High Performance Load Generator for Automated Trading Systems Testing

D.K. Guriev
"Innovative Trading Systems", LLC
Saratov, Russia
Email: dmitry.guriev@exactprosystems.com

M.A. Gai
"Innovative Trading Systems", LLC
Moscow, Russia
Email: maria.gai@exactprosystems.com

A.A. Terentyev
Yuri Gagarin State Technical University of Saratov
Saratov, Russia
Email: a_a_terentyev@mail.ru

I.L. Itkin
"Exactpro Systems", LLC
San Rafael, CA, USA
Email: iosif.itkin@exactprosystems.com

***Abstract*—The growing volume of orders generated by HFT (high-frequency trading) systems has posed a challenge of conducting exchange and brokerage systems testing in production-like environments. Specialized testing tools are used to ensure quality of high load trading systems with high availability. The main requirement for such tools is that they should be capable of creating realistic, high loads using limited hardware infrastructure. This article describes a load injection tool developed for testing automated trading systems and an approach that ensures high performance.**

***Keywords*: load testing, high-frequency trading (HFT), testing tools**

I. Introduction

High-frequency trading of financial instruments (HFT) allowing to reduce latency when trades are conducted has grown recently and currently accounts for about 30% of all equity trading in the UK and, possibly, over 60% of all equity trading in the USA [1]. Due to this, brokerage and exchange systems experience growing load from the stream of transactions generated by automated trading systems. The operators of trading platforms, regulators and trading participants must be sure of reliability of software and infrastructure of the trading platforms [2] in the conditions of constantly growing load.

In the course of software development, methods of load testing are used in order to determine the maximum throughput, possible bottlenecks and locating elements of the systems that can be problematic. Load testing is understood as the process of sending a large amount of orders into the system, verification of timeliness and correctness of responses received from it, and verification of the system's internal state.

Various commercial and open source load generators are currently used for software load testing. The following products can serve as examples: Apache JMeter, HP Load Runner, IBM Rational Performance Testing, Borland Silk Performer, and others [3-6]. The main concept used in these products is that a large number of virtual users are created emulating the behavior of real ones in order to model the conditions the program/system will operate in real life. When the testing tools imitate and support the connection with a large number of users under high load, performance limitations can surface. These are described in the second part of this paper.

A tool for testing high load trading systems with the required performance characteristics has been developed by Exactpro Systems LLC. The tool has been used to verify some of the largest exchange technology infrastructures in Western Europe [7; 8]. The developed instrument supports the following protocols: FIX (all versions), ITCH, LSE, Native, SOLA SAIL & HSVF, HTTP, SOAP, and various binary trading systems protocols. One of the architectural distinctions of the developed testing tool is its ability to support multiple protocols. Therefore, the addition of new protocols and new versions of protocols that are already supported is now a relatively low-cost task. The third part of the paper focuses on certain aspects of load testing data preparation. The fourth part presents the tool's capabilities of its tuning, including setting the load profile.

II. The Optimization of Load Creation Process

A number of sources, in particular [9], provide general information about the process of load creation. Load generators are classified into measurement-based and model-based ones [10]. Measurement-based load generators are useful for finding throughput of the system under test and building dependencies between response times and load. Model-based load generators are used for simulating entry data distribution that is as close as possible to what the system will experience in production. The high performance load generator developed by the authors supports both of the above mentioned approaches. The models are used to create configuration files before the generator is launched. Therefore, the tool can avoid using up the resources for processing information pertaining to the model at the time the tests are performed.

Load generators are also classified into closed-cycle and open-cycle ones [11]. In closed-cycle generators, having sent a message, the thread being executed waits for a response from the system before it starts sending the next portion of requests. An open-cycle generator can continue sending messages without waiting for a response from the system under test. The majority of web-testing tools are closed-cycle generators. This is due to using the concept of virtual users, each of which consecutively performs the steps of a test scenario. In comparison to an open-cycle generator, a closed-cycle one requires a much larger number of threads of execution and switching between them. Often, closed-cycle generators process system responses and messages sent into the system in the same thread, which additionally reduces the tool's performance and sometimes even affects accuracy. Thus, open-cycle generators require less hardware to create the needed load levels. The also do not require creating extra threads and their synchronization in order to create a fixed load level. The tool presented in this paper is an open-cycle generator.

When a load testing tool was being developed, consideration was given to the necessity of aligning the executable threads and processor cores in order to flatten the distribution of system's incoming messages over time [12]. The authors have come to the conclusion that the millisecond resolution of system timers, which is characteristic of the majority of contemporary Linux-systems, is sufficient for creating a realistic trading load. The absence of dependency on the processor cores releases a certain amount of hardware resources for the load generator and allows launching an optimal amount of threads in a centralized way. The developed tool uses a central controller, which makes it possible to configure the required number and parameters of protocol connections for each thread in advance. The presence of a central controller also allows issuing a command to perform coordinated actions by all threads. For example, a concurrent start of a messages stream or a concurrent drop of established connections.

When a trading system is being tested, a load generator substitutes a large number of automated trading systems using a multitude of servers. Hardware that is available for setting up testing tools is always limited, though, due to economical reasons [13]. The conditions of continuing financial unrest dictate that even the largest financial institutions must function under the regime of maximum spending optimization. It is therefore necessary that the process of creating outgoing messages is as light as possible. For the process of load creation to be optimal, it is necessary to prepare messages templates before the test is performed, thus reducing the processor time on the servers where the tools are deployed to. Similar concepts were applied by the developers of a load generator at Yandex, the largest Russian search engine. Their open source "Yandex.Tank" testing tool is designed for generating huge volumes of messages over HTTP protocol [14]. The high performance capabilities of "Yandex.Tank" are thanks to concentrating the load in a single session and a single thread as well as using a prepared file with static requests. Load generators for trading systems cannot use static data and have a few other limitations, which are examined in the next part of the article.

III. Specifics of Load Testing For Trading Systems

[15] examines the main requirements for load modeling of high-frequency trading . The logic of how such systems work presents significant limitations for using static data that was recorded earlier or is pre-determined. This section describes some of the specifics of creating load and preparing incoming data for trading systems testing.

3.1. Preparing messages from templates

Due to the fact that trading is not anonymous, in order to maintain the session the server must receive the names of existing users, correct sequential message numbers, as well as time stamps of when each message was sent. Our analysis has shown that building the messages right before sending them by using dictionaries is very expensive from the system resource usage standpoint. Therefore, a decision was made to use templates where the order of fields and the set of key values are set before the test begins. For example, in a NewOrderSingle FIX message, only a few parameters will be changed before the time it is sent. Such parameters must be unique (for example, ClOrdID(11) – the number of a client order) and must depend on the current time (ExpireTime(126) – the time when the order expires, ExpireDate(432) – the order's expiry date). The rest of the parameters in a new order do not change. The following parameters are changed in order to maintain the session:

BodyLength(9) – message lenghth;

SenderCompID(49) – the name of the company, which sent the order;

TargetCompID(56) – the name of the company the order was sent to;

MsgSeqNum(34) – a unique message number;

SendingTime(52) – current time;

CheckSum(10) – verification value.

All needed parameters taken from the NewOrderSingle message are placed into OrderCancelReplaceRequest and OrderCancelRequest messages. Such as:

OrderID(37) – order identifier;

Price(44) – order price;

Quantity(53) – order size;

Side(54) – order side (buy or sell);

Symbol(55) – instrument symbol.

The assumption that all parameter values are true makes it possible to significantly reduce the time for checking the field values and for verifying that their sequence in a message is correct.

3.2. Playback of earlier recorded data

Sending of recorded data prepared in advance leads to distorting the test. When test data is being prepared, it it required to keep the same proportion of trading events as observed in real trading. The analysis of real trading shows that one trade can consist of over 20 order changes.

Recorded data consist of a variety of new orders, their changes, and cancellations. Due to the fact that orders are sent from different streams to the same order book, they can arrive in sequence that differs from what was recorded. Even if the arrival time of an order to the market differs slightly, the consequences can be detrimental. For example, by the time the market receives an order, other orders may have been placed in the order book in a sequence that is different to what is in the recording and may be traded in a different sequence. Therefore, later changes or cancellation of the order will be impossible if the order has been traded and removed from the order book. This leads to a malfunction in the order of test execution which will later lead to a scenario that differs from the recorded one. Thus, the test market will witness events different from the ones that occurred when the scenario was recorded. For example, the market will send a much larger number of rejects to change or cancel orders. Also, the differences from the original scenario will accumulate, and the proportion of the trading events may significantly differ from the original one.

3.3. Usage of determined scenarios

One other possibility to set up a test is preparing two groups of orders: the first one will contain orders that are known to become part of trades (active orders); the second one consists of orders that must not be traded (passive orders). However, possible difficulties brought by the Market Surveillance System should be taken into account. One of the functions of the component is real-time monitoring of "prearranged" orders, i.e. the very same orders that must be traded when the test is conducted, and reporting such events to a service watching out for market price manipulation. Obviously, this option is not acceptable when test scenarios are created. Due to that, our specialists have built a randomized load generator that uses a feedback mechanism. Randomizing is used to generate a new price when order amendment message is sent. Three parameters are used for such randomization: original price, price change range and price tick.

The original price is set in the array of data for each instrument and for each side (buy or sell). The original prices must meet the following conditions:

- price of buying must be lower than price of selling;
- the difference between the original sell prices and buy prices must be around 2-3% of opening price.

The buy and sell price change range is selected in such a way that the prices of corresponding offers overlap, making trading possible, and that offer prices are not over 10% borderline – a condition which, if broken, may lead to stopping trading in an instrument or the entire segment of instruments. These parameters allow for flexible selection of desired average proportion of the number of orders and the number of amendments per one order on average. The smaller the area where buy and sell prices overlap is, the less amount of trades is possible, and the larger the number of amendments will happen on average. Note should be taken that this nonlinearly depends on the price overlap interval. As the prices are set for each instrument individually, it becomes possible to set up different number of trades on different instruments for one test.

Price tick is set based on instrument configuration. For example, some instruments can trade with 0.05 price tick, others can have 0.10 price tick.

The feedback mechanism is necessary in order to track the order status on the market and ensure the possibility of order amendment or cancellation. The order can have the following statuses:

New – the order hasn't participated in trading;

PartFilled – the order has been partially filled;

Filled – order has been fully filled;

Canceled – the client has canceled the order;

Expired – the order lifetime has expired;

Rejected – the exchange has rejected the order.

Only orders in a "New" or "PartFilled" status can participate in trading. As soon as an order changes its status, it becomes irrelevant and the information about it is deleted instantly.

IV. AN EXAMPLE OF THE DEVELOPED LOAD GENERATOR

In this part of the paper, we will examine several options of setting up the load generator developed by the authors for testing trading systems based on FIX protocol [16].

The test is set up with the help of 4 types of configuration files containing the following:

- setting up load parameters;
- session configuration;
- preparation of messages;
- distribution across messages.

4.1. Format of file with load parameters

In order to set up the load parameters a file with the following format is used:

```
#Configuration file with session parameters:
CONNECTIONS_CONFIG = fixConnections.cfg
#Sessions used from the sessions file:
CONNECTIONS_RANGE = 1-3, 5, 7-
#A file with messages stubs:
MESSAGE_TEMPLATES = fixMessageTemplates.dat
#A file with distribution by message:
MESSAGE_RATES = messageRates.cfg
#Sequence of actions before the start of the test:
INIT_CONFIG = connect(100ms), logon(3s)
#Load configuration:
LOAD_CONFIG = const(1000,5m)
#A steady load of 1,000 messages per second is set up
```

```
#Over the course of 5 minutes
#The number of repetitions of the load scenario determined by
#the LOAD_CONFIG parameter:
NUMBER_REPETITIONS = 10
#Seguence of events after the end of test:
SHUTDOWN_CONFIG = logout(1s), disconnect(10ms)
#Sequence of actions if connection is dropped
#unexpectedly
ON_RECONNECT_CONFIG = connect(10ms), logon(3s)
#Flag ot perform actions determined in
#ON_RECONNECT_CONFIG if connection is dropped:
HOLD_CONNECTION = 1
#If value = 0, actions in ON_RECONNECT_CONFIG are not
#performed, and connection is not re-established.
#Delay time between session authorization, in milliseconds
LOGON_INTERVAL = 1000
```

As the clients have the ability to use their own trading programs, a possibility exists that due to a reason, such as a large volume of messages sent by the system, the client's programs won't be able to read the data. This can negatively affect the behavior of the trading system. In order to test this scenario, the developed software product has a special limitation on the number of read data per second allowing emulating slow clients.

4.2. Format of file with session configuration

Connection parameters are set up in a file of the following format: the [COMMON] section is used for setting up general connection parameters

```
[COMMON]
HOST = 10.10.10.10
PORT = 5555
TARGET_COMP_ID = FGW
```

The [FIX] section is used for setting up unique parameters of a separate connection:

```
[FIX]
SENDER_COMP_ID = LOAD_1
RESET_SEQ_NUM_AFTER_LOGOUT = 0
PARTY_ID = LOAD_1
```

The [FIX] section must be repeated as many times as many connections are planned to be used. The connections that are to participate in the test are determined by the CONNECTIONS_RANGE parameter that is part of the file containing the load parameters.

4.3. Format of file with messages stubs

The file contains an array of named message stubs. The stubs have correct format and sequence of message fields. Some of the fields will be replaced by correct data right before a message is sent.

```
Logon
8=FIXT.1.1|9=61|35=A|34=1|49=SenderCompID|56=TargetCompI
D|98=0|108=3600|554=password|1137=9|10=135|EOM

NewOrderBuy
8=FIXT.1.1|9=199|35=D|34=1|49=SenderCompID|56=TargetComp
ID|1=CLIENT|11=ClOrdID|38=200|40=2|44=9.8|54=1|55=Symbol|
59=6|60=20130728-
13:34:03.194|432=20130730|528=P|581=3|1138=60000|9303=I|45
3=1|448=PartyID|447=D|452=76|10=047|EOM

NewOrderSell
8=FIXT.1.1|9=199|35=D|34=1|49=SenderCompID|56=TargetComp
ID|1=CLIENT|11=ClOrdID|38=150|40=2|44=10.2|54=2|55=Symbo
l|59=6|60=20130728-
13:34:03.194|432=20130730|528=P|581=3|1138=60000|9303=I|45
3=1|448=PartyID|447=D|452=76|10=047|EOM

Cancel
8=FIXT.1.1|9=134|35=F|34=1|49=SenderCompID|56=TargetCompI
D|11=ClOrdID|41=OrigClOrdID|54=1|55=Symbol|60=20130728-
13:34:03.178|9303=I|453=1|448=PartyID|447=D|452=76|10=050|
EOM

Replace
8=FIXT.1.1|9=179|35=G|34=1|49=SenderCompID|56=TargetComp
ID|1=CLIENT|11=ClOrdID|38=180|40=2|41=OrigClOrdID|54=1|5
5=Symbol|60=20130728-
13:34:03.178|432=20130730|1138=70000|9303=I|453=1|448=Party
ID|447=D|452=76|10=077|EOM

Logout
8=FIXT.1.1|9=29|35=5|34=111|49=SenderCompID|56=TargetComp
ID|10=249|EOM
```

4.4. Format of file with messages stubs

The file contains a proportion of messages number in portions per each message type:

```
NewOrderBuy = 15
Replace = 50
Cancel = 5
```

Depending on the setting, MESSAGE_SELECTION-ORDER = sequential or = random, the messages will be selected either sequentially or at random.

4.5. A simplified algorithm schema

Figure 1. shows the schema for an algorithm selecting and sending messages. The first stage consists of reading incoming data. If data exists, the analysis of received responses and the amendment of order statuses or their parameters take place. Then a new message is selected via random or consecutive sorting. If a command to create a new order was selected, its parameters are committed to memory for future use. If a message for amending or cancelling an order was selected, an order without a corresponding unanswered request is picked. The order parameters are inserted into the message and sent into the trading system. Then the time elapsed since the beginning of the iteration is computed. It is compared with the pre-calculated average time for one iteration, and a pause is made if needed. The "read-send-wait" sequence allows taking into account all latest changes within the system under test and sending functionally correct messages based on that.

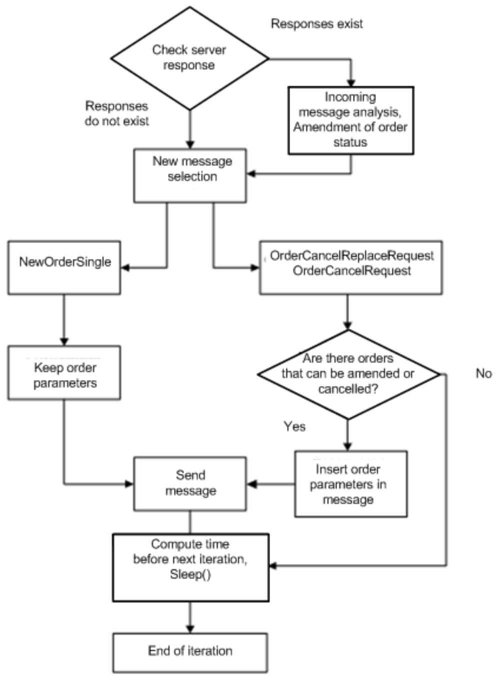
Check server response
Responses exist
Incoming message analysis, Amendment of order status
Responses do not exist
New message selection
NewOrderSingle
OrderCancelReplaceRequest OrderCancelRequest
Keep order parameters
Are there orders that can be amended or cancelled?
No
Yes
Send message
Insert order parameters in message
Compute time before next iteration, Sleep()
End of iteration

Fig. 1. Algorithm schema of receiving and sending messages

Launching of a test with maximum load on Intel(R) Xeon(R) CPU X5570 @ 2.93GHz produces up to 70,000 messages per second at exit from single core and can be linearly scalable if a higher number of cores is used. The results were confirmed when generating streams were distributed across 8 cores against an exchange system in production-like environment was used as the target. The indicated load volume created from a single server exceeds throughput capacity of existing equity trading systems. The 70,000 from a single core figure corresponds to the maximum metric for testing of web-infrastructures with the help of static requests [17].

4.6. Configuring the load profile

This section describes the load profile configuration. The load is determined by the following parameter: *LOAD_CONFIG = phase1 [, phase2, ... phaseN]*

The load phase can be as follows:

- *const (freq, dur)* – steady load with freq frequency dur duration. A shortened format can also be used – freq:dur;

- *step (freq, delta, steps, dur)* – rising load with starting frequency freq, the leg of frequency change delta, the number of steps steps, and the duration of one step dur

- *connect (dur)* – all sessions must establish connection with dur delay;

- *disconnect (dur)* – all sessions must drop connection with dur delay;

- *logon (dur)* – all sessions must send a message with authorization with dur delay;

- *logout (dur)* – all sessions must send a message about ending the session with dur delay.

Dropping of a connection is not a critical problem in itself. For example, all web-connections, and especially connections in mobile applications, are designed for being dropped. For financial protocols, this event may mean that a trading participant may lose control over his orders, and many systems are configured to cancel all open orders. If the trading participant is actively trading and sends many orders, the loss of connection will result in mass cancellation of his orders by the system, which will lead to higher load to the system's core. It is also necessary to know how the system will behave when the connection is restored and authorized under load. It is very important to have the ability to repeat instant loss and restoration of the connection under load. This is the reason why the phases described above (connect, disconnect, logon, logout) were created.

Figure 2 illustrates various load profiles: const, step and micro burst. The latter one is created with the help of the const phase characterized by having a short duration and a high load.

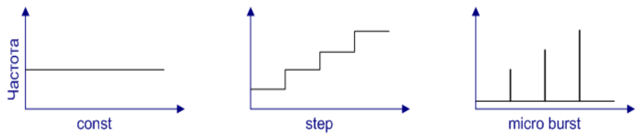

Fig. 2. The simplest load profiles

const: LOAD_CONFIG=const(1000, 20m)

step: LOAD_CONFIG=step(500, 500, 4, 4m)

micro burst: LOAD_CONFIG=200:5m, 40000:10ms, 200:5m, 75000:10ms, 200:5m

The authors' experience shows that the step load shape (load) is most suitable for determining the system's maximum performance. The micro-burst load shape is the closest one to replicate contemporary high load trading systems.

V. Conclusion

The developed testing tool described in this paper is used for measuring the throughput and response times of large scale exchange and brokerage systems supporting the technological infrastructure of the financial markets as part of projects supported by Exactpro Systems LLC. The results confirm the efficiency of selected methods: managing the executed threads via a central controller and using pre-configured templates for generating message streams.

Further work is planned to expand the list of open and proprietary communication protocols supported by the described testing tool. Despite the fact that the existing performance of the load generator allows creating a realistic data stream from a single server capable of overloading any of currently existing trading platforms and ensuring the quality of the trading systems to appear in the coming years, plans are being made to develop a scalable module capable of controlling load created from several servers.

The main direction of research work will be bringing the mechanisms of processing reverse data stream to perfection in order to expand the complexity of the load testing scenarios for trading systems and making the scenarios more realistic. It is also required to keep the testing tools efficient and economical.

REFERENCES

[1] The Future of Computer Trading in Financial Markets. Final Project Report. / Foresight. The Government Office for Science, London. // [Electronic resource]. – Access mode http://www.bis.gov.uk/assets/foresight/docs/computer-trading/12-1086-future-of-computer-trading-in-financial-markets-report.pdf

[2] Commission Roundtable on Technology and Trading: Promoting Stability in Today's Markets. / U.S. Securities and Exchange, October 2, 2012 // [Electronic resource]. – Access mode http://www.sec.gov/news/otherwebcasts/2012/ttr100212-transcript.pdf

[3] Apache JMeter manual // [Electronic resource]. – Access mode http://jmeter.apache.org/usermanual/jmeter_distributed_testing_step_by_step.pdf

[4] HP LoadRunner manual // [Electronic resource]. – Access mode ftp://ftp.itrc.hp.com/applications/HPSoftware/ONLINE_HELP/LoadRunner11.50_User.pdf

[5] Rational Performance Tester // [Electronic resource]. – Access mode http://www-03.ibm.com/software/products/ru/ru/performance/

[6] Silk Performer // [Electronic resource]. – Access mode http://www.borland.com/products/silkperformer/

[7] Penhaligan,P.: Equity Trading: Performance, Latency & Throughput. / ExTENT Conference // [Electronic resource]. – Access mode http://www.slideshare.net/extentconf/extent3-turquoise-equitytrading2012

[8] Benedetti E., Zanetti L.: London Stock Exchange - "The Focus Beyond Low Latency". / ExTENT Conference // [Electronic resource]. – Access mode http://www.slideshare.net/extentconf/extent-2013-obninsk-lse-the-focus-beyond-low-latency

[9] Cong, J.: Load Specification and Load Generation for Multimedia Traffic Loads in Computer Networks. // Ph.D. Dissertation, FB Informatik, Univ. Hamburg, 2006; also published at: Shaker-Verlag, Aachen 2006

[10] Jing Cong, Bernd E. Wolfinger: A Unified Load Generator Based on Formal Load Specification and Load Transformation // valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodolgies and tools

[11] Bodík P., Fox A., Franklin M., Jordan M., Patterson D.: Characterizing, Modeling, and Generating Workload Spikes for Stateful Services // SoCC'10, June 10–11, 2010

[12] David Mosberger, Tai Jin: httperf—a tool for measuring web server performance // SIGMETRICS Performance Evaluation Review , Volume 26 Issue 3, December 1998

[13] Itkin, I.L.: Testing of exchange systems in high-frequency trading mode // SQA Days #10:http://sqadays.com/talk.sdf/sqadays/11151/talks/12196

[14] Yandex.Tank Documentation // [Electronic resource]. – Access mode https://media.readthedocs.org/pdf/yandextank/latest/yandextank.pdf

[15] Itkin Iosif: Theory of High Frequency Trading systems testing // Software Development & Analysis Technologies Seminar http://sdat.ispras.ru/2011/09/20-октября-модели-тестирования-систем/ http://www.slideshare.net/IosifItkin/theory-of-high-frequency-trading-systems-testing

[16] Official website of FIXprotocol // [Electronic resource]. – Access mode http://www.fixprotocol.org/

[17] How to Generate Millions of HTTP Requests // [Electronic resource]. – Access mode http://dak1n1.com/blog/14-http-load-generate