Journal of Theoretical and Applied Information Technology
20th February 2013. Vol. 48 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

STUDY OF CEP BASED ON DYNAMIC DATA MANAGEMENT SYSTEM AND ALGORITHMIC TRADING

[1]KONG XIANGSHENG
[1]Department of Computer & Information, Xin Xiang University, Xin Xiang, China
E-mail: victor_kong@163.com

ABSTRACT

In recent years, dynamic data management systems and algorithmic trading systems have come to account for a majority of volume traded at the major US, European and Asia-Pacific financial markets. Complex event processing is a typical data processing technique which becomes the new spotlight of researches. Complex Event Processing over dynamic data management systems and algorithmic trading systems poses huge challenges with regard to efficient, scalable execution as well as expressive models and languages that account for the dynamics in long-running queries. In this paper we discuss the characteristics that a data event processing service should have in order to support in the best way the complex event pattern detection functionality, and present an assessment of a number of technologies that can be used to dynamic data. Especially we propose a corresponding event model and develop an algorithm that can efficiently detect complex event over event stream.

Keywords: *Dynamic Data Management System, Algorithmic Trading, Complex Event Processing, Volume Weighted Average Price, Complex Event Detection*

1. INTRODUCTION

Throughout the last years, the World Wide Web has moved from an Internet of documents to an Internet of services, and algorithmic trading (AT) for e-markets is more complex than electronic trading. AT for e-markets is the use of computer programs to enter trading orders with the computer algorithm deciding on characteristic of the order such as the timing, price, or quantity of the order and in many cases initiating the order without human intervention. In AT orders are placed with the algorithm which decides on various aspects of the order such as order price, size, timing of purchase etc [1]. Realizing AT for e-markets is a challenging task. One of the biggest challenges is to deal with a large amount of data produced in real-time. This is because e-markets involve a large number of users (possibly from all around the world) and have been growing in size rapidly over the last decade [2]. Other challenges include developing software components that interface effectively with the market feeds and handling the different types of data formats used to encode e-market transactions.

CEP provides flexibility in handling data in different formats without a pre-processing step and offers scalability in handling the increasing amount of data being produced in e-markets [3]. The conception of complex event that is typically expressed by means of patterns that declaratively specify the event sequences to be matched over a given data set origins from the research rule processing in active database.

An active DBMS could simulate a dynamic data management system (DDMS) through triggers, but is not optimized for such workloads, and even if support for state-of-the-art incremental view maintenance is present, performs very poorly. CEP Systems associate a precise semantics to the information items being processed: they are notifications of events happened in the external world and observed by sources. The CEP engine is responsible for filtering and combining such notifications to understand what is happening in terms of higher-level events (sometimes also called complex events or situations) to be notified to sinks which receive output events resulting from the queries running on CEP engines and act as event consumers [4].

2. ARCHITECTURE OF CEP BASED ON DDMS AND AT

2.1 AT & VWAP

There are various powerful algorithms being used by various organizations like Volume Weighted Average Price (VWAP), Time Weighted

Journal of Theoretical and Applied Information Technology
20th February 2013. Vol. 48 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Average Price (TWAP), Market On Close (MOC) and Information shortfall. Of all these VWAP has been the most popular model over the years [5]. The VWAP price as a quality of execution measurement was first developed by Berkowitz, Logue and Noser. They argue that 'a market impact measurement system requires a benchmark price that is an unbiased estimate of prices that could be achieved in any relevant trading period by any randomly selected trader' and then define VWAP as an appropriate benchmark that satisfies this criteria.

For instance the VWAP of a stock can simply be explained as the average price paid per share during a specified time, usually a day. This means that the price of each transaction in the market is weighted by its volume. In VWAP-trading the goal is to buy or sell a fixed number of shares at price that closely tracks the VWAP. VWAP is especially common in automatic trading algorithms, especially in optimal trading execution strategies [6]. The formula for calculating VWAP is as follows (1).

$$P_{VMAP} = \frac{\sum_j P_j * Q_j}{\sum_j Q_j} \quad (1)$$

where:

P_{VWAP} = Volume-Weighted Average Price

P_j = price of trade j

Q_j = quantity of trade j

j = each individual trade that takes place over the defined period of time, excluding cross trades and basket cross trades.

Here is an AT & VWAP example.

```
IF(
   MSFT's price moves outside 1% of MSFT-15-minute-VWAP
   FOLLOWED-BY{
      CSCO' price moves up or down by 0.5%
      AND
      IBM' price moves up by 3%
      OR
      MSFT' price moves down by 1%
   }
)ALL WITHIN any 120 seconds time period
THEN{
  BUY MSFT;
  SELL IBM;
}
```

2.2 Complex Events and Event Operators

An event is defined to be an instantaneous, atomic (happens completely or not at all) occurrence of interest at a point in time. It is the smallest, atomic occurrence in a system that may require a response. By atomic, we mean that either the event happens completely or it does not happen at all. A set of attributes can be associated with each primitive event. These attributes can carry information which can be used when a complex event occurs (at a later time) about the action that caused the event to occur.

Similar events can be grouped into an event type that gives the metadata for events that belong to the same class and includes the attributes of these events, and an event type is expressed by an event expression. An event instance is a single occurrence of an event of a particular type. This instance instantiates the attributes of the event type. We consider E_1, E_2, …, E_n as being primitive event types and e_1, e_2, …, e_n some of their respective instances.

Although an event is assumed to instantaneously occur at a time point, the event might be initiated at a prior time point, thus yielding a closed time interval between the start and end points. That is each event instance, whether primitive or complex, has both a start and end timestamp. Two special event types START and END are added internally by Synoptic to keep track of initial and terminal events in the traces [7]. A complex event is defined by applying an event operator to constituent events that are primitive or other complex events. In the absence of event operators, several rules are required to specify a complex event. Furthermore, some control information needs to be made a part of a rule specification .

An event E (either primitive or complex) is a function from the time domain onto the Boolean values, True and False. E : T → {True, False} given by (2).

$$E(t) = \begin{cases} T(rue) \cdots \text{if an event of type E occurs at time point t} \\ F(alse) \cdots \text{otherwise} \end{cases} \quad (2)$$

2.3 DDMS and AT

Trading algorithms often perform a considerable amount of data crunching that could in principle be implemented as SQL views. To understand the need to maintain and query a large data state, note that many stock exchanges provide a detailed view of the market microstructure through complete bid

Journal of Theoretical and Applied Information Technology
20th February 2013. Vol. 48 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and ask limit order books. The bid order book is a table of purchase offers with their prices and volumes, and correspondingly the ask order book indicates investors' selling orders. Exchanges execute trades by matching bids and ask by price and favoring earlier timestamps. Investors continually add, modify or withdraw limit orders, thus one may view order books as relational tables subject to high update volumes. The availability of order book data has provided substantial opportunities for automatic algorithmic trading.

To illustrate this, we describe the Static Order Book Imbalance (SOBI) trading strategy. SOBI computes a VWAP over those orders whose volume makes up a fixed upper k-fraction of the total stock volume in both bid and ask order books. SOBI then compares the two VWAPs and, based on this, predicts a future price drift. For simplicity, we present the VWAP for the bids only:

```
select avg(b2.price * b2.volume) as bid_vwap
from bids b2
where k * (select sum(volume) from bids)
> (select sum(volume) from bids b1
where b1.price > b2.price);
```

2.4 Architecture of CEP Based on DDMS and AT

Fig.1 shows the architecture of CEP Based on DDMS and AT. The core component of a DDMS is its runtime engine. Unlike a traditional database system where the same engine manages all database instances, each individual DDMS execution runtime is constructed around a specific set of queries provided by the client program (e.g., via SQL code embedded inline in the program), each defining an agile view.

The AT rule definitions is done by the Analyzer and the Constructor, well separated from the runtime tasks， represented by the Complex Event Detector, the Event Manager and the Executor. The following briefly describes these components.

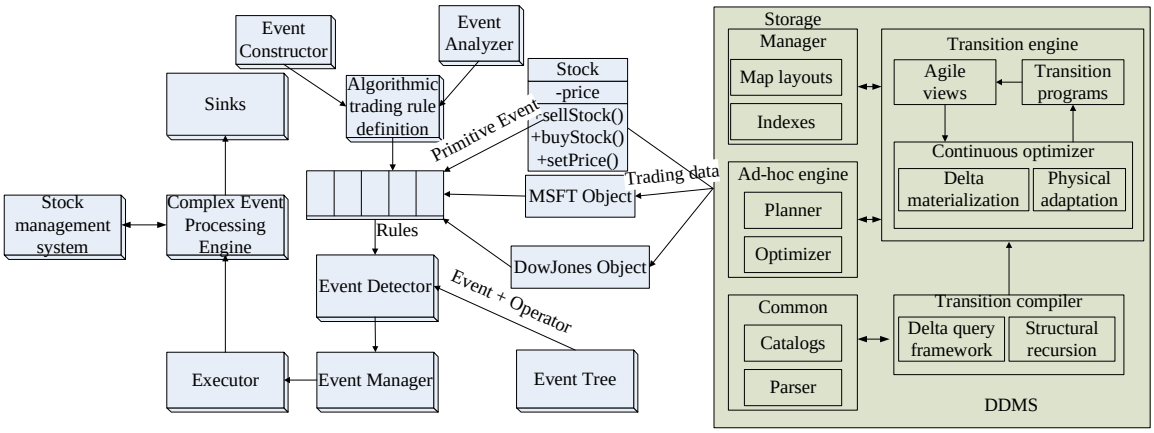

Fig.1. Architecture of CEP based on DDMS and AT

The Analyzer principally analyzes a rule definition and produces an intermediate representation of the rule which is sent to the constructor, and code corresponding to the condition and action of the rule.

The Constructor creates a persistent representation of rules and offers a low level interface well adapted for software integrators and developers who need basic reactive capabilities for supporting some functions of the system they want to implement.

The Event Executor is responsible for processing rules taking into account coupling modes, rule priorities. It realizes quite complex execution semantics and this combined with the need for runtime efficiency represents the main reasons for having implemented the part.

The Event Detector is responsible for detecting primitive events and for signaling them to the event manager. The latter recognizes complex events using a detection graph and signals both primitive and complex events to the Event Executor.

The event manager has to represent the information gained from the analysis of event definitions, i.e., it is responsible for managing the event base which consists of all defined events patterns. If the event type is primitive the Event Manger subscribes it to the CEP Engine. If the event type is complex, the Event Manager subscribes the primitive event types composing the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

event type to the event detector and builds an event tree representing the complex event type.

3. COMPLEX EVENT DETECTION

Events are detected on the server using an event graph. An event graph which consists of nodes and directed edges is a graph constructed to reflect the primitive and complex events declared in an application [8]. Each event is represented as an event node in the graph, and the event nodes are connected by their subscription relationships. An internal node of the event graph represents a complex event, and a leaf node represents a primitive event. Thus the event detector generates an event tree whose root node represents the complex event.

An event tree is created for each complex event and these trees are merged to form an event graph for detecting a set of complex events. This will avoid the detection of common sub-events multiple times thereby reducing storage requirements. Each node has a pointer to each of its subscribers. Thus each subscriber of a global event becomes one of its parent node that the event tree is built from. By default a subscriber is inserted in the end-list if it does not specify when to be notified. This organization reduces the search which is based on the class.

A event detector has a linked list whose nodes hold one reactive class of an application. Each node, in turn, has two linked lists, begin list and end list. The lists have the subscribers to be notified at the beginning or the end of these methods' invocations. For example:

event begin (e1) int sellStock(int number);

The primitive event e1 is bound to a method named sell stock and the method notifies its occurrence at the beginning of its invocation.

The event detector detects primitive events produced during an application processing. It detects only events for which event type subscription has been submitted and signals them and their environments to the event manager. The general principle for recognizing events is the following: primitive events are injected at the leaves of the event graph. Then these events flow upwards, following edges through internal nodes which represent component events [9]. When a triggering node is reached, the recognized triggering event is signaled and then taken into account for rule execution.

Whenever a primitive event is detected, it will propagate the event notification to its subscribers, that is, its parent nodes. Event occurrences flow upwards as in a data-flow computation. The parent nodes maintain the occurrence of its constituent events along with their parameter lists which are stored separately for each context set to the node. If the complex event occurs by the last notification, it is detected and further propagates to its subscribers. Each time an event is raised, it will check its "send back" flag. If the "send back" flag is true, the server will send this event notification to a specific application according to this event "site" attribute [10].Complex event detection algorithm is as follows:

```
function
complexEventDetection(eventStream,time){
  createEventTrees;
  foreach(event e of eventStream){
    if(e instance of Ei) then
    foreach(parentNode VE of VEi)
      call activeOperatorNode(VE);
    if(is_signaled(rootNode)) then
      detect(complexEvent);
}
function activeOperatorNode(VE){
  switch(VE){
    case "AND":
      if(is_signaled(childNode)) then
        if(!is_empty(the other's buffer queue))
then
          foreach(event ei of bufferQueue)
          create a pointer combines of e and ei;
          pass this pointer VE's buffer queue;
          clearBufferQueue();
        else
          append e to its own buffer queue;
    case "OR":
      if(is_signaled(childNode)) then
        pass pointer of e to the parent;
    case "sequence":
        if(is_signaled(leftChildNode)) then
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

append e to its own buffer queue;

if(is_signaled(rightChildNode)) then

if(!is_empty(left child's buffer queue)) then

foreach(event e_i of bufferQueue)

create a pointer combines of e and e_i;

pass this pointer V_E's buffer queue;

clear left child's buffer queue;

}

4. CONCLUSIONS

In this paper, we have investigated how events are defined, detected and managed and presented an expressive event specification language that supports DDMS and AT. We have illustrated the detection of complex events and proposed the architecture for its implementation based on DDMS and AT. Our approach clearly substantiates existing event-driven systems with declarative semantics. All the event detection algorithms we have developed extend readily when the identification of the object is allowed as an explicit parameter of a primitive event.

REFRENCES:

[1] Archit Bansal, Kaushik Mishra, and Anshul Pachouri, "Algorithmic Trading (AT)-Framework for Futuristic Intelligent Human Interaction with Small Investors," International Journal of Computer Applications, vol. 1, 2010, pp.1-5.

[2] Piyanath Mangkorntong, and Fethi A. Rabhi, "DETECTING EVENT PATTERNS IN E-MARKETS:A CASE STUDY IN FINANCIAL MARKET SURVEILLANCE," IADIS International Conference e-Commerce, 2009, pp.69-86.

[3] Alan Demers, Johannes Gehrke, and Biswanath P, "Cayuga: A general purpose event monitoring system," in Proc. of the Conf. on Innovative Data Systems Research(CIDR), 2007, pp. 412–422.

[4] GIANPAOLO CUGOLA, and ALESSANDRO MARGARA, "Processing Flows of Information: From Data Stream to Complex Event Processing," in Proceedings of the 5th ACM international conference on Distributed event-based system, vol. 5, 2011, pp.1-69.

[5] Nihal Dindar, Peter M. Fischer, Merve Soner, and Nesime Tatbul, "Efficiently Correlating Complex Events over Live and Archived Data Streams," in Proceedings of the 5th ACM international conference on Distributed event-based system, 2011.

[6] Erik Eiesland, "Simulating The Order Book:A Tool To Discover Trading Strategies," Master Thesis, Department of Computer Science, stfold University College, 2011, pp.1-124.

[7] Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar, and S.-K. Kim, "Composite Events for Active Databases: Semantics, Contexts and Detection," in Proceedings of the 20th International Conference on Very Large Data Bases, 1994, pp. 606-617,.

[8] Marcelo R. N. Mendes, Pedro Bizarro, and Paulo Marques, "A Framework for Performance Evaluation of Complex Event Processing Systems," in DEBS '08: Proceedings of the second international conference on Distributed event-based systems, 2008, pp. 313–316.

[9] R. BALDONI, S. BONOMI, G. LODI, M. PLATANIA, and L. QUERZONI, "Data Dissemination Supporting Complex Event Pattern Detection," in 1st International Workshop on Data Dissemination for Large scale Complex Critical Infrastructures, 2010, pp.1-25.

[10] Ivan Beschastnikh, Yuriy Brun, Sigurd Schneider, Michael Sloan, and Michael D. Ernst, "Leveraging Existing Instrumentation to Automatically Infer Invariant-Constrained Models," in Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, 2011.