Performance Benchmarking an Enterprise Message Bus

Sumant, Pramod, Anurag

sumant_vashisth@mcafee.com, pramod_sharma@mcafee.com, anurag_sharma@mcafee.com

Abstract

An enterprise message bus is used for designing and implementing the interaction and communication between one or more interacting software applications or services. A messaging architecture is a combination of a common data model, a common command set, and a messaging infrastructure to allow different software services to communicate through a shared set of interfaces.

One of the challenges in the messaging architecture is to monitor and measure the system performance, scalability bottlenecks and reliability in real time operation. Performance metrics, for example, expected event notification latency, utilization and throughput of various components (like event broker), handling of messages of different sizes, are essential to determine the bottlenecks in the design, capacity of the system, optimal configuration and other parameters. Also, the systems need to be continuously monitored and fine-tuned based on the changing requirements in real-time scenarios.

In this paper we will present the process, method and test suite for evaluating the performance, reliability, and quality of a real world implementation of the messaging architecture using different messaging patterns. This paper will cover the following topics:

1. Provide an overview of the messaging architecture implementation in the security domain with different messaging patterns (pub-sub, request-response) and communication modes (in process, out of process, out of box)
2. Key factors to be considered for performance benchmarking.
3. Automation for the benchmark setup, configuration and execution.
4. Example test scenarios and results for the performance benchmarking test.

Biography

Sumant Vashisth *is Director of Engineering, Security Management Business Unit at McAfee. Besides, managing and leading teams across various industries during his long career, Sumant is a techie at heart. You can often see him chatting with engineers about the latest in technology and day to day technical challenges.*

Pramod Sharma *is a Software Architect at McAfee, with more than ten years of industry experience. An acknowledged technologist, Pramod has eight patent pending inventions to his credit. Passionate about software quality and process improvements, Pramod is a vocal champion of adopting changes, and learning from the industry. His interests span security management for embedded and mobile devices, and scalable architectures for distributed systems.*

Anurag Sharma *is a Principal SDET Engineer at McAfee, with around eight years of industry experience in software testing and quality assurance.*

Introduction

The enterprise message bus is a lightweight framework that provides flexible, loosely coupled integration across a broad range of enterprise applications. The key to dynamic interconnection across security firewalls, application protocols and languages is the messaging backbone. This architecture provides guaranteed delivery of messages among services and endpoints, performing security checks and protocol and data conversions on the fly. The message buses are designed to be fault tolerant and fail-safe. It would be safe to say, the enterprise message bus architectures will drive the growth of many businesses in coming future.

Motivation

Enterprise level message-bus architectures are increasingly becoming common in various domains across IT industry from banking and financial sector to transportation, supply chain management and healthcare sector. Newer and novel applications using message buses are being designed every day. Most of the message buses are deployed in highly data intensive and distributed networks. Predicting the message bus throughput, configuration, topology and workload in real-life scenarios is always a challenge. McAfee is in forefront in adopting and implementing message bus architecture for security applications. Looking at the current landscape, there is no standard method or benchmark to understand and compare the performance of various message bus architectures. This information can be necessary to design the most efficient system. This information may also help the enterprises to decide between an existing open source message bus frameworks or to develop of their own.

Key Considerations

As we start developing the benchmarks for validating the performance and scalability of our message bus or any messaging architecture, in general, following points must be considered –

1. Benchmarks must be based on real world applications but not to be biased by any particular domain. This dilutes the usefulness across applications (*Financial applications were the early adopters of event based architecture. Current standards slightly lean towards these*).
2. Benchmarks standards and tests should measure the performance of the architecture, as a whole, not the various components in it.
3. Benchmark standards should not be implementation specific i.e. based on any particular platform, software, language of implementation, topology etc.
4. Tests should be reproducible, scalable and exhaustive.

Benchmark Categories

Configuration

The machine configuration used for benchmarking tests, is the single most important factor. If the tests are performed on a high end machine with 8 core processor, 8 GB RAM etc., the performance results are certainly going to be better, but, they will be misleading too. Not everyone in enterprise will have such a machine, therefore, the following points should be considered while selecting the configuration for running benchmark tests to

1. Identify the configuration and document it in advance. It is better to select a set of three configurations – one high end (best), one average and one low-end (worst) configuration and execute the benchmark test on all three configurations.

2. For out-of-machine (network) communication, network configuration parameters like network topology, the network bandwidth etc. have important impact on end results therefore; these parameters should be identified in advance and documented.

3. Platform parameters like – processor (speed and number of cores), operating system, architecture, physical memory etc. should also be documented.

4. No other processor or network hogging tasks should be running while the benchmark tests are conducted.

Communication Modes

The messaging architecture can have two communication modes – 1) Client-Server or Request-Response mode, where, the client sends a message to be server and then, receives the response either synchronously or asynchronously. 2) Publisher-Subscriber mode – In this mode, the subscribers for some particular “Topic” of interest and then, gets notified when topic of interest is available on the bus. The enterprise message bus may have different performance results for both communication modes, and more often than not, tuned for one specific communication mode, therefore, considering both communication mode while doing performance benchmarking is very important.

Message Reach

In enterprise message bus architecture, the message consumers can be within the same process, in a different process on same machine or out on network. The message delivery latency, throughput and reliability vary depending on the relative location of client and server. Therefore, this factor needs to be considered while designing the benchmark test suite.

Identifying Bottlenecks

Often, the overall efficiency of the messaging architecture depends on the performance of one or two entities in the architecture for example, the broker or the packet forwarder or the connection manager. These entities drive the limits of message bus scalability and throughput. Therefore, identifying such bottleneck nodes and targeting the tests, which push the limits of these nodes, is very important.

Latency

Latency is the time taken for a message round trip i.e. the time taken for a request from client to reach server and corresponding response to reach client. For a publisher-subscriber type communication mode, only one way time is measured. Latency is an important measure for the performance of message bus. Following factors must be considered while designing the tests for targeting the latency factor –

1. For within the process communication, the latency is negligible therefore, the in-the-process communication configuration is not a good choice for latency tests.

2. The out of process and out of machine, tests should be considered. In the out of machine tests too, the machines or nodes which are distant (IP hops wise, not physically), should be considered for communication.

3. The server or receiving node should perform some minimal processing before sending the response. Generally, to reduce the latency, the tests are designed such that the receiving node doesn’t perform any operation on the received message and just returns the response, which is not a practical scenario. Therefore, some minimal processing at receiving node should be performed.

4. Gradually, increase the number of simultaneous connections and message size and record the average time taken for response.

Throughput

Throughput is the measure of the average number of successful messages delivered and responses received over a communication channel. The throughput is an important factor while designing message bus or choosing any open source message bus for enterprise usage. Less throughput can be unsuitable for some highly data intensive networks. Following factors must be considered while designing the tests for targeting the throughput factor –

1. The machine configuration has a major impact on the throughput, therefore, the standard machine configuration should be used while designing and executing the throughput tests.

2. More number of threads spawned at the client and server sides can also increase the throughput, therefore, the throughput values should also correlate with the number of parallel threads of execution.

3. Moreover, some processing at the receiving node should also be taken into account while measuring the throughput.

It is also important to note here, that the throughput, reliability, latency are not independent factors, they should all be considered together while designing the test scenarios.

Thread Pool vs. Thread IO

Thread Pool is where a number of threads are created to perform a set of tasks. Typically, there are many more tasks than threads. As soon as a thread completes its task, it will request the next task from the queue until all tasks have been completed. The thread can then terminate, or sleep until there are new tasks available. Thread IO is where a single worker thread performs the tasks sequentially. Thread pool provides better performance and throughput compared to creating a new thread for each task. For thread pool, the number of threads used is a parameter that can be tuned to provide the best performance. Therefore, if the message bus architecture supports thread pool and is configurable between the thread pool and thread IO, benchmark tests should be conducted for both configurations.

Reliability

Reliability in this context is measured in terms of number of packets lost in a given period of time. Some applications like financial applications require highly reliable networks. Following factors must be considered while designing the tests for targeting the reliability factor –

1. Push the messages continuously without delay and determine the number of messages dropped. Also, monitor the parameters like the CPU and memory utilization. Note that some message bus can crash in this scenario.

2. Run the tests for a large duration for example, two to three days continuously and monitor the messages dropped and other vital characteristics.

3. Varying the network configuration i.e topology and number of processes/machines communicating, can also generate significant reliability information.

Resource Utilization

In every test scenario, monitoring the resource utilization is very important. The following important resources should be considered –

1. CPU utilization – By general principle, more than 50% CPU utilization is not considered good.

2. Memory usage – Should be kept to minimum.

3. Number of threads – Always a tradeoff between the throughput and reliability.

4. Disk I/O – Disk operations are expensive and increase latency. They should be kept to minimum.

5. Number of open sockets/pipes – Expensive and potential security threats. Try to keep it to minimum. Consider intelligent designs concepts like reusing the sockets/pipes, opening connections on demand etc.

Security

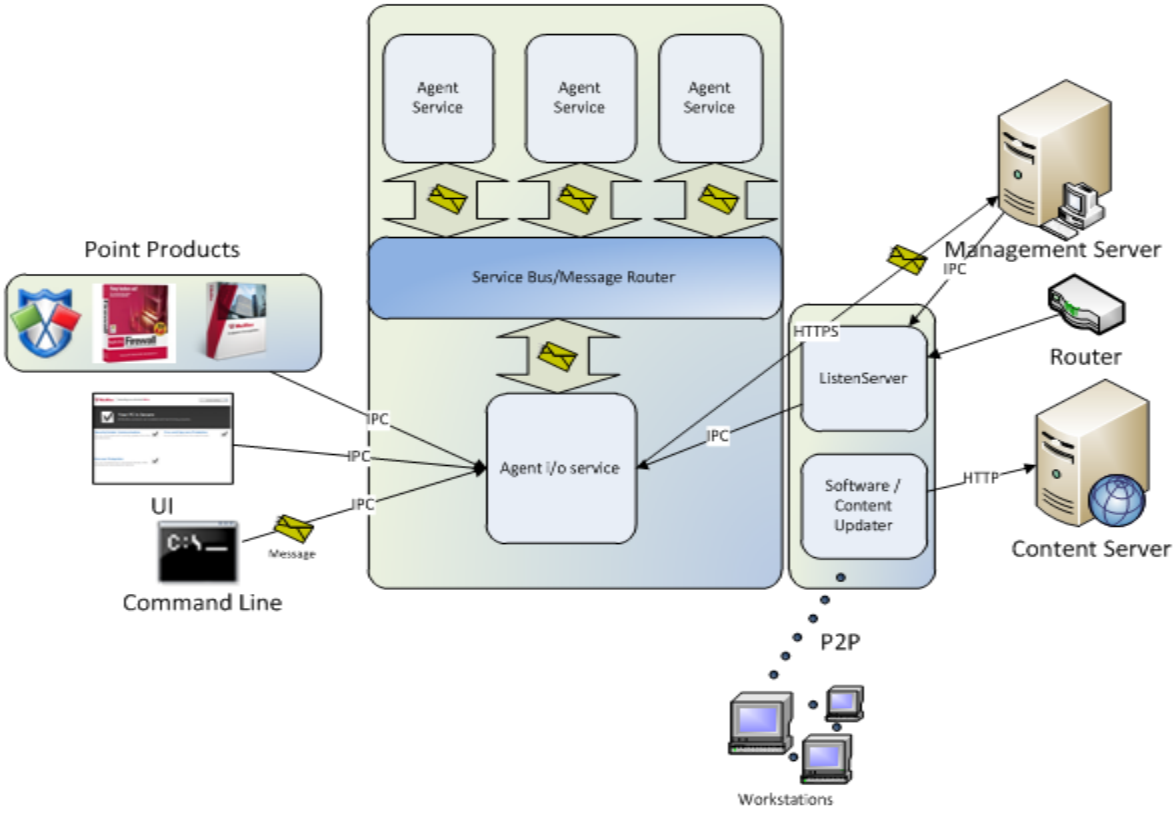
Most message bus architectures deploy some level of security either at message level or at connection level. Throughput can go down significantly when the message security aspect is introduced. Various security mechanisms like connection based authentication (connection based or message based), encryption, integrity check (hash algorithm), authorization etc., slow down the message bus at varying levels. Considering this, benchmark tests should be conducted for both secure and non-secure environments.

A Practical Example

McAfee Agent Messaging Architecture – A brief overview

The role of the McAfee Agent is to provide remote manageability of computing devices connected to a network, in particular to synchronize and enforce polices set forth by a management server. The agent is also responsible for sending device properties, events and operational data collected from device to the management server for reporting and other purposes. The agent will typically manage one or more security point products, each providing some kind of protection technology or 'business value' on the device. The Agent acts as a conduit for data between such point products and the remote management server, allowing exchange of predefined data types as well as private data formats. In turn, the management server typically manages several agents and associated point-products. The management server maintains a logical representation of each endpoint and allows the security administrator to configure policies for individual or groups of endpoints. The server also provides extensive reporting capabilities allowing the administrator to monitor the security status of the managed endpoints.

Endpoints and their associated management server can be deployed in an almost endless number of different network topologies. From the simplest, closely connected LAN environments to laptops and mobile devices travelling in and out of different network boundaries, possibly communicating via proxy servers, firewalls or even environments where devices communicate intermittently over low bandwidth but expensive satellite or dial-up connections. As you can see, Service-oriented Architecture (SoA) based on messaging architecture is the backbone of this enterprise application.



Example Test Scenarios

Some performance test scenarios are listed below. During each test, the resource utilization like CPU utilization, no. of threads, memory usage etc. should be monitored. Please note that this is not an exhaustive list of tests. Once the performance test setup is designed for automated test execution and monitoring resource utilization periodically, more test cases can be designed for message bus architecture performance fine-tuning.

1. Varying message size – Perform the communication between single synchronous server and client for a fixed amount of time varying the message size from 1 byte to 1 Gigabyte and count the number of successful. Perform similar test for asynchronous communication.

2. Varying number of clients – Gradually increase the number of clients communicating (fixed size messages) with a single server and measure the number of successful communications in a pre-determined fixed amount of time. Perform this test in both thread pool and thread IO mode.

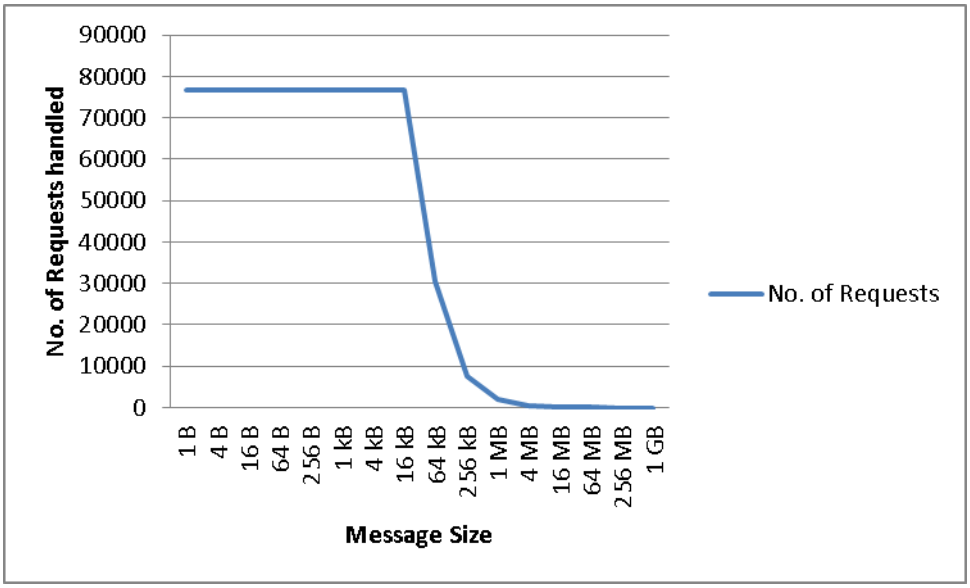
3. Implication of security - Perform the communication between single synchronous server and client for a fixed amount of time varying the message size from 1 byte to 1 Gigabyte and count the number of successful. Now, enable the message security and perform the test again. Observe the decrease in throughput when the message security is in place.

4. Thread Pool and Thread IO - Observe the number of clients that can be served by a single client in thread IO and thread pool configurations.

Example Performance Results

Following graph illustrates the behaviour of one server and one client process on the same machine communicating in the synchronous manner. The test is performed for 10 minutes duration where the client sends 1 request per millisecond. The message size is varying from 1 byte to 1GB. It is observed that the upper threshold for message size is 64MB, above that; messages could not be reliably transmitted. On the other side, the maximum ~78000 messages could be transmitted during the test duration. The throughput and latency parameters can be easily derived from this to data and performance can also be fine-tuned (Operating Platform – Intel ® Core ™ i7-2600 CPU @ 3.40GHz (8 CPUs), 8GB RAM running Windows 7 Professional 64-bit operating system. Network characteristics are not relevant for the out of process tests).

Approximate throughput calculation – 16 KB message size X 8 bits/byte X 2 (both request and response) X 78000 messages/ (10 X 60) seconds = ~32 Mbits/second



References

Zero MQ performance results at - http://www.zeromq.org/area:results