Market Data Publisher – In a High Frequency Trading Set-up

INTRODUCTION

The main theme behind the design of Market Data Publisher is to make the latest trade & book data available to several integrating applications at pre-configured intervals (1sec or smaller), at the same time. Majority of the use-cases lies with the front-end presentation tools, like Microsoft Excel (via RTD), that would let users (like traders) view data from different exchanges on their desktop. In addition to them, there are other applications, especially in a high-frequency trading set-up, which would require access to market data, like a Position server or a Risk Monitor application. These apps would consume to compute values like un-realized P/L, Pos PL, MV, Exposures, Average prices etc in real-time.

At Saven we address this need by designing a custom, C++-based Publisher application for one of the High-frequency trading firms. This application accesses trade and book information from a co-located box (Market Data solution from Celoxica) and publishing the same onto a messaging layer on fixed topics at periodic intervals. Applications that would need access to the market data will subscribe to these topics with the messaging layer. This application is designed to support high volume and high speed data transfers at times only limited by the subscribers' consumption speed. It uses one of the highly robust, widely used, open-source messaging frameworks – ActiveMQ (v5.3.2) from Apache.

Goal: The application's broad goal is to publish the market data for 1000+ symbols on 4 exchanges (ARCA,NSDQ, BATS, DIRECT EDGE) so that consumers (either downstream apps like Risk Monitor or traders' Excel workbook via RTD) can access real time data for an extended period (8-10 hrs Session).

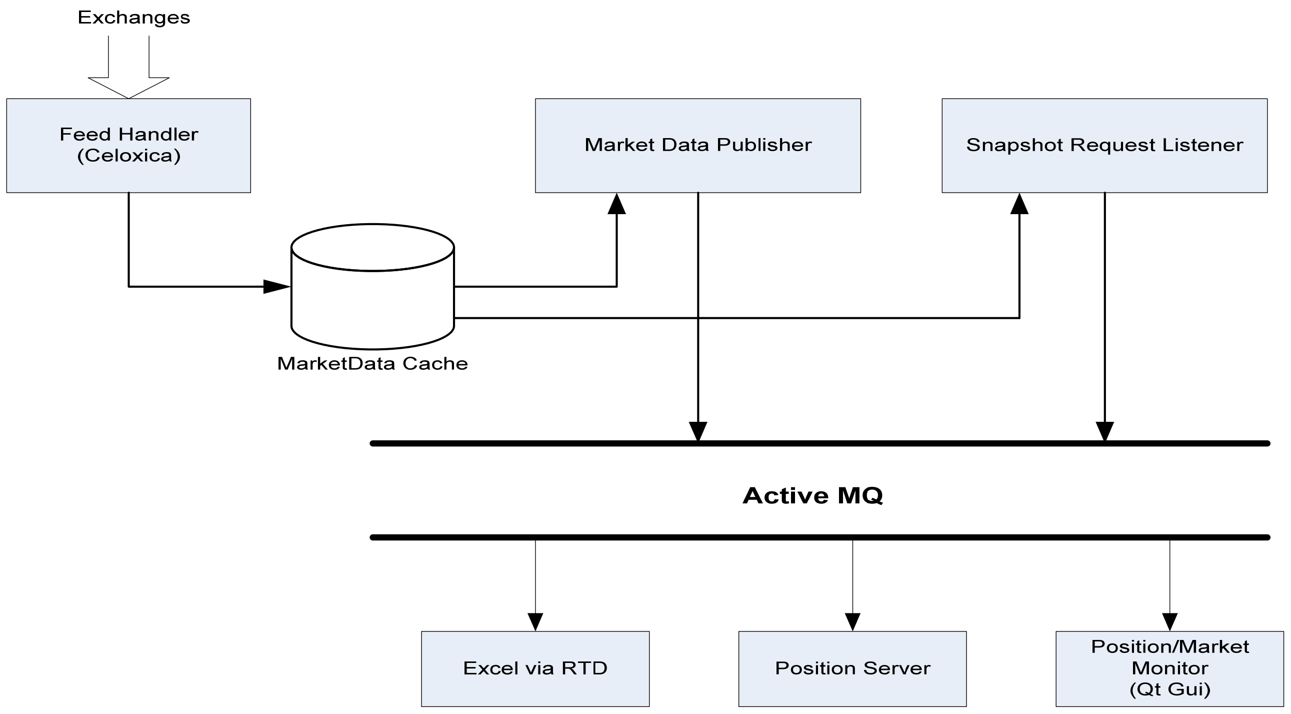
BASIC ARCHITECTURE

The primary requirement for Market Data Publisher is to support high speed, high-volume data transfers close to a magnitude of ~65K messages/sec during peak time, each message weighing approximately 100-900 bytes (average between trade and 10 deep book data). The Publisher has to accommodate approximately 5 exchanges and 6500 symbols per exchange with two types of market data published per symbol, viz., trade and order book up to 10 deep.

Below are some of the design decisions taken towards achieving these goals?

- Gather data from a feed-handler that is responsible for collecting and computing trade/book info on a separate thread without hindering the main publishing work-flow.
- Store the data in a thread-safe local cache in a most compact format possible.
- Provide thread-safety at the record-level only rather than at the entire table or cache level. [One table per exchange.]
- Main Publisher would sweep through the cache at a pre-configured interval (every 1 sec) and publish only the latest updated values on appropriate topics.
- One topic per exchange-Market data type (trade or order book) combination.
- A separate channel for initializing consumers that are joining late.

Following schematic picture describes the basic structure of the Market Data Publisher:



ARCHITECTURE DETAILS:

Application: Feed-handlers are very fast in their execution using hardware technologies like FPGAs providing ultra-low latency data transfers. We used Feed Handlers from Celoxica which uses FPGAs for feed handling and C APIs for software interfacing. They collect ticks and compute market data at a very fast rate in the order of nano seconds latency. Though these rates are suitable for trading engines they have to be slowed down for consumers like presentation tools & risk calculators to a large degree. These apps typically consume data every second(s). Our design took care of this by keeping the feed-handler on a separate thread from the main Publisher's and separately controlling their rates. The Publisher swept through the Cache and published only the last updated values. This kept the amount of data published at a reasonable limit as there would be a lot of messages for all the required exchange/symbol combinations. These values are published onto ActiveMQ message layer on appropriate topics.

Majority of the performance tuning parameters specific to Market Data Publisher is controlled through configuration settings external to the application. For example, the time-to-live for every message (1 sec default), Publisher's sweep interval (1 sec default) and book-depth (10 default) are some of them. They help in fast delivery of messages, accommodating slow consumers, and preventing running out-of-memory.

Late-joining consumers are accommodated with an asynchronous request/response channel to provide the latest snap-shot. Once up-to-date they start listening on appropriate topics for live updates.

The main entity – the data store or the Market Data Cache is a custom designed, group of hash tables, one per exchange and indexed by 'Symbol Name'. It provides constant access time (O(1)), record-level locking for the data and has been designed to efficiently reuse the space allocated. These improve speed and ensure data integrity.

Message Layer: ActiveMQ is a robust, stable JMS compliant messaging framework widely used in enterprise products. We use version 5.3.2 of ActiveMQ as the broker between the Publisher and the integrating applications. Since the volume of data published is huge there are some tunings needed at the broker as well as on the applications side. Every message is published with a time-to-live value of 1 second to avoid clogging the broker. Consumers can take some message loss due to expiration. NOTE: If the consumers are fast enough this would never be the case.

All messages are set to be non-persistent and asynchronous mode of sending is used. These boost the speed and efficiency of message transfers.

Since some of the consumers are over the network and running desktop applications (like Excel), their speed is much slower relative to the Publisher's. To cater to such slow consumers Producer Flow Control is turned on with a memory limit ~30MB at the broker. This would throttle down the publisher rate till the slow consumers catch up. Also, in cases where messages are pending at the dispatch queue the broker is configured to purge the old messages.

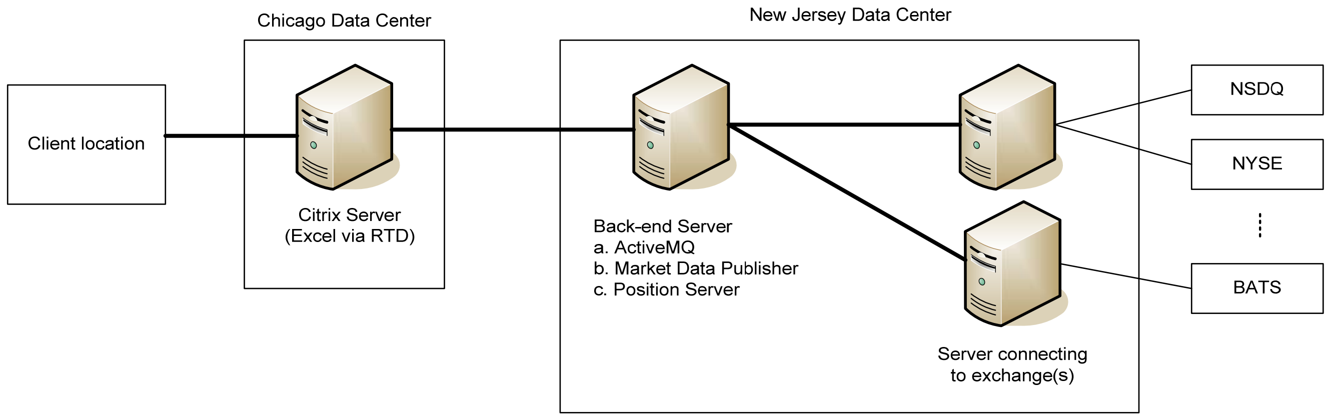
Consumers: The application is designed to support as many number of consumers as ActiveMQ can support. Usually it's a pretty big number. In our case, we have three applications that consume data from the Market Data Publisher – Microsoft Excel through RTD, Position Server and Monitor GUI.

Excel via RTD: Most of the traders are familiar and comfortable using Microsoft office tools like Excel. To bring the real-time data feed to Excel a custom RTD server is designed to consume the Market Data from the messaging layer. It first fetches the snap-shot data, initializes its internal cache, and starts listening for updates on the topics. Every periodic interval (pre-configured through a setting) it pushes the updates to Excel.

Position Server: Another consumer to the Market Data Publisher is the Position Server. It subscribes only to the trade data for its calculations. It fetches the trade data for all the symbols on all the exchanges and using these values computes the un-realized P/L values for the trades. It also fetches the initial values from the snap-shot channel.

Monitor GUI (Qt-based): This application is a cross-platform, custom presentation tool bringing the live market data, position information onto the desktop of the users. This is a Qt-based application and it is in the initial stages of prototyping.

Network Diagram: The following figure briefly captures the deployments of various applications across different locations:



In the above diagram some of the connectivity to the exchanges was co-located at the exchange centers itself. It is not depicted in the figure.

TEST SUMMARY

The application has been tested with both live and mock data to ensure its stability and to increase the confidence on the product. Some of the test results below:

Tests with Mock Data:

- 5 exchanges, 6500 symbols per exchange.
- Mock Feed Handler generating both Trade and Order Book data (Totally 10 topics).
- Ran for more than 8hours duration with one local Mock Subscribers and 3 Citrix based subscribers.
- Tests were repeated for several days.
- ~20 million messages published and ~44 million messages consumed on each day

Tests with Live Data:

- Subscribed to all SPY 500 symbols on NASDAQ and 1200 symbols on ARCA.
- Celoxica Feed Handler generating both Trade and Order Book data.
- Ran for close to 8 hours without any interruption.
- Tests were repeated for several days with 4 Citrix subscribers consuming data in EXCEL via RTD.

SUMMARY

Following table summarizes various performance and load points of the Market Data Publisher product:

Max. messages Published	> 20 million
Max. simultaneous Subscribers	5 (3 on Citrix; 2 on Linux)
Max. messages subscribed	> 44 million
Max. session duration	> 8 hours
Max. Exchanges	5 (mock test); 2 (live test)
Max. Symbols/exchange	6500 (mock test); 1200 (live test)
Max. Book Depth	10
Market Data Cache size (min)	479KB (500 symbols/2 exchanges/2 deep)
Market Data Cache size (max)	49.9MB (6.5K symbols/5 exchanges/10 deep)

The application has been tested on following platform:

Back-end: 16 cores, Intel Xeon CPU X5570 @2.93GHZ, 9.66GB RAM/127GB Disk Size, GNU/Linux; x86_64 architecture;

Front-end: Citrix Presentation Server on Microsoft Windows Server 2003 R2, Intel Xeon CPU 5160 @3.00GHZ, 3.24GB RAM/67.6GB Disk Size

By choosing appropriate technologies (like C++, ActiveMQ) and making right design choices we successfully designed, developed, and delivered the Market Data Publisher satisfying the most challenging functional & performance requirements. The Publisher was tested both with the live and mock data of various volume to ensure that it was able to withstand the load. The Publisher was tested for the absence of memory leaks as it would run starting pre-market hours through post-market hours. Any leak would prove disastrous for applications running this long.

TEAM:

Murthy Gudipati, Saven Technologies, mgudipati@saven.in
Sridhar Chelikani, Saven Technologies, schelikani@saven.in

Contact: Saven Technologies, 1051 Perimeter Drive, Suite 1175, Schaumburg, IL 60173.
www.saven.in, 877-728-3621, chicago@saven.in