Volume 6, Issue 12, December 2016 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Performance of Ecommerce Implementation

Vijay Datla
Belk Inc. Charlotte USA

Abstract— Performance tuning is an important activity for any Enterprise Application Integration (EAI) implementation to ensure that the project delivered to the customer satisfies high load, availability and scalability requirements. Performance tuning is a critical part of software lifecycle that addresses the several key questions like the best configuration, hardware selection to achieve optimal values for performance parameters such as server response time, maximum simultaneous users, duration of each service invocation, interval and wait times, session usage and expiry, and memory usage. This white paper presents an approach to tune performance of applications that are deployed using the webMethods Broker /Integration Server architecture and provides a case study from a large retail implementation, where these principles were applied. The next few sections will discuss in detail about webMethods architecture and performance parameters, followed by a case study.

Keywords— Performance, Ecommerce, Vijay, Datla, Vijay Datla

I. INTRODUCTION

This section provides an overview of webMethods application architecture and provides introductory information about tuning the webMethods applications for performance and scalability. It contains the following topics:

- ❖ Performance and Scalability
- ❖ webMethods Architecture and Infrastructure Areas for Tuning
- ❖ Sample webMethods User Request Flow Performance and scalability of webMethods are defined as follows in the context of this paper:

- **Performance:** A webMethods application's ability to process messages within a given time, generally measured in response time and/or throughput. For example, measures of performance may include the time required for a message to pass through the system, or the volume of transactions (sometimes referred to as requests) that a server component can process in a given time period.

Some typical inhibitors of performance are inadequate hardware, excessive network round trips, heavy customizations, and poor networking infrastructure.

- **Scalability:** A webMethods application's ability to continue to perform well as volumes increase. Scalability is generally measured in hardware terms—for example, maintaining acceptable performance after adding new processors on existing machines (vertical scalability) or new Server machines (horizontal scalability) to process an increased number of users. Some typical inhibitors of scalability are an inflexible application module structure and an inability to run parallel processes. Optimal tuning of webMethods achieves a balance between performance and scalability. Every implementation of webMethods is unique - application architecture, infrastructure, and configurations may differ depending on the business model.

II. WEBMETHODS ARCHITECTURE AND INFRASTRUCTURE AREAS FOR TUNING

The following list provides details on tuning specific areas of the webMethods applications architecture and infrastructure.

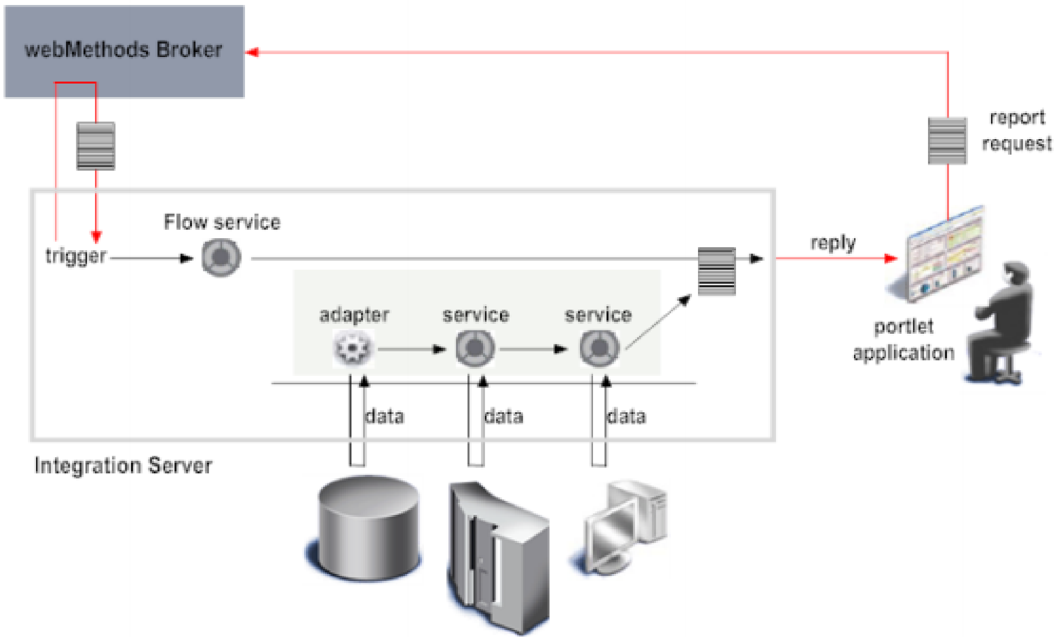
Performance in many of these areas can be monitored and analysed using various tools such as CA Wily Introscope, JConsole and also the webMethods products such as Optimize, Monitor, webMethods Reporting, and Mash Zone.

- ➢ **Architecture:** Architecture encompasses the selection of components, the interconnections between systems, the way different components are interacting and composition of these structural and behavioural elements into larger subsystems; and an architectural style that guides this organization. The key components are as given below:
- ➢ **Broker Server:**
  webMethods Broker provides the core messaging infrastructure of webMethods workflow. webMethods Workflow components act as a client to the messaging infrastructure to coordinate webMethods Workflow activity across the platform
- ➢ **Integration Server:** Integration Server facilitates the execution of services. E.g. it receives requests from client applications and authenticates and authorizes the requesting users, invokes the appropriate services and passes them input data from the requesting clients, receives output data from the services and returns it to the clients etc.

- **Operating System:** The way different operating system parameters are configured affects the performance of the application running on it. Some of areas which will be discussed further in this document are Server configuration, Network parameters, Process settings and Virtual memory settings.
- **Hardware:** The hardware selected for deploying the Broker and Integration server components should be capable of handling the throughput and scalability requirements expected.

III. WEBMETHODS REQUEST FLOW (GENERIC)

Figure 1 illustrates how a user request is processed within the webMethods Application and deployment architecture (generic).



3.1. Performance Considerations & Factors affecting webMethods solution performance

webMethods Broker and webMethods Integration Server interact using a high-speed wire protocol. But the speed at which these two components exchange documents is ultimately determined by resource availability (memory, threads, CPU cycles and so forth) and the characteristics of the solution (e.g., guaranteed or volatile documents, serial or parallel processing).

The following solution characteristics are important to understand / assess the performance & scalability:

Average document size, Maximum document size, Average document arrival rate, Peak document arrival rate, Average document processing time (by subscriber), Use of guaranteed versus volatile documents, Number of active triggers on the Integration Server, Number of trigger execution threads on the Integration, Server Volume of audit events, Overall performance of audit database, Number of concurrent connections allowed by audit database system.

IV. TUNING THE WEBMETHODS BROKER FOR PERFORMANCE

Broker Server is the container within which one or more Brokers reside. A Broker Server performs the communication related work of receiving client requests, dispatching requests to the requested resource (which in this case, is a Broker), and returning responses to clients. It also manages memory and disk resources for all the Brokers that reside on it. Some of the best practices to follow while designing a solution for performance are:

One broker should be deployed per broker server, although filters are defined in the Integration Server trigger, they should be applied in Broker to minimize document traffic. Client groups receiving ONLY synchronous request/reply documents should be volatile.

4.1. webMethods Broker Clustering

In webMethods Broker, components named Brokers execute requests for clients and maintain information about clients and their document types. You can improve messaging availability and reliability by clustering Brokers. webMethods Broker offers two types of clustering: policy based and high availability. Policy Based Clustering – round robin, weighed round robin.

4.2. Broker Queue Size

The Broker places documents that match a client’s subscription in the client’s queue. Each client has one queue, which is part of its client state object. A document remains in the queue until the client retrieves it (and acknowledges that it has retrieved the document successfully) or until the document expires. To reduce memory usage, volatile documents that have expired can be proactively deleted at regular intervals, based on the size of the queue, from the client queues and forward queues before the client tries to retrieve them. When the Broker queues a document for a client, it does not actually place a copy of the document in the queue. The Broker maintains only one copy of a document instance. Queues belonging to the subscribers of that document's type contain pointers to that instance.

- Controlling Queue Size for Volatile Documents You can proactively delete volatile documents from a queue by setting the "queue clean-up enable" and ""'queue-cleanupthreshold" parameters for each broker server instance.

- Queue Storage (QS) Queue storage refers to the file storage that Broker Server uses to persist guaranteed documents and non-volatile Broker metadata (e.g., Broker definitions, document type definitions, client group definitions) to disk.

Queue storage consists of two files: a log file, which is a fixed sized file that Broker can write data to quickly, and one or more storage files, into which data is finally, stored using a logged commit process.

Queue storage is shared by all Brokers on a Broker Server. Queue storage can be configured to operate in combined mode or separate mode. In combined mode, Broker metadata and runtime data are stored in the same log and storage files. In separate mode, the Broker uses two log files and two storage files. It stores metadata in one log.

Configuring Queue Storage: The queue storage system will seek a storage file that is not busy when it is moving an object from the log file to a storage file. Therefore, configuring multiple storage files can provide faster access than a single storage file. The number of storage files would depend on the input rate, parallel threads in the broker and number of subscribers.

4.3. Broker Server Memory

Broker Server requires a substantial amount of memory to hold the documents that clients publish. Most of the memory is used to hold volatile documents, which exist only in memory. However, a certain portion of memory is used to cache guaranteed documents and other objects that are kept in queue storage. To perform optimally, Broker Server needs enough physical memory to simultaneously hold:

- The maximum number of volatile documents that are awaiting delivery
- A fully loaded cache of guaranteed documents

If the operating system begins paging documents to virtual memory, the performance of the Broker Server drops significantly. Extreme out-of-memory conditions can cause the Broker Server to exit and restart.

Limiting Memory Usage by Broker Server

You can use the "max-memory-size" parameter to prevent Broker Server from receiving more documents than it has the physical memory to store. This parameter specifies an upper memory limit. As this memory limit is reached, the Broker Server stops accepting documents. When the max-memory-size parameter is set, Broker Server monitors the amount of memory that it is using for document storage. If the size of the incoming document plus the total amount of memory that is already "in use" exceeds the max-memory-size limit, Broker Server rejects the document and returns an "out of memory" error to the client. If the max-memory-size is not specified (that is, this parameter is omitted from the Broker Server's configuration file), Broker Server does not monitor its memory usage and will accept documents even though there might not be sufficient memory to process them.

- Value for the max-memory-size Parameter

The value of max-memory-size is an integer that represents the maximum amount of memory, in megabytes, that Broker Server an use for document storage.

Minimum: 50 megabytes. (If you specify a value less than 50, Broker Server automatically adjusts the max-memory-size to 50 megabytes.)

Maximum: (244-1) MB

- The preallocate-memory Parameter Optionally the "preallocate-memory" parameter, which causes Broker Server to allocate the amount of memory specified in max-memory-size when it starts, can be enabled to optimize

performance. If Broker Server cannot obtain the specified amount, it will immediately exit. Setting the preallocate-memory parameter ensures that the Broker Server can actually obtain the amount of memory that is specified by max-memory-size. If other applications besides Broker Server run on the same host machine or multiple instances of webMethods Broker run on the same machine and they use a significant amount of memory, it is recommended that you pre-allocate the memory space to ensure that the memory is actually available to the Broker Server. Setting the preallocate memory parameter will slow down the start-up sequence slightly.

- Selecting the Maximum Memory Size

Although there are no hard and fast rules for setting the memory threshold, you can use the following formula as a guide. This formula will provide an approximate value that you can use as a starting point and refine through testing.

MaxMemorySize = TotalMemory - (OS + OtherApplication + BrokerMemoryRequirements + Slack)*

4.4. Log File

The log file is where Broker Server initially stores the data which needs to be persisted to the disk. When a client publishes a guaranteed document, Broker Server commits the document to the log file before it returns control to the client. If the Broker Server cannot successfully write the document to the log, it returns an error so the publisher knows that the document was not received successfully. Broker Server is able to write to the log file very quickly, in part, because the file is a fixed size and is pre-allocated. When the log file becomes full, Broker Server transfers data from the log file to a storage file to make room for new, incoming data.

The maximum size for a log file is 2 GB on 32-bit platforms and 8 GB (as a practical limit) on 64-bit platforms. The log file for run-time data must be large enough to hold the largest document or batch of documents that a client will publish in a single transaction. The larger the log file, the longer it will take Broker Server to start. When Broker Server starts, it reads the entire log and moves items to long-term storage. Space for the entire log file is immediately allocated when you install or create the Broker Server. For optimal performance, it is recommended to place the log file on a fast device that is dedicated to the Broker Server and is separate from the storage files.

4.5. The Storage Files

Storage files are used for long-term storage of configuration data and for guaranteed documents that have not yet been retrieved by their subscribers. The Broker Server automatically moves data (in 8 MB chunks) from the log file to a storage file when the log becomes full. Log data is also moved to long-term storage each time the Broker Server is started. This process, which is often referred to as “playing the log,” effectively “empties” the entire log file by moving pending items to a storage file and purging items that are expired or have been acknowledged by all of their subscribers.
During storage file configuration reserve size and maximum size needs to be specified. The reserve size specifies the file's starting size (that is, the amount of space that is immediately allocated to the file). The maximum size specifies the size to which the storage file can grow.

- The maximum size for a storage file is 32 gigabytes.
- The minimum reserve size is 16 megabytes.
- Up to 62 storage files for each session can be created
- For optimal performance, place storage files on fast devices that are dedicated to the Broker Server and are separate from the log file.

4.6. Cache Memory

Broker Server uses a portion of memory to cache guaranteed documents and other objects (for example, Broker metadata and explicit-destroy client queues and subscriptions) that Broker Server keeps in queue storage. The max_cache_size setting restricts the amount of memory that Broker Server can use to cache data for queue storage. This parameter ensures that a certain amount of memory is available to retain queue storage objects in memory, but prevents the cache from depriving the Broker Server of the memory it needs to store volatile objects such as volatile documents and destroyon- disconnect client queues and subscriptions.

For separate storage sessions, there are two cache settings, one for each storage session. For combined storage session, there is a single cache setting. The actual size of the cache may exceed the specified maximum by up to 15% during peak loads. This15% allowance gives Broker Server the leeway to expand the cache in order to accommodate an entire document.

4.7. Configuring Broker Server to Use Asynchronous Write Mode

Broker Server can be configured to write data to disk asynchronously. In this mode, Broker Server permits the operating system to cache its disk write operations in memory. Enabling asynchronous write mode can significantly increase the performance of guaranteed documents. However, it also carries the risk of document loss in the event of a host system failure. If the host fails, any guaranteed documents that the operating system had written to cache, but had not yet committed to disk, are lost. If high degree of reliability is required, asynchronous write mode should not be enabled.

4.8. Client Sessions

A client can optionally enable “shared-state” when it connects to a Broker. The shared-state property enables multiple clients to connect to the Broker using the same client state object. (Typically, a client application does this to enable multiple client processes to retrieve and process documents from the client’s queue, thereby improving performance.) When shared state is enabled, the client state object contains a list of sessions. Each session identifies the IP address of a client that is currently connected to the Broker using that client state object.

4.9. Document Storage Type

The document storage type parameter determines how instances of that document are persisted. Documents published to Broker can be stored as volatile documents or guaranteed documents. Volatile documents are stored in local memory only. These documents are lost if the Broker host experiences a service interruption or the Broker Server is restarted. To reduce memory usage, volatile documents that have expired can be proactively deleted at regular intervals, based on the size of the queue, from the client queues and forward queues before the client tries to retrieve them. Volatile storage provides higher performance than guaranteed storage; however, there is a greater risk of losing documents if a hardware, software, or network failure occurs. All documents of a volatile document type and documents in a volatile client queue are lost when the Broker is shut down or when the computer restarts. This storage type is suited for documents that have a short life span or are not critical.

Validation

Integration Server can also validate documents at publication time. To improve overall performance, it is recommended to perform validation on either Integration Server or Broker, but not on both.

Using SSL Encryption

Although encryption provides the highest level of security, there are associated performance costs. Therefore, the trade-offs between the security needs of an implementation versus the requirement to implement an acceptable level of system performance need to be assessed before encryption is employed as part of the Broker security solution.

Specify Number of Logged Documents in Integration Server at One Time

Performance of the messaging system can be improved by retrieving a large number of documents from a Broker log queue at one time. However, too large a batch can degrade performance. The number of documents retrieved at once depends on environment-specific factors such as the amount of RAM on the Broker host machine, database, and the size of the logged documents. To determine the optimum batch size, it is better to run some normal workloads and tune the number. Broker can retrieve a maximum of 160 documents in a batch. This is controlled by the size parameter on the broker client.

V. TUNING THE WEBMETHODS INTEGRATION SERVER FOR HIGH PERFORMANCE

The Integration server is the processing engine, similar to the EJB container except the custom code in it is known as a service. webMethods Integration Server is one of the core application servers in the webMethods platform. It is a Javabased, multiplatform enterprise integration server. It supports the integration of diverse services, such as mapping data between formats and communication between systems.

5.1. Tuning Logging Performance

Set up customized logging for top level services only. Logging nested services would incur performance overhead. Log the Service failure or success. Only choose to log service failure or success and start when you need the greatest possible quality of service. Logging the pipeline can negatively affect performance especially if the pipeline contains large objects; because Integration Server has to make a copy of the pipeline every time the service is invoked. Store the input pipeline only for top level services, and only when absolutely necessary (for example, on failure only). Remove all unnecessary data from the pipeline to minimize the volume of data to store. To improve logging performance, avoid logging the same information twice by coordinating audit logging for services that are invoked by process steps.

5.2. Caching

Caching is an optimization feature that can improve the performance of the services on the integration server. You activate it on a service-by-service basis. When you enable caching, the server saves the service invocation results in a local cache for a specified period of time. While the results are in cache, rather than re-invoking the service, the server can quickly retrieve the service results for subsequent clients' requests for the service. Caching can significantly improve response time of services that retrieve information from busy data sources such as high traffic commercial Web servers or databases.

5.3. Managing the Server Thread Pool

Integration server performance and throughput can be controlled by configuring the minimum and maximum number of threads. The server uses threads to execute services, retrieve documents from the Broker, and execute triggers. When the server starts, the thread pool initially contains the minimum number of threads. The server adds threads to the pool as needed until it reaches the maximum allowed. If this maximum number is reached, the server waits until processes complete and return threads to the pool before beginning more processes. Optimum Integration Server thread pool settings will depend on the nature of the application, and can only be determined through performance testing/modelling. For any given scenario, there is an optimum thread pool setting; a thread setting above or below this setting will result in decreased performance.

Usually thread usage while running a minimal load would be an indicative value for minimum thread pool. It is best to determine the average thread pool value through load tests and as a guide max thread pool is usually twice the average value, however it should be estimated based on the expected peak load.

5.4. Setting the Session Timeout Limit

When a remote client connects to the Integration Server, the server starts a session for that client. That session remains active until the client application specifically issues a disconnect instruction to the server (which forces an immediate termination) or the session "times out" due to inactivity, whichever comes first. If a session is idle (inactive) for a long period of time, it usually means that the client is no longer active or that the connection between client and the server has been lost. The server constantly monitors for inactive sessions, and terminates sessions that are idle for more than the allowed period of time. If the server did not take steps to clear out such sessions, they would remain active indefinitely, wasting valuable server resources.

5.5. JVM Tuning

The heap size is controlled by Java parameters specified in the setenv.bat (Windows) or setenv.sh (Unix / Linux) file. JAVA_MIN_MEM is used to set the minimum heap size and JAVA_MAX_MEM is used to set the maximum heap size. By default, the minimum heap size is 256MB and the maximum heap is 512MB. Capacity planning and performance analysis should indicate whether there is a need to set higher maximum and minimum heap size values. Min/max memory can be set to the same value, e.g. 1024 MB / 1024 MB, to benefit from heap memory pre-allocation.

Tuning the JVM is a complex task which varies according the system, workload, architecture etc. It is always a good practice to determine the heap size using Verbose Garbage collection and using various garbage collection algorithms for each type of workload.

A heap that is too small will produce out-of-memory errors. A small heap may also cause sluggish performance due to frequent pauses for garbage collection (reclamation of memory that is no longer in use). Performance will also degrade significantly if the machine does not have enough physical memory to expand the heap to its configured maximum size and must then page portions of the heap to virtual storage (disk). Also oversized heap will incur higher memory management overhead. Integration Server Administrator can be used to examine the Available Memory and heap usage.

5.6. Service caching

A service is a server-resident unit of functionality that clients can invoke. A service might be an entire application or used as part of a larger application. There are several types of services: flow services (including Web service connectors), adapter services, Java services, and C/C++ services. To improve the performance of services, the server can be configured to cache the service results. Then, when the server receives subsequent requests for the service, it returns the cached results rather than executing the service

VI. BEST PRACTICES FOR WEBMETHODS SERVICE DEVELOPMENT FOR PERFORMANCE

webMethods provides two Service signatures: One based on the IData object, the other based on the Values object, use IData objects rather than Values objects wherever possible. IData objects consume less memory, and have more features (preserve order, allow duplicate keys, etc.) than Values objects. Drop pipeline variables as soon as they are no longer needed in the flow. This will make the flows easier to read and edit, and minimizes the Integration Server's memory consumption. Be careful to close all file descriptors, I/O streams, etc. using the java "finally" clause. This will ensure that objects are cleaned up even if an exception occurs. When you call a service within a flow, remember that the variables created in the service will be carried forward from the point of execution of that service. Also remember that input parameters of services that are set or mapped to will bleed into the pipeline, except for transformers. This can cause problems where multiple services with the same inputs/outputs names are called in succession or inside loops where the same service is called multiple times. So DROP what is not necessary immediately after calling a service. Always specify a timeout value, appropriate to the document's time-to-live, for request/reply services.

In the Catch Sequence, always make sure to close all FTP, GD, DB and other persistent connections. This will avoid leaving multiple connections open when services throw errors. Use the service wm.flow.getLastError to get the key or session ID from the pipeline to close the connections. Always check to make sure that calls to save Pipeline, savePipelineToFile, restorePipeline, restorePipelineToFile, are removed prior to moving to production. Defining a custom canonical offers the following benefits Because the canonical contains only the fields that are needed, the canonicals tend to be much smaller, which makes them perform better. If transaction volume is high and performance is potentially a concern, consider defining a custom canonical.

6.1. Hardware factors

Processor

Having more number of processors improves the performance. Also the thing to note is that the processors should also be supported by individual caches, high speed interconnection bus and their respective ALU, FPUs as sharing these would negate the benefits of having more number of processors.

Memory

Having sufficient memory is very useful and essential as inadequate memory causes slow paging to disk. This should be determined by taking into account memory requirements of all the applications running on the hardware.

Disk

The Broker and the Integration server should be backed by a high performing SAN. As the messages are constantly written to the disk and read from the disk, a slow IO device will slow down the overall throughput significantly.

Network

As the broker and the IS reside may reside on different machines, it is advisable to have a high bandwidth network connection between them so as to reduce any network latency or related delays, thus avoiding drops in throughput.

VII. WEBMETHODS PERFORMANCE TUNING – CASE STUDY

Background

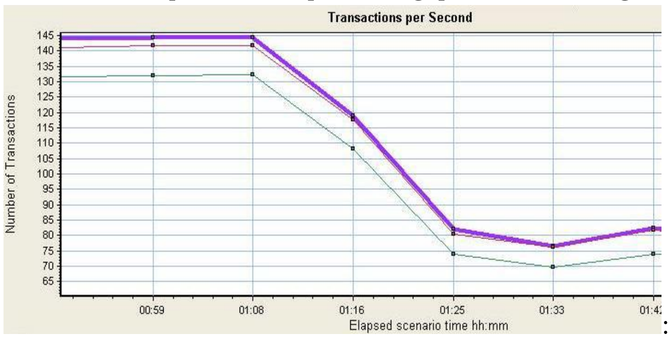
For the upgrade of an enterprise Retail sales suite, webMethods was chosen as the ESB. The requirement was for it to handle 140 messages per second for 8 hours. Eventually phase two of the implementation (in two years time) needed to support around 300 mps. The new retail store management suite would be sending messages to webMethods, which in turn would be processing and placing them on several destination queues or persist them in a database based on the requirements. The messages sent from the store vary in sizes between 30KB to 750KB but for non-functional assessment purpose, the 95th percentile of the message size was considered (variation between 30KB - 180KB). The requirement was that a message should flow within the webMethods system in 20 seconds i.e. the time from entry into the webMethods broker to the message being persisted in the database / destination queue should be less than 20 seconds. Performance testing this webMethods implementation revealed a variety of issues. The throughput of the implementation was nowhere near the target throughput and this put the entire project in jeopardy.

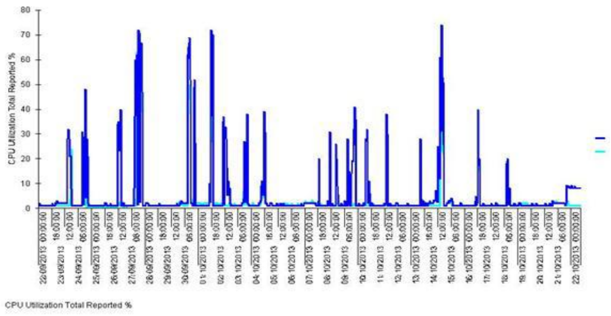
Setup

Environment consisted of 2 UltraSPARC T1 8 core CPU servers with 32 GB of RAM each and these servers were networked with the HP backend storage via Cisco Cat6505 switches. All the SAN used GigE connections to the switch, while half of the servers used 100-Mb connections and the other half GigE. The Brokers and the Integration servers were set up as 2 Brokers, one on each machine and 6 Integration servers split equally across the 2 machines. The broker was setup with 4 topics, and 1 trigger (for each) was reading messages from each of these topics. The number of IS Server threads was set at 10 and there was a message prefetch value of 10. All the validation and the code were written in Flow language and were present only in the Integration server.

Initial observation

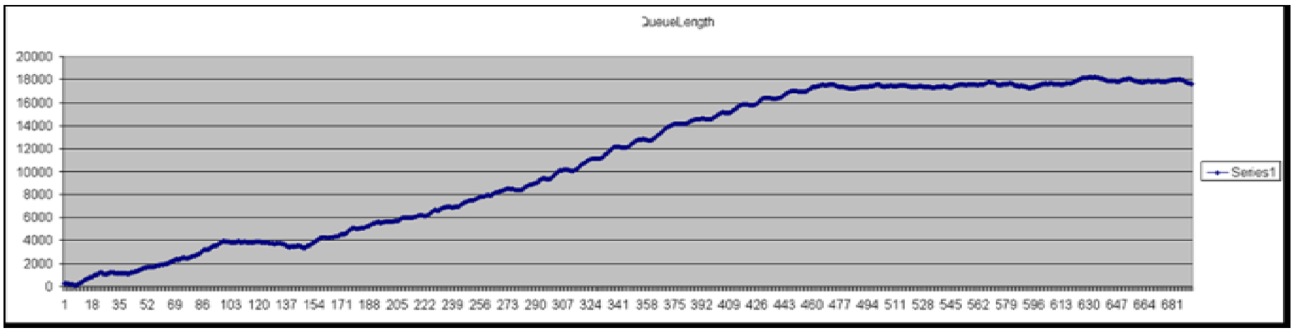
- The webMethods broker and the Integration server were running in the same physical machine.
- The size of the Broker log file was set to 1GB and this was filling up very quickly
- The throughput was dipping from 140 messages a second to ~70 messages after around 1 hour
- The CPU and Memory was not an issue with the CPU utilization well under 80% and the Memory under 40%. Below pic shows Dip in throughput and CPU usage





The Initial Prognosis

The initial assessment was that the messages were arriving at a rate which was more than the IS processing rate as a result of which the Broker log was eventually filling up dropping the throughput. This was supported by the statistics obtained from the broker and the IS during the tests. The presented in the next graph, the queue length was increasing at a pretty fast pace. There were 4 queues in the system and this behaviour was identical across all of them.
Below pic shows Que Length Increase:



The CPU and the Memory of the boxes remained well below the overload values. There was no other process or application running on these two machines.

Suspects

After identifying the slow processing of messages as the main for the dip in throughput, the possible causes were listed down.

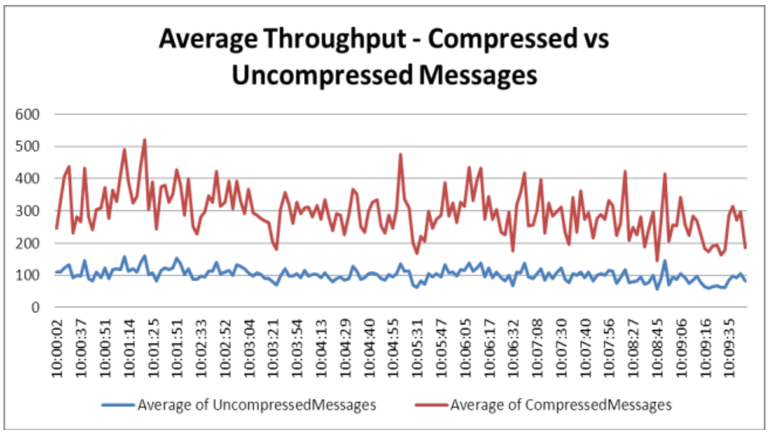
1. The size of the Broker log being too small for the workload and the messages sizes used
2. The validation / processing code present in the Integration server.
3. Poor configuration of the thread pool
4. Un-optimized JVM
5. Poorly performing IO
6. Logging

The problem could be one or a combination of the above factors.

Analysis

To rule out point 1 in the list of suspects, broker log file size was increased to 2 GB. This change allowed the throughput to remain at ~140 messages per second for double the period as compared to the earlier scenario. But the same dip was observed again after the Broker log was filled up and the swapping started. Keeping the log size at 2 GB, the I/O performance was checked and as the disk might be performing poorly taking into consideration the number of I/O operations going on. The disk subsystem found to be performing quite well and there have been no issues reported by other applications using the same SAN. This ruled out two suspects from the list. It was evident that the messages were arriving at a rate which was more than the IS processing rate as a result of which the Broker log was eventually filling up dropping the throughput. So to avoid filling up the broker log at the current rate, it was decided to compress the messages coming into the Broker. Decompression logic was incorporated in the Integration server as a part of the message processing logic. Using the same test setup, the message sizes were varied between 35KB to 180KB as was derived from production workloads.

The below figure highlights the variation in throughput when messages were compressed –



Although there was an increase in throughput with the decrease in message size, however an increased CPU utilization was observed in the Integration server as it now had an extra step of unzipping the compressed messages as a part of processing.

This increased the amount of time each message would spend in the Integration server thereby reducing the throughput at the IS tier eventually. With the increased end to end time of each message, the increase in CPU utilization, it was decided that the amount of gain due to reduction in the message size was being negated by the fact that a bigger delay was being added to the processing in the integration server. It was also noticed that slowly the broker log filled up as in the original scenario and the behavior was back to normal with the additional issue of dealing with an overloaded Integration server. The compression of messages will be beneficial when the overhead involved with uncompressing them is negligible in the context of overall processing time, which was not the case here. The following values of the Integration server were changed to accommodate the number of messages coming into the system.

- The number of threads was increased from 10 per trigger to 30

- The JDBC connections to the backend DBs were increased from 20 to 100 (all of them initialized beforehand)

- The prefetch from the broker was set to 20 from 5 to reduce pressure on the storage and increase bulk in memory processing the combination of the above changes the throughput was sustained for enough time to satisfy the business requirement. The changes to the key IS parameters worked like magic. Level of logging and persistence was looked into however to meet the business reliability requirements these could not be reduced or optimized any further. As a part of this tuning exercise, the JVM options were also tweaked to enable maximum throughput for as the characteristics of the system demanded, explained in further detail below. To check the performance of the JVM, the –verbose option was turned on and the logs from the Garbage collector were studied.Initial JVM setup was as follows:

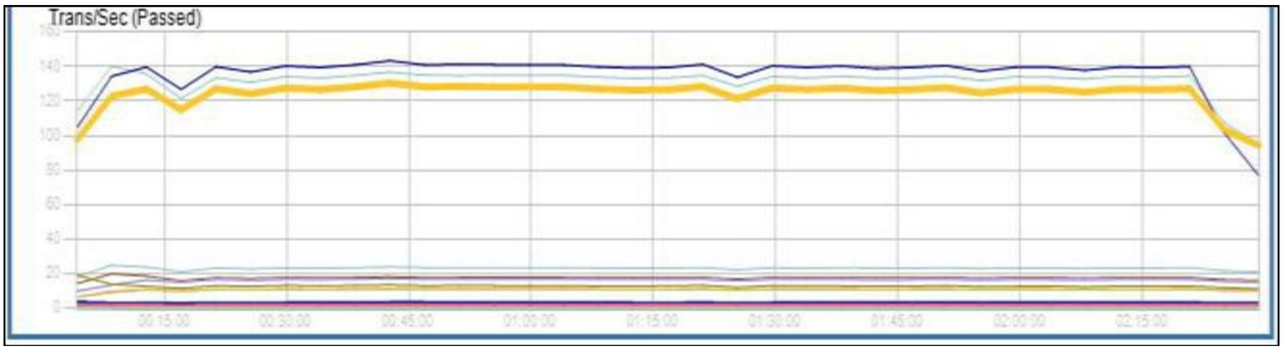
-Xms1024m –Xmx2048m -XX:SurvivorRatio=3 -verbose:gc

which was changed to the below even though no specific issues were identified from the GC usage patterns. These changes were done to ensure that the JVM was optimized as a part of the overall tuning exercise.

-Xms2560m –Xmx2560m -XX:SurvivorRatio=3 -verbose:gc

The above changes did not affect the performance of the system much. There was a very small improvement seen in terms of very few minor drops in throughput as compared to earlier scenarios where we had drops frequently. This is due to the decreased frequency of GCs now compared to earlier.

Below pic shows the system behaviour after tuning:



7.1. Results

The bottom-line results were excellent. The overall processing increased from 70 to 120 messages a second which was an improvement of ~70 percent. The main issue where the throughput decreases with time as a result of the queue build up was resolved to the satisfaction of the business requirement. There were still a few areas identified with the Flow language code and the validation code which were addressed separately as part of further fine tuning and scaling exercise (out of the scope of this document). At this point it is good to note that for the 2nd phase of the work the throughput requirement for this tier was more than doubled and it was evident that using the existing setup it was not possible to reach this target throughput. It was concluded that in another 18 months' time software and infrastructure upgrade and potential change in architecture for the EMS layer need to be considered for the phase 2 of the business implementation.

VIII. SUMMARY & CONCLUSION

This paper has covered basic concepts of webMethods ESB and what affects its performance, which was used as guide and selectively implemented during a specific business scenario. As noted, there are a number of tradeoffs involved in configuring the system. It really comes down to a tradeoff between performance/reliability and cost. By choosing the best solution/ implementation costly downtime and the possibility of losing data can be avoided.

There can always be mechanisms such as message compression, changing message formats from XML to JSON which help in improving the performance of the System as a whole by keeping the document size down but the tipping factor is how much overhead do they add to the processing and resource utilization for the gains they provide.

REFERANCES

http://www.softwareag.com