HFT Simulation System
Part I: Overview & Tutorial

Elaine Wah
ewah@umich.edu

February 2013

1 Introduction

The HFT simulation system employs agent-based modeling and discrete-event simulation to model communication latencies and current U.S. securities regulations.

2 Overview

The simulation system consists of:

- Market models: Each model specifies the number, type of, and configuration parameters of markets present. The HFT simulation system allows the simulation of multiple market configurations in parallel.

  For example, the `TwoMarket` model specifies two Continuous Double Auction (CDA) markets, and the `CentralCall` model specifies a single call market that clears at the same frequency as the NBBO update latency.

- Markets: Two types of markets are implemented in the system, a CDA market and a call market (which matches orders at regular, fixed intervals).

- Agents: There are two general types of agents in the system, which are distinguished by whether or not they have direct access to more than one market. Note that agents are duplicated (with common random number seeds) in each market model, but they execute their strategy within each model independently.

  - Single-market agents (`SMAgent`): These agents have direct, undelayed access to only a single market (which is specified at agent creation). Example: `ZIAgent`.
  - High-frequency trading agents (`HFTAgent`): These agents have access to multiple markets (all markets within a given market model), which models the direct feeds that many HFTs have to exchanges. Example: `LAAgent`.

ACTIVITY	DESCRIPTION
AgentArrival	agent enters a market (or markets)
AgentStrategy	agent executes its trading strategy
Clear	market clears any matching orders
Liquidate	agent liquidates any net position
ProcessQuote	SIP updates its best market quotes
SendToSIP	market sends a new quote to the SIP
SubmitBid	agent submits single-point bid to a market
SubmitMultipleBid	agent submits multi-point bid to a market
SubmitNMSBid	agent submits single-point bid, routed for best execution
UpdateAllQuotes	agent updates its quotes for its market(s)
UpdateNBBO	SIP computes and publishes updated NBBO quote
WithdrawBid	agent withdraws its bid from a market

Table 1: Partial list of activities in the simulation system.

3 Discrete-event simulation

In our system, we employ *discrete-event simulation*, a paradigm that allows the precise specification of event occurrences. It is particularly effective for modeling current U.S. securities regulation, specifically Regulation NMS—which led to the creation of the Security Information Processor (SIP) and which mandates the routing of orders for best execution. Components of the simulation system include:

- **Entity**: Objects present in the simulation system, e.g., traders, markets, and the SIP.
- **Activity**: Actions that entities can execute.
- **Event**: A sequence of activities, ordered by priority. If there is a tie, activities are inserted in order of arrival. Priorities are assigned based on activity type (e.g., bid submission, market clearing).
- **Event Queue**: Priority queue ordered by event time; executed sequentially until empty. Multiple events may occur during the same time step, but they are executed in the order in which they are enqueued.

To summarize, an *event* consists of a sequence of *activities* that are to be executed by various *entities* (traders, markets, and the SIP).

3.1 Activities

Each activity has a timestamp and each is associated with at least one entity present in the simulation system. Note that activities are chained (the next immediate activity is inserted at the end of the current one). See Table 1 for a partial list of activities within the system.

3.2 Example

To control the latency of the SIP, we specify three activities: `SendToSIP`, `ProcessQuote`, and `UpdateNBBO`. The `SendToSIP` activity is inserted when a market publishes a quote at

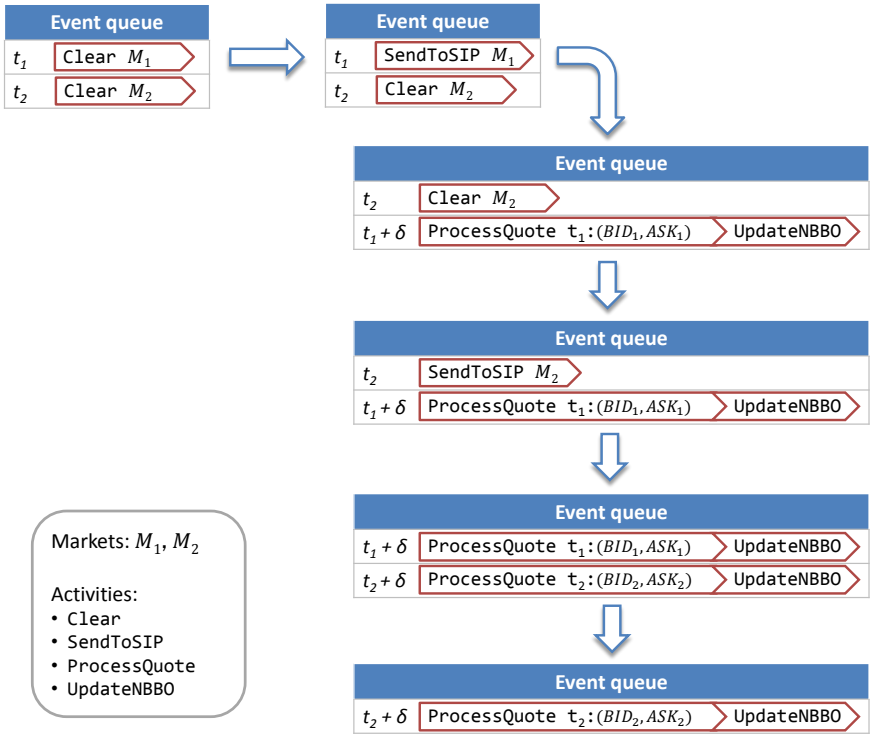

Figure 1: Event queue during the dissemination and processing of updated market quotes for NBBO computation, given latency $\delta > t_2 - t_1$. There are two markets, M_1 and M_2. When the NBBO update activity executes at time $t_1 + \delta$, the SIP has just processed the best quote (BID_1, ASK_1) at time t_1 from market 1; this is therefore the most up-to-date information that could be reflected in the NBBO at time $t_1 + \delta$.

time t; upon execution the market sends its updated quote to the SIP entity and inserts a `ProcessQuote` and a `UpdateNBBO` activity, both to execute at time $t + \delta$ in the future. When `ProcessQuote` is executed, the SIP updates its stored information on the best market quotes. It computes and publishes an updated NBBO based on this information during the execution of the `UpdateNBBO` activity.

Figure 1 illustrates how the activities are sequenced in our simulation system to reflect the communication latencies arising as a consequence of market fragmentation. Market 1 clears and publishes an updated quote at time t_1. Market 2 publishes its new quote at time t_2. For $\delta > t_2 - t_1$, a `ProcessQuote` followed by an `UpdateNBBO` activity are executed sequentially at $t_1 + \delta$, as well as at $t_2 + \delta$. The `UpdateNBBO` executing at $t_1 + \delta$ will not incorporate market 2's updated quote, as the `ProcessQuote` activity to add market 2's best quote (BID_2, ASK_2) will not be executed until $t_2 + \delta$. This process serves to approximate the behavior of the SIP with a delay of δ.

4 Simulation specification file

The parameters in the specification file are those that can be adjusted in simulations. Below is an example `simulation_spec.json` file that specifies a simulation configuration:

```
{
  "assignment": {
    "LA": ["sleepTime_0_alpha_0.001"],
    "DUMMY": [""]
  },
  "configuration": {
    "sim_length": "15000",
    "tick_size": "1",
    "primary_model": "TWOMARKET-LA",
    "TWOMARKET": "LA,DUMMY",
    "CENTRALCDA": "1",
    "CENTRALCALL": "NBBO",
    "MARKETMAKER": "0",
    "ZI": "25",
    "ZIP": "0",
    "nbbo_latency": "0",
    "arrival_rate": "0.075",
    "mean_value": "100000",
    "kappa": "0.05",
    "shock_var": "150000000",
    "bid_range": "2000",
    "private_value_var": "100000000",
    "marketmaker_sleep_time": "200",
    "marketmaker_num_rungs": "10",
    "marketmaker_rung_size": "1000"
  }
}
```

An explanation of select parameters follows:

- `assignment`: Assignment of strategies to players. Relevant primarily for EGTA experiments. If there are multiple market models in the simulation, then all players present in *any* model must have a strategy assigned in this section.
- `configuration`: Specify simulation configuration.
- `sim_length`: Length of simulation (in milliseconds).
- `tick_size`: Tick size. Prices are integers, so the smallest tick size is 1.
- `primary_model`: For EGTA purposes only. Specifies the market model that will have its players' payoff recorded separately.
- `[MARKETMODEL]`: Specifies number and configuration of market models.

  - Each model may have various configurations specifying, for example, the agents allowed in that instance of the model.
  - Each set of configurations is a comma-separated string in the specification file.
  - `"MARKETMODEL": "A,B"` would indicate that for the given model, there is one instance of configuration A and one instance of configuration B. The system, in this case, would create two instances of this model (with differing configurations).
  - For example, `"TWOMARKET": "LA,DUMMY"` creates a two-market model with LA and a two-market model without LA.

- `[SMAGENT TYPE]`: Specifies total number of each type of single-market (`SMAgent`) within a given market model. If there are multiple markets, the system will distribute these background agents as evenly as possible amongst all markets in the model.

5 Example: Running a simulation

To run a basic simulation, first compile the code (run `ant` in the base directory that contains the `build.xml` file. Then use the following command:

```
./example.sh <HFT type> <NBBO latency> <number of samples> <CSV>
```

For example:

```
./example.sh LA 0 10 test.csv
```

All simulations should be saved within subfolders in the `simulations` directory. In the previous example, the `example.sh` script will generate a folder with path `simulations\LA_latency_0`. The resulting simulations will be parsed and saved in a CSV file in the base directory.