Real-time Low Latency Data Processing using STM

Ryan Saptarshi Ray*, Prasun Bhattacharjee**, Utpal Kumar Ray*** and Parama Bhaumik****
*Senior Research Fellow, Jadavpur University, Kolkata, India
ryan.ray@rediffmail.com
** Cloud Architect, Tata Consultancy services
prasunb25@gmail.com
***Assistant Professor, Jadavpur University, Kolkata, India
utpal_ray@yahoo.com
****Assistant Professor, Jadavpur University, Kolkata, India
bhaumikparama@rediffmail.com

Abstract: Software transactional memory (STM) is an approach to solve the synchronization problems in parallel programs in a most elegant way. It allows end-users or developers to very easily handle threads and shared memory. In this paper we have compared performances of STM and lock implementations using C codes. Capital Market Surveillance is one of the prime sectors where real time processing is required. The data on which real time processing is required is enormous in size. We have chosen three use cases from real world stock market and implemented them using STM as well as lock methodologies using C codes to compare their performances.

Keywords: Parallel Programming, Multiprocessing, Locks, Transactions, Software Transactional Memory.

Introduction

The concept of Transactional Memory comes from database where end users and administrators are only bothered about data and queries. In database no one thinks about how concurrency, shared memory, parallel processing are being addressed. Concurrency is completely managed by database and end-users do not have any control over it. This same concept is being used to manage concurrency by STM.

There are many features in STM which could allow any organization to choose it as a concurrency management tool. But still STM is not a well known product in today's industry. One of the main reasons for this is that performance of STM is worse than that of locks. However nowadays in the latest versions of STM performance has improved a lot. In this paper we have done comparative study between STM and lock mechanisms using C codes [1],[2],[4],[7],[18].

Capital Market Surveillance

Overview

Thousands of users are doing stock transactions in each and every second. Each and every transaction has an impact on the stock market. There are a number of variants that determine prices of shares and the rate of changes of these variants are also very fast. As a result share prices may vary in every second and it is necessary to process transactions as fast as possible. So verification of transactions must be done very fast. Hence parallel processing is required as throughput should be huge.

State of The Art

A lot of propreitary and opensource products are available which can fit into these scenarios. A notable few are given below.

- Infosphere Streams is IBM's flagship product for stream processing. It offers a highly scalable event server, integration capabilities, and other typical features required for implementing stream processing use cases. The IDE is based on Eclipse and offers visual development and configuration. IBM Streams is an advanced analytic platform that allows user-developed applications to quickly ingest, analyze and correlate information as it arrives from thousands of data stream sources. The solution can handle very high data throughput rates, up to millions of events or messages per second. [25]
- Apache Storm is an open source framework that provides massively scalable event collection. Storm was created by Twitter and is composed of other open source components, especially ZooKeeper for cluster management, ZeroMQ for multicast messaging, and Kafka for queued messaging. Apache Storm is a free and open source

distributed realtime computation system. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language, and is a lot of fun to use! Storm has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees data will be processed, and is easy to set up and operate. Storm integrates with the queueing and database technologies already used. A Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed. [26]

- Apache Spark is a general framework for large-scale data processing that supports lots of different programming languages and concepts such as MapReduce, in-memory processing, stream processing, graph processing and machine learning. This can also be used on top of Hadoop. Yahoo uses Spark for personalizing news pages for web visitors and for running analytics for advertising. Conviva uses Spark Streaming to learn network conditions in real time. Apache Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing. Spark offers over 80 high-level operators that make it easy to build parallel apps. And one can use it interactively from the Scala, Python and R shells. Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. One can combine these libraries seamlessly in the same application. One can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, or on Apache Mesos.[27]

TIBCO StreamBase is a high-performance system for rapidly building applications that analyze and act on real- time streaming data. The goal of StreamBase is to offer a product that supports developers in rapidly building real-time systems and deploying them easily. Before TIBCO StreamBase ® CEP, processing real-time data feeds with high throughput was a difficult undertaking. With the TIBCO StreamBase enterprise-class CEP engine, more and more organizations are gaining faster processing and reaction to real-time complex event streams, shorter development cycles, with significantly easier maintenance, dramatically lower development and programming costs, flexibility to quickly adapt to changing business and analytic needs , reduced hardware and operational expenses, faster time-to-value from real-time initiatives. [28].

Proposed Work

STM is increasing in popularity day by day. It is overcoming its initial shortfalls and gearing up for market adaptability. But the main obstruction in the way is its performance.

Here in this paper we have done performance comparison between lock and STM mechanisms using C codes. We have considered three Capital market surveillance use cases here to be implemented by lock and STM mechanisms.

Use cases

Use Case-1: ***Trade Transaction after a gap of X days***

Alert will be generated if transaction of a share of a particular company happens after X days. This means, difference between last day of transaction and current transaction date should not be more than X days.

RULE:

READ[*Trade ID*][*Last Transaction Date*] *from historical data*

Generate Alert if

Difference between Current Transaction date and Last Transaction Date > X Days

Use Case-2: ***Price Variation***

Alert will be generated when Trade price surpasses last closing day's max price or becomes less than last closing day's min price.

RULE:

READ[*Trade ID*][*Last Day's Max Price*] *from summary data*

READ[*Trade ID*][*Last Day's Min Price*] *from summary data*

Generate Alert if

Trade Price > *READ*[*Trade ID*][*Last Day's Max Price*]

OR

Trade Price < *READ*[*Trade ID*][*Last Day's Min Price*].

Use Case-3: ***Anomaly in Running Average of Trade Value***
Alert will be generated when Running Average of trade values (trade price x transactioned trade quantity) crosses last closing day's trade values.
RULE:
READ[*Trade ID*][*Last day avg value*] from *historical data*
Generate Alert if
((*total value + (share price * share quantity)) / total share quantity) > READ[Trade ID][Last day avg value*]
*total value = total value(price*quantity)of a particular share since opening of trade market*
share price = Current transaction trade price
total share quantity = Summation of total number of shares being transactioned upto that point of time

Implementation

All the above mentioned scenarios are being implemented using locks as well as STM. All these use cases require two types of inputs. One of these inputs is data feeds of real time capital market transactions. The other input is stored historical data. Whenever and wherever a transaction happens data comes to the system through these data feeds. Then the system executes surveillance algorithms and if necessary compares recent transactions with historical data. Multi-threaded parallelism is used to implement surveillance algorithms.

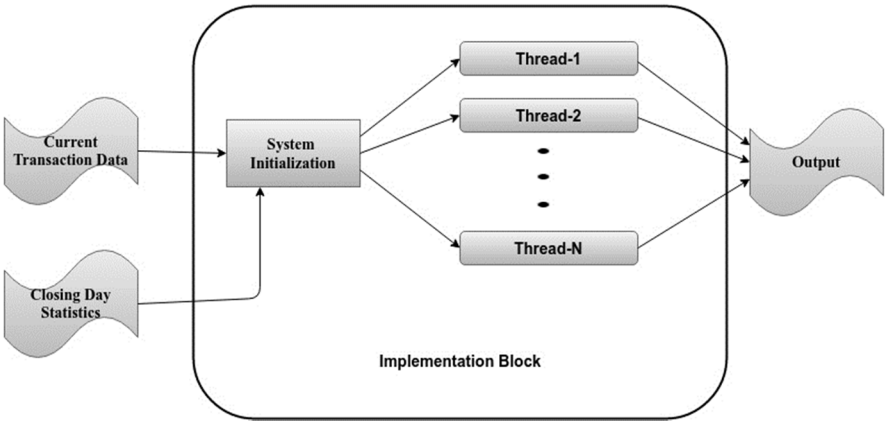

Figure 1. Implementation Diagram

Figure-1 shows how these use-cases are implemented. Main two inputs to the system are two files. As we do not have any real-time capital market data feed, we have used one day's transaction records in one file and its corresponding historical data in a database. At the process initialization phase data from these two files are loaded in the system buffer. Threads are created to share the processing load of the surveillance algorithms. Each and every thread is being assigned to one particular core. In this way the load is being distributed among all available cores. Each thread will get equal amount of load to process. If fraud is detected in any transaction then a detailed report is generated. We are saving this report in a file.

Using Locks

For first kind of implementation we have used locks. Lock helps to achieve synchronization between threads. Shared memory is required to maintain the state of each and every traded share. Lock helps to protect all the shared memory. Below we have given a code snippet to show how lock is used to implement these scenarios [2].

```
void takeRecord (char shareName [ ] , float sharePrice , float shareQty)
{
   int counter = 0 ;
   while (counter < runningRecords)
   {
      if (map [counter] . init == 0)
      {
         double temp_total_valtrd = shareQty * sharePrice ;
         double temp_total_run_avg = temp_total_valtrd/ shareQty ;
```

```
        pthread_ mutex _lock (&lock ) ;
        map [counter] . init = 1 ;
        strcpy (map [counter] . symb , shareName) ;
        map [counter] . rsto . total_qty = shareQty;
        map [counter] . rsto . total_count = shareQty ;
        map [counter] . rsto . total_valtrd = temp_total_valtrd ;
        map [counter] . rsto . total_run_avg = temp_total_run_avg ;
        pthread_mutex_unlock (&lock ) ;
        break ;
      }
      else  if (strcmp (map [counter] . symb ,    shareName )  == 0 )
      {
        double  temp_total_qty = map [counter] . rsto . total_qty + shareQty;
        double  temp_total_count = map [counter] . rsto . total_count + shareQty ;
        double  temp_total_valtrd = map [counter]  . rsto . total_ valtrd + (shareQty* sharePrice) ;
        double  temp_total_run_avg = map [counter] . rsto . total_valtrd / map [counter] . rsto . total_qty ;
        pthread_mutex_lock (&lock ) ;
        map [counter] . rsto . total_qty = temp_total_qty ;
        map [counter] . rsto . total_count = temp_total_count ;
        map [counter] . rsto . total_valtrd = temp_total_valtrd ;
        map [counter] . rsto . totalrunavg = temptotalrunavg ;
        pthread  mutex  unlock (&lock ) ;
        break ;
      }
      counter ++;
   }
}
```

Using STM

In the second part of our work we have implemented the scenarios using STM. The code snippets below show the implementations using STM.

```
while (counter < runningRecords)
{
   if ( map [counter] . c == 0)
   {
      double  temp_total_valtrd = shareQty * sharePrice ;
     double  temp_total_run_avg = temp_total_valtrd/ shareQty ;
      START(0 ,RW) ;
      STORE(&map [counter] . counter , 1 ) ;
      strcpy (map [counter] . symb , shareName ) ;
      STORE(&map [counter] . rsto . total_qty , shareQty ) ;
      STORE(&map [counter] . rsto . total_count, shareQty ) ;
      STORE(&map [counter] . rsto . total_valtrd , temp_total_valtrd ) ;
      STORE(&map [counter] . rsto . total_run_avg , temp_total_run_avg ) ;
      COMMIT;
        break ;
      }
     else  if (strcmp (map [counter] . symb , shareName )== 0 )
    {
      double  temp_total_qty ;
      double  temp_total_count ;
      double  temp_total_valtrd ;
      double  temp_total_run_avg ;
      START(0 ,RW) ;
      temp_total_qty = (double )
      LOAD(&map [counter ] . rsto . total_qty ) ;
      temp_total_qty += shareQt y ;
```

```
      STORE(&map [counter ] . rsto . total_qty , temp_total_qty ) ;
      temp_total_count = (double )LOAD(&map [counter] . rsto . total_count ) ;
      temp_total_count += shareQt y ;
      STORE(&map [counter] . rsto . total_count , temp_total_count ) ;
     temp_total_valtrd = (double)LOAD(&map [counter] . rsto . total_ valtrd ) ;
      temp_total_valtrd += ( shareQty * sharePrice ) ;
      STORE(&map [counter] . rsto . total_ valtrd , temptotal_valtrd ) ;
      temp_total_run_avg = (double)LOAD(&map [counter] . rsto . total_run_avg ) ;
      temp_total_run_avg = temp_total_valtrd /temp_total_qty ;
      STORE(&map [counter] . rsto . total_run_avg, temp_total_runavg) ;
      COMMIT;
      break ;
   }
   counter ++;
}
```

Experimental Results

The experimental results of Real Time Data Processing with locks and STM are given below.

Table 1. Results With Lock and STM Implementations

No. of threads/core used	Using Locks(sec)	Using STM(sec)
1	64	58
2	45	31
3	32	27
4	21	20

From the above experimental results we can see that STM has performed slightly better than locks. The corresponding graphs are shown below.

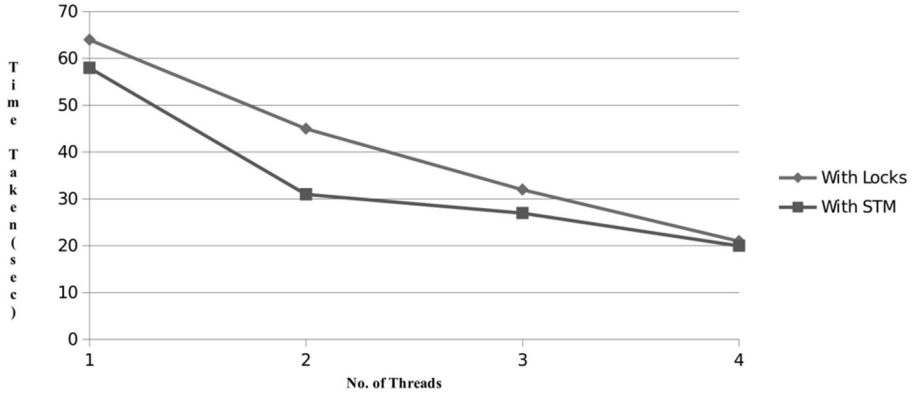

Figure 2. Time Taken vs No. of Threads

From the graph shown in Fig. 2 we can see that as the number of threads increases the execution time of the code decreases in case of both locks and STM. But the decrease is more in the case of STM. So we can say that STM performs slightly better than locks.

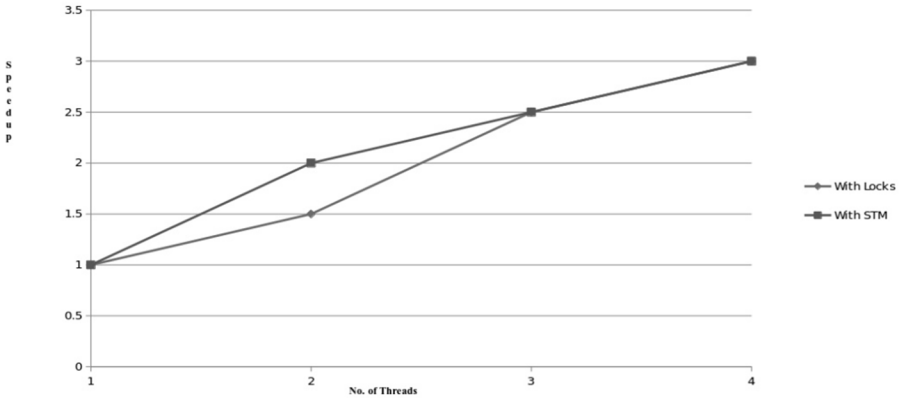

Figure 3. Speedup vs No. of Threads

From the graph shown in Fig. 3 we can see that as the number of threads increases the speedup also increases in case of both locks and STM. But at least initially the increase is more in the case of STM. So we can say that STM performs slightly better than locks.

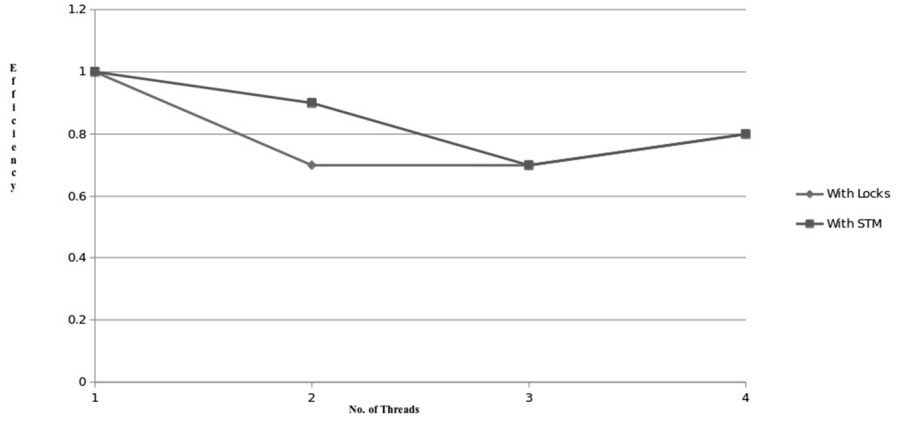

Figure 4. Efficiency vs No. of Threads

From the graph shown in Fig. 4 we can see that as the number of threads increases the efficiency varies around 1 in case of both locks and STM. But at least initially the efficiency is more in the case of STM. So we can say that STM performs slightly better than locks.

Conclusion

Locks solve the problem of synchronization in parallel programs, but suffer from many drawbacks. Till now very little work has been done to find out some alternative to locks. This paper has discussed an alternative to locks called STM and has also shown how STM can be used to solve the problem of synchronization in parallel programs. Capital market use case implementations of STM have been discussed. Performances of codes using locks and STM have been compared. In the future, further work can be done to improve the performance of STM so that STM gains wider popularity among the programming community.

References

[1] Simon Peyton Jones, “Beautiful concurrency”, included in “Beautiful code”, ed Greg Wilson, O’Reilly 2007.
[2] Elan Dubrofsky, “A Survey Paper on Transactional Memory”, www.researchgate.net.
[3] Pascal Felber, Christof Fetzer, Torvald Riegel, “Dynamic Performance Tuning of Word-Based Software Transactional Memory” published in Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming Pages 237-246 ,2008Information that you have referred, "full website link"

[4] http://en.wikipedia.org/wiki/Transactional_memory
[5] James Larus and Christos Kozyrakis. “Transactional Memory”, published in Communications of the ACM - Web science, Volume 51 Issue 7, July 2008, Pages 80-88
[6] Pascal Felber, Christof Fetzer, Patrick Marlier, Torvald Riegel, “Time-Based Software Transactional Memory” published in IEEE Transactions on Parallel and Distributed Systems Volume 21 Issue12, December 2010 Pages 1793-1807
[7] Tim Harris, James Larus, Ravi Rajwar, “Transactional Memory”, 2nd Edition, Morgan and Claypool Publishers, 2010
[8] Mathias Payer, Thomas R. Gross, “Performance Evaluation of Adaptivity in Software Transactional Memory” published in Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, pp 165-174, 2011
[9] Kevin E. Moore, Jayaram Bobba, Michelle J. Moravan, Mark D. Hill, David A. Wood., “LogTM: Log-based Transactional Memory” published in Proceedings of the 12th international conference on Architectural support for programming languages and operating systems, pp 359-370, 2012
[10] Dave Dice , Ori Shalev , Nir Shavit., “Transactional Locking II” published in Proceedings of the 20th international conference on Distributed Computing, pp 194-208, 2006
[11] http://tmware.org
[12] Maurice Herlihy, J. Eliot B. Moss, “Transactional Memory: Architectural Support for Lock-Free Data Structures” published in Proceedings of the 20th annual international symposium on computer architecture, pp 289-300, 1993
[13] Martin Schindewolf, Albert Cohen, Wolfgang Karl, Andrea Marongiu, Luca Benini, “Towards Transactional Memory Support for GCC” published in First International Workshop on GCC Research Opportunities. Held in conjunction with: the fourth International Conference on High-Performance Embedded Architectures and Compilers (HiPEAC), Paphos, Cyprus, 2009
[14] Virendra J. Marathe, Michael F. Spear, Christopher Heriot, Athul Acharya, David Eisenstat, William N. Scherer III, Michael L. Scott, “Lowering the Overhead of Nonblocking Software Transactional Memory” published in Proceedings of the First Workshop on Languages, Compilers, and Hardware Support for Transactional Computing (TRANSACT), June 2006
[15] Utku Aydonat, Tarek S. Abdelrahman, Edward S. Rogers Sr., “Serializability of Transactions in Software Transactional Memory” published in Workshop on Transactional Computing (TRANSACT), 2008
[16] Maurice Herlihy, Nir Shavit, “The Art of Multiprocessor Programming” published by Elsevier, 2008
[17] Brendan Linn, Chanseok Oh, “G22.2631 project report: software transactional memory”
[18] http://en.wikipedia.org/wiki/Software_transactional_memory
[19] http://research.microsoft.com/~simonpj/papers/stm/
[20] http://www.haskell.org/haskellwiki/Software_transactional_memory
[21] Ryan Saptarshi Ray, “Writing Lock-Free Code using Software Transactional Memory ”, M.E. Thesis submitted to Jadavpur University, 2012
[22] Ryan Saptarshi Ray,Utpal Kumar Ray “Different Approaches for improving performance of Software Transactional Memory ” published in International Journal of Engineering Research and Applications(IJERA) ISSN:2248-9622 Volume- 3, Issue-2 , March-April 2013, pp 1533-1540
[23] https://msdn.microsoft.com/en-us/magazine/cc163744.aspx
[24] http://www.infoq.com/articles/stream-processing-hadoop
[25] http://www-03.ibm.com/software/products/en/ibm-streams
[26] http://storm.apache.org/
[27] http://spark.apache.org/
[28] http://www.tibco.com/products/event-processing/complex-event-processing/streambase-complex-event-processing?benefits