IPRO PRESENTATION

- Design Group
- Software Group
- Hardware Group
Main Goal
• To create a system that can handle an input message rate of three million market ticks per second

Objectives
• Learn the Basics of the Industry
• Identify the Competitors and Their Systems
• Design an Alternate Solutions
• System Development
• System Optimization
• Experiment the Prototype to identify the Bottlenecks
• Provide Guidance for Future Work
Q: Why Fast Data?

A: Information Is Money!
Background

Need for Data Constantly Increasing

![Graph showing the increase in billions of messages per day in global options and equities markets from 2005 to 2010(e)],

Source: TABB Group Estimates
Background

• **Speed Limit: Speed of Light**

Example:

• 4320km: distance from Stanford to Boston
• 300 x 10^6 m/s: speed of light
• 200 x 10^6 m/s: speed of light through fiber

• One-way trip to Boston and back is at minimum 43.2 ms
Background

• Implementation of High-Speed Data Transmission in Different Industries

• Government
  – Simulations

• Life Science
  – Surgeries Over Internet

• Travel
  – Customer Notification

• Financial
  – Access Many Markets
  – Consolidate Market Data
  – Direct VPN and Web Access
  – Algorithmic Trading
Organization

- **Design Team**
  - IPRO Deliverables
  - Research
  - Coordination

- **Hardware Team**
  - Research
  - Experiment

- **Software Team**
  - Developing the Prototype
  - Testing Modules

Philip Pannenko, C.S. (Team Leader)
Devaraj Ramsamy, B.A.
Kenneth D. Buddell, B.A.

Michael Lenzen, AM (Sub Team Leader)
Jong Min Lim, ECE
Yunseok Song, ECE

Jong-Yon Kim, B.A. (Sub Team Leader)
Jesus Allan C. Tugade, C.S.
Jong Su Yoon, CS
Usman Jafarey, CS
Young Cho, AM
• Project plan
• Midterm report
• Abstract
• Presentation
• Project code
• Technical documentation
• Final Report
Generic Architecture

- **Feeds**
  - Options, Stocks, Futures

- **Feed Handler**
  - Translates incoming data to an internal format

- **Data Cache**
  - Stores data

- **Data Distributor**
  - Handles client connections and permissions

- **Clients**
## Categorization

<table>
<thead>
<tr>
<th>Company</th>
<th>Type</th>
<th>Areas of Optimization</th>
<th>Consolidator</th>
<th>Direct Feed Handler</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuters/RMDS</td>
<td></td>
<td>Network O</td>
<td>O</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>TA</td>
<td></td>
<td>S/W</td>
<td></td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Exegy</td>
<td></td>
<td>H/W</td>
<td>O</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>RTI</td>
<td></td>
<td></td>
<td>O</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>Wombat</td>
<td></td>
<td></td>
<td>O</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>Bloomberg</td>
<td></td>
<td></td>
<td>O</td>
<td></td>
<td>O</td>
</tr>
</tbody>
</table>
Architecture features

• Bloomberg
  – Terminal with direct server access

• Reuters
  – Dedicated network

• Wombat
  – Modular design
  – Feed handler split to line and message handlers

• Real Time Innovations
  – Feed Handler/Data Distributor direct connection
### Ideal Architecture

- **Direct connection between Feed Handler and Data Distributor**
  - Clients will receive updates as partial records
  - Remove Last Value Cache from chain of transmission
  - Reduces message size
  - Offloads work to client
Proposed Design

Data Distributor

Exchange 1
Exchange 2
Exchange 3

Updates

Last Value Cache

Subscription/Security

Archive

On Connection

Direct Update

Client

FH

TAL 4

TA 4

TA 4
## Prototype Design

### Current Status

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Bid</th>
<th>Ask</th>
<th>Ask</th>
<th>Ask</th>
<th>Last</th>
<th>Last</th>
<th>Client Sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOOG</td>
<td>XX</td>
<td>XX</td>
<td>XX</td>
<td>XX</td>
<td>XX</td>
<td>XX</td>
<td>1, 3</td>
</tr>
</tbody>
</table>

### Data Generator
- Simulate the exchanges
- Generates symbols and random updates
- Fixed size (57 bytes)

### Data Distributor / Last Value Cache
- Cache values into hash table
- Send updates to Clients

### Subscription
- Contains hash table with client list and subscribing symbol information
- Last value updates to client at initial login purpose

### Client List

<table>
<thead>
<tr>
<th>Client List</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client 1</td>
<td>ABB, AAB</td>
</tr>
<tr>
<td>Client 2</td>
<td>BBB, BAA</td>
</tr>
<tr>
<td>Client 3</td>
<td>ABB, CCC</td>
</tr>
</tbody>
</table>

### Diagram

- Data Generator
- UDP
- TCP
- Client List
- Subscription / Security
- Updates
- Client
Last Value Cache Optimization

- **Hashing Algorithm**
  
  - Hash table is a data structure used for efficient lookup (i.e. symbols)
  - Hashing algorithm
    - **Hash function**: Generates unique keys to indicate the address to be mapped in the container
    - **Hash container**: Actual table where the data is stored
  - Townsend Analytics
    - Hash function: CRC32
### Research

**Benchmarks on Hash Function**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Intel C++</th>
<th>MSVC</th>
<th>WATCOM C++</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC32</td>
<td>6.42</td>
<td>5.66</td>
<td>5.66</td>
<td>5.67</td>
</tr>
<tr>
<td>One at a Time</td>
<td>5.76</td>
<td>5.66</td>
<td>5.66</td>
<td>5.69</td>
</tr>
<tr>
<td>Alpha Numeric</td>
<td>3.29</td>
<td>4.06</td>
<td>4.06</td>
<td>5.67</td>
</tr>
<tr>
<td>FNV Hash</td>
<td>4.88</td>
<td>4.84</td>
<td>4.83</td>
<td>4.87</td>
</tr>
<tr>
<td>Bob Jenkins</td>
<td>2.08</td>
<td>2.36</td>
<td>2.03</td>
<td>2.07</td>
</tr>
<tr>
<td>SuperFast</td>
<td>1.54</td>
<td>1.92</td>
<td>1.92</td>
<td>1.34</td>
</tr>
</tbody>
</table>

Data is time in seconds taken to hash a random buffer of 256 bytes 5 million times.

Optimization

• Hash Container

• Different containers with same STL hash function
  – Sparse Hash (by Google)
    • Hash map store the data in a sparse table container
    • Memory efficiency in the expense of speed
  – Dense Hash (by Google)
    • Similar to sparse hash map.
    • Speed in the expense of memory.
Unit Testing

• Agenda
  – Performance testing
    • STL
    • Sparse
    • Dense
  – Experiment to find the behavior of different hashing algorithm
• Variables
• Varying character length (5, 7, 10 character)
• Varying hash table size (0.5M, 1M, 2M unique symbols)
  – Unique symbol = Unique characters ^ Character length
• Test Configuration
• Data Size: 57 bytes
• Input messages: 20 Million (Insert + Updates)
• Coded: C++ / Compiled: Window Visual Studio
Unit Testing

・STL Hash Algorithm

![Graph showing STL Hash and performance metrics]

By Character Length
5 Char > 10 Char > 7 Char

By Table Size
0.5 M > 1.0 M

<table>
<thead>
<tr>
<th>Number of Characters</th>
<th>Unique symbols</th>
<th>Memory used</th>
<th>Performance (Msg/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 char</td>
<td>537,824 (=14^5)</td>
<td>67 Mb</td>
<td>671,600</td>
</tr>
<tr>
<td>5 char</td>
<td>1,048,576 (=16^5)</td>
<td>129 Mb</td>
<td>585,771</td>
</tr>
<tr>
<td>7 char</td>
<td>2,097,152 (=8^7)</td>
<td>257 Mb</td>
<td>475,649</td>
</tr>
<tr>
<td>10 char</td>
<td>1,048,576 (=4^10)</td>
<td>129 Mb</td>
<td>482,512</td>
</tr>
</tbody>
</table>

※ Performance is measured starting from the critical point
Unit Testing

- Sparse Hashing Algorithm

![Graph showing performance metrics for different character lengths and table sizes.](image)

### By Character Length
- 5 Char > 7 Char > 10 Char

### By Table Size
- 0.5 M = 1.0 M

<table>
<thead>
<tr>
<th>Number of Characters</th>
<th>Unique symbols</th>
<th>Memory used</th>
<th>Performance (Msg/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 char</td>
<td>537,824 (=14^5)</td>
<td>56 Mb</td>
<td>1,012,201</td>
</tr>
<tr>
<td>5 char</td>
<td>1,048,576 (=16^5)</td>
<td>110 Mb</td>
<td>1,003,827</td>
</tr>
<tr>
<td>7 char</td>
<td>2,097,152 (=8^7)</td>
<td>218 Mb</td>
<td>872,994</td>
</tr>
<tr>
<td>10 char</td>
<td>1,048,576 (=4^10)</td>
<td>109 Mb</td>
<td>793,005</td>
</tr>
</tbody>
</table>

※ Performance is measured starting from the critical point
Unit Testing

• Dense Hashing Algorithm

By Character Length
5 Char > 7 Char > 10 Char

By Table Size
0.5 M = 1.0 M

<table>
<thead>
<tr>
<th>Number of Characters</th>
<th>Unique symbols</th>
<th>Memory used</th>
<th>Performance (Msg/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 char</td>
<td>537,824 (=14^5)</td>
<td>101 Mb</td>
<td>1,412,862</td>
</tr>
<tr>
<td>5 char</td>
<td>1,048,576 (=16^5)</td>
<td>136 Mb</td>
<td>1,404,268</td>
</tr>
<tr>
<td>7 char</td>
<td>2,097,152 (=8^7)</td>
<td>273 Mb</td>
<td>1,192,174</td>
</tr>
<tr>
<td>10 char</td>
<td>1,048,576 (=4^10)</td>
<td>137 Mb</td>
<td>1,013,039</td>
</tr>
</tbody>
</table>

※ Performance is measured starting from the critical point
Unit Testing

- **Results**
  - Performance: Dense > Sparse > STL
    - Dense over twice as fast as STL
  - Memory usage: Sparse > STL > Dense

![Graph showing performance comparison between STL, Sparse, and Dense for different unique character counts and symbol lengths. The x-axis represents the unique character count and symbol length, and the y-axis represents the message per second. The bars indicate the performance of STL, Sparse, and Dense.]
System Testing

• Agenda
  – To find the throughput of the system (Last Value Cache)
  – Performance testing on the system varying the hashing algorithm
  – To find the bottleneck of the system through experimental approach

• Hypothesis
  – Last Value Cache is the bottleneck of the system
System Testing

- Testing Configuration
  - All the components are connected using network
  - Varying hash algorithm (STL, Sparse, Dense)
  - Data Size: 57 bytes
  - 1 Million Unique Messages
### Testing specification

<table>
<thead>
<tr>
<th>Machine 1 - 3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>Intel® Core 2 CPU @ 2.40 GHz</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>2.00 GB</td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
<td>Window XP SP2</td>
</tr>
<tr>
<td><strong>Network Card</strong></td>
<td>Broadcom NetXtreme 57xx Gigabit Controller</td>
</tr>
<tr>
<td><strong>Network Equipment (Router)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Belkin F5D8230-4 Wireless 802.11x Pre-N Router</strong> (Maximum throughput of 108 Mbps)</td>
<td></td>
</tr>
<tr>
<td><strong>UTP cable 100 base T</strong></td>
<td></td>
</tr>
</tbody>
</table>
# Test Results

## 1 Data Generator

<table>
<thead>
<tr>
<th>Hash Algorithm</th>
<th>Data Generator (msg/sec)</th>
<th>Last Value Cache (msg/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STL</td>
<td>82,000</td>
<td>82,000</td>
</tr>
<tr>
<td>Sparse</td>
<td>82,000</td>
<td>82,000</td>
</tr>
<tr>
<td>Dense</td>
<td>82,000</td>
<td>82,000</td>
</tr>
</tbody>
</table>

## 2 Data Generators

<table>
<thead>
<tr>
<th>Hash Algorithm</th>
<th>Data Generator (msg/sec)</th>
<th>Last Value Cache (msg/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STL</td>
<td>120,000</td>
<td>109,000</td>
</tr>
<tr>
<td>Sparse</td>
<td>120,000</td>
<td>109,000</td>
</tr>
<tr>
<td>Dense</td>
<td>120,000</td>
<td>109,000</td>
</tr>
</tbody>
</table>

## Remarks
- Performance measurement is rounded off average message per second
- No improvements in messages per second when 3 data generators were used
Analysis

• Bandwidth of the network equipment (Router and UTP cable 100 base T) is limited to 100Mbps

  – A Message contains
    • Payload : 57 bytes
    • UDP Overhead : 8 bytes
    • IPv4 Overhead : 20 bytes
    • Total Data Size : 85 bytes

  – Max Possible Throughput: 120,000 messages/sec

• Need to find the ways to Bandwidth of the network equipment (Router and UTP cable) is limited to 100Mbps
Other Technologies

- Playstation 3
  - Cell Architecture
- FPGAs
- Infiniband
Obstacles

• Lack of programmers and programming experience
• Hardware Constraints
• Network Constraints
• Time Constraints
Future Works

- Use network equipments with higher bandwidth
- Find other ways to get around the network limit
- Incorporate efficient hash functions and provide experimental benchmark (i.e., SuperfastHash, CRC32)
- Extend the system to implement the ideal architecture
- Optimization (Multi-threading)
Conclusions

- Created a platform for future works
- Better understanding of how to optimize system
- Built a prototype system that is easily extensible for optimization
Any Questions?