International Journal of Scientific and Technical Research in Engineering (IJSTRE)
www.ijstre.com Volume 5 Issue 5 ǁ Nov-Dec 2020.
ISSN: 2581-9941

Coordinated batch processing for multi-stage MapReduce of huge session data using GPGPU

Ruo Ando[1]
[1](*Center for Cybersecurity Research and Development, National Institute of Informatics, Japan)*

ABSTRACT: *The Science Information Network (SINET) is a Japanese academic backbone network for more than 800 universi-ties and research institutions. We propose a method of coordinated patch processing for coping with large scale session data using MapReduce framework. Besides, we introduce a multi-stage Reduce processing using GPU clusters to accelerate the processing of our pipeline. In proposal system, design patterns of pairwise reduction for GPU and merge scatter reduction for CPU are applied for processing huge session data effectively. Then, Splunk time-series indexer provides the operational insight for handling security incident response of security operation team in SINET. In experiment, we have measured CPU utilization in processing large scale session log file. It has been turned out that proposal system is robust to the size of session data ranging from 97GB to 400 GB in the point of CPU idle time. We can conclude proposal system can provide stable resource utilization regardless of the size of huge session data.*

KEYWORDS – *Batch Processing, MapReduce, GPGPU, CUDA Thrust, Intel TBB*

I. INTRODUCTION

1.1 SINET

The Science Information Network (SINET) is a Japanese academic backbone network for more than 800 universities and research institutions. It connects many research facilities in such fields as seismology, space science, high-energy physics, nuclear fusion, computing science, and so on. It is now being used by over 2 million users and supports international research collaboration through international lines. On March 2019, National Institute of Informatics (NII) builds the world's first round-the-globe ultra-high-speed 100 Gbps academic communications network. Since 2016, NII have been running a service of NII-SOCS (NII Security Operation Collaboration Ser-vices). Our team of NII-SOCS have deployed security monitoring system consists of PA-7080, Elasticsearch, Splunk and NVidia Multi-GPU server. In this talk, we introduce our system and some operational experience of handling huge session data ranging from 400,000,000 to 800,000,000 per day. During four years of 2016-2019, We have faced many challenges in terms of number of hosts, protocol profiliferation, probe placement technologies and security incident response.

In this paper, we propose a coordinated batch processing for multi-stage MapReduce to cope with huge session data on SINET. MapReduce [3] is an algorithmic framework, like divide and conquer or backtracking. MapReduce has been getting a lot of promise as an algorithmic framework which can be executed concurrently. In this paper, we illustrate the design for large scale MapReduce with different levels of parallelism.

1.2 PaloAto 7080 and session data format

The PA-7000 Series is powered by a scalable architecture for the purpose of applying the appropriate type and volume of processing power to the key functional tasks of networking, security, and management. Session data format is shown in Table 1. No.1 - 9 is concerned about TCP/IP packet header. NO 19-23 is retrieved to generate statistics. Particularly, No.12 (application) and No.17 (category) is inspected in detailed. Firewall such as PaloAlto-7080 plays an essential role in network security. Also, as cyber attacks become sophisticated, the language to achieve the efficiency and flexibility is required for complex intrusion detection tasks.

Table 1 Palo Alto session data format

No	item name	Value
1	capture time	2018/01/01 00:00:00.000
2	generated time	2018/01/01 00:00:00.000
3	start time	2018/01/01 00:00:00.000
4	elapsed time	3
5	source IP	xxx.xxx.xxx.xxx
6	source Port	64354
7	source country code	JP
8	destination IP	yyyy.yyyy.yyy.yyyy
9	destination Port	2939
10	destination country code	US
11	protocol	Tcp
12	application	NA
13	subtype	NA
14	action	NA
15	session end reason	NA
16	repeat count	0
17	category	NA
18	packets	0
19	packets sent	0
20	packets received	0
21	bytes	0
22	bytes sent	0
23	bytes received	0
24	device name	NA

II. OVERVIEW

Batch processing usually divided into two phases: (1) duplicating and producing multiple different outputs and (2) fetching multiple outputs back together. Second phase is designed for generating some sort of aggregate output. These two phases are called as coordinated batch processing as shown in Figure 1. One of the main purposes to adopt coordinated batch processing is implementing MapReduce pattern. Broadly, map phase is for sharding a work queue. Then, reduce phase is for interoperating processing which eventually reduces many outputs down to a single aggregate response. In reduce phase, there could be a large number of different aggregate patterns for normal batch processing. Therefore, we need to extend normal batch processing to alternative version such as coordinated batch processing.

2.1 Coordinated Batch Processing

2.1.1 Map

The basic concept of map is (1) taking a collection of data and (2) associating a value with each item in the collection. Consequently, a collection of key-value pairs are generated by matching up the elements of each input with some related value. Also, the number of collection by mapping operation should be equal to the number of input data items in the collection. In the view of concurrency, it is important that pairing up keys and values are independent for each input in the collection.

2.1.2 Reduce

The basic concept of reduce is reducing or merging several different outputs from map phase into a single output. Reduce operation yields the representative data required for producing the answer to the batch computation under processing by reducing the data from data item. Similar to map phase, a range of input should be equal to the one of output. Besides, the reduce phase can be repeated as many or as few times required in order to yield the output down to a single value over the entire data set. In the view of concurrency, coordinated batch processing is a typical example because it can frequently happen regardless of the number of inputs split up. In this sense, it is similar to join which is grouping together the parallel output of different batch operations.

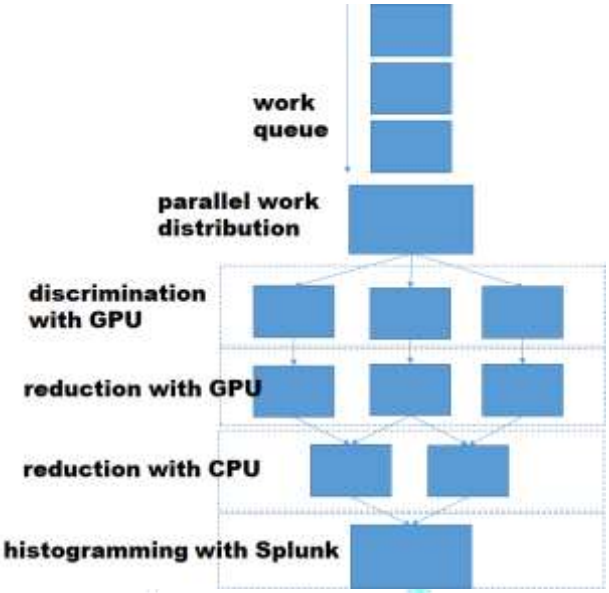

Fig. 1. Coordinated Batch Processing

2.2 Work Queue

We adopt work queue which is the basic form of our pipeline. In Figure 2, work queue manage runs in batch processing. The right side on Figure 2 shows discriminator threads by which each piece of work is independent of each other and can be processed without interactions. The main purpose of design of the work system is to ensure that each chunk of work is processed within a certain amount of time.

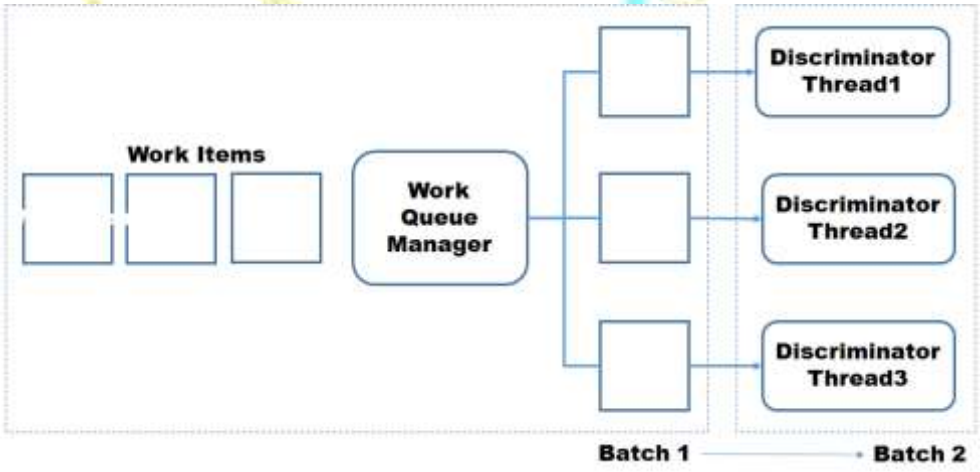

Fig. 2. Work Queue Pattern

In Figure 2, each discriminator thread adopts the function for differentiating incoming / outgoing traffic. Whole process run in work queue design pattern. At batch 1, data chunks are put in several storage area. At batch 2, discriminator thread is assigned to each data piece on the right side of Figure 2. Workers are scaled up or scaled down to ensure that the work can be handled. An illustration of a generic work queue is shown in Figure 2.

2.3 Discriminator Thread

Discriminator threads discussed in section 1.1 adopts the Thrust transform function for dividing session into ingoing or outgoing.

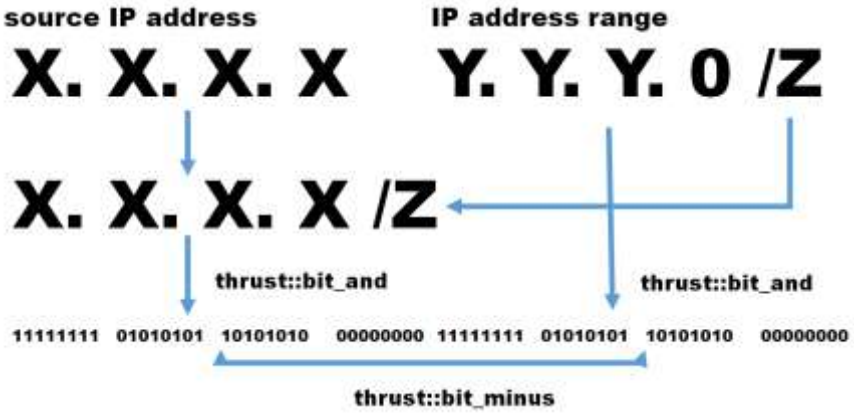

Fig. 3. Traffic Discrimination with Thrust::transform()

The detailed illustration of the discrimination process is shown in Figure 3.

Listing.1. Discrimination Threads

```
1: thrust::transform(IPaddress_dv.begin(), IPaddress_dv.end(), netmask_dv.begin(), masked_IPaddress_dv.begin(), thrust::bit_and<unsigned long>());

2: thrust::transform(masked_IPaddress_dv.begin(), masked_IPaddress_dv.end(), address_to_match_dv.begin(), result_dv.begin(), thrust::minus<double>());
```

2.4 Reduction with SPL Commands

At reduction phase, we also use Splunk. Splunk provides SPL (Search Processing Language) which is an expansive processing language for reducing or transforming large amounts of data into specific and relevant pieces of information. While SQL is implemented to search relational database based on columns, SPL is implemented to search events based on fields. SPL makes us check data which refers to fields whereas in SQL we check data which referes to table or column. We use two SPL commands of timechart and streamstat as follows:

Timechart command provides a statistical aggregation to a field to yield a chart with X-axis used as time. By using timechart command, we can specify a split-by field, in which each value of the split-by field corresponds to a series in the chart. Besides, with the split-by clause, you can construct a queue in detal further. An example of timechart command in our system is as follows:

Listing.2. Timechart Command

```
source="/mnt/data/sinet/aws/*" host="h-dev03" sourcetype="csv" earliest=-15d@d latest=-1d@d | timechart sum(count) span=10m
```

Streamstats command provides cumulative summary statistics of search result in a streaming manner. This command calculates statistics for each event at the time when the event is seen. The output of streamstats is figured out by applying the values in the specified field for every event under processing, up to each current event. Consequently, streamstas works on the group of outputs as a whole by calculating statistics for each event. An example of streamstats command in our system is as follows:

Listing.3. Streamstats Command

```
source="/mnt/data/geo//*" host="h-gpu03" sourcetype="csv"
earliest=-15d@d latest=-1d@d
| timechart count as count span=10m
| streamstats window=12 avg(count)
```

```
as avg stdev(count) as stdev
| eval lower_bound = avg - stdev * 2
| eval upper_bound = avg + stdev * 2
| eval isOutlier = if(count>upper_bound
OR count<lower_bound, 1, 0) | fields - avg stdev
```

III. TOOLS

3.1 Thrust Template Library

Thrust is a C++ template library, both implementing and facilitating the implemtation of parallel algorithms.Thrust's syntax resembles the Standard Template Library (STL), making it easy for seasoned C++ programmers to transition to parallel programming without going through the process of mastering complex tools like CUDA. Data in Thrust are stored in two types of vectors, which are functionally equivalent to the STL vector template class:

(1) For data residing in host memory.
(2) For data residing in device memory.

The novelty about using these vector types in that data transfer between the host and the device or vice versa, is implemented by overloading the assignment operator. Thrust provides efficient implementations for a number of important algorithms which can be used as building blocks to problem solutions. These including sorting, scanning, subset selection and reduction implementations. Not only does Thrust boost programmer productivity and program readability and maintenance, but it can also boost performance because it can adjust the execution configuration to the available GPU capabilities and resources. These algorithms fall into five categories.

(1) Transformations.
(2) Sorting and searching
(3) Reductions
(4) Scans/prefix-sums
(5) Data management/manipulation

Transformations operate an input sequence by applying a supplied operation on each element. Contrary to reduction, the produced output is equal to size (in terms of granularity) to the input.

3.2 Intel Threading Building Block

Intel Threading Building Blocks offers a rich and complete approach to expressing paarallelism in a p++ program. It is a library that helps you leverage multi-core processor performance without having to be a threading expert. Threading Building Blocks is not just a threads-replacement library; it represents a higher-level, task-based parallelism that abstracts platform details and threading mechanisms for performance and scalability. Highly concurrent containers are very important because standard Template Library (STL) containers generally are not concurrency-friendly, and attempts to modify them concurrently can easily corrupt the containers. As a result, it is standard practice to wrap a lock (mutex) around STL containers to make them safe for concurrent access, by letting only one thread operate on the container at a time. But that approach eliminates concurrency, and thus is not conducive to multi-core parallelism. Intel Threading Building bblocks provides highly concurrent containers that permit multiple threads to invoke a method simultaneously on the same container. At this time, a concurrent queue, vector, and hash map are provided.

(1) concurrrent queue
(2) vector
(3) hashmap

All of these highly concurrent containers can be used with this library, OpenMP, or raw threads.

3.3 Splunk and SPL commands

Splunk is a semi-structured time series database which can be used to index, search and analyze massive heterogeneous datasets. Now, as we have a large amount of data, there is a need for a platform or toll which can be used to create visualizations and derive insights and patterns to make informed business decisions beforehand. To overcome all these challenges of big data, Splunk came into the picture. Splunk is a big data tool that generates insights and reveals patterns, trends and associations from machine data. It is a powerful and robust big data tool used to derive real-time or near real-time insights, and it enables you to take informed corrective measures. Splunk can be put to use data generated from any source and available in a human readable format. Splunk is a feature-rich tool.

SPL provides over 140 commands that allow you to search, correlate, analyze and visualize any data—an incredibly powerful language that can be summarized in five key areas. Splunk's search processing language (SPL) helps you rapidly explore massive amounts of machine data to find the needle in the haystack and discover the root cause of incidents. IT operations that used to take days or months can now be accomplished in a matter of hours. Once you learn how powerful SPL is, you will wonder how you ever managed without it. When translating from any language to another, often the translation is longer because of idioms in the original language. Some of the Splunk search examples shown below could be more concise, but for parallelism and clarity, the SPL table and field names are kept the same as the SQL example.

IV. DESIGN PATTERNS

4.1 Pairwise Reduction Pattern using CUDA Thrust

A common way to accomplish parallel addition using GPGPU is pairwise reduction. In pairwise reduction, a chunk contains a pair of elements (key value). A thread sums two elements to yield one partial result.

These intermediate results are stored in-place in the original input vector. Then, new values (sums) are used to input for summing in the next generation. For each iteration, the number of input values halves, which results in that a final sum has been figured out when the length of output vector reaches one. Parallel reduction is one of the most popular parallel pattern.

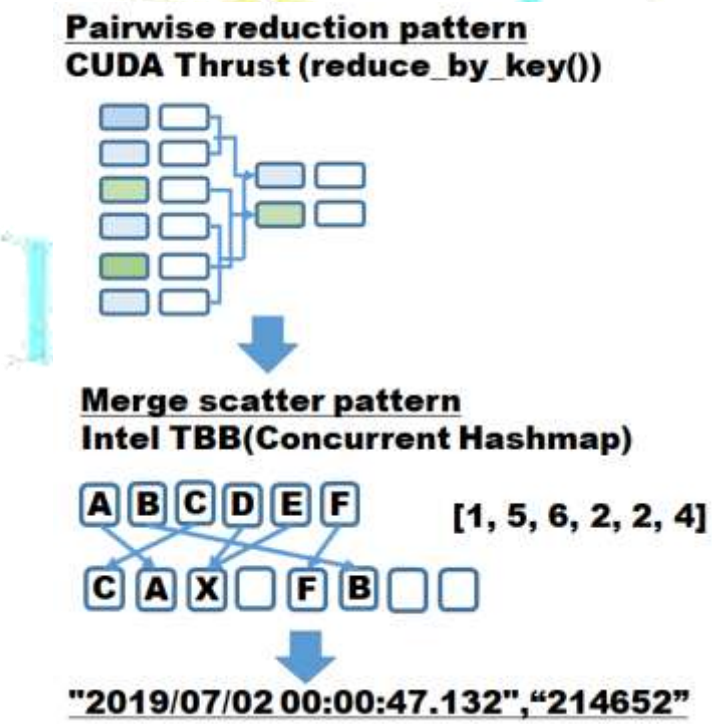

Fig. 4. Two Design Patterns of MapReduce

The output of histogramming is map such as <timestamp, count> and <timestamp, bytes>. Once the session data is loaded into device memory, histogramming can be executed on GPU. In this step, we use CUDA Thrust template library. Thrust is a C++ template library for CUDA based on the Standard Template Library (STL). Thrust allows you to implement high performance parallel applications with minimal programming effort through a high-level interface that is fully interoperable with CUDA C.

Thrust provides a rich collection of data parallel primitives such as scan, sort, and reduce, which can be composed together to implement complex algorithms with concise, readable source code. By describing your computation in terms of these high-level abstractions you provide Thrust with the freedom to select the most efficient implementation automatically. As a result, Thrust can be utilized in rapid prototyping of CUDA applications, where programmer productivity matters most, as well as in production, where robustness and absolute performance are crucial. We use cudaSetDevice(i) which sets device as the current device (i) for the calling host thread. The key point here is that GPU device ID is corresponding to thread ID by taking the argument of cudaSetDevice for thread_id. The result_A and d_B is stored in local memory for each thread. As shown in upper side of Figure 4, CUDA Trust API is invoked in eash Pthread in parallel. At line 8, cudaSetDevice is invoked to assign GPU to the reduction operation. We use cudaSetDevice(i) which sets device as the current device for the calling host thread. The key point here is that GPU device ID is correnponding to thread ID by taking the argument of cudaSetDevice for thread_id. The result d_A and d_B is stored in local memory for each thread.

Listing.4. Pairwise Reduction

```
1: void transfer(unsigned long long *key,
2:          long *value, int kBytes, int vBytes,
3:          int thread_id)
4: {
5:   unsigned long long *d_A;
6:   long *d_B;
7:
8:   cudaSetDevice(thread_id);
9:   cudaMalloc((unsigned long long**)&d_A,
10:                                        kBytes);
11:  cudaMalloc((long**)&d_B, vBytes);
12:  cudaMemcpy(d_A, key, kBytes,
13:  cudaMemcpyHostToDevice);
14:  cudaMemcpy(d_B, value, vBytes,
15:  cudaMemcpyHostToDevice);
16:  reduce();
17: }
```

At line 12-15, program transfer data to device memory of each GPU. At line 16, reduce() is called.

4.2 Merge Scatter Pattern using Intel TBB

In the merge scatter pattern, associative and commutative operators are provided for merging elements in case of a collision. With the nature of this pattern, scatter could occur in any order. Therefore, both associative and commutative properties are required. An example that uses addition as the merge operator is shown in Figure 4. It is straightforward to adopt merge scatter pattern to implement histograms. with the adding operation. The interval length of aggregation of our system is millisecond. Approximately, the granularity of histogramming is around 86,000,000 (60 * 60 * 24 * 1000 = 86,400,000). From our experience, it is difficult to evade lock contention to store 86,000,000 key-value into hash map in parallel. We give up using concurrent hash map which is effected by lock contention. Instead, key-value can be represented by using the defining

namespace such as X1<timestamp>, X1<count> and X2<timestamp> and X2<bytes>. Containers provided by Intel TBB offer a much higher level of concurrency, via one or both of the following methods:

Multiple threads operate on the container by locking only those portions they really need to lock. As long as different threads access different portions, they can proceed concurrently.

Different threads account and correct for the effects of other interfering threads.

Notice that highly-concurrent containers come at a cost. They typically have higher overheads than regular STL containers. Operations on highly-concurrent containers may take longer than for STL containers. Therefore, use highly-concurrent containers when the speedup from the additional concurrency that they enable outweighs their slower sequential performance. A concurrent_vector $ <T> $ is a dynamically growable array of T. It is safe to grow a concurrent_vector while other threads are also operating on elements of it, or even growing it themselves.

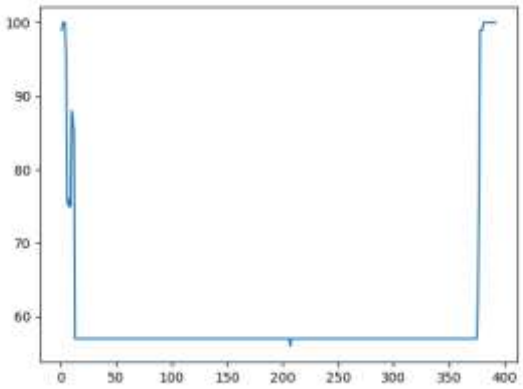

Fig. 5. CPU Idle Time in Processing Session Data of 97GB

V. EXPERIMENTAL RESULTS

In experiment, we use workstation with Intel(R) Xeon(R) CPU E5-2620 v4 (2.10GHz) and 512GB RAM. Figure 5 depicts CPU idle time in processing session data with 400,000,000 lines of 97 GB. X-axis of processing time (sec). Y-axis is CPU idle time. We have launched 48 POSIX Pthreads in both cases of Figure 5 and 6. In Figure 5, CPU idle time is decreased in first 15 seconds down to about 56 \%. Then, it is increased to about 100 \% in the last 15 seconds. Total elapsed time in processing 400,000,000 lines of 97 GB is about 390 seconds.

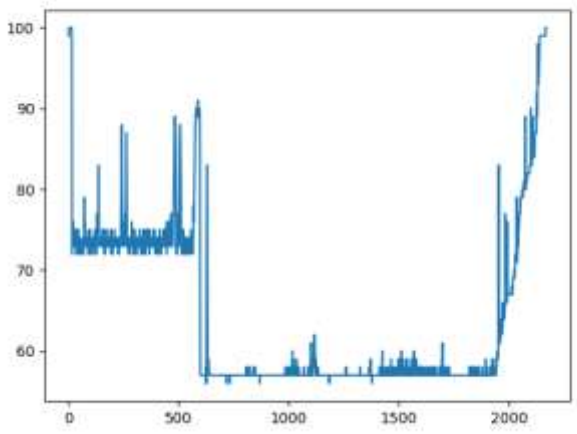

Fig. 2. CPU Idle Time in Processing Session Data of 400GB

Figure 6 depicts CPU idle time in processing session data with 2,000,000,000 lines of 400 GB. X-axis of processing time (sec). Y-axis is CPU idle time. On contrast to Figure 5, CPU idle time goes to plateau around 15 to 600 seconds. Then, it is decreased to about 55 \% during from about 600 to 1900 seconds

It has been turned out that proposal system is robust to the size of session data ranging from 97GB to 400 GB in the point of CPU idle time. That is, proposal system can provide stable resource utilization regardless of the size of huge session data.

VI. RELATED WORK

Fusco et al. proposes a methodology for indexing large amount of packets data per second by leveraging GPGPU [4]. Shredder [1] leverages GPU for the design of a high performance content-based chunking framework for supporting incremental storage and computation systems. There have been many research efforts on understanding traffic pattern of large-scale networks. Particularly, Li et al. proposes an empirical analysis of mobile user access pattern of huge cloud storage service [6]. In [10], Wang et al. show mobile traffic patterns of large scale networks in urban environment by gathering data in cellular towers. Sandiana National Laboratories (SNL) adopts Splunk for managing the Red Sky Supercomputer [9]. Bitincka et al. adopts Splunk for optimizing data analysis with a semi-structured time series database [2]. GPUs were initially designed for graphics rendering, but, because of their cost effectiveness, they were quickly adopted by the HPC community for scientific computations [7]. GPUs have also been used to accelerate functions such as pattern matching [5], network coding [8]. Ando [11] proposes a lock-free algorithm of data clustering using GPGPU. Ando [12] proposes a Multi-GPU based pipeline system with ELK stack.

VII. CONCLUSION

The Science Information Network (SINET) is a Japanese academic backbone network for more than 800 universities and research institutions. We have introduced the method of coordinated patch processing for coping with large scale session data using MapReduce framework. Our coordinated batch processing is divided into two design patterns: pairwise reduction and merge scatter reduction. In pairwise reduction, a chunk contains a pair of elements (key value) on GPGPU. On the other hand, the merge scatter pattern enables our system to associative and commutative operators are provided for merging elements in case of a collision on the container of Intel TBB. Besides, we introduce a multi-stage Reduce processing using GPU clusters to accelerate the processing of our pipeline. At the end of our pipeline, Splunk time-series indexer provides the operational insight for handling security incident response of security operation team in SINET. In experiment, we have measured CPU utilization in processing large scale session log file. It has been turned out that proposal system is robust to the size of session data ranging from 97GB to 400 GB in the point of CPU idle time. We can conclude proposal system can provide stable resource utilization regardless of the size of huge session data. In addition, we can avoid the unreasonable CPU utilization with the help of GPGPU. For further work, time-series analysis such as LSTM [13] could be applied to our pipeline design.

VIII. Acknowledgements

Author would like to thank NII-SOCS operation team.

REFERENCES

[1] Pramod Bhatotia, Rodrigo Rodrigues, and Akshat Verma. Shredder: Gpu-accelerated incremental storage and computation. In the 10th USENIX conference on File and Storage Technologies, page 14, 2012.

[2] Ledion Bitincka, Archana Ganapathi, Stephen Sorkin,and Steve Zhang. Optimizing data analysis with a semistructured time series database. In Workshop on Managing Systems via Log Analysis and Machine Learning Techniques, 2010.

[3] Brendan Burns. Designing distributed systems: Patterns and paradigms for scalable. In March 13, 2018, 2018.

[4] Xenofontas A. Dimitropoulos Francesco Fusco, Michail Vlachos and Luca Deri. Indexing million of packets per second using gpus. In Internet Measurement Conference, pages 327–332, 2013. Internet Measurement Conference.

[5] SHOJANIA H., LI B, , and WANG X. Nuclei. Gpuaccelerated many-core network coding. In IEEE Infocom, pages 459–467, 2009.

[6] Zhenyu Li, Xiaohui Wang, Ningjing Huang, Mohamed Ali Kafar, Zhenhua Li, Jianer Zhou, Gaogang Xie, and Peter Steenkiste. An empirical analysis of a large-scale mobile cloud storage service. In Internet Measurement Conference, pages 287–301, 2016. Internet Measurement Conference.

[7] J. D. LUEBKE OWENS, D.GOVINDARAJU, N.HARRIS, M.KRGER, J.LEFOHN, , and T. J. PURCELL. A survey of general-purpose computation on graphics hardware. In Computer Graphics Forum 26, 1 (2007), pages 80–113, 2007.

[8] R. SMITH, GOYAL N., ORMONT J., SANKARALINGAM. K, and ESTAN. C. Evaluating gpus for network packet signature matching. In Ie International Symposium on Performance Analysis of Systems and Software, 2009.

[9] Jon Stearley, Sophia Corwell, and Ken Lord. Bridging the gaps: Joining information sources with splunk. In Workshop on Managing Systems via Log Analysis and Machine Learning Techniques, 2010.

[10] Huandong Wang, Fengli Xu, Yong Li, Pengyu Zhang, and Depeng Jin. Understanding mobile traffic patterns of large scale cellular towers in urban environment. In Internet Measurement Conference, pages 225–238, 2015.

[11] Ruo Ando, A lock-free algorithm of tree-based reduction for large scale clustering on GPGPU. In Proceedings of the 2nd International Conference on Artificial Intelligence and Pattern Recognition, ACM 2019, ISBN 978-1-4503-7229-9, AIPR 2019, pp129-133, 2019

[12] Ruo Ando, "Multi-GPU Accelerated Processing of Time-Series Data of Huge Academic Backbone Network in ELK Stack", Usenix LISA 2019 (33th Large Installation System Administration Conference Large Installation System Administration Conference) October 28–30, 2019 Portland, OR, USA 10 2019

[13] Ruo Ando, Yoshiyasu Takefuji: “A constrained recursion algorithm for batch normalization of tree-sturctured LSTM”, CoRR abs/2008.09409 (2020)