Design and Analysis of A Distributed Multi-leg Stock Trading System

Jia Zou[1], Gong Su[2], Arun Iyengar[2], Yu Yuan[1], Yi Ge[1]
[1]IBM Research – China; [2]IBM T. J. Watson Research Center
[1]{ jiazou, yuanyu, geyi }@cn.ibm.com; [2]{ gongsu, aruni }@us.ibm.com

Abstract: **We present the design, optimization and analysis of a highly flexible and efficient multi-leg stock trading system. Automated electronic multi-leg trading allows atomic processing of consolidated orders such as "Buy 200 shares of IBM *and* sell 100 shares of HPQ". While the expressive power of multi-leg trading brings significant value to investors, it also poses major challenges to stock exchange architecture design, due to additional complexities introduced in performance, tradability, and fairness. Performance can be significantly worse due to the need to coordinate transactions among multiple stocks at once. This paper studies the performance of multi-leg trading under different fairness constraints and variability in order price and order quantity. We identify the major performance bottlenecks when using traditional atomic commitment protocols such as 2-Phase Commit (2PC), and propose a new look-ahead algorithm to maximize transaction concurrency and minimize performance degradation. We have implemented a base-line 2PC prototype and a look-ahead optimized prototype on IBM z10 zSeries eServer mainframes. Our experimental results show that the look-ahead optimization can improve throughput by 58% and reduce latency by 30%.**

Keywords- computer-driven trading, distributed coordination, multi-leg trading, transaction processing, two-phase commit.

I. Introduction

Electronic stock and commodity trading has revolutionized financial markets. Major stock exchanges such as the NYSE and NASDAQ handle large volumes of requests electronically. These exchanges must handle high request rates, serve requests with low latencies, and be highly available. Because of the need for high performance, the systems are designed for requests involving single stocks and not requiring coordination among multiple stocks which can add considerable overhead.

There is considerable interest in multi-leg stock trading in which multiple stocks are traded atomically in the same transaction. To illustrate the problem, consider the following multi-leg order: "buy 200 shares of IBM at price ≤ \$130, sell 100 shares of HPQ at price ≥ \$30, and buy 300 shares of MSFT at price ≤ \$20." This multi-leg order will not trade unless stock prices for IBM, HPQ, and MSFT allow each order on an individual stock, known as a leg, to execute. If the multi-leg order is tradable, then all legs are executed atomically. The key issue is that in order to determine if the entire multi-leg order is tradable and to execute it atomically, trading on all three stocks has to be suspended for a period of time. This reduces performance considerably.

Currently, stock exchanges do not support automated multi-leg trading of this type due to the overhead and complexity. A transaction of this type would have to be executed by a human. Stock exchanges and investment banks are aware of the fact that true automatic multi-leg trading would bring significant business value if it could be efficiently implemented. Among other things, it will enable investors to submit and trade only one order rather than multiple related orders with significantly lower transaction cost and enhance existing financial products like ETF[21] or basket swap[22] by offering investors more flexibility.

This paper examines the complexities and performance problems associated with multi-leg trading, and presents a new algorithm called look-ahead for alleviating the high coordination overhead. Our look-ahead algorithm could be a key component of distributed systems which need to atomically coordinate buying and selling of stocks and other commodities. The optimizations that we present in this paper minimize the time during which trading on a stock comprising a multi-leg order is suspended.

In order to achieve scalability, stock exchanges use multiple nodes for trading in which different stocks might trade on different nodes. Therefore, processing a multi-leg order will often incur communication overhead for coordinating two or more nodes. Thus, multi-leg trading is typically significantly slower than single-leg trading. One way to solve the problem is to only allow the stock exchange to specify the stock and trading quantity of each leg and disallow mixed-trading between single-leg and multi-leg orders. That way, multi-leg orders could be processed independently from single-leg orders which would avoid slowing down single-leg orders. However, this would severely limit the flexibility and value of multi-leg trading. Given that mixed-trading between single-leg and multi-leg order is needed, performance of single-leg trading will also be slowed down by multi-leg trading.

Another issue is that different fairness of trading rules and variance of prices and quantities of requests will affect performance and tradability. Because tradability, fairness and performance are all important, trade-offs must be made among them in many aspects, e.g. trading rule design, financial product design, capacity planning, and overload control. It is thus important to understand their quantitative relationships.

We have implemented two stock trading systems. The first is a base-line prototype based on 2PC[1,2,[9] and efficient session management. The second uses our look-ahead algorithm to reduce the overhead of coordinating multi-leg transactions. The main idea is that even if stock trading on a symbol is suspended due to processing a multi-leg order, some single-leg orders can continue to execute if certain identified conditions can be satisfied. In this way, the look-ahead algorithm uses fine-grained state machines to schedule requests for maximum concurrency while maintaining fairness constraints. Our results show that, the proposed look-ahead solution can achieve up to **58%** throughput gain and up to **30%** latency reduction.

We also quantitatively evaluated a number of performance factors like percentage of multi-leg orders, number of legs, price/quantity variance and different priority ordering rules. We found that increasing the number of legs decreases performance and tradability. Increasing the percentage of multi-leg orders will decrease performance and tradability, but will not always decrease the throughput gain from the look-ahead optimization. Using priority ordering rules requiring more legs to be at the front of the priority queue before a multi-leg order can trade will decrease performance and tradability, but will increase throughput gain from the look-ahead optimization. Variance in price and quantity will decrease tradability, but increase performance, however, not always increase the throughput gain from the look-ahead optimization.

The key contributions of this paper include: i) Design and implementation of a highly flexible and efficient prototype multi-leg trading system; ii) A look-ahead algorithm which can speedup multi-leg trading significantly and is useful for a broad class of coordinated transaction systems with ordering constraints; iii) A detailed performance evaluation quantifying a number of performance factors, with useful insights for making the right trade-off in many aspects of multi-leg trading.

The organization of this paper is as follows. Section II introduces a common stock exchange architecture and defines the multi-leg trading problem. Section III presents the design and implementation of our base-line multi-leg trading system. Section IV describes the novel look-ahead optimization scheme. Section V provides a detailed performance evaluation and analysis. Section VI describes related work. Section VII concludes the paper and future works.

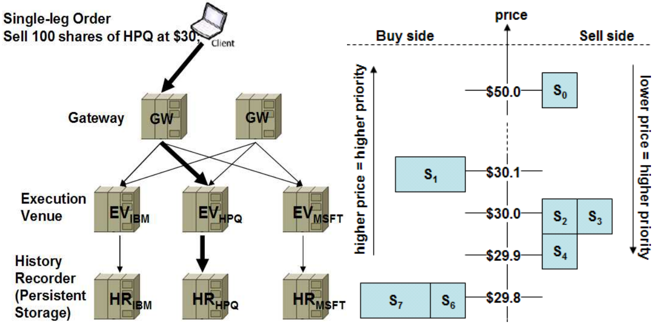
II. BACKGROUND

A. *Single-leg Trading and Stock Exchange Architecture*

It is beneficial to understand general stock exchange architecture and how orders are handled for single-leg trading. As shown in Fig. 1(a), the architecture is typically multi-tiered and consists of multiple gateway nodes (GW) running order dispatchers, multiple execution venue nodes (EV) matching orders for one or more specified stock symbols, and multiple history recorder (HR) nodes writing trading results into persistent storage. EVs are the most critical components of the stock exchange where all trades are executed.

A single-leg order is a buy or sell order involving only one stock symbol. GW nodes are responsible for dispatching received orders to corresponding EVs. The order matching process carried out by EVs should be strictly consistent with the following ***price-time priority rules***[9]: i). **price-priority rule:** for buy orders, higher price has higher priority; but for sell orders, lower price has higher priority; ii). **time-priority rule:** if multiple orders offer the same price, the order arriving earlier has a higher priority.

For example, when a GW node receives a single-leg order saying "Sell 100 shares of HPQ stock at $30.1", it will dispatch the order to the EV corresponding to HPQ for processing. When the EV receives the order, it matches the order against its *order book*, which is an in-memory list of all outstanding orders (those orders that didn't have a match since they were received). If a match is found, for example there is an order buying 300 shares of HPQ stock at $30.1, a trade of 100 shares of HPQ stock at $30.1 occurs. Otherwise, the order is entered into the order book. The fact of whether a trade happens or not is sent to the HR to write to persistent storage. The GW sends an acknowledgment to the client at the end of each transaction. Fig. 1(b) shows a typical order book of an EV. It has two sides: buy and sell. Orders on each side are first ordered by their price, and then ordered by their arrival time, according to the price-time rule. Readers should pay particular attention to the fact that the order book immediately gives the highest-to-lowest priority ordering for both sides: [S_1, S_6, S_7] for the buy side and [S_4, S_2, S_3, S_0] for the sell side (S_5 is not in the book because it was traded).



(a) Distributed Stock Exchange (b) Book Structure for Open Orders

Figure 1. Common Architecture for Single-leg Trading

B. *Multi-leg Trading Problem Definition*

Multi-leg (also known as bundled) stock trading allows the investors to submit multiple stock trading orders (legs) in a consolidated order, which will be executed atomically as a single transaction.[10] The consolidated order could be traded if and only if all legs can be traded. There are two large categories of multi-leg orders: fixed and arbitrary.

The fixed type only allows the stock exchange to specify the stock and trading quantity of each leg, comprising a basket of specified stocks with the amount of each stock in the basket also fixed. Investors can submit buy or sell orders to trade those baskets. Fixed multi-leg trading, i.e. ETF[21] or basket swap[22], is automated in many stock exchanges by treating the basket as a new stock. Baskets are only allowed to be traded among themselves and not with single-leg orders, which severely constrains market liquidity and trading flexibility.

The arbitrary type allows investors to specify the stock symbol, price limit and trading quantity for each leg, as illustrated in Table 1.

Table 1. A multi-leg order with three limit order legs

Leg no.	*Sequence no.*	*Symbol*	*Action*	*Price*	*Amount*
1	**0000-0002**	**IBM**	**Buy**	**$120.00**	**200**
2	**0000-0002**	**HPQ**	**Sell**	**$30.00**	**100**
3	**0000-0002**	**MSFT**	**Buy**	**$40.00**	**300**

Arbitrary multi-leg trading has not yet been automated in electronic stock exchanges due to the high coordination overhead. This is the problem we attempt to solve in this paper. The challenges are from three aspects of trading requirements: tradability, fairness, and performance.

i) Tradability. Tradability relates to how frequently orders are traded. It is one important metric because stock exchanges should guarantee that, at most times, there is enough supply and demand to allow prices to be determined consistently to attract buyers and sellers. In arbitrary multi-leg trading, the content of each leg is unknown *a priori*, so if multi-leg orders are only allowed to trade among themselves as practiced for the fixed type, the tradability will be low[11]. Therefore, it is necessary to support mixed trading of multi-leg orders with single-leg orders. In addition, smaller variance in price and quantity typically will also bring better tradability.

ii) Fairness. There is no existing priority ordering rule directly applicable to the situation where a multi-leg order can trade with a single-leg order, which is also a challenge this work is facing. But one intuitive principle is that the price-time priority rule should be maintained among all single-leg orders. In addition, fairness rules between a single-leg order and a multi-leg order, and those between two multi-leg orders are needed.

iii) Performance. A typical performance requirement for single-leg trading is to handle orders within 10 milliseconds, from the time a GW receives an order to the time it notifies the client that the order has completed. An EV should be able to handle thousands of orders per second. In addition, single-leg order performance should not be significantly compromised by introducing multi-leg orders. This is difficult to achieve because multi-leg trading can require blocking multiple EVs concurrently in order to determine if an order can be executed atomically.

Tradability, fairness, and performance also impact each other. For example, enforcing stricter fairness results in fewer orders being traded and more outstanding multi-leg orders, thus worse tradability and performance. Our goal is to understand the trade-offs among these factors and provide an efficient design and implementation of a multi-leg trading system for distributed stock exchanges.

III. Base-line Prototype

A. Trading Rule Design

Although it is straightforward to determine the trading priority ordering of single-leg orders, it becomes tricky to define the trading priority ordering of multi-leg orders. We propose a flexible priority rule for multi-leg orders that can be adjusted with a parameter to obtain desired degree of fairness. We call this the ***K-top-priority rule***:

> A multi-leg order with n total legs can be traded if and only if at least K legs ($1 \le K \le n$) have the highest priority and a tradable match in their respective order books.

In practice, two special cases are of particular interest. When K=n, a multi-leg order can be traded if and only if all the legs have the highest priority and a tradable match in their respective order books. This is called ***all-top-priority rule***. When K=1, a multi-leg order can be traded if and only if at least one leg has the highest priority and a tradable match in its order. This is called the ***single-top-priority rule***.

Intuitively, as K increases, the degree of fairness increases but the tradability decreases. For example, the all-top-priority rule is strictly consistent with the price-time priority rule in all EVs (fairness is guaranteed between all combinations of single-leg orders and multi-leg orders) and thus offers the best fairness. However, the tradability of this rule is also the worst because the probability of a multi-leg order getting traded is the lowest. Both the all-top-priority rule and single-top-priority rule are supported in our prototype and are compared in Section V.

B. Processing Flow

The stock symbol attached in each leg in arbitrary multi-leg orders can only be determined at run-time. The GW dispatches an incoming multi-leg order to all the legs' corresponding EVs. Use the example order in Table 1, the first leg will be dispatched to the EV handling IBM stock, the second leg will be dispatched to the EV handling HPQ stock, and so on, as shown in Fig. 2.

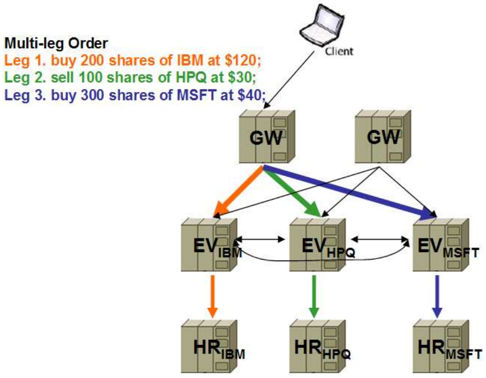

Multi-leg Order
Leg 1. buy 200 shares of IBM at $120;
Leg 2. sell 100 shares of HPQ at $30;
Leg 3. buy 300 shares of MSFT at $40;
Client
GW
GW
EV_IBM
EV_HPQ
EV_MSFT
HR_IBM
HR_HPQ
HR_MSFT

Figure 2. Dispatching of Multi-leg orders

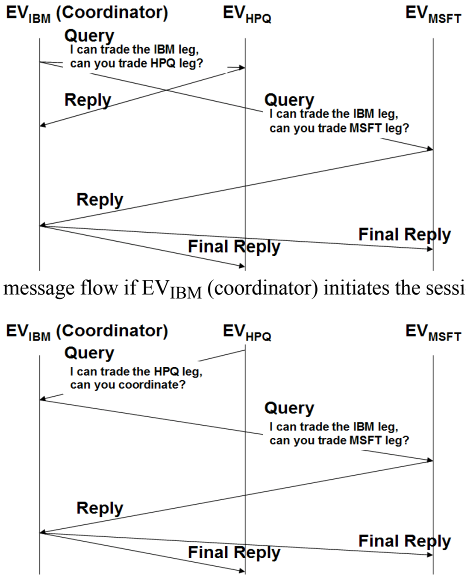
EVs process all incoming legs of multi-leg orders and single-leg orders in the order in which they are received. Each time, the EV will fetch one order from the queue for processing. The order will be matched against the order book as shown in Fig. 1(b) according to the price-time priority rule. If the order does not have a match, it will be inserted into the order book as an outstanding order and remain there until being triggered by later incoming orders.

Otherwise, if the order has a match, there could be the following possibilities: **i) Single vs. Single**: an incoming single-leg order matches an outstanding single-leg order; **ii) Single vs. Multi**: an incoming single-leg order matches an outstanding leg of a multi-leg order; **iii) Multi vs. Single**: an incoming leg of a multi-leg order matches an outstanding single-leg order; and **iv) Multi vs. Multi**: an incoming leg of a multi-leg order matches an outstanding leg of a multi-leg order in order book.

In the Single vs. Single case, the order and its match will be traded as described in Section II.A. In the Single vs. Multi and Multi vs. Single cases, one multi-leg order is involved. To maintain atomicity and ensure that a multi-leg order is traded only if all legs can be traded, the EV should coordinate with other EVs corresponding to other legs of the multi-leg order (also called partner EVs) to achieve a consensus on whether the trade should go through. During the time of coordination, in our base-line design, all involved EVs are locked and are not allowed to fetch and process new incoming orders from the receiving queue. This is to avoid violating the time-priority and price-priority rules. Locking all EVs for coordination is a simple but safe design for the base-line prototype. If the consensus is "trade", the multi-leg order will be traded and removed from the order book; otherwise, the multi-leg order will be inserted into the order book to wait to be triggered by later incoming orders. The "trade" or "not trade" information will be sent to an HR node to write to persistent storage. The Multi vs. Multi case can result in extremely complex coordination scenario due to the possibility of cascading of matches. Therefore, it is disallowed in our prototype.

C. Distributed Coordination Protocol Design

Once a leg of a multi-leg order becomes the top priority order in its EV, a match with an incoming single-leg order triggers a multi-leg session to coordinate with other legs' EVs to determine whether this multi-leg order could trade or not. Given a specified priority rule, coordination logic should define message flows to determine whether this multi-leg order can trade with other single-leg orders based on the status of each leg. This is very similar with the distributed consensus problem, which exists in systems like distributed database/file system, multicasting, streaming and so on. 2PC is a widely used distributed consensus protocol that decides on a series of Boolean values ("commit" or "abort"). In 2PC, the coordinator node communicates the state of a transaction to peer nodes. When the transaction state transitions to "prepare" at a peer node, the peer responds with a "yes" or "no" vote. The coordinator counts these responses; if all peers respond "yes" then the transaction commits. Otherwise it aborts. In our base-line prototype, we utilize 2PC for multi-leg coordination and the basic message flows are shown in Fig. 3(a) and Fig. 3(b).


EV_{IBM} (Coordinator)
EV_{HPQ}
EV_{MSFT}
Query
I can trade the IBM leg, can you trade HPQ leg?
Reply
Query
I can trade the IBM leg, can you trade MSFT leg?
Reply
Final Reply
Final Reply
EV_{IBM} (Coordinator)
EV_{HPQ}
EV_{MSFT}
Query
I can trade the HPQ leg, can you coordinate?
Query
I can trade the IBM leg, can you trade MSFT leg?
Reply
Final Reply
Final Reply

(a) The message flow if EV_{IBM} (coordinator) initiates the session

(b) The message flow if EV_{HPQ} (non-coordinator) initiates the session

Figure 3. Message Flows for Distributed Coordination

Fig. 3(a) is the case when a coordinator EV initiates a multi-leg session. The message flow is essentially the same as a traditional 2PC. Fig. 3(b) is the case when a non-coordinator EV_{HPQ} initiates a multi-leg session, which is different from a traditional 2PC where a session is always initiated by the coordinator. As shown in the figure, there are actually fewer messages involved in this case. Since the query from the non-coordinator EV_{HPQ} immediately tells the coordinator EV_{IBM} that its leg of the multi-leg order is tradable. Therefore, there is no need for EV_{IBM} to send a query to EV_{HPQ}.

3PC[2,19,27] and Paxos[3] are similar consensus protocols but can handle more failure situations and are more adaptive to unreliable massive distributed systems. After coordination, if all involved EVs agree to "trade", all legs will be removed from their respective books and traded; otherwise, all legs will remain in their respective order books. The reasons we introduce the coordinator into multi-leg sessions include: i) without a coordinator, many message transmissions will be redundant and incur unnecessary communication overhead; ii) without a coordinator, it will be more difficult to mark the global end of a distributed coordination session.

D. Architectural Design Considerations

Now we discuss various architectural design decisions and optimizations for implementing an efficient multi-threaded EV system supporting highly flexible and efficient multi-leg trading, as shown in Fig. 4.

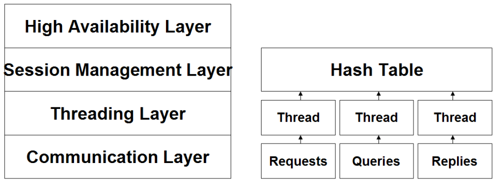


Figure 4. Layered design in EV

1. Messaging and Threading. We use Websphere MQ Low Latency Messaging (LLM)[4], an IBM product, for messaging services among GW, EV, and HR nodes. LLM provides high performance unicast and multicast messaging services with reliable and ordered delivery of messages. It uses a publish-subscribe model in which a topic corresponds to a stream of messages. We implement a multi-threading model with a scheduler dispatching messages of different topics (Requests, Queries or Replies/Final replies) to different threads respectively. The states of order books and on-going multi-leg sessions are stored in shared memory and could be accessed by different types of threads concurrently.

2. Session Management. To support fine-grained processing and maximum parallelism, we need an efficient mechanism to store the states of all the multi-leg sessions in each EV and make sure that all EVs have consistent session data for each multi-leg order. We designed an EV-specific hash-table for this purpose. The hashing index is computed on the global unique sequence number of multi-leg orders. Each hash bucket contains a list of cells holding a structure representing a multi-leg session. Each cell structure contains a mutex object for synchronization and a log facility for deadlock debugging. The session management is consistent with the 2PC coordination protocol.

In addition, the session management uses various optimizations to minimize message communication overhead**.**

For example, if the coordinator finds a queried leg can not trade, it will send a negative final reply to the initiator of the query and terminate the session, instead of sending queries to all other EVs, because according to the "all or nothing" atomicity of multi-leg order definition, if one leg can not trade, the entire multi-leg order can not trade. A similar optimization is that if the coordinator receives a negative reply, it will send final replies immediately instead of waiting for all replies. Another type of optimization is based on inferring the content of replies from received queries. For example, EV_A is waiting for EV_B's decision about a multi-leg order M_1, while EV_B is waiting for EV_A for the same multi-leg order. However, because EV_A will send queries to EV_B if and only if EV_A can trade M_1, EV_B can infer that the trading result of M_1 at EV_A must be positive. Similarly, EV_A can infer the trading result of M_1 at EV_B to be positive. Thus, both EV_A and EV_B can stop waiting and continue processing, reducing unnecessary waiting.

3. Deadlock Avoidance and Lock Granularity Tuning. As with all distributed systems, care must be taken to avoid deadlocks. There are two types of deadlock in our base-line prototype using 2PC. One is cross-EV deadlocks, e.g. EV_A sends a query to EV_B and waits for EV_B's reply, EV_B sends a query to EV_C and waits for EV_C's reply, and EV_C sends a query to EV_A and waits for EV_A's reply. This will form a deadlock loop if each EV has a single thread to handle all message types. As mentioned in messaging and threading, our prototype uses a separate thread and topic for each message type, and thus avoids this deadlock.

The other type of deadlock situation is caused by fine-grained locking inside an EV. We designed fine-grained locks for optimal performance, e.g. locks on order books, session tables, and other shared state, etc. In addition, we also have locks on individual hash table buckets instead of the whole hash table. Deadlock due to two threads have different locking sequences is well-known and is avoided by enforcing consistent locking sequences by all threads.

4. Primary-Primary High Availability. High availability is also an important requirement for stock exchanges. In our prototype, we achieve continuous availability by using the primary-primary architecture described in Su and Iyengar.[6]. The basic idea is to maintain a mirror of each EV through the use of a reliable shared memory called Coupling Facility[7] and an efficient total ordering algorithm. The mirror allows non-disruptive failover of any single EV failure. Interested readers are referred to the primary-primary paper[6] for more details.

E. Analysis of Fairness and Performance

Fairness Analysis. From each EV's local point of view, the base-line design of the multi-leg trading prototype satisfies the following fairness properties:

i) **Single-Single constraint.** A higher priority single-leg order always trades before a lower priority single-leg order. This is based on the observation that on the arrival of a single-leg order, one of the following happens:

- It has the highest priority, and it may or may not match the highest priority order on the opposite side
- It does not have the highest priority, and it is not allowed to trade

It is clear that when a trade does occur, it is always between the two highest priority orders on the opposite sides.

ii) **Multi-Multi constraint.** A higher priority multi-leg order always trades before a lower priority multi-leg order. This is not difficult to see because each multi-leg order is traded logically the same way as a single-leg order by blocking all EVs involved. Thus, the argument for the single-single constraint above applies here equally.

iii) **Single-Multi constraint.** If the ***all-top-priority*** rule is adopted, then a higher priority single-leg order always trades before a lower priority multi-leg order. The all-top-priority rule dictates that all legs of a multi-leg order must have the highest priority for it to be tradable. Therefore, by definition, all higher priority single-leg orders must have been traded before the multi-leg order becomes tradable.

Performance Analysis. Compared with single-leg trading, the major overhead incurred by multi-leg trading is in coordinating a consensus, during which all involved EVs are blocked as described in Section III.B. In each multi-leg session, there are three different EV roles: ***initiator*** which is the first EV sending out a query and initiating the multi-leg session; ***coordinator*** which is specified by the GW for the multi-leg order; and ***participants*** which represent all other EVs corresponding to other legs of the multi-leg order. The blocking time for the initiator is defined as the elapsed time from the time it sends out a query until the time it receives a final reply. The latency for the coordinator is defined as the elapsed time from the time it sends out queries on behalf of the initiator until it sends out the final replies. The blocking time for a participant is defined as the elapsed time from the time it receives a query sent by the coordinator until the time it receives a final reply. Although the session-level optimizations described in Section III.D optimize the blocking time for the base-line prototype, the performance degrades significantly due to the blocking time introduced by multi-leg trading, as shown in Fig. 11-13 in Section VI. Therefore, more sophisticated solutions are required to further reduce the blocking time.

IV. LOOK-AHEAD OPTIMIZATION

A. Main Idea of Look-ahead Algorithm

In Section III, we identified that a major performance overhead of the base-line prototype is in coordinating a consensus: trade or not trade, for a Multi vs. Single or Single vs. Multi trade. A solution is to allow incoming orders to continue being processed before the coordinator sends final replies, which we called ***look-ahead***. However it is not easy to determine which orders should be allowed to trade or not in the blocking duration. That's because the final tradability of the multi-leg order is unknown until the end of the multi-leg session and the priority ordering constraint must be enforced.

To preserve consensus atomicity, the commitment that "the blocked multi-leg order can trade in EV_{HPQ}" should not be compromised in the blocking duration before it receives its final reply (for non-coordinators) or before it sends out the final reply (for the coordinator). We find that as long as the above commitment could be kept (which means at least one single-leg order remaining in the order book could be traded with the blocked order), allowing more orders to be traded in the blocking duration will neither affect the tradability of the multi-leg order nor violate its priority ordering (the proof is given in Section V.B). Based on this finding, a look-ahead algorithm is designed, which allows orders in the order book to trade freely in blocking duration as if the blocked multi-leg order doesn't exist, if no conflicting condition is detected as violation of the commitment.

To implement the solution, we classify incoming orders into different types based on whether allowing their trade will violate the tradability of the blocked multi-leg order. We also designed one state machine to keep track of the order's transition from one type to another. We use an example to illustrate the idea.

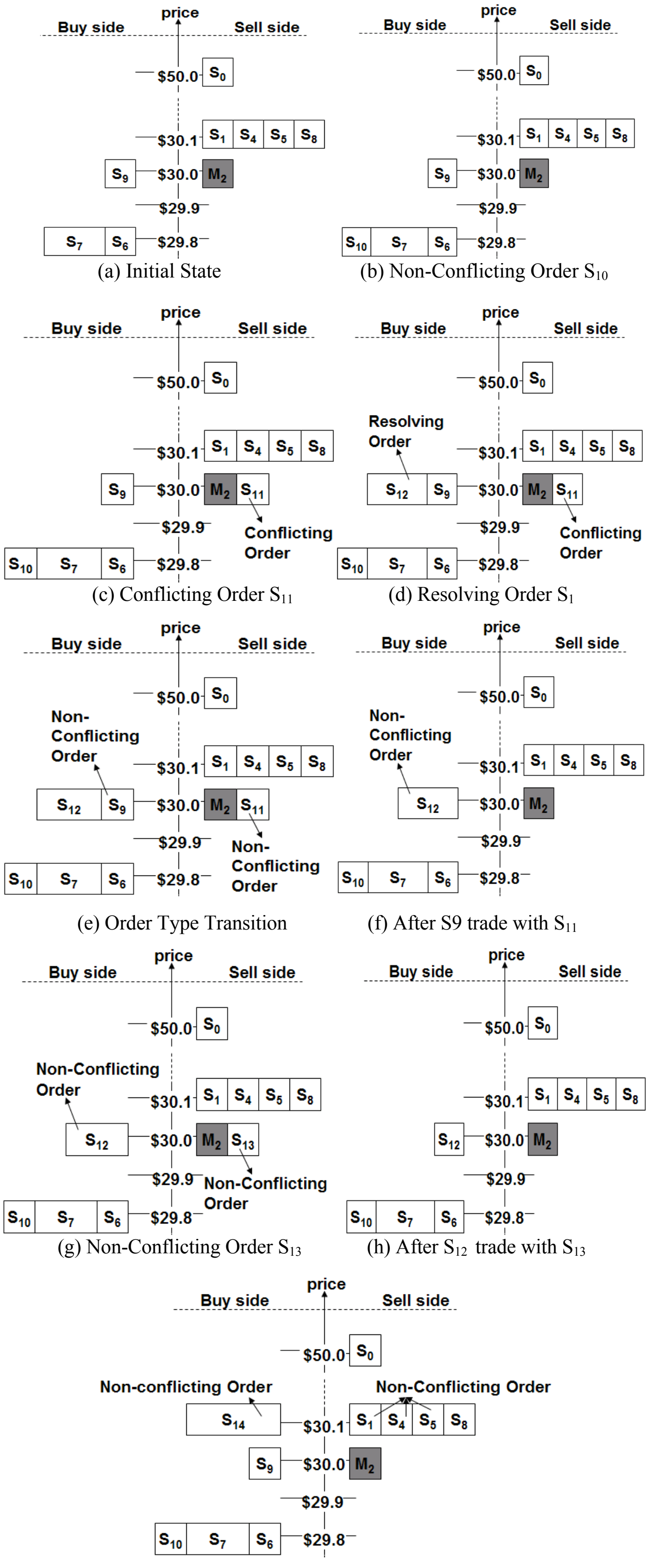
Fig. 5 shows the order book described in Section II.A. Each block represents an order, the subscript represents the arriving sequence, S represents the order is single-leg, M represents the order is a multi-leg and the length of the block represents the quantity of this order. Fig. 5(a) shows the order book of an EV when an incoming single-leg order S_9 matches an outstanding multi-leg order M_2, which triggers the multi-leg session and look-ahead. Readers should observe that it is clear from the order book that S_9 and M_2 are the highest priority buy and sell order, respectively. In Fig. 5(b), S_{10} arrives and is classified as a ***non-conflicting*** order since it has no match and thus will not affect M_2's tradability.

Next in Fig. 5(c), S_{11} arrives and is classified as a ***conflicting order***, because S_{11} matches S_9 but allowing S_{11} to trade with S_9 would violate the tradability of M_2. So we cannot allow S_{11} and S_9 to trade and we continue to look ahead. In Fig. 5(d), S_{12} arrives and is classified as a ***resolving order*** because it provides more buying quantity at the same price as S_9 does. As a result, if we were to allow S_{11} to trade with S_9, M_2 would still have S_{12} to trade with. So the tradability of M_2 is not violated.

In Fig. 5(e), S_{11} transitions to non-conflicting orders. In Fig. 5(f), S_{11} trades with S_9 and both disappear from the order book; S_{12} transitions to a non-conflicting order. Next in Fig. 5(g), S_{13} arrives and is classified as a non-conflicting order because S_{12} has enough buying quantity to satisfy both M_2 and S_{13}. Therefore, S_{13} is allowed to trade with S_{12}, leaving enough quantity for M_2, as shown in Fig. 5(h). Finally, in Fig. 5(i), S_{14} arrives and is classified as non-conflicting because S_{14} can trade with S_1, S_4, and S_5 without violating the tradability of M_2.

The look-ahead process continues until either the consensus session has ended, or a ***blocking order*** is encountered. A blocking order is a tradable multi-leg order that is on the opposite side of the current blocking multi-leg order. Simultaneous look-ahead with two blocking multi-leg orders on the opposite side of the order book will make the quantity on both sides unpredictable. Therefore, an EV stops the look-ahead and goes to the state of blocking when a blocking order is encountered.

The benefit of look-ahead is illustrated in Fig. 6. For readers interested in more details of our solution, we present the order type state transition and EV state transition diagrams in Fig. 7 and 8. We also present the look-ahead algorithm pseudo code in Fig. 9.



(a) Initial State

(b) Non-Conflicting Order S_{10}

(c) Conflicting Order S_{11}

(d) Resolving Order S_1

(e) Order Type Transition

(f) After S9 trade with S_{11}

(g) Non-Conflicting Order S_{13}

(h) After S_{12} trade with S_{13}

(i) Non-Conflicting Order S_{14}

Figure 5. Running Example to Illustrate Look-ahead

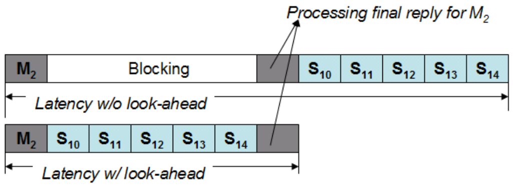
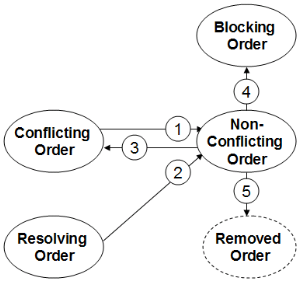
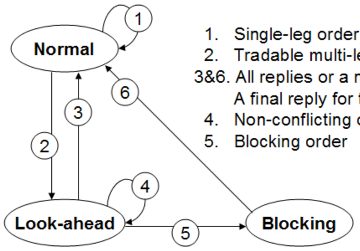


Figure 6. Performance benefits of Look-ahead



1. On arrival of a resolving order
2. After resolvable conflicting orders are all traded
3. Being matched by a non-conflicting order which is also the blocked order's match
4. The non-conflicting multi-leg order at the opposite side becomes tradable
5. The non-conflicting order is traded.

Figure 7. Order type state transition



1. Single-leg order, and not tradable multi-leg order
2. Tradable multi-leg order, and tradable query

3&6. All replies or a negative reply for the coordinator
A final reply for the non-coordinator

4. Non-conflicting order, conflicting order and resolving order
5. Blocking order

```
Parameters: new_order, blocked_order
WHILE (new_order=recv() AND STATE==LOOKAHEAD) {
  new_order.type=get_order_type(new_order);
  SWITCH (new_order.type) {
    CASE NON_CONFLICTING:
      trade with matchable non-conflicting orders based on priority;
      IF  new_order.remaining_quantities>0
          insert new_order into book;
      END
     BREAK;
    CASE BLOCKING:
      block();
      STATE=NORMAL;
      BREAK;
    CASE CONFLICTING:
       insert new_order into the book;
       BREAK;
    CASE RESOLVING:
       update the type of resolvable conflicting orders to non-conflicting;
       trade tradable non-conflicting orders based on priority rule;
        new_order.type=NON_CONFLICTING;
        trade with matchable non-conflicting orders based on priority rule;
        IF new_order.remaining_quantities IS NOT NULL
             insert new_order into book;
        END
        update the type of non-conflicting orders and take actions;
        BREAK; }
  IF final-reply received
        STATE=NORMAL
        IF final-reply is negative
              rematching blocked_order's current matching orders in book;
       ELSE
             trade blocked_order with its current matching order;
       END
  END }
```

Figure 9. Lookahead Algorithm Framework

B. *Correctness and Fairness Proof*

The concept of "being traded" for a multi-leg order's leg could be extended to "being guaranteed to have a match". Thus the Multi-Multi constraint from one EV's local view of point can be extended to "a higher priority multi-leg order is always guaranteed to having a match before a lower priority multi-leg order is guaranteed to have a match". The Single-Multi constraint can be extended in the same way. Such extension of fairness constraints will not degrade fairness: for a blocked leg, it is not as important when a multi-leg order can conclude a trade as when the multi-leg order's match is guaranteed, because in either case its tradability is decided by other legs' tradability which can not be affected by the priority ordering in local EV. The extended fairness constraint is the basis of our look-ahead optimization.

Look-ahead algorithm guarantees that no conflicting orders are traded during a blocking duration, because as shown in Fig. 7, trades concluded in the blocking duration can only happen between two ***non-conflicting orders***. Therefore, the time a blocked order guaranteed for a match is always before the time any order gets traded in the blocking duration.

The **Single-Single constraint** defined in Section III.C is maintained for the duration of the look-ahead algorithm. That is because blocked multi-leg orders will not lock single-leg orders matched at the opposite side (Our definition guarantees that a blocking order, which will stop the look-ahead, is only possible on the opposite side of the blocked orders; all blocked orders are on the same side.), and single-leg orders can always trade freely if the trade is allowed by the priority rule and if there are enough lower priority single-leg orders satisfying all blocked multi-leg orders. Therefore, no single-leg orders will be compromised by any lower-priority single-leg orders.

The **Multi-Multi fairness constraint** is maintained for the duration of the look-ahead state. First, the algorithm guarantees that a higher priority multi-leg order MUST be triggered for a multi-leg session earlier than a lower priority multi-leg order. Secondly, although lower priority multi-leg orders might get traded before the blocked multi-leg order during look-ahead when multiple multi-leg orders are blocked, it will not compromise the blocked multi-leg order's fairness due to the extended fairness constraints and a lower priority multi-leg order is only allowed to get traded if the EV guarantees that after it is traded, there is still enough quantity for the blocked multi-leg order to find a match in the book. Otherwise, the lower priority order is not allowed to get traded until this conflicting condition is resolved.

Particularly, if the all-top-priority rule is implemented, the **Single-Multi constraint** is maintained because no lower-priority multi-leg orders can jump the queue and trade before higher priority single-leg orders due to the semantics of the all-top-priority rule, and on the other side, single-leg orders will not be locked by any blocked multi-leg orders.

V. Performance Evaluation and Analysis

A. Environment Setup

We have implemented both the base-line prototype described in Section III and an optimized prototype based on the look-ahead algorithm described in Section IV on an IBM z-series z10 eServer mainframe[7], which is a NUMA system having 64 4.4GHz CPU cores and 1.5TB memories. We virtualized 8 LPARs (LPAR refers to virtualized partition via hypervisor[5]) running z/OS on the mainframe with each LPAR holding 8 CPU cores. Each GW and EV node runs in a separate LPAR. All three HR nodes run in one LPAR for performance reasons. All cross-LPAR communication is via UDP/IP over hypersocket[8], which is a network stack implemented upon shared memory instead of Network Interface Cards, as shown in Fig. 10.

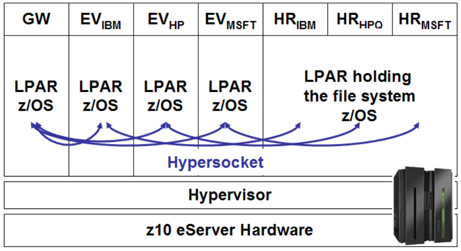


Figure 10. Experimental Environment

Each EV can handle multiple symbols with each symbol representing a type of stock. But in this paper's experiments, we made each EV only process one symbol to test the maximum per-symbol throughput which is one of the most important performance metrics for stock trading. We deployed load generators on one GW node which generates and dispatches requests to each EV as fast as possible.

We measured the following metrics: i) ***Throughput:*** the maximal number of requests (including single-leg orders and multi-leg orders) one system can process per second; ii) ***Gateway-to-gateway latency:*** the time elapsed from the point when the GW receives a message from the client (load generator) to the point when the request completes processing and an ACK message is sent back to the client by the GW; iii) ***Blocking time***: the average blocking time per multi-leg session measured on one EV in one minute. Blocking time is defined in Section III.E; iv) ***Trading ratio***: the ratio of traded multi-leg sessions (i.e. multi-leg sessions ending with a "trade" final reply generated by the coordinator) to the total number of multi-leg sessions. Each test run lasted at least a minute, and all above metrics were computed as an average of ten measurements.

B. Evaluation of Different Performance Factors

The parameters we can control in the test are illustrated in Table 2. Each of them represents a factor impacting the performance of multi-leg trading. Understanding how those factors influence performance and tradability is important to help stock exchanges to make the right trade-offs in designing their systems. In all tests, buy and sell orders were balanced with about the same number of each.

Table 2. Controlling parameters

Parameters	*Description*
multi-leg order percentage	the ratio of multi-leg requests in input requests
number of legs	the number of legs of a multi-leg request
fairness of priority rule	all-top-priority or single-top-priority
price/quantity variance	the price and quantity of multi-leg orders are uniformly distributed over a range; we varied these ranges

Multi-leg Order Percentage and Number of Legs. Fig.11-13 measure throughput, latency and blocking time for the base-line prototype and the optimized prototype with look-ahead. Single-top priority is used as the priority rule, and the price and quantity are uniformly distributed over the ranges [100, 105] and [1,100].

Fig. 11 and Fig 12 illustrate that the average per-symbol throughput and latency decrease significantly when the percentage of multi-leg orders increases. This is because the number of multi-leg sessions triggered also increases and causes the total blocking time to increase (This is indicated in Fig. 14(a) and Fig. 15(a) which show the average number of multi-leg sessions triggered from processing 50,000 requests). We also observe from Fig. 11 and Fig. 12 that the performance of 3-leg trading is significantly worse than 2-leg trading. There are two reasons: i) 3-leg multi-leg orders will involve more messages in a multi-leg session on average, causing longer blocking time per multi-leg session as shown in Fig. 13; ii) there will be more multi-leg sessions triggered for the 3-leg case than the 2-leg case (Fig. 14(a)), that is mainly because at a given time point, the probability of at least one EV triggering a multi-leg session in the 3-leg case is larger than in the 2-leg case.

As shown in Fig. 11 and Fig. 12, we observed up to 58% throughput gain with up to 30% reduction in latency. The figures also show that when the multi-leg order percentage is smaller than 20%, the performance gain after applying look-ahead will increase when the multi-leg order percentage increases. This is because more multi-leg sessions can be optimized by look-ahead. But when the percentage is larger than 20%, the performance gain will decrease when the multi-leg order percentage increases. This is because of the increasing percentage of blocking orders and decreasing interval between two blocking orders, both of which prevent the EV from staying in the look-ahead state. Fig. 13(a) measures the average of average blocking time measured in each EV, and Fig. 13(b) illustrates the aggregate of average blocking time measured in each EV. Both of them illustrates the significant reduction in average blocking time per multi-leg session after applying the look-ahead optimization, which shows how look-ahead optimizes performance.

As shown in Fig. 14(b) and Fig. 15(b), the trading ratio will decrease when the multi-leg order percentage increases. This is because when the multi-leg order percentage increases, the single-leg order percentage will decrease preventing multi-leg orders from finding a single-leg order match. As shown in Fig. 14(b), the tradability for the 3-leg case is significantly lower than for the 2-leg case because the probability that all EVs can trade a multi-leg order at the same time for the 3-leg case is significantly lower than for the 2-leg case.

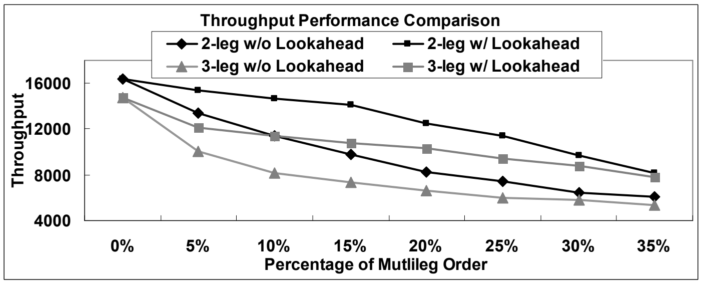


Figure 11. Throughput with Different Multi-leg Order Percentage

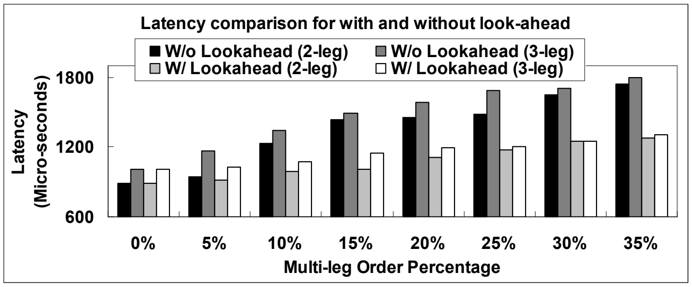
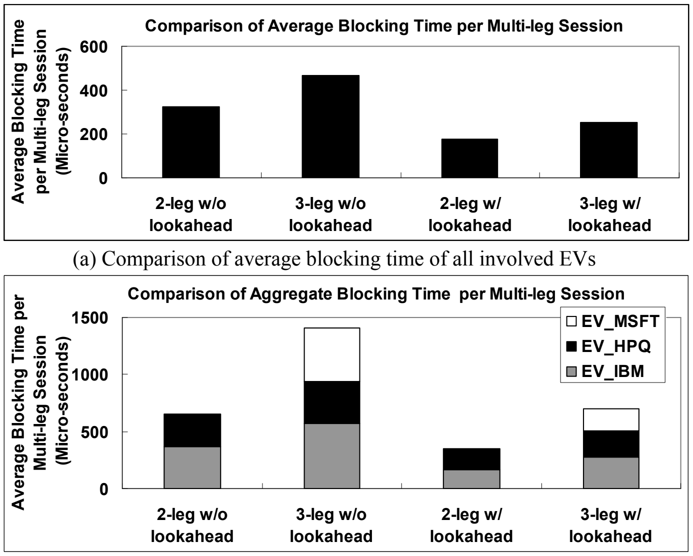


Figure 12. Latency with Different Multi-leg Order Percentage



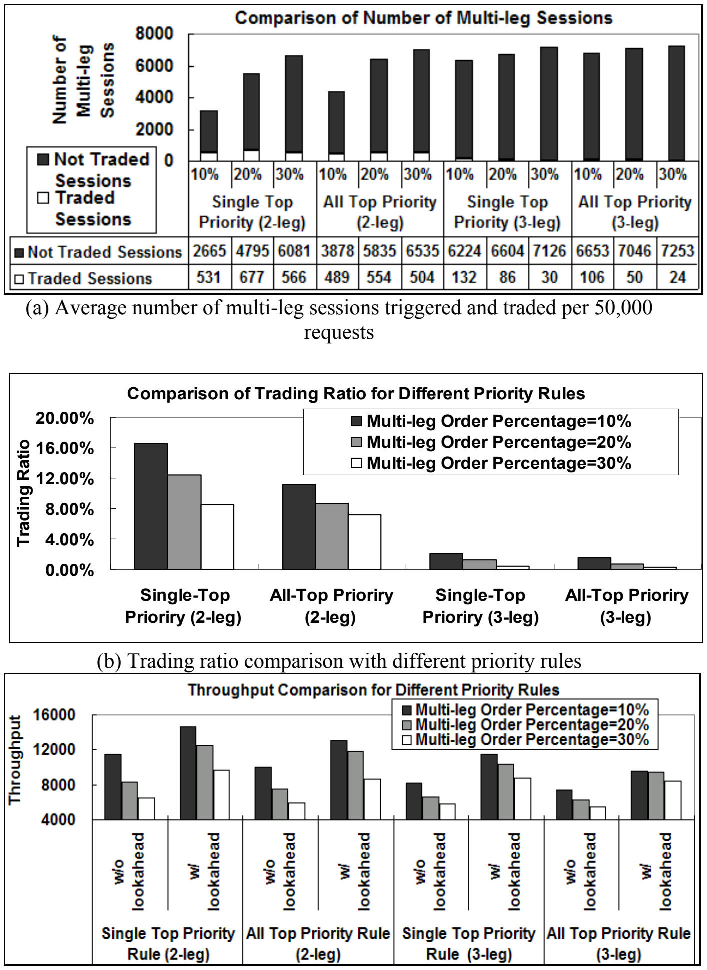
(a) Comparison of average blocking time of all involved EVs

(b) Comparison of aggregate blocking time of all involved EVs

Figure 13. Blocking Time Measurements

Priority Rules. Fig. 14 compares the throughput and trading ratios for different priority rules with varying multi-leg percentages (i.e. 10%, 20% and 30%) and number of legs. The price and quantity are randomly distributed over the ranges [100, 105] and [1,100] respectively. Fig. 14(a) shows that the all-top-priority rule will trigger more multi-leg sessions than the single-top-priority rule given the same percentage of multi-leg orders and same number of legs. This is because the trading ratio is smaller for the all-top-priority rule than for the single-top-priority rule (as shown in Fig. 14(b)). The smaller trading ratio will cause more multi-leg orders to remain in the order book, which increases the total number of multi-leg sessions triggered per unit time. As a result of having to handle more multi-leg sessions, the throughput for the all-top-priority case becomes worse than that of the single-top-priority case as shown in Fig. 14(c).

Fig. 14(c) also illustrates that look-ahead can improve throughput for both priority rules but provides more improvement for the all-top-priority rule. This is because more multi-leg sessions can be optimized by look-ahead for the all-top-priority rule.



	Single Top Priority (2-leg)			All Top Priority (2-leg)			Single Top Priority (3-leg)			All Top Priority (3-leg)		
	10%	20%	30%	10%	20%	30%	10%	20%	30%	10%	20%	30%
■ Not Traded Sessions	2665	4795	6081	3878	5835	6535	6224	6604	7126	6653	7046	7253
□ Traded Sessions	531	677	566	489	554	504	132	86	30	106	50	24

(a) Average number of multi-leg sessions triggered and traded per 50,000 requests

(b) Trading ratio comparison with different priority rules

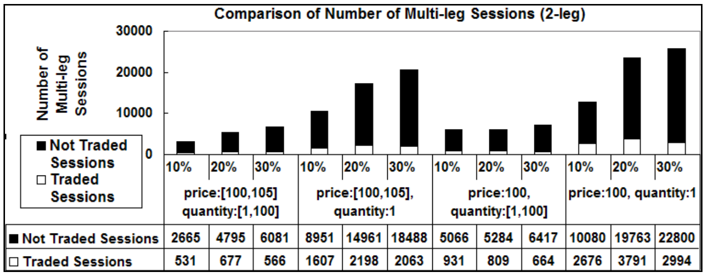
(c) Throughput comparison with different priority rules

Figure 14. Impact of Different Fairness Rules

Price/Quantity Variance. Price and quantity are uniformly distributed over a range. As described in Table 2, we also did experiments varying the ranges over which price and quantity are distributed. Fig. 15 compares the throughput and trading ratio for different variances in price and quantity with varying multi-leg percentage (i.e. 10%, 20% and 30%). Single-top-priority is used which is demonstrated to have better performance and tradability. Fig. 15(a) and Fig. 15(b) compare the number of multi-leg sessions triggered and trading ratio with different multi-leg order percentages for varying price and quantity ranges. It shows that the trading ratio increases significantly when the variance decreases.

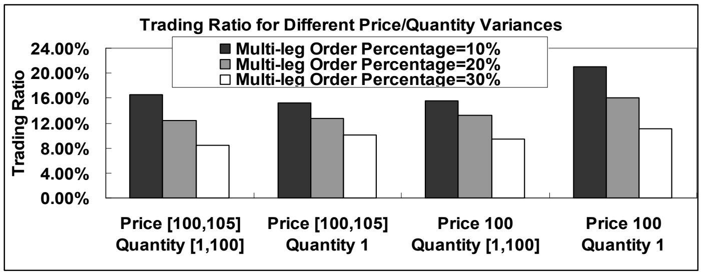
Fig.15(c) compares the throughput while varying multi-leg order percentages and the price and quantity ranges. It shows that decreasing variance significantly degrades throughput. This is because lower price/quantity variance will increase the probability of a multi-leg order finding a match in a book, causing more multi-leg sessions to be triggered as shown in Fig. 15(a). Fig 15(c) also illustrates that look-ahead can benefit performance for different variances in price and quantity except for the rightmost case with constant price and quantity and 30% of multi-leg orders. That is because the number of multi-leg sessions for this case increases significantly as shown in Fig. 15(a). That will cause the ratio of blocking orders to significantly increase and the time spent in the state of look-ahead minimized.

Since quantity often has larger variance than price, we further studied how throughput varies with the size of the quantity range as shown in Fig. 15(d). It shows that when the multi-leg order percentage is 20%, the throughput gain brought by look-ahead will first decrease when the variance increases, which is due to the number of multi-leg session has been reduced causing less opportunity for look-ahead optimization. Then when the size of quantity range increase to larger than 8, the throughput gain will increase when the variances increase, because the number of blocking orders will be reduced as to the reduction in the probability that a n multi-leg order can find a match in its respective EV. This trend is also valid for other percentages of multi-leg orders.

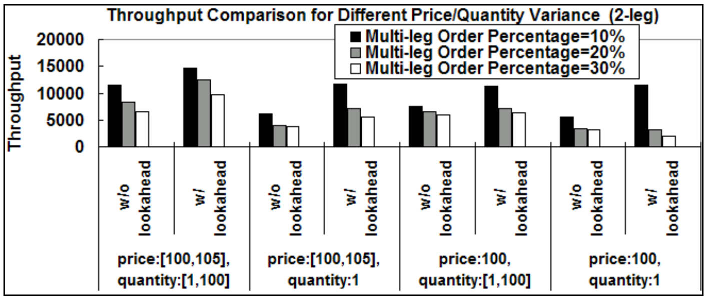


	10%	20%	30%	10%	20%	30%	10%	20%	30%	10%	20%	30%
	price:[100,105] quantity:[1,100]			price:[100,105], quantity:1			price:100, quantity:[1,100]			price:100, quantity:1		
■ Not Traded Sessions	2665	4795	6081	8951	14961	18488	5066	5284	6417	10080	19763	22800
□ Traded Sessions	531	677	566	1607	2198	2063	931	809	664	2676	3791	2994

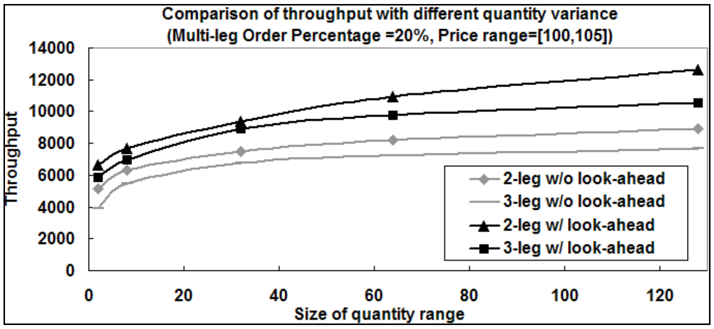
(a) Average number of multi-leg sessions triggered and traded per 50,000 requests



(b) Trading ratio comparison with varying price/quantity ranges



(c) Throughput comparison with varying price/quantity ranges



(d) Throughput comparison with varying size of quantity range

Figure 15. Impact of Quantity Variance

C. *Summary of Results*

Table 3 provides a summary of the results we obtained: ***look-ahead gain*** represents the throughput improvement after using look-ahead optimization; for ***fairness of priority rules***, the all-top-priority rule has higher fairness while the single-top-priority rule has lower fairness; "+" indicates that the two connected factors are directly related, "-" indicates that they are inversely related, "+-" indicates the two factors are directly related first and inversely related later, and "-+" indicates the two factors are inversely related first and directly related later. Important observations include the following:

- Increasing the number of legs decreases performance and tradability, but increases the throughput gain from the look-ahead optimization;
- Increasing the percentage of multi-leg orders will decrease performance and tradability, but will not always decrease the throughput gain from look-ahead optimization;
- Using priority ordering rule with better fairness will decrease performance and tradability, but will increase throughput gain from look-ahead optimization;
- Variance in price and quantity will decrease tradability, but increase performance, however, not always increase the throughput gain from the look-ahead optimization.

Table 3. Summary of Observations

	number of multi-leg sessions	*trading ratio*	*through -put*	*look-ahead gain*
number of legs	+	-	-	+
percentage of multi-leg order	+	-	-	+-
fairness of priority rules	+	-	-	+
price/quantity variance	-	-	+	-+

VI. Related Work

A. Similar Applications with Multi-leg Trading

Existing electronic stock and commodity exchanges do not support multi-leg trades in an automated fashion [10]. Many exchanges and investment banks support a similar idea known as ETF[21] and basket swap[22] which treat a basket of stocks as one stock. Thus, trading the basket of stocks is just like trading a single stock. However, it is actually a fixed type of multi-leg trading, and the number of stocks and quantity to trade are strictly constrained. Bundle trading[11~13] , proposed in the early 1990s, is a similar concept. However, existing research focuses on how to model the matching problem to maximize the market surplus by using mathematical optimization approaches assuming a centralized book. These references do not address the problem of applying bundled trading to distributed stock exchange architectures, which have significant synchronization overhead for multi-leg trading.

Combinatorial auctions[14~16] study mechanisms for combinatorial bidding by enabling the agents to express their preferences for bundles of items rather than individual items. However, they are focused on how to match supply to demand so that the market surplus can be optimized, while our work is based on existing stock exchange trading rules and how to minimize the blocking time introduced by trading multi-leg orders in a distributed architecture.

B. Coordination Transaction Processing

Distributed Consensus. In traditional transaction processing [18] and distributed database systems,[1] there are 2-phase commit protocols[1,2,19], 3-phase commit protocols[1,2,19,27] and Paxos[3], which are designed for distributed sites to achieve consensus in a cooperative way. ExchangeGuard[25] proposes a consensus protocol with high reliability and fairness for distributed exchanges. Such protocols are focused on reliability and failure recovery. By contrast, our work is mainly focused on maximizing parallelism within ordering constraints enforced by stock trading rules. We use a look-ahead approach to continually process later transactions before earlier transactions without compromising fairness constraints. Our work can be combined with those existing distributed consensus protocols to provide better parallelism and performance for similar systems.

Parallelism and Concurrency Control in Distributed Database. Concurrency control tries to exploit the parallelism of concurrent transactions while keeping data integrity. A number of existing works[17,23,24,28,29,32] study the concurrency constraints enforced by lock synchronization for accessing shared data. In OPT [23], which is a variant of 2PC, transactions are permitted to "borrow" dirty data while a transaction is in the prepared phase. Jones and Abadi[17] proposed a similar speculative approach. In those works, if the speculation is incorrect, all following transactions processed in the blocking time have to roll back. Reddy and Kitsuregawa[24] proposed speculative locking, where a transaction processes both the "before" and "after" version for modified data items. At commit, the correct execution is selected and applied by tracking data dependencies between transactions. By contrast, our look-ahead approach is not based on speculation and need not roll back or keep track of the data dependencies, which is one important advantage. Another difference is that existing concurrency optimization schemes are not aware of the fairness constraints of multi-leg trading. However, when the state of multi-leg trading system transitions from look-ahead to blocking, speculations can also be applied as a further optimization.

Some works[30~31] explore concurrency control strategies by rebuilding the transaction ordering, e.g. **commit ordering**[31]. However, those works are focused on coordination with pre-defined global ordering, which is not applicable to our problem, because, in multi-leg trading systems, the global ordering depends on the consensus of each multi-leg session and is not pre-defined.

VII. Conclusions

Multi-leg stock trading requires coordinating transactions across multiple stock symbols. When the stock symbols are on different nodes, coordinating the transactions across the nodes can incur significant overhead. Existing concurrency control approaches such as optimistic locking[29, 32] or speculative locking[17, 23, 24] are not sufficient to achieve good performance, because those algorithms are not aware of the fairness constraints enforced by stock trading rules. Although such fairness constraints limit concurrency significantly, they also provide flexibility for allowing out-of-order processing in certain situations. This paper identifies those situations and proposes a new look-ahead approach to alleviate the coordination overhead of multi-leg trading by a fine-grained state machine which allows transactions to be processed on a node before a previous transaction has finished. The state machine is designed so that the fairness constraints enforced by stock trading rules are always maintained.

We have implemented two stock trading systems. The first is a base-line prototype based on 2PC[1,2,19] and efficient session management. The second uses our look-ahead algorithm to reduce the overhead of coordinating multi-leg transactions on multiple nodes. Our results show that, the proposed look-ahead solution can achieve up to **58%** throughput gain and up to **30%** latency reduction.

We also quantitatively evaluated how a number of factors (e.g. percentage of multi-leg orders, number of legs, price/quantity variance, different priority ordering rules) affect performance and tradability. We found that increasing the number of legs decreases performance and tradability. Increasing the percentage of multi-leg orders will also decrease performance and tradability, but will not always decrease the throughput gain from the look-ahead optimization. Using priority ordering rules requiring more legs to be at the front of the priority queue before a multi-leg order can trade will decrease performance and tradability, but will increase throughput gain from the look-ahead optimization. Variance in price and quantity will decrease tradability, but increase performance, however, not always increase the throughput gain from the look-ahead optimization.

These observations can facilitate the design and deployment of multi-leg trading systems. Our work can also benefit a broad class of coordinated transaction processing systems with ordering constraints such as on-line auction [14~26, 20] and stream computing with priority constraints[26] .

There are a number of ways in which we are continuing this work. These include enhancing our system to handle multi-leg orders with more symbols, optimizing tradability based on our look-ahead algorithm, and generalizing our look-ahead algorithm to integrate it with existing concurrency control algorithms.

REFERENCES

[1] M. Tamer Ozsu. 2007. *Principles of Distributed Database Systems* (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.

[2] Michael J. Fischer. 1983. The Consensus Problem in Unreliable Distributed Systems (A Brief Survey). In *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*, Marek Karpinski (Ed.). Springer-Verlag, London, UK, 127-140.

[3] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. 2007. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing* (PODC '07). ACM, New York, NY, USA, 398-407.

[4] WebSphere MQ Low Latency Messaging, http://www.ibm.com/software/integration/wmq/llm

[5] Hypervisor, http://en.wikipedia.org/wiki/Hypervisor

[6] Gong. Su, and Arun. Iyengar, A Highly Available Transaction Processing System with Non-Disruptive Failure Handling, IBM Research Report. December 2009.

[7] IBM Redbook, IBM System z10 Enterprise Class Technical Guide, http://www.redbooks.ibm.com/abstracts/sg247516.html

[8] IBM Redbook, HiperSockets Implementation Guide, March 2007.

[9] Torsten. Layda, An Order Matcher for an Electronic Stock Exchange, OOPSLA DesignFest'97.

[10] Fidelity, Trading Multi-leg Options https://scs.fidelity.com/webxpress/help/topics/learn_trading_multileg_options.shtml

[11] Ming Fan, Jan Stallaert, and Andrew B. Whinston. 1999. The design and development of a financial cybermarket with a bundle trading mechanism. *Int. J. Electron. Commerce* 4, 1 (September 1999), 5-22.

[12] Ming Fan, Jan Stallaert, and Andrew B. Whinston. 1999. A Web-Based Financial Trading System. *Computer* 32, 4 (April 1999), 64-70.

[13] J Jawad Abrache, Teodor Gabriel Crainic, Michel Gendreau, Models for bundle trading in financial markets, European Journal of Operational Research, Volume 160, Issue 1, Applications of Mathematical Programming Models, 1 January 2005, Pages 88-105

[14] Sven de Vries and Rakesh V. Vohra. 2003. Combinatorial Auctions: A Survey. *INFORMS J. on Computing* 15, 3 (July 2003), 284-309..

[15] Aleksandar Pekec and Michael H. Rothkopf. 2003. Combinatorial Auction Design. *Manage. Sci.* 49, 11 (November 2003), 1485-1503.

[16] Kalagnanam, J., Parkes, D.: Auctions, Bidding and Exchange Design. In: Simchi-Levi, Wu, Shen: Supply Chain Analysis in the eBusiness Area, Kluwer Academic Publishers, 2003.

[17] Evan P.C. Jones, Daniel J. Abadi, and Samuel Madden. 2010. Low overhead concurrency control for partitioned main memory databases. In Proceedings of the 2010 international conference on Management of data (SIGMOD '10). ACM, New York, NY, USA, 603-614.

[18] Jim Gray and Andreas Reuter. 1992. Transaction Processing: Concepts and Techniques (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[19] Dale Skeen and Michael Stonebraker. 1987. A formal model of crash recovery in a distributed system. In Concurrency control and reliability in distributed systems, Van Nostrand Reinhold Co., New York, NY, USA 295-317.

[20] eBay, http://www.ebay.com

[21] ETF, http://en.wikipedia.org/wiki/Exchange-traded_fund

[22] Basket Swap, http://www.equityderivatives.com/what-the-experts-say/glossary/basket-swap/

[23] Ramesh Gupta, Jayant Haritsa, and Krithi Ramamritham. 1997. Revisiting commit processing in distributed database systems. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data* (SIGMOD '97), ACM, New York, NY, USA, 486-497.

[24] P. Krishna Reddy and Masaru Kitsuregawa. 2004. Speculative Locking Protocols to Improve Performance for Distributed Database Systems. IEEE Trans. on Knowl. and Data Eng. 16, 2 (February 2004), 154-169.

[25] Mudhakar Srivatsa, Li Xiong, and Ling Liu. 2005. ExchangeGuard: A Distributed Protocol for Electronic Fair-Exchange. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01* (IPDPS '05), Vol. 1. IEEE Computer Society, Washington, DC, USA.

[26] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, and Patrick Valduriez. 2010. StreamCloud: A Large Scale Data Streaming System. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems* (ICDCS '10). IEEE Computer Society, Washington, DC, USA, 126-137.

[27] Dale Skeen. 1981. Nonblocking commit protocols. In Proceedings of the 1981 ACM SIGMOD international conference on Management of data (SIGMOD '81). ACM, New York, NY, USA, 133-142.

[28] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. 1987. On the minimal synchronism needed for distributed consensus. J. ACM 34, 1 (January 1987), 77-97.

[29] Michael J. Carey and Miron Livny. 1989. Parallelism and concurrency control performance in distributed database machines. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data* (SIGMOD '89). ACM, New York, NY, USA, 122-133.

[30] Ahmed K. Elmagarmid and Weimin Du. 1990. A Paradigm for Concurrency Control in Heterogeneous Distributed Database Systems. In *Proceedings of the Sixth International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 37-46.

[31] Yoav Raz. 1992. The Principle of Commitment Ordering, or Guaranteeing Serializability in a Heterogeneous Environment of Multiple Autonomous Resource Mangers Using Atomic Commitment. In Proceedings of the 18th International Conference on Very Large Data Bases (VLDB '92), San Francisco, CA, USA, 292-312.

[32] Atul Adya, Robert Gruber, Barbara Liskov, and Umesh Maheshwari. 1995. Efficient optimistic concurrency control using loosely synchronized clocks. In Proceedings of the 1995 ACM SIGMOD international conference on Management of data (SIGMOD '95), ACM, New York, NY, USA, 23-34.