# DBToaster Runtime: System for Streaming Data Processing.

July 14, 2009

## 1 Introduction

DBToaster application contains three major components. Data generation component is a source of the incoming data to the system. In our application it will be Exchange Simulation Server. Next component is Algorithms Simulation Engine. This component is responsible for the decision making and outside influence. In out example it is a Trading Runtime Platform. The last component is DBToaster Runtime. The component is responsible for collecting the data from various sources unifying it, dispatching it to appropriate queries and forwarding the query results to Algorithms Simulation Engine.

The three components interact in the following way. Once started data generation component generates data, these data comes from a variety of sources stock exchange, sensor network, remote applications etc. The data comes as a continuous stream of information, this information is presented in a well known format. These data is sent to DBToaster runtime. The runtime is responsible for collecting data from data sources and feeding it a dynamic set of user defined queries. These set of queries is dynamically loaded/compiled and removed during the execution of the runtime. The results of the queries are then forwarded to Algorithms Simulation Engine. The Engine's job is to load and run algorithms. Engine receives the data from the DBRuntime based on this information each algorithm makes a decision this decision is forwarded onwards. These forwarded information can be in various forms; for instance, buy/sell orders for the stock exchange or a commands to switch street lights to alleviate traffic congestion.

The particular application we have chosen to demonstrate the capabilities and performance of our system is a NASDAQ trading simulation. In this simulation there is a a set of servers; each server arranges stock exchanges

between different trading parties. The results of those exchanges as well as information about interest in stock sells and purchases is transmitted to all parties interested in such information (for a fee of course). Based on these information different trading algorithms decide to buy and sell stocks.

The arrangement of all the components in the system can be seen in figure 1.

# 2 DBToaster Runtime

DBToaster Runtime consists of three major components: Compiler, Query Processing and Data Stream Processing. Each of these components needs to interact with the outside world.

## 2.1 Compiler

Compiler is subdivided into two parts: SQL query pre-compiler and C++ code compiler. Compiler interacts with the user by user making requests to add a SQL query to the Runtime or by user adding an additional data stream to the input. Compiler Structure can be seen in figure 2

### 2.1.1 SQL Compiler

SQL complier takes an SQL query from the user and creates a c++ coded to be added to the runtime.

### 2.1.2 C++ Compiler

C++ compiler takes either C++ code of an SQL query, compiles it and adds it to Query Processing or it take a C++ code for the data stream processor compiles it and adds it to the List of Data Listeners.

## 2.2 Query Processing

Query Processing is done with a set of Query Threads each thread picks a query-datum pair from a pull of of available pairs and processes it. As data comes in after been processed it is sent to the appropriate query. As soon as the query receives a datum it becomes available for execution. The structure for query processing can be found in figure 3.

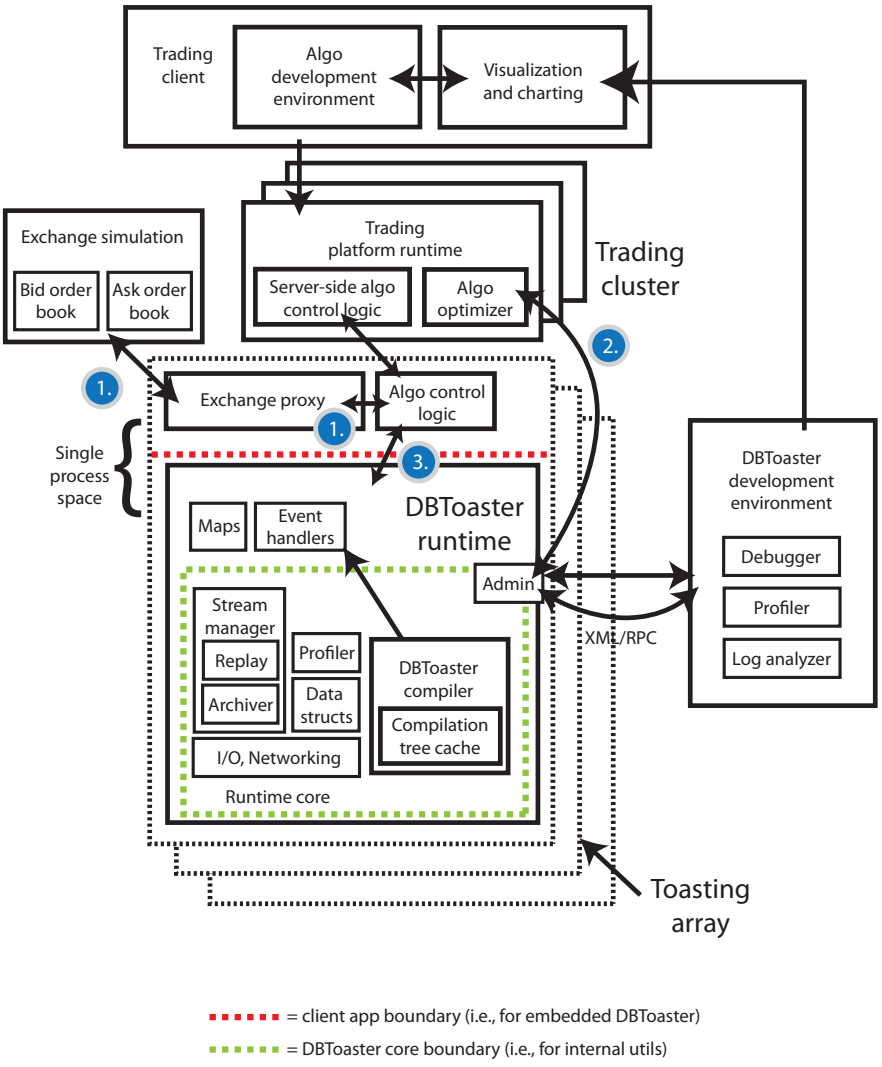


Figure 1: Application Overview.

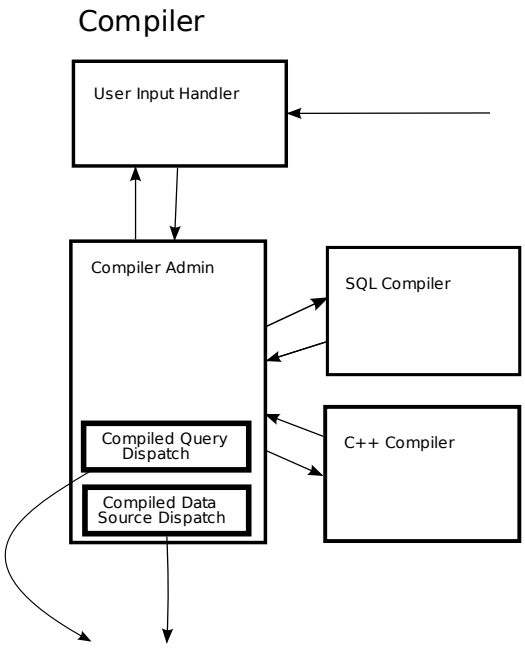


Figure 2: Application Overview.

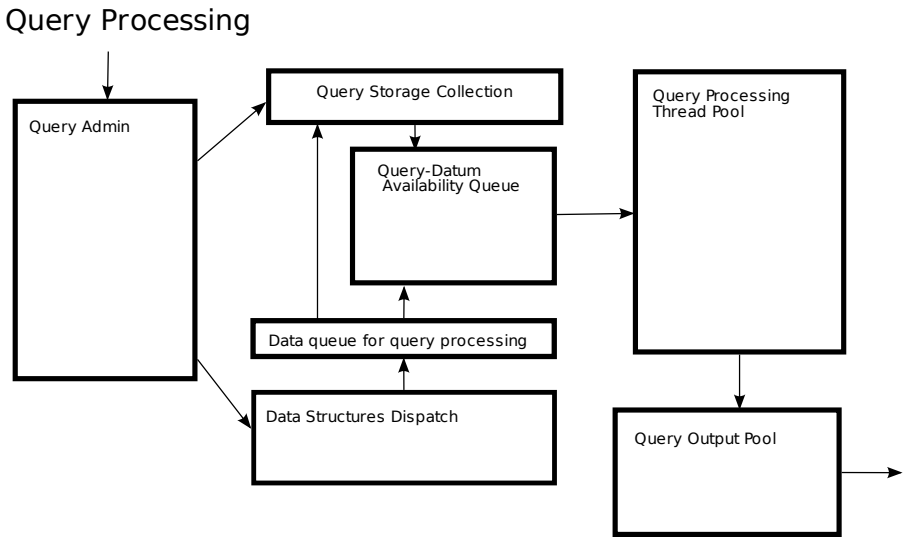


Figure 3: Application Overview.

### 2.2.1 Communication

The results of the query processing are pushed to the The transfer hub where they are stored until they are requested by the client.

## 2.3 Data Source Processing

Date Stream processing consists of the list of list of stream points. Each point is given by the user and signifies a way for a DBToaster to connect with the input stream of some process which dispatches data to the DBToaster. Each Stream Point is written by the user and user can fully specify how the data should be received and processed. Stream point is responsible for sending data to the Data Dispatch unite in a pre-specified format. The structure for data source processing can be found in figure 4.

# 3 Data Generation

Dada can come in a variety of forms. For our particular application we have developed an Exchange Simulation Server. The server is responsible for processing and broadcast of all stock exchanges to the clients that are connected to it. Server can also is capable to augment data flow by adding data from files containing historical traces of real stock exchanges. At the moment the Exchange Server designed to simulate trade exchanges for one stock.

## 3.1 Exchange Server Simulator

ExchangeServer creates server port and handles all incoming client connections. Each connected client is given it's own thread (ExchangeThread). ExchangeServer also creates a DataThread, which is responsible for reading historic trace data. The server also creates structure for datastorage which is shared by all of the clients.

## 3.2 Exchange Thread

On a client connection Server creates an ExchangesThread to deal with client needs. All ExchangesThreads share data storage structure DataBook as well as a list of all currently active clients. On a start up thread expects to receive a message from a client indicating the type of a client.

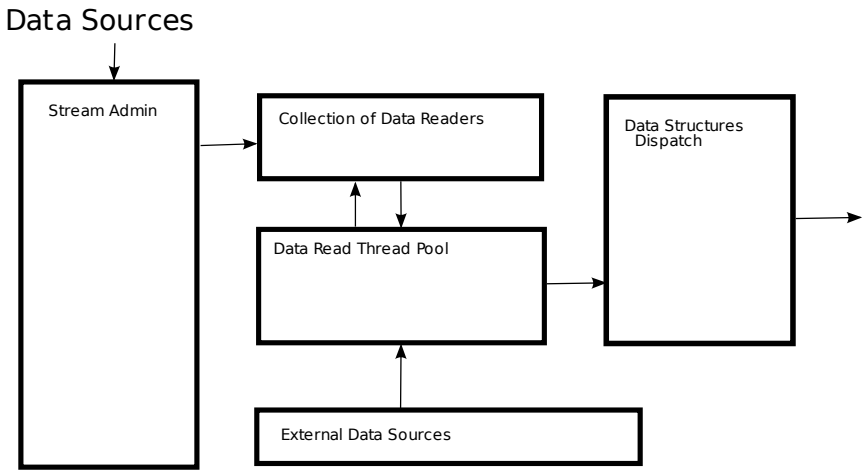


Figure 4: Application Overview.

**Code 1** ExchangeServer: inputDataFile.cvs

```
Server_socket;
SynchronizedBooks DataBook;
Data_socket(inputDataFile.cvs);

while (Server_socket.listen)
{
get(client);
run(client, DataBook);
}
```

| Client type | |
|---|---|
| 0 | Interactive |
| 1 | Passive Listener |

*Interactive* clients send data and are only interested in receiving messages in response to their transactions. *Passive Listener* clients are interested in all of the transactions but tend not to send the messages themselves.

The messages from clients are coming in the pre-specified format. Depending on the client's request several actions occur:

| Request | Action |
|---|---|
| 'B' | buy request, check ask book for match if no add to bid book |
| 'S' | see request, check bid book for match if no add to ask book |
| 'D' | delete request from appropriate book if one exists |
| 'X' | cancel order remove trade request from appropriate book |

Each transaction is announced to all Passive Listeners as well as to Interactive client which intiated the transaction.

### 3.3 Data Storage

SynchronizedBooks is a shared data structure. It is shared between all of the clients. Each client can modify it by adding/removing buy/sell requests. The data structure consists of two SortedBooks. Each SortedBook is a structure that has properties of a sorted set and a hashtable.

When an addition of a sell/buy request is invoked data structure tries to find a matching buy/sell request and if found the update message is sent to the clients additionally to the original sell/buy request. The update mes-

sage contains information about the fact that one or both of the orders are partially/fully satisfied. The exchange message is of the form:

| Request | Action |
|---|---|
| 'E' | order number, update amount |
| 'F' | order was executed in full |

### 3.4 Data Generation

ExchangeServer is capable of augmenting the data flow between clients with the additional data from historical data files. To accomplish this goal server has a DataThread. It can be initiated at any point by a server (for example, once there is a client connected to the server). DataThread reads a historical trace file in a .cvs format and transmits each message to the server as if it was a real request.

## 4 Algorithms Engine

Algorithm Engine code developed by the end users. The code makes use of DBToaster Runtime to query the continuous data from input streams specified by the users. The query results are then processed by user defined algorithms to achieve their goals.

In our example Algorithmic Engine creates and runs a set of monitoring queries for the stock trading depending on the results of the queries and the internal mechanics - algorithms decide to buy or sell stocks on a NASDAQ Trading Exchange Simulator.

### 4.1 Data Sources

Algorithmic Engine interacts with environment in three different ways. First of all the Engine needs to receive the information from from DBToaster runtime. This interaction is unavoidable since the Engine needs the results of the queries. Another interaction which needs to be added is the way to specify the information stream received by a DBToaster runtime. This information needs to be passed the the Runtime in the form of c++ code for the Runtime to compile. Another optional source of interaction is infulencing the enviroment. In our example is the Engine's capability to add/remove stock requests from the Exchange Server.

### 4.1.1 Data Stream

There are two interactions with outside systems needed to be implemented by the user. The first interaction is with DBToaster. A system needs to pull information from the DBToaster runtime. *TODO description of the the pulling mechanism.*

Another interaction, which needs to be passed to the DBToaster Runtime, is a way for Runtime to access the data streams relevant to the Engine and over which the queries needed to be post. This information is specified as a C++ code which is sent for the compilation to the Runtime to be added to the system.

### 4.1.2 External Influence

If the Engine implements interactive process, which needs to access outside processes or interact (influence) them then this is interactivity needs also to be added. In our example Algorithmic Engine can buy and sell stock on our Exchange Simulation Server. Thus acting like one of the clients of the server. *TODO describe the system in more details*

## 4.2 Algorithm

This is the part which is the most relevant to the user and will vary from system to system. In out example we encoded the learning and trading strategies for NASDAQ stock exchange. *TODO describe the system in more details*

## 4.3 Query creator (or something like that)

Queries posed to the DBRuntime can come from various sources they can either be predefined by the user system or written and modified dynamically. In order to be active they need to be passed to the Runtime where they are compiled and added to stream observations. *TODO describe the process more*