Engineering Persistent Queue System for a Unified Stock Transaction Platform

Jingcun Wang and Carol Blum

Information Technology Group, Bear, Sterns & Co. Inc.,
383 Madison Avenue, New York, 10179 NY, USA
jcwang@bear.com

Abstract. In building a unified platform for investors to trade securities globally, we need a persistent queue system to hold all the order handling messages before they are actually delivered to the corresponding stock exchange. This paper presents underlying principles of a persistent queue system based on the concept of journal, and its implementation in Java. We contrast performance of our queue with existing products, and show that the design meets the goals of fast I/O performance, specified messages purging, and timely crash recovery.

1 Introduction

Nowadays most secondary stock markets, whether based on auction system or over-the-counter [1], trade securities and derivatives electronically; and almost every stock exchange around the world has its own interface, no matter message or API (application programming interface) oriented, for clients to access its trading system. On the other hand, it is not uncommon for stocks of a company to be listed for trading on various stock exchanges, e.g., China Telecom is listed in Hong Kong as 0728, as well as in NYSE (New York Stock Exchange) as CHA [2]. Therefore, it is desirable for investors to have a uniform platform to access stock exchanges globally. To achieve that goal, we have implemented a system, called GOR (Global Order Routing); and Fig. 1 shows architecture of it.

There are numerous challenges, excluding legal issues and regulations imposed by respective stock exchange and country, to design and implement such a platform, among which a high-performance, scalable, persistent queue is of great importance. Conceptually, that queue should fulfill several functionalities as follows:

1) Preserve every incoming message, such as placing an order, modifying an order, subscribing to price of a specific symbol, and forward to the corresponding processor, while the latter accesses trading system of the specified exchange to complete the operation;
2) Cancel pending orders in the queue as well as in the order book of the stock exchange. In case there is an error input, that order, if not filled, should be cancelled immediately when trader/clerk realizes that.

3) Crash recovery. In case GOR crashes, all pending orders should be recovered quickly and resent to the corresponding stock exchange.

While PERSISTENT message in JMS (Java Message Service) [3] seems to fit the requirement, it lacks the interface to cancel a message in the queue and proprietary products prohibit us from further customizing. On the other hand, the queue deployed in open-source MTA (mail transfer agent) qmail [4] relies heavily on UNIX file system, and it is difficult for that queue to be scalable and portable.

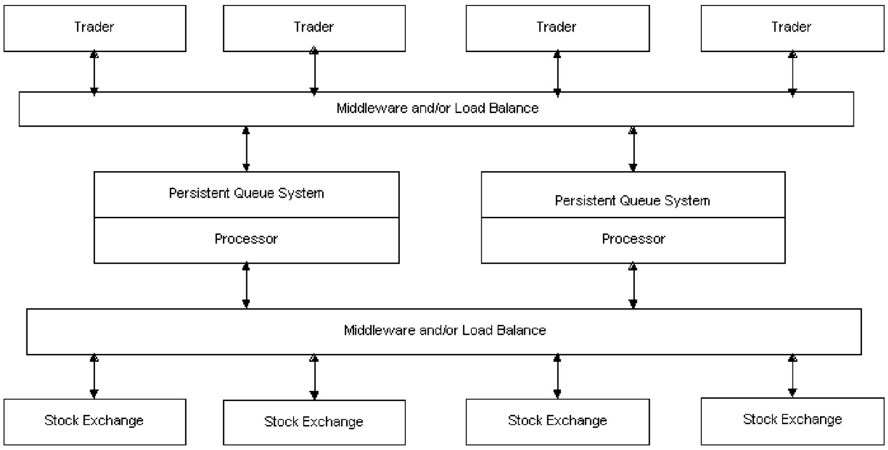

Fig. 1. N-tier Architecture of GOR

In this paper, we propose a high-performance file-system based queue to complete all the requirements. Meanwhile, a prototype has also been implemented by Java in application level for GOR.

Rest of this paper is organized as such. Section 2 presents principle and implementation details of the persistent queue system. Then in Section 3 we provide some simulation results of the system compared with SUN ONE Message Queue. The final section gives concluding remarks and some future directions.

2 Persistent Queue System

A Persistent Queue System (PQS) comprises of multiple queue blocks and a system control file, which records number of queue blocks deployed in current system as well as size (SZ) and location (PATH) of each queue block.

2.1 Queue Block Structure

Physically a queue block is a file either on local file system or mounted on an NFS file server. When PQS starts up, it generates a file with SZ Megabytes at PATH designated in the PQS control file. So, from file system perspective of view, except modification time the queue block's directory information will not change during lifetime of PQS.

A queue block (QB) consists of four elements: header, checkpoint, data area, and, tail. Refer to Fig. 2 for detail illustration.

1) Header describes structural information for this QB. It contains positions of checkpoints and starting point of the data area among other parameters.
2) Checkpoint is used to reduce recovery time. There are two checkpoints within each queue block, and they are written alternatively. If PQS crashes when writing one checkpoint, the other can be used to recover.
   The more frequent PQS updates checkpoint, the less time PQS will spend on recovery. However, checkpoint update is a synchronous I/O, which will degrade PQS throughput. So, we need to pick an optimum interval time between each checkpoint update, and that parameter is configurable.
3) Data Area is composed of segments, and a segment contains many trading messages. In GOR, we use FIX (Financial Information Exchange Protocol) [5] to generate exchange specific message; and every segment can contain several of such messages, called item. Along with the real content, every item has a delivery status to denote if it is cancelled or delivered or pending. No trading messages can span over multiple queues.
4) Tail serves for data verification only.

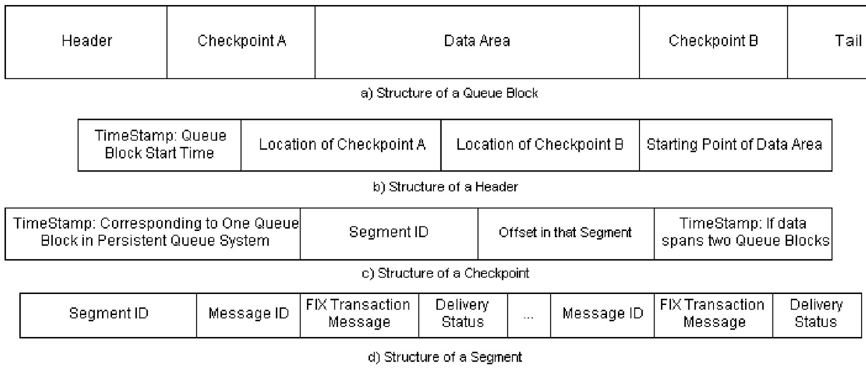

Fig. 2. Structure of a Queue Block in detail

2.2 Incoming Message Handling

When a trader triggers an action, PQS will accept the incoming message sent out from the trader client system and save it into the file system. For instance, if a trader wants to place a market order to buy 100 shares of IBM, trader client system will send out the FIX message via a function call, and the function call will not finish until PQS sends back an acknowledgement. After trader client system gets the acknowledgement, even if GOR crashes, that order is saved and will be re-placed when GOR recovers in the same day (because of high liquidity of stock markets and some SEC – Securities and Exchange Commission regulations, pending orders will only make sense on the same day that order placed). And later when stock exchange actually processes the order routed by GOR, trader client system will receive information via registered callback functions.

In order to minimize numbers of disk operation, which is synchronous and time-consuming, PQS holds all the incoming messages until a journal (an internal buffer) filled or timeout before flushing to file system, and then acknowledges the trader client systems. The following lists pseudo-code for the saving process.

```
1. Start a size pre-specified buffer, called a journal;
2. Accept incoming messages, assign each with a unique message id and put it into that journal;
3. If the journal is not full and not timeout, go to Step 2;
4. Assemble contents of the journal to a segment, and flush it to PQS;
5. Acknowledge each trader client system along with the message id.
```

Generally there may be several queue blocks in PQS. When space of one QB is used up, the journal will be flushed to another. However, that's impossible to have infinite disk space, so we need to recycle queue blocks.

Each queue block has an active message count and range of message ids. When a message is active, it means that message has not been routed to the specified exchange yet. Thus, if active message count drops to zero, that queue block is eligible to be recycled.

2.3 Specified Message Purging

Erroneous orders can result in increased market volatility and significant financial risk and exposure to members and member organization [6], so although order cancellation is not frequent it is of great importance.

If a trader wants to cancel all orders, all queue blocks will be locked, and active message count will be set to 0; in the meantime, corresponding CancelOrder instruction will send to stock exchanges to cancel those already submitted but still remain in the order-book (not filled).

If a trader wants to cancel a specified order with the message id obtained previously, PQS will go through the following steps:

1. Find the corresponding queue block according to range of message id associated with each one;
2. Lock the queue block, and find the message in that queue block;
3. If not delivered, set the delivery status to CANCELLED, and decrement the active message count;
4. If already delivered, forward that request to the specified stock exchange.

2.4 Crash Recovery

PQS might crash because of hardware or software reasons. We want to minimize recovery time so that new incoming messages can be accepted as soon as possible. And, in fact structure of queue block is designed to achieve timely crash recovery. The recovery involves the following steps:

1. Get PATH of each queue block from PQS control file;
2. Retrieve information of each queue block from its head/tail information;
3. Find the queue block with the latest timestamp. And use checkpoint of that queue block to locate the segment being checked;
4. Roll forward from that segment until the end of that queue block.

3 Simulation Study

The Sun[tm] ONE Message Queue is an implementation of the Java[tm] Message Service (JMS) specification, an industry-standard API for handling reliable, asynchronous messaging between Java applications; and according to benchmark comparison, it is 7 times faster than IBM WebSphere Java Message Service Provider [7].

The testing environment is one SUN-Fire-280R Server with 4GB memory; and the two subjects are SUN ONE Application Server 7 integrated SUN ONE Message Queue 3.0.1 system (A) and GOR PQS (B) .

Firstly, we send persistent messages to a queue in A, and the same sized messages to B. This case measures how fast a queue can accept messages. For clarification, we normalize result of A as 1, and result of B is ratio of A. Table 1 lists message per second of B compared with A.

Table 1. Measurement results for PQS on message accepting

Message Size	1K	2K	4K	8K	10K
Messages per Second of B	1.07	1.11	1.19	1.25	1.30

Secondly, we deliver persistent messages from a queue in A, and the same sized messages in B. This case measures how fast a queue can deliver messages.

Table 2. Measurement results for PQS on message delivery

Message Size	1K	2K	4K	8K	10K
Messages per Second of B	1.00	1.00	1.01	1.02	1.02

4 Conclusions and Future Work

In this paper we describe in detail the persistent queue system deployed in one version of our unified stock exchange platform – Global Order Routing system. Similar to concept of journal in database management system, this persistent queue system holds incoming messages in an internal journal, flush it to file system while updating the checkpoint; meanwhile, it also supports the functionality to purge a specified message in the queue. Simulation study shows that averagely throughput of the queue system can be 5 -10% better than proprietary products.

This is the first step to implement a unified stock exchange platform. In our future work, we plan to extend that platform to dynamically route orders to the best stock exchange to trade, while publish/subscribe market data in real-time.

References

1. Frank J. Fabozzi, Franco Modigliani: Capital Markets: institutions and instruments. 3rd ed. Prentice Hall, Upper Saddle River NJ (2002)
2. http://www.chinatelecom.com.cn/
3. SUN Microsystems: Java Message Service Specification, Version 1.1. SUN Microsystems, Palo Alto CA (April 12, 2002)
4. Dave Sill: The qmail Handbook. Apress, Berkeley CA (2001)
5. FIX Protocol specification, Version 4.3. http://www.fixprotocol.org (August, 2001)
6. Nicole Maestri: NYSE tells firms to guard against erroneous orders. Reuters, New York NY (December 18, 2002)
7. SUN Microsystems Web Site: http://www.sun.com.