BNP Paribas: Equity Smart Order Router

A Major Qualifying Project
For BNP Paribas, New York

Submitted to the faculty of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements of the
Degree of Bachelor of Science

by

Xiaoyun Wang, Actuarial Mathematics
Huan Tu Lai, Computer Science

Project Sponsor: BNP Paribas

Submitted to:
On-Site Liaisons: Scott Visconti
Christophe Poulmarc'k
David Jobet

Project Advisors: Professor Arthur Gerstenfeld, Department of Management
Professor Dan Dougherty, Department of Computer Science
Professor Jon Abraham, Department of Mathematics

Submitted on

Thursday, December 17, 2010

Table of Contents

Abstract

Blizzard is BNP Paribas' smart order router implementation, a system that takes order requests from clients and algorithmically executes those orders. The goal of this project is to develop a general purpose visualization toolkit that can be used to dynamically display and monitor the status of Blizzard. This tool, the Blizzard Visualization Tool (BVT), will provide an easy way to detect anomalies and find the impact of factors such as router and market latency and market movement on the performance of the router, as measured by the metric of the fill ratio, or the percent of orders successfully executed. A second goal of this project is to use the toolkit, in addition to statistical analysis, to find improvements for the algorithms in place.

Acknowledgements

We would like to thank all of the people who have made the success of this project possible. We would like to thank WPI and the professors who have guided us through the entire project: Professor Arthur Gerstenfeld who oversaw the entire Wall Street project center, and Professor Jon Abraham and Professor Dan Dougherty who helped to give us direction over the course of the project. We would like to thank the staff at BNP Paribas who sponsored the project. We would like to especially thank David Jobet, our day-to-day liaison, who took the time out of his busy schedule to help us fully understand the details of the project and taught us all that we needed to know in order to successfully complete the project.

Executive Summary

Blizzard, BNP Paribas' smart order router implementation, is a complex system with multiple components. The monitoring software in place is essentially a filtered log of events, leaving it up to the user to interpret what is actually happening. Because of this, development and support for Blizzard is not as efficient as it could be, since the process of finding undesired behavior or performance and isolating the problem is tedious and difficult. In addition, there are a number of diverse algorithms in place that are difficult to visualize and understand for a person unacquainted with the smart order routers without some sort of visual aid.

The Blizzard Visualization Tool (BVT) is a solution to these problems. By providing an easy-to-use interface for displaying and analyzing orders, BVT allows the user to quickly isolate problem areas. In addition, BVT also calculates various metrics on all orders on a day-to-day basis, making anomalies and trends apparent. And because the actions taken by Blizzard, the parameters of the customer's order and the market conditions and responses are all consolidated into a simple timeline view, the user can quickly see the behavior and performance of Blizzard.

In addition to the BVT, another goal of this project is to find factors that influence the successfulness of trades, and then improve the router algorithm accordingly. Factors involved include the quality of quotes obtained from the market, the time taken to complete an order, the price changes of a stock, and latencies in the Blizzard system. In order to determine their impact, linear regression models and correlation tests between fill ratio and each factor are performed.

The result of data analysis is that the quality of quotes and latencies of Blizzard have the largest impact on the success of trades. The average performance of the router is measured as a by-product of the data analysis, and outstanding performance is seen: when a customer posts an order, there is a 90% probability that the order will be completed.

Background

1. Stock

Capital stock of a business represents the partial ownership in the business and entitles each shareholder to part of that corporation's earnings and assets. The stock of a business is divided into shares. The per-share prices can fluctuate up and down, depending on market supply and demand. Whenever shares are bought and sold, the prices move up and down.

The highest price that a buyer is willing to pay for immediate purchase is called "bid", while the lowest price a seller offers for immediate sale is called "ask". The difference between these prices is called the bid/ask spread. The size of the bid/ask spread is an important measure of the stock's market liquidity, or its ability to be sold without causing as significant movement in price. The bid/ask spread always exists because the moment there is an intersection, a transaction will immediately happen.

In a stock market, some crucial data which reflect the price changes of stocks involved are "Open", "Close", "High" and "Low". Opening price refers to the price of the first trade of a stock made on a given trading day, while closing price refers to the price of the last trade made. These two show the trend of price change of a stock in that particular day. The highest price reached in a day by a stock is called High, and the lowest price in record is called Low. The difference between High and low indicates the day's range of price change. Sometimes the High and Low are reported as "52 week High" or "52 week Low".

Another commonly used market data measure is day volume, which is the number of shares traded in a market during a day. If a high volume is detected during a price move, there is a large possibility that the movement is strong.

2. Stock Exchange

A stock exchange provides a means for stock brokers and traders to trade stocks. In the United States, the three largest stock exchanges are New York Stock Exchange (NYSE), NASDAQ Exchange and BATS Exchange. In addition to these, there exist several other exchanges in the United States and most other countries also have their own exchanges. Because of their nature of publishing price quotes for all of their securities, these markets are considered "lit" markets (as in "illuminated").

Besides the stock exchanges mentioned above, there are also dark liquidity pools. In a dark pool, prices of a stock are not published and orders are matched anonymously. Dark pools are widely used by institutional traders, who try to execute large stock transactions while minimizing market impact. Since all information of orders placed by other traders are invisible, there are few ways to know the algorithm of the trading system and is hard to determine whether the price of a transaction is relatively low or high. However, it is possible to suggest a price range from the normal exchange markets, because there should not be large differences of prices between the two kinds of market. If a wide gap exists, then brokers will gradually move to the market which provides a cheaper price and the price in the other one will drop, by the rule of supply and demand.

The volume available at a listed price is sometimes understated to minimize the market impact of large transactions. For example, if an ask price of $10 is offered with 100 shares listed in the market, there may be another 200 shares at $10 is hiding behind it. Only after the 100 shares have been traded, the hidden shares would be disclosed. These are sometimes referred to as iceberg orders, which specify both a display quantity and an overall order quantity at the time of queuing, and the hidden volume is called reserved quantity.

3. Regulation

In the United States, all stock exchanges must follow the regulations published by the U.S. Securities and Exchange Commission (SEC). Regulation National Market System (Reg NMS) is a series of regulations by the SEC to promote a national market system. National Best Bid and Offer (NBBO) refers to SEC regulations that requires brokers to execute customer trades at the best available ask price when buying securities and the best available bid price when selling securities. For example, the ask price is $10.01 per share on NYSE, while at the same time it is $10.00 on NASDAQ. When an investor places an order in NYSE, as per NBBO, NYSE has to transfer the order to NASDAQ, but then charges the buyer a routing fee.

However, NBBO only provides a protection on price but not volume. To look into details, take the example that if the ask price on NYSE is $10 for 200 shares while that on NASDAQ is $10 for 100 shares, when a buyer puts a bid of $10 for 200 shares on NASDAQ, NASDAQ will complete the transaction on $10 for 100 shares while return another 100 share uncompleted, rather than 100 shares on NASDAQ and transfer another 100 shares to NYSE.

4. Smart Order Router Strategies

Because of the complexities introduced by the dynamic and semi-opaque nature of the market, there are no trivial ways of handling exchanges without paying routing fees. This is where Smart Order Routers (SORs) come in. SORs algorithmically execute orders without human intervention. Using a set of customer given parameters and real-time market data, the SOR decides upon the price, timing, quantity and target exchanges for each order, often spreading the order out into multiple waves. For the example above, a typical SOR might send 100 shares on $10.00 to NASDAQ and 100 shares on $10.01 to NYSE. In this way, the best price that could found in the market may be taken. Since transactions move quickly, it is very possible that at the time of submitting request of 100 shares on $10 to NASDAQ, the price is already gone. Then the request will be denied and sent back to the router. After this, the router then finds the updated lowest price and tries to request the trade again.

Since not every order is the same, each having their own business requirements, there are a number of parameters that can be passed to the router to customize the algorithm. The most obvious of these parameters are the timeframe during which the router should work and the maximum price per share that the customer is willing to pay. The more complex parameter is the aggressiveness of the strategy to use to fill the order. More aggressive strategies generally target prices closer to the ask side of the spread, usually resulting in buying at a higher price and causing more market impact, but also completely filling the order more quickly. Less aggressive strategies can usually achieve lower prices by finding the hidden liquidity in the spread, but they are slower by nature because of the trial-and-error nature of these strategies. But because of the dynamic nature of the market, changes in prices could happen during the course of the order, causing it to be more expensive than aggressive strategies. More passive strategies have little to no market impact due to their tendency to target hidden liquidity and dark pools.

The SOR implementation at BNP Paribas is codenamed Blizzard. There are three primary strategies currently used in production in Blizzard, listed more aggressive to less aggressive: Omega/Omega+, PriceMarch and Dark/Dark+.

Omega is a simplistic algorithm that targets all lit markets at NBBO, doing an even split of the order amongst all of the targeted markets to cope with reserves. After all queries are sent out, it has to wait for the execution feedback and send another wave of orders if the order is not yet completely filled. Omega+ (also known as FOES or DMA) behaves the same as Omega, except when the order is flagged with “market hour only” and we’re in the pre-market session or

the order is flagged with “after hour only” and we are in the pre-market or market session. In that case, we send the order as is to the primary market with a queuing flag.

PriceMarch targets both lit markets and dark pools, finding and exhausting the hidden liquidity in the spread. The algorithm does an exhaustive search of the spread, “marching” from the bid price to the ask price, splitting the order amongst all dark pools and lit markets. By analyzing the result of each wave, adjusts its price point accordingly and sends out another wave. Because PriceMarch is very slow by nature, it is generally only used on stocks that have low liquidity and volatility.

Dark targets dark pools only and behaves very much like PriceMarch, except it never targets displayed liquidity. Because of this, it is completely stealthy, having no market impact, except for some information leakage through trade reports. As soon as it reaches the end of the spread, it will post aggressive orders in the dark pools. Dark+ is the same as Dark, except it also targests lit markets.

The smart order router has three essential priorities: successful execution, low fees and price improvement. Because the smart order router makes money purely through order commission, the highest priority is execution. In addition to the commission associated with orders, execution is the greatest factor in customer retention, since failing to fully execute customers’ orders decreases customer satisfaction and the firm’s reliability. The minimization of fees is important because fees are a primary motivation for the existence of SORs in general, instead of just relying on a single market to ensure NBBO. The third priority of price improvement, or the difference between the average price per share paid and the NBBO price per share at time of order, is not vital, but can improve customer satisfaction and is a metric that can lure new customers.

The process of determining whether or not a particular algorithm is effective can be difficult because of the dynamic nature of the market. Some metrics that can be used are execution speed, price improvement and fill ratio (the ratio of orders sent out to each of the different markets and the actual fills returned). It is possible to compare the metrics of algorithms, but because of uncontrollable external factors, the returned metrics for the same algorithm can, and does, change every day. This is especially a problem when trying to determine whether or not modifications to an algorithm are actually improvements. Consider the price improvement as an example: it is calculated by dividing the original ask price by the difference between ask price and actual pay. Therefore, price improvement has a large reliance on the ask price which is

constantly changing. Since different stocks have different volatilities and fluctuation on ask price, it is hard to compare the price improvement between stocks even without changes on strategies.

5. Router Specifications

The router typically makes thousands of trades per day. It works in the way that once an order was received, the router determined which exchange markets satisfy the NBBO condition – looking for the best price - after gathering prices from markets. Then it split the total quantity need to be filled into several parts in order to send several submissions to different markets. Because of the existence of latency, when some submissions reached their destination, the quote listed before may be already taken by other people and the price was not available anymore. Then these submissions would not have any quantities filled and thus failed. At the same time, other submissions maybe filled or partially filled, depends on the quantity available in that specific market they were sent to. After the result of all submissions had come out, the remaining quantity could be calculated and the router would send another round of submissions again. This loop stops until the order is completed, modified or canceled by the customer or at the end of the day. The time taken to complete an order can range from milliseconds to days, depending on which strategy is used and the changes of price in markets.

Orders can be divided into two main categories by their trading type: market and limit. Market orders require completing trades under the best prices listed on markets. Filling the market order will have impacts on the NBBO price and quantity listed, so most of them are aggressive. On the other hand, limit orders typically have specific price limits required by customers. Some of them are aggressive if the limit lies in the NBBO price range, while others are passive unless price changes on markets or orders are modified.

Note in this project we only interested in aggressive trades and the criteria we set to distinguish between aggressive and passive is defined by:

- All market orders processed by the router are aggressive.
- For a limit order, if it is on buy side and the ask price on market is less than the order price, or it is on sell side and the bid price on market is larger than the order price, then the order is aggressive.

6. Metrics used

6.1. Router fill ratio

Router fill ratio is a measurement of successfulness of trades from the router's view. It is calculated by dividing filled quantity by total quantity submitted, and thus the result is between zero and one. Zero means that after an order is submitted, no quantity is filled, while one indicates that all of the quantity submitted is filled. The larger the router fill ratio is, the more successful the trade is.

6.2. Hit ratio

Hit ratio measures the fraction of successful submissions in an order regardless of the quantity executed each time. The number of submissions which have some quantity executed (does not need to be the full submitted quantity) is counted, and then it is divided by the total number of submissions. A high hit ratio indicates more successful submissions in an order.

6.3. Customer fill ratio

Customer fill ratio is a measurement of successfulness of trades from the customers' view. It is calculated by dividing filled quantity by order quantity, where the order quantity refers to the quantity required to fill by customers. Just as with the router fill ratio, the larger it is, the more successful the trade is (as seen by customers).

6.4. Stalled ratio

Stalled ratio is a measurement of goodness of quotes. First, if several successive submissions executed zero quantity at the same price from the same destination over a period of time, then there may be some error in the quote itself. Then the number of such submissions is counted and is divided by total number of submissions. Therefore, the larger the stalled ratio is, the more inaccurate quotes were taken and thus the quality of total quotes is lower.

6.5. S&P 500 ratio

S&P 500 ratio shows the percentage of orders which trade the stocks in S&P 500 in the total number of orders in a day. Basically, stocks in S&P 500 are more volatile compared to others, but trading for them is more competitive since more people want to get the same quote at the time.

Blizzard Visualization Tool

1. Requirements

1.1. Functional Requirements

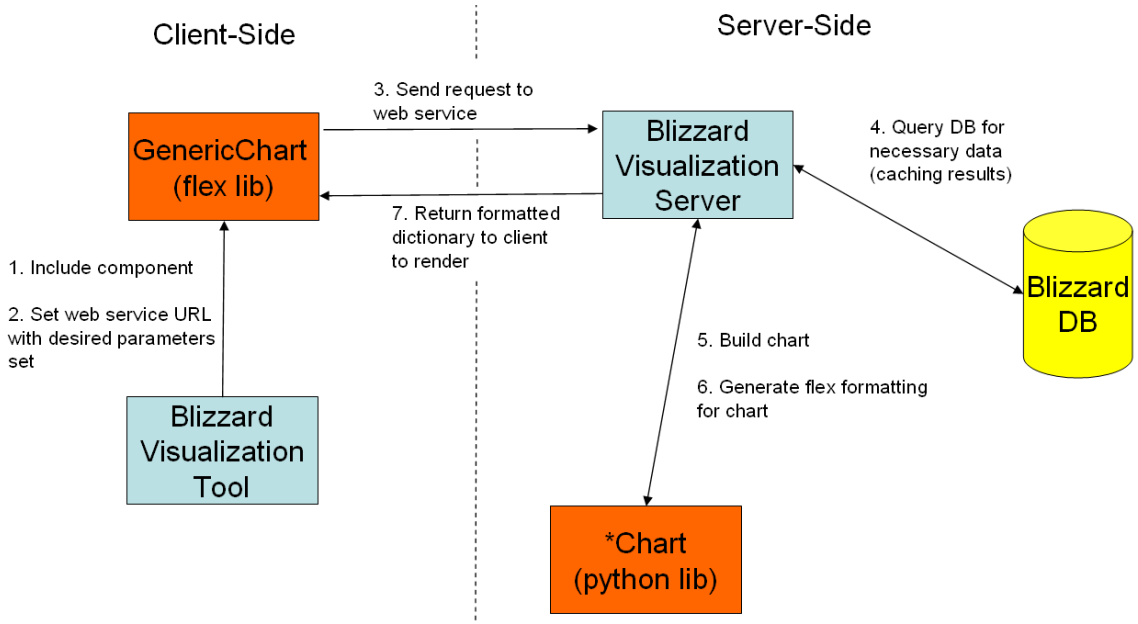
- Develop a tool that can be used to visually monitor and analyze the behavior and performance of Blizzard
- Develop a tool that can be used to compute and display various metrics on orders to find anomalies and problem areas

1.2. Non-Functional Requirements

- Minimize perceived response time
- Minimize performance impact to database
- Develop the tool to be extensible and easy to maintain
- Make components of the tool reusable for other projects in the future

2. Architecture / Flow

The tool that we are developing is a stand-alone application, connected to Blizzard by sharing the same database. Because one of the requirements is to minimize the performance impact to the database, the tool only requires read-only access. This architecture is broken down into four components, two making up the charting framework and two making up the tool itself. Of the two charting framework components, one is the server-side component that is written in Python and handles the building and formatting of charts, and the other is the client-side component, written in Flex/ActionScript, that handles the rendering of the chart as described by the server-side component and dealing with user interactions with the chart. Of the two components that make up the tool, one is the server-side component that is written in Python and accesses the Blizzard database to build and performs the computations necessary to build the charts and metrics tables; the other is the client-side component that is written in Flex/ActionScript, and serves as the user interface for the entire application.



The server-side components are run in a separate server, with the Blizzard Visualization Server component acting as an HTTP server, listening to requests from any number of clients. The client-side components are distributed to any number of users' machines, with a configuration file that designates the URL of the server to connect to. The reason for this is so that, for most updates, an update package doesn't have to be distributed to all clients running the application.

3. Technologies used

3.1. Flex (front end / client side)

For the front-end component of our application, we used Adobe Flex 3.2, a software development kit (SDK) used for development of cross-platform rich internet applications (RIA) on top of the Adobe Flash platform. Development was done using the proprietary Flex Builder 3 application, also developed by Adobe Systems, and deployment was done on Microsoft Internet Explorer 6, as it is the standard web browser of BNP Paribas (although because of the cross-platform nature of the Flex SDK, it should work the same in all browsers that support Flash).

Development using the Flex SDK composed of the creation of two file types: ActionScript (.as) files and MXML (.mxml) files. ActionScript is a scripting language originally developed by Macromedia (now owned by Adobe) that is primarily used for development on the Flash platform. Being a dialect of ECMAScript, it is very similar to the widely used JavaScript. MXML is an XML-based markup language also developed by Macromedia. Ultimately, MXML

is compiled down to ActionScript, so the two have the same capabilities, except MXML significantly simplifies the development of the user interface component.

3.2. Python (application layer / server side)

For the back-end component of our application, we used Python on top of the Twisted framework. Python is a dynamic, interpreted programming language used for a variety of purposes, but is popularly used for scripting and rapid prototyping. The version of Python that we used was 2.4.3. Twisted is an MIT License event-driven network programming framework.

3.3. Oracle (database)

The application connects to the production database used by Blizzard to provide accurate, real-time statistics. Blizzard is built on top of Oracle, a relational database management system produced by the Oracle Corporation. Like most RDBMS, the querying language for Oracle is SQL.

4. Development of charting framework

The secondary goal of this project was to produce a general-purpose charting framework for BNP Paribas. The framework consists of two libraries, a client-side Flex library and a server-side Python library. In order to use the library, a developer must develop two components, a server and a client. The user's server component does any sort of computations and business logic they want and use the Python charting library to format it into a way that the Flex library can understand. They then set up a web server that responds with the formatting that the Python library gives. The client-side component that the user creates must include the Flex chart library, create a chart object and point it to the URL where their server lives.

The model that we used for creating the framework allows for the developer to focus entirely on the business logic on the server-side. When the logic for charts change, the client-side program that gets deployed to different computers need not to be updated to reflect these changes. It also doesn't enforce a strict technology stack. Any sort of HTTP server can be used, and either the client-side or server-side components could be completely rewritten in any other language or platform and no changes have to be made to the other component so long as the message format remains the same.

4.1. Server-side Component

The server-side component of the charting library consists of a series of classes that all inherit from a base Chart class, each representing a different type of basic chart (BarChart, PlotChart, AreaChart and LineChart).

A chart consists of three pieces: metadata, axes information and data. The metadata of a chart includes the title of the chart (if there is one), the type of chart it is and any other type of information describing the chart and how it is to be rendered. The chart stores information about a single x axis and a single y axis. Each of these axes can be one of three types, linear (a continuous, numeric sequence with a minimum and maximum value), temporal (a continuous, temporal sequence with a start and end time/date) or categorical (a finite set of categories). The data component of the chart holds information about the series and elements in the chart. All of these pieces of information are abstracted away from the user, allowing them to build a chart piece-by-piece.

There is also a special type of chart, the ComboChart that acts as a collection of any number of the standard charts. If the axes of multiple charts inside the same ComboChart have the same name, they share that axis. Otherwise, they will have separate axes.

Once the chart has been completed, the user can call the get_flex_formatting() function of the chart, which returns a single dictionary that describes the chart in a format that the client-side component of the library can understand. The user of the library is responsible for passing this data back to the client-side component.

4.2. Client-side Component

The client-side component of charting library (implemented in Flex) handles most of the heavy lifting for the user. The user only has to create a chart object in Flex and then point it to the URL of the server that is serving the chart, passing it any necessary parameters as basic HTTP GET parameters.

After being given the URL of the server that is serving the chart, the makes an HTTP query to the server to get all of the necessary information it needs to build the chart using the Flex charting library. After building the chart itself, the library also builds the legend to go alongside the chart and allows the user to click on different components within the legend to disable and re-enable the components in the chart.

Besides greatly simplifying the creation of complex charts using the Flex charting framework, the chart library also adds a few new features. The most significant of these are the

zooming features and features that were added to improve the user interface associated with zooming.

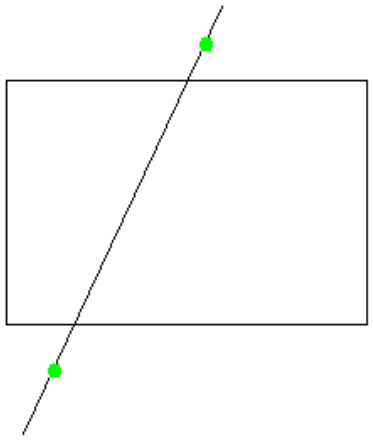
The most basic method for zooming is by using the mouse scroll wheel. Because of the requirement for charting library to be able to handle charts with multiple axes that can zoom and scroll independently of each other, zooming of the horizontal and vertical axes were handled independently. Normally, zooming is done on the x axis and holding the Alt key while scrolling the mouse wheel allows for controlling the y axis. The camera zooms into wherever the mouse is currently pointing. If the dimension that is currently being zoomed has multiple axes, the series (or axis) that the mouse is currently pointing at will be the one to zoom. If the mouse is pointing at blank space, the primary series (the first one to be added) is the one that is zoomed. The Ctrl and Shift keys can also be held to decrease and increase, respectively, the speed at which zooming is done.

The chart can be in one of two modes, scrolling mode or selection zooming mode. When in scrolling mode, clicking and dragging the chart allows the user to move the camera around, focusing on different parts of the chart (especially if they're zoomed in). When in selection zooming mode, clicking and dragging the chart allows the user to create a box around the section of the chart that they're most interested in. When they release the mouse button, the camera will zoom to that section. If the chart has multiple axes, all of them will zoom appropriately.

Because the width of the bars in bar charts is equal to the distance between the values on the horizontal axis, an interesting problem is presented when trying to display a bar chart with a linear or temporal horizontal axis. Because of the sheer number of possible values, bar widths become infinitesimal. To deal with this, the values of continuous axes are seamlessly clustered into chunks. The size of these chunks is based on the difference in the value between the minimum and maximum values of the axis, and the size of the chart in pixels. The size of the chunks is calculated so that bars are always of a minimum width and is recalculated every time the chart is resized zoomed.

The other challenge that came up was the rendering of lines. By default, if one of the end points in a line segment is outside of the viewing window, Flex simply doesn't render the entire line segment. The rendering had to be changed so that it did not automatically drop points that were out of view unless the line connecting them also doesn't go through the view. But because of this, when zoomed very far in, there existed line segments that were several times the width of the screen. Unfortunately Flex does not correctly render these lines, causing the entire chart to be unusable when this happens. To work around this, the charting library automatically predicts when lines in view are getting too long and inserts temporary points, breaking up that line

segment. The temporary points are placed out of the viewing window as to hide the fact that they exist from the user. In the example line and view window below, the green dots represent where the temporary points would be placed.



These points are automatically removed when they're no longer needed to maintain the integrity of the lines in view.

4.3. Sample Code

A chart can be inserted into any existing Flex project in only a few lines:

```
<mx:Script>
        private function onLoad_(e:Event) : void {
        barChartView.src = makeUrl("test_bar_chart", {});
        }
        </mx:Script>



        <mx:Canvas label="Bar Chart" width="600" height="600>
                <chart:GenericChart id="barChartView“
                        width="100%" height="100%“ />
        </mx:Canvas>
```

The server side code that builds the chart is also only a few lines:

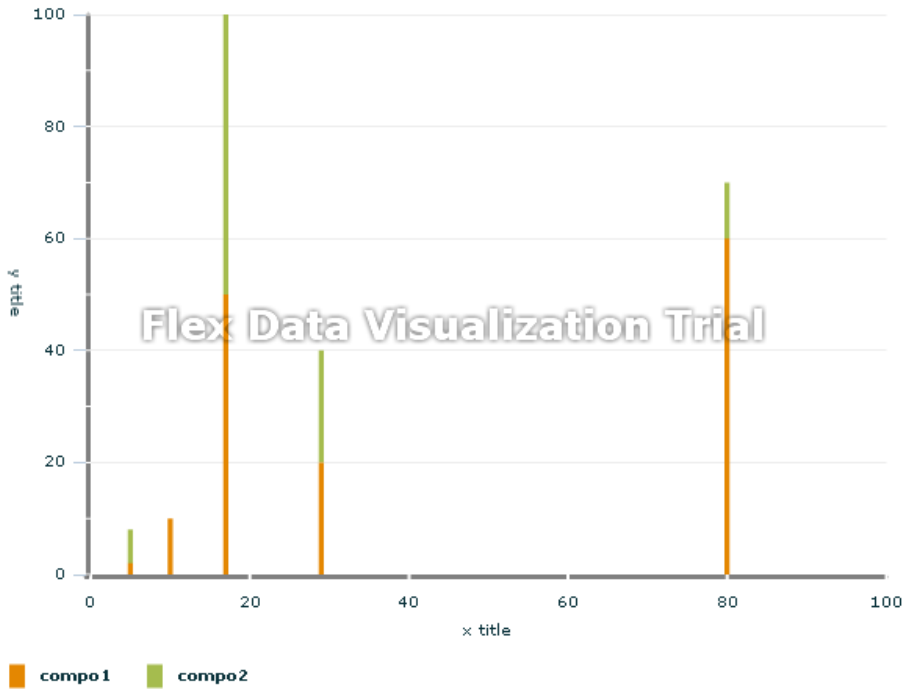
```
chart = BarChart()

chart.set_linear_x_axis('x title', 'bottom',  0, 100)
chart.set_linear_y_axis('y title', 'left', 0, 100)

chart.set_stack_type('stacked')
chart.add_bar_composition('compo1')
chart.add_bar_composition('compo2')

chart.add_bar(5, [2, 6], ‘bar one')
chart.add_bar(10, [10], 'bar two')
chart.add_bar(17, [50, 50], 'bar three')
chart.add_bar(29, [20, 20], 'bar four')
chart.add_bar(80, [60, 10], 'bar five')
```

The resulting chart would look like this:



5. Development of visualization tool

The visualization tool was designed to be used by support to easily analyze how orders are handled by Blizzard and quickly spot anomalies. The interface is broken into two sections, a section for the metrics table and a section for the charts.

5.1. Metrics Table Section

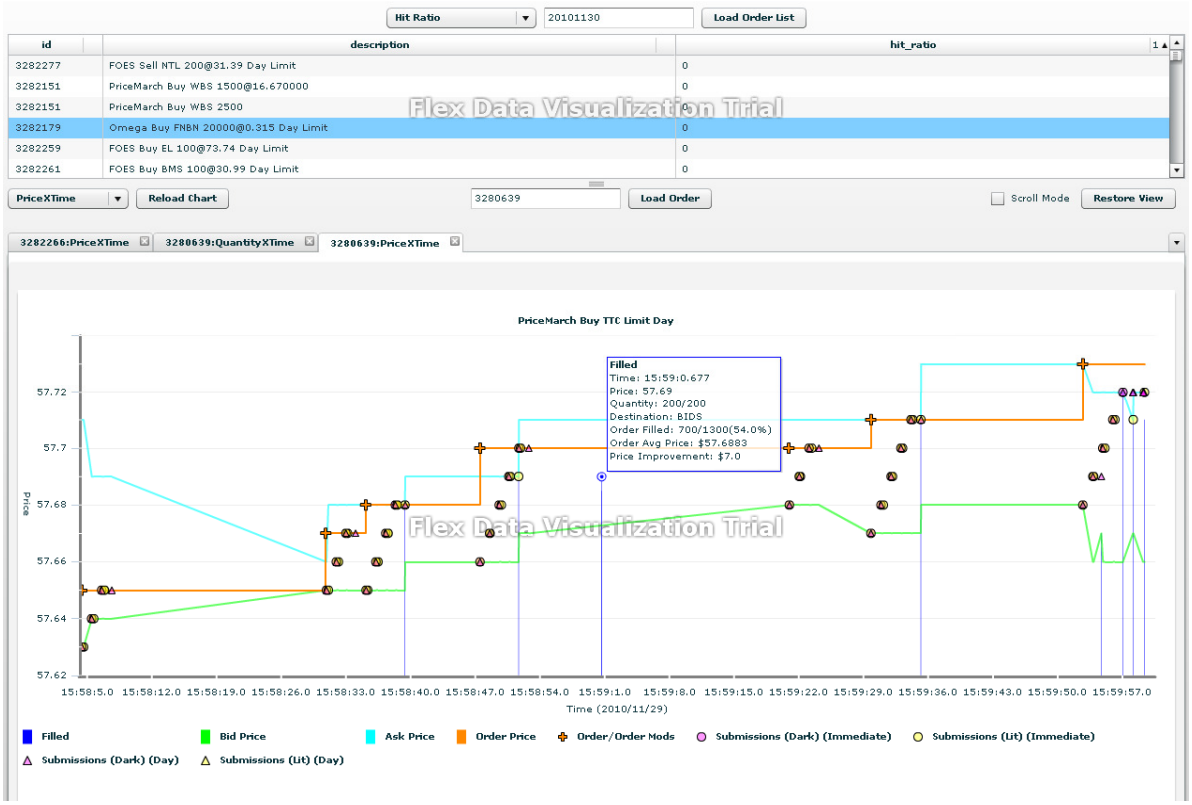
The metrics table allows the user to select any given day and computes the desired metric over all orders that were processed that day. The table then returns the top N orders that have the "worst" scores for that metric. This allows the user to quickly isolate orders that have issues and possibly find a bug in the router, or a place that needs improvement. Because the computations required to get these values are very intensive, these results are cached on the server and should return almost instantly the next time anyone else requests these metrics for orders in that day.

5.2. Charts Section

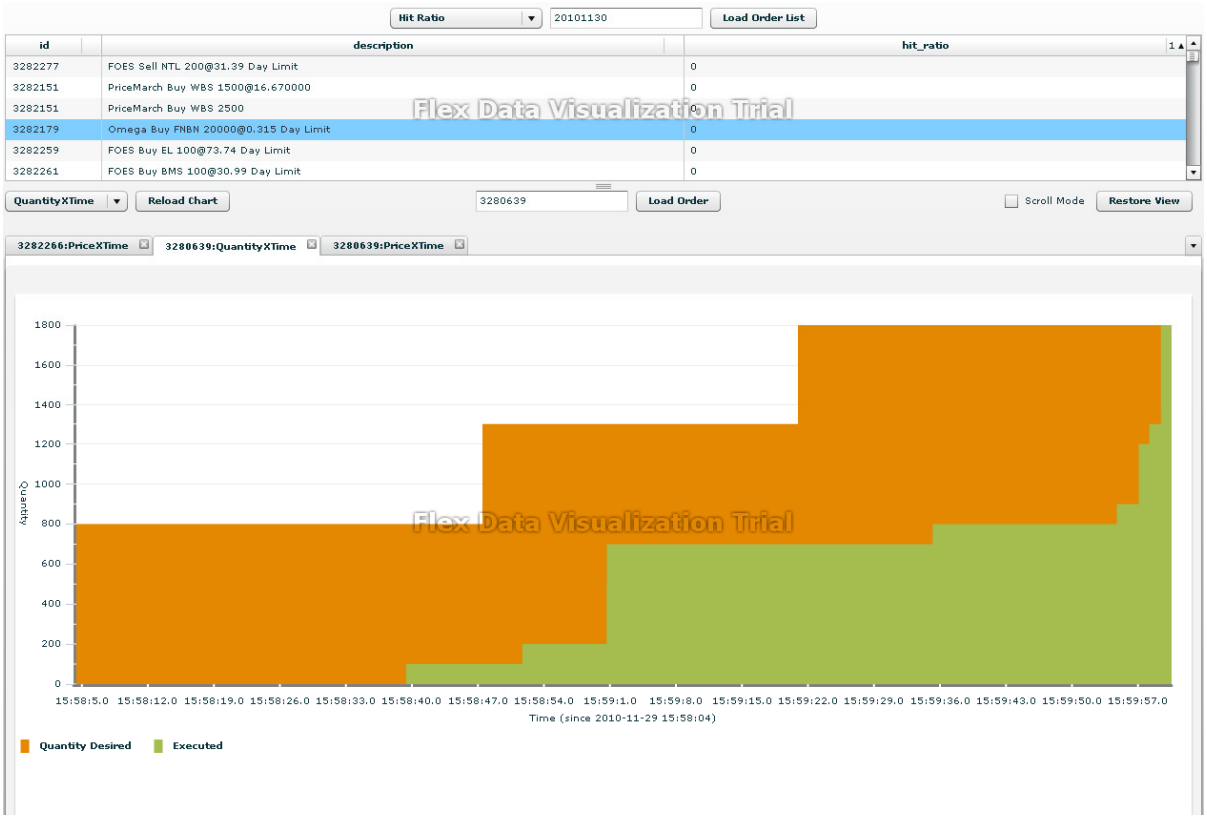
The charts section of the tool allows for multiple charts to be opened at once and stored locally in tabs so that the user can quickly jump back and forth between several orders (or several different types of chart representing those orders). Currently implemented are two types of charts

that show a timeline for an individual order, a Price-By-Time chart and a Quantity-By-Time chart, and one type of chart that shows different types of latencies for all orders in an entire day:

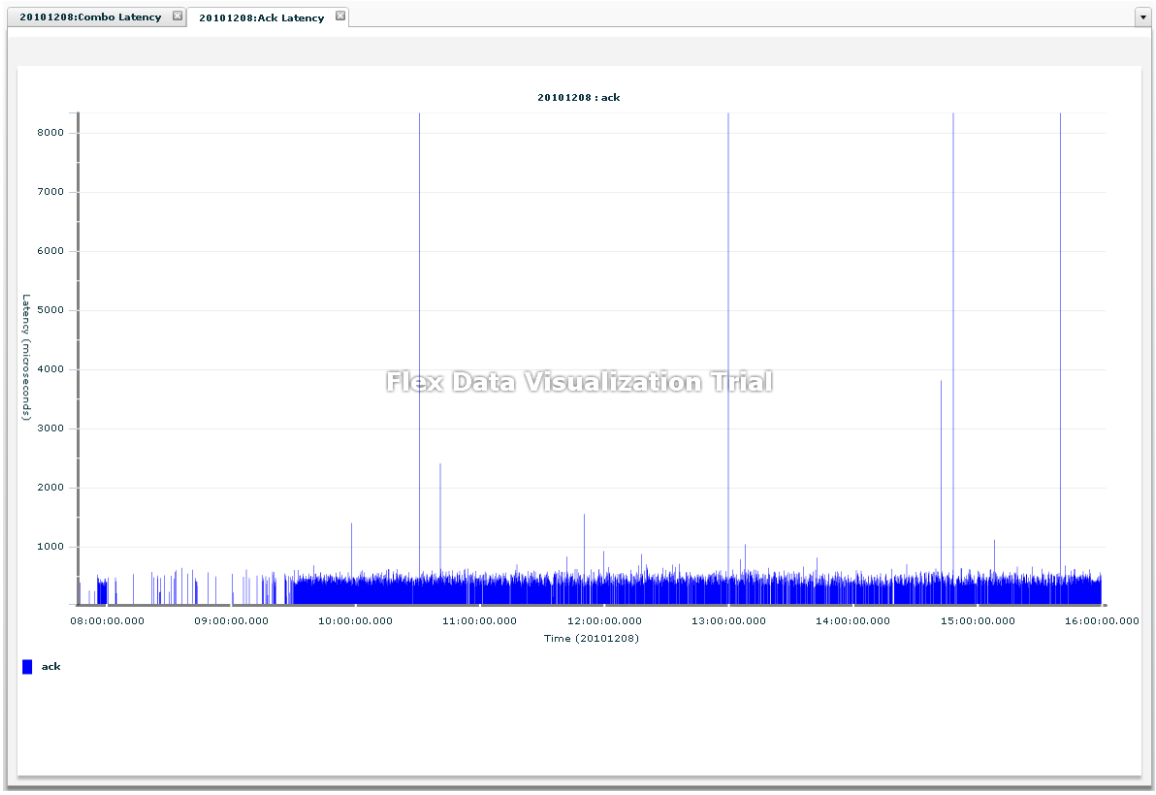
The Price-By-Time chart is a price-oriented timeline of life of an order. The orange line represents the price at which the customer desired, with orange crosses representing modifications to their order. The blue and green lines represent the bid and ask prices of the market. The lines are artificially stable because their values are only recorded whenever submissions go out. The points represent submissions to the market; circles represent immediate submissions and triangles represent day submissions; pink points represent submissions to dark pools and yellow points represent submissions to the lit markets. The blue bars represent fills. Hovering over each of the elements will display additional information about that element.



The Quantity-By-Time chart is a quantity-oriented timeline of the life of the order. The orange area represents the quantity of shares the customer desires to buy or sell and the green area represents the quantity of shares already filled by the router.



The latency chart is a timeline of the latency of all orders over the course of an entire day. The chart can display a variety of different types of latencies (explained in detail in the next section) over the course of the day.



6. Fulfillment of requirements

6.1. Fulfillment of Functional Requirements

The Blizzard Visualization Tool, even at this stage, allows the business to visually analyze the behavior and performance of Blizzard in a way that they haven't been able to before. The order-oriented charts allow the user to quickly drill down to the details of an order and see exactly things going on. This proves to be invaluable to those who develop and maintain Blizzard. The latency charts allow the user to get a high-level overview of the entire router over the course of the day. This view can show trends and anomalies, allowing the user to quickly see that something went wrong and track down the source.

The chart also serves as a visual aid to understanding the behavior of Blizzard. For a person that is not familiar with the algorithms used for routing orders, the tool can be used to easily see how the router behaves in conjunction to the parameters that the customer specified and the market conditions at the time.

6.2. Fulfillment of Non-Functional Requirements

Because the tool is meant as a support tool, we made sure to minimize the impact to the production database. This was accomplished by locally caching any and all complex or large queries to the database and by moving as much computations away from the database as we could. Besides this, its nature as a way of allowing the user to quickly browse through orders, the perceived responsiveness of the tool is very important. To accomplish this, a lot of the information that the chart requires is asynchronously pre-fetched from the server, minimizing the wait time between actions.

But the most important non-functional requirement that was accounted for during the design and development of this tool is its extensibility and the reusability of the components of the tool. Knowing that the feature set of the tool as it stands today are only the tip of the iceberg, the architecture is designed to be generic and componentized. Because of this, new features and behavioral changes can be easily implemented with limited integration testing and fears of side effects. Also, the different components of this tool provide generic enough interfaces and behaviors that they can be reused for other projects.

Data analysis

1. Introduction

To determine the effect of different factors, such as the stalled ratio and latencies, on the successfulness of trades which is measured by the fill ratio and the hit ratio, we analyze data sets using the following method.

1.1. Factors influencing success of trading

Factors we are considering contain a group of latencies caused by the router itself and the internet transmission. Basically, six main measurable latencies are recorded: ack latency, internal latency, order new internal latency, force latency, force ack latency and market ack latency.

Ack latency, where ack stands for the acknowledgement, is the time taken for the router to acknowledge an order and the average is around 0.4 milliseconds.

Internal latency is the time taken for the router to process the order. After an order is received, the Blizzard needs to decide where submissions should be sent to and the quantity of each submission to send. This latency usually ranges from 0.5 to several milliseconds.

Order new internal latency is similar to the internal latency, but it only modified orders are concerns. In theory, it should be a little less than the internal latency because some data of the order has been already stored.

After submissions are sent, they reached a server before posted on the market. In theory, force latency measures the time taken for the server to process submissions, but in reality, it is measured by the router side, which takes account of the time between submissions sent and proceeded confirmation received, so the internet transmission latency is also involved.

Force ack latency in theory should be the time taken for the server to acknowledge the order. Again, because only the time of submissions sent and that of acknowledgement received are recorded, there involves the internet transmission problem.

Market ack latency overviews the time spent for submissions to arrive the market. It is measured by the time between the submission sent and the acknowledgement of market received by the router. Therefore, the force latency and the force ack latency is included.

Besides latencies, we also examine other factors that might have influence on the fill ratio. Duration is the time taken for an order to be completed. When the fill ratio is high, an order should be filled quickly and thus has a short duration. Number of submissions live in the router may have an impact on latencies and thus another factor affecting the result. A higher stalled ratio means more submissions are not filled and will reduce the total fill ratio as well. The percentage

of the order quantity in total traded volume of the respective stock on that day may also have an effect. Price range is calculated by dividing the difference between the highest and the lowest price of a day by the average of the open and the close price. If the price changes a lot in a day, there is a large possibility for us to complete the order.

1.2. Linear correlation

Correlation measures the relationship between two or more random variables or measured data values. The correlation between the fill ratio/hit ratio and latencies are examined in order to find the evidence of influence.

A typical correlation coefficient is the Pearson correlation coefficient which measures the linear correlation between two variables and is calculated by (covariance of variable x and y) / (product of standard deviation of x and y). The coefficient ranges from -1 to 1, where -1 indicates a strong negative correlation and 1 for a strong positive correlation. No correlation between the two variables will return a value of zero. A scatter plot of x and y may help to explain it (above the scatter plot is the corresponding correlation coefficient):

The correlation test gives a 0.9 coefficient between the fill ratio and the hit ratio, which shows these two are closely related and have a strong positive relationship. Since they are both measurements of successfulness of trades, the result is reasonable.

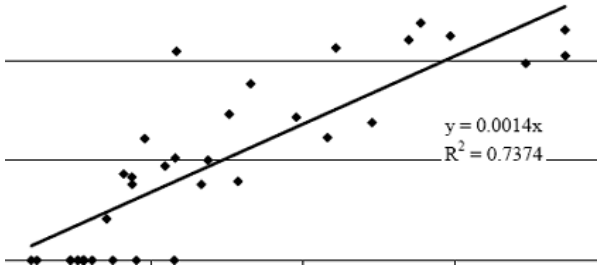
Along with the correlation test, confidence intervals are also presented. A confidence interval gives an estimated range of values being calculated from a given set of sample data and is used to indicate the reliability of an estimate. A 95% confidence interval of (0.85, 0.95) for a correlation coefficient 0.9 means that it is 95% sure the correlation coefficient lays between 0.85 and 0.95 with an estimated value of 0.9.

1.3. Least squares regression

Linear least squares is a method for estimating the unknown parameters in a linear regression model. 'Least squares' means that the overall solution minimizes the sum of the squares of the errors made in solving every single equation. It helps to fit data into a linear regression model which assumes an approximately linear relationship between the dependent variable y and the independent variable x, and is given by the equation $y = \alpha x + \varepsilon$ where α is a constant and ε represents the error term. In addition, to see how well a regression line

approximates real data points, R squared is examined. An R squared of 0.8 indicates that 80 percent of the change in y is explained by the change in x.

The following graph shows a typical linear regression line with an equation and an R square estimated by a set of dots.



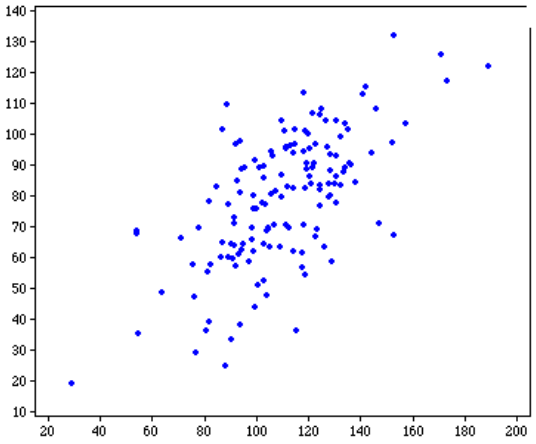
1.4. Multivariable regression

Given a set of independent variables x_i, the regression model mentioned above can be changed to $y = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n + \varepsilon$, which takes account of each factors and is called multivariable regression. Take an example, in this project, one of the models is constructed as

fill_ratio ~ ack_latency +duration +nb_submissions_live +stalled_ratio +price_range +quantity_volume +internal_latency +order_new_internal_latency +force_latency +force_ack_latency +market_ack_latency

where the fill ratio is the dependent variable y and the following factors are independent variables x.

After the model is built, predictions can be made and scatter plots of original data and predictions on the same graph may give a vivid view of the goodness of predictions. The scatter plot may also help on examine the linear relationship between the y and any x. (a typical scatter plot shown below)



2. Results

2.1. Per day

To analyze data and find the relationship between the fill ratio and all factors, we first generate data grouped daily – trades in one day are summarized and then averaged to calculate metrics.

2.1.1. Customer fill ratio on average

The customer fill ratio is used as a measurement of successfulness of aggressive trades here. The following result shows the weighted average and the standard deviation of the customer fill ratio per day from September 1 to October 30. The customer fill ratio is calculated by dividing the sum of executed quantity of aggressive orders in a day by the sum of total quantity respective to those orders. The second column displays the overall customer fill ratio while the following columns are ratios broken out by different strategies.[1]

	All strategies	PriceMarch	Omega	Dark	DarkPlus	Foes
% of orders	100%	10.1%	9.4%	0.3%	0.2%	79.9%
Avg	0.83	0.66	0.96	0.62	0.35	0.89
STD	0.057	0.124	0.078	0.357	0.354	0.052

As can be seen from the percentage of orders row, Dark and Dark Plus strategies are seldom used and the most used one is Foes strategy.

From the table, different strategies performed differently because of their own characteristics. On average, Omega has done the best job, whereas Dark Plus, mostly not under our control and with limited market price information, did the worst.

The standard deviation varies. Dark and Dark Plus strategies have larger standard deviation because they are less used compared to other strategies and thus lack of data. Prick March changes a lot over the time while others are more stable than it.

Then data are further grouped by order side, order type and combinations of them:
Limit Orders:

	Customer_fill_ratio	PriceMarch	Omega	Dark	DarkPlus	Foes
Avg	0.82	0.66	0.95	0.62	0.33	0.88
STD	0.060	0.124	0.106	0.358	0.356	0.056

[1] The full record is attached in the appendix as Table 1.

Market Orders:

	Customer_fill_ratio	PriceMarch	Omega	Dark	DarkPlus	Foes
Avg	1.00	0.99	1.00	N/A	0.97	1.00
STD	0.004	0.04	0	N/A	0.04	0.001

The customer fill ratios of market orders are high at nearly 1, whereas those of limit orders are lower. This is because market orders adjust their prices according to the changes of market prices, so most of them can be filled. The calculated ratios of limit orders are about the same as that of the overall group, which indicates most aggressive orders taken in count are limit orders.

Buy Side:

	Customer_fill_ratio	PriceMarch	Omega	Dark	DarkPlus	Foes
Avg	0.85	0.70	0.95	0.67	0.25	0.90
STD	0.076	0.181	0.103	0.391	0.336	0.061

Sell Side:

	Customer_fill_ratio	PriceMarch	Omega	Dark	DarkPlus	Foes
Avg	0.82	0.64	0.98	0.57	0.49	0.89
STD	0.081	0.162	0.035	0.383	0.422	0.072

Limit Buy:

	Customer_fill_ratio	PriceMarch	Omega	Dark	DarkPlus	Foes
Avg	0.83	0.70	0.94	0.67	0.21	0.88
STD	0.082	0.183	0.126	0.391	0.305	0.066

Limit Sell:

	Customer_fill_ratio	PriceMarch	Omega	Dark	DarkPlus	Foes
Avg	0.81	0.63	0.98	0.57	0.49	0.87
STD	0.087	0.154	0.039	0.383	0.423	0.080

In general, sell side has a little bit lower ratios than the buy side, but it also depends on the strategy used. If the standard deviation is considered, then no difference between the two averages can be seen. Omega and Dark Plus performed better on the sell side while others are better on the buy side. The result is consistent after they are further decomposed by order type.

2.1.2. Correlation test

To find out factors that may impact the customer fill ratio, correlations between the overall customer fill ratio and ack latency, total number of orders in a day (denoted as nb_orders), stalled ratio and S&P 500 ratio are tested and results are showed below. [2] [3]

Some correlation between the customer fill ratio and the ack latency is found:

- -0.40 with 95% confidence interval: -0.6384 -0.0819

Though -0.4 is not a very significant number in the correlation test, it does show some impact brought by the ack latency, because in reality, the customer fill ratio must be affected by many factors and it is impossible to have a near-1 correlation coefficient on a single possible factor. However, the confidence interval is large and thus the -0.4 correlation is somewhat unreliable.

Nearly no correlation between the customer fill ratio and:

- nb_orders: -0.13 with 95% confidence interval: -0.437 0.201
- stalled_ratio: 0.08 with 95% confidence interval: -0.253 0.392
- SP500_ratio: 0.02 with 95% confidence interval: -0.309 0.339

Since these confidence intervals contains zero, it is most possible that there are no correlations according to this set of data.

2.1.3. Linear regression

The linear regression model for this set of data is built as:
customer fill ratio ~ ack_latency + nb_orders + stalled_ratio + SP500_ratio

The estimated coefficients are showed below:

	Estimate	Std.Error	t value	Pr(>|t|)
(Intercept)	1.085e+00	1.075e-01	10.096	1.80e-11
ack_latency	-6.228e-04	2.386e-04	-2.610	0.0136
nb_orders	-1.888e-06	3.338e-06	-0.566	0.5756
stalled_ratio	3.545e-01	2.565e-01	1.382	0.1766
SP500_ratio	-3.027e-03	8.645e-02	-0.035	0.9723

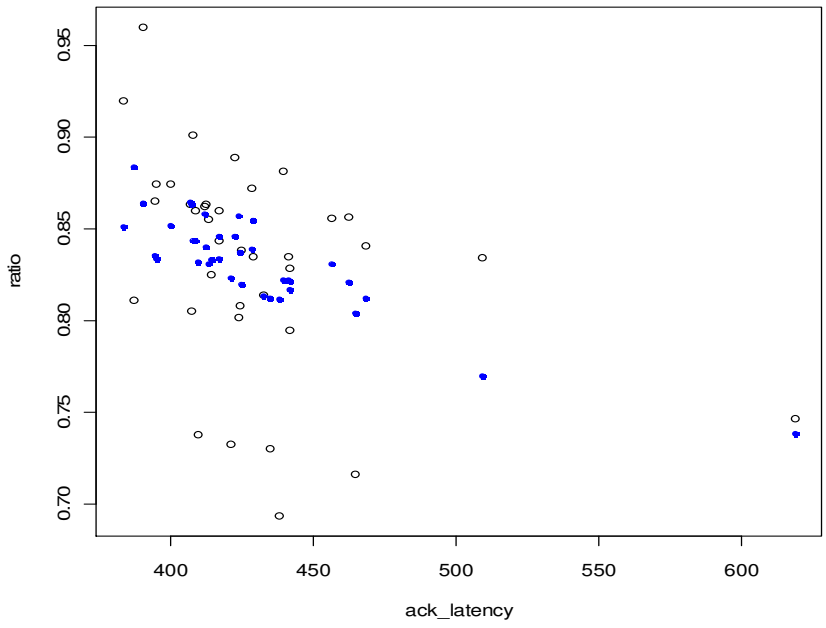
with a multiple R-squared 0.21, which is not a good estimation.

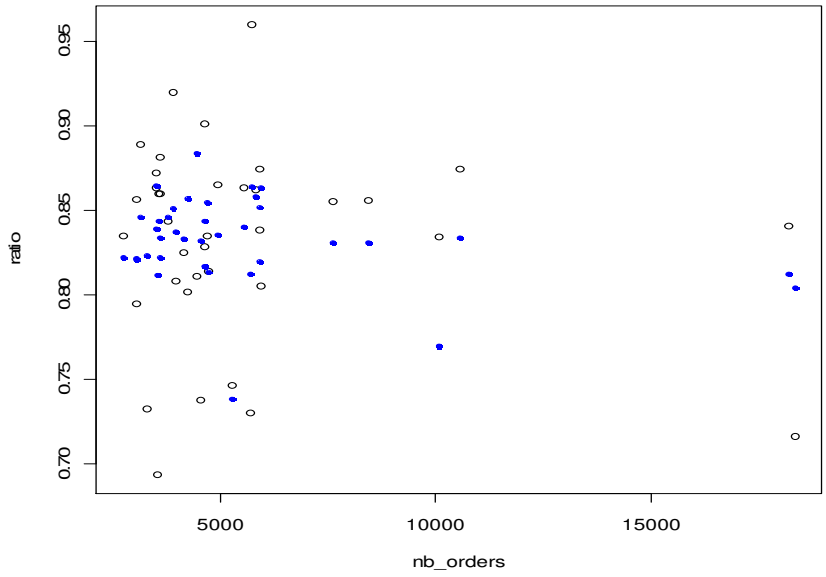
[2] Data of ack latency, nb_orders, stalled ratio and S&P 500 ratio are attached in the appendix in Table2.
[3] Results for orders grouped by side and type are similar and thus not presented here.

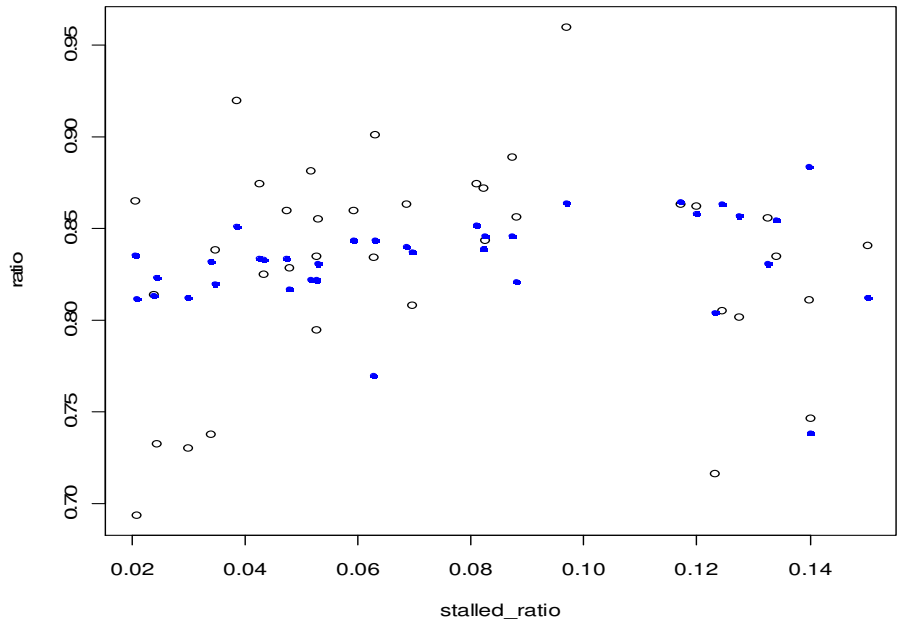
A scatter plot may give a clearer view of the results:

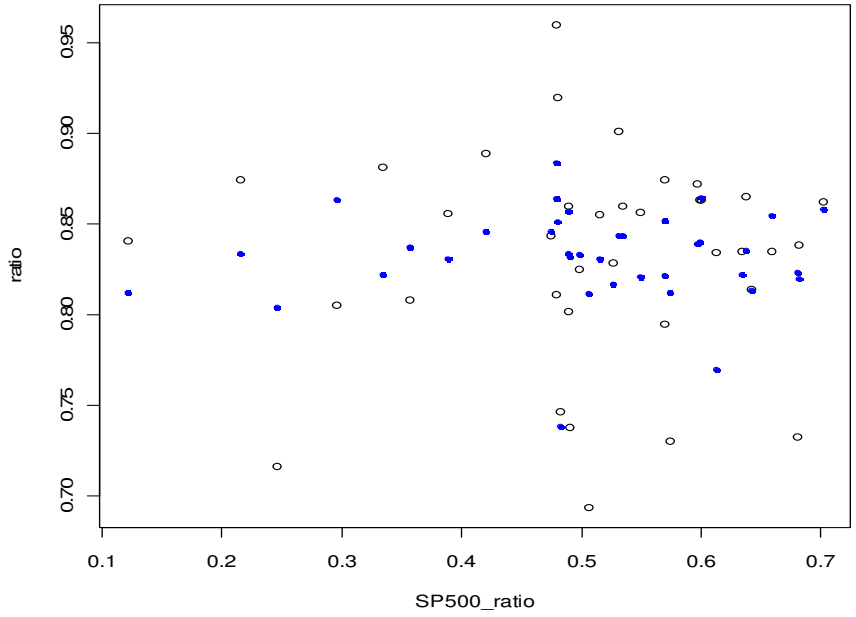
The customer fill ratio is shown on the y-axis, with other factors are on x-axis.

Black circles represent the original data while blue dots are predicted points.









The graphs verify relationships given by the correlation test: some for ack latency – a clearer trend shown, no for others – points scattering without any trend.

We are not satisfied with the correlation result. Since orders are merged daily, there may be some intra-day details neglected, so we continue the study to the per order view.

2.2. Per order

To repeat the correlation test and the linear regression model, data of each aggressive order in a day is collected. Because this only requires retrieving data from several days ago, more kinds of latencies are accessible. To look into details, metrics including hit ratio, router fill ratio, customer fill ratio and stalled ratio are calculated; latencies, number of live submissions (denoted as nb_submissions_live later) and market data are retrieved; others are duration, quantity over volume and price range.[4]

2.2.1. Correlation test

Correlation tests between the router fill ratio and all possible factors are done first. The following table shows the respective correlation coefficients calculated by per order stats. All tests are repeated for several days in order to examine whether results are consistent over time. The average is shown in the last column.[5]

Correlation between router fill ratio and:	Nov.19	Nov.23	Nov.24	Nov.29	Nov.30	Average
hit ratio	0.84	0.84	0.89	0.89	0.87	0.87
Ack latency	-0.26	-0.33	-0.39	-0.42	-0.38	-0.36
Internal latency	-0.24	-0.28	-0.37	-0.39	-0.37	-0.33
Order new internal latency	-0.24	-0.28	-0.37	-0.39	-0.37	-0.33
Force latency	-0.14	-0.14	-0.21	-0.13	-0.13	-0.15
Force ack latency	-0.19	-0.21	-0.23	-0.17	-0.22	-0.20
Market ack latency	-0.50	-0.42	-0.54	-0.42	-0.38	-0.46
duration	-0.15	-0.14	-0.20	-0.17	-0.17	-0.17
Number of live submissions	-0.16	-0.14	-0.21	-0.18	-0.17	-0.17
Stalled ratio	-0.37	-0.50	-0.40	-0.53	-0.57	-0.48
Quantity volume	-0.13	-0.27	-0.13	-0.18	-0.12	-0.17
Price range	-0.23	-0.18	-0.05	-0.03	-0.05	-0.11

As it shows, coefficient values are similar for each day. At least, they have the same pattern – greatest correlation for the hit ratio, larger value for most latencies and the stalled ratio, smaller value for other factors. Since the hit ratio is another measurement of successfulness of trades, the 0.87 correlation is not surprising.

It is noticeable that an around -0.5 correlation between the stalled ratio and the router fill ratio is found, which makes a large difference from the per day result. The impact of the market

[4] Outliers are removed for the following calculation.
[5] Weekends, Thanksgiving and the day after Thanksgiving are excluded.

ack latency is large, followed by the ack latency and the internal latency. Other factors do not have large influences.

Then correlation between the hit ratio and other factors were tested for the day of November 19. Values in the following table show a similar result as previous tests on the router fill ratio, though coefficients are generally larger for the hit ratio than the router fill ratio.[6]

	Hit ratio	Router fill ratio
Ack latency	-0.35	-0.26
Internal latency	-0.32	-0.24
Order new internal latency	-0.32	-0.24
Force latency	-0.25	-0.14
Force ack latency	-0.30	-0.19
Market ack latency	-0.67	-0.50
duration	-0.09	-0.15
Number of live submissions	-0.17	-0.16
Stalled ratio	-0.48	-0.37
Quantity volume	-0.02	-0.13
Price range	-0.30	-0.23

2.2.2. Linear regression

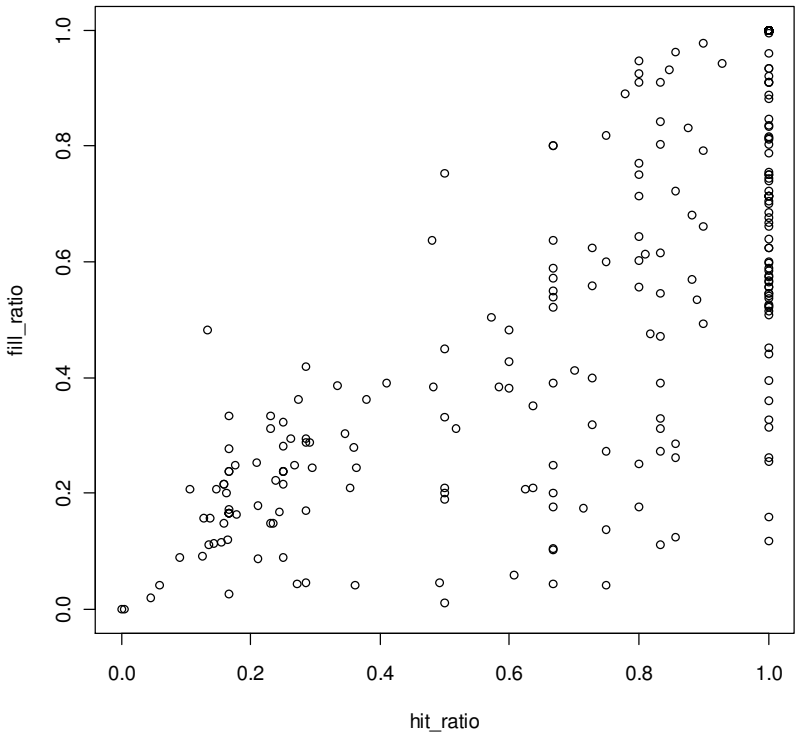
Since we have more data available, all factors are taken into account. The linear regression model is built as

fill_ratio ~ ack_latency + duration + nb_submissions_live + stalled_ratio + price_range + quantity_volume + internal_latency + order_new_internal_latency + force_latency + force_ack_latency + market_ack_latency

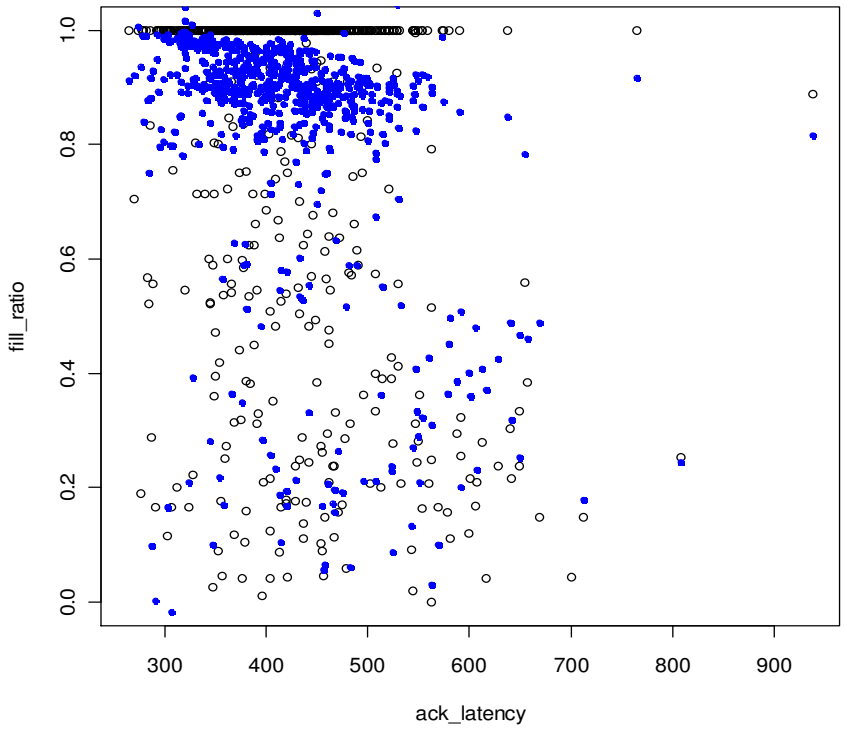
with a multiple R-squared 0.50 which is better than the per day model.[7]

Some scatter plots below shows the predicted points and verifies the correlation calculated previously. Same as before, black circles are original data while blue dots are predictions.

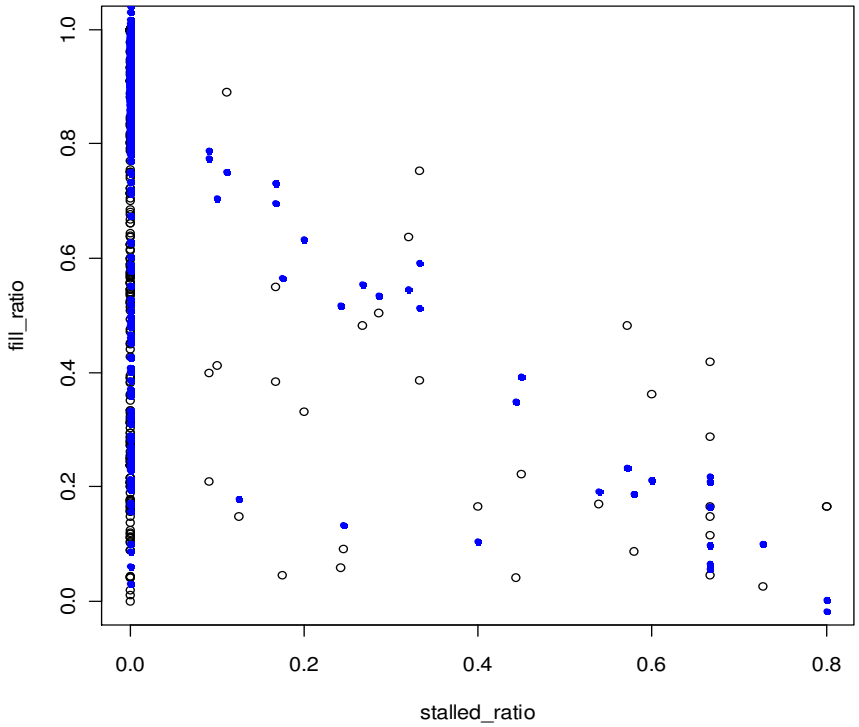
[6] Similar results for other days, so not shown repeatedly.
[7] Details of the built model attached in the appendix in Table 3.



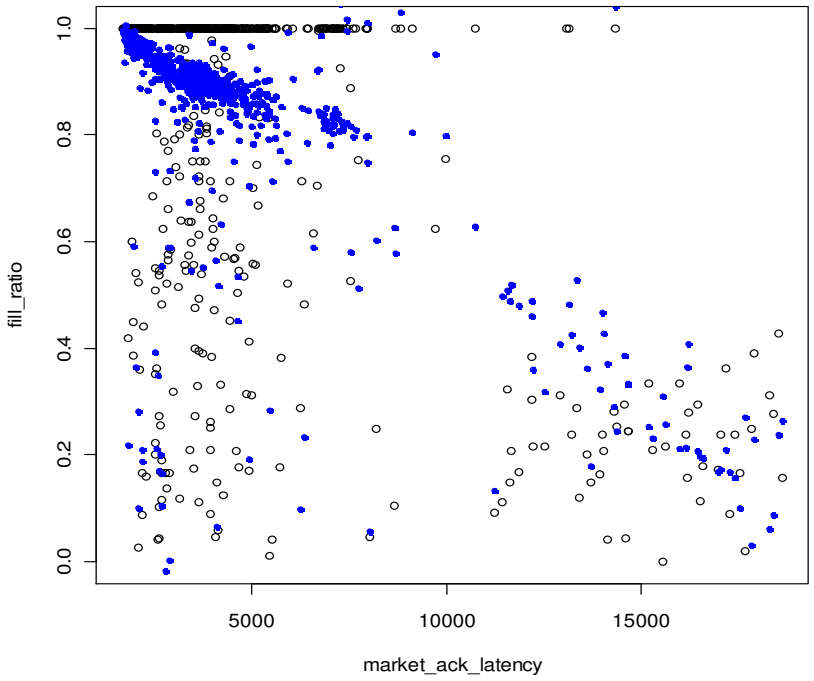
It is the graph of the router fill ratio plotted against the hit ratio. Most router fill ratio is lower than the hit ratio, which does not contradict with their definitions.



Plot of the router fill ratio against the ack latency is more scattered, since they have a smaller correlation of -0.3. Internal latency and force latency have similar patterns as this.



The router fill ratio against the stalled ratio has a clearer trend if the zeros are disregarded.



Although it is hard to find any trends when the latency is below 10000, we do see lower router fill ratios for higher latency (in the bottom-right corner).

2.2.3. Customer fill ratio

The customer fill ratio for each aggressive order in Nov.19 is calculated as well. The average is 0.96 (different from the average value obtained per day because this one is not weighted by the order quantity) with 91% of these values are one.[8] This can be interpreted as when a customer post an aggressive order, there is a 0.91 possibility that the order would be fully filled.

However, it also provides an obstacle for doing the correlation test and the linear regression model, since most values are one and hence the data set is not usable.

2.3. Per 5 minutes

The per order calculation might be over detailed and may contain noises. A way to smooth the data is to group orders created in the same 5 minutes time period together, and then redo the calculations. Taking orders created from the beginning of the day at 9:30 to the closing time 16:00, there should be 76 records in total.

2.3.1. Correlation test

Results of the correlation test are shown below. Strengthened correlations, especially between latencies and the router fill ratio, are found. Generally, absolute values of all correlation coefficients for latencies, despite the internal latency, have increased by more than 0.1. Market ack latency keeps the leading effect.

Correlation between router fill ratio and:	Nov.19	Nov.23	Nov.24	Nov.29	Nov.30	Average
hit ratio	0.91	0.89	0.90	0.93	0.92	0.91
Ack latency	-0.44	-0.33	-0.40	-0.39	-0.73	-0.46
Internal latency	-0.12	-0.27	-0.18	-0.52	-0.62	-0.34
Order new internal latency	-0.46	-0.24	-0.42	-0.02	-0.56	-0.34
Force latency	-0.44	-0.41	-0.19	-0.33	-0.38	-0.35
Force ack latency	-0.43	-0.41	-0.24	-0.44	-0.46	-0.40
Market ack latency	-0.68	-0.61	-0.40	-0.67	-0.73	-0.62
Number of live submissions	-0.15	-0.05	0.01	0.23	0.45	0.10
Stalled ratio	-0.37	-0.46	-0.29	-0.43	-0.30	-0.37
Quantity / volume	-0.13	-0.50	-0.41	-0.14	-0.26	-0.29
Price range	-0.23	-0.27	-0.01	0.23	-0.30	-0.12

[8] Results for other days are similar and thus not shown repeatedly.

2.3.2. Linear regression

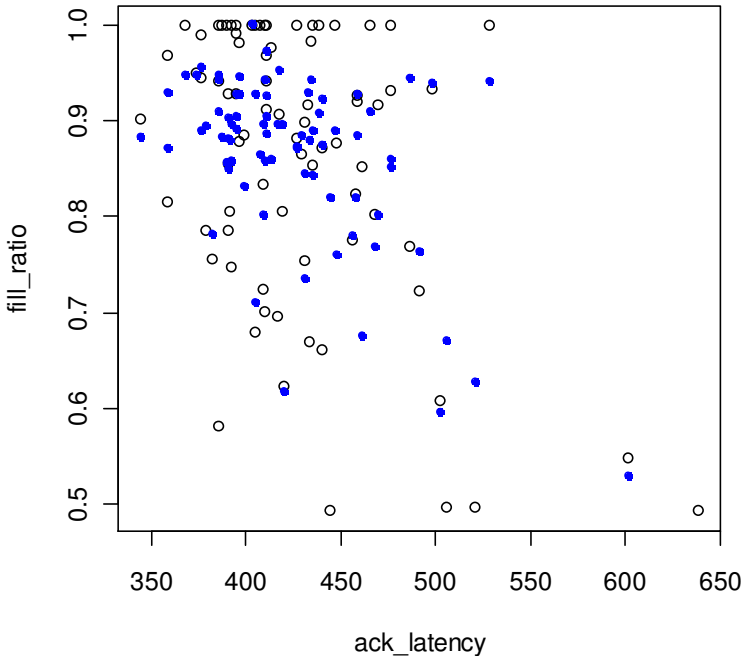
Again, the linear regression model helps to assess the linear relationships, and the model used here is the same as the per order model.

To provide a overview of the general effects from each factor, the approximate coefficients and averages of each independent variable in the model above are shown in the following table, and then their multiplications are calculated in the third column. The multiplication result shows how each factor affects the predicted router fill ratio in general. Note the ranking of these number is not necessary the same as that of correlation coefficient. (Data of November 19 are used)

	Coefficient	Average	Effect
Ack latency	-0.00004	400	-0.016
Internal latency	-0.000008	1700	-0.0136
Order new internal latency	-0.0001	500	-0.05
Force latency	-0.0001	900	-0.09
Force ack latency	-0.00005	1500	-0.075
Market ack latency	-0.00005	4500	-0.225
nb_submissions_live	-0.00002	500	-0.01
Stalled ratio	-1.9	0.01	-0.019
Quantity / volume	-0.3	0.02	-0.006
Price range	-0.6	0.01	-0.006

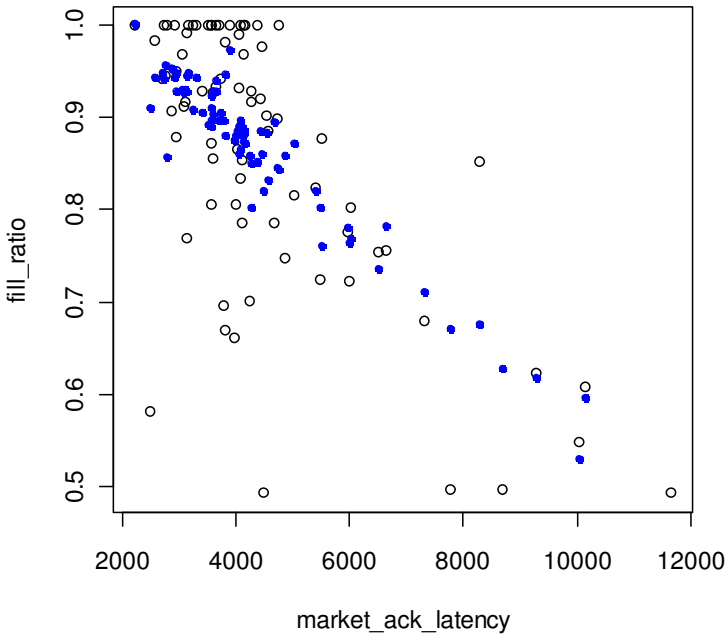
By comparing values in the third column, market ack latency has the largest impact, followed by force latency and force ack latency. However, this may not true for particular cases, for example, an order with a large internal latency and small market ack latency. All factors have a negative impact on the router fill ratio, so the intercept should be over 1.

Two scatter plots are attached below to show the strengthened trend after orders are grouped.



Compared to the scatter plot of per order stats showed previously, ack latency gives a clearer trend with a correlation -0.5.

The noticeable improvement also happens to other latencies, especially the market ack latency which is shown below.



3. Algorithm Implementation

While waiting for the computer to generating data, we did a small algorithm implementation used by the router in the submission process.

3.1. Problem description:

Find the most efficient way of sending submissions to different destinations having different latencies in one wave, so that the time taken to complete the order is minimized. (Only need to determine which destination to send to in this part.)

Destination	D1	D2	……	Dn
Listed Quantity	Q1	Q2	……	Qn
Latency	L1	L2	……	Ln

* Destinations are listed in the ascending order of latencies. (L1< L2<……< Ln)

* Total quantity (order quantity assigned by customer) is Q.

* Destinations are exchange markets, e.g. NYSE and BATS, which are pre-determined by the router.

*Assumptions:

- Listed quantities and the fill ratio does not change over time.
- The latency of each market does not change.

3.2. Purpose

Choose destinations that would give the most amount of quantities for sure in a given time period. (Amount of quantities to sent is to be determined by the router)

Core Idea:

1. Determine the number of destinations is needed to fill the order if Q is less than the total quantity listed on the market.
2. Compare the total amount of quantity can be got for sure (listed quantity) from all destinations to the total amount can be got from all destinations despite Dn, which has the largest latency time Ln, during time Ln.
3. In a certain time period (Ln), we will choose the strategy that would give a larger amount.

3.3. Algorithm:

// determine whether Q is less than the total quantity listed and find out the number of destinations needed

//find k so that: $\sum_{i=1}^{k-1} Q_i \le Q < \sum_{i=1}^{k} Qi$ (k is between 1 and n)

k = 0

total_listed_quantity = 0

While Q > total_listed_quantity

    If k < n:

        k = k + 1

        total_listed_quantity = total_listed_quantity + Q_k

    Else:

        End

// do the comparison

For i from k to 1:

    if $Q_1 + Q_2 + \cdots + Q_i \ge \left[\frac{L_i}{L_{i-1}} \right] (Q_1 + Q_2 + \cdots + Q_{i-1})$:

        choose destination {D1, …, Di}

        update Q to the remaining quantity

        end

* [] donates:

    if x < n+0.5, [x] =n

    if x >= n+0.5, [x]=n+1

    if x < 1, [x]=0

  where n is the largest positive integer

3.4. Examples:

Destination	D1	D2	D3
Listed Quantity	Q1 =100	Q2 =200	Q3 =300
Latency (in ms)	L1 =2	L2 =3	L3 =8

Scenario 1

Total quantity Q = 2000

First find k:

Q = 2000 > total listed quantity = 100 + 200 + 300 = 600

So k = 3

Choose destinations to send to

$Q_1 + Q_2 + Q_3$ = 100+200+300 = 600 < $\left\lceil \frac{L_3}{L_2} \right\rceil (Q_1 + Q_2)$ = 3*(100+200) = 900

So send submissions to only D2 and D1 (quantities to be determined by a router)

update Q after all results of submissions are received

Scenario 2

Total quantity Q = 250

First find k:

Q = 250 < total listed quantity by D1 and D2 = 100 + 200 = 300

So k = 2

Choose destinations to send to

$Q_1 + Q_2$ = 100+200 = 300 > $\left\lceil \frac{L_2}{L_1} \right\rceil (Q_1)$ = 2*(100) = 200

So send submissions to both D2 and D1 (quantities to be determined by a router)

update Q

3.5. Comments

Consideration of expected quantity:

Now only the listed quantity is considered when doing the comparison. A more precise way is replacing quantity listed by expected quantity calculated by the router (can include the expected reserved quantity) for each destination. For example, let Q1 = listed quantity + expected reserved quantity.

However, there is a trade off of some decision errors when the actual filled quantity is less than our expectations.

Consideration for subwaves

This algorithm is designed to send a big wave to all chosen destinations, which does not involve any subwaves. A subwave means sending another wave to destinations which respond quicker without waiting for the response from other slow destinations. We could further improve the algorithm by taking into account the parallelism.

From collected data, destinations with latencies less than 4ms (for example) can be chosen to be included in a subwave under the full wave which is determined by latencies that are greater than 4ms. Then the efficiency of completing an order will increase, but any fixed cost involved in sending waves would rise.

Recommendations and Conclusions

The Blizzard Visualization Tool (BVT) successfully meets all of the functional requirements as specified at the beginning of this project. The tool can be used to visually monitor and analyze the behavior of Blizzard, allowing the user to get a detailed look at individual orders. It gives the user a high level overview of Blizzard's behavior and performance over the course of an order, but also allows the user to zoom into the order to get a closer look at a specific section of the order to isolate issues. To fulfill the second functional requirement, the metrics table section of the tool computes various metrics on all orders within a specified day and displays the N orders with the worst results, greatly simplifying the search for anomalies and problem areas. The latency charts also serve to improve the ease of finding problem orders.

The architecture of the system takes all of the non-functional requirements into consideration and successfully meets all of them. Because of the performance of the native charting library of ActionScript and the client-side caching and pre-loading strategies of the BVT, the tool has a very low perceived response time, making the user experience very smooth. Because of the server-side caching of function and query results, the performance impact to the Blizzard production database is also minimized. Extensibility and reusability were heavily prioritized in the design and development of this tool and all of its supporting components. And hence, future modifications will be relatively painless and feature requests can be rapidly implemented. But because of the emphasis on generic and application agnostic development for each of the individual components, the different parts of this application can also be easily integrated into other projects.

The purpose of the data analysis is to find factors that affect the fill ratio most. To examine the linear relationship between the fill ratio and different factors, the correlation test and the linear regression model are applied, along with the scatter plot which provides a vivid demonstration and helps to verify the relationship between two variables.

Overall, the weighted average customer fill ratio calculated per day is around 0.83, with most market orders filled and more orders using Omega and Foes strategy filled than the average. Per order analysis gives a result of over 90% aggressive orders are fully filled. Combining the three kinds of assessing methods (per day, per order and per 5 minutes), the conclusion reached is factors influencing the fill ratio most are the stalled ratio, ack latency, internal latency, force latency and market ack latency, each with an absolute value of correlation coefficient over 0.4 and up to 0.7.

Due to the limit time and the access of detailed market data, we do not have a chance to examine the correlation between the fill ratio and market indexes such as the price movement and the volatility. It would be valuable to compare the impact of these factors to the fill ratio to the result achieved above, since in this project most factors considered are from the router side which is under our control.

Appendix

Table 1 Calculated customer fill ratio per day

date	ratio	PriceMarch	Omega	Dark	DarkPlus	Foes
9/1/2010	0.730136021	0.661259052	0.98902379		0.054921001	0.772174053
9/2/2010	0.814176255	0.689876636	0.900484246	1	0.411955	0.891260782
9/7/2010	0.864994434	0.626234098	0.995995996		0.161219931	0.938771417
9/8/2010	0.838271239	0.304512347	0.992546417	0.904	0.405544528	0.88842077
9/10/2010	0.859788536	0.641092442	0.997644382		0.796812749	0.91138274
9/13/2010	0.881535843	0.663803957	0.994930233	1	0.829512894	0.954570125
9/15/2010	0.874354607	0.740685509	0.994389182	0	0.539170507	0.90272009
9/16/2010	0.737927739	0.624631483	1		0.02087111	0.967551307
9/17/2010	0.807864999	0.695012729	0.998296713	0.522885057	1	0.864165709
9/20/2010	0.960405616	1	0.997060553			0.959547945
9/21/2010	0.810766739	0.59171844	1	0.43902439		0.819459204
9/23/2010	0.802009673	0.875	1	0.517241379	0.995867769	0.798689597
9/24/2010	0.889210438	0.793235942	0.999880433	1		0.89395524
9/27/2010	0.91989832	0.864037225	1	0.932773109		0.925207167
9/28/2010	0.746252864	0.649955596	0.970406926	0.942350333	1	0.76477006
9/29/2010	0.855257362	0.547234224	1		0.05628	0.900728535
9/30/2010	0.716060453	0.603790983	0.997384481		0.439111531	0.869357738
10/1/2010	0.805195601	0.713402932	0.907337088	1	0	0.817480235
10/4/2010	0.834303644	0.444170499	0.997806921		0.048294977	0.88675694
10/5/2010	0.843959157	0.575042364	0.948818898	0.75		0.894269159
10/6/2010	0.825147812	0.565319182	0.982683983		0.0003	0.915094294
10/7/2010	0.863453377	0.548976183	1	0.681818182		0.900708072
10/8/2010	0.693274228	0.666130955	0.952829582	0.990868788	0.023929523	0.833084241
10/11/2010	0.732380755	0.827631704	0.692381679		0.068890768	0.79409548
10/12/2010	0.794471158	0.59739755	0.991003894		0.138622048	0.862216946
10/13/2010	0.859911362	0.570485481	0.962424114	0.175682638	0.396384969	0.891451873
10/14/2010	0.828607367	0.759189501	0.943207295	0.330827068	0.236135647	0.859773702
10/15/2010	0.863189374	0.614661359	0.833823938		0	0.886020352
10/18/2010	0.87428386	0.606490749	1	0.0166	0.0323	0.932460387
10/19/2010	0.835014369	0.741972537	0.952081753	0.186666667	0.1583	0.874936223
10/20/2010	0.855623913	0.635196655	1	0.079807692		0.945178598
10/21/2010	0.840488724	0.739814156	1	0.74004499		0.922056469
10/22/2010	0.901178157	0.803361272	0.995860516	1		0.906980946
10/25/2010	0.85628488	0.545332966	0.830107057			0.945950378
10/26/2010	0.862528699	0.746538261	0.947826087			0.924661396
10/28/2010	0.872039666	0.621956892	0.670457892			0.949011002
10/29/2010	0.834723862	0.666524191	0.982603653	0.349663339	0.826581353	0.93279969

*Blanks without data represent no such strategy used on that day.

*Weekends and days having error are excluded.

Table 2 Per day data for correlation test

date	customer_fill_ratio	ack_latency	nb_orders	SP500_ratio	stalled_ratio

9/1/2010	0.730136021	435.0226793	5689	0.573914572	0.029863753
9/2/2010	0.814176255	432.6025722	4744	0.642074199	0.023748086
9/7/2010	0.864994434	394.5926375	4945	0.636804853	0.020467693
9/8/2010	0.838271239	424.6712398	5904	0.680894309	0.034792727
9/10/2010	0.859788536	408.5487122	3579	0.534506845	0.059290383
9/13/2010	0.881535843	439.4689407	3610	0.33434903	0.051511487
9/15/2010	0.874354607	399.9527199	5902	0.569467977	0.080954556
9/16/2010	0.737927739	409.7431152	4540	0.490528634	0.033917869
9/17/2010	0.807864999	424.3937941	3964	0.356710394	0.069531627
9/20/2010	0.960405616	390.4588051	5741	0.478488068	0.096902863
9/21/2010	0.810766739	387.1166704	4457	0.479021764	0.139834407
9/23/2010	0.802009673	423.7729206	4240	0.488679245	0.127523911
9/24/2010	0.889210438	422.3116842	3141	0.419929959	0.087335552
9/27/2010	0.91989832	383.3723187	3916	0.479826353	0.038603426
9/28/2010	0.746252864	618.4294374	5293	0.48233516	0.140158687
9/29/2010	0.855257362	413.4815156	7609	0.515442239	0.052866384
9/30/2010	0.716060453	464.4910768	18330	0.24599018	0.123254097
10/1/2010	0.805195601	407.5403402	5937	0.295940711	0.124321303
10/4/2010	0.834303644	508.8963191	10081	0.61194326	0.062697352
10/5/2010	0.843959157	417.0348745	3791	0.474544975	0.082417091
10/6/2010	0.825147812	414.2048569	4160	0.498317308	0.043392352
10/7/2010	0.863453377	407.0242372	3509	0.60045597	0.117019926
10/8/2010	0.693274228	437.9202489	3536	0.506504525	0.020783658
10/11/2010	0.732380755	421.0297511	3296	0.679915049	0.024323205
10/12/2010	0.794471158	441.7341855	3052	0.569790301	0.052578208
10/13/2010	0.859911362	417.0330923	3597	0.489018627	0.047248018
10/14/2010	0.828607367	441.5758033	4639	0.526406553	0.047818241
10/15/2010	0.863189374	412.5354118	5557	0.599064243	0.068465587
10/18/2010	0.87428386	394.8238411	10573	0.215454459	0.042470187
10/19/2010	0.835014369	428.8946246	4688	0.658916382	0.134121345
10/20/2010	0.855623913	456.5206798	8440	0.388744076	0.132490379
10/21/2010	0.840488724	468.3593466	18183	0.12137711	0.150144092
10/22/2010	0.901178157	407.6076741	4640	0.53125	0.062977516
10/25/2010	0.85628488	462.3638743	3056	0.548756545	0.08791952
10/26/2010	0.862528699	411.7968723	5826	0.701338826	0.11994017
10/28/2010	0.872039666	428.3738584	3505	0.596291013	0.082141906
10/29/2010	0.834723862	441.0615217	2749	0.633321208	0.052680094

Table 3 Linear regression model coefficients from per order data

Coefficients	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.11E+00	4.06E-02	27.465	< 2e-16
ack_latency	-1.94E-04	2.68E-04	-0.724	0.4695
duration	3.19E-09	4.59E-08	0.07	0.9445
nb_submissions_live	-1.88E-05	2.51E-05	-0.75	0.4537
stalled_ratio	-1.18E+00	8.27E-02	-14.248	< 2e-16
price_range	-6.29E-01	4.51E-01	-1.393	0.164

quantity_volume	-1.32E-01	2.67E-02	-4.965	8.75E-07
internal_latency	-9.66E-06	2.15E-04	-0.045	0.9642
order_new_internal_latency	NA	NA	NA	NA
force_latency	5.65E-05	2.78E-05	2.032	0.0426
force_ack_latency	2.45E-05	2.22E-05	1.104	0.2698
market_ack_latency	-5.63E-05	3.52E-06	-15.975	< 2e-16

* 1 variable not defined because of singularities