A Parallel Workflow for Real-time Correlation and Clustering of High-Frequency Stock Market Data

Camilo Rostoker, Alan Wagner and Holger Hoos

University of British Columbia
Department of Computer Science
Vancouver, BC V6T1Z4 Canada
{rostokec, wagner, hoos}@cs.ubc.ca

Abstract

We investigate the design and implementation of a parallel workflow environment targeted towards the financial industry. The system performs real-time correlation analysis and clustering to identify trends within streaming high-frequency intra-day trading data. Our system utilizes state-of-the-art methods to optimize the delivery of computationally-expensive real-time stock market data analysis, with direct applications in automated/algorithmic trading as well as knowledge discovery in high-throughput electronic exchanges. This paper describes the design of the system including the key online parallel algorithms for robust correlation calculation and clique-based clustering using stochastic local search. We evaluate the performance and scalability of the system, followed by a preliminary analysis of the results using data from the Toronto Stock Exchange.

1. Introduction

Due to the recent shift away from traditional (physical) stock markets to electronic markets (partially or completely), the ability to perform real-time computational analysis has become a major focus for many financial institutions. The evidence is a growing number of high-throughput electronic exchanges offering real-time, low-latency data feeds. The result is an electronic trading environment where in order to out-compete, you not only need to out-smart, but you also need to out-compute. It is already accepted in the industry that both computational speed and computational intelligence are critical components for any successful financial trading platform [8, 1]. A strict requirement for such a system is that the underlying algorithms and mathematical models must be able to process and analyze thousands of high-frequency, multi-dimensional and inhomogeneous time series, in real-time and in an online fashion. Unfortunately, like in other real-world problems, the issue of data processing and analysis becomes increasingly complex as the size and frequency of data increases. As a result, many new challenges and opportunities exist for applications of parallel computing in the finance industry.

This work builds on a long list of recent research that examines the stock market network as a complex system [17, 2, 3, 4, 26, 20]. A stock market network, often referred to as a *market graph*, is a model in which stocks are nodes, and links between nodes represent a relationship between the stocks. Previous research on stock market networks have focused on historical data; in this context the edges represent the cross-correlation of the two stocks' log daily price return over a given time period. The exact configuration and topology of this type of market graph reflects the complex interactions within the stock market network over the time period in which the data was collected. Instead of representing interaction patterns spanning several years, our approach considers the dynamics involved with the intra-day evolution of the market graph. The topology of this graph, which we call the *dynamic intra-day market graph*, is only a snapshot of the stock market at a specific moment — it *evolves* over time. Valuable information can be extracted from a market graph at any given point in time, and further computational analysis can be employed to find clusters or "cliques" of stocks that exhibit highly correlated trading patterns. A clique is a fully connected set of vertices in an undirected graph, *i.e.*, between any two vertices in the set there exists an edge. A clique is *maximal* if and only if it cannot be expanded by adding another vertex from the given graph, and a *maximum clique* is the maximal clique(s)

1-4244-0910-1/07/$20.00

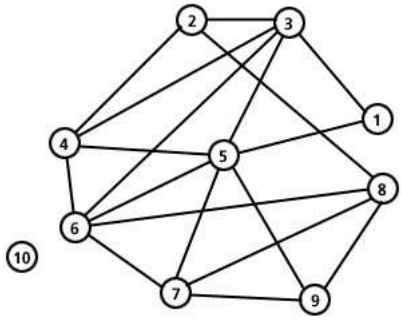

Figure 1. A graph with maximum clique size of 4.

with greatest cardinality. The problem of finding a maximum clique in a graph, also known as MAX-CLIQUE, is a well-known NP-hard combinatorial optimization problem. Figure 1 shows a small graph with 10 vertices and maximum clique size of 4 (vertices $\{3,4,5,6\}$).

While existing market graph studies involving long-term historical data focus on price-price interaction patterns, our method utilizes newly available high-frequency intra-day data to compute short-term technical indicators. Similar to existing approaches, we are looking for cliques of stocks with highly correlated trading activity, but instead we define "trading activity" to be the value of a user-defined technical indicator. Using this data, our system is able to compute and analyze the real-time evolution of the dynamic intra-day market graph.

Given the dynamic, online nature of the intra-day market graph, the challenge was to design a system with two key properties:

- The system must be able to operate on a real-time flow of data and thus the algorithms in the system must be online and attempt to minimize the calculation for each of the steps.

- The system must be able to scale to the size of the input data for the computationally-intensive methods such as correlation and clique-finding, and obtain high quality solutions in a reasonable amount of time.

The Toronto Stock Exchange (TSX), for example, is a relatively small exchange trading just under 2,000 securities, averaging approximately 1,000,000 quotes and 200,000 trades per day. There are much larger exchanges, including new electronic crossing networks (ECNs) which provide transparent access to multiple stock exchanges.

In order to update the intra-day market graph in real-time, we investigated the use of two algorithms for performing the compute intensive phases of the workflow. To detect correlations between stocks, we use a robust correlation technique called the Maronna method [18] because of its ability to produce higher quality solutions and its iterative nature that make it well suited for online computation. Although previous work had investigated its scalability as an offline algorithm [6], its use in an online setting has not been investigated. To find cliques we use a state-of-the-art stochastic local search algorithm called Phased Local Search (PLS) [23]. To make it applicable in our system, we extended PLS such that it works on online data and in parallel on a number of processors. We also modified the algorithm to find sets of cliques rather than a single maximum clique.

This paper makes the following contributions. It introduces and presents a solution for the parallel real-time computation of intra-day market graphs to aid in the analysis of stock market data. In solving this problem we adapted two offline algorithms, the Maronna method and Phased Local Search, and investigated their scalability with respect to the response time of the system. We constructed a prototype workflow environment to experiment with these algorithms in an online setting that allows for the dynamic adjustment of different controls, such as thresholding and accuracy, as well as the addition of visualization clients for viewing results.

The remainder of this paper is structured as follows. Section 2 presents a general overview of the system. In Section 3 we describe the time series data used in the correlation analysis. Section 4 describes our online implementation of the Maronna method. Section 5 describes our parallel online clique-based clustering approach, and discusses how it can be effectively used as an "embarrassing parallel" algorithm. In Section 6 we evaluate the system with respect to its response time and validate its ability to discover patterns and trends in high-frequency stock market data. Finally in Section 7 we present our conclusions.

2. Overview of the System

Time	Symbol	Bid Price	Ask Price	Bid Size	Ask Size
13:23:01	AE.DB	161.506	173.804	454	104
13:23:01	RGL	31.502	23.603	303	13
13:23:02	RIM	103.305	104.505	135	65
13:23:03	TRP	37.509	38.009	429	829

Table 1. Sample time-ordered quote data for four stocks.

Our system uses streaming quote data to construct the dynamic intra-day market graph. Each time-stamped quote specifies, among other things, the current best bid and ask

price for a given stock. The *bid price* is the highest price someone is willing to pay for a stock, and the *ask price* is the lowest price someone is willing to sell a stock. An example of quote data is shown in Table 1. In order to

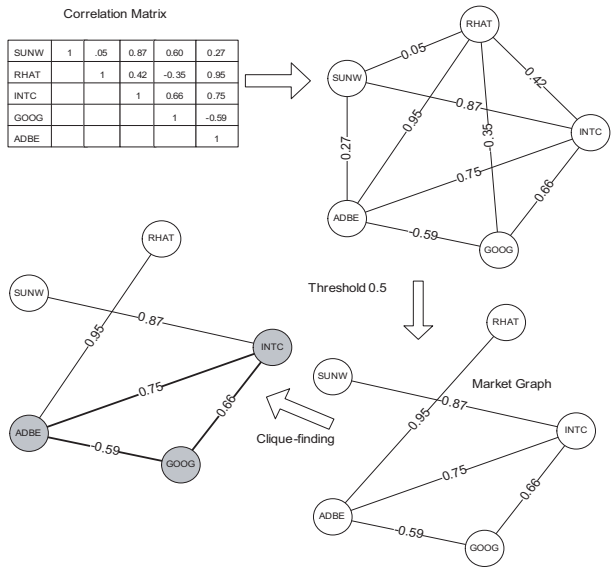

Figure 2. Example of a market graph and clique structure.

create a network model representation, we correlate a user-defined technical indicator (e.g., rate of change in the price of a stock). Details on how technical indicators are used by the system to create a similarity measure is given in Section 3. The values of a particular technical indicator give a time series for each stock. The time series between all pairs of stocks are correlated, resulting in a completely connected market graph. Thresholding is then used to filter out edges with low correlation, and finally PLS identifies maximal cliques in the resulting market graph. An example of this process for five stocks is illustrated in Figure 2.

2.1. Design of the System

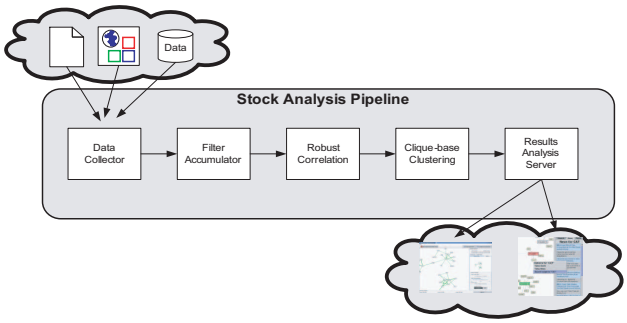

Figure 3. Multistage stock analysis pipeline.

As shown in Figure 3, the system architecture is a multi-stage pipeline. It is comprised of the following five stages. The **data collector** is the initial stage that receives stock quotes from an Internet feed, file or database. The **filter and accumulator** stage receives unfiltered data items from the data collector (e.g., a single quote), performs any required filtering of the data (e.g., interpolation for missing data) and calculates the technical indicator samples. After the indicator values are sent to the **correlation** stage, all pair-wise correlation coefficients are computed and sent to the **clustering** stage. The correlation and clustering stages are the most computationally intensive stages; further detail on each are presented in Sections 4 and 5, respectively. The final stage, the **results analysis server**, receives the set of clusters (maximal cliques) and packages the data into XML, which is then transmitted to an external visualization client.

2.2. Details of the Design

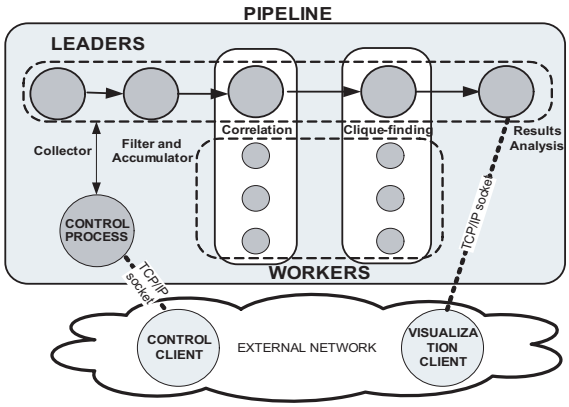

Figure 4. Process structure of the pipeline.

Message-passing in the system is implemented in MPI, including the correlation and clique-finding stages which are parallelized using a manager-worker architecture. The overall process structure of the system is shown in Figure 4 where the two middle stages may contain an arbitrary number of workers. As a batch of quotes enter the system, the manager routine activates workers to perform a parallel correlation calculation. Upon completion, the results are passed onto the the clique-finding stage which in turn activates workers to search for cliques. The system can perform load-balancing between stages to avoid any one stage from becoming a bottleneck. This is accomplished by either adjusting the work load or increasing the number of workers allocated to the middle two stages, thus minimize the effect of these two stages on performance (as measured by the response time of the system).

The control process shown in Figure 4 uses an intra-communicator to send messages to the leader group, which is composed of one process from each stage of the pipeline.

The control process accepts socket connections from an external client, allowing it to initialize the system as well as to dynamically update control parameters. The parameters that can be dynamically updated include:

- threshold value, which sets the threshold for the correlation values above which the correlation between two stocks induces an edge in the graph;
- batch sizes, sets the amount of data to accumulate before sending the data onto the next stage;
- parameters for correlation and clique-finding that can adjust the balance between the quality of the solution and the length of execution time;

3. Similarity Measures

Deciding on an appropriate measure for correlation is a difficult problem which depends on many factors, such as the time scale on which patterns are sought, how and by whom the results are to be interpreted, and the number and variety of stocks included in the analysis. Different similarity functions will produce vastly different market graphs. In previous market graph research [17, 2, 3, 4], which model the long-term (historical) market graph, the standard measure is the log return $R_i = log(P_i(t)) - log(P_i(t-1))$, where $P_i(t)$ is the closing price of stock i at time t (where time is measured in days). Clearly this exact measure cannot be used on intra-day data, as it would result in only a single data sample. Even if R_i was evaluated using $P_i(t)$ as the price after each trade, the number of trades within a day for most stocks are too few to justify a statistically significant correlation.

Fortunately, the rapid advancement of electronic stock markets has enabled real-time access to high-frequency intra-day quote data. Using this newly available data, our system constructs a dynamic intra-day market graph whose edge weights correspond to the correlation coefficients between pairs of evenly spaced technical indicator time series. Our use of short-term technical indicators is motivated by Dempster et al., who use genetic programming to construct an optimal portfolio of short-term technical indicators for use within their automated trading system [9, 11, 10]. For simplicity, we experimented with three basic indicators: the Quote Frequency Simple Moving Average (QFSMA), Quote Volume Simple Moving Average (QMSMA) and the Quote Midprice Rate of Change (QMROC). The quote midprice is a close approximation of the true price, while other measurements such as the spread (difference between bid/ask), quote frequency and arrival times are all closely linked with liquidity and volatility. These higher-frequency time series represent alternative views of the underlying market microstructure, from which potentially economical information can be extracted [5, 26, 15, 19].

Due to the sporadic nature of the data, if the approach described above is applied to a set of stocks, the sample points will not be evenly aligned in time. Since standard correlation-based analysis assumes homogenous time series, we transform the raw inhomogeneous time series into a homogenous one through interpolation and aggregation [8, 25]. It has also been suggested that patterns spanning multiple time scales are somehow connected to the emergent behaviour of the heterogenous investors (agents) participating in the market [16]. Other work suggests that the temporal irregularity of these time series should be considered a feature of the system, rather than a problem which needs to be eliminated [1]. But how can we compute and analyze such correlations across inhomogeneous time series? This is the question we wish to address by treating the presence of new data samples (e.g., quotes) as *events*; that is, we sample the data stream at a regular interval, but do not use interpolation or aggregation to homogenize the time series. This approach, however, also presents challenges of its own, raising questions such as "How can such patterns be interpreted?", or "What does it mean for these events to be correlated?".

There is, however, an obvious trade-off in the frequency at which data is sampled. Sampling more frequently brings the analysis closer to 'real time'. The problem is that for stocks with infrequent quote updates, interpolation produces a stream of constant values, causing the correlation analysis to produce spurious results. On the other hand, if we increase the sampling window to say one minute, the number of constant values decreases, but the result/solution response time also increases. The issue of how to deal with inhomogeneous, high-frequency stock market data is an open problem in high-frequency finance [22, 8, 1], and thus there are no obvious answers to these questions. By providing a framework such as the one proposed in this paper, we offer a novel tool for investors and market analysts to interactively explore this new and interesting information space.

4. Correlation Calculation

Calculating the correlation between a collection of random variables is a standard technique used to determine the strength of a linear relationship between variables. It is widely used in finance for portfolio optimization, derivatives pricing, risk management and pairs/spread trading [22]. A correlation matrix records the correlation coefficients between all pairs of variables, in our case stocks in the exchange. Unfortunately, correlation matrices are very sensitive to the presence of outliers in the data. As a result, robust techniques have been developed to produce estimates

of the correlation. These robust methods require more computation and are not widely used because of their computational requirements. In this paper we use a robust technique called the Maronna method [18]. The computational time complexity of the Maronna method is $\mathcal{O}(np^2)$ for p variables and n data samples. However, as shown in [6], it can be easily parallelized and scales to large numbers of processors. Scalability tests were performed on a gene expression dataset with 6068 variables and synthetic datasets with 10,000 variables. The 6068 variable correlation matrix was calculated in 15.5 seconds using 128 processors (3GHz Xeons). In our case, as we show in Section 6, the correlation matrix for the data from the Toronto stock exchange, which trades just under 2000 stocks, can be calculated in less than 4 seconds using 18 processors. Details on the Maronna Method and its parallelization are described in [18].

The Maronna method can be structured into an embarrassing parallel algorithm where each pair-wise correlation coefficient can be calculated independently. We use a manager-worker architecture; the manager assigns to each worker processes a batch of jobs to compute, where a job corresponds to calculating a single correlation coefficient. Upon completion the worker sends back the batch of results and subsequently asks for another batch. This continues until the correlation computation is complete. The pair-wise correlation calculation is an iterative process where it is possible to specify the accuracy of the correlation as well as limit the number of iterations. Experiments from our previous work show that the number of iterations required to reach a given accuracy depends on the outliers in the data. Each iteration reduces the effect the outlier has on the coefficient value. The accuracy and number of iterations provide controls that can be used reduce the computation time, offering a flexible trade-off between the quality of the results and response time.

As mentioned, this use of the Maronna method differs from our previous work in that it is online rather than offline. Thus, as stock updates flow into the system the correlation values need to be constantly updated. We restructured the algorithm for the Maronna method to allow us to independently re-compute the correlation value between any two stocks. Thus, as a stock quote update enters the system we can re-compute the correlation coefficient ρ_{ij} by calling ρ_{ij} = maronna_correlation(X,Y,n,ϵ,$limit$) where X and Y are the data columns for stock i and j respectively, and n is the number of samples in X and Y. The parameter ϵ is the precision at which the iteration of the correlation of two variables stop, while $limit$ specifies the maximum number of iterations. Each time a new quote is received for stock i, its correlation coefficients with all other stocks need to be updated; that is, we need to compute ρ_{ij} for all $j \in \{1, ..., p\} \setminus \{i\}$ where $S = \{s_1, \ldots, s_p\}$ denotes the set of all stocks. It is not efficient to perform this calculation on a per-update basis; instead, it is preferable to define a time window where the accumulator stage collects the stock updates into a batch that is then forwarded to the Maronna stage. The Maronna stage then broadcasts the updated samples to the workers and the processor farm is started. A manager process dispatches correlation tasks to workers which perform the independent correlation computation and return the result. In a given time window, k, there will be a set of stocks $S_k^* \subseteq S$ corresponding to the stocks that received at least one quote update within that window. If $m = |S_k^*|$, then the number of updates to the $p \times p$ correlation matrix is $p(p-1)/2 - (p-m)(p-m-1)/2 = mp - m(m-1)/2$. Notice that as m, the batch size, increases there is less wasted computation, since the coefficients between the stocks within a batch are calculated only once. Thus, there is a trade-off between having a small batch size that results in some wasted calculation but faster response time versus larger batch sizes over larger time windows that give a slower response time.

5. Clique-based Clustering

Clustering of the market graph is done by finding a large set of maximal cliques using Phased Local Search (PLS), a state-of-the-art Stochastic Local Search (SLS) algorithm for finding maximum cliques in unweighted graphs [23]. Stochastic local search is one of the most prominent and widely used approaches for solving computationally difficult combinatorial problems [14]. It is based on the general idea of iteratively expanding or improving a current candidate solution by means of small modifications, using randomized decisions to avoid or overcome stagnation of the search in locally optimal solutions. The PLS algorithm has been shown to achieve state-of-the-art performance on a broad range of MAX-CLIQUE problems. It is derived from DLS-MC [24], an earlier SLS algorithm for MAX-CLIQUE. Both algorithms alternate between phases of iterative improvement, where suitable vertices are added to the current clique, and plateau search, where vertices in the current clique are swapped out and replaced by other vertices; furthermore, dynamic vertex penalties are used to achieve effective search diversification, which prevents the search from stagnating in locally maximal, but sub-optimal cliques. Unlike DLS-MC, PLS has no parameters that control the behaviour of the search process.

For use in the context of our system, we have modified PLS in two respects: Firstly, we have parallelized the search process using the simple, yet effective multiple independent runs approach and modified it to work on a dynamically changing input graph. This is motivated by the need to analyze the rapidly changing market graph as efficiently as possible. Secondly, we have altered the algorithm to not

only find one maximum clique, but a set of maximal cliques. This allows us to find a potentially large number of clusters of correlated stocks, each of which may be of interest to an analyst using our system. In the following, we describe each of these modifications in more details. For further information on the PLS algorithm and these new modifications, we refer to the literature [23, 25].

5.1. Parallel PLS

One of the big advantages of SLS algorithms over other approaches for solving hard combinatorial problems is the fact that in principle, they can be parallelized very easily by simply performing a number of runs independently in parallel. Obviously, this parallelization strategy makes no sense for deterministic search algorithms (unless they are initialized at carefully chosen starting points). For randomized algorithms, it is well-known that the effectiveness of parallelization by means of multiple independent runs depends on the respective run-time distributions (RTDs) [14]. RTDs capture the distribution of time needed by a randomized search algorithm, such as PLS, to reach (or exceed) a specific solution quality (here: clique size) when applied to a given problem instance. It has been shown analytically that by performing multiple independent runs on p processors, a parallelization speedup of p is achieved if, and only if, the RTD for the given algorithm and problem instance is an exponential distribution, *i.e.*, has a cumulative distribution function of the form $f(x) = 1 - e^{\lambda x}$ [27]. Many high-performance SLS algorithms, including PLS, tend to exhibit approximately exponential RTDs when applied to hard problem instances [14]. Therefore, close-to-optimal parallelization speedup can be achieved by performing multiple independent runs of the algorithm.

In a realistic setting, where the market graph changes dynamically in response to real-time stock data and user interaction, maximum clique sizes will not be known *a priori*. Therefore, we run each PLS process for the same amount of time and subsequently determine which of them have achieved the largest clique size. During the initial search phase the probability of finding an optimal solution within each given time unit is significantly reduced, which limits the number of processors for which close-to optimal speedup can be obtained. However, this effect is much less pronounced for hard than for easy problem instances, and for the latter, the efficiency of the clique finding algorithm is clearly much less critical. It should be noted that more complex parallelization strategies could be applied to PLS, including cooperative schemes in which the search processes running on the individual processors share information [7], for example in the form of vertices that have been found to belong to large cliques or frequently encountered suboptimal cliques.

5.2. Online PLS

As a consequence of the frequent changes of the stock data monitored by our system, the market graph analyzed by PLS is subject to dynamic modifications. In response to any such modification, we need to solve the clique finding problem for the new graph before further changes may invalidate the new solutions. In principle, the scalable parallelization of PLS allows us to address this challenge in a very simple manner, namely by solving the problem instance arising from each modification of the market graph by performing parallel PLS on sufficiently many processors. Furthermore, SLS algorithms such as PLS have a desirable 'anytime property', in that at any point during the search they can return the best candidate solution found so far. Therefore, even if parallel PLS has not found a maximum clique before the market graph changes, it can produce a meaningful suboptimal solution.

However, this simple scheme for solving online clique finding problems does not take into account that in many cases, the changes occurring with each update of the market graph are fairly small. It has been previously shown for other combinatorial problems subject to dynamic changes over time that SLS algorithms often work best in finding solutions to the corresponding series of related problem instances when not forced to restart the search process after each change (see, *e.g.*, [13]). Instead, it is beneficial to continue the search across changes in the input data with as little disruption as possible. This approach is known as trajectory continuation (TC) and can be easily applied to most SLS algorithms; in the case of PLS, it works as follows. When the input graph changes, first the data structures representing the graph are updated. Next, the sets of vertices used in the iterative improvement and plateau phases are adjusted to accurately reflect the effects of added or removed edges. Finally, the current (working) clique and the best (saved) clique found so far are repaired if they have become invalid as a result of edge removals; this is done by deleting vertices incident to the missing edges until the remaining set of vertices is fully connected again. To illustrate the performance improvements achieved by the use of trajectory continuation, we evaluated PLS with and without trajectory continuation on several dynamic graph series based on a previously studied data set of 6556 stocks [3]. Table 2 shows the cumulated number of search steps (vertex selections) required by PLS to find maximum cliques at all stages in each of three series of graphs; the add and remove series represent successive graphs in which the number of edges is monotone increasing and decreasing, respectively, while the mixed series is a random ordering of the add and remove series. As shown in the table, trajectory continuation improves the search performance on all three series, with an average speedup of 101%. Speedup on the mixed series

series	# stages	# search steps		% speedup
		no TC	TC	
add	10	14,099	6,151	129
remove	10	10,865	5,937	83
mixed	20	29,735	15,425	92

Table 2. Performance differences between PLS without and with trajectory continuation on dynamic market graph series.

was slightly lower at 92%, which represents a more real-life scenario (where edges are both added and removed between successive stages). Although the speedup afforded by the online adaptation is not overwhelming, when operating in a highly time-constrained environment, every bit of speedup counts.

5.3. Recording sets of maximal cliques

Our final modification to the PLS algorithm is motivated by the fact that in our clique-based analysis of the market graph, we are interested in finding not just a single maximum clique, but a large and diverse set of maximal cliques. As a first way of addressing this goal, we modified online parallel PLS to record the set of all maximal cliques encountered since the initialization of the search process or last change in the graph. Currently, the sets of maximal cliques collected by parallel online PLS are combined into the so-called 1-clique graph (an instance of a k-clique graph), whose vertices corresponds to the stocks involved in at least one maximal clique and whose edges are precisely those found in the set of maximal cliques [12]. We have a simple visualization client that can be used to explore this graph. The clique data collected by our algorithm contains additional useful information that can be extracted by further analysis. For example, we have now started analyzing the clique overlap structure, which has been shown to exhibit small-world scaling properties [21].

6. Evaluation

The current system was tested on a dataset consisting of the Trade and Quote (TAQ) data from the Toronto Stock Exchange (TSX) for March of 2005. We choose different days and different times within those days as well as testing on synthetic data to ensure the correctness of the techniques.

We evaluate two different aspects of the system. First, in Section 6.1 and 6.2, we provide evidence to demonstrate the type of trends that we obtained from running our system and collecting results. Second, we evaluate the response time of the system, the time required for the system to react to changes in the input.

We note here that the task of evaluating a system composed of many interacting components significantly increases the parameter space, thus making a thorough empirical analysis much more complicated. Therefore, we have chosen to focus our evaluation on highlighting the performance of the system *as a whole*, since individual performance and scalability results for Maronna and PLS are documented in their respective publications [6, 25]. Furthermore, a concise performance evaluation is heavily dependent on the underlying hardware configuration, and thus we provide only basic proof-of-concept results demonstrating the capability of such a system.

6.1. Homogeneous Time Series

Interpreting homogenous time series correlations is a relatively easy task. Determining the cause of the correlation is another question altogether (and hence out of the scope of this paper). As discussed in Section 3, applying standard time series analysis to high-frequency data presents many challenges, e.g., the presence of a many consecutive constant values. For example, we found that correlating metrics using a sampling frequency of less than 10 seconds often resulted in an entire set of constant values, which gave rise to an undefined correlation coefficient (since standard deviations of zero occur). Our original hypothesis was that using high-frequency data would eliminate this problem, but we see now that while the frequency of the input data has increased, much of the data is repetitive, resulting in little or no change in the underlying time series. As a result, we have a small but representative set of examples illustrating the type of patterns we wish to extract from the intra-day market graph. For this analysis we did not perform any backtesting over a number of days; instead, we extracted a single day of trading data which we calculated as a 'normal' day on the TSX[1]. Figure 5 shows five stocks with correlated QFSMA indicators calculated using a 1 minute sampling interval with a sample queue size of 100. The stocks, excluding Cognos, are loosely related in the sense they fall within the resource/mining sector. All pair-wise correlation coefficients for these time series are between 0.9 and 1.0. While at first glance these values may seem surprisingly high, the reason is that Maronna down-weights samples which appear to be outliers and would normally decrease the value of the correlation coefficient. Notice how the indicator values appear highly correlated for some time, and then they all suddenly drop to a very weak signal — this type of activity could represent the situation when all the stocks were simultaneously correcting (reversing a recent gain/loss). The detection of such correlated activity could be a valuable tool

[1] We did this by calculating averages for volume, price volatility and number of quotes over the entire month (March 2005), and then chose the day showing the least deviation from these averages (March 22)

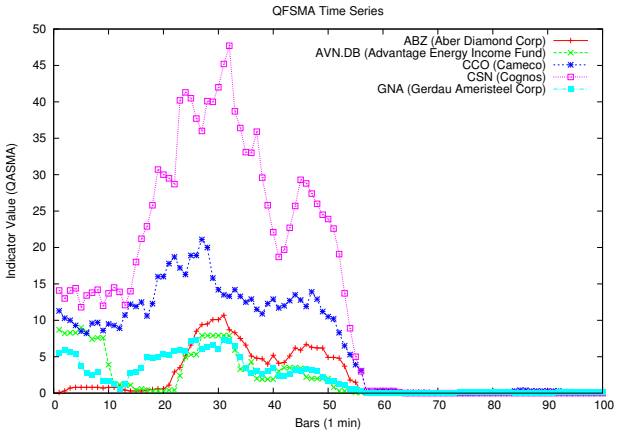

Figure 5. Resource/mining stocks showing correlated QFSMA indicators with 1 minute sampling interval.

for momentum traders who rely on the ability to predict short-term movements. Figure 6 shows stocks which ex-

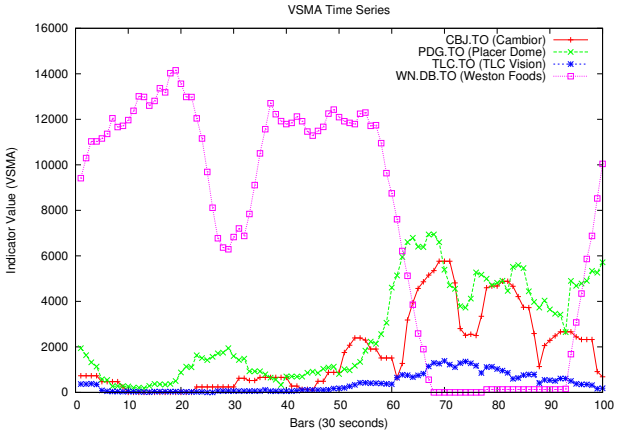

Figure 6. Gold stocks showing correlated QVSMA indicators with a 30 second sampling interval.

hibit correlated QVSMA indicators using a 30 second sampling interval. The resulting time series for Cambior and PlacerDome (gold mining companies) are positively correlated with $\rho = 0.84$. TLC Vision is also found to be positively correlated with Cambior ($\rho = 0.95$) and PlacerDome ($\rho = 0.80$), although the magnitude of change in the indicator series is noticeably less (which may indicate a spurious result due to noise). Weston Foods, on the other hand, exhibits a QVSMA indicator series which is highly negatively correlated to all three stocks, with an average correlation coefficient of $\rho = -0.82$.

6.2. Inhomogeneous Time Series

Analyzing correlated behaviour from inhomogeneous time series is difficult because of the potential variations in frequencies between the time series. For example, consider a scenario where a stock s_i has a price pattern over time window t_i defined by $P(s_i)$, which is activated each day at a particular time, in response to daily news relevant to the company. Now consider another stock s_j with a price pattern over time window t_j defined by $P(s_j)$; the time windows t_i and t_j can vary significantly, and as such the correlated price movements will not be detected by traditional time series correlation methods. How exactly to extract meaningful economic information from such results is another question, and better left to our target users with experience in short-term trading.

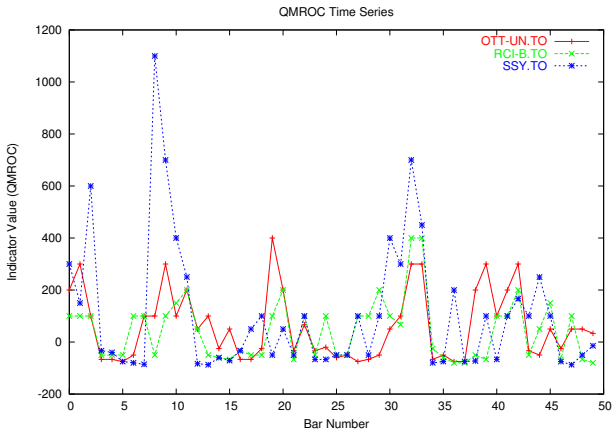

Figure 7. Correlated event-based QMROC patterns

Figure 7 shows three stocks exhibiting correlated event-based patterns using the QMROC indicator. The three stocks are Otelco (wireless telecom services), Rogers Communications, and Spectrum Signals (software developer). The average correlation coefficient of these time series is 0.52; while not convincingly high, a quick glance a the chart shows a clear co-movement in the indicator values. Figure 8 shows the same data points when they are shown in their correct time series order. It is difficult to discern a relationship between those same stocks.

6.3. System flexibility and scalability

Our current testing environment is a small compute cluster comprised of 14 dual Intel Xeon 3.06GHz CPU's with 2GB RAM, and another small cluster with 6 dual Intel Xeon 2.0GHz CPU's with 4GB RAM. We tested the scalability and throughput of the system by varying the number of processors and measuring the time delay through the pipeline.

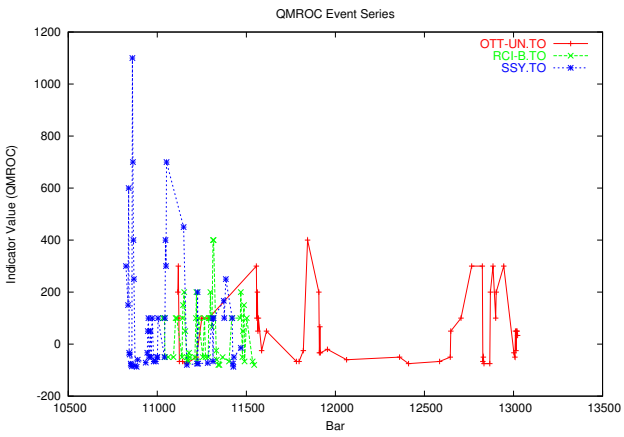

Figure 8. The underlying time series for the QMROC events in Figure 7.

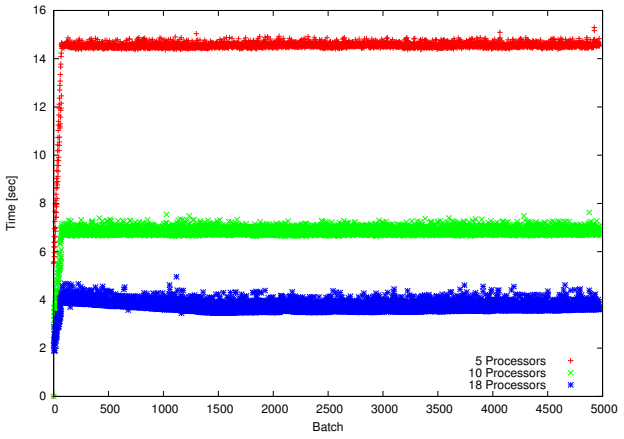

Figure 9. Response time for a dataset with 2000 stocks, batch size 2000, and three processor configurations.

Figure 9 shows the result of executing the system with 5, 10 and 18 processors on Maronna and PLS. We do not include the processors executing the single processes corresponding to the first two stages, the last stage and the control process. The response time increases at the beginning of each run because as the input queue for each stock fills, the size of the samples for correlation increases. Eventually the queues fill, after which the sample sizes remain constant. A communication protocol was established between the two stages so that they can be overlapped onto the same processors. This is possible because when one task is executing, the other need not be, thus guaranteeing that only one of the two tasks be utilizing the system resources at any given time. For example, when Maronna is computing correlation values, PLS is waiting to receive these updated values; then when the correlation task is complete, PLS find cliques

Number of Processors	Avg. response time (seconds)	Speedup (w.r.t. 5)
5	14.49	1
10	6.83	2.12
18	3.74	3.87

Table 3. Speedup for workflow environment with 2000 stocks and batch size 2000.

while Maronna waits for new batches of data. The number of iterations PLS performs between each successive check is adaptive with respect to the data input frequency; i.e. if data is coming in fast and Maronna is continuously computing, then PLS does less work since the relevance of the results degrade with time. However, if the data rate is slow and Maronna has no work to perform, then PLS can utilize the spare compute resources and continue to refine the clique results. Table 3 shows the speedup results of the aver-

Number of Processors	Response time statistics (seconds) avg	med	stddev	var
5	14.49	14.55	0.69	0.48
10	6.83	6.85	0.32	0.10
18	3.74	3.73	0.23	0.054

Table 4. Response time statistics for varying processor configurations.

age response times for each of the series shown in Figure 9. Table 4 reports several response time statistics for various processor configurations. It is worth noting that the standard deviation (stddev) and variance (var) of the response times decrease when more CPU's are used, a result of the inherent load-balancing mechanism of the task farm architecture.

7. Conclusion

In this paper we have explored the design and prototype implementation of a highly parallel workflow environment capable of correlating and clustering time series data streams of a large portfolio of stocks with minimum response time. The system we propose, which is fully distributed and can run almost anywhere on any architecture, is built using MPI, an open-source and platform-independent API for high-performance message-passing applications.

First we introduced the individual components of the system, and describe how they interact in order to process the high-frequency data stream. We then introduced the

concept of the dynamic intra-day market graph, which utilizes a real-time correlation matrix to model the evolving interaction patterns between stocks. The correlation matrix is dynamically updated using an online parallel implementation of Maronna, a powerful correlation method robust to outliers and noisy data — a key feature for dealing with sporadic, highly-irregular time series. Finally, we presented an online parallel clique-finding algorithm which finds within a dynamic input graph, a large set of maximal cliques representing subsets of highly correlated stocks.

The need for high-performance computing in the finance industry is clear; the question is how to best approach the problem. We believe the best solution lies in developing scalable, online algorithms that are able to meet the demanding computational requirements imposed by the sporadic and unpredictable nature of high-frequency stock market data. To this end, we have proposed an approach which emphasizes a real-time market-wide analysis, along with a flexible system design which enables easy interchanging of computational components, thus creating a powerful framework for designing, testing and evaluating novel applications in high-frequency finance. With potential applications ranging from automated trading systems to exploratory knowledge discovery, our system empowers its users with a deeper insight into the complex underlying network structure of the stock market.

Acknowledgments: We thank Bernard Orenstein of Agents Pty Ltd, a proprietary trading R&D company, for his invaluable feedback. Also, we thank Kevin Hynd (CFA) for input concerning various advanced trading topics.

References

[1] J. B. Arseneau. At the edge of trading: Analyzing high frequency time-series data in real-time using computational intelligence (working paper).

[2] V. Boginski, S. Butenko, and P. M. Pardalos. Statistical analysis of financial networks. *Computational Statistics & Data Analysis*, 48:431–443, 2005.

[3] V. Boginski, S. Butenko, and P. M. Pardalos. Mining market data: A network approach. *Computers & Operations Research*, 33(11):3171–3184, 2006.

[4] V. Boginski, S. Butenko, and P. M. Pardalos. On structural properties of the market graph. *Innovation in Financial and Economic Networks*, pages 29–45, London.

[5] G. Caldarelli. Emergence of complexity in financial networks. In *Proceedings of the 23rd conference of CNLS Los Alamos*, 2003.

[6] J. Chilson, R. Ng, A. Wagner, and R. Zamar. Parallel computation of high-dimensional robust correlation and covariance matrices. *Algorithmica*, 45(3):403–431, 2006.

[7] T. Crainic. *Metaheuristic Optimization Via Memory and Evolution: Tabu Search and Scatter Search*. Kluwer Academic Publishers, Norwell, MA, USA, 2005.

[8] M. Dacorogna, R. Genay, U. A. Muller, R. Olsen, and O. Pictet. *Introduction to High-Frequency Finance*. Academic Press, 2001.

[9] M. Dempster and C. Jones. The profitability of intra-day fx trading using technical indicators, 2000.

[10] M. Dempster and C. Jones. Can channel pattern trading be profitably automated? *The European Journal of Finance*, 8(3):275–301, 2002.

[11] M. A. H. Dempster and C. M. Jones. A real-time adaptive trading system using genetic programming. *Quantitative Finance*, 1:397–413, 2001.

[12] M. Everett and S. Borgatti. Analyzing clique overlap. *Journal of the International Network for Social Network Analysis*, 21:49–61, 1998.

[13] H. H. Hoos and K. O'Neill. Stochastic local search methods for dynamic sat - an initial investigation. In *AAAI-2000 Workshop 'Leveraging Probability and Uncertainty in Computation'*, pages 22–26, 2000.

[14] H. H. Hoos and T. Stutzle. *Stochastic Local Search Foundations and Applications*. Morgan Kaufmann, 2005.

[15] J. Idicula. Highly interconnected subsystems of the stock market, 2004.

[16] B. LeBaron. Time scales, agents, and empirical finance. *Medium Econometrische Toepassingen (MET)*, 14(2), 2006.

[17] R. N. Mantegna. Hierarchical structure in financial markets. *Computer Physics Communications*, pages 153–156, 1999.

[18] R. Maronna. Robust m-estimators of multivariate location and scatter. *Annals of Statistics*, 4(1):51–67, 1976.

[19] K. V. Nesbitt and S. Barrass. Finding trading patterns in stock market data. *IEEE Computer Graphics and Applications*, 24(5):45–55, 2004.

[20] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[21] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, 2005.

[22] O. Precup and G. Iori. Cross-correlation measures in the high-frequency domain, 2005.

[23] W. Pullan. Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization*, 2006.

[24] W. Pullan and H. Hoos. Dynamic local search for the max-clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, 2006.

[25] C. Rostoker. A parallel stochastic local search algorithm for finding maximal cliques in a dynamic graph - with applications to real-time stock market analysis. Master's thesis, University of British Columbia, 2007.

[26] N. Vandewalle, F. Brisbois, and X. Tordoir. Non-random topology of stock markets. *Quantitative Finance*, 1(3):372–372, 2001.

[27] M. Verhoeven and E. Arrts. Parallel local search. *Journal of Heuristics*, 1(1):43–65, 1995.