HFT Simulation System
Part II: Logging

Elaine Wah
ewah@umich.edu

February 2013

All log files are saved in the folder `logs` that is created in the directory that contains the `simulation_spec.json` file. Logs (.txt files) are named according the following convention: There are four sections in the filename, each separated by underscores (examples refer to the filename `simulations-test_2_12-Feb-2013_13.45.00.txt`):

(1) *Directory or path* with the spec file, with forward slashes replaced by hyphens (e.g., the base directory is `simulations/test`)

(2) *Observation number* (e.g., here it is observation #2)

(3) *Date* (dd-MMM-YYYY format)

(4) *Time* (hh.mm.ss format, 24-hour clock, e.g., time here is 1:45:00 PM)

Within the log files, the first 18 characters on each line can be ignored, as they are just generated by the logging utility. There are three parts for the log file, which are described in depth in the following sections. Some basic aspects to be aware of:

- Model IDs are within curly brackets, e.g. `{1}`.
- Market IDs are negative, and are within square brackets, e.g., `[-1]`.
- Agent IDs are positive, and are within parentheses, e.g., `(1)`; each agent has a *unique* agent ID.
- Agents are assigned *log IDs* in addition to agent IDs, which are used to identify agents that have the same properties (arrival time, private valuation) across multiple market models. Therefore, multiple agents may have the same log ID.

Note that within the `config` directory there is a file called `env.properties`, which is used to specify the logging level. Log level 2 will be adequate for most debugging purposes. While printing to `System.out` will be useful for initial debugging, going through the logs and observation files will be necessary in order to verify that agents are behaving as expected.

1 Market model creation

When market models are created, the following are specified: (1) number and type of each model, (2) Model configuration and model ID (in curly brackets), and (3) markets in each model and the market IDs (in square brackets). For example:

`Models: 2 TWOMARKET` — there are 2 instances of the TwoMarket model.

`TWOMARKET-LA: {2} {config=LA}` — one instance is specified by configuration string "LA" (recall this was given in the spec file), the name of this model instance is "TWOMARKET-LA" and the model ID is 2 (in curly brackets)

`Markets: CDA, [-2]` — within the TwoMarket model, there is a CDA with ID=−2

An example of a complete market model creation sequence is given below:

```
1360634585206|X|2|-----------------------------------------------
1360634585206|X|2|    Creating MARKET MODELS
1360634585206|X|2|Models: 1 CENTRALCDA
1360634585208|X|2|CENTRALCDA-1: {1} {config=1}
1360634585212|X|2|Markets: CDA, [-1]
1360634585212|X|2|Models: 2 TWOMARKET
1360634585212|X|2|TWOMARKET-LA: {2} {config=LA}
1360634585213|X|2|Markets: CDA, [-2]
1360634585213|X|2|Markets: CDA, [-3]
1360634585213|X|2|TWOMARKET-DUMMY: {3} {config=DUMMY}
1360634585213|X|2|Markets: CDA, [-4]
1360634585213|X|2|Markets: CDA, [-5]
1360634585213|X|2|Models: 1 CENTRALCALL
1360634585213|X|2|CENTRALCALL-NBBO: {4} {config=NBBO}
1360634585215|X|2|Markets: CALL, [-6]
1360634585215|X|2|Primary model: TWOMARKET-LA
```

2 Agent creation & initialization

2.1 Agent creation

Agnets are created for each model, based on the model's configuration. This section lists the number of each type of agent within a model, as well as the strategies associated with any (HFT) agents in a model (set in the spec file).

An example of a complete agent creation sequence is given below. In thix example, in each model there are 10 ZI agents, and the LA agent is only present in the TWOMARKET-LA model:

```
1360634585215|X|2|-----------------------------------------------
1360634585215|X|2|    Creating AGENTS
1360634585215|X|2|MODEL: CENTRALCDA-1 agent types:
1360634585215|X|2|Agents: 0 LA
```

```
1360634585215|X|2|Agents: 0 ZIP
1360634585222|X|2|Agents: 10 ZI
1360634585222|X|2|Agents: 0 MARKETMAKER
1360634585222|X|2|Agents: 0 DUMMY
1360634585222|X|2|MODEL: TWOMARKET-LA agent types:
1360634585222|X|2|LA: {sleepVar=100, sleepTime=0, alpha=0.001,
strategy=sleepTime_0_alpha_0.001}
1360634585222|X|2|Agents: 1 LA
1360634585222|X|2|Agents: 0 ZIP
1360634585224|X|2|Agents: 10 ZI
1360634585224|X|2|Agents: 0 MARKETMAKER
1360634585224|X|2|Agents: 0 DUMMY
1360634585224|X|2|MODEL: TWOMARKET-DUMMY agent types:
1360634585224|X|2|Agents: 0 LA
1360634585224|X|2|Agents: 0 ZIP
1360634585226|X|2|Agents: 10 ZI
1360634585226|X|2|Agents: 0 MARKETMAKER
1360634585226|X|2|DUMMY: {strategy=}
1360634585226|X|2|Agents: 1 DUMMY
1360634585226|X|2|MODEL: CENTRALCALL-NBBO agent types:
1360634585226|X|2|Agents: 0 LA
1360634585226|X|2|Agents: 0 ZIP
1360634585228|X|2|Agents: 10 ZI
1360634585228|X|2|Agents: 0 MARKETMAKER
1360634585228|X|2|Agents: 0 DUMMY
```

2.2 Agent initialization

During agent initialization, agents are assigned: (1) arrival times; (2) private valuations, if they have one; (3) agent IDs; and (4) log IDs. The general format for logging agent initialization is:

```
<agentID>: (<logID>,{modelID})::<type>::arrivalTime=<time>, pv=<private value>
```

A partial example of agent initialization logging follows (for a simulation with 10 ZI agents and a single LA agent in market model #2, the two-market model with LA):

```
1360634585228|X|2| 1: (2,{1})::ZI::arrivalTime=22, pv=101525
1360634585228|X|2| 2: (3,{1})::ZI::arrivalTime=24, pv=80236
1360634585228|X|2| 3: (4,{1})::ZI::arrivalTime=43, pv=102504
1360634585228|X|2| 4: (5,{1})::ZI::arrivalTime=60, pv=58402
1360634585228|X|2| 5: (6,{1})::ZI::arrivalTime=140, pv=114905
1360634585229|X|2| 6: (7,{1})::ZI::arrivalTime=150, pv=133427
1360634585229|X|2| 7: (8,{1})::ZI::arrivalTime=158, pv=110723
1360634585229|X|2| 8: (9,{1})::ZI::arrivalTime=173, pv=104414
1360634585229|X|2| 9: (10,{1})::ZI::arrivalTime=234, pv=87728
1360634585229|X|2| 10: (11,{1})::ZI::arrivalTime=247, pv=116336
```

```
1360634585229|X|2| 11: (1,{2})::LA::arrivalTime=0
1360634585229|X|2| 12: (2,{2})::ZI::arrivalTime=22, pv=101525
1360634585229|X|2| 13: (3,{2})::ZI::arrivalTime=24, pv=80236
1360634585229|X|2| 14: (4,{2})::ZI::arrivalTime=43, pv=102504
...
1360634585230|X|2|    SETUP COMPLETE
1360634585230|X|2|------------------------------------------------
```

Explanations of some select lines:

`1: (2,{1})::ZI::arrivalTime=22, pv=101525` — initializes a ZI agent within market model 1 with agentID=1, logID=2, arrival time of 22 and private valuation of 101525

`12: (2,{2})::ZI::arrivalTime=22, pv=101525` — initializes a ZI agent with the same pseudorandom number generator seed but in market model 2 with agentID=12, logID=2 (the same as before), and the same arrival time/private value.

`11: (1,{2})::LA::arrivalTime=0` — initializes the latency arbitrageur with agentID=11 and logID=1. Note that LA is only initialized for market model 2.

3 Market simulation

After market models, markets, and agents are created, the simulation begins. Each line will have a timestamp (which is the number following the first 18 characters). For example, in

```
1360634585232|X|2|0 | (1,{2})->[-2],[-3]
```

the first section

```
1360634585232|X|2|
```

can be ignored, and the timestamp here is 0.

In general, agents in the log files are referred to by log ID and market model (e.g., `(1,{2})` is agent with log ID 1 in market model 2) as this makes it much easier to compare agent behavior across models. Some of the types of activities that will be logged:

Arrival Agent with log ID 1 in market model 2 arrives in markets –2 and –3.

```
0 | (1,{2})->[-2],[-3]
```

Communication with SIP See first tutorial for explanation of these activities. The last line gives the updated NBBO quote after the execution of these activities. Any price of –1 means that it is undefined. In the following example, the activities are executed by market –6. The NBBO quote is represented as being the best between multiple markets by multiple IDs within square brackets, e.g., `[-2,-3]` (see last line).

```
0 | [-6] SendToSIP(-1, -1)
0 | [-6] ProcessQuote: (Bid: -1, Ask: -1)
0 | [-6] UpdateNBBO(Bid: -1, Ask: -1) --> NBBO(Bid: -1, Ask: -1)
0 | [-5] SendToSIP(-1, -1)
0 | [-5] ProcessQuote: (Bid: -1, Ask: -1)
0 | [-4, -5] UpdateNBBO(Bid: -1, Ask: -1) --> NBBO(Bid: -1, Ask: -1)
```

Updating quotes These lines just give the updated quotes at timestamp 60 for (1) agent 1 in model 2 and (2) agent 5 in model 1. Agent 1 is the LA agent, so it is the only one that will be able to act on the Global quote (best buy/sell prices in all markets). Non-HF traders can only act based on knowledge of their primary market and the NBBO quote.

```
60 | (1,{2}) Global(Bid: -1, Ask: 84184), NBBO(Bid: -1, Ask: 84184)
60 | (5,{1}) Global(Bid: -1, Ask: 84184), NBBO(Bid: -1, Ask: 84184)
```

Bid submission ZI agent 2 in model 1 submits a bid that will be routed according to Regulation NMS. The bid it submits to market –1 is an order `(102954,-1)`, i.e. to sell 1 unit (represented by `-1`) at price 102954. See `SMAgent.java` for details on order routing.

```
22 | (2,{1}) ZI::submitNMSBid: +(102954,-1) to [-1]
22 | (2,{2}) ZI::submitNMSBid: NBBO(-1, -1) better than [-2] Quote(-1, -1)
```

Order matching and clearing A ZI agent (log ID=6) in model 1 (single-market CDA model) arrives and submits an order to buy 1 unit at price 114184 at timestamp 140. There is only one market in the model (with ID=–1).

```
140 | (6,{1})->[-1]
140 | (6,{1}) ZI::submitNMSBid: NBBO(-1, 59534) worse than/same as
[-1] Quote(-1, 59534)
140 | (6,{1}) ZI::submitNMSBid: +(114184,1) to [-1]
```

The submitted order matches with an order to sell at price 59534. Note that `5:(1 114184)` indicates that agent with logID=5 submits the aforementioned order. `MB` gives the list of matching buy orders, `MS` gives the list of matching sells. Since this is a CDA, the transaction clears at the price of the incumbent order.

```
140 | [-1] Active bids: (-1 102954) (-1 84184) (-1 105796) (-1 59534)
(1 114184)
140 | [-1] FourHeap::logSets::Buys:  Sells: 2:(-1 84184)1:(-1 102954)
3:(-1 105796) MB:   size: 1 5:(1 114184) MS:   size: 1 4:(-1 59534)
140 | [-1] Prior-clear Quote(Bid: 59534, Ask: 84184)
140 | [-1] Quantity=1 cleared at Price=59534
```

The agent processes the transaction:

```
140 | (5,{1}) Agent::updateTransactions: New transaction received:
(mktID=-1, transID=0 buyer=6, seller=5, price=59534, quantity=1, timeStamp=140)
140 | (5,{1}) Agent::updateTransactions: BUYER surplus: 114905-59534=55371,
SELLER surplus: 59534-58402=1132
140 | (5,{1}) Agent::updateTransactions: SURPLUS: 56503
140 | (5,{1}) Agent::logTransactions: CENTRALCDA-1: Current Position=-1,
Realized Profit=0, Unrealized Profit=-24650
```

```
140 | (6,{1}) Agent::updateTransactions: New transaction received:
(mktID=-1, transID=0 buyer=6, seller=5, price=59534, quantity=1, timeStamp=140)
140 | (6,{1}) Agent::updateTransactions: BUYER surplus: 114905-59534=55371,
SELLER surplus: 59534-58402=1132
140 | (6,{1}) Agent::updateTransactions: SURPLUS: 56503
140 | (6,{1}) Agent::logTransactions: CENTRALCDA-1: Current Position=1,
Realized Profit=0, Unrealized Profit=0
```

The matching orders have been removed from the order book (the `MB` and `MS` lists are now both empty):

```
140 | [-1] Active bids: (-1 102954) (-1 84184) (-1 105796) (0 59534) (0 114184)
140 | [-1] Cleared bids: (0 59534) (0 114184)
140 | [-1] FourHeap::logSets::Buys:  Sells: 2:(-1 84184)1:(-1 102954)
3:(-1 105796) MB:   size: 0  MS:   size: 0
140 | .....[-1] CDAMarket::clear: Order book cleared: Post-clear Quote(Bid: -1,
Ask: 84184)
```

4 Note on parsing observation files

Observations are saved in the format `observation#.json`, where # is the observation number (from 1 to the max number of samples gathered). As such, save different simulation runs in different directories; otherwise you run the risk of overwriting previous observations.

When observation files are parsed, the JSON headers are flattened (in the JSON file itself, each observed feature is stored as a hierarchical object, so flattening is necessary). To parse a directory of observations, use the following command:

```
./parse_single.sh <CSV> <directory of observations>
```

For example, `./parse_single.sh test.csv simulations/test` will parse all observations in the `simulations/test` folder.

Note that the parser will only generate column headers if the output CSV does not yet exist; if you are not seeing header files, delete the CSV and re-parse.