



**Speed, High Availability,
Scalability: The Goals of
Trading System Design**

Reader ROI

- The importance of speed, high availability and scalability, the main goals for trading systems software designers, are objectives that must be clearly understood.
- Messaging can and should be used as a backbone of the trading systems.

Speed, High Availability, Scalability: The Goals of Trading System Design

A user places an order to buy 500 shares of stock XYZ. The order is executed in less than a second, and the user places another order. An examination of the steps involved in these tasks underscores the need for speed, high availability, and scalability in a brokerage firm's trading system.

An order's lifecycle

- A trade order is received by the server and has been placed in a so-called incoming message queue.
- A number of parallel readers retrieve messages from this queue as soon as they arrive and place them into a database.
- The order is validated and is either rejected or saved in the database.
- The order is sent to a stock exchange (SE) using an application program interface (API) provided by the SE, which can also be placed in a message queue.
- The stock exchange sends back an order acknowledgement that is saved in the brokerage firm's database.
- Most market orders (at least the smaller ones) are executed almost immediately and the SE sends executions or rejections to the brokerage firm's dedicated queue(s).
- Executions or rejections are saved in the database and the user gets the appropriate notification.
- Since the brokerage industry has very strict regulations, some trades must be reported immediately to the third-party systems to avoid penalties.

Speed is the king on a round trip

A brokerage's firm's first goal is to design and implement a system that can support dozens of orders per second. Major brokerage companies place hundreds of thousands of trades per day and even slight delays lead to substantial losses. Its second goal is to create a scalable system that can be upgraded to support additional load without any modification of the source code. Finally, the firm must build a system that has loosely coupled components allowing easy configuration if the primary data center is lost.

Message-oriented middleware: the backbone of a trading system

Message-oriented middleware, or MOM, provides the transport and the storage to messages that travel from one system component to another. Messages are placed in **queues** by **message producers** and are retrieved by **message consumers**. A message can be as simple as "Buy 1000 shares of XYZ" or as complicated as dozens of megabytes of binary data. For example, a program running under a Web server sends a message to an application server hosting transactional components that apply business rules, store the data in DBMS, and forward the message to the stock exchange or a legacy application.

MOM enables communication between Visual Basic, Java, and Cobol applications with minimum programming. MOM servers can be configured to support guaranteed delivery, which is important because each lost or stalled order is a liability.

MOM supports various message modes:

- **Point-to-point (P2P)**. In this case, each message producer has one message consumer, e.g., the application puts the trade order in a queue and only one SE consumer reads the message.
- **Publish-subscribe (Pub-Sub)**. In this case, multiple subscribers can receive the same message, e.g., stock price quotes or some other market data feeds.

But one of the most important features of MOM is its ability to work in so-called **asynchronous mode**. In trading systems this means that, when a server places an order in a queue, it does not have to wait in a blocking mode until the order gets executed. For example, if the order placed is a limit order (buy/sell only at a certain price), the order may not be executed for hours. But when it does, the stock exchange will place the message about this trade into a dedicated queue and the reader program will receive it immediately and asynchronously.

Selecting a MOM provider

When choosing among MOM providers, it is important to consider:

- Does this provider support clustering, failover, and load balancing?
- Does this provider offer an API from the programming languages that are used in your organization?
- Is the API generic? This is important if the firm will decide to change the MOM provider in the future.
- What are the licensing costs? It's wise to check to see if the brokerage firm already has some licenses available.
- Are there any costs involved in establishing a team that will provide MOM servers administrative and technical support?
- How quickly can transactions be completed?

It is obviously critical that you balance speed, availability, and reliability in an operational trading environment.

The big picture

Figure 1 shows a sample design of a highly available trading system. It includes multiple clusters of servers running on various platforms and talking to each other using messaging.

The top-left corner of the diagram depicts the outside users that place an order using a Web interface, hence the presence of the Web server.

Internal brokers do not need to use the Web interface and can place their orders using rich clients running on their workstations.

All orders are routed by the Java (JEE) application server using messaging. MOM servers transport the messages between the Java server, legacy systems, and the stock exchange. Java application servers use multiple parallel readers, so-called message-driven beans, and senders to increase the message throughput.

Lightweight Directory Access Protocol (LDAP) servers are used to store all configuration properties of the system, including IP addresses of the JEE servers, names and properties of the message queue managers and the queues et al. LDAP servers are highly optimized for reads and also can substantially simplify reconfiguration of the entire system on the fly.

Database Management System (DBMS) servers are used to maintain all transactional and reference data. JEE application servers allow easy grouping of several actions (e.g., saving an order in a database and sending a message to an SE into one transaction, which can be reversed if necessary).

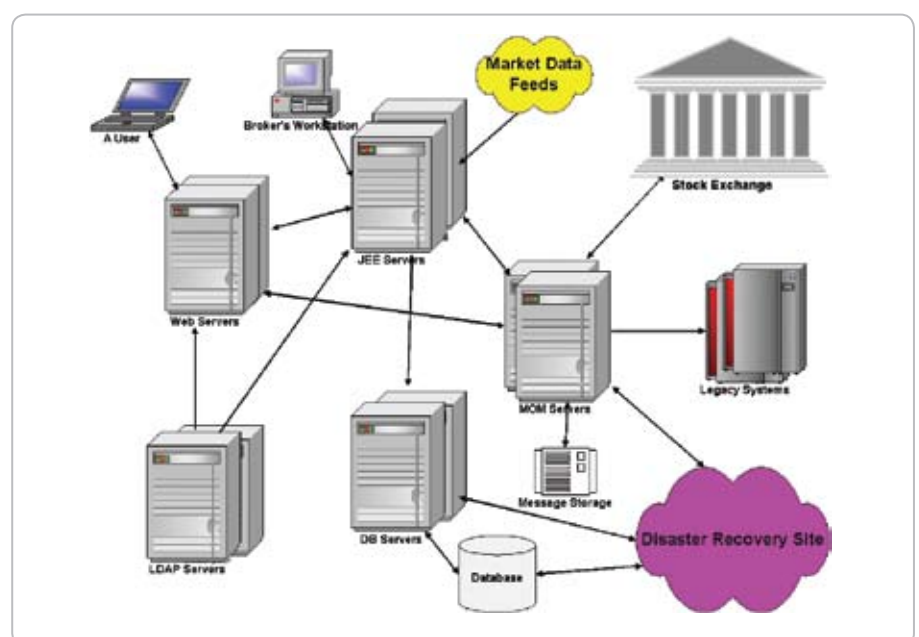


Figure 1. A Sample Design of a Multi-Tier Highly Available Trading System

Important design considerations

Software architects face many challenges while designing trading systems, for example:

- How many parallel threads of execution to use in the available hardware to actually improve performance?
- How to process corrupted messages?
- How to switch to the disaster recovery site with minimal or no downtime?
- How to create a system that will allow easy reconfiguration in anticipation of an important event that may dramatically increase the volume tomorrow and how to switch back the day after?
- Whether to use persistent or non-persistent messaging?
- Whether to use P2P or Pub-Sub mode?
- Is XML format a good choice for messages representing trade orders?
- What kind of message acknowledgments should be used?
- How to interface with compliance or other software deployed in your company?

There are no general answers to these questions. The safest approach is to work with seasoned architects and IT professionals who can design and develop a high-performance, scalable, and reliable trading system.



EMC Corporation
Hopkinton
Massachusetts
01748-9103
1-508-435-1000
In North America 1-866-464-7381
www.EMC.com