Software and Hardware Optimisation Patterns in Financial Exchanges

Name: Karl O'Brien

Email: karlobrien@gmail.com

1 Introduction

"If you're reading this, it's too late: a machine got here first"

The Financial Times, April 16, 2007, p. 1, [1]

"Moving Markets: Shifts in trading patterns are making technology ever more important"

The Economist, Feb 2, 2006, [2]

This term paper will discuss the use of software and hardware optimisations in financial software, typical patterns that are emerging from this optimisation. It will also cover a typical pattern in how data is published from one of Europe's top exchanges.

The report will also outline how the financial industry is constantly looking at how it can become faster; reusing solutions from a number of different industries such as computer game development. The financial software industry tends to be an early adopter of technology in a bid to gain an advantage on the opposition. The quote from the Economist above is from an article describing how important technology is becoming in trading patterns. The situation described where a trader sends 1,000 small buy orders to the exchange, the exchange instantly responds, then a split second later, 99% of the orders are cancelled, the trader has found the best price from the 1,000 orders submitted and withdraws the rest of the orders. This paper will describe both sides of the technology cycle – receiving the data from the exchange and then placing the order.

Imagine a situation like the one described above happening thousands of times a day. There are two sides to this:

1) The exchange must employ both hardware and software that can deal with the overheads that these orders generate
2) The trader must constantly find methods to keeping finding the best price – technology will play an important part of this process.

This paper will discuss a number of technologies, how those technologies are being used across a completely different industry.

2 Financial Systems

There are two sides to technology when referring to stock exchanges.

2.1 Market Data

Market Data is numerical price data, reported from a stock exchange regarding a particular product. Decisions as to buy or sell are made based on the value of market data. A typical market data message might appear like the following:

Ticker Symbol	Vodafone
Bid Price	10.23
Ask Price	10.46
Bid Size	100
Ask Size	300
Last Sale	10.30
Last Size	250
Quote Time	14:32:45
Trade Time	14:32:30
Exchange	LSE
Volume	4573

Fig 1 Message from London Stock Exchange

In recent years with the onset of automated trading there has been an explosion in the amount of messages such as those in figure 1 being sent. The delivery of this data is more often than not carried out in real time as the trader will require that the data is sent from the exchange into the trading application as quickly as possible in order to make a decision on the data. As you can imagine the sheer volume of this data puts tremendous pressure on both the hardware and software side of technology.

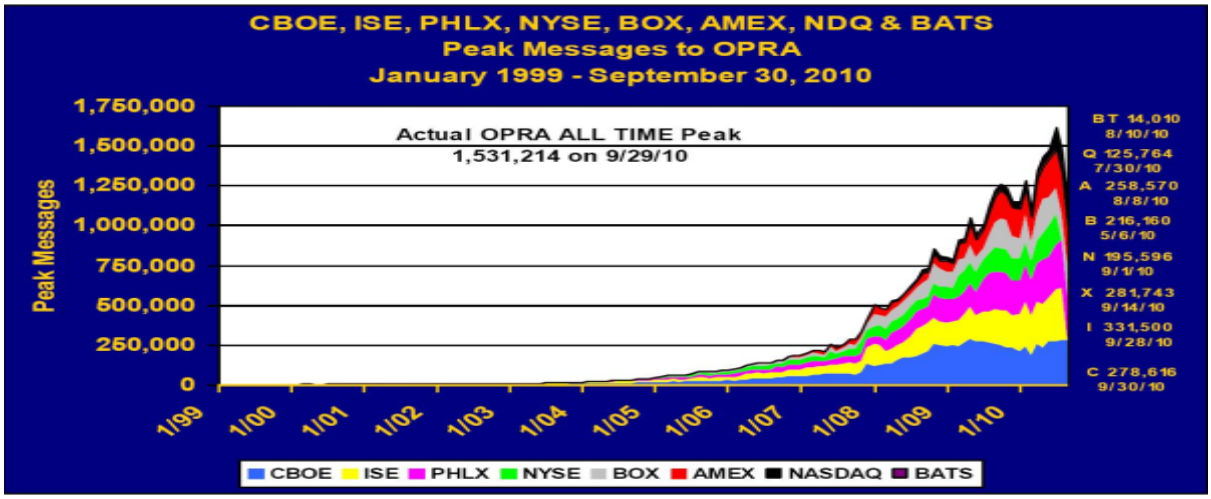

Fig 2. Market Rates from various US exchanges [3]

From Fig 2, it is very obvious that Market Data rates are increasing exponentially each year. The challenge for technology groups is to put systems in place to manage this data, to bring the data from source to destination without dropping any packets (messages).

2.2 Order Routing

The second side to technology is called Order Routing. Once the data is pulled from the exchange and placed in the view of the trading application, another race begins to get the response back to the exchange as quickly as possible in order to secure a trade at the current price.

2.3 Data Standards:

In order to avoid each exchange creating its own protocol for publishing messages from publisher to subscriber, the industry has come up with a number of standards for sending messages. One such example is the FIX protocol [18]. This allows the exchange to publish in either TCP or multicast and the subscriber will have a common interface to subscribe to

2.4 Case Study: Deutsche Boerse Eurex

Tracking the path of market data

2.4.1 Section 1: Exchange to Consumer Site.

It is possible to track the path of an option contract from being published on the German Stock Exchange [4] to delivery at a client site. When building the system, the designers faced a problem; how do you publish vast amounts of real time data without dropping any messages? The obvious answer here is use a standard TCP connection, but this means that you are limited by the size of the TCP channel and the exchange will also have to create and maintain a separate TCP channel for each user that connects to the exchange. The designers used a system that can trace its roots to the observer pattern commonly seen in software engineering circles [17]. The exchange (Eurex) would act as the publisher and the traders would act as the subscribers. The primary role of the exchange is to publish the data in a reliable fashion and it is down to the client to make sure the data reaches them.

In order to further separate the exchange from the customer, the data transmission format is published over multicast data instead of TCP. Multicast enables the exchange to stream its data continuously in a single transmission to a group of destination computers in a single transmission. This significantly reduces the loading that the exchange will have to manage. Multicast is also built on top of the User Datagram Protocol (UDP). There is no handshaking involved with this protocol; UDP assumes that any error checking is performed in the application itself.

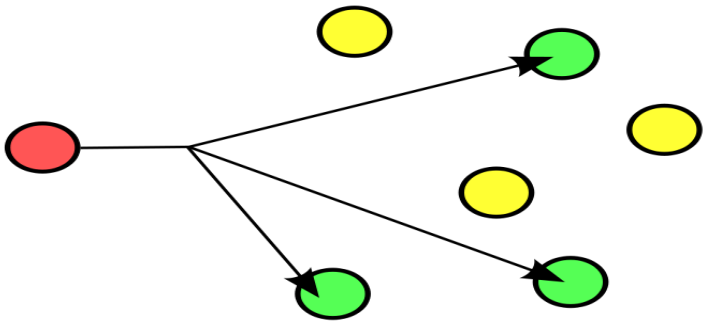

Fig 3. Outline of multicast publishing.

Multicast also allows the exchange to break up its data in various multicast channels. This means that instead of maintaining state for every TCP connection, the consumer indicates to the exchange which multicast channel it is interested in and joins the stream to listen to the data the exchange is publishing. If a consumer is not interested in a particular product it will not connect to that channel and so will not see any data. This reduces the amount of data that the consumer has to handle. It also makes the system very scalable for the exchange. They can add new products to separate channels without affecting existing users.

2.4.2 Section 2: Consumer Site to Client (Trader Application).

To deal with client connections, there is the concept of a feed handler. This is a host (usually a high powered UNIX host) which will connect to the exchange channels described in section 1 and distributes the data down to the clients (trader applications). This host can maintain state; keep track of updates from the exchange and publish to the client. It can also aggregate updates to shield the client from unnecessary data. This feed handler is another example of an observer pattern. The client subscribes for certain instruments and can also unsubscribe itself. Only clients that are subscribing to updates will be able to see data coming from the exchange. It acts a proxy between the clients and the exchange itself. Some common feed handler vendors are NYSE with their wombat platform [19] and Thompson Reuters using RMDS [20]. The main goal of these systems is to provide a common interface for applications to connect to.

3 Common Patterns

3.1 Standard Trading Day

Take a standard trading day for example. From a European point of view the day can be broken down as follows:

Open: 08:00

US Open: 14:30

Close: 16:30

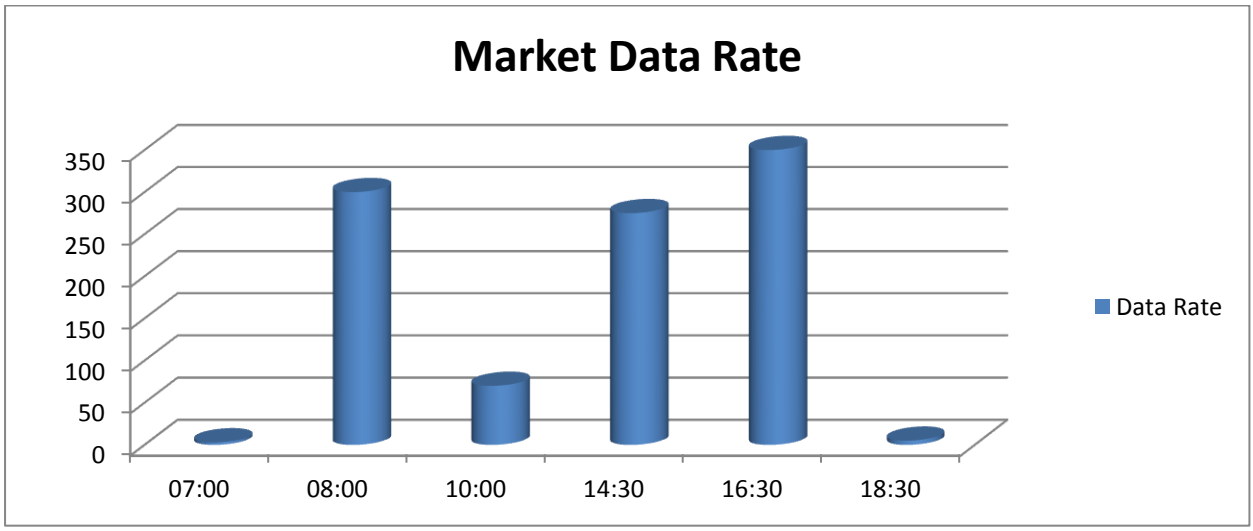

Fig 4. Typical Data Rate during the day.

When the data rates are mapped over the course of the day, it is possible to anticipate the busy periods; the start of the day will always see a surge in data as the market reacts to overnight events, 14:30 is also busy as the happenings on the US stock exchanges will filter down to the European markets. 16:30 is statistically the busiest period as the market prepares to close for that day. In terms of the feed handlers described above, a pattern in the usage of the data becomes very important. The average value of the data rates is not important; it is the spikes in data that is the most important. The ability to jump from a 3 MB/s before the open to 300 MB/s at 08:00 is the key. If the feed is tuned to withstand these events then none of the downstream consumers will be affected by the jumps in data rates. This usage pattern is also evident in other industries – in online gambling, a lucrative poker tournament would see huge spikes in online gaming traffic. If a user is experiencing slow frame rates in online gaming, this could be related to congestion.

3.2 Reacting to Events.

Another pattern in the financial world is the dissemination of news. The quicker a trader is updated with a news event, the quicker they can place their stock into a position to take advantage of the effects of the news. In previous sections a description of a feed handler was given which published prices. There is now the concept of news based feed handlers [5]. This was evident in recent events such as the earthquake in Japan [6]. News events like this are hitting the financial world milliseconds

after the event. The growth in services such as twitter also means that events happening in Asia can affect almost instantly markets in Europe. Items like Need-To-Know news can be used in systems such as Complex Event Processing (CEP) [7]. In this situation an application can be configured to listen to specific news events and react in a certain automated manner.

This pattern of events is also used by regulators to scan for suspicious activity – i.e. large trades occurring just before a takeover announcement. In an interesting take on the presence of news feed handlers, it could come about that instead of a journalist picking up a story and publishing and the markets reacting to that story, the algorithmic trades might pick up on the news events and react in a certain way and journalists would get the story from reading the trades.

3.3 Hardware Patterns

3.3.1 Data mining of System Statistics

Another side to seeing high spikes in data rates and interacting with busy financial markets is trying to predict how your systems will react to the massive leaps in activity. Buying the latest hardware does not always solve the problem as utilising a top of the range Intel processor only to be limited by the messaging protocol being used between hosts is a common problem. It is not only hugely important to be able to withstand a large event on a financial market but it is also very useful to have detailed statistics outlining how each part of your enterprise system coped with the event. The ability to look at the data and identify patterns where the network card is causing a bottleneck instead of CPU or memory can enable an organisation to make instant decisions on system architecture. An interesting development is that companies such as Facebook are actively pursuing a similar idea. An open source project called scribe [8] enables a server to aggregate logging data streamed in real time from a large number of servers. A system like this enables Facebook to access performance data at blinding speed from entire data centre's all at once. When describing the architecture of the system, Facebook maintained that the decisions not to lock into a particular network topology, system reliability and a simple data model were very important factors. What this shows is that with the speed at which environments are changing in the present climate, it is important that an organisation can change its focus to new platforms and still have efficient access to its underlying data.

3.3.2 Middleware Components

In financial systems a very common usage pattern is once you have the data from the exchange, how do you get it to the users as quickly as possible while also maintaining reliability? One of the major players in this space is TIBCO [9]. Tibco Rendezvous is a product whose purpose is to transfer data between client and host machine. Running as a daemon process on a client host, the process could speak to a daemon on the server host allowing an efficient transfer of data. This usage pattern is referred to as a broker model as each host has no idea about the location of another host. They only need to know the address of the broker. The application can send a message and forget about it, if the application fails the broker will still retain knowledge of the message. The broker system works well on a congestion free system but when the system is placed in an unpredictable environment, a

well know issue in the financial sector – “RV Storms” where one particular host gets into difficulties and uses up the bandwidth of the entire system causing an outage for all hosts.

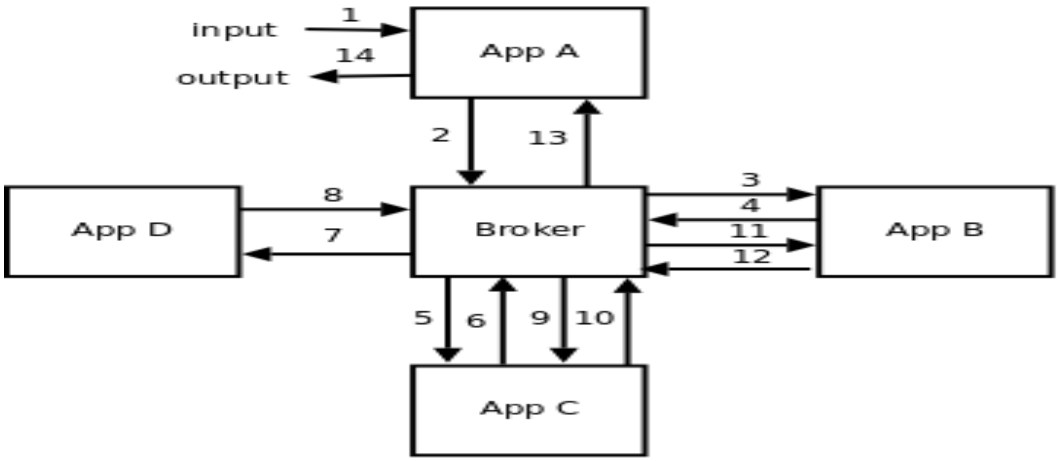

Fig 5. Broker Model [10]

The “No Broker” model moves to a system of sending messages without needing a broker in the middle. In latency sensitive situations, this model has tremendous advantages, the ability for applications to talk directly without a broker in the middle. A major player in this area has been a company called 29west [12]. The product called Ultra Messaging Streaming (UMS) is a hugely configurable and low latency messaging solution.

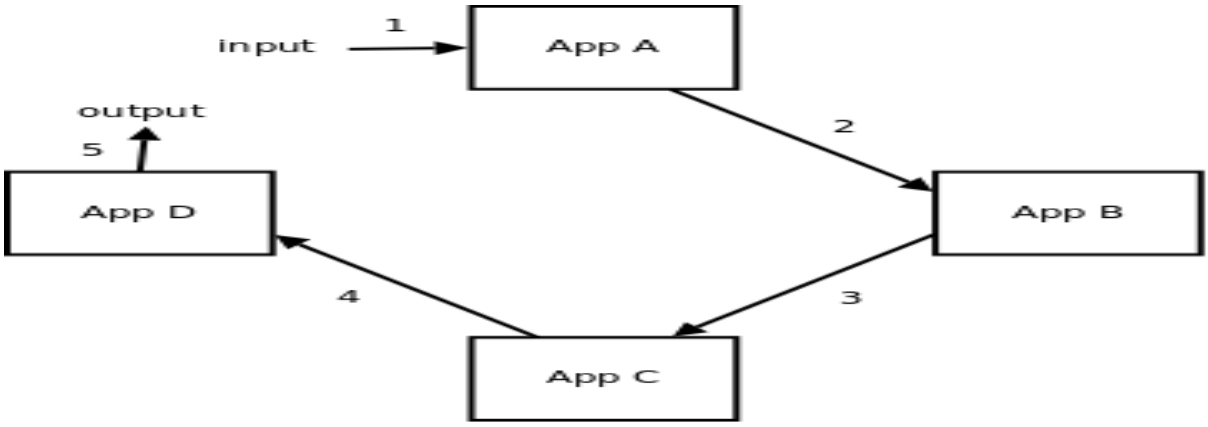

Fig 6. UMS solution [11]

As in Fig 6 UMS allows each application to talk directly to another without the need for a broker in the middle. This approach tends to use more CPU and memory than the “broker model” as the applications have to maintain state but with recent gains in hardware specifications and the cheaper cost of hardware itself; this has not been an issue.

Again in this situation, it can be seen that this technology is making its way into other technology sectors. Google has a messaging protocol - Protocol Buffers [13]. Google had an issue transferring data between large numbers of machines and used ideas first seen in financial sectors to solve its problem. Facebook also has a similar technology in thrift [14].

4 Conclusions and Future Trending of Data

Each pattern show so far in this report is talking about increased data rates and how to successfully capture all this data and distribute to clients without any issues. The amount of data now being captured on a daily basis is also increasing. An entry in the Science Journal calculated that the amount of data stored in the world as 295 Exabyte's. [15]. It is now a huge challenge to industries to make sense of the data that is captured. Technologies such as Map Reduce [16] are enabling companies to make sense of data a lot quicker. A crucial side to the amount of data being generated is to be able to predict with some degree of certainty how much data will be needed at certain points in the future. Knowing these limits will enable organisations to plan for future hardware and software optimisations.

It is interesting to note that there is an increasing move into non-traditional domains, areas where software is meeting with hardware. Systems like FPGA cards are now increasing in use. These systems allow all processing to be carried out in a separate environment to the Operating System; the capture and parsing of data is no longer subjected to scheduling by the OS but will happen as a constant on the FPGA card. There is a growth in systems such as TCP offload engines and kernel bypass [21] where traditional methods of parsing TCP in the OS is no longer fast enough. Similarly the growth of multicore processors is now introducing a different programming model where the need to split loads efficiently across multiple cores is a common pattern.

A large area of my work is in dealing effectively with large rates of data coming from various exchanges. One of the key messages mentioned above is dealing with spikes in data effectively. Once you can withstand the largest spike, you can have a certain degree of confidence in the system. This is following a similar pattern in database design, load testing websites – planning for the worst case is often the best policy. Another large growth area and briefly mentioned above is the move to multicore systems. Although it has been touted for years now, concurrent programming is still quite difficult to implement. It still requires knowledge of threading, locks and race conditions. Design Patterns perhaps hold the key to simplifying the problem. If a common framework can be put in place for multicore programming – a design pattern can often encompass best practise passed down from experienced engineers. An example of such a pattern is the Active Object pattern [21]. As concurrent programming becomes more widespread, more patterns in this area will be defined thereby increasing the knowledge surrounding multicore programming.

References

1	FT	http://www.ft.com/cms/s/bb570626-ebb6-11db-b290-000b5df10621.html
2	Ecomonist	http://www.economist.com/node/5475381?story_id=E1_VQSVPRT
3	Fig 2.	http://marketdata.exegy.com/blog/bid/48654/Long-view-on-market-data-1990-2011-from-derivatives-exchanges
4	Eurex	http://www.eurexchange.com/index.html
5	NTKN	http://www.ntkn.com
6	JAPAN	http://www.bbc.co.uk/news/world-asia-pacific-12711226
7	CEP	http://www.thecepblog.com/what-is-complex-event-processing
8	Scribe	https://github.com/facebook/scribe
9	RV	http://www.tibco.com
10	Broker	http://www.zeromq.org/whitepapers:brokerless
11	Broker-less	http://www.zeromq.org/whitepapers:brokerless
12	UMS	http://www.29west.com
13	Proto	http://code.google.com/p/protobuf
14	Thrift	http://incubator.apache.org/thrift
15	BBC	http://www.bbc.co.uk/news/technology-12419672
16	Mapreduce	http://static.googleusercontent.com/external_content/untrusted_dlcp/labs.google.com/en//papers/mapreduce-osdi04.pdf
17	GOF	http://www.amazon.co.uk/Design-patterns-elements-reusable-object-oriented/dp/0201633612/ref=sr_1_1?s=books&ie=UTF8&qid=1303465198&sr=1-1
18	FIX	http://www.fixprotocol.org/specifications
19	NYSE	http://www.nyse.com/technologies/1207087035761.html
20	Reuters	https://customers.reuters.com/Home/RMDS.aspx
21	TCP Offload	http://ttthebear.blogspot.com/2008/07/linux-kernel-bypass-and-performance.html
22	Active Object	http://www.cs.wustl.edu/~schmidt/PDF/Act-Obj.pdf