WHITE PAPER

Capacity Planning for Adobe's Macromedia Flex™ 1.5

Alex Glosband and Bob Tierney

April 2006

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

04/06

Content

Introduction

The purpose of this document is to help the reader better estimate what the hardware requirements for Adobe's Macromedia® Flex 1.5 application 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, they will most likely reflect the user's expectations. The following are two simple examples of performance requirements:

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

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 to the total.

On further analysis, it's discovered that there are 4 hours during the day (peak hours) when the application is most often used. It's during these peak hours where the engineer needs to be certain that capacity can be met. Even if, in the unlikely event, 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.

It is also important to keep in mind that there are external factors which 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.

Capacity Planning and Flex

Flex Architectural Overview

Flex is a platform for creating rich interfaces for web applications. The Flex server-side application can be deployed as a standard J2EE web application and is supported on a number of popular J2EE application servers. Those familiar with the ASP/JSP model understand that in order to produce an HTML page, the server side code must first gather the necessary data from the business tier, and then embellish that data with HTML tags which determine how the data is presented. All this translates to server CPU cycles.

Flex departs from this model by providing a client technology (Macromedia Flash™ from Adobe) for the user interface, relying on the server solely for the data. The user interface is authored using a text based markup language (MXML) and compiled into a binary format (SWF). When a user requests an application via an URL, the SWF file is transferred to the client where it begins to execute. All requests for data are handled by the server and subsequently transferred to the client. This separation of processing allows the client to handle simple tasks such as field validation, data formatting, sorting, and filtering–thus freeing up valuable server CPU cycles.[1]

[1] Flex Performance Brief: A Comparison of Flex and Java Server Pages Applications: http://www.macromedia.com/devnet/flex/articles/performance_brief.html

Adding Flex to your production environment often does not require more hardware capacity. 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, etc. can be offloaded to the client.[2] However, applications with a high level of user interaction will see larger overall client performance gains with Flex compared to 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 saves on network because it requires fewer requests to the server, such as navigating to a different page in the application or downloading redundant graphics for each page request.

Application Server Tuning

The Flex web application can be deployed on a number of J2EE Application Servers or servlet containers.[3] 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.

Capacity Planning for BEA WebLogic
http://edocs.bea.com/wls/docs81/capplan/

Capacity Planning for IBM WebSphere
http://www.redbooks.ibm.com/abstracts/SG246198.html

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 server-side application is a J2EE war file, it will participate nicely in these environments. That's one of the benefits utilizing a J2EE architecture.

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.

---

[2] Flex Performance Brief: A Comparison of Flex and Java Server Pages Applications:
http://www.macromedia.com/devnet/flex/articles/performance_brief.html

[3] See the Flex Install Instructions for a list of supported application servers:
http://www.macromedia.com/support/documentation/en/flex/1_5/install.html.

MXML Compilation

MXML compilation is CPU intensive. In most cases this will not be a factor as 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 pre-compile the application using the command line utility that is included with Flex.

There may be scenarios where dynamically generated MXML is necessary, requiring the compiler to be invoked at request time. This can be implemented by utilizing the Flex JSP Tag Library.[4] The Flex server caches each unique version of the application that is generated.

Forcing the server to recompile an MXML page for every request is not recommended, as 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. For example, rather than dynamically generating MXML for each user, it is better to generate the MXML based on a user's role. For example, if it takes 10 seconds to compile an application, the first user to request the application will experience a delay of 10 seconds, where the next user will be served that application from cache, which takes 10 fewer seconds. If you want to server a different application for employees, managers, and executives, only the first request from a user in one 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 server application 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.

It is possible to bypass the Flex proxy, though it means not taking advantage of the useful services it provides. 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.

The following Macromedia tech notes for provide some useful OS specific performance tuning tips, including optimized TCP/IP settings for both Windows and Solaris.

[4] See the Developing Flex Applications guide in the Flex documentation for more info on using the Flex tag library: http://download.macromedia.com/pub/documentation/en/flex/15/flex_dev_apps.pdf

Windows
http://www.macromedia.com/cfusion/knowledgebase/index.cfm?id=tn_17277

Solaris
http://www.macromedia.com/cfusion/knowledgebase/index.cfm?id=tn_18229

Load Testing

Capacity planning is not an exact science. No two applications are exactly alike, as there are many factors which 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 baseline application described below was load tested using the Microsoft Web Application Stress Tool.[5] This tool is available for free, making it relatively easy for the reader to reproduce this test. There are also many commercial tools available for load testing. In many cases, these tools provide additional features such as the ability to randomize test data and customize test scripts using scripting languages.

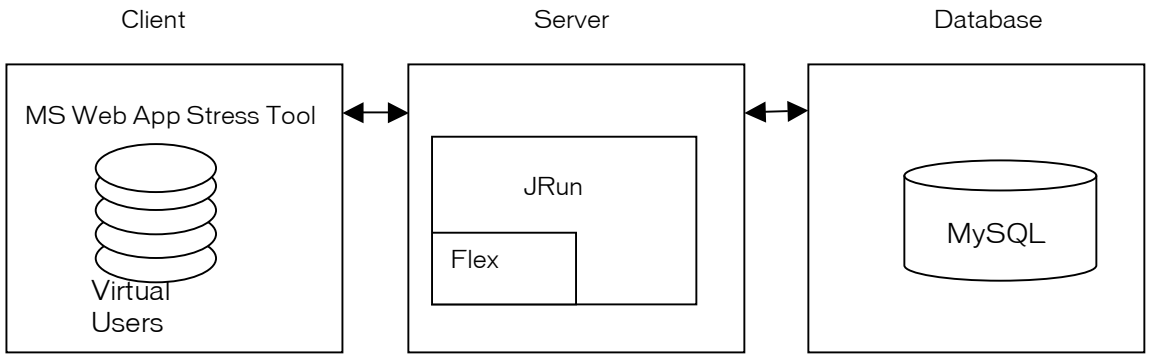


Figure 1: Three-tier configuration used for testing

[5] The MS Web Application Stress Tool can be downloaded from the following URL:
http://www.microsoft.com/technet/archive/itsolutions/intranet/downloads/webstres.mspx

Baseline Application

The RestaurantFinder application is a sample Flex application written by Christophe Coenraets.[6] The application provides a means for the user to search for a restaurant in the Boston area. It's an RIA that allows the user to select an area on a map. Once selected, the system will return a list of restaurants for that area. Users may enter reviews if they wish.

This application employs a three tier architecture which includes a distinct client-tier, application-server-tier, and data-tier. The business layer is made available via a Service-Oriented Architecture (SOA).

All business objects are exposed using well-defined service methods.[7] These service methods are responsible for all database interaction such as data retrieval, updates and the mapping of data to objects which are returned to the Flex client. The benefit of this approach is to provide and abstraction between the business and presentation layers.

The RestaurantFinder application has two modules, an End User module and an Administration module. Each module is a distinct application, represented by its own top-level MXML page. For this document, only the End User module was tested.

A few modifications were made to the application for testing purposes. The Hypersonic database was replaced by the MySQL database server. The ConnectionManager class, used for database connectivity, was modified to take advantage of connection pooling provided by the JRun™ software application server. Additionally, the RestaurantService class was changed to retrieve the same list of restaurant reviews from the database, regardless of which restaurant was selected. This was done to accommodate the fact that all test threads write the same review to the database, and then immediately request all reviews for that restaurant. Without modifying the application, the performance results would have been greatly skewed.

The MS Web Application Stress Tool was used to generate load against the server. The tool's 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 following scenario was used to test the RestaurantFinder application. The test results are measured in transactions per second, where a transaction is defined as the completion of all the requests in the scenario.

1. From a browser, a user requests the "finder.mxml" application (RestaurantFinder). A SWF (flash application) is downloaded* and begins to execute within the Flash player. Once running, the first Remote Object call is made to the Flex server, returning a list of all Restaurants in the database (name, city, phone number). Once the list is complete, an additional Remote Object call is made to retrieve detailed information for the first restaurant in the list.

[6] More information about the RestaurantFinder application, including a link to download the application can be found on Christophe's blog: **http://coenraets.com/viewarticle.jsp?articleId=85**

[7] In the RestaurantFinder, the service interface for the business layer of the application is contained in the RestaurantService class. The source code for this class is distributed with the application.

*Note: When the test script is played back, we put in assumption that the SWF is downloaded only 25% of the time. This allows for repeat visits to the site, which would result in the SWF loading by way of the browser cache. In your application, the number of unique SWF downloads, or how often it is served from browser cache depends on the type of users you have for your application, their browser settings and other variables you should consider when making your testing assumptions.

2. The user will select a different restaurant, in this case "Blue Ginger", causing an additional Remote Object request to the server. An object containing the details for this selection is then returned to the client. Clicking on the "Reviews" tab forces the application to request the review information from the server (see figure 2 below).

3. The process is reversed by choosing a different restaurant from the list (Argana). The "General Info" tab is selected, forcing a call to the remote object for detailed information once again.

4. Finally, the user writes a review for this restaurant and posts it via a Remote Object call. The service method which is invoked writes the review information back to the database.

Figure 2: RestaurantFinder Application

Configuration for RestaurantFinder Baseline Application

A three-tier configuration was employed for testing both Solaris and Windows. For all tests, the client-tier consisted of two Windows machines running the MS Web Application Stress Tool. The data-tier consisted of a Windows machine running the MySQL database. The RestaurantFinder application was deployed on JRun 4 from Macromedia Inc. As you will see below, a number of different configurations were used for the server-tier.

For each server configuration, the test script was run with an increasing number of virtual users until the test configuration was saturated - meaning that the number of transactions per second either leveled off or began to decrease. No additional "think-time" was added to the test script for load testing.

"Think-time" is sometimes added to a test script to allow for the virtual user to more closely represent the actions of a real user (pauses, delays, etc.). Calculating "think time" is very difficult to approximate, which would explain why this technique is not often used.

It's generally assumed that a virtual user represents a greater number of actual users, usually by about a factor of 10. This means that 100 virtual users would be roughly equivalent to 1000 actual users.

Server Configurations

Solaris

The Solaris™ platform was used to test Vertical Scalability. The goal of Vertical Scalability is to determine how the application scales as CPUs are added to the machine. Tests were run against a 4 CPU Solaris machine with 1, 2, and 4 CPUs enabled. On Solaris, CPUs can be programmatically enabled and disabled using the "psradm" command.

Host: Solaris 9

CPU: 4 x 900MHz v

Memory: 16GB

Disk: Two 36GB hard drives

Software: Flex 1.5 running on JRun Updater 4 with Sun JDK 1.4.2_07

Windows

The Microsoft® Windows® platform was used to test Horizontal Scalability. The purpose of Horizontal Scalability testing is to determine how an application scales as server nodes are added to a cluster. Testing was done with a single non-clustered instance of JRun, a cluster of two JRun instances running on separate machines and a cluster of four JRun instances running on four separate machines.

In all cases, one of the test machines was configured with both IIS 6.0 and a JRun instance. The other server machines only had an instance of JRun. The Flex web application was deployed to each of the JRun instances.

IIS uses a native web server connection module to communicate with JRun. When multiple JRun servers are used in a cluster, IIS load balances between the JRun servers using a load balancing algorithm to determine which server to send a request to. In our testing the round-robin load balancing algorithm was used which simply rotates through all available servers in the cluster.

Host: Windows 2003 Server

CPU: 2 x 2.8GHz

Memory: 2GHz

Disk: 67GB

Software: Flex 1.5 running on JRun4 Updater 4 with Sun JDK 1.4.2_07

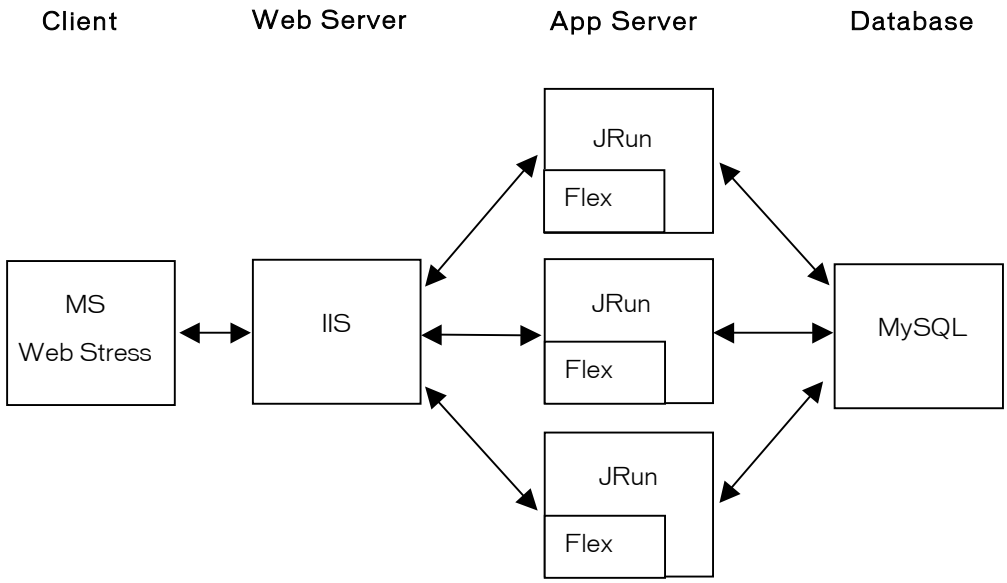


Figure 3: Clustered configuration

Database Configuration

A MySQL database was used for the data-tier. The MySQL Server Instance Configuration Wizard was used to configure the database with the following options: The database was configured as a "Dedicated MySQL Server Machine"; the database was configured to use the Non-Transactional MyISASM storage engine (the sample application does not use transactions) and the OLTP option was selected to allow for a limit of 500 connections.

Host: Windows 2000

CPU: 2 x 2.8 GHz

Memory: 2 GB

Disk: 67 GB

Software: MySQL 4.1.10a-nt

Client Configuration

Two client machines with the following configuration were used to generate loads against the server.

Host: Windows Server 2003

CPU: 2 x 3.0 GHz

Memory: 2 GB

Disk: 67 GB

Software: MS Web Application Stress Tool

Network Configuration

A 100 Base T (100 mbps) network was used.

Baseline Application Performance Results

Vertical Scalability

Looking at the benchmark test results in **figure 4**, you can see that for each CPU configuration, throughput increases as the number of virtual users is increased until a leveling off point is reached. At this point the Flex web application is saturated. It is important to note that once the application has become saturated, the number of transactions per second stays relatively level. There is no dramatic drop-off in the number of transactions the Flex web application can handle as additional load is added.

With the single CPU machine, max TPS (transactions per second) was reached with 50 virtual users. With the dual CPU machine, max TPS was not reached until 75 virtual users. This means that on the two CPU machine, the Flex web application was able to handle an additional 25 virtual users before becoming saturated. A virtual user generally represents a tenth of an actual user, where 100 virtual users is roughly equivalent to 1000 actual users.

With four CPUs, the Flex web application became saturated with roughly the same number of virtual users as with two CPUs but the maximum number of transactions the application was able to handle was higher. Running on the four CPU machine, the Flex web application showed an approximate 16% increase in throughput over the 2 CPU machine and close to a 50% increase in throughput over the single CPU machine. This shows that Flex scales well as CPUs are added to the machine although there is a diminishing rate of return. The benefit of moving from two CPUs to four CPUs was smaller than the benefit of moving from a single CPU to two CPUs.

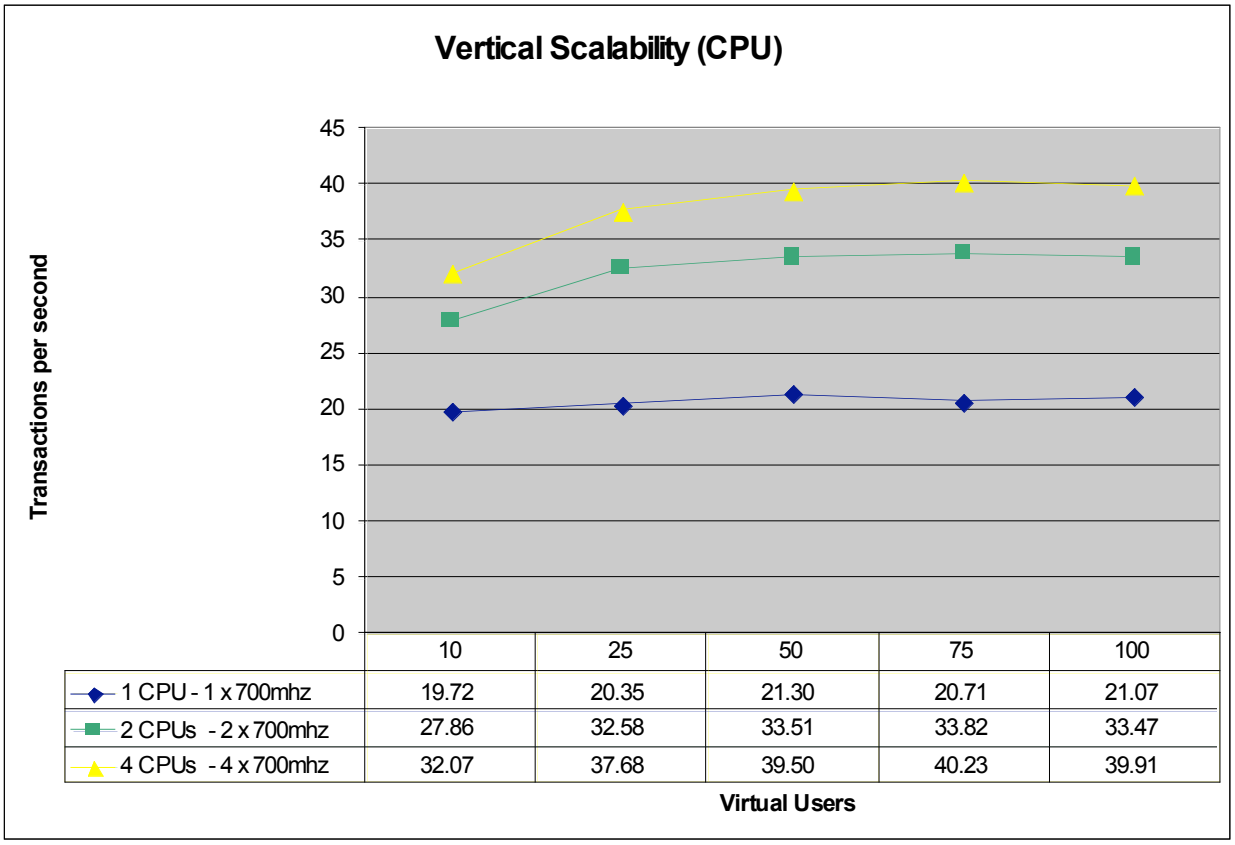


Figure 4: TPS (transactions per second) on Solaris with 1, 2 and 4 CPU configuration

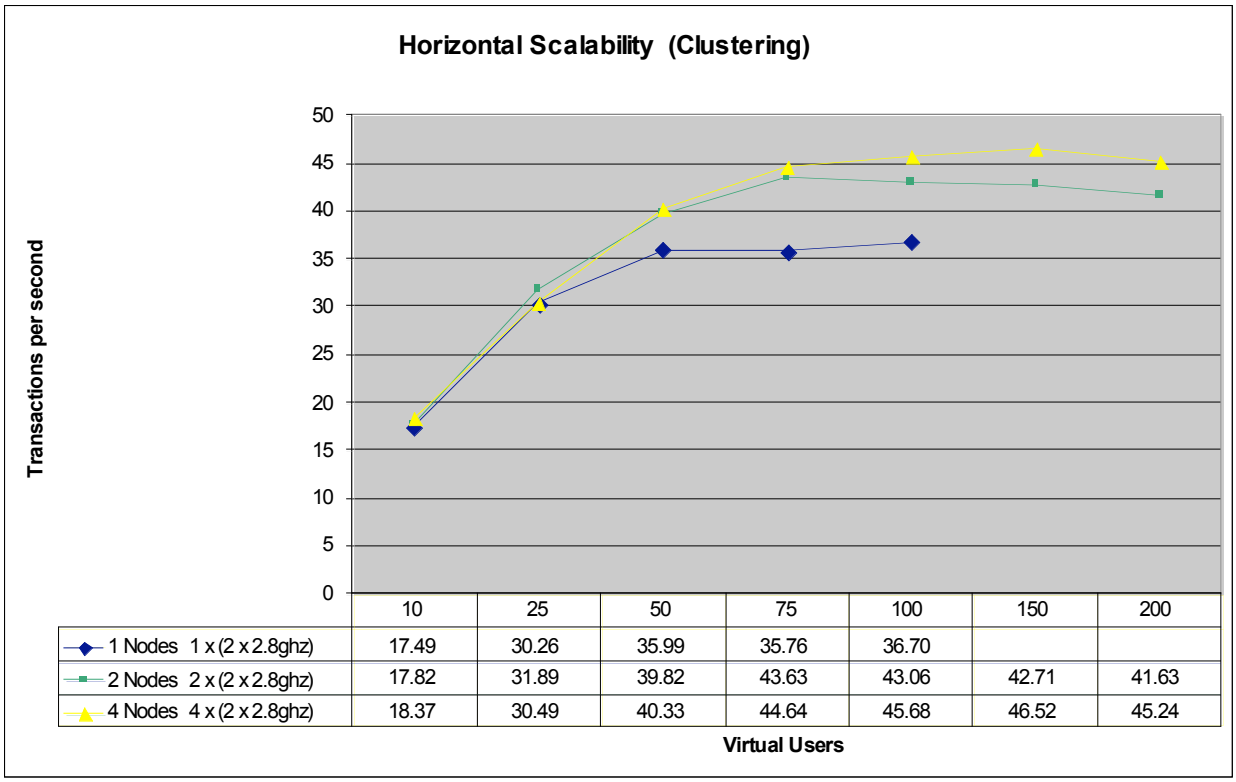
Horizontal Scalability

The benchmark test results for the clustered configurations in **figure 5**, indicate that there is minimal overheard associated with using the Flex web application in a cluster. Under lighter loads, throughput with the two and four node cluster was roughly the same as the non-clustered server. However, under heavier loads throughput was significantly higher for the clustered servers.

For all configurations, the number of transactions per second increases as the number of virtual users is increased until the server becomes saturated. Once the server has been saturated the number of transactions per second stays relatively level, even as the number of virtual users is increased. There is no dramatic drop-off in the number of transactions the Flex web application can handle as additional load is added.

The test results show that the Flex application scales well in a clustered environment.

The two server cluster showed an almost 16% increase in throughput over the non-clustered server, while the four server cluster showed a 6% increase in throughput over the two server cluster. Additionally, the four server cluster was able to handle an additional 75 virtual users before becoming saturated than the two server cluster. The two sever cluster and the non-clustered server became saturated at roughly the same point although the two server cluster had much higher throughput. A virtual user generally represents a tenth of an actual user, where 100 virtual users is roughly equivalent to 1000 actual users.



	10	25	50	75	100	150	200
1 Nodes 1 x (2 x 2.8ghz)	17.49	30.26	35.99	35.76	36.70		
2 Nodes 2 x (2 x 2.8ghz)	17.82	31.89	39.82	43.63	43.06	42.71	41.63
4 Nodes 4 x (2 x 2.8ghz)	18.37	30.49	40.33	44.64	45.68	46.52	45.24

Figure 5: TPS (transactions per second) on Windows with 1,2 and 4 server nodes

Conclusion

Load and performance testing is a must for any application that serves more than a handful of users. It will aid you in ensuring you have the right amount and type of hardware and that you are meeting your users expectations–ensuring 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, Flex's ability to scale linearly, and Flex's ability to allow you to 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. For more information on turning Flex applications, Flex performance data, and other Flex topics, please see the following resources:

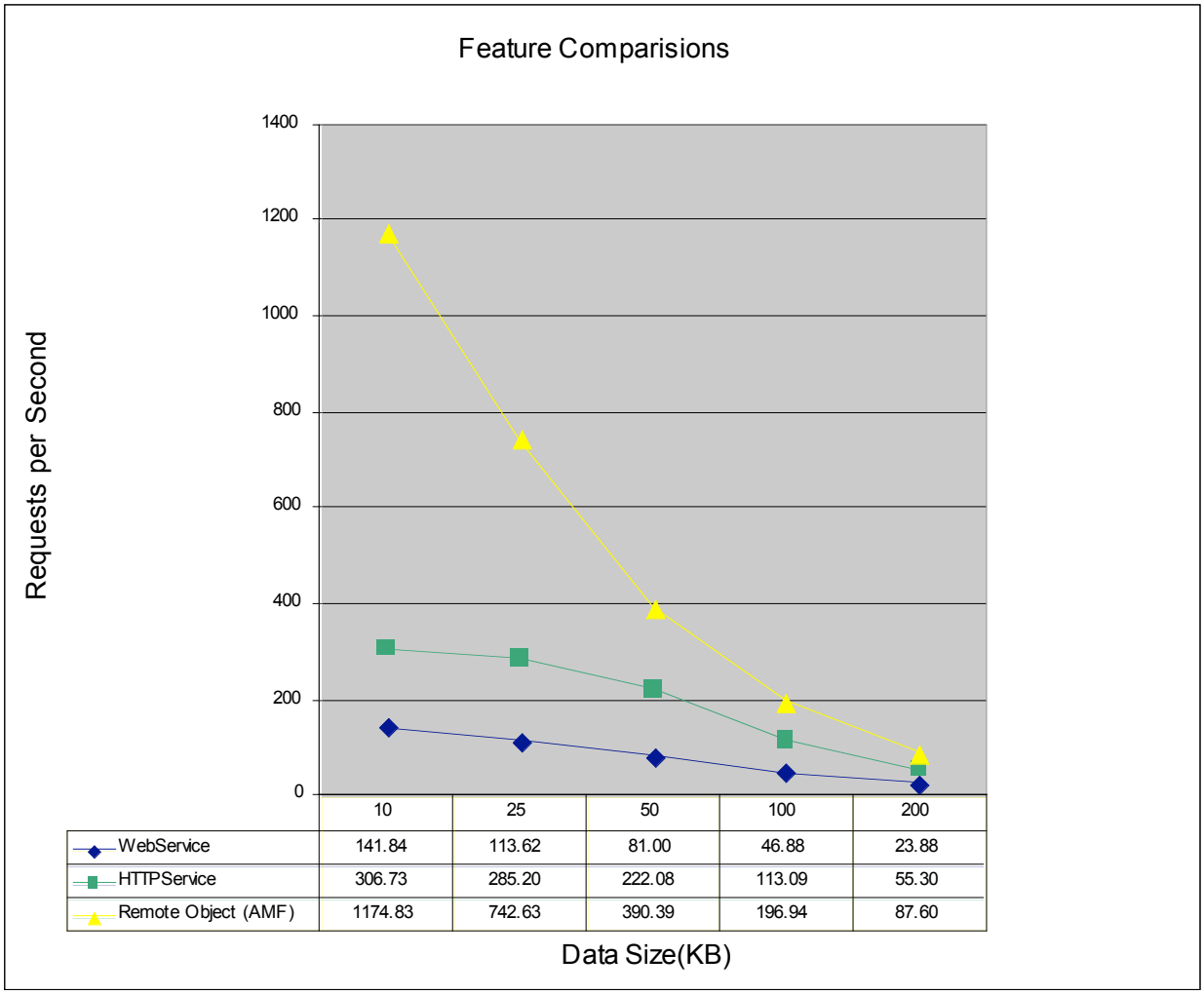
Flex Developer Center

Flex Evangelist's Blog

Flex Example Applications

Appendix

The following chart shows a performance comparison of the different data services features in Flex. Each of the data services were tested with a variety of data packet sizes. As you would expect, as the size of the data packet increases, the number of packets that can be transmitted in a given time period decreases. Results here are measured in requests per second. The Remote Object feature, which uses the binary AMF protocol had the best performance by far, especially when using smaller data packet sizes.



	10	25	50	100	200
WebService	141.84	113.62	81.00	46.88	23.88
HTTPService	306.73	285.20	222.08	113.09	55.30
Remote Object (AMF)	1174.83	742.63	390.39	196.94	87.60