Time Series Queries Processing with GPU Support*

Piotr Przymus[1] and Krzysztof Kaczmarski[2]

[1] Nicolaus Copernicus University, Poland
eror@umk.mat.pl
[2] Warsaw University of Technology, Poland
k.kaczmarski@mini.pw.edu.pl

Abstract. In recent years, an increased interest in processing and exploration of time-series has been observed. Due to the growing volumes of data, extensive studies have been conducted in order to find new and effective methods for storing and processing data. Research has been carried out in different directions, including hardware based solutions or NoSQL databases. We present a prototype query engine based on GPGPU and NoSQL database plus a new model of data storage using lightweight compression. Our solution improves the time series database performance in all aspects and after some modifications can be also extended to general-purpose databases in the future.

Keywords: time series database, lightweight compression, data-intensive computations, GPU, CUDA.

1 Introduction

Time Series analysis plays a crucial role in many important computational applications. Various hardware or software which must be monitored in order to ensure proper level of service quality emit time series as self describing data. This kind of *machine generated databases* are often growing with square factor since they represent monitoring relations between a distributed system's components in 'all-to-all' fashion. Number of time series generated in this way grows very quickly and falls under category of *Big Data*: problems in which size of data is a problem itself. Classical statistical systems (like R or SAS [7,2]), although capable of performing advanced analysis, are no longer able to handle the newly appearing challenges:

- Very large volumes of data must be consumed by a database in real time. If 1000 machine reports 1000 values every 10 seconds the systems must store $8.64 \cdot 10^9$ data points every day. Because of continuous operation of the system there is no possibility of batch processing.
- Resolution of data cannot be lost. Industrial systems benefit from ability to track single events as well as global tendencies or changes. Correlations between quickly appearing events cannot be found on general level.
- System must be able to answer any kind of queries to the database in reasonable time even if a query involves billions of points. This tight efficiency constrain may be only fulfilled if computation power is not bounded and scales well, possibly linearly.

* The project was funded by National Science Centre, decision DEC-2012/07/D/ST6/02483.

B. Catania et al. (eds.), *New Trends in Databases and Information Systems*,
Advances in Intelligent Systems and Computing 241,
DOI: 10.1007/978-3-319-01863-8_6,

– Storage may not be limited and should scale transparently. SQL databases with centralized indexes are no longer sufficient for these requirements or cannot meet limited budget requirements.

New systems like OpenTSDB [4] or Tempo-DB [6] try to address the above needs by using *Big Table* [8] data model. They are able to import data very efficiently while distributing it in a cloud-like storage. Querying is done by retrieving fragments of data from the distributed regions and putting them together with a map-reduce algorithm. One of the bottlenecks for a time series database is IO bandwidth and centralized aggregation process. Query processing for a longer period of time may need to process hundreds millions of data points. In such cases system reaction time often becomes too long.

1.1 General-Purpose Computation on Graphics Processing Units (GPGPU)

GPU programming offers tremendous processing power and excellent scalability with increasing number of parallel threads. However, vector-like processing in GPU has some limitations. One of them is obligatory data transfer between RAM (random-access memory) of the host machine and the computing GPU device, which generates additional cost when compared to a pure CPU-based solution. This barrier can make GPU-based algorithms unsatisfactory especially for smaller problems. One of the goals of this work is to improve efficiency of data transfer between disk, through RAM, global GPU memory and processing unit.

One of the possibilities, and often the only option, to optimize the mentioned data transfer is to reduce its size by compressing. Classical compression algorithms are computationally expensive (gain from the transfer data does not compensate the calculations [18]) and difficult to implement on the GPU [16]. Alternatives such as lightweight compression algorithms which are successfully used for CPU and GPU are therefore very attractive [18,10,12].

This paper addresses optimizations in time series systems like OpenTSDB allowing for faster query response. We present a new model of data storage using lightweight compression and parallel query processing using GPU. The rest of the paper is organized as follows. In the rest of this section we motivate and presents some of the related works concerning time series, big data and GPU processing. In section 2 we explain the prototype system architecture and querying process, while in section 3 a reader may find experimental results. Section 4 concludes.

1.2 Motivation

There are evidences of configurations pushing tens of billions data points a day into a monitoring system (like Facebook or Twitter). In such complicated cases system often stores very detailed measurements taken in different metrics and configurations for example every 10 seconds and therefore must deal not only with many points in the same time series but also with a huge number of time series as well.

What we observed in our industrial experience is that a user often performs many different queries working on the same time series in the fixed period of time. The reason for this is that users want to observe the same point in time from many angles, which

means performing different types of aggregations of different dimensions. Obviously, this analysis strategy cannot be predicted and aggregations cannot be preprocessed. OpenTSDB saves data points in cache in order not to repeat very expensive hbase scan operation. However, serialized points aggregation was noticed to be slower for large number of time series. Therefore, we propose to use GPU as an alternative query co-processor using our novel lightweight time series compression strategy. What is more important many users already own powerful graphical devices which may be used as local coprocessors for data analysis. Database querying should take into account this new possibility of supporting time consuming computations.

The main motivation of this work is to open new possibilities in time series querying by utilization of GPU processors for ultra fast time series aggregation on both server and client side. In this paper we also show that GPU processor may be used to perform computations on compressed data without introducing any additional costs. This in turn allows for application of GPU processors not only in computation-intensive problems in which time of copying data is amortized by numerical computations but also in data-intensive problems. This achievement opens a new filed for general database algorithms. Our solution improves the overall time series database performance by: minimizing communication footprint between a data storage and a query engine (by using i.a.: data compaction and lightweight compression) and moving data decompression and time series query processing to GPU.

1.3 Related Works

There is huge interest in efficient time series processing both in industry and science since large (and growing fast) data sets need to be queried and stored efficiently. OpenTSDB [4] build on top of HBase [1] and offers tremendous speed of data insertion and scanning together with high scalability. However, its data model is limited and so far cannot handle many important cases (like data annotations) Unde et al. [15] claim that OpenTSDB reaches much better performance than DB2 RDBMS for time series processing. Our experiments showed that OpenTSDB performance degrades if there are more than just a few tags attached to single metric which means that it has to aggregate too many time series.

Real time data analytic is offered by ParStream [5] data base system using GPU nodes. Data is equally distributed along machines. Combination of CPU and GPU processing together with load balancing enables to achieve almost real time processing of terabytes of data [11]. Also Jedox [3], an OLAP database system, offers possibility of using GPU as a coprocessor in queries. Both solutions are not strictly focused on time series processing and therefore probably cannot offer many optimizations which could be potentially possible. Our research is aimed at similar goals but with stress on large number of time series.

In [17] authors present interesting study of different compression techniques for WWW data in order to achieve querying speed-up. A general solution for data intensive applications by cache compression is discussed in [18]. Obviously the same technique may be used for time series and the efficiency may be increased if decompression speed is higher than I/O operation. In this paper we also show that decoding is really much faster and eliminates this memory bottleneck. The compression schemes proposed by

Zukowski et al. [18] offer good trade-off between compression time and encoded data size and what is more important are designed especially for super scalar processors which means also very good properties for GPU.

2 System Architecture

This section presents our system architecture. A set of collectors performs some treatment of source data and send it to the storage. Then data is sent to a storage which can be easily realized by column based NoSql database systems like Hbase [1] or Cassandra [9]. During the insertion process time series are compacted into larger records (taking into account the metric name and tags) containing a specified period of time (eg 15 minutes, 1 hour, 2 hours, 24 hours – depending on the number of observations) which differs from OpenTSDB design. The last important part of the system is the query engine responsible for user-database interactions. Again it must retrieve and process data in time acceptable for a user even if queried time period is long and covers many time series and data points. Following other solutions, as an answer to this problem we propose a analytic coprocessor but with GPU computing support. Since data transfer time is critical for distributed systems the key improvement over any other time series solution is decreasing size of necessary data transfer. In our solution we combine storing data internally compressed with adapted lightweight compression and ultra fast decompression done directly into GPUs memory. This strategy minimises not only storage size but also significantly increases transfer speed and processing time. In the last stage, utilization of GPU allows for very fast query processing.

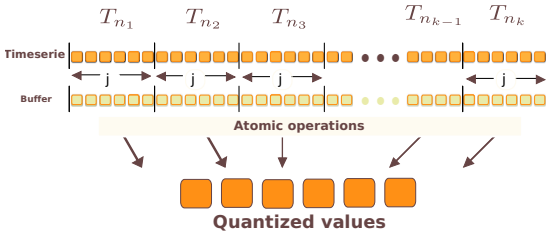
2.1 Query Processing

The overall process of query execution is shown in Fig. 1a, while detailed query processing steps are presented below.

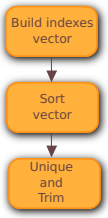
Decompression: OpenTSDB uses HBase support for lightweight compression algorithms such as Snappy or LZO. However, our observations suggest that the use of specialized lightweight compression algorithms like PFOR and PFOR-DIFF can significantly raise performance. Moreover, lazy decompression can be considered as a one of the stages of query processing, which minimizes the cost of memory transfers. Results are further improved by ultra fast decompression done by GPU processor. Obviously, better compression coefficients can be obtained due to the well-known characterization of the stored data. In this work we use modified PFOR and PFOR-DIFF from our earlier work [12].

Quantization: An important aspect is the analysis of the data at different levels of detail. This means that we have the opportunity to analyse the long-term general aspects as well as short-term detailed ones. Moreover, it allows us to limit the number of details in data, and thus reduce the initial size of it prior to processing. This important part of query processing may be efficiently performed on GPU: each thread examines j data elements (Fig. 1b). In a loop, it makes the quantization of the time series. Quantization is carried out using threads buffers to reduce the number of atomic operations needed for global memory. In the end, the partial results are stored in memory using global atomic operations.

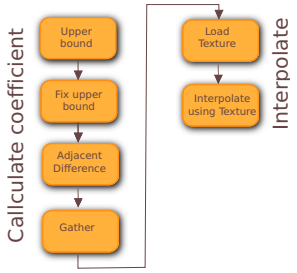
(a) **Query processing steps.** Quantization and Aggregation parts are optional



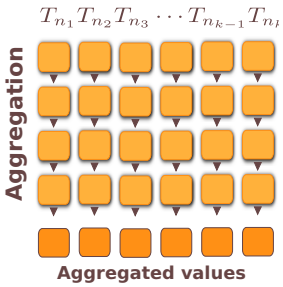
(b) Quantization



(c) Union



(d) Interpolation



(e) Aggregation

Fig. 1. Query operations (where $T_{n_1}, T_{n_2}, \ldots, T_{n_{k-1}} T_{n_k}$ are threads)

Union: In order to calculate aggregation for non evenly sampled time series, we need to transform them into evenly sampled ones (through interpolation). The first step is to determine a set union of timestamp indices for all time series. Again this stage can be efficiently implemented using *Thrust* GPU library offering basic operations for vectors (the interface is similar to the Standard Template Library vector for C++). In the first step, we build the vector of time series. Then the timestamps are sorted using sort method – which performs highly optimized Radix Sort. Subsequently unique operation is performed which removes duplicate items. See outline in Fig. 1c.

Interpolation: In the previous step we calculated the union of timestamp indices (t_i). Here, we need to interpolate values for selected timestamps in every time series. Finally, we obtain an evenly sampled time series. To improve efficiency of this part we used textures with CUDA hardware support for linear interpolation. There are also efficient implementations of other types of interpolation [14]. The procedure consists of two parts (see Fig. 1d). First we calculate linear interpolation coefficients, i.e. for each t in the union, we search for $t_i < t < t_{i+1}$ and calculate $t_{i+1} - t_0$. Since the time series are non-uniformly sampled this operation uses vectorized search (upper_bound from *Thrust*). The second step uses a linear interpolation GPU hardware support and uses previously computed factors.

Aggregation: Aggregation works on equally sampled time series. For each time point we calculate aggregation across data values in all time series. Each thread in a loop analyses the values of all time series for a single point in time. Then it writes aggregated value to global memory. See Fig. 1e for overview.

3 Prototype Query processing

Query Processing: The experiments were carried out using a common query for monitoring systems: *Calculate an aggregation of all the time series for a given metric for a specified period of time*. It is a general task which is a starting point for many other analytical methods like finding extreme values, pattern matching or other data mining algorithms. It covers all important aspects of query processing: data retrieval, combination of many time series, missing data and data aggregation.

Data Settings: A synthetic set of time series for a single metric with different tags was prepared. It may be treated as one parameter measurement on a set of distributed sensors. The simulated measurements correspond to 600 time series with measurement every 300 seconds with random 10% of elements missing in each time series, which gives approximately $600 \times 16.1K \approx 9.6M$ data points. Additionally, the synthetic data has been prepared to obtain different compression ratios seen in real applications [13].

Environment Settings: Experiments were carried out on one instance of HBase, query processing was conducted on the database server. Hardware configuration: Two six core processors *Intel® Xeon® E5649 2.53GHz*, 8GB RAM and *Nvidia® Tesla M2070* card. Tests were carried out using 600 time-series containing from $2.0K$ to $16.1K$ of observations, average processing time for 25 launches was taken. Because processed data in most cases fit in the HBase cache, configuration with LZO (Lempel-Ziv-Oberhumer) compression achieves only slightly better results than with no compression. In industrial applications, queries are less likely to hit the cache and the acceleration of LZO compression is higher.

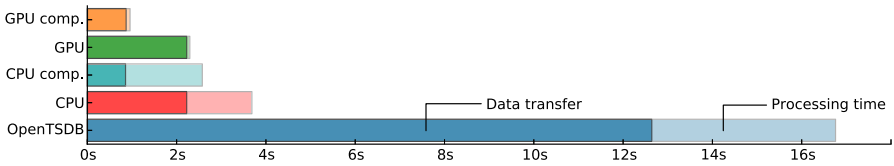
OpenTSDB: A modified version of a console client (modified to log query execution time after doing a warm-up phase of Java virtual machine) and Hbase configured with LZO compression were used in experiments.

Prototype: Developed using C++ and CUDA C++ and Thrift protocol. HBase was configured without compression, instead highly tuned lightweight (de)compression algorithms for time series where used.

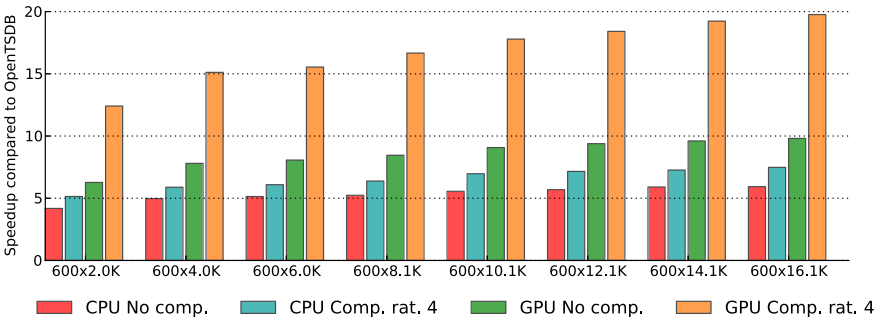
3.1 Results and Discussion

The comparison of OpenTSDB and our prototype performance is eligible due to similar architecture of both solutions and identical query processing work-flow. All differences are discussed bellow. The following factors were considered: the data transfer time (the time between sending a query to HBase and receiving the results) as well as the time needed to process the data (including data decompression), the time required to exchange data with the GPU (if used) and the processing time.

A detailed timeline for query execution with fixed data size (600 time series $\times 16.1K$ observations) is provided in Figure 2a. We can observe that processing in OpenTSDB



(a) **Execution time** 600 time-series, 16.1K observation each (lower is better)



(b) **Prototype speedup compared to OpenTSDB** (higher is better)

Fig. 2. Measured results

takes only 25% of query time and despite being 2.8 times slower than CPU prototype, it is not the main bottleneck. Better performance is not just a matter of changing Java to C++. It is *data compaction* to reduce processed data size and number of fetched rows and columns and increase efficiency of data transfer. Using data compaction, data transfer performance significantly increases (5.7 times faster) for both prototypes (CPU an GPU). But still most of the time is spent on communication with the database. What is more GPU is 21x faster than CPU when comparing data processing speed. Thus calculations are limited by the data transfer. It is therefore necessary to improve data transfer in order to achieve better results. This is done by using efficient lightweight compression implementation [12]. Lightweight compression introduces only a slight improvement in CPU prototype. This is because of the relatively long time needed for the data processing (almost 1/4 of total time – see *CPU* in Fig. 2a). This is because the lightweight compression significantly reduces data transfer time, but it also increases the data processing time (see *CPU comp.* in Fig. 2a). Computation and communication with the GPU is only a small fraction of the entire query and decompression adds only a small overhead to the query (see GPU and GPU comp. in Fig. 2a).

Figure 2b presents the resulting acceleration obtained on CPU and GPU prototype (in comparison to OpenTSDB query) on different data sizes. Both figures include CPU and GPU prototypes with and without lightweight compression. Due to the page limit only results for one compression ratio (4) are presented. Notice that the size of the data is important and better results can be obtained on larger data sets. It is also worth of noting that the data transfer is often a bottleneck in many GPGPU applications. This was also the case, however, through the use of a lightweight compression, data transfer is highly improved, thereby significantly speeding up the execution of the query.

4 Conclusions and Future Work

Time series databases play a crucial role in many branches of industry. Machine generated measurements require fast, real-time insertion and almost real-time querying. We showed that in case of computations dedicated to time series the existing solutions may be improved by utilization of GPU processors. So far data intensive application had to overcome the problem of additional CPU to GPU data transfer cost. Only algorithms of more than linear computation time complexity could benefit from parallel GPU processing. In this paper we showed that by introduction of fine tuned compression methods we can improve these results. Especially time series processing may speed-up significantly when compared to industrial solutions or experimental CPU prototypes.

Our future work will concentrate on query optimization in hybrid CPU/GPU environment, query execution on partially compressed data and on developing dynamic compression planer.

References

1. Apache HBase (2013), http://hbase.apache.org
2. Business Intelligence and Analytics Software - SAS (2013), http://www.sas.com/
3. Jedox - website (2013), https://www.jedox.com
4. OpenTSDB - A Distributed, Scalable Monitoring System (2013), http://opentsdb.net/
5. ParStream - website (2013), https://www.parstream.com
6. TempoDB – Hosted time series database service (2013), https://tempo-db.com/
7. The R Project for Statistical Computing (2013), http://www.r-project.org/
8. Chang, F., et al.: Bigtable: A Distributed Storage System for Structured Data. In: OSDI 2006: Seventh Symposium on Operating System Design and Implementation, pp. 205–218 (2006)
9. Cloudkick. 4 months with cassandra, a love story (March 2010), https://www.cloudkick.com/blog/2010/mar/02/4_months_with_cassandra/
10. Fang, W., He, B., Luo, Q.: Database compression on graphics processors. Proceedings of the VLDB Endowment 3(1-2), 670–680 (2010)
11. ParStream. ParStream - Turning Data Into Knowledge - White Paper. Technical report (2010)
12. Przymus, P., Kaczmarski, K.: Improving efficiency of data intensive applications on GPU using lightweight compression. In: Herrero, P., Panetto, H., Meersman, R., Dillon, T. (eds.) OTM-WS 2012. LNCS, vol. 7567, pp. 3–12. Springer, Heidelberg (2012)
13. Przymus, P., Rykaczewski, K., Wiśniewski, R.: Application of wavelets and kernel methods to detection and extraction of behaviours of freshwater mussels. In: Kim, T.-h., Adeli, H., Slezak, D., Sandnes, F.E., Song, X., Chung, K.-i., Arnett, K.P. (eds.) FGIT 2011. LNCS, vol. 7105, pp. 43–54. Springer, Heidelberg (2011)
14. Ruijters, D., ter Haar Romeny, B.M., Suetens, P.: Efficient gpu-based texture interpolation using uniform b-splines. Journal of Graphics, GPU, and Game Tools 13(4), 61–69 (2008)
15. Unde, P., et al.: Architecting the database access for a it infrastructure and data center monitoring tool. In: ICDE Workshops, pp. 351–354. IEEE Computer Society (2012)
16. Wu, L., Storus, M., Cross, D.: Cs315a: Final project cuda wuda shuda: Cuda compression project. Technical report, Stanford University (March 2009)
17. Yan, H., Ding, S., Suel, T.: Inverted index compression and query processing with optimized document ordering. In: Proc. of the 18th Intern. Conf. on World Wide Web, pp. 401–410. ACM (2009)
18. Zukowski, M., Heman, S., Nes, N., Boncz, P.: Super-scalar ram-cpu cache compression. In: ICDE 2006, Proc. of the 22nd intern. conf. on Data Engineering, pp. 59–59. IEEE (2006)