[IPRO313]Ultra-High-Speed Market Data System

IPRO PRESENTATION

- Design Group
- Software Group
- Hardware Group

Townsend analytics

ILLINOIS INSTITUTE OF TECHNOLOGY

November 30, 2007

Objectives

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

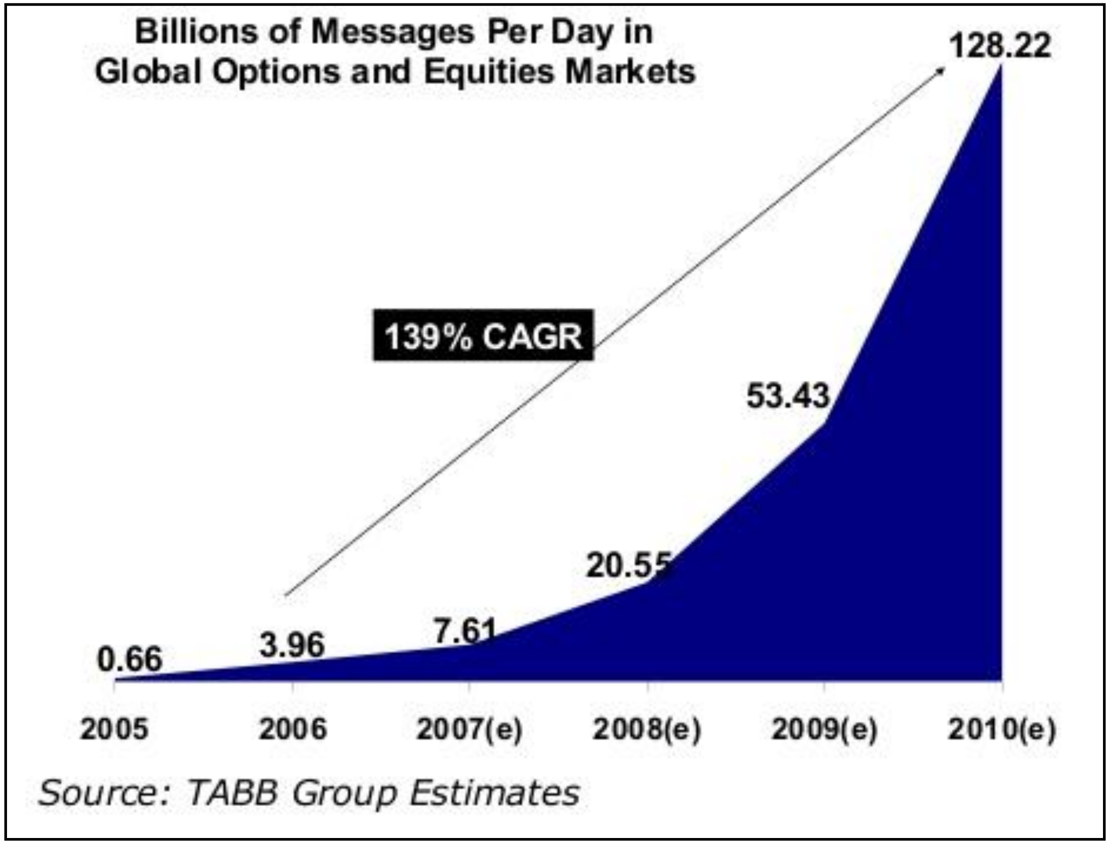
Background

Q: Why Fast Data?

A: Information Is Money!

Background

Need for Data Constantly Increasing



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

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

- **Hardware Team**
  - Research
  - Experiment

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

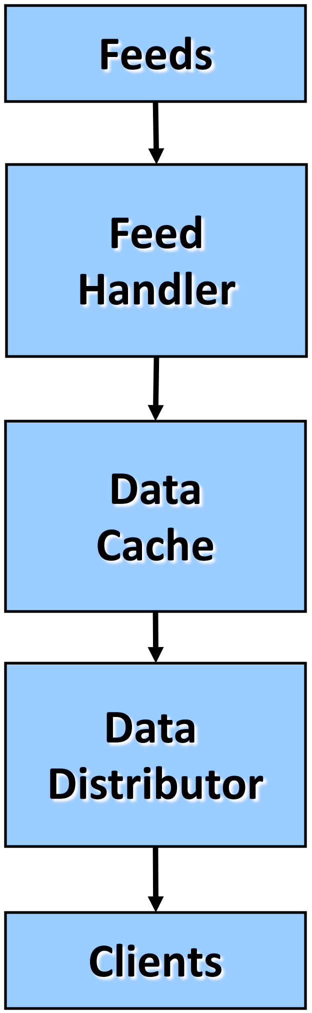
- **Software Team**
  - Developing the Prototype
  - Testing Modules

Jong-Yon Kim, B.A. (Sub Team Leader)
Jesus Allan C. Tugade, C.S.
Jong Su Yoon, CS
Usman Jafarey, CS
Young Cho, AM

IPRO Deliverables

- Project plan
- Midterm report
- Abstract
- Presentation
- Project code
- Technical documentation
- Final Report

Generic Architecture



- Options, Stocks, Futures
- Translates incoming data to an internal format
- Stores data
- Handles client connections and permissions

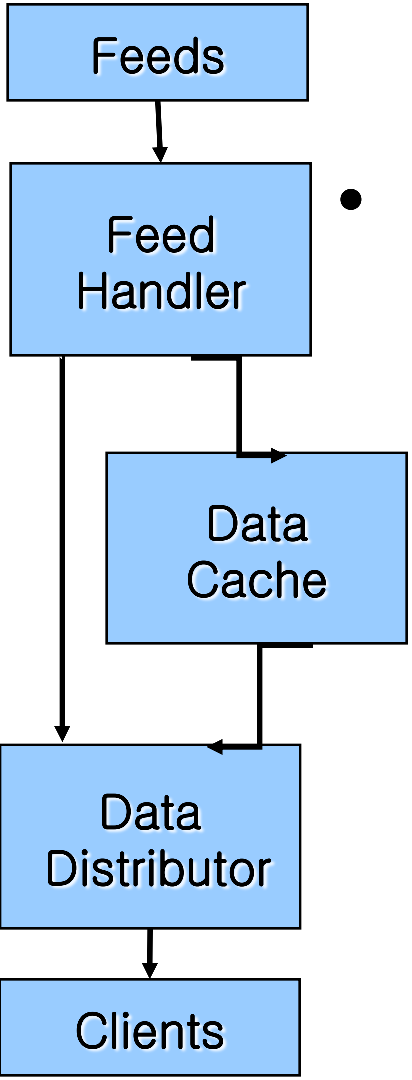
Categorization

Company \ Type	Areas of Optimization			Consolidator	Direct Feed Handler	Application
	Network	S/W	H/W			
Reuters/RMDS	O			O		O
TA				O		
Exegy			O		O	
RTI	O			O		
Wombat		O			O	
Bloomberg	O			O		O

Architecture features

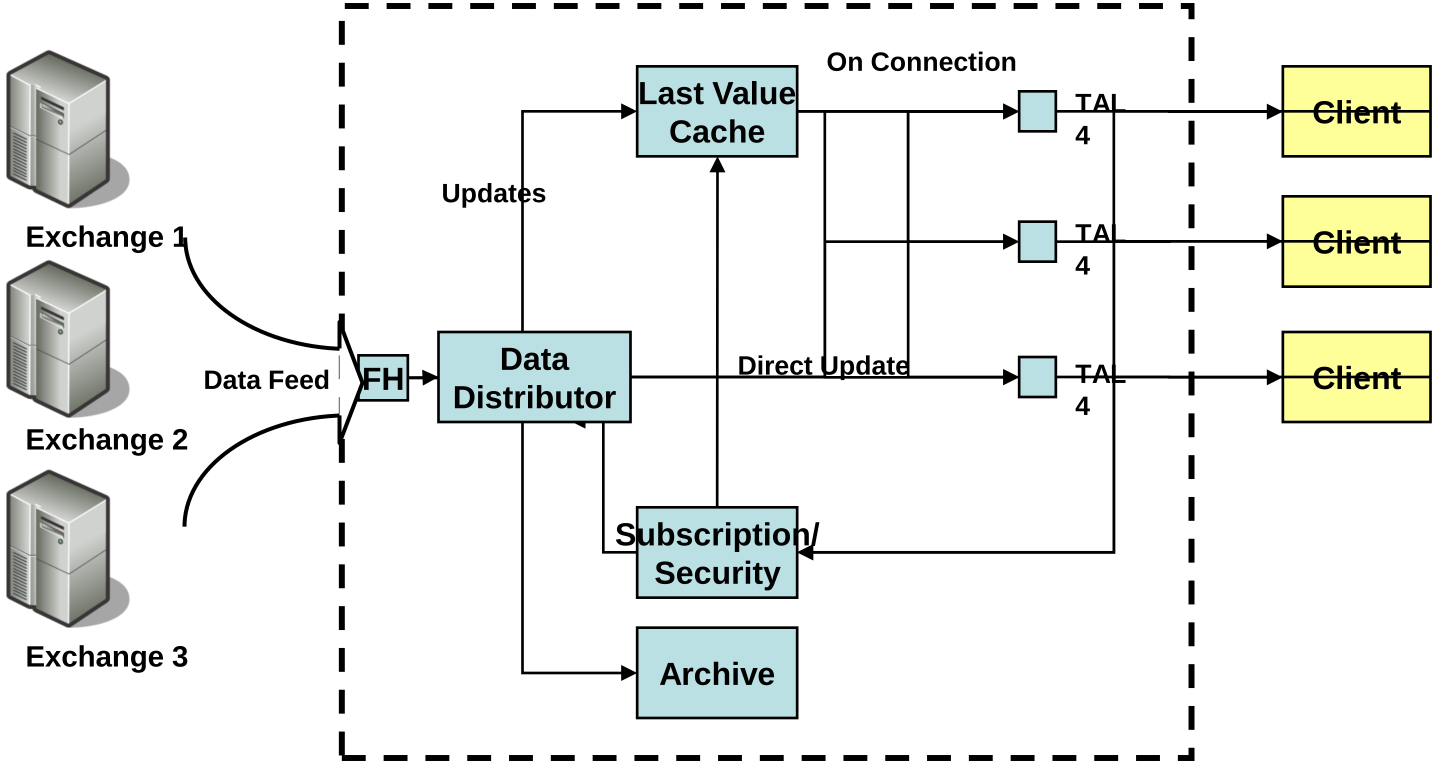
- **Bloomberg**
  - Terminal with direct server access
- **Reuters**
  - Dedicated network
- **Wombat**
  - Modular design
  - Feed handler split to line and message handlers
- **Real Time Innovations**
  - Feed Handler/Data Distributor direct connection

Ideal Architecture



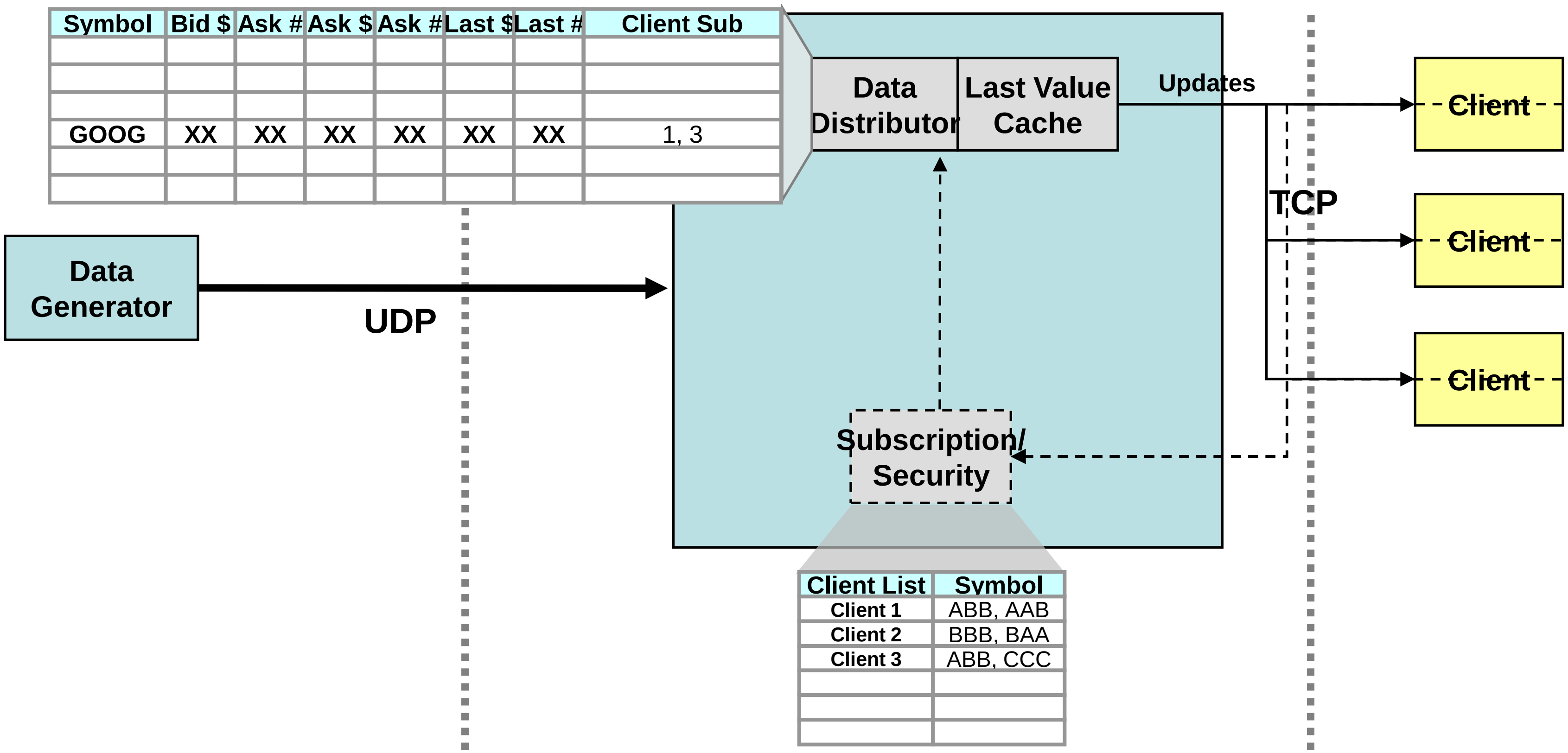
- **Direct connection between Feed Handler and Data Distributor**
  - Clients will receive updates as partial records
  - Remove Last Value Cache from chain of transmission
  - Reduces message size
  - Offloads work to client

Proposed Design



Prototype Design

• Current Status



Data Generator

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

Last Value Cache /Data Distributor

- 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

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**

	AMD Athlon XP 1.620Ghz			
	Intel C++	MSVC	WATCOM C++	GCC
CRC32	6.42	5.66	5.66	5.67
One at a Time	5.76	5.66	5.66	5.69
Alpha Numeric	3.29	4.06	4.06	5.67
FNV Hash	4.88	4.84	4.83	4.87
Bob Jenkins	2.08	2.36	2.03	2.07
SuperFast	1.54	1.92	1.92	1.34

http://www.azillionmonkeys.com/qed/hash.html

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

Optimization

• Hash Container

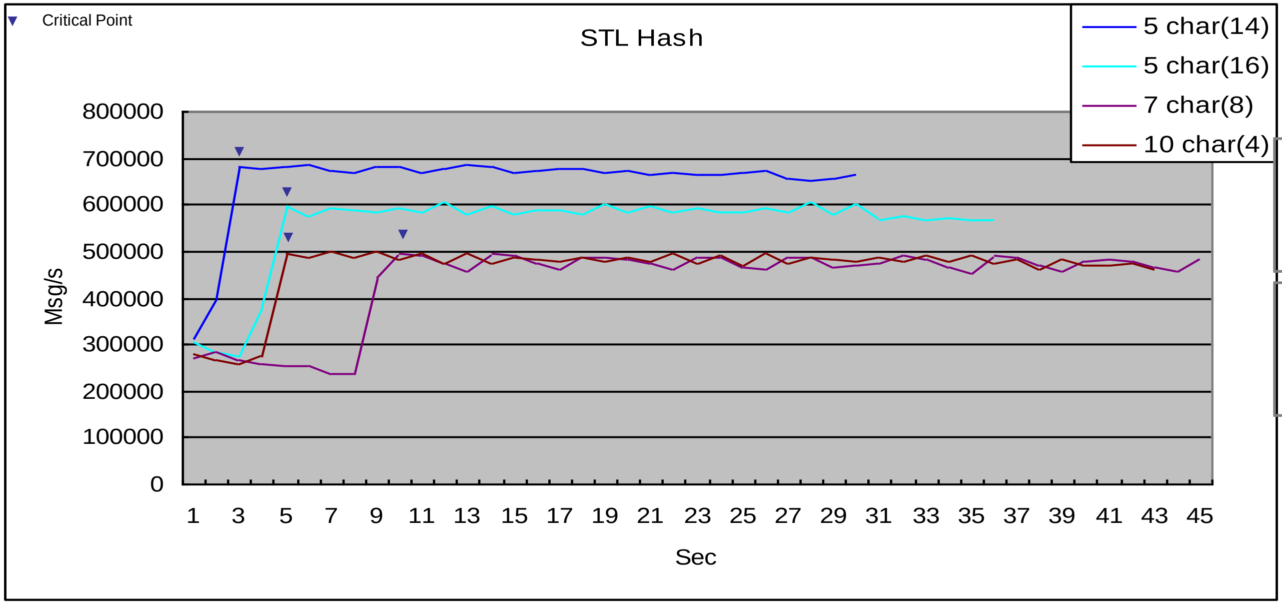
- **Different containers with same STL hash function**
  - **Sparse Hash (by Google)**
    - Hash map store the data in a sparse table container
    - Memory efficiency in the expense of speed
  - **Dense Hash (by Google)**
    - Similar to sparse hash map.
    - Speed in the expense of memory.

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 Configuartion
- Data Size : 57bytes
- Input messages : 20 Million (Insert + Updates)
- Coded : C++ / Compiled : Window Visual Studio

Unit Testing

- **STL Hash Algorithm**



By Character Length

5 Char > 10 Char > 7 Char

By Table Size

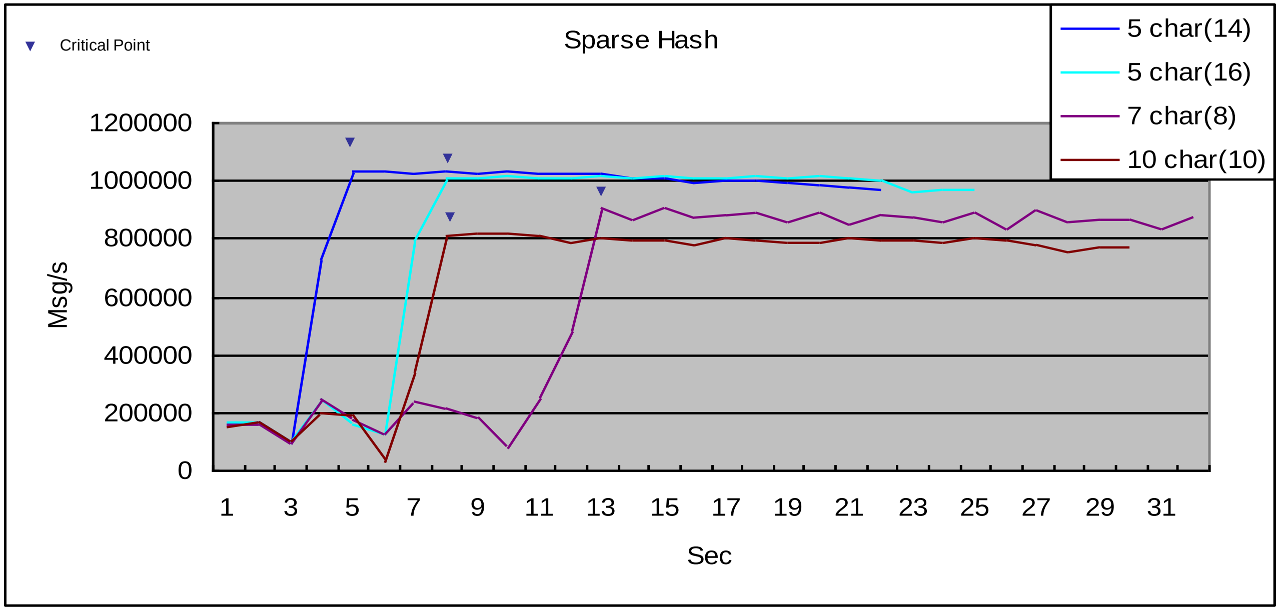
0.5 M > 1.0 M

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

※ *Performance is measured starting from the critical point*

Unit Testing

- **Sparse Hashing Algorithm**



By Character Length

5 Char > 7 Char >10 Char

By Table Size

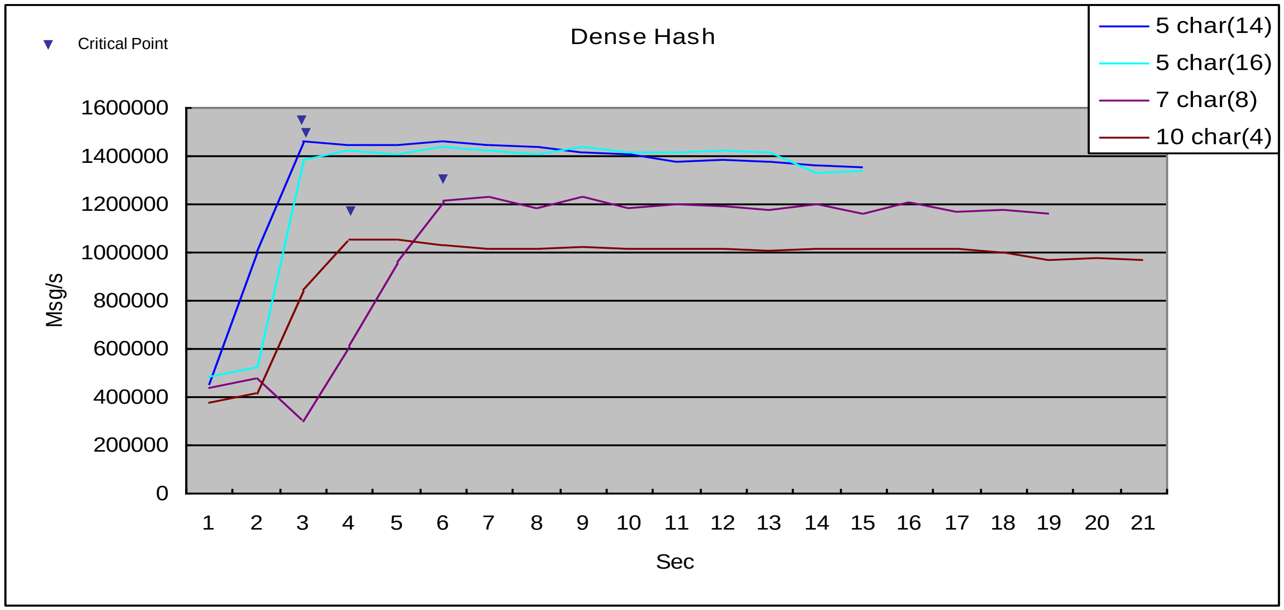
0.5 M = 1.0 M

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

※ *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

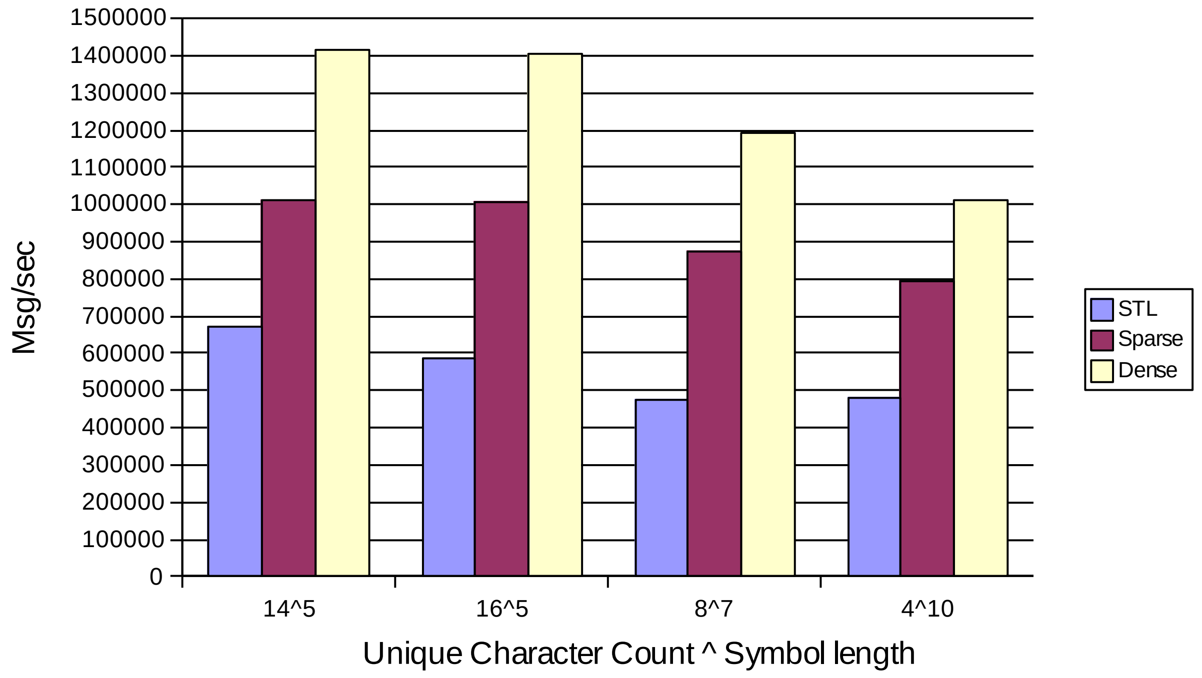
Number of Characters	Unique symbols	Memory used	Performance (Msg/s)
5 char	537,824 (=14^5)	101 Mb	1,412,862
5 char	1,048,576 (=16^5)	136 Mb	1,404,268
7 char	2,097,152 (=8^7)	273 Mb	1,192,174
10 char	1,048,576 (=4^10)	137 Mb	1,013,039

※ *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

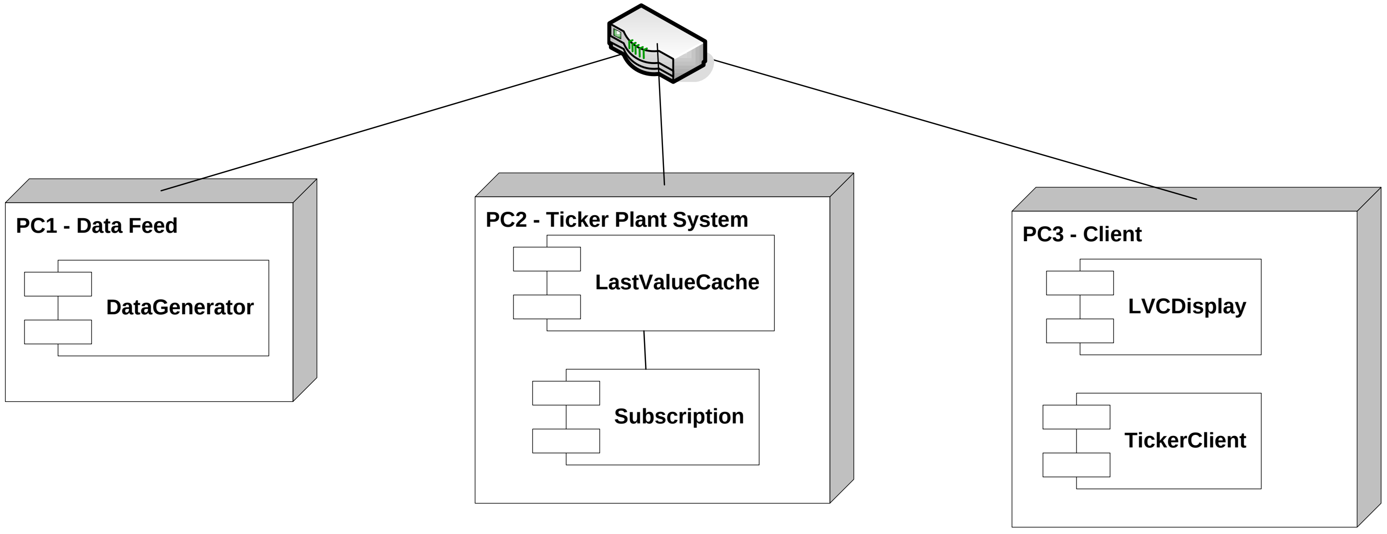
System Testing

- **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 : 57bytes
  - 1 Million Unique Messages



Test Specifications

• **Testing specification**

Machine 1 - 3	
CPU	Intel® Core 2 CPU @ 2.40 GHz
Memory	2.00 GB
Operating System	Window XP SP2
Network Card	Broadcom NetXtreme 57xx Gigabit Controller
Network Equipment (Router)	
Belkin F5D8230-4 Wireless 802.11x Pre-N Router (Maximum throughput of 108 Mbps)	
UTP cable 100 base T	

Test Results

1 Data Generator

Hash Algorithm	Data Generator (msg/sec)	Last Value Cache (msg/sec)
STL	82,000	82,000
Sparse	82,000	82,000
Dense	82,000	82,000

2 Data Generators

Hash Algorithm	Data Generator (msg/sec)	Last Value Cache (msg/sec)
STL	120,000	109,000
Sparse	120,000	109,000
Dense	120,000	109,000

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 UPT 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**

Q & A

Any Questions?