A Real-Time Futures Information Processing Strategy Based on Multi-level Cache and Multi-threading Cooperation

Li-kai SU
Center of Faculty Development & Educational Technology, Guangdong University of Finances & Economics, Guangzhou, China

Keywords: Futures program trading, Real-time data processing, Multi-level cache, Multi-threaded collaboration.

Abstract. Futures market data processing is a basic problem in the research of futures program trading. Because of the large amount of futures market data, there are many problems in the real-time data processing, such as high delay, low storage efficiency, poor stability and high input cost. In order to solve the above problems encountered in the development of futures program trading software, a non-blocking real-time market data processing strategy based on multi-level cache and multi-threaded collaboration is proposed and implemented, with the characteristic of low delay, high storage efficiency and low investment cost in market data processing, exhibits a high applicable value for the design of program trading software for small and medium-sized investment companies.

Introduction
With the support of computer and network technology, based on the pre-set trading model, the program trading is a kind of trading method which takes advantage of the high-speed calculation of computer to track the price changes, generate the transaction instructions in real time and complete the transaction quickly. Because it uses the mathematical model established by computer to trade, which helps to strengthen the objectivity of trading, thus effectively avoiding the intuitive mistakes brought by traders' emotional fluctuations [1]. According to statistics, 70% of transactions in the U.S. market are completed by program trading, which may be higher if the concept of program trading is redefined more broadly [2].

The program trading in China's securities market started slowly, and now it is widely used in futures market. The general program trading products are customized based on the open interface provided by the comprehensive transaction platform.

The real-time market quotations are the basic input of the program trading calculation. According to the market changes, the transaction algorithm calculates, on the basis of a certain logical relationship, to get the transaction signal so as to trigger the specific transaction behavior automatically. Therefore, the real-time and stability of the market data are the key factors that affect the results of the program trading. It is of great significance to study and improve the processing methods of the market data.

At present, there are two mains methods to speed up the real-time market data processing. One is the software acceleration, which is generally based on the single level cache or single thread processing [3,4]. This method does not take into consideration the mismatch between the speed of remote data storage and the speed of the market data cache, which leads to poor stability. The other is the hardware acceleration, such as the single chip FPGA proposed by Zhang Yanbin [5] which accelerates the processing of futures market data. The problem of this method lies in increasing hardware investment and making the system more complex. Without increasing the complexity of the system nor the cost of input, this paper proposes an improved method for the software acceleration, which improves the efficiency and stability of data processing and achieves good effect in practical applications.
Analysis of Traditional Method of Software Acceleration

The traditional method of software acceleration adopts in general the single level cache or the single-thread processing method. The data received by the market quotations interface is directly stored in the cache area, and then written to the storage device in batch after the data in the cache area is full. But an issue remains with this approach: once the write speed of the storage device is far lower than the processing speed of the data in the cache area or the storage device fails, the thread receiving the quotations will be blocked or wrong. The following figure shows the process of traditional software acceleration method.

Figure 1. Traditional method of software acceleration.

Aiming at the problems existing in the traditional software acceleration method, based on the analysis of the application features of real-time market data, this paper classifies the market data into three levels, namely: (1) initial market data, which generally refers to the in-depth market data of Tick-level futures; (2) processing market data, which generally refers to the technical index data obtained from the initial market calculation; (3) storing market data, which is mainly composed of the data that needs to be stored in the historical database, is primarily used for back test research of the program trading algorithm, and generally includes a part of the first two types of data.

Based on this classification result, an improvement is proposed: firstly, three kinds of corresponding data buffer areas are set up in the system design to realize the classified storage of market data; secondly, with the original process of market data decoupled and the multi-threading process adopted, the risk of blocking is highly reduced while the data processing thread deals with the data, thus enhancing efficiency and stability of market data. The improved market processing design is presented as follow.

Figure 2. Design scheme of multiple-level cache and multi-threading collaboration.
Key Technologies Study

Status Feed-Back Form

The status feedback form mainly records the life cycle of all working threads (including the start and end time of each day), running status, processed data, data to be processed, average processing speed of data, real-time speed, etc., providing decision-making basis for monitoring threads in order to coordinate all kinds of threads to work orderly.

Monitoring Thread

Monitoring thread runs through the whole cycle of futures market processing, and its main function is to coordinate the working threads to work correctly according to the information of the status feedback table. As shown in Figure 3, the monitoring thread checks the real-time data of the status feedback table in accordance with a certain frequency and decides whether to start or end the thread according to the thread life cycle. It detects the real-time running status of all threads. If the status is Error, re-initializing and starting threads will be required.

Receiving Thread of Market Data

Through the open interface of CTP, the receiving thread of market data inquires all trading contracts of the day from the exchange, obtains all contract codes, subscribes to the CTP for the in-depth market data of all contracts, receives the market data and caches the data to the initial market cache queue. If the cache queue of initial market data is expressed as the dynamically changing market data queue MQ (T, n), t is the time variable at a certain time, n is the number of elements in the queue, and the receiving thread of market data is expressed as the function MT (t) varying with time, then the receiving thread of market data can be defined as: MT (t) = MQ (t, n).

Data Processing Thread

The data processing thread accesses the initial market data cache queue MQ (T, n) according to a certain frequency, obtains a subset of the queue MQ (a), where a is a constant, and then calculates the futures market technical index queue K (a) according to the time series, and adds K (a) to the processing data cache queue. If the processing data cache queue is expressed as PQ (T, n), t is the time variable at a certain time, n is the number of elements in the queue, and the data processing thread is
expressed as the function $P(T)$ varying with time, then the data processing thread can be defined as: $P(T) = PQ(T, n)$.

**Storage Conversion Thread**

The storage conversion thread accesses the processing data cache queue $PQ(T, n)$ according to a certain time frequency, obtains a subset $PQ(a)$ of the queue, where $a$ is a constant, and then converts $PQ(a)$ into a batch stored database instruction according to the characteristics of the storage device and database, and caches it into the storage instruction cache queue. If the storage instruction cache queue is represented as $CQ(T, n)$, $n$ as the number of elements in the queue, $t$ as the time variable at a certain time, and the storage conversion thread as the function $ST(T)$ changing with time, then the data processing thread can be defined as: $ST(T) = CQ(T, n)$.

**Storage Execution Thread**

The traditional method of market data processing based on software acceleration writes the cached market data directly into the database, especially in the way of serial writing, which leads to low storage efficiency. When the speed of data storage is lower than the speed of receiving and processing market data, it is easy to cause data overflow error in buffer area, and increases the storage instruction buffer area, so as to decouple the receiving and calculation process of market data and data storage process, and improve the stability of market processing. In addition, a single storage path may lead to the failure of the storage device, which may cause the processing thread of market data to be blocked. In this paper, multiple storage paths are used to ensure the stability of data storage.

The storage execution thread accesses the storage instruction cache queue $CQ(t, n)$ according to a certain time frequency, takes out a storage instruction and submits it to the remote database for processing; if the remote database responds and executes the command, the next instruction is executed circularly; if the remote database responds overtime or feeds back error information, the storage instruction is stored in the local file, after the remote database recovers, then take out and execute the storage instruction of the local file.

**Experiment Analysis**

**Operation Effect**

In order to verify the strategy of futures market storage proposed in this paper, we use C# language to encapsulate the CTP market interface, develop the futures market management system, and realize the storage of futures market data based on MySQL database. In order to verify the operation efficiency of the system, the following experimental environment is adopted, as shown in Table 2:

<table>
<thead>
<tr>
<th>Operation system</th>
<th>Windows Server 2012 R2 Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel(R) Xeon(R) CPU E5620 @ 2.40GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>32.0 GB</td>
</tr>
<tr>
<td>Data base</td>
<td>MySQL 5.7</td>
</tr>
</tbody>
</table>

The results of system operation show that: the peak value of futures market data reception is 1500 pieces/s, and the data storage speed of database is about 1000 pieces/s, which can meet the demand of market data reception, processing and storage, with low latency; the system has been running for more than three months without any failure, which shows its high stability.

**Storage Optimization**

The storage conversion thread accesses the processing data cache queue $PQ(T, n)$ according to a certain time frequency, and obtains a subset $PQ(a)$ of the queue, which is converted into a batch storage database instruction, where $a$ is a constant. How to determine the value of $a$ is related to the
final storage efficiency. In order to optimize the working efficiency of the storage conversion thread, different values are used for experiments, and the results are shown in Figure 4 and Figure 5 below.

![Figure 4](image1.png)

Figure 4. The variation curve between a and storage time.

![Figure 5](image2.png)

Figure 5. The variation curve between a and average time.

The experimenting results show that, when a = 200, the batch storage effect is optimized.

**Conclusion**

In this paper, the storage strategy of multi-level cache and multi-threaded cooperation is used to improve the traditional method of software acceleration of futures market data, which has proved its good applying effect with features such as low delay, good stability and low cost and has high reference value in the development and application of software for the program trading of futures.

**References**


