Optimizing WebCenter Interaction Performance

Introduction
Web application performance can mean different things to different people. To IT administrators, it boils down to how many boxes they need to run their system. Implied in this question are stability and responsiveness. They generally want to run an application on as few servers as possible to reduce license and maintenance costs. To end users, performance means how fast their requests are fulfilled. Answering these requires analyzing the various aspects of an application that can affect scalability and performance.

Optimizing a portal deployment can have multiple benefits. From the end user perspective, they will get a more responsive experience that will make them generally more receptive and accommodating to portal initiatives. For IT administrators, they can potentially increase portal performance and stability and reduce the number of machines needs to meet their portal initiatives, which can reduce overall maintenance and even license costs.

This paper will discuss the WebCenter Interaction architecture, how it operates, and how different components affect the scalability, performance, and perceived performance of the system. It is not a complete discussion of performance and scalability however. That would require a much longer discussion and knowledge of the unique characteristics of a particular deployment. Instead, it is intended to provide enough information to start companies thinking about their performance needs to optimize their deployments of the WebCenter Interaction.

There are two keep points to keep in mind when reading this paper. First, all of the performance data reported on the charts and graphs in this paper are based on un-customized system in a controlled laboratory environment with approximations of real world conditions. Since nearly all deployments are customized to some degree and the hardware and network environments are different, the results should be used as generalizations of system performance and not of expected system performance in an on-site deployment. To maximize system performance or optimize the use of system resources, the system should be evaluated on-site if possible.

Second, it is not necessary to perform rigorous performance tests for many deployments. Generally it is better to overestimate computing needs to ensure performance so there may be unused system resources for system that has not been optimized. This paper is intended for companies that wish to optimize their systems as much as possible.

Perception of Performance
Before discussing factors that affect performance, it is important to note that actual performance and perceived performance of an application are not necessarily the same thing. For example, refreshing an entire web page is generally perceived as having a less-rich user experience than a page that has components with in-place refresh even if refreshing the entire page shows content faster than the in-place refresh. This is similar to what happened in the early days of the browser wars between Internet Explorer and Netscape. While both browsers finished generating pages fairly evenly, Internet Explorer was quicker to display something initially even if it took the same time or longer to finish displaying the entire page as Netscape. This led users and even reviewers to claim that Internet Explorer was a higher performance browser.

Portals that use AJAX or other user UI technologies to provide a richer UI experience can often increase the perception of performance by end users. In the case of a portlet using in-place refresh, the portal page containing the portlet is displayed first and then the data in the portlet is displayed. If the request is made directly to the portlet, only the portlet is changed and not the rest of the portal page. The tradeoff is that creating a richer UI generally requires more infrastructure components which could affect page size and download times. However, most end users seem to prefer the richer, more dynamic interfaces even if they are (somewhat) slower overall than simply refreshing the entire page.

Definitions

Because it seems everybody has a different definition of performance and scalability, this paper will use those terms as defined in this section.

Performance

In this paper, performance will be used to mean the speed at which a request it fulfilled. In that context, performance and scalability are not necessarily the same thing since even highly scalable applications may be slow in returning content.

Application Scalability

Scalability is the ability of an application to function well when it (or its context) is changed in size or volume. There are several different measures of the scalability of an application.

- **Vertical Scalability**: Vertical scalability is the ability to effectively utilize machine resources as the power of a machine increases. For example, how a system responds to increasing the amount of RAM on a machine or the speed or number of processors. Generally utilization does not scale linearly with the power of machines.

- **Horizontal Scalability**: Horizontal scalability is the ability to effectively utilize resources as the number machines increases. For example, instead of making a machine more powerful by adding more RAM or using faster processors, another machine of similar specification is added to the pool. If an application is properly architected, utilization can scale closer to linearly with the increased number of machines than with vertical scalability up to a certain point.

Platform Scalability

Another factor to consider in any discussion of scalability is the scalability of the platform on which the application is running. This includes the operating system, the application server, Java Virtual Machines (JVMs), and anything else that application being considered requires for operation. Platform scalability has a direct impact on the vertical scalability of any applications that run on it since the platform itself must also be able to effectively utilize increasing resources. A platform is also an application after all.

Some applications can make up for the lack of platform vertical scalability by being horizontally scalable. This is similar to parallel processing strategies where instead of using a single very powerful processor, an application utilizes many less powerful processors. For example, the SETI@Home project uses the spare capacity of home PCs instead of a single supercomputer.

Page-Load Time

Page load time is simply how long it takes for an end-user request is fulfilled in the browser. While page load time is not a strict measure of scalability, it is affected by application scalability and it is what the end user notices the most. Even applications that are highly scalable in the classic sense can seem slow to an end user because of issues such as page size or bandwidth limitations.

WebCenter Interaction Architecture

In order to maximize the performance of an application, it is important to understand how the application works. The WebCenter Interaction is architected to be a scalable platform to both serve portal pages and be a platform for creating new applications.

Distributed Processing

One method of improving the scalability of an application is to use distributed processing. This is different than horizontal scalability in that each machine performs a different task. In horizontal scalability, each machine would perform the same task. Figure 1 below shows the base WebCenter Interaction and its distributed architecture. Each component has a specific purpose.

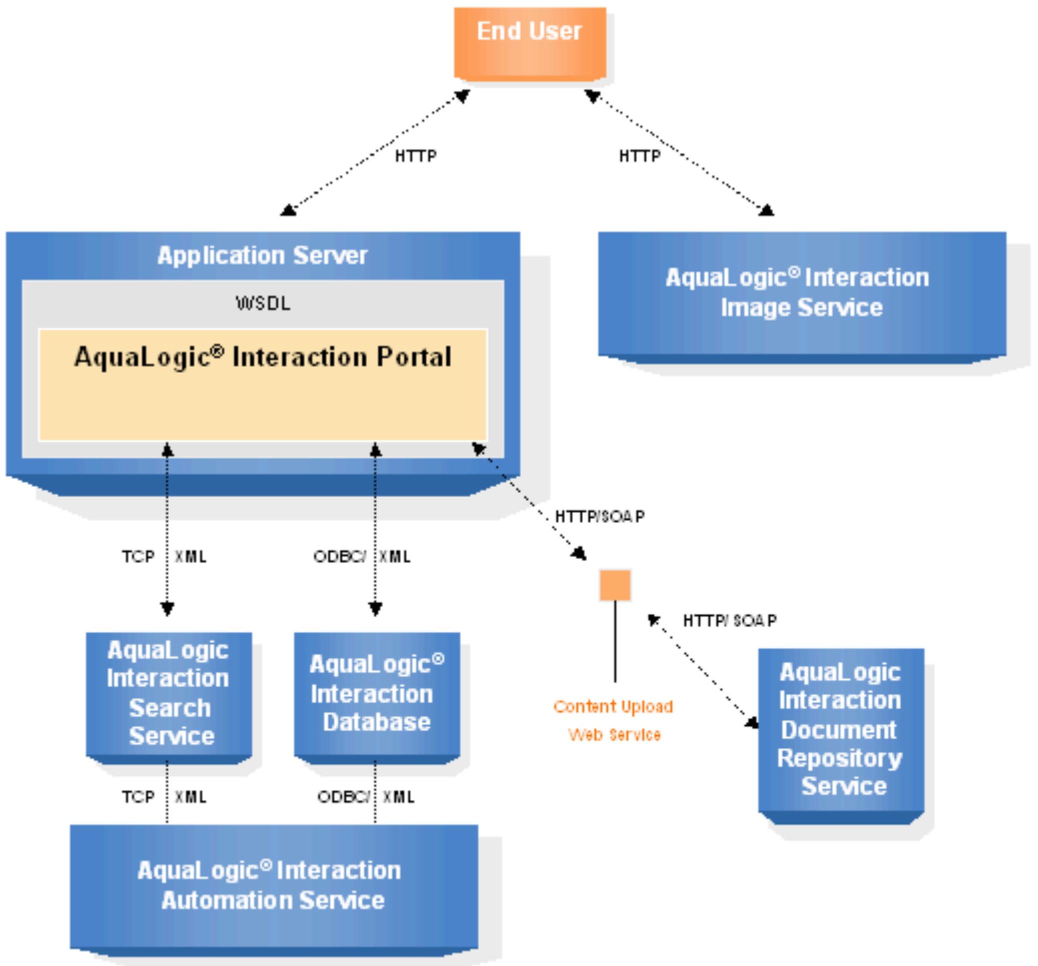
End User
HTTP
HTTP
Application Server
WSDL
AquaLogic® Interaction Portal
AquaLogic® Interaction Image Service
HTTP/SOAP
TCP XML
ODBC/ XML
HTTP/ SOAP
AquaLogic Interaction Search Service
AquaLogic® Interaction Database
Content Upload Web Service
AquaLogic Interaction Document Repository Service
TCP XML
ODBC/ XML
AquaLogic® Interaction Automation Service

Figure 1: WebCenter® Interaction Base Portal Distributed Architecture

Distributed processing has several benefits.

- **Performance**:
  - Since the total processing requirements are spread over multiple servers, load on any one service does not affect the performance of any of the other services. For example, if the Automation Service is performing a resource intensive import, end users will not notice because the portal servers are separate and therefore not affected by the resource constraints of the Automation Service.
  - Each server can be optimized for its specific function. For example, the WebCenter Interaction Automation server does not host a portal server and therefore does not need an application server. Not installing or enabling an application service can free resources for other processes.

- **Security**: End users only need to access the portal and Image Service(s). All other components can be placed behind the firewall to increase the security of the system. The WebCenter Interaction administrative portal servers can be secured separately from the end-user portal servers to increase the security of the system.

- **Fault Tolerance**: Should any component undergo a catastrophic failure, only that single server is disabled. The rest of the components will continue to operate normally. This also makes maintenance easier since patches and upgrades can be applied to specific areas without affecting the entire system.

- **Total Cost of Ownership**: The system can scale in the direction that is necessary. Only specific servers need to be upgrade has the scope of the deployment grows. For example, if the size of the portal audience doubles, only the number or capacity of the portal servers needs to be increased. Other components such as the Automation Service or document repository do no need to be upgraded. Lower-power machines can also be used for those components since they do not the performance that will be noticed by end users.

The WebCenter interaction also uses remote web services for integration components to distribute processing of portal content. Adding more integration components does not introduce any more strain on the portal servers since the portlet processing is performed on a separate server. Using remote portlets also enjoys the same security, fault tolerance, and cost benefits as the main system itself.

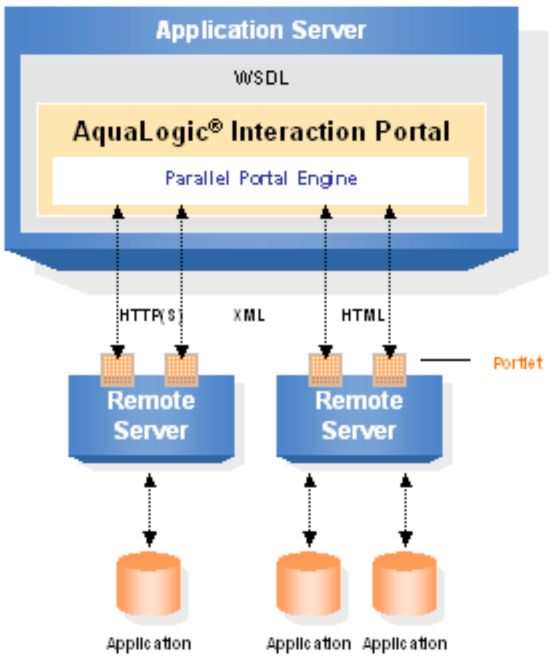

Figure 2: WebCenter® Interaction Remote Web Services Architecture

While it is recommended that all portal components be distributed, the components can be co-located on the same machine to reduce the total number of servers as well. Single-box installations are common for testing and development purposes. In certain scenarios, it is possible to combine various components to reduce the total number of servers. The portal components that should be combined depend on a variety of factors including portal usage, security considerations, and performance requirements.

Horizontal Scalability

While each component in the portal can handle a large number of requests, certain key components can be scaled horizontally as well.

- **Portal and Administrative Portal Server**: Administrative Portal Servers can be load balanced using standard load balancing tools. The portal servers will scale linearly up to the limits of the database server. Fault tolerance can be achieved by also intelligently configuring the High Availability feature.

- **Image Service**: Image Services can be load balanced using standard load balancing tools. Since the Image Service simply serves files and does not connect to the portal or the portal database, there are no limits to the horizontal scalability of the Image Service. On large deployments of many web applications, using dedicated image servers separated from the main application can increase scalability of the entire system by up to 20%.

- **Automation Service**: Automation Services are load balanced through the WebCenter Interaction itself. No external load balancing is needed. Automation Service jobs are designed to run in batches to limit the resource draw on the database. Even the largest deployments generally utilize only a few Automation Services. The actual number that is needed depends on content and update requirements. A single CPU can index at least at least one document per second. This translates to approximately 85K per day. However, there are other operations such as update and user synchronization operations that need to be performed. Many companies also prefer to run as many automated operations at night to reduce load on the database and search index while end users are actively using the portal. This may increase the number of Automation Services that are necessary to meet the performance requirements of the portal.

  While advanced disk arrays are not necessary, it is recommended that the Automation Service have at least 3GB of free disk space to store the temporary local copies of crawled file during the indexing process.

- **SOAP/API Service**: API Services can be load balanced using standard load balancing tools or multiple instances can be used concurrently with different portlets connecting to different instances of the API Service that are part of the same portal deployment.

The components mentioned above will scale linearly up to the capacity of the portal database. The actual number of each component depends on the specific requirements of each deployment. For example, deployments with a large number of users but a smaller number of administrative functions may have 20 portal servers but only 1 Automation Service. Deployments with a small number of users but a large number of administrative functions may have 2 portal servers and 5 Automation Services.

The following services can be scaled horizontally with a few restrictions.

- **Search Service**: The search collection can be divided into 16 total nodes. These nodes can be divided into 16 partitions for scale or fault tolerance. Each partition contains a separate part of the overall search collection. For example, for fault tolerance for a small collection, the system could be configured to have 1 partition with 2 nodes. Each node would contain the same copy of the search collection since they were in the same partition. For maximum scalability, the system could be configured to have 16 partitions with 1 node each. In this case, the index in each node would be different from every other node since each node is in a separate partition. Each node can contain up to 10GB of index data for 32-bit operating systems and 100GB for 64-bit (not Linux) operating systems.

  End-user search performance can be affected by very large search collections in a single node. Remember that the portal also indexes not just documents but every object in the portal including users and administrative objects. Documents tend to affect the size of the search collection more than other types of objects because document text is indexed in addition to the properties of the object. End-user search performance can be increased for large collections by splitting up the collection over several nodes.

- **Document Repository**: The Document Repository service can be load balanced just like any web application. However, there can only be one instance of the document store itself. This is the location of the actual files, not the service that passes or retrieves the files. Since the document store is a simple file store, standard fail-over or NAS can be used.

In addition to the main portal components, integration components such as portlets can be horizontally scaled as well. This is dependent on the portlet itself and is not limited by the WebCenter Interaction.

Parallel Portal Engine

A key mission of portals is integrating external applications and assembling them into a variety of composite applications. The WebCenter Interaction uses the Parallel Portal Engine (PPE) to maximize the speed and performance of the aggregation.

As the name implies, the Parallel Portal Engine communicates with portlets in parallel to reduce the overall time required to aggregate portlet content. The time required to aggregate content is not limited by the total number of portlets displayed on a page. If the portlets were aggregated in serially, total aggregation time would be the sum of the time required to import the content from each portlet. If each portlet server takes 1 second to create a portlet and there are 10 portlets to a page, a portal would take 10 seconds just to get the content in addition to the time required to render the final page. The WebCenter Interaction parallel portal engine would only take 1 second to aggregate all the portlets. This is true regardless of the number of portlets on a page.[1] The parallel portal engine can also bypass unresponsive portlets to guarantee a certain page response time for the rest of the portals. One failed portlet will not prevent the use of other portlets on a page. The parallel portal engine also has other features to improve portal performance.

- **Caching**: The Parallel Portal Engine has intelligent caching mechanisms to reduce the load on portlet servers and bandwidth necessary to transfer the content if the portlet content has not changed since the last time it was accessed. It is recommended that regular, simple caching intervals be set for all portlets. The portlets themselves can also override the preset simple caching intervals when new content is available.

- **Connection Pooling**: Connection pooling reduces the data overhead needed to access portlet content. A new connection is not needed for each user accessing that portlet content. This can increase performance by up to 20 times for large systems.

- **Load Balancing**: If the portlet supports load balancing in general, the parallel portal engine can perform simple load balancing to increase the performance, scalability, and fault tolerance portlet servers.

It is important to note that the parallel portal engine is not simple a frames based tool to access content in parallel. Frames act as individual browsers so each frame containing a portlet would require a separate connection to each portlet server. Frames also do not have caching or connection pooling built int. Because of this, frames tend to be a less secure and higher overhead method to “integrate” portlets than the parallel portal engine.

Infrastructure Factors

Hardware and Operating System

Different platforms have different throughputs for the WebCenter Interaction. One difficultly in comparing the performance of platforms is that some platforms require different hardware and it is often difficult to compare “power” since power is not just a factor of the number or speed of processors. For example, the amount of RAM and even the speed of the memory bus can have a bigger impact on performance than the number or speed of the processors.

The following chart shows a very rough comparison of portal performance on various platforms. The hardware used for the in the various platforms were not equivalent and therefore the ratios are not accurate in displaying of the relative performance between platforms. However, it does show the relative performance in general terms.[2]

[1] In reality, the size and number of portlets on a page does have a minor affect on throughput due to some overhead that is required to do security checks, perform transformations, and simply write the final page. However, the extra time required is on the order of milliseconds and is much less than the time required for portlets to generate content. Therefore, the extra time required is negligible and will not be noticeable by end users.

[2] Please see the appendix for a listing of the specifications of the machines used in the tests.

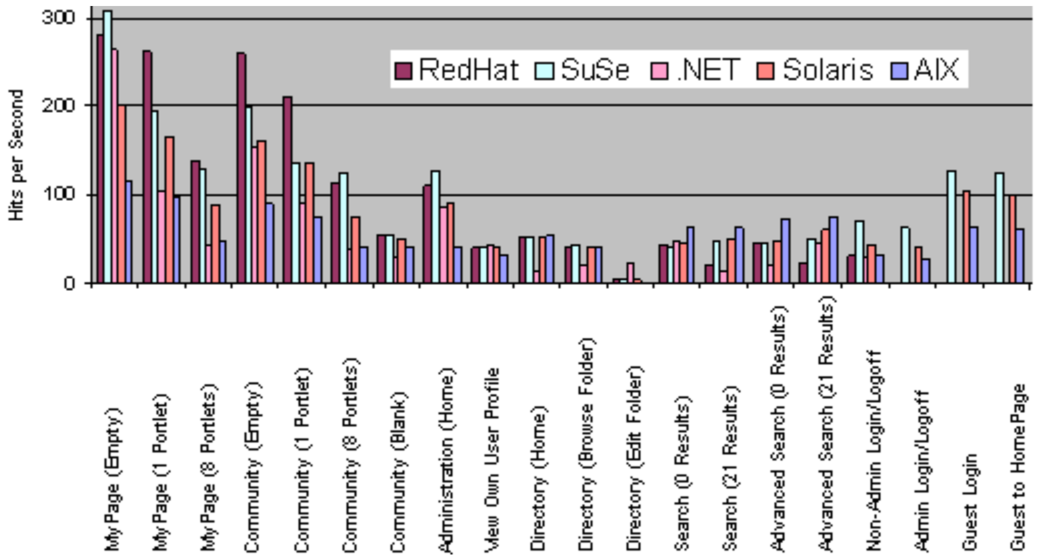

Figure 3: Performance on Various Platforms

As a very rough estimate, platform system performance is ranked as follows.

1. Linux (RedHat/SuSE)
2. Windows (Java)
3. Windows (.NET)
4. Solaris
5. AIX

CPU

Vertical scalability can be measured as a percentage of the Ideal Scalability. Ideal Scalability for vertical CPU scaling occurs when throughput increases linearly with the number of CPUs. In other words, if a 2 CPU server is twice the speed of a single CPU server, the vertical scalability is ideal, or 100%. If a four CPU server can only sustain three times the throughput as an otherwise equivalent single CPU machine, then the scalability is an un-ideal 75%.

Ideal Throughput = (# of CPU) * (Throughput w/ 1 CPU)

CPU Scalability % = (Measured Throughput) / (Ideal Throughput)

No hardware is capable of perfectly ideal throughput for any reasonably sophisticated software. Only trivial software can fit entirely within processor caches and avoid cache contention between CPUs or stress on shared hardware resources. The maximum CPU scalability is dependant on the hardware as well as the software, with various hardware producing different results for the same software.

Since no application has ideal vertical, a higher capacity portal system can be achieved with the same number of processors by using more machines with fewer processors per machine. The tradeoff is that more maintenance may be required due to the increased number of machines.

Memory and Memory Bus Speed

The amount and speed of the memory bus is often more important than number or speed of the CPUs being used on a server. This is because the portal utilizes many caching mechanisms to increase portal performance. Reading and writing of that cache benefits from memory and the speed of the memory bus. Both the standard and administrative portal servers benefit the most of improvements in the memory system of a server.

Just like the performance with increasing number CPUs, the performance does not increase linearly with increasing amounts of RAM. For example, on a Windows server, going from 1GB to 2GB of RAM will give a much larger performance boost than going from 2GB to 4GB of RAM. Going from 4GB to 8GB of RAM provides more benefits but not as much as the previous jumps. Therefore it is recommended that Windows portal servers have 4GB of RAM with the fastest memory bus possible. Going to 8GB is generally not necessary on a Windows server. Other hardware like Sun SPARC stations can derive more marginal benefit with higher amounts of RAM.

Databases

The portal connects to one instance of the database and therefore can affect the scalability and performance of the portal system. Standard database load balancing and fail-over tools can be used to improve fault tolerance. In terms of scalability and performance, the database requirements depend on many factors such as number of users, usage patterns, and page variance. Tests on previous versions of the portal show that database can handle increasing loads linearly until it reaches approximately 80% of CPU utilization, at which point its ability to handle increasing number of requests diminishes.

As a general rule of thumb, a database server can support up to 8 portal servers of the same specification. For portal servers running near the edge of the performance window, it is recommended that the number be reduced to 4 for equivalent machines. For some very large deployments, it is not uncommon to run a large number of Windows-based portal servers (over 25 in some cases) and use a Unix-based database.

The number of users does not have a significant impact on the performance of the system. Figure 4 shows that the difference between databases with one million users and 20 users is negligible.

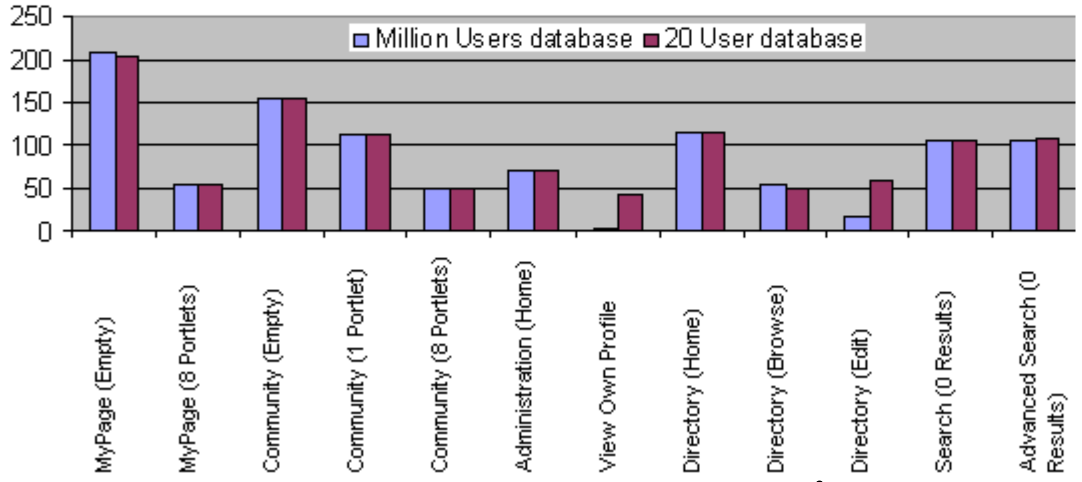

Figure 4: Affect of Large vs. Small Database[3]

Theoretically, increasing numbers of other objects in the portal such as documents and groups could have an impact on system performance. However, the effects have not been established although they are expected to be relatively small. Any noticeable impact would require portal object counts in the hundreds of thousand or millions.

Networks, Proxies, and Firewalls

Certain portal components utilize perform significant file transfers and therefore benefit faster I/O. For example, the Image Service simply serves images used by the portal. Generally all of the images can be stored in memory and very little processing is done so a fast disk subsystem or processor or even processor cache is not necessary. However, because every page uses images, the Image Service would benefit from fast network I/O.

[3] Database infrastructure was Oracle 10G running on 8xSparc Solaris 64-bit machine, 32 GB RAM

The Automation Service on the other hand does benefit from processors, RAM, and I/O. This is because for certain operation such as document crawls, the Automation Service transfers a temporary copy of the documents locally to extract the text and metadata of the documents for indexing. The file is then loaded into memory for text extraction. As a result, the Automation Service is limited more by RAM and network I/O to transfer the files than processor speed.

Proxies and firewalls can also affect I/O and the overall performance of the system depending on the layout of the system. For example, a common layout for extranets is to put a portal server inside a DMZ and the portal database behind the firewalls. A slow firewall will inhibit communication between the portal and the database and reduce system performance. It is important to reduce communication overhead as much as possible between the portal servers and database to maximize performance. There is no one best solution to where portal components should placed in the network since security considerations must also be taken into account.

A fully distributed architecture may increase the bandwidth needed for all the machines to communicate with each other. As a result, for maximum scalability, it is recommended that portal servers use the highest speed network connections available and be on a separate, dedicated subnet so that portal traffic does not need to compete for bandwidth with regular network traffic.

The portal can cache much of the commonly accessed data to reduce network traffic and latency issues. It is recommended that caching be utilized as much as possible to improve performance and reduce network traffic.

Application Server Settings

The application server used can have a significant impact on the performance of the overall portal system. For example, the performance of the WebCenter Interaction is higher on BEA WebLogic 8.1 than Tomcat 5 on the same hardware. Performance for certain page types can be up to twice as fast on BEA WebLogic as on Tomcat.

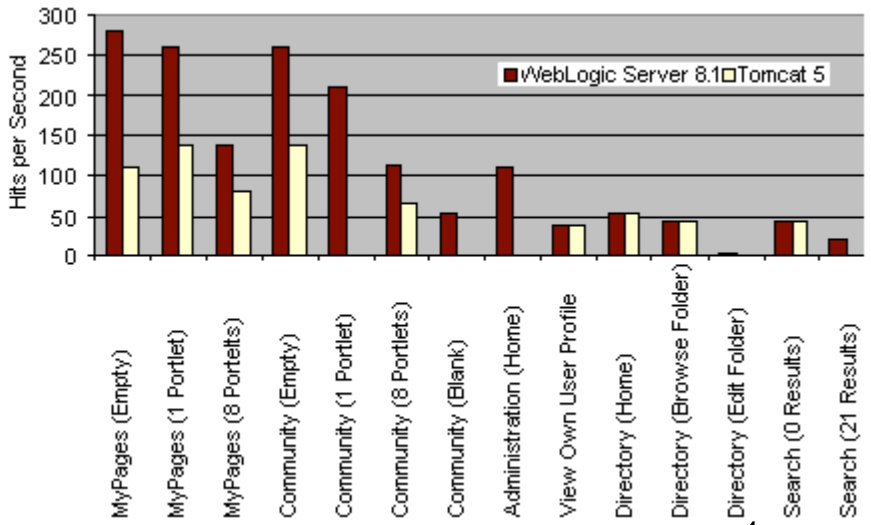

Figure 5: Hits Per Second for Various Page Types[4]

Much of the performance gains depend on the tuning of the application server specific parameters such as the number of WebLogic execution threads. Because they are application server specific, they will not be discussed in this paper but they should be considered in investigations of deployment performance.

[4] The portlets used in these tests were small static files so portlet generation time was not a significant factor in portal Hits per Second (HPS). As a result, portal overhead such as security checking and page creation time had a higher impact on HPS.

Garbage Collection

The most significant difference with respect to performance and scalability between a managed code environment such as Java or .NET and an unmanaged one such as a program precompiled from C++ is the way in which memory management is handled. A managed environment runs code in a virtual machine such as a Java Virtual Machine (JVM for Java applications) or a Common Language Runtime (CLR for .NET applications). This virtual machine controls all object memory allocation and cleanup, and the machine level specifics of code execution.

An unmanaged environment leaves it up to the program to manage every aspect of object allocation and cleanup. In C++ this is typically done through library or system specific memory allocators and cleanup is through deallocators or destructors. Because objects are referenced directly to their memory location, and operations are allowed directly on memory addresses, objects must remain at one address for their lifetime. If 90% of objects are quickly destroyed after creation, the remaining 10% are scattered throughout the address space and future allocations must search for allocation space in-between these remaining objects. Heap fragmentation and multithreaded allocation bottlenecks are common in unmanaged code environments, and these decrease vertical scalability. Such environments also lack centralized allocation or management as each library can allocate and destroy its objects in its own way.

Each virtual machine environment has its own unique way to deal with garbage collection efficiently. However they have a few things in common. All allocation and garbage collection is centralized and managed by the virtual machine. Memory allocation has been very fast and multithreaded in these environments for the last several major releases. The last generation of JVMs from Sun and IBM feature multithreaded garbage collection. .NET also features multithreaded garbage collection in the .NET CLR. As we will see this is essential for vertical scalability in a managed code environment.

Worker Processes in Windows 2003

Recycling worker processes can improve the stability of .NET applications but can add to the perception of slow performance. The issue is that when worker processes are recycled, the application is essentially stopped and restarted again. If users attempt to access the portal while the worker process is being restarted, the users will have to wait for the portal to fully start up before they get their pages. It is recommended that worker processes be recycled on a fixed schedule when portal usage is low to reduce the chances that users are accessing the portal during a recycle process.

There is also a similar feature to recycle idle processes that shuts down a process after a certain period of inactivity. The process is restarted only after a user attempts to access the application. However, restarting processes take time so the user will experience a delay in portal responsiveness. It is recommended that this feature be disabled to ensure maximum responsiveness of the portal.

SSL

Just as with any web application, enabling SSL on a portal server can affect system performance. This is due to both the portal having to encrypt content and the client browser having to decrypt that content. The use of SSL can drop overall scalability by up to 25%. The use of an SSL accelerator can significantly reduce the encryption burden on the portal server.

The other area where SSL may factor into the portal is communication with integration components. The WebCenter Interaction communicates with web services over HTTP or HTTPS. Communicating over HTTPS can increase server load just as with server to client communication. Generally it is not necessary to use HTTPS as a communication protocol between the portal and web services even if the end user access the portal over HTTPS because the portal proxies access to those applications.

Antivirus Scanning

Antivirus scanning can have a significant impact on various parts of the portal if it is not properly implemented. This is especially true for the search processes. Crawled files are written locally to extract file content to pass to the Search Service. Virus scanners can lock these files prevent the proper indexing of those files. The Search Service itself also performs many write operations as it indexes file content and properties and object properties.

While it is ideal for maximum performance not to have virus scanning at all, it is often policy to have some sort of scanning implemented. Therefore if virus scanning is required, it should be configured to be as low impact as possible. This involves configuration settings such as:

- Disabling On-Access Scanning
- Excluding certain portal directory folders from scans
- Scan during times of low portal use to lower the performance impact to end users.

WebCenter Interaction Factors

Page Type Variance

When estimating the size and performance requirements of a portal deployment, it is important to take into account the expected usage of different parts the portal. This is because different portal page types have different throughputs. For end users, the most resource intensive pages are the directory pages i. e. browse and edit the directory. The least resource intensive pages are the community pages and MyPages (personal pages). Fortunately these are also the most commonly accessed pages in a typical portal deployment.

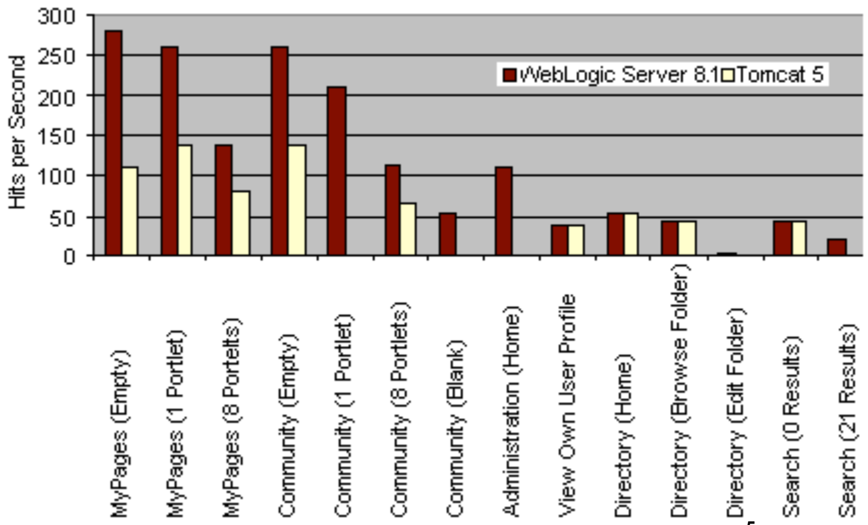

Figure 6: Hits Per Second for Various Page Types[5]

Administrative functions such as directory management and object creation can be very resource intensive.[6] Fortunately, administrative functions can be performed on a separate server. This is why the WebCenter Interaction gives the option of deploying administrative portal servers separate from the regular end-user portal servers. For example, Knowledge Directory security can be propagated just like in Windows Explorer. Since security propagation affects not only all of the objects in a folder but also all the objects in the child folders, and there can be 23 levels of folders, this can be a very resource intensive operation. Administration operations such as this should be done either during period of low portal use or on separate, Administrative portal servers so as not to affect portal responsive for other users.

[5] The portlets used in these tests were small static files so portlet generation time was not a significant factor in portal Hits per Second (HPS). As a result, portal overhead such as security checking and page creation time had a higher impact on HPS.
[6] Figure 6 only displays throughput for accessing the initial administration home directory. Administrative tasks such as security propagation can require much more resources.

Portal Page Size

The size of the portal page may be noticeable by end users that are using slow network connections. The size of the page is affected by many factors.

- The size of the WebCenter framework itself
- Portlet content
- Images
- Cascading Style Sheets
- Javascript and other libraries

Much of that content is loaded by default with the portal for backwards compatibility or to ensure the proper function of other add-on components that may require those files. Fortunately, in most cases these files that are downloaded with the infrastructure are downloaded one, stored in the browser cache, and simply reused. The files can also be trimmed to remove all used resources such as styles to reduce the initial download size if required.

Images can often be the largest items on a page. To reduce the download size of the page, the WebCenter Image server can be set to compress images before passing them to the client browser. This marginally reduces the throughput of the Image Service since processing power is required to compress images and also requires marginally more process power on the client side to uncompress the images.

Portal Customizations

All of the performance tests mentioned in this paper have been run with on un-customized portals with the standard navigation schemes. However, many portals are heavily customized to meet the specific requirements of a company. For example, some companies choose to not to use any of the several default navigation options and completely custom UI. Depending on how the customization is implemented, this could have a significant impact on portal performance.

If the customization interacts with eternal applications, more factors come into play. For example, if a navigation customization presents options based on inventory whose data is stored in an external application, performance is not only affected by the time required to run the customization itself on the portal server, but also network latency to pass the information, and the external applications ability to retrieve the information itself. There have been cases where slow performance was initially blamed on the portal and but was later found to be caused by the performance of an external application integrated with portal.

The WebCenter Interaction does have many built in customization methods that minimize the impact on performance. For example, using adaptive tags to specify navigation components is a highly scalable, easy to code, and easy to maintain method to customize the UI. Using tags will generally not have a significant impact on performance.[7] Portal Event Interfaces (PEI) generally have a greater impact on performance because they involve more custom code in both amount and degree.

Experience Definitions

The WebCenter Interaction 6.0 and later allows different experiences to be applied to users based on a set of administrator-defined rules. The experience defines the branding – header, footer, colors – navigation, and available features. These rules are evaluated with every page request. For example, a popular use of experience definitions is to change the look-and-feel of communities when each community is intended to be a unique composite application. Since there are a number of factors that can make up a rule, rules must be evaluated with each page request. As seen in figure 7, the number of rules has a minimal impact on the throughput of the portal. Most deployments will not have more than 40 rules. Typically, companies will use fewer than 5 rules.

[7] Of course, portal performance can be affected if a very large number of tags are used.

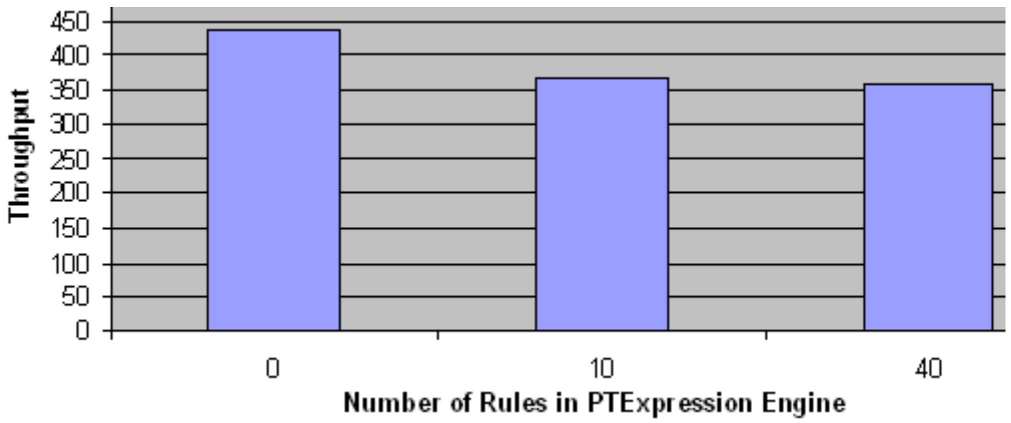

Figure 7: Performance Affects of Experience Definition Rules

High Availability

The WebCenter Interaction High-Availability feature replicates session information within a cluster of servers so if one server in a cluster goes down, then the user is seamlessly migrated to another server in the cluster. This means that all machines in a cluster replicates the sessions of all the other machines in a cluster. Tables 1 and 2 show the performance effects of high availability on throughput.

Measured Parameter	Server 1	Server 2
Throughput (Hits/sec)	253	
%CPU Utilization	87	84

Table 1: High Availability Off

Measured Parameter	Server 1	Server 2
Throughput (Hits/sec)	240	
%CPU Utilization	84	83

Table 2: High Availability On

Turning on the high availability feature in a 2-machine cluster shows a marginal decrease of throughput of less than 10%. However, the cluster can be configured to have many more machines. Throughput will decrease as the size of the cluster increases because the number of sessions that are handled by each machine goes up linearly with the number of machines in the cluster. As a result, it is recommended that clusters be limited in size to 4 to 6 machines to achieve a good balance of maximum scalability and fault tolerance. Fortunately, 4 to 6 machines are generally able to handle all but the largest deployments. Also, many load balancers allow the use of clusters of clusters. For example, for a deployment of 25 portal servers, instead of using one 25-machine cluster, the system can be configured to be five clusters of 5-machines each being load balanced within the cluster.

Gateway

While there are configurable settings[8] for the portal gateway, the biggest factor affecting gateway performance is the amount of data being passed through the gateway. Since the gateway is part of the portal server, gateway performance can affect overall portal server performance.

The gateway is used to pass information from the portal or an integrated application through the portal to the client browser. However, the design of the integration component such as portlets can affect the amount of data passed through the gateway. For example, many portlets often display images or use their own styles sheets as part of their design. These resources are generally static and do not have any security ramifications. Images can also be relatively large. Therefore, instead of passing images through

[8] The gateway can be turned off when used with old proxies that do not support the http 1. 1 protocol.

the gateway, higher performance can be achieved by putting the images on the Image Service and referencing the images directly from there. This is similar to the situation of having separate portal servers and Image Services except in this case, it has even a greater impact because it is affecting the portal process itself in addition to web server.

Portlets and Performance

The biggest factor when it comes to how fast the page is displayed to an end user is portlet integration. Generally this is not because of any portal infrastructure issues with the portal such as passing information through the gateway, but instead the simple fact that it often takes portlets time to generate it's content. As a simple test, go to page that is currently considered "slow". Remove all the "slow" portlets from the page. Note how much faster the page loads. Even highly customized portals are generally gated by portlet performance.

Because integration components are generally not controlled directly by the portal, it is mostly up to the portlet developer to ensure that the portlet satisfies the performance criteria necessary for overall portal performance. The portal does have several features however to help work around any limitations of portlet performance.

- **Ajax Infrastructure**: As mentioned in previously, end users often perceive performance as when they see something first, not when it finishes. Taking advantage of this trick of the mind, portlets can be written using Ajax technologies to not only refresh in place, but also allow the rest of the portal to be displayed and have the portlet display its content when the portlet is ready. The benefit is that the page is not limited by the performance of the portlets and other parts of the page can be used while other content is being generated.

- **Caching**: Instead of constantly going back to the portlet to retrieve content, the portal can store a copy of the content in a cache. This is useful for portal content that is not dynamically generated per user. Portlets can also be set to use cached content can still be used even after the maximum cached time has passed if the portlet has not changed.

- **Transformer Tags**: If a portlet needs to display some information stored in the portal such as user information, an adaptive tag can be used instead of having the portal retrieve the information itself. As the portlet content is passed through the portal gateway, the tags are transformed into the desired content. This often requires lower overhead than the portal accessing the portal API to retrieve the content.

- **Pre-Caching**: Generating dynamic content can be resource intensive and take some time. The WebCenter Interaction Automation Service can create a static HTML file from the dynamic portlet content asynchronously and store it in a web accessible folder. Since the portal can use any HTML page as a portlet, this portlet can be looped back to display the now static content on the portal. The benefit is that the caching time can be controlled more tightly.

- **Portlet Timeouts**: The portal allows timeouts to be set by the portlet managers and by end users themselves. This guarantees end-user response times.

Even though portlet performance has the greatest impact on the perceived performance of the portal by end users, they do not affect the actual portal performance or scalability that much since low performance portlets can be bypassed. This is one of the major benefits of the WebCenter Interaction distributed architecture.

There are instances where the distributed architecture could not resolve all issues with the performance of external applications and integration with the portal. For example, one customer made a login customization where after logging into the portal, they were redirected to specific pages based on profile information stored in their LDAP system. Unfortunately, while they did access the LDAP system remotely, they did not set a reasonable timeout since the target page was dependent on information stored in LDAP. The LDAP system was so slow that it often took 10-20 seconds for end users to be logged into the portal. They finally resolved the issue by importing the profile information into the portal and modifying the login customization to access the portal for the profile information instead of LDAP.

Portlets designed to take advantage of the features built into the WebCenter Interaction from the beginning can show higher performance than portlets that were simply tuned to be faster by optimizing code. Doing both will achieve the greatest performance benefits.

Example Deployment

The following is an example deployment of the WebCenter Interaction with a large number of users. A financial services company uses the WebCenter Interaction as a customer facing extranet. This means support for over 1.25 million users and sometimes up to 8 million hits per day. Using the rough ballpark starting point that many companies use of 10K users per portal server, that would mean using over 125 machines just for the portal servers, not counting other portal components.

After consulting BEA Consulting services, analyzing usage patterns, and optimizing the performance of each machine, they were able to reduce the number of portal servers to just 10 3GHz, dual CPU Windows servers with 3GB RAM. They found they actually only need 8 servers but have 2 extra servers for extra capacity when needed. They also use 10 machines of the same specification for portlets, 2 for Automation Services, and 2 for search. They achieve an uptime of over 99.36% with this configuration.

This is not the ideal configuration for every deployment, of course. For example, another customer with a fewer number of users, approximately 50K, but a larger number of indexed documents, over 500K, uses a fewer number of portal servers but a much larger number of Automation Services and Search Service machines.

Conclusion

Portal performance, or any applications for that matter, is dependent on many factors. While the out-of-the-box performance may be acceptable for some deployments, others will require analysis of those factors in order to optimize performance. It may also require changes to development practices to take advantage of features of the WebCenter Interaction. It is recommended that any company seeking to maximize the performance of their deployment of the WebCenter Interaction contact BEA Consulting Services for guidance in areas where their deployment can be optimized.

Appendix

Machine Configurations

- Linux, RedHat and SuSe, were conducted on the same machine with 2 CPU/HT, 3GHz, 2GB RAM, 800 MHz bus speed.

- The Solaris tests were conducted on a 4 CPU 1GHz Sparc II 2GB RAM machine therefore the portal throughput is expected to be lower in these tests than that in the Linux tests. A rough estimate of the hardware affect between Linux and Solaris tests is (3.5 CPU * 3GHz / 4 CPU * 1GHz = 2.625) a factor of 2 difference in performance if the CPU were pegged (100% utilization) in both, Linux and Solaris, tests. But in most tests CPU utilization in Linux tests was lower than that in the Solaris tests, therefore the hardware factor would be close to 1.5.

- The AIX tests were conducted on a 2 CPU, 1.5GHZ (POWER5) machine therefore the portal throughput in these tests is expected to be lower than that in the Solaris tests.

- The .NET tests were conducted on a 4 CPU/HT, 2.8GHz 4GB RAM machine. However, the memory bus on this machine was 400 MHz as compared to a 800 MHz bus of the machine used for Linux tests. Therefore, despite a higher number of CPU, a lower throughput should be expected from the .NET tests than from the Linux tests.