Tradenet: A Collaborative Stock Trading System

Ali Hisham Malik
College of Computing
Georgia Institute
of Technology

hisham@cc.gatech.edu

Alexander G. Gray
College of Computing
Georgia Institute
of Technology

agray@cc.gatech.edu

Abstract

Development of semi- and fully- automated trading systems is becoming increasingly popular in the financial market. Our aim is to develop a *scalable lightweight distributed* trading system that is self-configuring, self-healing, and provides fast execution of trades with high availability. We used a peer-to-peer (P2P) platform which would allow for the flexibility of non-static nodes dispersed across the globe to securely communicate with each other and configure accordingly. A P2P network also allows machines in the network to discover available services. In such a domain, the machines in the network can be fine-tuned to heal from faults in the network, the most damaging being the network partitioning issue. These features were found very useful for development of our trading system.

1 Introduction

The Internet has had a significant impact on the financial markets. Several stock brokers provide API connectivity for executing trades using a trading software. This has led to a strong interest in developing an automated trading system using time-tested trading strategies. As more and more stock exchanges across the world are becoming accessible through the click of a button, the standalone trading system does not scale well in managing a diverse trading portfolio. As a result, the move towards distributed trading systems is ever more important.

Our goal is to develop a system that, while being easy to manage and maintain, can scale up to several machines distributed across the globe for trade execution and simulation while being easy to maintain. It was also anticipated that this distributed may be semi or fully automated and the trade may be executed within the network through a machine algorithm or through manual intervention. Furthermore, as periods of non-functionality or inconsistency may lead to heavy losses, the trading system must be able to quickly recover from faults within the network. Finally it was realized that the timing for the execution of trade is crucial and that the lag between the time that the trade execution is requested in the network and the time when the trade is executed must be minimized.

Since the desirable property of minimizing the mean time to recover (MTTR) requires that all machines in the network are in a consistent state when the trade is executed, the requirement of minimizing the trade execution time, at a first glance, may seem to be conflicting desire. We show that by using a peer-to-peer network model, we can achieve these seemingly conflicting goals as well as gain on some features of the peer-to-peer network which make the distributed system more fault-resistant.

In addition to the protocol for execution of trade within the network, protocol for sharing stock data among nodes in the network was also developed. This protocol helps in maintaining the notion of one-stop serves all point, to machines within the network for all their stock data needs (for example, for generating trading strategies and back-testing). Furthermore, it allows us to limit the number of machine that have to contain sensitive account information and be exposed to the outside world.

1.1 Peer-to-Peer Platform

Use of P2P networks remains unexplored in the development of distributed trading system. The P2P networks have been found to be successful in sharing large amounts of data [RM 01], to deploy VoIP communication [SKYPE 06], and to use computer's idle time for performing some complex calculations [ACKLW 02].

This paper discusses our implementation of distributed trading system and framework built over a open-source P2P platform, JXTA [JXTA 06]. There were many reasons for selecting JXTA as the platform of choice for developing this trading system of which the most evident being:

1. Plug n' Play environment: A newly added machine can automatically discover other machines and start communication with them.
2. Interoperable: Not dependent on a single operating system.
3. Security: Support for industry standard security mechanisms such as transport layer security, security certificates etc.

Another useful feature of the peer-to-peer platform is its natural resistance to faults in the networks. In a statically configured system, when network faults develop between a pair of machines, they would not be able to communicate with each other. On the other hand, in a peer-to-peer model, it is possible for a pair of machine to be able to communicate with each other by relaying through a third machine even when they cannot communicate directly with each other.

2 Tradenet Framework

2.1 Peer Roles in the Trade Execution Network

In our model, each trading peer group within the network corresponds to a brokerage account. All the nodes in a specific trading group assume that they are using a unique brokerage account that is specifically used for this trading peer group only. Assuming such uniqueness, we can make some safe assumptions about the status of the network. As discussed later, this allows us to provide higher tolerance and consistency in the network.

There are two main types of managerial roles that a peer can be while executing in the trading peer group, 1) Queen, and 2) Backup and one type of client role, 3) TradeAdvisor.

2.1.1 Queen

The Queen is responsible for listening for trade advises and placing the trade with the broker. Each trade advise received passes through the following two stages of approval,

1. *Preapproved TradeAdvise*

   This is the part of the approval process where the business logic of trade request approval is applied and a unique order number is associated with it. Once the transaction has been preapproved it will only get disapproved in next stage under exceptional circumstances.

2. *Approved/Disapproved TradeAdvise*

   Part of the trade request approval where the trade request is sent to the broker with the unique order number associated with the trade advise during the preapproval stage. The broker may also accept or reject the trade request.

The Queen also listens for synchronization requests from the Backup nodes and notifies them of the ordered list of successors to the Queen when the node servicing as the Queen exits. It is also responsible for providing trade notifications to the Backup nodes as the trade advise passes through the above mentioned approval stages. Also notice here that, if the Backup does not receive a trade notification from the Queen, it can directly retrieve the list of pending trades through the brokerage account directly.

There can be only one Queen active in each trading peer group and multiple Queens in the group is considered to be a fault that needs to be fixed 'immediately'. When a Queen conflict is detected in the network and this node is not the primary Queen, then only in such circumstances can the secondary Queen node(s) switch to the role of the Backup if this node is not the primary Queen.

Thus ideally, at any given point in time, only the Queen node is placing the trades using the brokerage account. Thus when the Queen exits abruptly, the node that is the next in line Queen verifies that the broker has the record of all the preapproved trade advises. This can be done since the order number associated with the preapproved trade advise is also unique at the brokerage account level. Thus the number of preapproved trade advises that might get dropped before reaching the broker, is reduced.

Note the use of the term 'reduce' rather than 'eliminate' above. In order to guarantee that all the preapproved trade advises are always placed with the broker in all cases, a modified protocol can be used. In this modified protocol, the Queen would have to send a notification to each of the Backup nodes when it receives a trade advise with the unique order number associated with it at this point rather than after the preapproval stage. Also, the Queen would not be able to proceed to preapproving the advise until all the Backups have received the notification. Using this modification we can guarantee that any preapproved request is sent to the broker since, the next in line Backup can always preapprove the request received and verify with the broker if the corresponding trade request has been placed or not.

Although the modified protocol, eliminates the case of preapproved trade advises being lost, this protocol is undesirable due to two main reasons. First of all, this protocol requires that each trade advise be sent to the Backup regardless of whether it is a viable trade advise. This can result in excessive message passing in the system. Second, the trade advise cannot be preapproved and sent to the broker until all the Backups have received the notification for getting the trade advise. This results in the latency between the time a trade advise is received to the time it can be executed. Thus, there would be a higher threshold for the minimal expiration time trade advise (one with little time before expiration) that can be executed.

2.1.2 Backup

Each node that is part of the manager-side role starts as a Backup node. Only when no Queen is discovered in the group, does a node start servicing as Queen. The Backup node listens to synchronization notifications from Queen as discussed above. Upon initial entrance to the trading peer group, while searching for the Queen, if the Backup node finds multiple Queens, it generates the Queen Conflict Alarm.

2.1.3 Trade Advisor

Apart from the two managerial roles, a peer can also be in the client role, termed as TradeAdvisor. The TradeAdvisor is responsible for generating a trade request/advise, communicating it to the Queen and optionally, listening for the response. Note here that the Queen only sends the 'unsuccessful' response message to the trade advisor when it is shutting down or has lost connectivity with the broker.

The trade advise generation mechanism of the trade advisor is abstract and depends on the trading strategy employed which generates the trade advise. For example, in implementation, the trade advise could simply be generated manually using a GUI interface, or through a machine learning algorithm analyzing the movement of the markets.

The trade advisor can also optionally process the response received from the Queen which could indicate that the trade advise was accepted or denied. By default when the trade advisor receives any response that indicates unsuccessful processing it connects to a different trading peer group and communicates the trade advise to the Queen within that peer group.

Note that a node in the network can simultaneously be operating in one of the managerial roles, client role or both. Furthermore, since the base TradeAdvisor has a very minimalist set of requirements, it may very well be configured to run over cell phones.

2.2 Manager Composition

Each node capable of operating in one of the managerial roles is composed of the objects and can provide the services as follows,

2.2.1 AccountManager

The AccountManager object provides connectivity with the stock broker for placing trades. It is responsible for providing a consistent snapshot of the stock account. In the current implementation, this is done by preemptively adjusting the account when a trade is placed (but not executed) and doing the final adjustments when the trade is executed. Each implementation of AccountManager provides a common minimalist set of trade execution and control APIs. This conformance allows us to plug-in any stock brokerage account without modifications to other parts of the system.

2.2.2 SynchronizationProvider

The SynchronizationProvider service runs on the node servicing as a Queen. This service is responsible for processing requests for synchronization from backup. It is also responsible for providing the Backup peers with an updated ordered list of successors to the Queen each time that list changes (due to the addition of deletion of nodes). When the Queen voluntarily exits from the network, it uses this service to notify the Backup nodes. Similarly, the Backup peers perform a clean exit by notifying the Queen through this service. This service is also used for providing the trade notifications to the Backup peers.

2.2.3 Synchronization Listener

The SynchronizationListener service runs when the node is servicing as a Backup. This service is used to listen to the synchronization updates provided by the Queen using the SynchronizationProvider service. Another important function of this service is to periodically verify connectivity with the Queen and activate the Queen exit handler when this service is unable to connect with the Queen.

2.2.4 TradeAdviseListener

The TradeAdviseListener service runs on the node servicing as a Queen. This service is used to listen for trade advise from the TradeAdvisors within the group. This service reroutes the trade advise to the trade advise handle which processes the trade advise and returns a response back to the service. This service is then responsible for communicating the response back to the TradeAdvisor.

2.3 Stock Data Sharing

The nodes in the system can also share data related to stocks such as historical prices, financial news, options data, etc. In our design, the client for the stock data could be working in standalone mode and fetching the data directly from the source, or be using the P2P network to fetch the data it requires. To support extensibility, a layer of abstraction is introduced so that the once the data fetcher is configured, the client need not be aware of whether it is getting the data through the source directly or through the P2P network.

While fetching the data directly from the source is certainly faster, the support for data sharing through the P2P network allows for network configurations where only a few machines need to be able to access the source data provider, whereas others do not need to be authenticated or store sensitive account information. Finally accessing the data through the network allows for combining different source data providers at the DataProvider node and the DataRequestor only needs to know the services provided by the DataProvider node rather than the individual data sources.

Two types of peers exist in the system for data sharing; 1) DataProvider, and 2) DataRequestor.

2.3.1 DataProvider

The DataProvider is analogous to the Queen. Note here that, while there can be only one Queen in the trading peer group, there can be multiple DataProviders in the group. Furthermore there can be different flavors of DataProviders that exist in the group in the sense that one DataProvider might be specialized in providing financial news whereas the another might be specialized in providing historical data.

2.3.2 DataRequestor

Similarly, the DataRequestor is similar to the TradeAdvisor, except that it requests stock data from the DataProvider instead of extending a trade advise. Also, while the TradeAdvisor registers a callback to get the response asynchronously, the DataRequestor waits until the response is received. Currently, the configuration of different flavors of DataProviders available need to be static and known to the DataRequestor before initialization so that it knows which DataProvider to contact for the required information.

2.4 Delivery Channels/Communicators

There are three types of delivery channels/communicators being employed

2.4.1 Synchronous Communicator

This communicator provides user the abstraction of a simple function call. The message is sent to the desired node and the communicator waits until a response is received. In case of non-responsive node,

attempt to fix the communication is made and a null may be passed back to the caller if this communicator gives up on fixing the communication issue. Currently, this communicator is being used to request financial data such as historical stock prices, financial news, and options within the network.

2.4.2 Asynchronous Lazy Communicator

This communicator provides non-blocking communication of message to the caller. The caller can call this communicator with an optional response listener that is used to pass the response back in case 'any' response is received. This communicator is unreliable as only a certain number of attempts are made each time to fix any communication issue. After certain number of unsuccessful attempts to deliver the message, the message is dropped. This communicator is currently being used in Tradenet for passing and collecting trade recommendations [MAH 06] in the network.

2.4.3 Asynchronous Keep-Alive Communicator

This communicator maintains a pulse with the node being contacted. At each pulse, the connectivity with the node is verified and any attempt to fix any communication issue is made. It also contains a client provided handler which is notified when the communication issue with the node cannot be resolved. This handler is responsible for reinitializing and perhaps using a different node to communicate with. Apart from calling the handler, this communicator returns the original message back to the client when communication issue cannot be resolved. In case of communication timeout where the message was sent, but response message not received within certain time period, this communicator returns an empty message to the registered listener. Currently this communicator is being used as part of the TradeAdviseCommunicator.

2.4.4 TradeAdviseCommunicator

This is a specialized communicator for handling trade advise communication. It uses Asynchronous Keep-Alive Communicator internally to send the request. On communication failures with Queen reported by the internal communicator, this communicator attempts to connect to a Queen in a different trading peer group. This communicator also parses and handles the response from the Queen informing it of any error messages. In case the error message can be handled by this communicator itself, the client is not alerted of the error.

3 Network Partitioning

It is possible for peers within the group become partitioned in a way that they are unable to communicate with each other. This would result in creation of multiple Queens. Given our protocol for detecting and resolving multiple Queens helps in healing from network partitioning issue. However, network partitioning scenarios where multiple Queen issue is undetectable. To illustrate this, assume that we have four nodes; node Q and B that act as Queen and Backup respectively, and node A1 and A2 communicate with the Queen as TradeAdvisors. Now we run into the issue of network partitioning in such a way that node B and A2 cannot communicate with node Q anymore and vice versa. Furthermore, assume that node A2 is not able to discover the advertisement for Queen service provided by machine Q but is able to communicate with node B and discover the services provided by it. In such a situation, node B will eventually mark machine Q as dead and start servicing as Queen. Node A2 will discover the Queen service provided by node B and start sending the trade requests to it. Figure 1 illustrates this scenario.

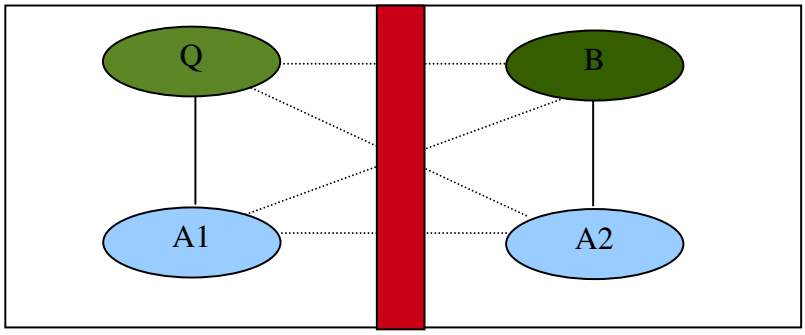

Figure 1: The two separated blocks represent the two partitions of the network. Nodes in each partition can communicate with each other (represented by non-breaking lines). The dotted lines represent the communication that cannot happen due to the network partitioning face. Queens are represented by nodes colored green and trade advisors are represented by nodes colored blue.

Network partitioning tends to be a major problem in highly distributed systems. Many papers handle this problem by applying domain-specific knowledge. Network striping is used for handling network partitioning scenario in [SM 98]. In network striping strategy, the different nodes are statically configured in such a way that a node (or a group of node that are physically close to each other) is specialized to do a task. When network partitioning issue arises, these nodes can operate independently and we guarantee that inconsistencies will not occur. When using network striping, it is assumed that the distributed machines affected by network partitioning will eventually come back to life. Until then, any transaction that requires collaboration between the machines is withheld. Network striping would work as a solution for network partitioning in Tradenet at the level of Trading Peer groups since each peer groups represents a set of peers associated with a different broker and hence there is no conflict of interest between different peer groups if they get partitioned. In our current implementation, there is no collaboration taking place between the different trade groups except for switching of trade groups by the peers as the last resort for resolving communication issues without creating conflicts. Thus network striping is not applicable for inter-group communication. Also, network striping is not useful at the granularity of a single trading peer group as there can be only one Queen within the group and further striping is not possible.

An approach taken to tackle network partitioning issue is to use delayed commit [SM 98]. In such an approach, when the nodes get partitioned, none of the transactions are fully committed. Only when all the nodes are able to communicate with each other are the delayed transactions tallied and committed. Conflicting transactions are resolved using a undo policy which allows for the transactions which have been delayed committed to be denied. This approach does not deal with scenario of prolonged network partitioning. After sometime the partitioned systems may start working independently or the system may stop functioning as a whole, depending upon the policy used.

If delayed commit approach was used within Tradenet, a trade could not then be executed by the Queen until the notification has been sent to all the Backups. As a result, when faced with a network partitioning problem, there is a higher expiration time threshold for trade advises that can be executed. Further, using the delayed commit approach in this domain meant that it is possible for the trade advise to expire during the time that we are receiving the message acknowledgments (Acks) back from the Backups. When this happens, the trade advise would have to be rejected with a notification to the Backups for this trade advise being reject along with the standard recursive communication policy for when the notification could not be sent to the Backup. Finally even with using delayed commit, the scenario illustrated in Figure 1 can still occur in the case of a prolonged network partitioning scenario.

Based on these observations, it was decided against using delayed commit. The approach used here is to optimize for the common case and maximize the number of trades that can be executed based on the short-lived trade advises. Thus when a trade advise is approved, it is immediately sent to the broker and a notification to the Backups is sent in parallel. If the notification cannot be sent after certain attempts, it is marked as dead. The Backup node on the other hand, makes sure that it is able to communicate with the Queen. If the Backup does not receive a notification from Queen for a certain period of time, it sends a 'subscribe' message to the Queen. A duplicate 'subscribe' message at a Queen's end is harmless, but on the Backup's end, a subscribe message means that apart from receiving other state information, the node gets

the updated list of pending transactions at Queen node [1]. Furthermore, while the Queen is quick to mark a Backup as dead, the Backup makes several attempts to communicate with the Queen before marking it as dead. This communication mechanism takes care of inconsistencies that may arise during short lapses of network partitioning.

Lastly, in the case of prolonged network partitioning or the scenario illustrated in Figure 1, the existence of multiple Queens in the peer group is detected through the communication with the broker. Each Queen has subscribed with the broker to inform it of any account updates and trade executions. When a Queen receives a notification for a trade execution that it cannot tally, it realizes that it might be facing a network partitioning issue. At this point, that Queen exits out of this peer group marking this peer group as bad forever (not using this peer group and broker account ever again, or until some manual intervention or restart) and joins another peer group. If no peer group is available, then the Queen node simply shuts down requiring manual intervention.

Upon getting the notification for Queen exit due unknown order, all the Backups synchronizing with the Queen reinitialize themselves, rather than following the standard Queen exit protocol of communicating with the Backup node in line to be the Queen. Any trade advise received during the time Queen is exiting out of the system is sent back with an 'unsuccessful' response which indicates to the TradeAdvisor that it needs to reinitialize itself and send the trade advise to the new Queen. This allows the Backup to search for the duplicate Queen. If the Backup nodes are able to discover that Queen, the inconsistency caused due to network partitioning is considered resolved. When the Queen is not found, the Backups resort back to following the ordered list of successors in choosing the Queen.

Note here that the above protocol used by the Backup nodes is very optimistic, since it is assumed that if the Queen is not discovered, it was probably because that node had also detected the network partitioning issue and exited out of this group, or that node is simply not in operation anymore. Thus, if the network partitioning problem still existed, we would not have gotten rid of the multiple Queens in the group. A pessimistic protocol that would avoid multiple Queens would be that the Backup, upon getting the notification from the Queen, also exit and connect to a different trading peer group (and broker). This pessimistic approach, while minimizing the inconsistencies, can quickly result in the trading peer group being completely abandoned by all the partitioned nodes leading to no node using that peer group and broker. Finally in case of limited brokerage accounts, this would quickly result in a complete shutdown of the entire system.

To explain this further, lets continue with the network partitioning example provided above. Assume that the nodes are communicating in partitioned mode in trading peer group (and broker) X and we can connect to another peer group Y. In addition to a Queen and TradeAdvisor, suppose we have Backup J and K for Queen Q and Backup L for Queen B. Assume that both the Queens Q and B have placed request for trades with the broker X. Figure 2a illustrates this configuration.

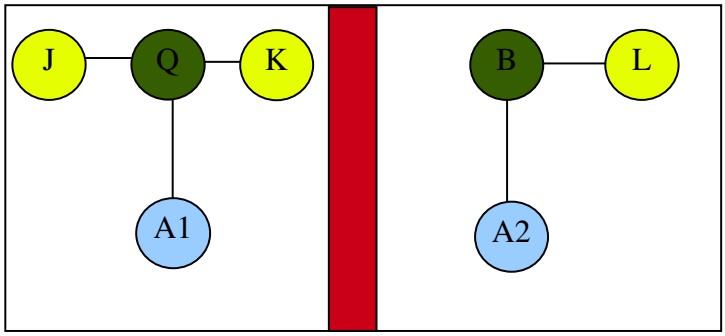

Figure 2a: The two separated blocks represent the two partitions of the network. Lines between the nodes represents active communication between the nodes. Queens are represented by nodes colored green, Backups by yellow, and TradeAdvisor by blue.

[1]Transactions pending at the broker side are the ones where the order has been placed, but the trade has not been executed yet. On the other hand, transactions pending on the Queen node are the ones which have been preapproved, but not yet sent to the broker.

Let us look at the behavior of the Queens when the trades get executed at broker's end and the broker notifies the Queens. Upon getting the notification, node Q sees an executed trade that it did not request, exits, and connects into peer group Y. Similarly, node B sees an executed trade that it had not placed, exits, and connects into peer group Y. Figure 2b illustrates configuration of trading peer group X after the two Queens exit. Since the nodes are facing the network partitioning issues, each node becomes the Queen and starts placing trade requests with the broker Y. Again, these two nodes observe the trade request that they had not placed and follow the same procedure. However, this time, we do not have any other brokers to connect to and hence the nodes halt completely requiring manual intervention.

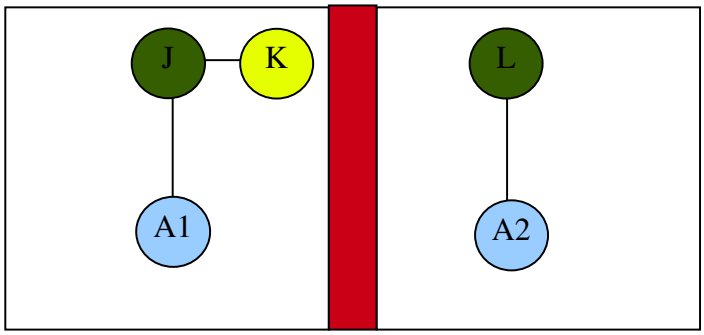

Figure 2b: Configuration of trading peer group X after nodes Q and B detect network partitioning and exit X. The remaining nodes, follow the optimistic approach. The two separated blocks represent the two partitions of the network. Lines between the nodes represents active communication between the nodes. Queens are represented by nodes colored green, Backups by yellow, and TradeAdvisor by blue.

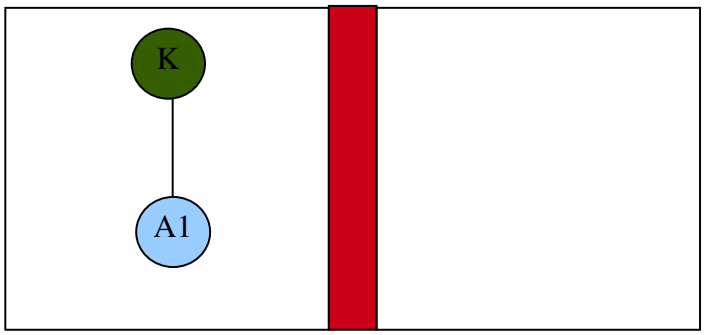

Figure 2c: Configuration of trading peer group X after nodes J and L detect network partitioning and exit X. The remaining nodes, follow the optimistic approach. The two separated blocks represent the two partitions of the network. Lines between the nodes represents active communication between the nodes. Queens is represented by nodes colored green, and TradeAdvisor by blue. Since A2 does not find any Queen, it also exits from peer group X.

Now, let us look at what happens at the end of Backup J, K and L. If we were to follow the pessimistic approach, then the Backups J, K and L would follow their Queens and as a result the entire system would eventually halt. When we follow the optimistic approach, J and L would become the Queen, detect network partitioning, move to trade peer group Y and eventually halt. Figure 2c illustrates the configuration after J and L exit the system. Note what happens to node K. When the J and K become Queens, node K becomes the backup for node J (in common case). When J and L exit from the trade peer group, K reinitializes itself, becomes the Queen with no conflicts and this partition of the network can still keep on executing trades. Thus, using the optimistic approach, we are at a higher risk of authorizing a trade request without having the full knowledge of our broker account.

The benefit of the optimistic approach is that we can have a higher chance of healing the system before halting it completely. Both the approaches are viable and depend on how important it is to have the

'complete' control of the broker account to authorizing a trade. We favored in the direction of optimistic approach, but Tradenet can be easily configured to follow the pessimistic approach as well.

4 Optimizations

4.1 Transaction Serialization and Message Queues

To maintain high availability and throughput at the Queen's end, the notifications by the Queen to the Backup nodes are sent in parallel to the main thread execution. The order in which the Backups receive these notifications has to be maintained, to ensure transactional consistency. Finally, it is desirable that a slower responding Backup node does not affect the throughput of the notifications that are sent to the Backups.

Keeping in view these goals and properties, the initial design was to broadcast the notification to each Backup node, creating a separate thread for each node being contacted. To make the broadcast non-blocking, each broadcast would run in a separate thread, but the actual broadcast would not start until the previous broadcast had finished sending its messages and performed related required actions.

It was observed that although the solution maintained the desired properties of non-blocking communication, it did not scale well as the Queen would use excessive system resources. As a consequence, it was not able to process the trade advise before the Trade Advisor timed out, or denying the trade advise with short expiration duration as it could not process it in time. An analysis of the system showed that the scalability issue was due to the creation of threads for each broadcast and Backup node. In general case, the action performed in these threads (ensuring delivery of message) was fast and creating a separate thread for it did not justify the cost. Also, since only one broadcast could be active at a time, these threads would continue increasing (waiting on the lock) in case of a slow responding peer. This increase in the number of threads would cause latency in the processing of the trade advise as a side-effect.

The solution for this scalability bottleneck was to, first, relax the notion of consistency among Backup nodes. It was decided that for a given window of notifications, it is okay if one Backup node is behind the other Backup nodes in receiving these notifications. Thus at any given time, multiple broadcast can be active while being sent to each Backup in order. Second, using messaging queues for each Backup node and associating a single thread with each of these queues, we can avoid the excessive creation of short-lived threads. This approach significantly improved the scalability of the system.

4.2 Potential Backups

While having Backup nodes for the Queen are desirable, the addition of each Backup node means that there is one more node that the Queen needs to notify each time there is a change in the network or account state. While having several Backup does not block the trade executions, it does consume valuable machine resources and also indirectly affect the latency of the entire system.

This issue was resolved by addition of another role in the network named, 'PotentialBackup'. Each time a Manager node discovers the Queen and request synchronization with the Queen, the node automatically assumes that it is granted the role of 'PotentialBackup' meaning that it will not be receiving notifications from the Queen until the time when there is space available for Backup node in the network. The Queen on the other hand, upon receiving the synchronization requests, checks its list to see if it has the desired number of Backups. If the Queen already has the desired number of Backups, then the node requesting synchronization is registered as a Potential Backup meaning that only when some Backup node exits out of the system, would the Potential Backup be upgraded to a Backup and notified of this upgrade. If however, upon receiving the synchronization request, the Queen is still short of the desired number of Backups, it immediately upgrades the node to a Backup and notifies it of the upgrade. Thus, this role of 'PotentialBackup' allows the flexibility of having the desired level of fault tolerance in the system without the penalty of unnecessary message passing.

4.3 Synchronization Coordinators

It was observed that during period of network partitioning, when a Backup node B is not able to communicate with the Queen, marks the Queen as dead and tries to synchronize with the Queen's successor node S in the network, it may not receive a response back from node S. This happens since the successor

node S might still be able to communicate with the Queen and as a result is not providing the Queen service of listening for synchronization messages. As a result, the node B might eventually mark the successor as dead and try to synchronize with the next in line Queen's successor. This process continues all the way until the next in line Queen's successor is node B itself. At this point the node B starts servicing as Queen node and we have an issue of multiple Queens in the network.

Notice here that even during the period of time, node B is requesting synchronization with other nodes in the network. If the network partitioning issue was resolved during this time, node B would not start synchronization with the Queen. This situation can arise quite frequently during short periods of network failures. Hence, to minimize the creation of multiple Queens in the network, we introduced the notion of SynchronizationCoordinator.

A SynchronizationCoordinator service is an extension of SynchronizationProvider service. Whereas SynchronizationProvider service can only be provided by the Queen, the SynchronizationCoordinator service can also be provided by the Backup node as well. The important difference between providing SynchronizationCoordinator service in Backup mode and Queen mode is that in Backup mode, it is operating as a dummy service. This means that when the node receives a request for synchronization in Backup mode, it simply notifies the whereabouts of the current Queen in the system to the requesting node. In the Queen mode, the SynchronizationCoordinator operates identical to the standard SynchronizationProvider.

Therefore, introducing SynchronizationCoordinators we are able to avoid one more network fault in the network without overloading the system. Also, note that since SynchronizationCoordinator is already initialized when the Backup node switches to start servicing as a Queen. Thus when a Queen node exits, a successor is able to start servicing as Queen quicker leading to a reduction in MTTR.

4.4 Message Marshalling/Unmarshalling

During any communication, such as trade advise, data request, synchronization request, etc, message passes through different layers of handlers before being sent. Marshalling and unmarshalling the data at every layer becomes have a significant effect on the performance as discussed in [HP 91]. To reduce this cost, all messages are marshalled and unmarshalled to JXTA native message format at exactly one layer on each side of communication endpoint.

5 Implementation Details

As mentioned earlier, a node in the network can simultaneously be operating in one of the managerial roles, client role or both. In our current implementation, we have implemented the three roles as separate nodes in the trading peer group as we believe that to be the most common desirable configuration. The system takes advantage of the new Java 5.0 concurrency package. More specifically the ReentrantReadWriteLock is utilized where the standard ‘synchronized’ keyword is not sufficient. The system makes use of centralized resource files to make the system easier to configure and fine tune. Further Log4j is used extensively throughout to provide higher control over logging and debugging.

5.1 Data Services

Currently the data services provided in this distributed trading system are based on the data feeds provided through Yahoo Finance [YF 06]. These include,

1. Historical Prices

   This feed contains the high, low, open and close prices as well as trading volume for the given period of time for the provided stock symbol. The current granularity supported is Daily, Monthly and Yearly.

2. Options Data

   This feeds contains the call and put options that are expiring in the given month for the provided stock symbol. The fields within the call and put options are strike price, option symbol, last price, change, bid price, ask price, volume and open interest.

3. Stock Headline News

This feed contains the recent news on the stock or where the stock symbol(s) was referenced. This feed follows the RSS feed format which is a industry standard syndication format. Currently all the RSS specifications up till 2.0 are supported.

4. Stock Industry News

   This feed contains the recent news on all the stocks that fall into the industry that the given stock symbol(s) belong to. Like the headline news, this feed also follows the RSS specification.

Since Yahoo Finance does not provide a feed-friendly API for fetching options data, an open-source HTML parser, Jericho-HTML-parser [JHP 06] was used to parse the page and extract the required options data. In order to accomplish compliance with the various RSS formats, an open-source set of utilities for RSS/Atom syndication formats, Rome [CET 06], was employed.

It is important to mention here that while the Yahoo Finance was used as a point of reference, the trading system is not bound to using Yahoo Finance to provide these services.

5.2 Charting Analysis

In order to support trade execution/simulation using strategies based on charting analysis, the trading system provides the following sets of indicators,

1. Moving Averages
2. MACD and MACD-Histogram
3. Directional System
4. Momentum, Rate of Change, and Smoothed Rate of Change
5. Williams %R
6. Stochastic
7. Relative Strength Index

Furthermore, the standard trading strategies as described in [EA 93] using these indicators were also implemented. The trading system also supports on the fly generation of charts based on trading signals produced by the trading strategies. This helps in performance analysis of the different trading strategies as well as fine tuning of the parameters.

The indicators and trading signals were coded in C due to performance reason. However, to speed up development by making use of open-source software, Java was employed. An open-source project, JfreeChart [JFC 06] was employed for generation of charts and the interaction between Java and C was done using the JNI interface [LS 02]. Finally trading strategies given in appendix A and [MKDK 05] were also experimented with.

5.3 Connectivity with Interactive Brokers

Connectivity with Interactive Brokers [IB 06] is implemented using the API provided by the broker. This API provides asynchronous communication, and real-time view of the account which includes partial order fills and other adjustments to the account. Important thing to note here is that the account value is adjusted only when the trade has been executed, rather than when the trade is placed.

For our system, it was desired to have a system that prevents going over account limit and avoiding duplicate order placement which could happen when only the current stock portfolio is checked instead of checking the open orders as well. To achieve this property, the APV provided by Interactive Brokers was encapsulated in an object to provide an instant, point in time, safe estimate of account value and portfolio holding keeping in view the open orders.

5.4 JXTA Advertisement Publisher

It was observed that when the advertisement for a service would stay in the network even when the service is no longer available. This meant that when a lookup for service is performed, many stale advertisements are found and time is wasted in connecting to the service that is no longer available.

In JXTA, there is mechanism for providing an expiration date for the advertisement. However this means that if the service is to be made available after that period, then the advertisement needs to be published again. This republishing mechanism is not very convenient for the service provider.

A generic mechanism was developed where whenever a service is started it is published to the Advertisement Publisher with minimum expiration time. The advertisement publisher runs in a separate thread and is responsible for republishing the advertisement when the advertisement expires or before that time. This mechanism led to massive decrease in startup time.

5.5 Security Model

The system utilizes the JXTA Transport Layer Security as well as certificates using the JXTA provided PSEMembership service. Using this security model, a node that has been authorized to participate in the group, is sent an encrypted invitation using symmetric key to join the group. We assume this passing of invitation is done offline. Using the symmetric key, the client decrypts the invitation and uses it to join the group, generates a certificate and authenticates itself in the group.

6 Measurements

A wealth of experiments can be performed on the system to evaluate its performance, due to limitation of time and available resources, some basic measurements were taken. The results from these experiments are encouraging and show scalability and availability of the system. All the measurements taken using a shared Intel P4 1.3Ghz machine with 768MB of RAM. The measurements were taken for different peer roles during startup, normal and abrupt exit of Queen. These measurements are summarized in table 1 and biased towards worst case scenario. These are summarized in Table 1 and graphed in figure 3, 4, and 5.

	Startup	Trade Advise	Queen Exit Normal	Queen Exit Abrupt
Queen	37.163s	0.06s+0.191 s	0.19s+10.045s	N/A
Backup	17.966s -38.555s	0.932s	0.13s	10.185s+0.39s
TradeAdvisor	6.319s	0.171s	5.859s+5.748s	5.859s+5.748s

Table 1: Time taken by peers for different activities.

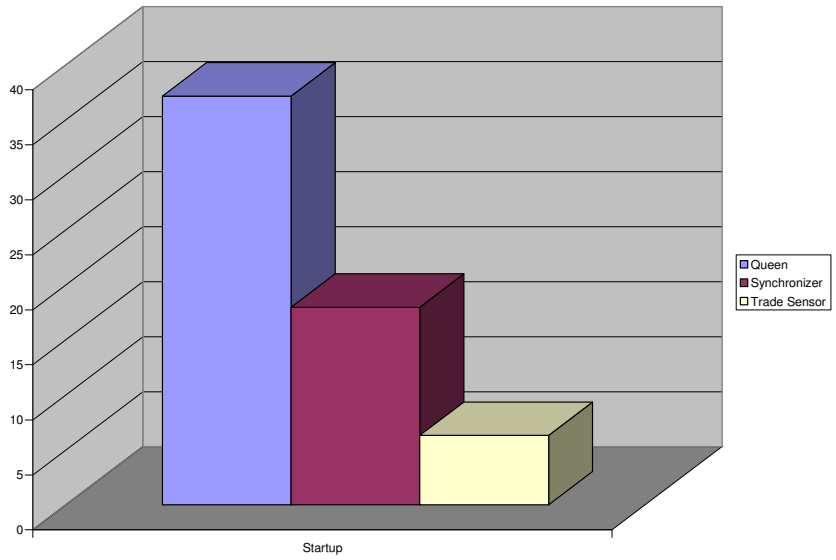

Fig 3: Startup time (in milliseconds) for different peers

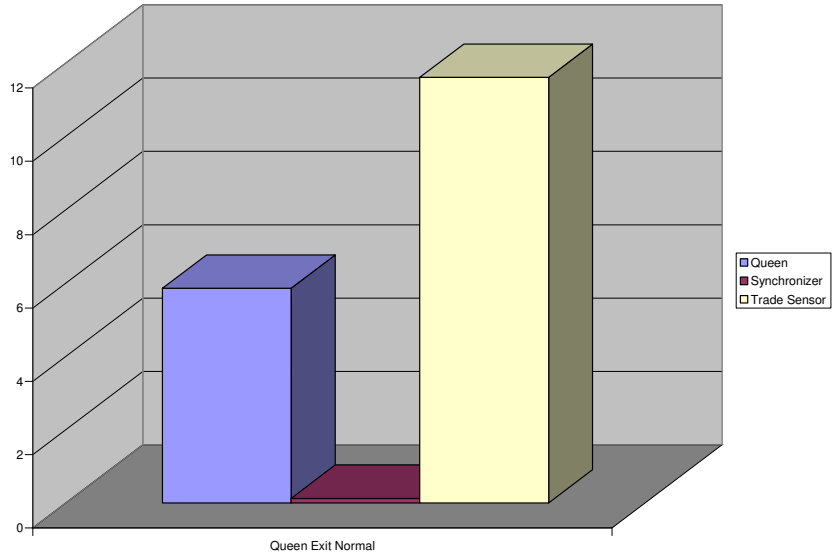

Fig 4: Time taken (in milliseconds) by the peers to recover when the Queen exits normally.

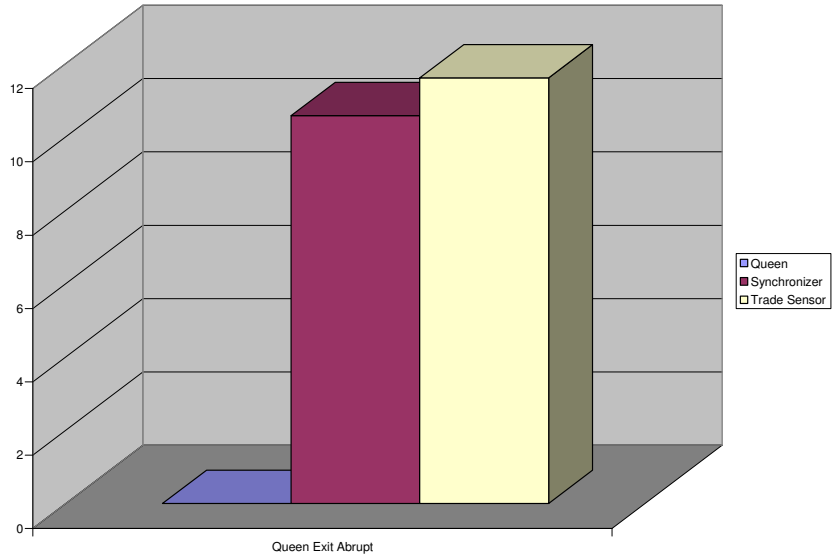

Fig 5: Time taken (in milliseconds) for the peers to detect and recover when the Queen exits abruptly.

From the graphs, it is due to the protocol devised, the peers are showing the desired behavior of low latency. There is a heavy startup cost for Queen. However that is tunable and the reason for this hight startup cost is that we need to ensure that there is only one Queen in the network.

7 Related Work

Development of trading and simulation systems, unfortunately, has not been significantly explored academic literature. Automated and semi-automated trading systems, however, have gathered increased attention in the private and corporate sectors. Good examples of such trading system include TradeStation by TradeStation Securites, ModelStation by Clarifi, Inc and TradeBolt by TradeBolt LLC. Although ModelStation is only a trade simulation system, it provides a grid computing and scheduling framework.

Unfortunately, the details of these systems are not available publicly. Saito, Bershad, and Levy developed a distributed mailing system, Porcupine [SBL 99], which is similar in many ways to our system. Like Tradenet, the goals of Porcupine are to develop a system that would self-configure, self-heal, provides fault containment and good performance. This system also assumes network of commodity machines where new machines can be easily added or removed from the system without making it unavailable. The differences between Porcupine and Tradenet are mainly due to the domains for which the two systems have been developed. Although not mentioned directly by Saito, Bershad, and Levy in their paper, in Porcupine, it is possible for a message sent once by a user to be received twice at the recipient's end. This inconsistency due to duplication however, eventually disappears in Porcupine. Although tolerable in the case of a mailing system, this case of duplication is not tolerable for trading system since it leads to an irreversible duplicate transaction. As a result, we move towards introducing a Queen node to maintain high throughput and avoid such duplicate transactions. Secondly, while Porcupine uses protocols developed in-house for peer/resource discovery and communication, Tradenet makes use of the open-source industry standard, JXTA for its requirements.

At the time of development of this framework, no paper was found that explored P2P networks for creating a distributed collaborative trading system. The closest system developed were Auction Trading[5].

Our approach of using 'potential backups' is similar in essence to the overflow nodes used by Fox, Gribble, Chawathe, Brewer, and Gauthier in their paper [FGCBG 97]. These overflow nodes are used during periods of high load only or when the primary server nodes are not available due to some fault. In Tradenet however, we assume that the machines used as Backup and PotentialBackup are homogeneous and thus the roles are assigned simply for scalability reasons while maintaining the desired level of fault tolerance.

8 Conclusion and Future Work

We have shown we can develop a large scale trading system that is highly available and fault tolerant and can run over commodity machines distributed across the globe with the TradeAdvisor module being able to run even on cell phones. By providing 'best effort' ACID transactions, the developed system introduces minimal delay between the time the trade request is generated and when it is executed. We argue that for a large scale trading system, such consistency guarantees should be sufficient, while the throughput and availability is critical. The developed system automatically reconfigures and recovers from problems in the network ensuring that the trade execution does not halt due to failures in the machines or network. We found JXTA to be very useful in developing this self-healing system. In the future, we plan on benchmarking the performance of the system and provide comparative analysis with existing trading systems. These experiments and measurements include:

1. Measure and analyze the maximum and average number of trade executed per second by introducing failure rates at manager nodes.
2. Analyze the decrease in the number of trades executed per second with the increase in number of Backup nodes.
3. Measure gain in throughput and response time by introducing messaging queues instead of threads.
4. Compare the cost of collecting the data from the P2P network instead of directly grabbing it from the source.

References

[ACKLW 02] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, Dan Werthimer (2002): "SETI@home: an experiment in public-resource computing" *Commun. ACM* 45, 11 (Nov. 2002), 56-61. [http://doi.acm.org/10.1145/581571.581573]

[CET 06] (2006): "ROME: RSS and Atom Utilities for Java" [https://rome.dev.java.net/]

[CINC 06] Clarifi, Inc. [http://www.clarifi.com/server_data_caching.htm]

[EA 93] Alexander Elder, (1993): "Trading For A Living", 1993.

[FGCBG 97] Armando Fox, Steven Gribble, Yatin Chawathe, Eric Brewer, and Paul Gauthier, (1997): " Cluster-based Scalable Network Services ", Sixteenth ACM Symposium on Operating System Principles, Oct. 1997.

[HP 91] Norman C. Hutchinson, Larry L Peterson, (1991): "The x-Kernel: An Architecture for Implementing Network Protocols", IEEE Transactions on Software Engineering, 17, 1, pgs. 64-76, January 1991.

[IB 06] Interactive Brokers, LLC. [http://www.interactivebrokers.com]

[JHP 06] Jericho HTML Parser, (2006) [http://jerichohtml.sourceforge.net/doc/index.html]

[JXTA 06] Sun Microsystem, Inc, (2005) "JXTA v2.3.x: Java Programers Guide" [http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf]

[JFC 06] JfreeChart, Object Refinery Limited, (2006) [http://www.jfree.org/jfreechart/]

[LS 02] Sheng Liang, (2002): "The Java™ Native Interface Programmer's Guide and Specification", 2002. [http://java.sun.com/docs/books/jni/]

[MKDK 06] Ali Hisham Malik, Farhan Saleem Khan, Stuart Duerson, Victor Kovalev, (2005): "Applications of Reinforcement Learning in Online Stock Trading Systems"

[MAH 06] Ali Hisham Malik, (2006): "Recommendation Framework for Stock Trading using Pseudo-P2P Network"

[SBL 99] Yashushi Saito, Brian N. Bershad, Henry M. Levy, (1999): "Manageability, Availability, and Performance in Porcupine: A Highly Scalable Cluster-based Mail Service", 17th ACM Symposium on Operating System Principles, OS Review, Volume 33, Number 5, Dec. 1999.

[SKYPE 06] Skype Limited, (2006) [http://www.skype.com]

[SM 98] Peter Michael Melliar-Smith, Louise Elizabeth Moser, (1998): "Surviving Network Partitioning" *Computer*, vol. 31, no. 3, pp. 62-68, Mar., 1998.

[RM 01] M. Ripeanu (2001): "[15] Peer-to-Peer Architecture Case Study: Gnutella Network," *P2P*, p. 0099, First International Conference on Peer-to-Peer Computing (P2P'01).

[TBOLT 06] TradeBolt, LLC. [http://www.tradebolt.com}

[TC-JXTA 06] Trading Center [http://tradingcenter.jxta.org]

[TSTAT 06] TradeStation Securities [http://www.tradestation.com/]

[YF 06] Yahoo! Finance [http://finance.yahoo.com]

Appendix A

Trading Simulation using Stochastic Trading Signals

Apart from experimenting with trading strategies outlined in [TODO reference the trading strategies paper], it was hypothesized that the stochastic trading signals would be able to capture the good trading points, however the best set of stochastic trading signals to use would vary depending on bearish and bullish moods in the markets. To analyze this hypothesis, a simulation environment was developed which used a brute-force search over parameters for stochastic trading signals as well as some other trade optimization parameters which are as follows:

1. Bad Transaction Limit

Number of bad transactions after which the trading strategy is changed. A transaction is defined as the pair of buy and sell. Currently if we sold at a price lower than the price that we bought the stock at, then the transaction is considered to be bad.

2. Lookback Transactions/Period

Flag that indicates whether to select the new trading signal stream based on the performance of the trading signal stream over a certain number of transactions or over a number of trading periods. This separation is important since a trading signal may be performing well over the previous transactions, however the

duration of the transaction may be so long that the trading signal is not desirable. Hence the lookback over periods might be more desirable in an active and aggressive trading environment, whereas the lookback on transaction might be more useful for long-term investing.

3. Lookback Count

The number of transactions/period over which the performance of a trading signal streams is analyzed and the best performing stream is selected.

4. Force Sell At Percentage Value

The percentage drop in the price of the stock since the stock was bought, after which the stock is sold regardless of whether the trading signal is buy or sell.

5. Number of Different Smoothed Stochastic Signals

The desired number of smoothed stochastic signals of different widths to generate.

6. Number of Different Raw Stochastic Signals

The desired number of raw stochastic signals of different widths to generate.

7. Initial Smooth Stochastic Window Width

The initial value of period over which the smooth stochastic signals are generated. All other smooth stochastic are generated based on a step over this initial value

8. Smooth Stochastic Step Size

The step size for generating smooth stochastic window, starting from the initial set of smooth stochastic signals generated using the provided window width all the way to the last one when the generated set of smooth stochastic signals is equal to the desired number of smooth stochastic signals.

9. Simple/Complex Stochastic Window Step

This flag indicates whether to use simple step iterations generating window width for raw stochastic signals or to use a more sophisticate curve fitting formula. The curve fitting formula has been configured to generate the width 5,14, 21 for smoothing window width of 3. Thus curve fitting formula tries to mimic the generally accepted values of raw stochastic windows.

10. Signal Generation Strategy

This flag indicates one of the following signal generation strategy is being used:

11. Voted Signal

Voted signal means that the at each trade point, the decision to buy or sell is taken by aggregating the signal generated by the different sets of stochastic signals.

12. Reactive Signal Changing

This indicates that the trading signals selected to do the trading should be reevaluated after a certain number of consecutive bad transactions.

13. Continuous Signal Changing

This indicates that the trading signal used for trading should be reevaluated after every iteration.

14. Shorting Stock Percentage

The percentage of amount corresponding to the current capital that will be used to perform shorting. Using shorting, we maximize our profits in case the signal was correct. However, in case of a bad signal, the losses are also increased and to avoid going bankrupt, only a certain amount of the current capital we have is used.

Back-testing of this trading strategy showed good performance on some stocks. However, more analysis needs to be performed, especially to discover the common properties of the stock symbols and the trading period where this trading strategy is most suitable. Furthermore, selection of brute-force parameter search needs to be reanalyzed as the search space of the parameters is quite wide and increases exponentially with the addition of each new parameter.