Proceedings of the IASTED International Conference
COMPUTATIONAL INTELLIGENCE
November 20–22, 2006, San Francisco, CA, USA
ISBN Hardcopy: 0-88986-602-3 / CD: 0-88986-603-1

A DYNAMIC STORAGE METHOD FOR STOCK TRANSACTION DATA

Jiarui Ni and Chengqi Zhang
Faculty of Information Technology
University of Technology, Sydney
GPO Box 123, Broadway, NSW 2007, Australia
email: {jiarui, chengqi}@it.uts.edu.au

ABSTRACT
Stock transaction data have become very detailed and enormous with the introduction of electronic trading systems. This makes it a problem to store and to access the data in later analyses such as mining useful patterns and backtesting trading strategies. This paper investigates several storage methods in terms of both storage space and access efficiency and then proposes a new dynamic storage method which provides a flexible mechanism to balance between storage space and access efficiency for storing huge intraday transaction data.

KEY WORDS
data preparation, storage, stock transaction, database, data compression

1 Introduction

With the help of modern computer systems, it is now possible to record every piece of transaction information during the operation of a stock market and save the information for later analyses. Professional and amateur traders rely on these information to understand the market and make decisions. Researchers can also use these data to study various financial or social theories such as market dynamics, herd behaviour, etc. Mining useful patterns and backtesting technical trading strategies based on historical stock transaction data have also been very popular in recent years [1, 2, 3].

Many data providers provide stock transaction data to their subscribers in various formats. As the data become more detailed, the space required to store them and the effort required to utilize the data also expand dramatically. For example, three years' intraday data (2001 – 2003, in CSV format) take up 3 gigabytes (GB) disk space for Microsoft (MSFT) and it takes about two minutes to simply read through the data. This makes it necessary to develop efficient storage methods in terms of both storage space and access efficiency.

There are various techniques to store time series data efficiently [4]. Some storage methods for time series data use lossy compression techniques, such as discrete cosine transform (DCT), fractal compression, wavelet compression, etc. However, for the analysis of stock transaction data, a small change in the data could cause severe distortion in the result. Therefore lossless compression techniques are the only choice. The Lempel-Ziv (LZ) compression methods are among the most popular algorithms for lossless storage. And the widely used `gzip`[5] utility uses a variation of LZ, DEFLATE, which is optimized for decompression speed and compression ratio, although compression can be slow [6].

This paper investigates several storage methods and then proposes a new dynamic storage method in detail. The rest of this paper is organized as follows. Section 2 explains the characters of stock transaction data and their normal access patterns. Section 3 shows several storage methods and discusses their pros and cons. Section 4 presents our dynamic storage method. Section 5 gives the experimental results. And finally, Section 6 concludes this paper.

2 Characters of Stock Transaction Data

Stock transaction data can be roughly categorized into two groups: daily data and intraday data. Daily data usually include volume, open price, close price, highest price and lowest price. Daily data may also include other aggregation data such as volume weighted average price (VWAP), etc. There may also be other information such as dividend. Daily data contain only aggregate information and there is only one record per day. A short sample is shown below, where the first column is the security code, the second column shows the date, and the final five columns are the open price, highest price, lowest price, close price and volume, respectively.

```
ASX,"01 May 2006",32.84,33,32.5,32.61,100807
ASX,"02 May 2006",32.49,32.49,31.8,31.81,383398
ASX,"03 May 2006",31.81,32.04,31.2,31.61,1046849
ASX,"04 May 2006",31.35,32.19,31.35,32,382486
ASX,"05 May 2006",32.14,32.17,31.8,32.15,242832
ASX,"08 May 2006",32.47,32.48,32.11,32.2,294295
ASX,"09 May 2006",32.15,32.29,31.9,32,244347
ASX,"10 May 2006",32.32,32.44,32,32.2,266441
ASX,"11 May 2006",32.47,32.47,32.06,32.15,246636
ASX,"12 May 2006",31.86,32.2,31.72,32.19,259179
```

Intraday data may include the following details: 1) the time, price and volume of every single trade; 2) the market depth at any time during the day; 3) the details of every order and its amendment, cancellation or execution, etc. There may also be other information such as the identity

of the initiator of every order, etc. Depending on the stock under consideration, there may be zero or tens of thousands of records per day. An example of intraday data is shown as follows, where two kinds of records are available. A QUOTE record shows the best bid price/volume, best ask price/volume at a given time, and a TRADE record shows the time, price and volume of a trade transaction, and the best bid price and ask price immediately after the trade.

```
QUOTE,20020102,14:18:51,,,,,11.38,3910,11.39,1000
TRADE,20020102,14:18:51,11.38,3090,11.38,11.39,,,,
QUOTE,20020102,14:25:09,,,,,11.37,200,11.38,1090
TRADE,20020102,14:25:09,11.38,3910,11.38,11.39,,,,
QUOTE,20020102,14:29:34,,,,,11.37,5200,11.38,1090
QUOTE,20020102,14:31:04,,,,,11.37,5200,11.38,90
TRADE,20020102,14:31:04,11.38,1000,11.37,11.38,,,,
QUOTE,20020102,14:31:50,,,,,11.37,200,11.38,90
TRADE,20020102,14:31:50,11.37,200,11.37,11.38,,,,
TRADE,20020102,14:31:50,11.37,4800,11.37,11.38,,,,
```

As a time series, stock transaction data are usually retrieved sequentially. For example, to display a historical chart, or to backtest a trading strategy, we normally have to read the data from the start to the end of a certain period. Only in the case of historical price lookup, we need to search the data according to a given condition such as the date. However, this does not happen often since a separated record here provides hardly any useful information.

Usually people focus on one stock at a time. However, sometimes it is also necessary to read the data for multiple stocks simultaneously. One example is the pairs mining which considers multiple stocks and trading strategies at the same time [7].

3 Existing Storage Methods

Various storage methods have been used to store stock transaction data. One simple and widely used method is a formatted plain text file, such as a comma separated values (CSV) file. Many stock data providers, such as Yahoo!Finance[1] and Commonwealth Securities Ltd.[2], provide daily data in this format. The sample data shown in Section 2 are displayed in this format, too.

Plain text files are easy to understand, to create and to read. It is normally good enough for daily data since daily data are not too big. However, it requires very large storage space for intraday data. As mentioned in Section 1, three years' intraday data (2001 – 2003, in CSV format) take up 3 GB disk space for MSFT. One solution to this problem is to compress the file. For example, with default settings, the `gzip` utility can deflate the MSFT data by near 90 percent to 330 megabytes (MB).

There are several intuitive variants to store intraday data in plain text files. One is to use one file per day for all stocks. Another one is to use one file per stock for all dates. A third one is to use one file per stock, per day. The first one is good for replaying the whole market in a time sequence, however, it is very inefficient to retrieve the data for a particular stock, which is often needed in many mining and backtesting processes. As a result, it will not be considered in this work. The second variant has the problem that the files would be pretty large when there is a long history of data. As a text file has to be read sequentially, it is very inefficient to read a chunk of records from the file since everything before that chunk has to be read and discarded. This problem might be alleviated by the introduction of some indices marking the start positions of each day in the file, however, when compression is used to reduce the file size, it becomes pretty hard to apply such a technique to the compressed file. The third variant is the finest one. It has little problem with the data access efficiency. However, for those stocks which are not heavily traded (unfortunately, a large portion of stocks fall into this category), there are not much transaction data for one single day, and this variant results in a lot of tiny files which occupy much more storage than what the data really deserve.

A second storage method is to use a relational database such as the commercial Oracle[8] database or the open source PostgreSQL[9] database, and take advantage of the power of the database management system (DBMS). In this way, the data can be retrieved more efficiently, especially for random access requirements. However, the storage problem is worsened because a DBMS usually creates a lot of metadata such as indices to facilitate its fast operations. And it is usually not possible to compress the data in a database.

There are other customized storage methods. For example, SMARTS[3] stores its data in FAV format which splits the data into multiple files where each file contains only one day's data of all stocks, and then compresses these files separately. It also uses a separate file to store daily data apart from the intraday data.

The FAV format bears the same disadvantages as the first variant of plain text file described above because it also mixes the data for all stocks into a single file. Another problem with the FAV format is that it uses its own encoding method and every transaction is encoded into a variable-length record. Proprietary utilities or APIs are necessary to create and read FAV files.

4 A Dynamic Storage Method

Based on the pros and cons of the storage methods discussed in Section 3, we have developed a dynamic storage method for large intraday data to make use of the advantages and avoid the disadvantages as much as possible. This section explains how it works and gives some theoretical analysis of its performance.

[1] finance.yahoo.com
[2] www.comsec.com.au
[3] www.smarts.com.au

4.1 The implementation

In our storage schema, we use compressed text files to store stock transaction data. There is one folder for each stock. In this folder, the data will be further split into several files. The size of each file (before compression) is controlled to be around S, which is to be tuned according to performance tests. Each file may contain data for a single day or multiple days, depending on the number or transaction records. An index file is used to map the date to data files. The algorithm to create such data storage for one stock is shown below.

1. let N be the total number of days;
2. let $S_i (i = 1, 2, \cdots, N)$ be the data size for each day;
3. let S be the desired size for each data file;
4. let C be the size of the current data file;
5. $C = 0$;
6. for $i = 1; i < N; i++$ do
   if $C \geq S$ or
   $C + S_i > S$ and $C + S_i - S > S - C$ do
   close current data file;
   update index file;
   add data for day i into a new data file;
   $C = S_i$;
   else do
   add data for day i into current data file;
   $C = C + S_i$;
   end if
   end for
   close current data file;
   update index file;

The data files are named according to a certain schema. We use `XYZ_YYYYMMDD.csv` where `XYZ` is the stock code or name, and `YYYYMMDD` represents the start date of the data contained in the data file. The index file stores the mapping from the date to data files, such as

```
20030108 => BHP_20030108.csv
20030115 => BHP_20030115.csv
20030121 => BHP_20030121.csv
20030129 => BHP_20030129.csv
```

Note that the dates here are not consecutive because one data file may hold data for several days.

With the help of the index file, it is possible to fast identify the data file containing the data for any specific date. And because the size of each data file is restricted to somewhere around S, the maximum data to be read and discarded in order to read the data for any specific date is also limited. Further, since plain text files are used here, we can use any standard compression utilities to compress the data file to save storage space.

When new data are available, it is also easy to merge new data with existing data under this storage method. Only the index file and the last data file (if necessary) have to be rebuilt.

4.2 Performance analysis

Storage space and access efficiency are the two performance factors of the most concern in this work. In our storage method, the storage space depends mainly on the compression ratio, which is influenced by the compression technique in use, by the file size, and particularly by the property of the stock data to be stored. It has been mentioned that the compression ratio for MSFT data is near 10. The result is similar for the data of other stocks from the same data source. However, for another set of stocks from a different data source, the highest compression ratio is only around 4, as will be shown in Section 5. The property of the data, however, is out of control, and can only be accepted as it is. What can be done is to select a proper compression technique and to adjust the file size.

As to the access efficiency, we have concern about the time needed to read the transaction data for any specific date from the data reservoir. For our storage method, suppose the average data size for one day is SD, the selected file size $S = k \times SD$, and the reading speed is v, then the average reading time for one day is

$$t = \frac{k+1}{2} \times \frac{SD}{v} \quad (1)$$

Therefore, a larger S means a longer reading time. For better access efficiency, a small S is preferred.

When S is small enough, i.e., no more than the average data size for one day, our method becomes the same as using one file per stock, per day. On the other hand, if S is arbitrarily large, our method turns to be using one file per stock.

5 Experimental Results and Discussions

In this section we show the experimental results of the performance of the dynamic storage method and compare it with the first two methods presented in Section 3, i.e., plain text files and databases. We do not try to compare with FAV format because its utilities and APIs are not available. All experiments described in this section are carried on in a Red Hat Linux box with an Intel(R) Xeon(TM) MP CPU (2.00GHz) and 4 GB memory.

5.1 Storage space

We use the standard `gzip` utility shipped with Red Hat Linux to compress the plain text files. As a first step, we study the relationship between the compression ratio and the size of the file. Figure 1 displays the average compression ratio with respect to file size. The sample files here contain three years (2003–2005) transaction data for stocks traded on Australian Stock Exchange (ASX). Each file corresponds to one stock. There are 2190 files in total.

The results show that when the size of the data files are larger than 1 MB, the compression ratio keeps stable

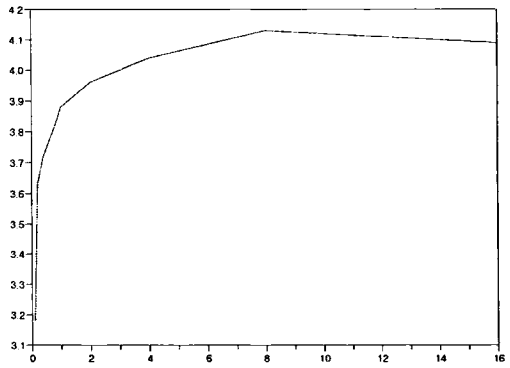

Figure 1. Compression ratio vs. File size (MB)

around 4. There is no big improvement for larger files, however, when the file size goes below 1 MB, the compression ratio drops significantly as the file size decreases. This means that the size of the data files S should be kept no less than 1 MB.

In our experiment, we test several different values for S. Table 1 lists all the storage methods we have tested. Table 2 lists the space used by these storage methods[4] for three stocks with decreasing total data size, namely, BHP, LHG and TSE. The average data size for one day is around 300 kilobytes (KB) for BHP, 60 KB for LHG, and 16 KB for TSE. And among the 2190 stocks, there are more than 2000 stocks that have less data than TSE.

Table 1. Storage methods

Method	Description
SM_1	One file per stock (plain text)
SM_2	One file per stock (compressed)
SM_3	One file per day (plain text)
SM_4	One file per day (compressed)
SM_5	PostgreSQL database
SM_6	Dynamic method (S = 1 MB)
SM_7	Dynamic method (S = 2 MB)
SM_8	Dynamic method (S = 4 MB)

The PostgreSQL database consumes astonishingly large disk space, which is highly discouraging.

A fixed S has the biggest impact on TSE, less impact on LHG, and the least impact on BHP. When S is set to

[4]The values for PostgreSQL database are not very accurate. We simply calculate the size difference after and before the data are inserted into the database. Due to the implementation of PostgreSQL database, this may not be the exact space used by the inserted data. Besides, for faster access, an index is created on the date field, which contributes about 18 per cent to the disk usage.

Table 2. Storage space for 3 stocks

Storage Method	Storage space (MB)		
	BHP	LHG	TSE
SM_1	224.5	42.75	10.49
SM_2	48.3	9.86	2.68
SM_3	228.9	45.70	12.03
SM_4	51.7	11.44	4.40
SM_5	585.6	109.17	27.36
SM_6	49.5	10.11	2.74
SM_7	48.9	9.98	2.70
SM_8	48.5	9.92	2.69

1 MB, our dynamic method can save 37 percent storage space comparing with the method SM_4 for TSE, 11 percent for LHG, and only 4 percent for BHP. For stocks like TSE, the method SM_4 consumes much more storage space than our dynamic method.

A larger S can further save the storage space but the improvement is not too much. And it can be seen that the result of SM_8 is already very close to SM_2.

5.2 Reading time

To test the reading time, we randomly select five days within the available time range and try to read the data for those days for a single stock from various data sources (based on different storage methods) and compare the reading time. The stock BHP is used in this experiment. Table 3 lists the results.

Table 3. Reading time for BHP

Storage method	Reading time (ms)				
	day 1	day 2	day 3	day 4	day 5
[SM_1]	340	1410	3000	4600	11800
[SM_2]	390	1510	3000	5800	13200
[SM_3]	5.8	5.8	5.8	5.8	6.0
[SM_4]	9.6	9.6	9.6	9.0	12.0
[SM_5]	136	136	138	132	160
[SM_6]	70	36	40	80	34
[SM_7]	70	90	58	140	126
[SM_8]	114	122	112	56	150

Again the results for the database approach (SM_5) are beyond the wildest expectation. It turns out that database is really a bad choice for such an application where sequential access of data dominates.

SM_3 achieves the shortest reading time in the experiment, which is quite reasonable since no extra work than necessary is done here. The single compressed file (SM_2) is the worst approach in terms of reading time. Our dynamic method makes a compromise between storage space

and reading time. It achieves much shorter reading time than SM_2, and much less storage space than SM_4 for stocks with little data per day.

It can also be seen that a larger S causes a longer reading time, which could hardly justify the little gain in storage space.

6 Conclusion

In this paper we investigate several storage methods for stock transaction data. After examining their pros and cons, we present a new dynamic storage method which tries combine the advantages of these existing storage methods. The new method provides a flexible mechanism to balance between storage space and access efficiency. With this method, it is easy to trade storage space for access efficiency and vice versa.

Acknowledgement

This work was supported in part by the Australian Research Council (ARC) Discovery Projects (DP0449535 and DP0667060), National Science Foundation of China (NSFC) (60496327) and Overseas Outstanding Talent Research Program of Chinese Academy of Sciences (06S3011S01).

References

[1] J. Ni, C. Zhang, An efficient implementation of the backtesting of trading strategies, *Proc. Third International Symposium on Parallel and Distributed Processing and Applications (ISPA'2005)*, Nanjing, China, 2005, 126–131.

[2] L. Lin, L. Cao, J. Wang and C. Zhang, The applications of genetic algorithms in stock market data mining optimization, *Proc. Fifth International Conference on Data Mining, Text Mining and their Business Applications*, Malaga, Spain, 2004, 273–280.

[3] L. Lin, L. Cao, C. Zhang, Genetic algorithms for robust optimization in financial applications, *Proc. The IASTED International Conference on Computational Intelligence (CI 2005)*, Calgary, Canada, 2005, 387–391.

[4] M. Last, A. Kandel and H. Bunke (Eds.), *Data mining in time series databases* (New Jersey: World Scientific, 2004). to

[5] http://www.gzip.org

[6] D. Salomon, *A guide to data compression methods* (New York: Springer, 2002).

[7] L. Cao, D. Luo, C. Zhang, Fuzzy genetic algorithms for pairs mining, *Proc. The Pacific Rim International Conferences on Artificial Intelligence 2006*, Guilin, China, 2006.

[8] http://www.oracle.com

[9] http://www.postgresql.org