BNP PARIBAS | The bank for a changing world

Equity Smart Order Router

Huan Lai
Xiaoyun Wang
December 14, 2010

WORCESTER POLYTECHNIC INSTITUTE · 1865 · WPI

Goal / Purpose

The problem:

- Blizzard is a complex system with lots of moving parts
- The monitoring software in place is essentially a filtered log events, leaving it up to the user to interpret what's going on

So what can we do?

- Develop a tool that can be used to visualize Blizzard
- Use that tool in order to come up with useful statistics for the business

Functional Requirements

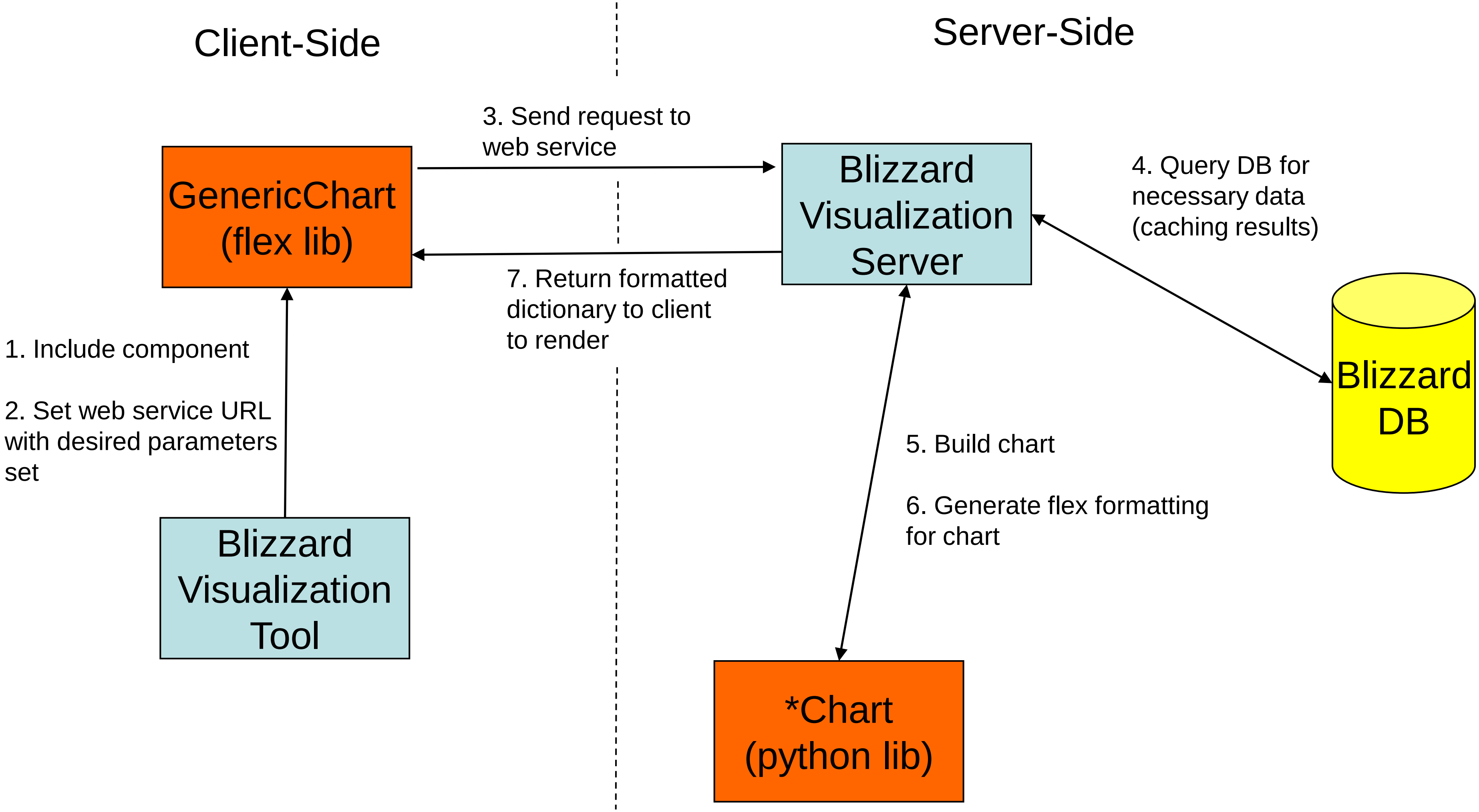
- Develop a tool that can be used to visually monitor and analyze the behavior and performance of Blizzard

- Develop a tool that can be used to compute and display various metrics on orders to find anomalies and problem areas

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

Architecture / Flow



Client-Side

Server-Side

3. Send request to web service

4. Query DB for necessary data (caching results)

7. Return formatted dictionary to client to render

1. Include component

2. Set web service URL with desired parameters set

5. Build chart

6. Generate flex formatting for chart

Charting Framework

Implementation of functional requirements:

- *Develop a tool that can be used to visually monitor and analyze the behavior and performance of Blizzard*
  - Framework supports all of the major charting functionality within Flex while significantly improving development time
  - Framework adds user interactions not built into the Flex Charting library and fixes some of the inherent limitations

Charting Framework

Considerations for non-functional requirements:

- *Make components of the tool reusable for other projects in the future*
  - Develop the framework to be general purpose and data agnostic
  - All major design and implementation of the framework is done before, not simultaneously with, the Blizzard Visualization Tool
- *Develop the tool to be extensible and easy to maintain*
  - Features required by the Blizzard Visualization Tool but not initially implemented added later, but coupling minimized
  - Broken into two components:
    - – Client side component (Flex)
    - – Server side component (python)

Client Side Component

- Handles making request to web service and parsing results
  - URL of the server and formatting of the parameters passed to the framework by user application to avoid coupling
  - Formatting assumed to be as given by server side library

- Renders chart based on specifications given by web service
  - All details of chart are specified by the web service – allowing for changes to be deployed without having to force the clients to update

- Handles advanced user interactions
  - Zooming and scrolling
  - Enabling/disabling different components of the chart

Server Side Component

- Allows for user to build the chart up piece-by-piece
- Generates response that describes the chart in a format that the client side component of the library can understand

Sample Code – Client Side

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>
```

Tell GenericChart to load content from URL

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

Add component to flex project

Sample Code – Server Side

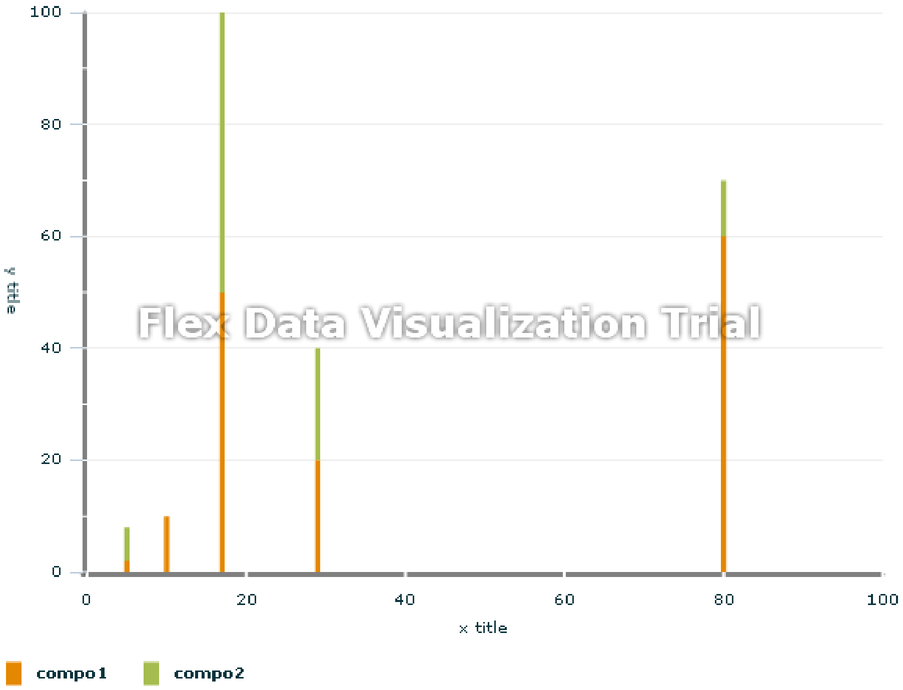
```
chart = BarChart()

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')
```

And the resulting chart is:



Blizzard Visualization Tool

Implementation of functional requirements:

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

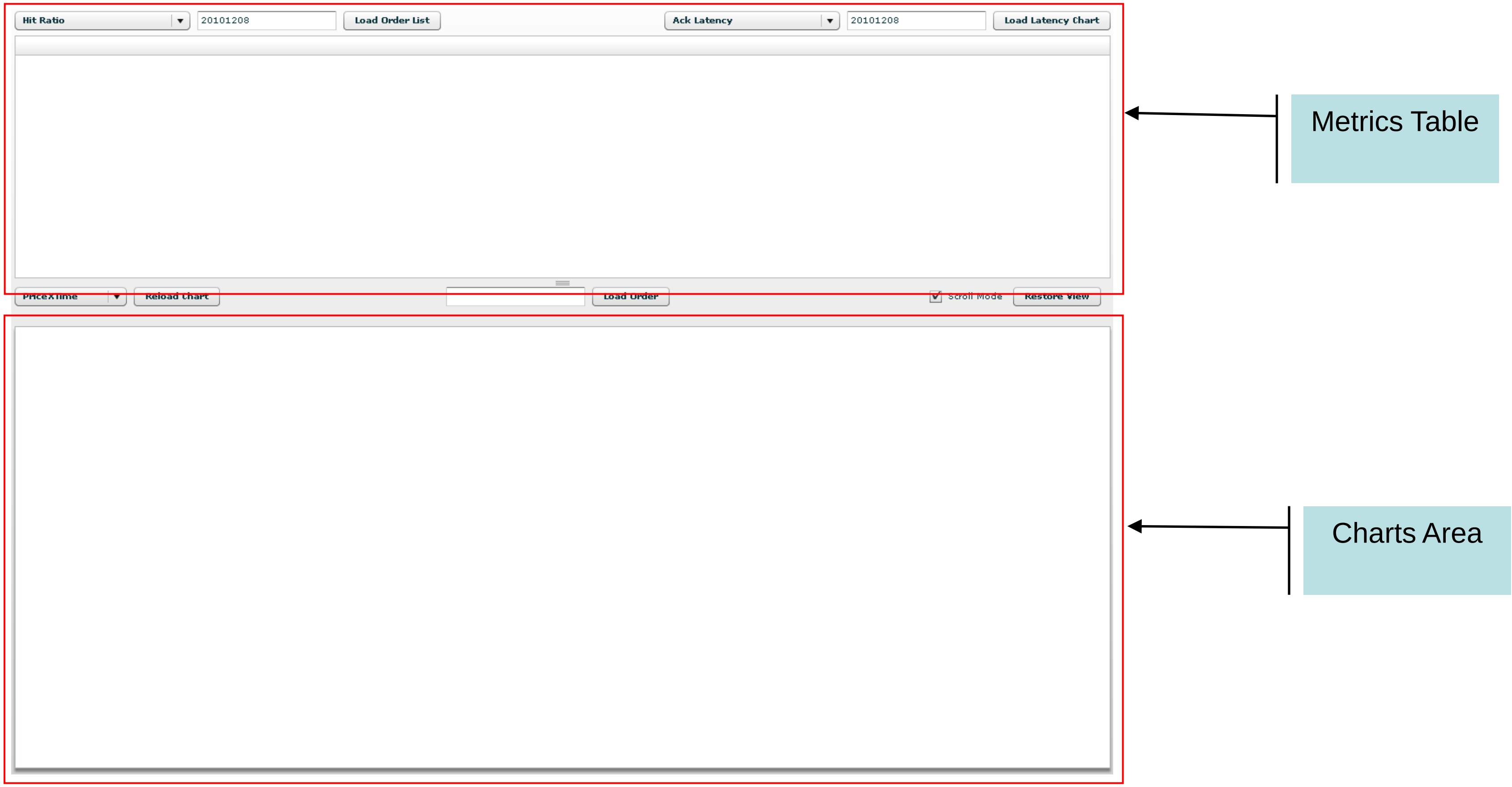
Blizzard Visualization Tool

Considerations for non-functional requirements:

- *Develop the tool to be extensible and easy to maintain*
  - Types of charts and metrics simply lists of titles and URL, so adding/removing types are one line changes
  - No dependencies between each type of chart or metric
- *Minimize performance impact to database*
  - Results of SQL queries that require complex calculations or joins of large tables cached
- *Minimize perceived response time*
  - Tabbing allows for multiple charts to be open and only rendered once

Blizzard Visualization Tool

Tool split into two sections:



Hit Ratio ▾ 20101208

Ack Latency ▾ 20101208

PriceXTime ▾

☑ Scroll Mode

Metrics Table

Charts Area

Metrics table

- The metrics table allows the user to select any given day and computes the desired metric over all orders that were processed that day
  - Hit ratio, stalled ratio, fill ratio, submission fill ratio
  - Internal latency, order new internal latency

- Returns the top N orders that have the “worst” scores for that metric
  - Allows the user to quickly isolate orders that have issues

Charts

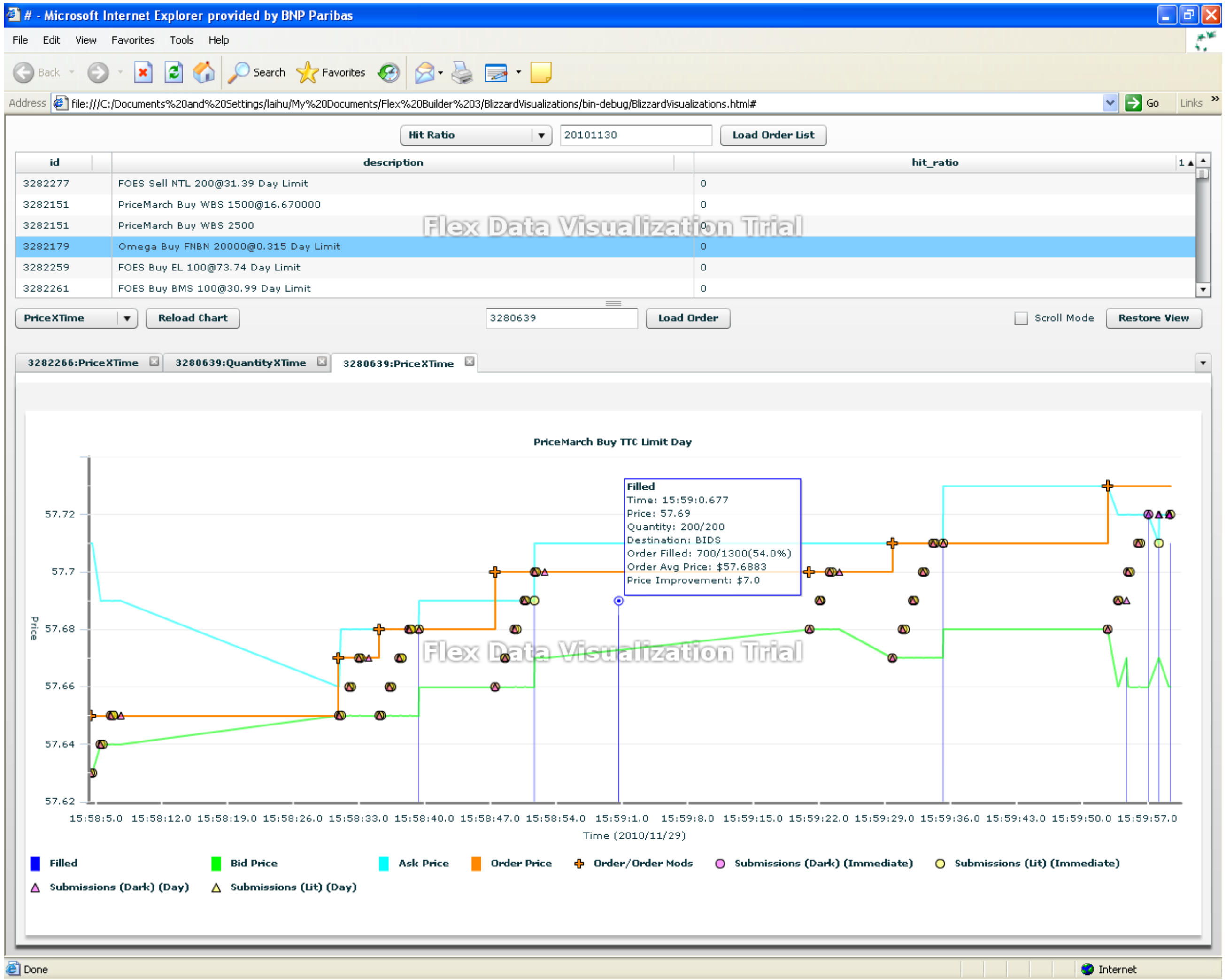
- Currently implemented are three types of charts:
  - Price-by-time chart
    - Shows the historic price of the order and all order mods over time
    - Shows all submissions to the markets and fills associated with those submissions
    - Shows the historic market prices for the desired stock
  - Quantity-by-time chart
    - Shows the quantity of shares the order calls for over time
    - Shows the quantity of shares executed over time
  - Latency-by-time chart
    - Shows the latency of every order over the course of a day
    - Supports multiple types of latency measurements

Blizzard Visualization Tool – Live Demo

- http://wbedevserv12.us.net.intra/blizzard_dev/Visualization/devserv11.html#

- Orders to demo:
  - 3118464 (Omega)
  - 3280639 (PriceMarch)

Price-by-time Chart



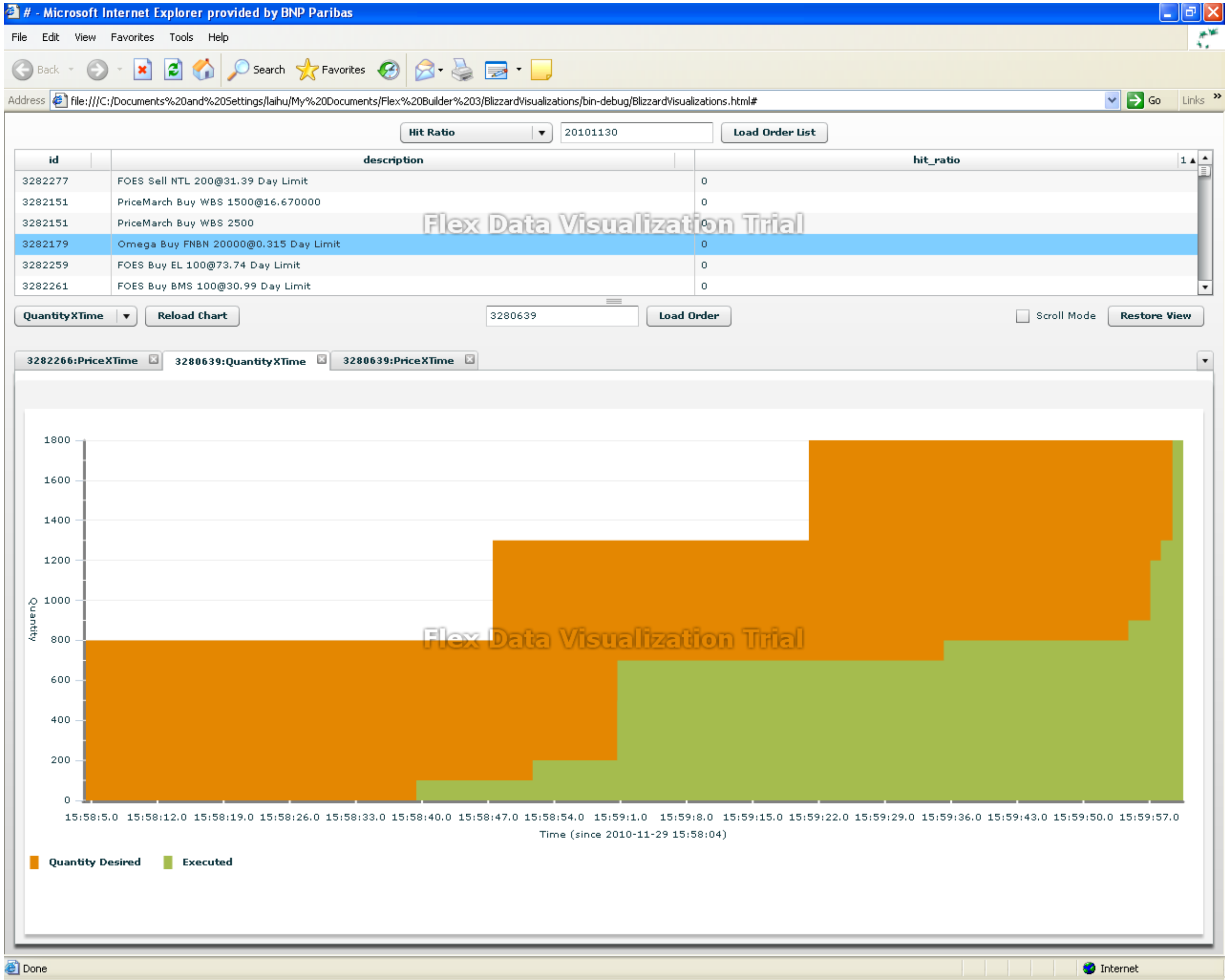
# - Microsoft Internet Explorer provided by BNP Paribas
File Edit View Favorites Tools Help
Back Search Favorites
Address file:///C:/Documents%20and%20Settings/laihu/My%20Documents/Flex%20Builder%203/BlizzardVisualizations/bin-debug/BlizzardVisualizations.html#
Go Links
Hit Ratio
20101130
Load Order List
Flex Data Visualization Trial
PriceXTime
Reload Chart
3280639
Load Order
□ Scroll Mode
Restore View
3282266:PriceXTime
3280639:QuantityXTime
3280639:PriceXTime

id	description	hit_ratio
3282277	FOES Sell NTL 200@31.39 Day Limit	0
3282151	PriceMarch Buy WBS 1500@16.670000	0
3282151	PriceMarch Buy WBS 2500	0
3282179	Omega Buy FNBN 20000@0.315 Day Limit	0
3282259	FOES Buy EL 100@73.74 Day Limit	0
3282261	FOES Buy BMS 100@30.99 Day Limit	0

PriceMarch Buy TTC Limit Day

Filled
Time: 15:59:0.677
Price: 57.69
Quantity: 200/200
Destination: BIDS
Order Filled: 700/1300(54.0%)
Order Avg Price: $57.6883
Price Improvement: $7.0

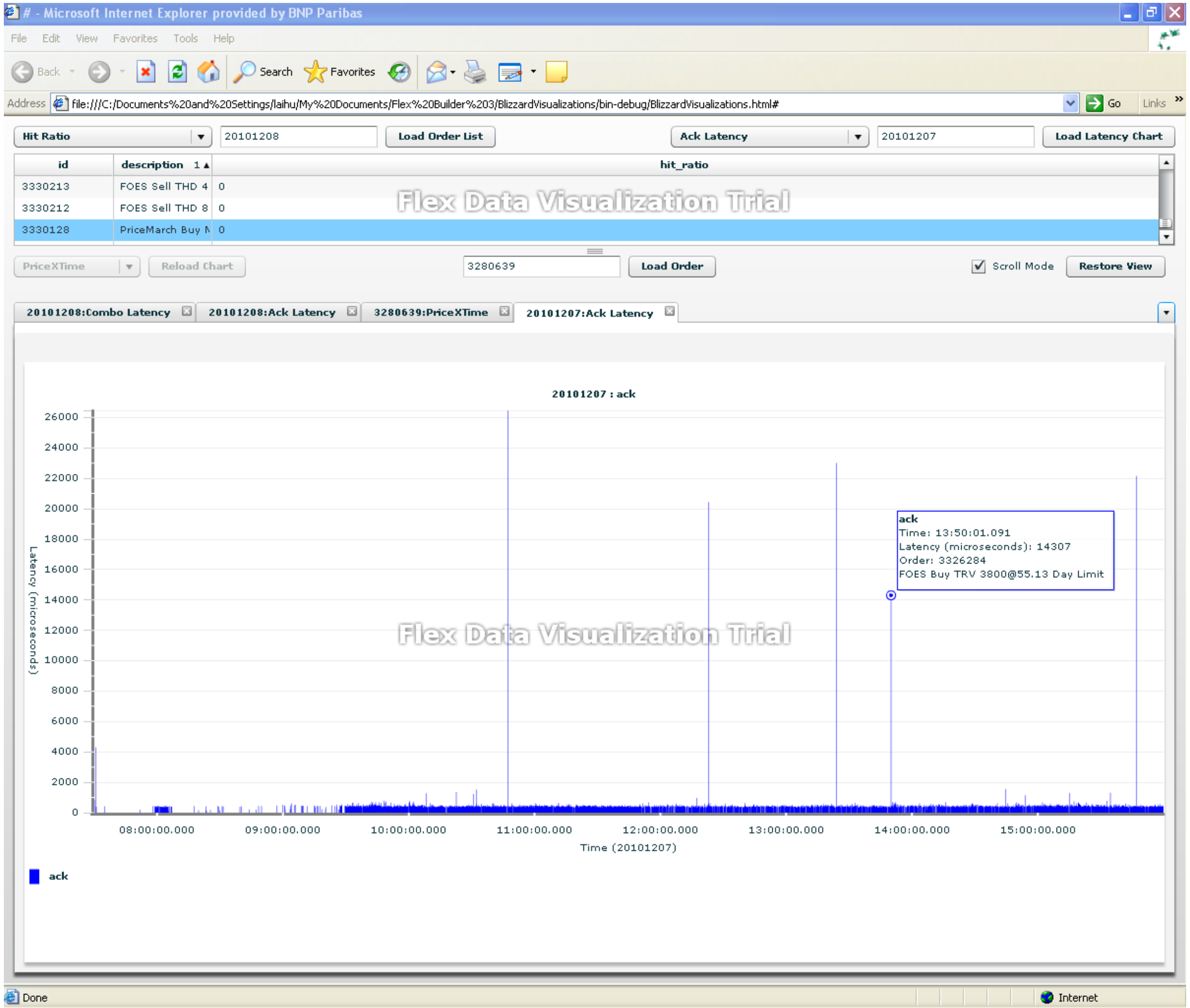
Quantity-by-time Chart



# - Microsoft Internet Explorer provided by BNP Paribas
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
file:///C:/Documents%20and%20Settings/laihu/My%20Documents/Flex%20Builder%203/BlizzardVisualizations/bin-debug/BlizzardVisualizations.html#
Go
Links
Hit Ratio
20101130
Load Order List

id	description	hit_ratio
3282277	FOES Sell NTL 200@31.39 Day Limit	0
3282151	PriceMarch Buy WBS 1500@16.670000	0
3282151	PriceMarch Buy WBS 2500	0
3282179	Omega Buy FNBN 20000@0.315 Day Limit	0
3282259	FOES Buy EL 100@73.74 Day Limit	0
3282261	FOES Buy BMS 100@30.99 Day Limit	0

Latency-by-time Chart



- Microsoft Internet Explorer provided by BNP Paribas

Address file:///C:/Documents%20and%20Settings/laihu/My%20Documents/Flex%20Builder%203/BlizzardVisualizations/bin-debug/BlizzardVisualizations.html#

Hit Ratio ▾ 20101208 Load Order List

Ack Latency ▾ 20101207 Load Latency Chart

id	description 1 ▴	hit_ratio
3330213	FOES Sell THD 4	0
3330212	FOES Sell THD 8	0
3330128	PriceMarch Buy N	0

PriceXTime ▾ Reload Chart 3280639 Load Order ☑ Scroll Mode Restore View

20101208:Combo Latency ☒ 20101208:Ack Latency ☒ 3280639:PriceXTime ☒ 20101207:Ack Latency ☒

ack
Time: 13:50:01.091
Latency (microseconds): 14307
Order: 3326284
FOES Buy TRV 3800@55.13 Day Limit

Done

Data Analysis

- Goal:
  Overview of router performance
  Find factors that influence the successfulness of trades which is measured by the fill ratio.

- Result:
  Stalled ratio and latencies have the largest impacts

Background

Per day

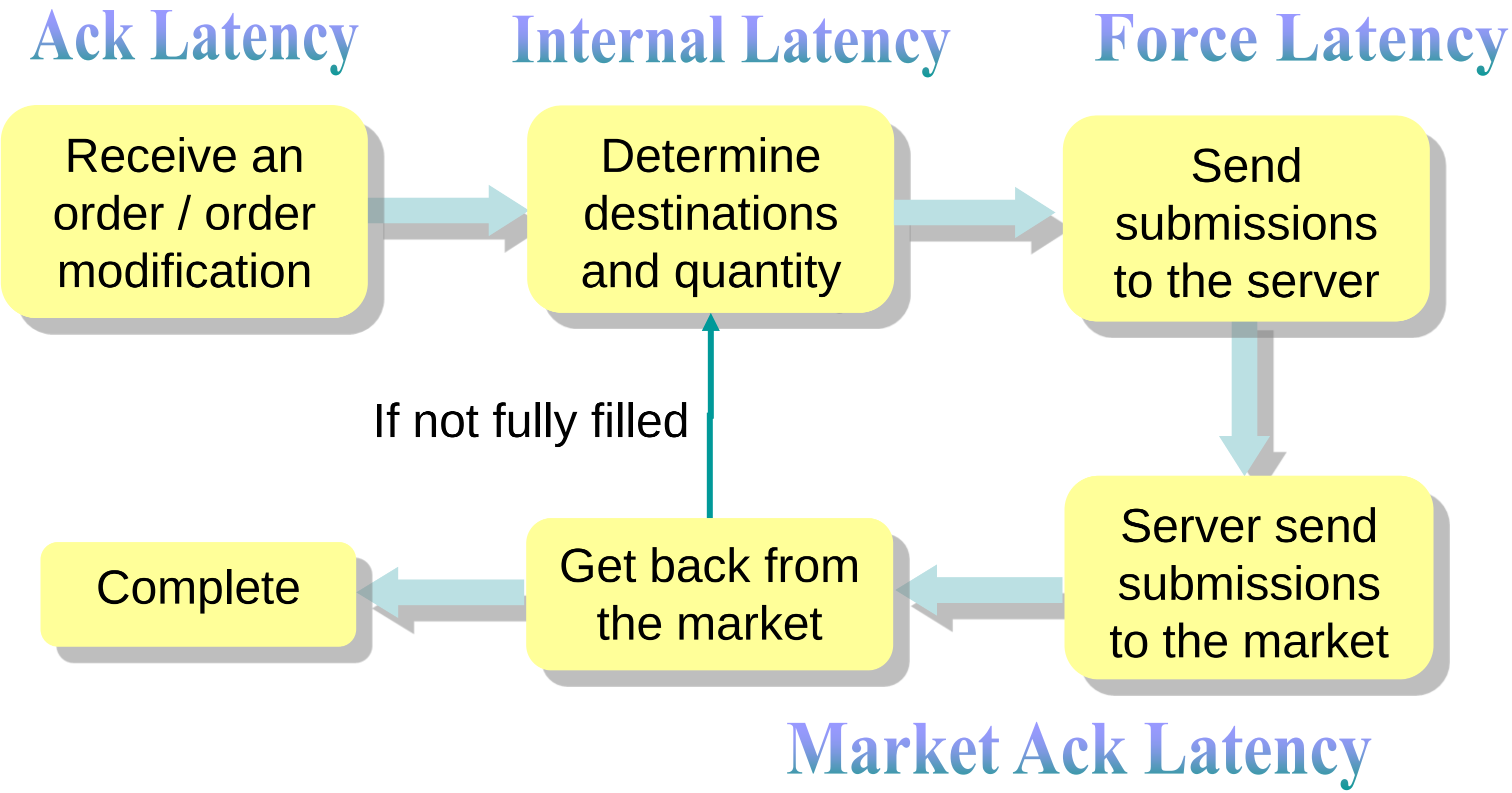
Per Order

Per 5 minutes

Summary

- Note: only aggressive orders are counted

Background – Router Specifications



Ack Latency

Internal Latency

Force Latency

Receive an order / order modification

Determine destinations and quantity

Send submissions to the server

Server send submissions to the market

Get back from the market

If not fully filled

Complete

Market Ack Latency

Background – Metrics used

- Router fill ratio

  Filled quantity / total quantity sent

- Customer fill ratio

  Filled quantity / order quantity

Order Qty = 500
Total Sent Qty = 1000
Filled Qty = 500

- Hit ratio

  Number of executed submissions / total number of submissions

- Stalled ratio

  Number of successive unexecuted submissions at the same price from the same destination / total number of submissions

- Latencies

Per Day Result

- Group data daily:
  - Customer fill ratio = sum of filled quantity / sum of total quantity
- Customer fill ratio on average (from Sep.1 to Oct.30) :

	Customer fill ratio	PriceMarch	Omega	Dark	DarkPlus	Foes
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**
Percentage	**1**	**10.1%**	**9.4%**	**0.3%**	**0.2%**	**79.9%**

- Different strategies have different performance
- Foes is the most used strategy, while Dark and Dark Plus are seldom used

Per Day Result – further grouped by

Market & Limit

Market order:

- Completing orders under the market price
- Average fill ratio = 0.998

Limit order:

- Have specific price limits required by customers
- Similar results as before, because most orders are limit

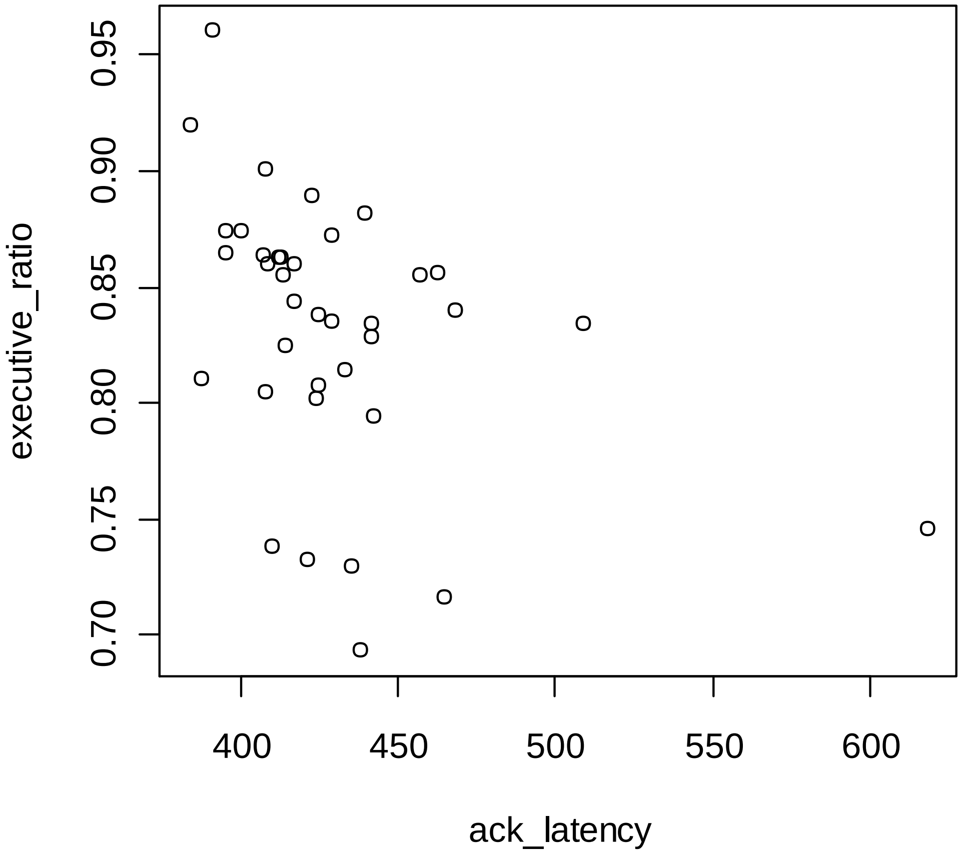
Buy & Sell

- Group by side:
  Slightly higher fill ratio on buy side than sell side
  Varies day to day

- Consistent when further grouped to limit buy and limit sell

Per Day Result – Correlation

Found -0.4 correlation between ack latency and customer fill ratio



Per Order Result – Router fill ratio

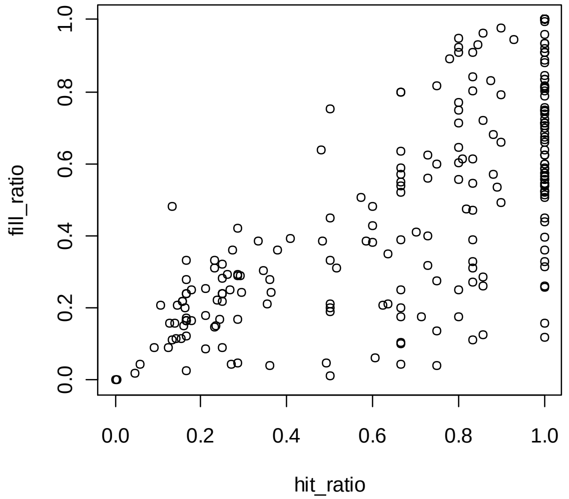
- Get data and calculate stats for each order
- Do the correlation test
- Repeat it for several days
- Stalled ratio, ack latency and market ack latency have more impacts

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**

Per Order Result – Nov.19

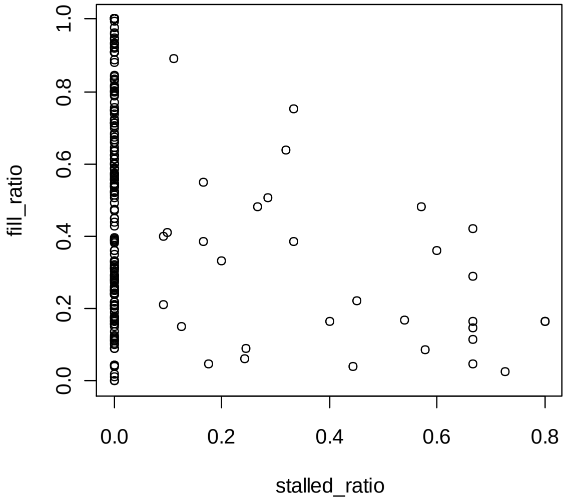
Hit ratio:

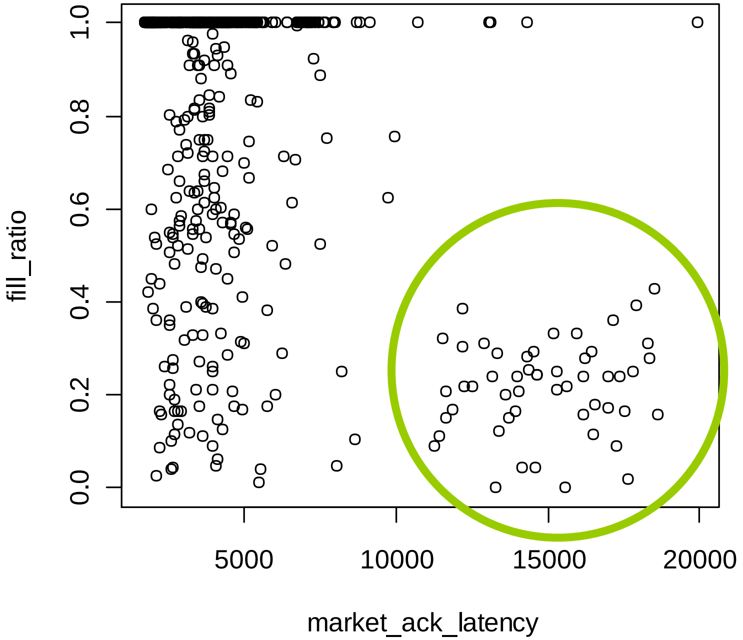
0.8



Stalled ratio:

-0.4





Market ack latency:

-0.5

Per Order Result – Customer fill ratio

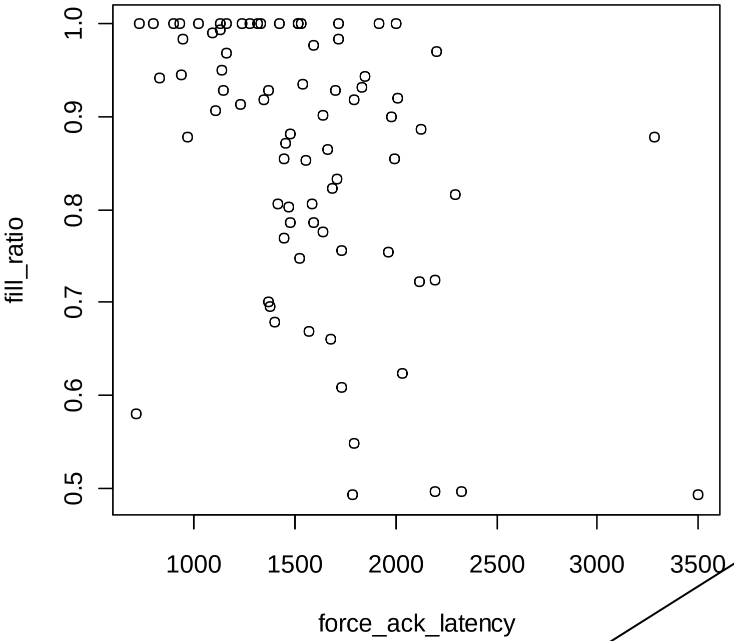
- Lack of data to do the correlation test
- 91% customer fill ratios are 1 on Nov.19
  - 0.91 possibility to get fully filled
  - Average = 0.96
  - (different from 0.88 because not weighted by quantity)
- 92% for Nov.24
  - Average = 0.95

Per 5 minutes Result

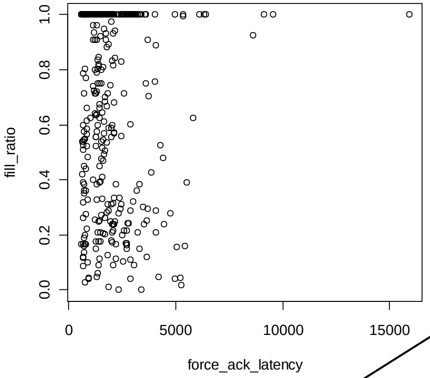
- Group orders created in the 5 minutes period
- Strengthened correlation for latencies

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	**0.92**
Ack latency	-0.44	-0.33	-0.40	-0.39	-0.73	**-0.48**
Internal latency	-0.12	-0.27	-0.18	-0.52	-0.62	**-0.39**
Order new internal latency	-0.46	-0.24	-0.42	-0.02	-0.56	**-0.35**
Force latency	-0.44	-0.41	-0.19	-0.33	-0.38	**-0.37**
Force ack latency	-0.43	-0.41	-0.24	-0.44	-0.46	**-0.41**
Market ack latency	-0.68	-0.61	-0.40	-0.67	-0.73	**-0.65**
Number of live submissions	-0.15	-0.05	0.01	0.23	0.45	**0.10**

Per 5 minutes Result – compared to per order

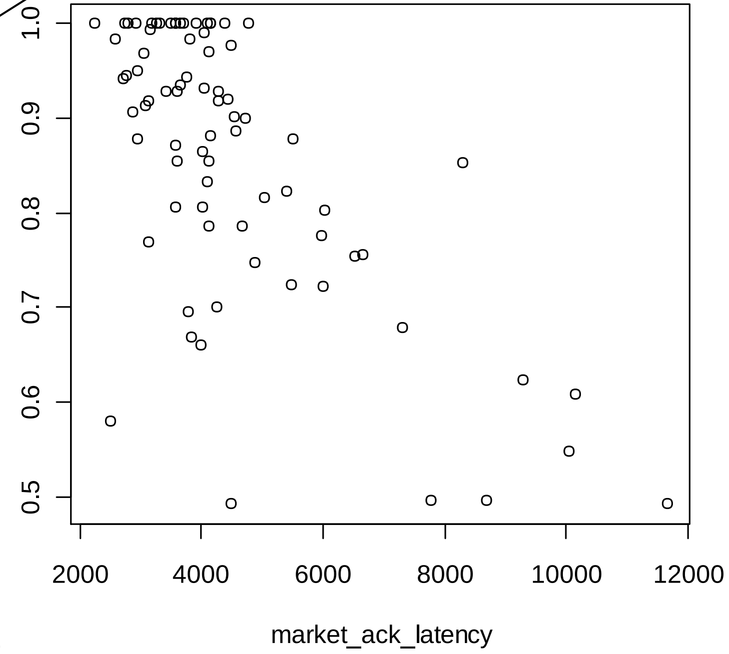


-0.4

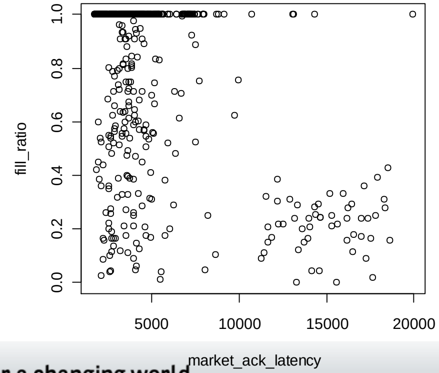


-0.2

-0.7

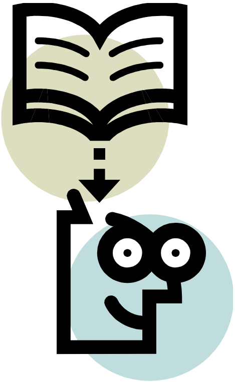


-0.5



Summary

- Weighted average customer fill ratio is around 0.83 (from Sep.1 to Oct.30)
- Over 90% aggressive orders are fully filled (late November)
- Factors influence the fill ratio most:
  - Stalled ratio
  - Ack latency
  - Internal latency
  - Market ack latency



Questions?

Free to Contact us:

Xiaoyun Wang wang@wpi.edu

Huan Lai huanlai@wpi.edu

Thanks to BNP Paribas and people who support us:

David Jobet (our mentor)
david.jobet@americas.bnpparibas.com

Scott Visconti

Christophe Poulmarc'k

Thanks to WPI and our advisors:

Prof. Gerstenfeld ag@wpi.edu

Prof. Dougherty dd@cs.wpi.edu

Prof. Abraham jabraham@wpi.edu