Code Jam Proposal:
Algorithmic Trading Application

Contents

1 Overview

You have been selected by Morgan Stanley's Electronic Trading group (MSET) to provide a platform for testing out new low-latency trading strategies on electronic exchanges. Trading strategies are a predefined set of rules for making trading decisions. Certain strategies are better suited to be executed algorithmically, via a computer, to take advantage of faster processing time. On a given exchange, there are many trading strategies being executed, and often the speed in which an algorithm reacts to a given market condition can greatly influence its profitability. MSET would like to benchmark several strategies in order to find the best one.

Your task is to build an application that will handle *trading*, *scheduling*, and *reporting* for MSET. Your application will connect to an exchange's price feed, process incoming prices and make trades according to the several strategies. The application will also display real-time graphs for the price feed, as well as the indicators of the various strategies. Finally, the trade history must be tracked, and supervised by trade managers that your application will schedule.

2 Strategies

MSET would like to try out several variations of *Moving Average Crossovers.* These involve tracking two moving average signals for a given stock price and executing a trade according to the movement of these indicators.

A moving average, also called a running average, is a set of data points, each of which is the average of the corresponding subset of a larger set of data points.

> Given a series of numbers and a fixed subset size, the first element of the moving average is obtained by taking the average of the initial fixed subset of the number series. Then the subset is modified by "shifting forward", that is excluding the first number of the series and including the next number following the original subset in the series. This creates a new subset of numbers, which is averaged. This process is repeated over the entire data series.[1]

Moving averages of price feeds show the trend of that price over the period averaged. Often, in technical analysis, we observe two moving averages: A short-term (i.e. with a smaller period) and a long-term average (i.e. with a larger period). A short-term average is considered *faster* because it only considers the latest prices and is more reactive, where as a long-term average is *slower* as it includes prices over a longer period of time and reacts more sluggishly. The slower average also tends to smooth out the volatility of short-term price changes. A moving average crossover occurs when the faster average crosses the slower average; it indicates a change in trend for a given data-set. The faster average can cross the slower average "from the bottom," indicating an upward trend (an indication to buy) or "from the top," indicating a downward trend (an indication to sell).

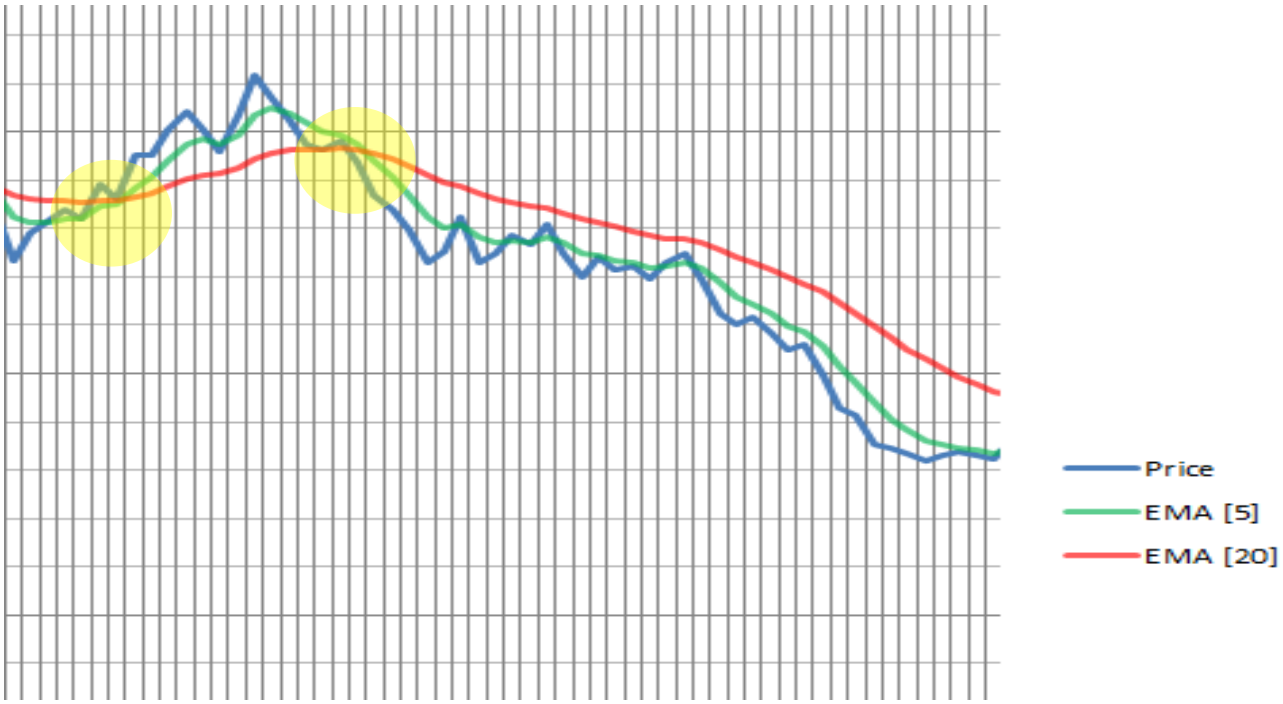


Figure 1. Example price feed with EMA crossover

[1] Moving average. *Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/wiki/Moving_average

There are several types of moving averages that can be computed, MSET would like to try the following types:

I Simple Moving Average (SMA)

The SMA is the unweighted mean of the last N data points. When calculating successive values, a new value comes into the sum and an old value drops out, meaning a full summation each time isn't necessary. We can use the formula:

$$SMA_t = SMA_{t-1} - \frac{price_{t-N}}{N} + \frac{price_t}{N}$$

However, this would require gathering at least N data points before we can obtain any values. Therefore, for the first N data points we simply take the average:

$$for\ t \leq N,$$

$$SMA_t = \frac{price_t + price_{t-1} + \ldots + price_1}{t}$$

II Linear Weighted Moving Average (LWMA)

The LWMA is similar to the SMA, however it uses weighting factors to assign more importance to the most recent data points. For a period N, the current price is multiplied by N, the previous price is multiplied by N-1, and so on. The sum of these weighted prices is then divided by the sum of the weighting factors. This can be represented by the formula:

$$LWMA_t = \frac{\sum_{i=1}^{N} price_{t-N+i} \times i}{\sum_{i=1}^{N} i}$$

As with the SMA, for the first N data points, we have to modify the formula:

$$for\ t \leq N,$$

$$LWMA_t = \frac{(price_t \times t) + (price_{t-1} \times (t-1)) + \ldots + (price_1 \times 1)}{\sum_{i=1}^{t} i}$$

III Exponential Moving Average (EMA)

The EMA, is similar to the LWMA, except it applies exponentially decreasing weighting factors to the data points, once again, giving more importance to the latest data points. For a price feed, the EMA for a period N can be calculated recursively using the formula:

$$EMA_t = EMA_{t-1} + \alpha \times (price_t - EMA_{t-1}) \quad where\ \alpha = \frac{2}{N+1}$$

Because N and *t* are not dependant on each other in this formula, we can let $EMA_1 = price_1$, then use the formula for all $t > 1$

IV Triangular Moving Average (TMA)

The TMA is a smoothed version of the SMA. It can be calculated at a given time, over a period N, as follows:

$$TMA_t = \frac{\sum_{i=1}^{N} SMA_{t-N+i}}{N}$$

For the first N values, it is calculated similar to LWMA:

$$for\ t \leq N,$$

$$TMA_t = \frac{SMA_t + SMA_{t-1} + \ \ldots + SMA_1}{t}$$

All of the strategies employed will use a “fast” period of 5, and a “slow” period of 20.

3 Architecture & Specifications

I Trading

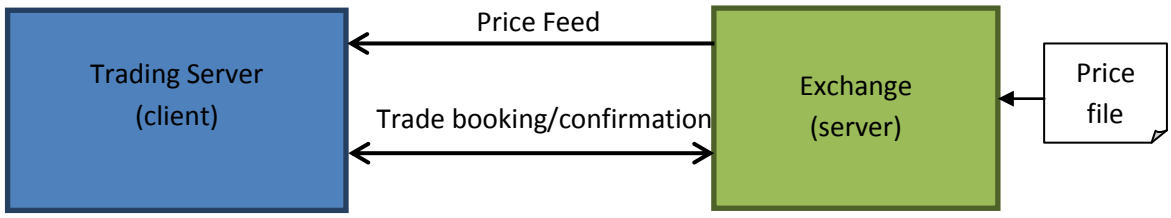


Figure 2. System Architecture

The exchange will be using two channels for communication. The price feed, a TCP socket, and the trade booking and confirmation service, a TCP socket. The port numbers for the two sockets will be configurable.

The exchange has the following usage:

```
msexchange OPTIONS
-p <priceFeedPort>
-t <tradeBookingPort>
-d <dataFileLocation>
```

The exchange will read in the prices from a supplied input file. The file is a standard ASCII text document with one price per line. We have included several sample files along with the exchange.

Once the exchange is started, it waits for a control signal to begin the price feed. The client must send the character 'H' to start the price feed. This should be done manually on the GUI with some sort of button or visual tool.

The exchange will send prices with a delimitating character '|' (pipe). The prices will be positive decimal numbers, from 0 to 999, with 3 decimal places of precision.

In order to execute a buy trade, the client sends the character 'B'. In order to execute a sell trade, the client sends the character 'S'. In both cases, the exchange responds with the price at which the trade was executed. These trade executions should be perform automatically and autonomously by the application according to the strategies.

When the trading session is over, the exchange will send the special control character 'C'. Note that the exchange will stop accepting trades as soon as the exchange closes (i.e. the last price was sent).

The exchange will respond with the error code 'E' for any unsupported operations (i.e. A trade sent after the exchange close or an unknown message).

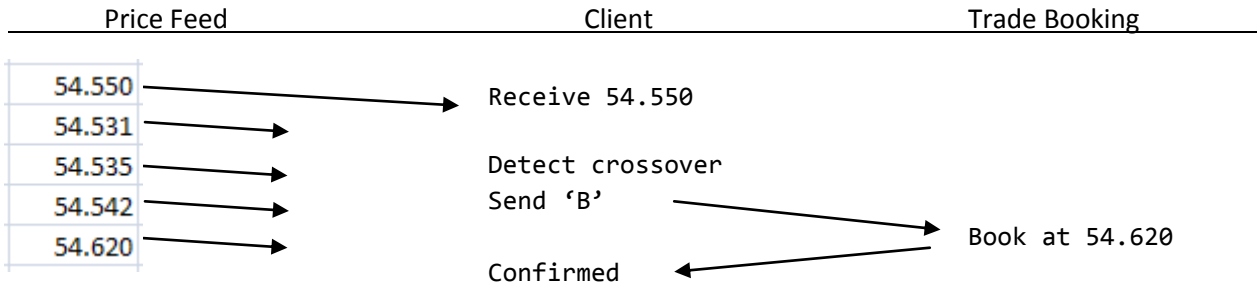


Figure 3. Sequence diagram for a trade

II Scheduling

The stock exchange is open from 9:00 AM to 6:00 PM for electronic trading. Because MSET is using your application to test out a new strategy, they want to have a trade manager to look over each trade. Your application must take care of the scheduling for these managers. For simplification, we will run the exchange on a modified time scale. The exchanges opening hours (9 hours) can be represented as 32400 seconds. We will assume that each time a price is received from the exchange it corresponds to 1 second; therefore, your application must keep track of "time" and increase it for every price you receive.

Here are the specifications for manager scheduling:

- A manager can monitor an unlimited number of trades.
- A manager can only monitor trades for a maximum of 2 strategies at a time.
- We assume there is a limited pool of managers available.
- For simplicity, name your managers Manager1, Manager2, …, ManagerX.
- A trade manager works a maximum of 4.5 hours per day (total) including a break of exactly 0.5 hours after exactly 2 hours of work.

Each trade must be assigned to a manager, and the application should minimize the number of managers required during the exchange hours. The manager's schedule should be available before the start of the trading session.

III Reporting

At the end of the trading day all of the trades need to be entered into the company's books for record keeping. Due to the official nature of this information, the file must be submitted as a signed PDF. We are going to use Silanis' e-SignLive service for this purpose. Your application must have a button that becomes enabled only at the end of the trading day. By clicking the button your application will send a list of executed trades to the Silanis API. The call is an HTTP POST to https://stage-api.e-signlive.com/aws/rest/services/codejam.

To authenticate your request you must provide the CodeJam API key by setting an HTTP request header `"Authentication: Basic Y29kZWphbTpBRkxpdGw0TEEyQWQx"`

The body of the POST request must contain the following:

- CodeJam Team Name
- The Judge's information (`mcgillcodejam2012@gmail.com`)
- List of transactions
    - Each transaction should include the time (in milliseconds from the start of the day), the type of trade ("`buy`" or "`sell`"), the executed price, the trade manager's name, and the strategy used ("`SMA`", "`LWMA`", "`EMA`" or "`TMA`").

Note: A sample JSON document is provided in the Appendix.

The return value of this request is a JSON formatted signing ceremony id. Your application needs to display this ID upon successfully completing the API call.

4 Judging Criteria

Your application will be judged on the following criteria:

I Trading

- The correct number of trades was made for each crossover strategy given the price feed.
- The latency of trades with respect to the point of the crossover. [minimized]
- The GUI
    - Real-time graph of price, fast and slow moving averages for each strategy. [4]
        - In order to get a decent granularity of the prices and averages on the graph, it is suggested to show a “rolling window” of the last N (i.e. 100) prices. [optional]
    - Trade history, for each strategy, indicating, the time, type and price. Can be tabular, integrated with the graph or some other visual representation.

II Scheduling

- A schedule was generated prior to the start of the trading session.
    - The schedule should be represented visually from your application. It can be tabular, in the form of a Gant chart or otherwise.
- Each trade was assigned to the appropriate manager.
- The number of managers required was minimized.

III Reporting

- Button to send report is only functional at end of trading session.
- Successful API call to e-SignLive service (Judge receives signing ceremony email).
- A copy of the JSON data is saved locally.
- Ceremony ID is displayed on GUI.

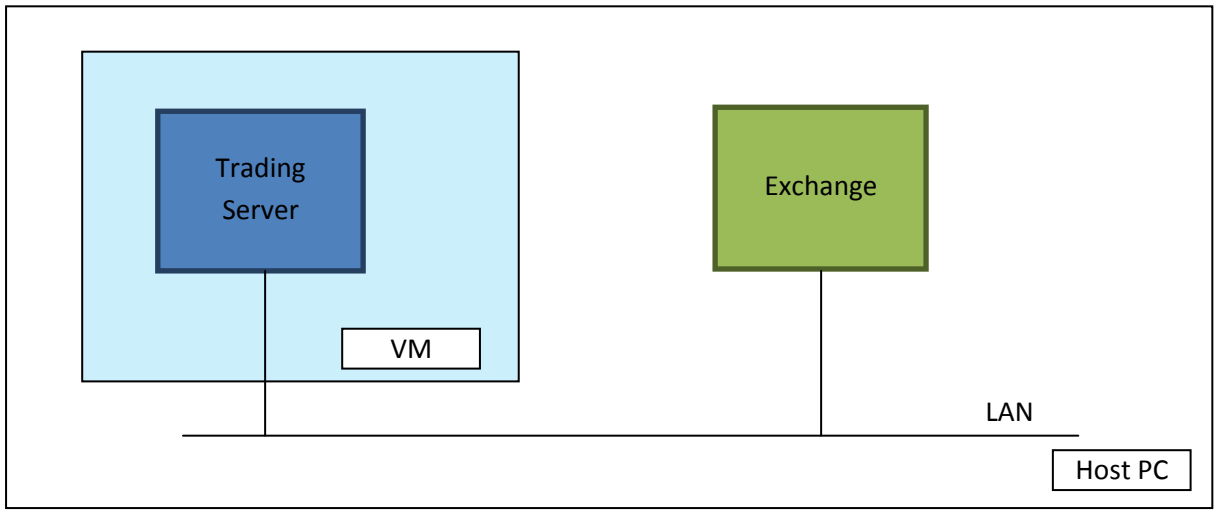
5 Judging and Submission Procedures

You will be provided with a virtual box image (Windows and/or Linux) on a USB key. You will install your application into this image. You will return the USB key at the end of the weekend. Make sure you provide with your image the following information:

- How to start and stop your application from inside the virtual image. There should be clear and simple instructions for starting your application. The use of a start-up script is encouraged when multiple processes need to be started.
  Note: Failure for the judges to run your application will result in DISQUALIFICATION
- The TCP ports you are using for the price and trade confirmation feed. (optionally, these can be made configurable in your application)
- If you are providing your GUI through a web based paradigm include the home page URL (e.g. http://localhost:8080/mset/home) of your GUI.

Make sure you leave yourself plenty of time to configure the application in your virtual OS image. Last year a surprising number of submissions didn't run for the committee in the judging phase after the weekend.

Make sure that your trading application can connect to the exchange running outside the virtual machine from another physical host on a local LAN. This is how the judges will test your application.



6 Appendix

I Strategies

Here are some sample values for each type of Moving Average:

Time	Price	SMA[5]	LWMA[5]	EMA[5]	TMA[5]
1	61.590	61.590	61.590	61.590	61.590
2	61.440	61.515	61.490	61.540	61.553
3	61.320	61.450	61.405	61.467	61.518
4	61.670	61.505	61.511	61.534	61.515
5	61.920	61.588	61.647	61.663	61.530
6	62.610	61.792	61.988	61.979	61.570
7	62.880	62.080	62.351	62.279	61.683
8	63.060	62.428	62.677	62.539	61.879
9	63.290	62.752	62.965	62.790	62.128
10	63.320	63.032	63.154	62.966	62.417
11	63.260	63.162	63.230	63.064	62.691
12	63.120	63.210	63.216	63.083	62.917
13	62.240	63.046	62.893	62.802	63.040
14	62.190	62.826	62.607	62.598	63.055
15	62.890	62.740	62.629	62.695	62.997

II Exchange

If you want to test the exchange before writing your application you can use Telnet.

From one terminal, start the exchange:

```
java -jar msExchange.jar -p 3000 -t 3001 -d data/amd.txt
```

From another terminal

```
telnet localhost 3000

//send 'H'
```

From another terminal

```
telnet localhost 3001

//send B or S
```

III Reporting

Example JSON data: (codejam.json)

```
{
  "team" : "Flying monkeys",
  "destination" : "mcgillcodejam2012@gmail.com",
  "transactions" : [
    {
       "time" : "8004",
       "type" : "buy",
       "price" : 120,
       "manager" : "Manager1"
       "strategy" : "EMA"
    },
    {
       "time" : "9589",
       "type" : "sell",
       "price" : 122,
       "manager" : "Manager2",
       "strategy" : "LWMA"
    },
    {
       "time" : "16542",
       "type" : "buy",
       "price" : 118,
       "manager" : "Manager1",
       "strategy" : "TMA"
    }
  ]
}
```

Example HTTP Request using curl:

```
#! /bin/sh
# create signing process
curl -X "POST" -H "Authorization: Basic Y29kZWphbTpBRkxpdGw0TEEyQWQx" -H "Content-
Type:application/json" --data-binary @codejam.json "https://stage-api.e-
signlive.com/aws/rest/services/codejam"
```

Example response:

```
{ "ceremonyId" : "12345..." }
```