RC 21762 (97997) 26MAY2000 **Computer Science/Mathematics**

Research Report

A Scalable Stock Web Service

Katherine Betz
IBM Research Division
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

A Scalable Stock Web Service

Katherine Betz
IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
katherin@us.ibm.com

Abstract

We built an internet stock service and focused on its scaling characteristics. This service allows a person to request eMail notification when a specified stock reaches a price. For example, one could request to be eMailed when IBM reaches or exceeds $150. We decided to leverage the capabilities of an underlying publication/subscription (pub/sub) system which has a JMS interface. We explored ways to configure the pub/sub system as well as how to tie it to a Web Application server. We also explored performance results on different machines. Our goal was to determine what was necessary to scale to 3 million user stock requests as well as to have 100,000 satisfied each day and 100,000 new requests entered each day. We proved this possible.

Introduction

More and more users of computers and the internet are requesting services available on the web. A service could be travel reservations given certain criteria, notification of stock price changes, forwarding news articles of interest, etc. Some people refer to these services as personal agents. We selected a stock service for this scalability study. Our goal was to explore scalability issues first and then extend the system to more complicated requests and also provide notification to different pervasive (PvC) devices.

The service we choose to explore is a common one: tell me when a stock I'm interested in reaches a certain price. We believe that there are three categories of users for such a stock service: brokerage traders, day traders, and people who are less time sensitive to trades because it is not their job. This study targets the third audience. We determined our scaling targets based on customer input. We decided to support 100,000 customers, at 10-30 stock choices each (3 million active). We assumed that the stock trade rate would have peaks at approximately 2,200 trades/second. We did not have data on how often a person's stock request might get satisfied. We figured an aggressive assumption would be that each customer would get notified once per day. Therefore, the steady state we want to support is 100,000 being satisfied per day which generates 100,000 eMail notifications, and 100,000 new stock requests added each day.

Architecture

The software architecture we have chosen is web based. The user enters a stock request on a web page, that is forwarded to a servlet on IBM's WebSphere application server. The servlet creates a stock request EJB [1]. The stock request persists in a DB2 database. The servlet then forwards the stock request to an Agent Server that in turn forwards the request, in the form of a subscription, to a pub/sub system. The stock ticker tape is published to the pub/sub system. When the Agent Server submits the subscription it also provides a call-back method to be executed when the subscription is satisfied by a publication. The call-back method knows how to eMail the user to inform him that his stock reached the specified price. See Figure 1.

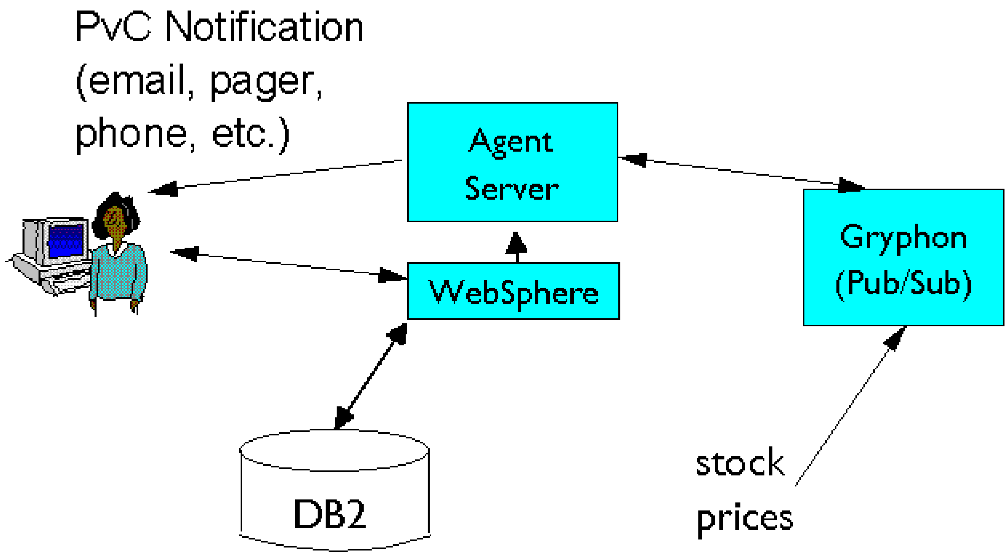

Figure 1 - Architecture

The pub/sub system we are using is an IBM internal system called Gryphon [2][3]. Gryphon is referred to as the broker. Gryphon is an evolving system. As mentioned above, it implements the JMS interface [4]. A pub/sub system was chosen because (1) there is no need for the publisher (stock ticker) to know about the subscriber (web user) and (2) it has good performance and scaling abilities. It allows topics to be created that can be subscribed and published to. Gryphon supports content based subscriptions. This means the subscriber can subscribe not just to a message associated with a topic, but to a subset of the topic's messages. For example, if the user is interested in when IBM reaches or exceeds 150, the topic could be 'stock.*' and the subscriber would be looking for content that matched: "symbol='IBM' and price>=150", or the topic could be "stock.IBM" and the subscriber would be looking for content that matched: "price>=150". The stock ticker tape entries get published to a topic. If the stock on the ticker tape says that IBM is at or over 150, a match will be made and callback code (in our case to eMail the person who requested it) will be executed.

We made some decisions about the software architecture to enable a simple solution to improve fault tolerance. First we added an Agent Server which is a Java RMI server [5]. It was not reasonable to use the Web Server and Gryphon alone. If the servlet submits a subscription to Gryphon, it has to provide a call-back method. Since the servlet engine can bring servlets up and down as it chooses, it is a poor choice to put call-back code in a servlet. If the servlet is unloaded the call-back is lost, which means the person's request is dropped. Therefore, the call-back method cannot safely be attached there. The Agent Server gives protection against unknown servlet lifetime.

The second decision pertains to the relationship between Gryphon and the Agent Server. If Gryphon goes down, the subscriptions are lost. The Agent Server detects the outage and cleans up the JMS resources (the call-back methods are dropped). If the Agent Server goes down, Gryphon detects the lost client and cleans up the subscriptions. Therefore, if either the Agent Server or Gryphon goes down, it is simplest to restart both and issue lost subscriptions again. Because of this situation, we favor an architecture where the Agent Server and the Gryphon are paired. We ran performance tests and determined that this affects the performance slightly. The stock request EJB contains the identity of the Agent Server/Gryphon pair that received the request. If an Agent Server or Gryphon broker fails, we can reissue the stock requests (i.e. find all entries in the database that have that Agent Server/Gryphon pair and that have not been satisfied already).

Planned Tests

We were able to vary the software architecture used in our tests. The software architecture can be varied at the pub/sub system, as well as how WebSphere communicates to the pub/sub system. At the pub/sub system we are able to vary the number of brokers, connections, topics, clients (the Agent Server), JMS sessions, and whether sessions are associated with threads. It is possible to have one

Agent Server attached to multiple brokers or many Agent Servers attached to one broker. As mentioned earlier, we decided to keep the Agent Server and Gryphon broker in a one-to-one relationship with each other for fault tolerance.

Our plan is to explore the scaling of the stock service on various machines. These are:

- Ducky: a single processor NT machine: 366 MHz and 320 Megabytes of memory.
- Mach12: an RS/6000 Enterprise Server S70 Advanced Model S7A. This is a 12 way, shared memory machine with 4 Gigabytes of memory.
- Cheyenne: a network of RS/6000 44P Model 270 machines connected via 100 mbps ethernet lan. Each machine is a 4-way SMP, operating at 375 MHz, with 8 Gigabytes RAM.

The same code can run on all the machines. The Cheyenne system requires that a process be started on a particular machine. Note: it is clear that 3 million subscriptions will not be satisfied by a single processor NT machine, but it is an easy machine to perform initial tests for code validity and to test some of the parameters (number of brokers, number of sessions, etc.) that can be varied with respect to the pub/sub system.

We performed the tests in the following order:

1) Get started quickly - use the same subscription and same publication in a loop. Vary some of the pub/sub parameters to get some initial performance information.
2) Test the end-to-end functionality from entering a request on the web page to receiving an eMail notification that our stock request was satisfied.
3) Perform the tests in step 1 but with varying subscriptions and publications. Also run tests with actual historical stock data.
4) Test the various hardware configurations to see what processing capability is needed to support 3 million stock requests.
5) Test the steady-state load on the Agent Server and pub/sub portion of the system. That is we will prime the system with 3 million subscriptions and ensure that we can create 100,000 new subscriptions per day and satisfy 100,000 a day.
6) Test the end-to-end functionality to ensure that the servlets and eMail can handle the initialization of 3 million subscriptions as well as the steady state load. We were not able to run this entire test on the same machine. We tested part of this goal on a lesser machine.

Test Results

The duration to enter subscriptions was measured to determine which configuration performs best. When more than one server/broker pair is used, there are multiple durations - one for each pair. For example if 12 stock requests were tested on 3 Agent Server/broker processes running in parallel, 4 stock requests would be given to each Agent Server and there would be 3 durations. The total duration is the max(duration(1), duration(2), duration(3)).

Results on Ducky

A fixed number of stock requests (10,000) were issued on Ducky varying the number of Gryphon brokers, Agent Servers, and sessions/broker (where a session has a thread). See the results in Table 1.

Table 1 - NT test results. 10,000 stock requests (subscription)

# Gryphon Brokers = # Agent Servers	#Sessions per Broker	Duration in seconds If brokers > 1 min,max
1	1	1,011
1	2	73
1	3	75
1	4	74
1	6	74
1	8	74
1	12	74
1	20	74
1	50	74
2	1	506, 507
2	2	67, 73
2	3	79, 81
2	6	75, 77
2	20	65, 74
3	3	84, 99

4	1	256, 259
4	2	68, 82
4	3	62, 84
4	10	66, 87
6	3	185, 320

Some tests were performed to determine if the same stock request was issued repeatedly, totally random stock requests were generated and issued, or stock requests issued from a historic feed affected performance. For the totally random feed, we generated a symbol table that contained all possible 3 letter sequences and indexed through it using varied prices. Initial results indicate that the method by which stock requests are generated has little impact on performance. See Table 2 below.

Table 2 -Various Stock Requests

	Duration in seconds to enter 5000 stock requests
one request repeated	38
random	39
requests from historical feed	38

We completed end-to-end testing and can accept a stock request from a web page and eMail a notification when the request is satisfied. We have not yet tested that we can send 100,000 eMails because we don't actually want 100,000 eMails to delete! We have tested that over 100 eMails can be sent when data is published rapidly.

We tested the publication rate which causes eMails to be sent. On NT, we can publish at 3.2 messages per second.

Mach12 and Cheyenne

We performed testing on the Mach12 and Cheyenne machines. Many of the initial tests were the same stock request issued repeatedly. When testing 3 million requests, we always used varying subscriptions.

Results on Mach12

We ran a number of tests on the Mach12. The primary goal was to load the system with 3 million stock requests so we could then test the effect of processing 100,000 stock requests per day (the steady state condition). We never quite reached 3 million, we did exceed 2.5 million.

See test results in Table 3 below.

Table 3 - Mach12 test results

# Gryphon Brokers = # Agent Servers	#Sessions per Broker (Total Sessions)	#Subscriptions (# per Agent Server)	Avg Duration / Max Duration in seconds	Test Failed: approximate # subscriptions completed
1	16 (16)	300k (300k)		>250,000
1	50 (50)	300k (300k)		>270,000
3	3 (9)	300k (100k)	7,142 / 7145	
3	8 (24)	300k (100k)	1,441 / 1488	
3	16 (48)	300k (100k)	761 / 769	
3	50 (150)	300k (100k)	472 / 481	
3	75 (225)	300k (100k)	482 / 508	
3	100 (300)	300k (100k)	470 / 486	
5	3 (15)	300k (60k)	7,143 / 7152	
5	32 (160)	300k (60k)	115 / 117	
5	60 (300)	300k (60k)	226 / 229	
5	75 (375)	300k (60k)	219 / 226	
5	100 (500)	300k (60k)	217 / 222	

5	32 (160)	3m (600k)		>1,280,000
10	3 (30)	1m (100k)	7209 / 7218	
10	3 (30)	300k (30k)	2103 / 2107	
10	50 (500)	300k (30k)	173 /175	
10	100 (1000)	300k (30k)		
10	50 (500)	3m (300k)		>1,050,000
15	16 (240)	3m (200k)		> 2,510,000
15	50 (750)	3m (200k)		> 2,580,000

Results on Cheyenne

Some tests were run with 4 machines of Cheyenne. The number of subscriptions approached but did not reach 3 million. Five machines of Cheyenne were needed. Fine tuning was not performed on Cheyenne due to availability. On Cheyenne we were able to reach our goals of 3 million subscriptions. See Table 4 below.

After 3 million stock requests were added as subscriptions on Cheyenne, 5000 publications were made, 1000 on each processor. The longest duration was 2.3 seconds (or 2173.9/second). Then 5000 publications on one processor took 6 seconds.

We then added 100,000 new stock requests to one of the 5 Agent Server/broker pairs (this causes one of the Cheyenne machines to get all the subscriptions). This took 864 seconds or 14.4 minutes.

Table 4 - Cheyenne test results

#Machines	# Gryphon Brokers = # Agent Servers	# Sessions per Broker	#Subscrip-tions	Max Duration in seconds	Test Failed: approximate # completed
1	1	15	200k	1,104	
4	4	50	3m		>2,920,000
4	4	75	3m		>2,840,000
5	5	50	3m	1,855	

Conclusions

Ducky conclusions

1) Using 2 or more sessions greatly improves performance over 1 session. Using more than 2 or 3 does not seem to improve performance. See Table 1.
2) Given 3 sessions per Gryphon broker, increasing the number of Agent Server/Gryphon broker pairs causes performance to degrade between 4 and 6 pairs. See Table 1.

Mach12 conclusions

1) In general, as the brokers and sessions increased on Mach12, so did the performance. See Table 3.

Cheyenne conclusions

1) Five machines were needed to reach the goal of 3 million stock requests. After reaching 3 million, satisfying 100,000 and accepting 100,000 new stock requests per day proved trivial. See Table 4.

Final conclusions

1) The assignment of threads to sessions greatly improves performance.
2) Our initial tests used the same subscription repeatedly. When we changed to use random subscriptions or subscriptions generated from historical stock trade data, we did not see a difference in performance
3) Our target for publishing was to handle peaks of approximately 2,200 msgs/second. We feel we were successful since we handled 2,174/second. After the 3 million subscriptions were entered on Cheyenne, 5000 publications were made, 1000 on each processor. The longest duration was 2.3 seconds. Then 5000 publications on one processor took 6 seconds (833.3/second). Clearly publishing to more machines increases the rate, and a sixth machine would push us over our target.
4) In a heavily loaded system, we could add 100,000 new stock requests in less than one day - in fact in 14.4 minutes.
5) We did not get to test the 100,000 eMails and web requests on Cheyenne, but we tested end-to-end functionality on NT. When publishing on NT, the

publication rate was 3.2/second. This is 11,520/hour and 276,480/day which meets our target of 100,000 publications per day, each of which cause an eMail to be sent.. Cheyenne would have much better performance.

Future work

We are planning a number of extensions to the current work. We will enhance the Agent Server to allow the service to notify a user when several different criteria are met simultaneously. For example, a user may request "notify me when IBM stock reaches 150, the Dow reaches 12,000, there is news about Lou Gerstner, and ...." We plan to extend the service to allow the user to be notified via a number of different pervasive devices. Currently, we can eMail, fax, print, and update a calendar.

We would like to look at other services to determine abstractions and components that could be provided to enable developers to create web services and agents more easily.

References

1. Enterprise JavaBeans, http://java.sun.com/products/ejb/index.html
2. Gryphon System Development, http://www.research.ibm.com/gryphon/System_Development/system_development.html
3. Gryphon Project, http://www.research.ibm.com/gryphon/Gryphon_ASSR/gryphon_assr.html
4. Java Message Service API, http://java.sun.com/products/jms/indx.html
5. Java Remote Method Invocation, http://java.sun.com/products/jdk/rmi/index.html