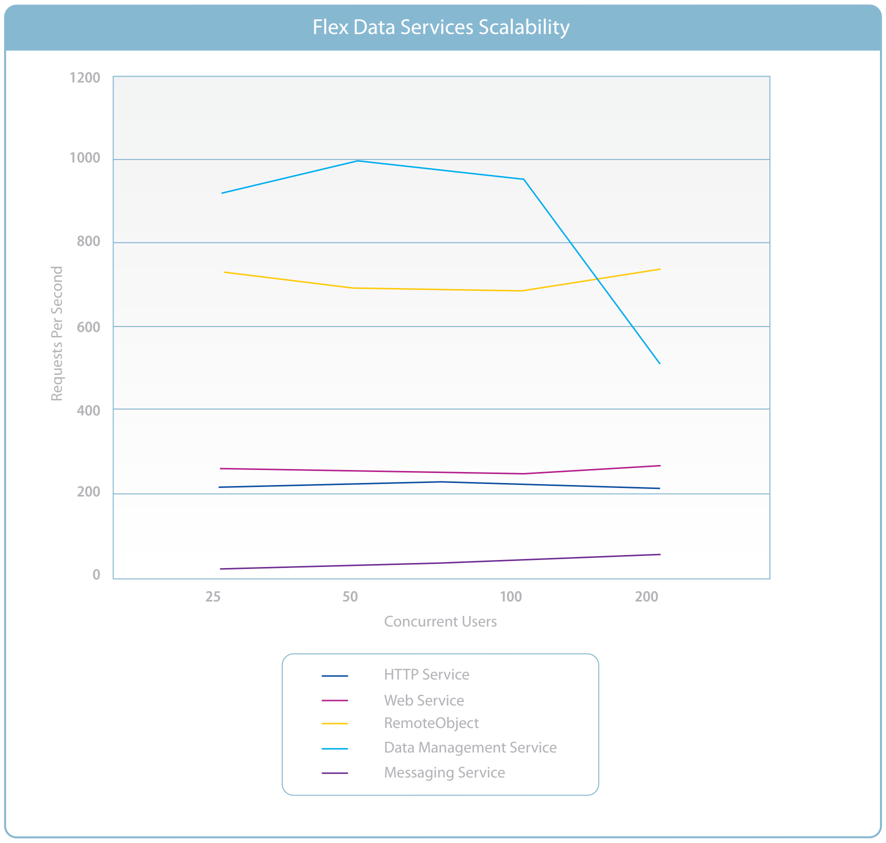
Data service performance by channel measures the performance of an atomic data service operation at a specific data payload size and user load metric—from the time the client sends data to the server to the time the server responds with an acknowledgement that the action is complete. This kind of performance test is very useful when trying to understand how your application responds to various data payload sizes. As the figure below illustrates, for small data payloads all of the Flex Data Services data service channels have similar performance characteristics. However, as data payloads increase in size, the processing time for web service and HTTP channels increases at a much higher rate than RemoteObject, messaging, or data management services.



Round-trip data-processing time for each Flex Data Services data channel

Data service scalability

Data service scalability measures the scalability (or number of users that a single Flex Data Services server can support) of individual data services given a data payload size. This test is useful in identifying the maximum number of concurrent users that can be supported by a specific application. As shown below, the highest performance is seen using RemoteObject, with data management service providing the second highest. Web service and HTTP service both support the lowest overall throughput at any data payload size.



Scalability for each Flex Data Services data channel

Sizing an application

Capacity planning is not an exact science. No two applications are exactly alike, and there are many factors that may impact performance. Once an estimate has been made with regard to hardware capacity, it will still be necessary to load test your application in order to validate this estimate. After load testing, additional tuning may be required in order for the application to meet the goals set for it. It still may be necessary to adjust your hardware requirements in order to meet your goals.

The data provided below is meant to showcase Flex Data Services performance capabilities under a variety of real-world circumstances. This data in isolation, however, is not particularly useful. To size your specific application, we recommend that you use the same methodology in this document applied to your application. Once you understand the performance of your data service, you can make an accurate estimate of how many servers you will need to support your application.

To size an application you need to answer three questions. First, what is the response time of your application-specific data services at your average data payload size given your hardware, application server, and databasee choice? Second, what is the maximum number of concurrent users that a single server instance can support under load? Third, what is the maximum number of users your application will need to support (remember to take into account peak usage patterns for your specific application)?

Once you answer these three questions, use the following formula to determine how many Flex Data Services servers your application will require.

$$\text{Total Flex Data Services servers} = \frac{\text{Peak number of application users}}{\text{Number of concurrent users at a specific response time}}$$

Conclusion

Load and performance testing is a must for any application that serves more than a handful of users. It will help you ensure you have the right amount and type of hardware and that you are meeting your users, expectations—so that internal workers are productive and external customers continue to use your application. This guide shows you a number of characteristics related to the scalability of Flex applications, the ability of Flex to scale linearly, and its ability to help you more efficiently use your server resources. 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.