exactpro

EXITUS ACTA PROBAT

Performance Testing Principles

DECEMBER 2009

Contents

1. Introduction

This document sets out the principles to be considered when the performance of a trading system is tested. It is based on our experience in the testing of matching engines, smart order routers, order management and execution management systems, straight through processing, risk management and position keeping systems and front- middle- and back-office platforms. We describe items we believe will have a significant impact on the effort involved in load testing and on the credibility of test results.

2. Test Requirements

Before starting any testing, you should identify performance targets for your system in terms of specific, clear, verifiable, realistic and agreed requirements. Static analysis of these requirements allows particular problems to be predicted prior to starting the tests.

Keep up to date any documentation that describes the test environment configuration, the settings and the maintenance procedures.

3. Test Scenarios

Use simple and repeatable tests to assist development with optimisation. Use a realistic comprehensive business model to drive the tests used to verify the system. Start with a reasonably low load to ensure that the testing is functionally correct.

Check the system behaviour during specific market events that result in message outbursts, e.g. volatility auctions. Enlist high availability and fail-over scenarios supported by your system and decide whether they can or should be tested.

If you test complex workflow processes that involve multiple participants, do not create long scenarios with many dependencies. Instead, try to test them using a set of independent agents with simple logic. Not only does it make your testing more realistic, but it also helps to make scenarios much more stable.

Because extra delays may occur if all orders go from a single participant or are for a particular security, try to use realistic data distribution. Whenever possible, use scrubbed production data to generate test data sets.

When trading desktops are involved and apart from automatic measurement and storing transaction timings, ensure that manual verification is in place and that QA analysts check how the application really behaves when the server back-end is under load.

4. Load Testing Tools

Good overall system performance starts with good performance at the unit and component level; unit tests should incorporate performance measurements.

Load injectors should be as light-weight as possible. This decreases the hardware footprint on the test lab and, importantly, the results will not be affected by internal latencies within the test tool.

Concentrate on the system, not on the test tools. Do what is required for your system, as opposed to what can just be easily obtained from the test tools.

One of the most time consuming steps in performance testing is processing of the test results; try to optimize processing time as it affects overall test duration.

Reconcile information obtained from various points in the system: logs, persistence and network capture. This allows transactions lost under load to be identified.

5. Latency Measurements

Time measurement and synchronisation is a particular challenge when working on a near-millisecond scale. So be critical: always challenge measurement methods and always assess the validity of test results.

Be specific on the precise meaning of statements that latency is below a given figure: is it average; does it include spikes; can one assume that a certain percentage of messages will be processed faster (e.g. 99%)?

Optimal throughput and latency is achieved when the processing rate equals the rate at which the messages appear in the inbound queue. Latency figures obtained on this assumption may appear good, but do they represent anything close to reality?

- If order volume is low, the system will have to constantly query an empty queue and, as a result, there will be either 100% CPU load on a particular core or additional delays caused by context switching and other tiny effects that will outweigh the target latency figure.
- When messages arrive in batches, extra delays come into play: while the system is busy processing one of the messages in the batch, the other ones have to wait.
- A real-life scenario should represent a mixture of these two extreme cases: most of the time a queue is empty, and when messages arrive they are not alone.

6. Test Environment

Be prepared to repeat the tests many times. In order to do this efficiently, you should plan the test environment maintenance procedures in advance: automatic restart, clean-up and redeployment, monitoring information and logs collection.

Ensure that logging and monitoring of the test environment are at the same level they would be in production. Extra logging usually produces a heavy load on the system; on the other hand, production-level logging should be sufficient to investigate the problems.

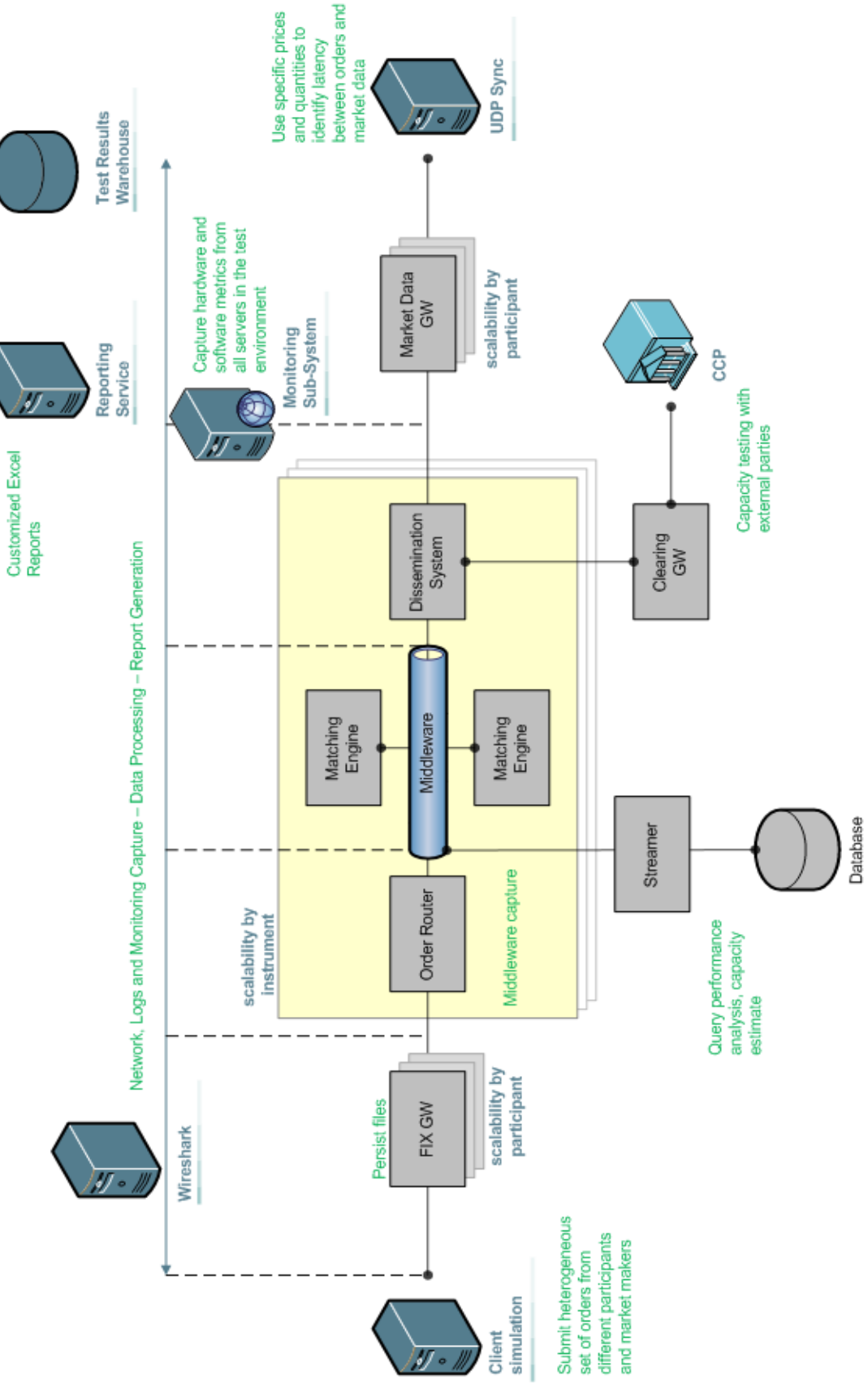
Be careful that results obtained in the test lab are not extrapolated too optimistically when the system is moved to more powerful production hardware:

- Only a limited percentage of systems have proven horizontal scalability;
- Linear increases in throughput are often not achievable by adding extra servers;
- Extra agents can sometimes even produce extra load and reduce a system's capacity;
- For complex systems, it is difficult to be sure that vertical scalability can be achieved; and increases in particular parameters (memory, CPU, I/O, etc.) will not necessarily speed-up the system.

Few load-testing projects run without logs occupying the whole disk space or database table space running out, at least once. Plan in advance and have clean-up procedures in place.

7. Typical Load Testing Project

The diagram on the next page shows a typical load testing solution. It represents the testing of a simplified version of an MTF or ATS. A realistic business model is used to submit orders into the system. Information is collected via monitoring agents, network capture, log files, persist files and the database. All information is processed on the reporting server and stored in the test results data warehouse.


Test Results Warehouse
Use specific prices and quantities to identify latency between orders and market data
UDP Sync
Reporting Service
Capture hardware and software metrics from all servers in the test environment
Monitoring Sub-System
Market Data GW
scalability by participant
CCP
Customized Excel Reports
Dissemination System
Clearing GW
Capacity testing with external parties
Matching Engine
Middleware
Matching Engine
Network, Logs and Monitoring Capture – Data Processing – Report Generation
Streamer
Database
scalability by instrument
Order Router
Middleware capture
Query performance analysis, capacity estimate
Wireshark
Persist files
FIX GW
scalability by participant
Client simulation
Submit heterogeneous set of orders from different participants and market makers