SocketPro performance comparison with window communication framework (WCF) and RabbitMQ

Contents:

1. Introduction

Performance is one of most important software quality metrics. It largely determines application responsiveness, scalability, and throughput. Every software engineer knows that software performance is extremely important with any real enterprise application. In regards to distributed applications, many factors such as hardware, software design architect, and code quality affect performance from a very high level of view.

We have created a socket-based framework based on inline continuous data batching, asynchrony, and parallel computation to accelerate data movement across computers running on different major platforms such as Windows, Linux, and Apple. One of its greatest advantages is that this framework delivers extremely high performance and scalability when compared to other well-known frameworks such as Java RMI, WCF, SOAP/XML web services, etc. This article, along with source codes, demonstrates the performance improvements of SocketPro framework over that of the latest Microsoft technology Window Communication Foundation (WCF) and RabbitMQ. Hopefully, these improvements catch your attention.

SocketPro and WCF comparison samples are located at the directory ../SocketProRoot/samples/performance.
SocketPro and RabbitMQ comparison samples are located at the directory ../SocketProRoot/samples/qperf.

2. Experiment conditions

Tests are completed with a home desktop as server or broker for persistent message queue comparison and a laptop as a client for cross-machine communication with network bandwidth equal to 1 Gbps. All numbers listed on figures are real and measured in units of milliseconds.

A *DataTable* instance is set with *BinaryFormat* for WCF remoting. Additionally, security of WCF is disabled under all testing cases. Note that the used database is the sample database AdventureWorks2012 for SQL server 2014. All of the other experiment conditions are shown on these figures.

In regards to SocketPro and WCF performance comparison tests, all numbers listed on their figures are obtained from executing 10,000 times for the request Echo and 100 times for four SQL statements. These figures also list the number of rows and the size of each of their record sets in megabytes.

In regards to SocketPro and RabbitMQ performance comparison tests, results are obtained by en-queuing and de-queuing 100,000 times of messages with three different sizes (200, 1024 and 10240 chars).

Note that source codes of RabbitMQ sample project come from the sites https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/dotnet/NewTask/NewTask.cs for en-queue and https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/dotnet/Worker/Worker.cs for de-queue with slight modification.

3. Experiment results and analysis

SocketPro vs WCF

Overall, as far as we have known, SocketPro is significantly faster than WCF under all cases with no exceptions. Typically, SocketPro is about 4 times faster than WCF. In extreme cases, it could be more than 30 times faster than WCF if SocketPro request async/batch feature is used, as you will witness in the upcoming sections.

Cross-application communication at the same machine: The following Figure 1 lists obtained performance comparison data for cross-application communication at the same machine.

WCF	SocketPro + Sync	SocketPro + Async/Batch
Echo request (10000 times)		
712	239	30
*Select * from Production.Culture (100 times, 8 rows, 0.008M)*		
55	18	14
*Select * from Production.Product (100 times, 504 rows, 0.102M)*		
1518	362	269
*Select * from Production.BillOfMaterials (100 times, 2679 rows, 0.156M)*		
4380	729	566
*select * from Production.WorkOrder (100 times, 72591 rows, 4.086M)*		
135800	20660	20220
		Test Machine: Acer AT3-605-UR22 Desktop
	**time unit=ms*	*SQL server 2012 (x64) on Acer AT3-605-UR22*
		Intel Core i7-4770 CPU 3.40 GHz, 16 GB RAM
		WCF tcp binding with secure transaction disabled
		Window 8.1 pro (x64)

Figure 1: Performance comparison between SocketPro and WCF for cross-application communication at the same machine

In regards to simple requests like echo, SocketPro is about 2.5 times faster than WCF with synchronous communication, as shown in the above Figure 1. If requests are processed with async/batch communication, SocketPro could be easily more than 20 times faster!

In regards to exchange of ADO.NET objects, SocketPro is typically 2 times faster than WCF for a small set of records, and about 4 or 6 times faster than WCF for middle or large-size sets of records. If the SocketPro request async/batch feature is used, improvement will be considerably better especially for small set of records.

Cross-machine communication: The below Figure 2 shows you performance comparison data for cross-machine communication case.

In regards to simple requests like echo, SocketPro is about 1 time faster than WCF with synchronous communication, as shown in the below Figure 2. If request async/batch feature is employed, SocketPro could be easily more than 50 times faster!

WCF	SocketPro + Sync	SocketPro + Async/Batch
	Echo request (10000 times)	
3802	**2901**	**58**
	*Select * from Production.Culture (100 times, 8 rows, 0.008M)*	
433	**360**	**46**
	*Select * from Production.Product (100 times, 504 rows, 0.102M)*	
2820	**670**	**541**
	*Select * from Production.BillOfMaterials (100 times, 2679 rows, 0.156M)*	
9565	**1491**	**1259**
	*select * from Production.WorkOrder (100 times, 72591 rows, 4.086M)*	
245608	**37648**	**34442**
		Test Machines: CyberPower Desktop; Server: Windows Server® 2012R2 VMware
		**time unit=ms SQL server 2014 (x64) on Windows Server® 2012R2 Vmware*
		Client: Intel Core™ i7-3770 @ 3.40GHz 16 GB RAM; Server: Intel Xeon(R) E5-2630 v2 @ 2.60 GHz 2.60 GHz 4 GB RAM
		WCF tcp binding with secure transaction disabled
		Client: Window 7 Professional (x64); Sever: Windows Server® 2012R2 Vmware
		Network bandwidth = 1 Gbps

Figure 2: Performance comparison between SocketPro and WCF for cross-machine communication

In regards to exchange of ADO.NET objects across machines, SocketPro is typically 1 time faster than WCF for small sets of records, and about 4 to 6 times faster than WCF for middle or large size sets of records. The improvement is obviously larger *especially* for small sets of records with its request async/batch feature turned on.

Reasons for SocketPro better performance than WCF: As you can see from the above two figures, our SocketPro is significantly faster than WCF. This is not surprising at all, considering that SocketPro has the following features to make this huge performance improvement realized.

- SocketPro is written with non-blocking and inline continuous data batch in mind so that networking and CPU processing at both client and server side are extremely efficient.
- Efficient serialization and de-serialization with streaming style can happen concurrently at both client and server sides simultaneously. This is not possible with WCF at this writing time, or could never be possible for WCF.
- ADO.NET objects are especially engineered within SocketPro for chunk and streaming push approach. SocketPro supports remoting ADO.NET datareader object online and directly without the requirement of preloading a whole set of records in memory. However, WCF doesn't support this feature yet.

SocketPro vs RabbitMQ

The performance test is designed to focus on persistent message queue with auto acknowledgement at the server or broker side only since RabbitMQ does not support client side message queue.

Overall insofar, SocketPro is significantly faster than RabbitMQ under all cases without exception. Typically for de-queue, SocketPro is about 8 times faster than RabbitMQ for both cross-application and cross-machine communication situations. In extreme cases, it could be more than 25 times faster than RabbitMQ as you will see in the upcoming sections. In regards to en-queue, SocketPro is typically 10 times faster than RabbitMQ under cross-application communication for the same machine. SocketPro is still about 8 times faster than RabbitMQ under cross-machine communication situation, whose improvements will be largely dependent on network bandwidth.

Cross-application communication at the same machine: Figure 3 billow lists the following obtained performance comparison data for cross-application communication at the same machine.

RabbitMQ	SocketPro
Test string with 200 chars (100,000 times)	
6171/14129	270/544
Text string with 1024 chars (100,000 times)	
12697/14182	786/833
Test string with 10240 chars (100,000 times)	
32811/17130	7192/4777
	Test Machine: Acer AT3-605-UR22 Desktop
	Intel Core i7-4770 CPU 3.40 GHz, 16 GB RAM
	Durable with auto acknowledgement
**time unit=ms*	***Enqueue/Dequeue*** *Window 8.1 pro (x64)*

Figure 3: Performance comparison between SocketPro and RabbitMQ for cross-application communication at the same machine

In short, SocketPro is significantly faster than RabbitMQ for same-machine cross-application communication for both en-queue and de-queue as these numbers clearly support. Improvements are between 4 and 25 times greater.

Cross-machine: The below Figure 4 lists testing numbers obtained from cross-machine communication over a network with bandwidth 100 Mbps.

Test results at Figure 4 clearly show that our SocketPro wins over RabbitMQ easily in persistent message de-queue under all cases. If messages are small in size, SocketPro could be 50 times faster than RabbitMQ. In regards to big messages, SocketPro is still 9 times faster, which is largely dependent on networking bandwidth.

In regards to en-queue, it seems that RabbitMQ does a good job. RabbitMQ is able to en-queue 15,000 small messages persistently per second. For middle or large sizes of messages, RabbitMQ is poor since network utilization could be 10% under this test networking condition. RabbitMQ performs really poor in comparison to our SocketPro in en-queue; SocketPro is typically between 1 and 10 times faster, as shown in the below Figure 4.

RabbitMQ	SocketPro
Test string with 200 chars (100,000 times)	
6873/44400	**631**/820
Text string with 1024 chars (100,000 times)	
9210/45373	**2144**/2180
Test string with 10240 chars (100,000 times)	
108900/202000	**19684**/20514
Test Machines: CyberPower Desktop; Broker: Acer AT3-605-UR22 Desktop	
Client: Intel Core™ i7-3770 @ 3.40GHz 16 GB RAM; Broker: Intel Xeon(R) E5-2630 v2 @ 2.60 GHz 2.60 GHz 4 GB RAM	
time unit=ms* ***Enqueue*/*Dequeue* *Durable with auto acknowledgement*	
Client: Window 7 Professional (x64); Broker: Windows Server® 2012R2 VMware	
Network bandwidth = 1 Gbps	

Figure 4: Performance comparison between SocketPro and RabbitMQ for cross-machine communication

4. Conclusions

SocketPro is written with inline continuous data batching, non-blocking and parallel computation in mind so that networking and CPU processing at both client and server sides are extremely efficient. Its design is considerably different from that of other communication frameworks, which are typically created with single request and blocking communication style. You may use worker threads at client side to fake asynchronous request processing at client side, but usually doing so doesn't boost performance at all which may cause extra thread-context switches. Contrarily, SocketPro does not need worker threads at client side, though it does create a pool of threads for hosting non-blocking sockets. Note that SocketPro uses thread pool at client side for processing returning results in parallel at client side only, not for sending requests.

Although we don't compare SocketPro with other common communication frameworks or technologies, it is guaranteed that all of other frameworks run like snails in speed when compared to SocketPro.