International Journal of Advanced Computer Technology
ISSN: 2319-7900 www.ijact.org Volume-X, Issue-VI, December 2020

PARALLELIZING TIME-SERIES SESSION DATA ANALYSIS WITH A TYPE-ERASURE BASED DSEL

Ruo Ando, Center for Cybersecurity Research and Development, National Institute of Informatics, Japan; Youki Kadobayashi, Laboratory for Cyber Resilience Information Science Division, Nara Institute of Science and Technology, Japan; Hiroki Takakura, Center for Cybersecurity Research and Development, National Institute of Informatics, Japan

ABSTRACT

The Science Information Network (SINET) is a Japanese academic backbone network. SINET consists of more than 800 universities and research institutions. In the operation of a huge academic backbone network, more flexible querying technology is required to cope with massive time series session data and analysis of sophisticated cyber-attacks. This paper proposes a parallelizing DSEL (Domain Specific Embedded Language) processing for huge time-series session data. In our DESL, the function object is implemented by type erasure for constructing internal DSL for processing time-series data. Type erasure enables our parser to store function pointer and function object into the same *void type with class templates. We apply to scatter/gather pattern for concurrent DSEL parsing. Each thread parses DSEL to extract the tuple timestamp, source IP, and destination IP in the gather phase. In the scattering phase, we use a concurrent hash map to handle multiple thread outputs with our DSEL.

In the experiment, we have measured the elapsed time in parsing and inserting IPv4 address and timestamp data format ranging from 1,000 to 50,000 lines with 24-row items. We have also measured CPU idle time in processing 100,000,000 lines of session data with 5, 10 and 20 multiple threads. It has been turned out that the proposed method can work in feasible computing time in both cases.

INTRODUCTION

The Science Information Network (SINET) is a Japanese academic backbone network. SINET consists of more than 800 universities and research institutions. SINET serves various research facilities in space science, seismology, high-energy physics, nuclear fusion, and computing science. Currently, SINET is being used by over 2 million users. Also, SINET supports international research collaboration in the academic backbone network. Since 2016, NII has been running a service of NII-SOCS (NII Security Operation Collaboration Services). Our NII-SOCS team has deployed a security monitoring system consisting of PA-7080, Elasticsearch, Splunk, and NVidia Multi-GPU server. We introduce our system and some operational experience of handling huge session data ranging from 400,000,000 to 800,000,000 per day in this talk. During four years of 2016-2019, We have faced many challenges regarding the number of hosts, protocol proliferation, probe placement technologies, and security incident response.

The PA-7000 Series leverages a scalable architecture to adopt the flexible and powerful processing 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) are inspected in detail. A firewall such as PaloAlto-7080 plays an essential role in network security. Also, as cyber-attacks become sophisticated, the language to achieve efficiency and flexibility is required for complex intrusion detection tasks.

Table 1. Pa-7080 Data Description

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	dest country code	US
11	protocol	TCP
12	application	NA
13	subtype	NA
14	action	NA
15	session end reason	NA
16	category	NA
17	packets	0
18	packets sent	0
19	packets received	0
20	bytes	0
21	bytes sent	0
22	bytes received	0
23	device name	NA

For example, the query such as capture_time = 2020/11/** (No.1), source_IP=X.Y.0.0/16 (No.5), application=web_browsing (NO12). is required to detect session data under inspection. Unfortunately, although popular intrusion detection systems have their policy language with complicated logic requires architecture-dependent code. This paper proposes a DSEL (Domain Specific Embedded Language) for network traffic processing that can be real-world time-series session data on a huge academic backbone network.

OVERVIEW

In our system, we adopt folk-join pattern to handle multiple threads (flows). In folk-join parallelism, control flow folks (divides) into multiple flows which join (combine) later. After the folk, one flow is divided into two separate flows. Each flow is independent. After the join, only one flow continues.

In the aspect of reading chunks of time-series data, we apply the scatter-gather pattern. Specifically, the scatter/gather pattern enables you to achieve parallelism in servicing requests, enabling you to service them significantly faster than you could if you had to service them sequentially. Scatter/gather is quite useful when you have a large amount of mostly independent processing that is needed to handle a particular request.

In Figure 1, we apply our DSL for each data of time-series session data. Each thread parsing DSL stores key-value <timestamp, address_pair> into concurrent hashmap in the join phase. In other words, multiple flows of parsing DSL are reduced to one flow which uses concurrent highly concurrent hashmap as the lower side of Figure 1.

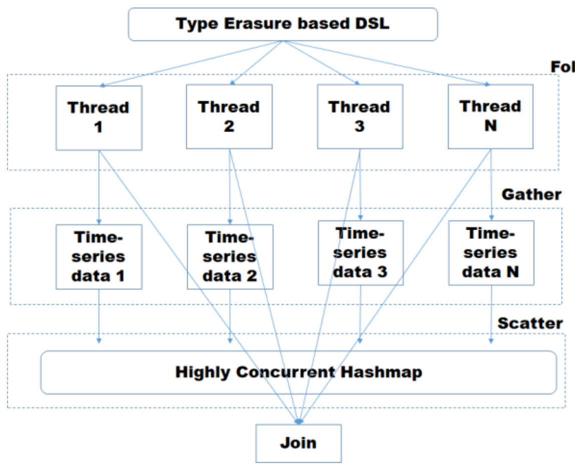

Figure 1. Folk-Join Pattern

Domain-Specific Language

This paper copes with two sorts of computer language: A domain-specific language (DSL) and a general-purpose language (GPL). DSL is designed for specializing a particular application domain, whereas GPL is designed for applicable across domains. Nowadays, DSL has a variety ranging from pervasively used languages such as HTML, XML, SQL, etc. DSL is further classified by the kind of language including domain-specific markup, domain-specific modelling and domain-specific programming languages. Also, DSL is sometimes called mini-languages because a single application uses it.

A. External and internal DSL

There are two main categories of DSL: external and internal. In external DSLs, a language is parsed independently of the host GPL. CSS with regular expressions is a good example of an external DSL. Internal DSLs are implemented with a particular form of API in a host GPL. A fluent interface [1] is often adopted in internal DSL.Mocking libraries such as JMock and Ruby on Rails are good examples of internal DSL. There has been a long tradition of usage of internal DSL, particularly in the LISP community.

Figure 2 shows the architecture of internal and external DSLs. In the view of typical compiler architecture, two kinds of DSL are common: parser, type checker and generator. However, in external DSL, the language is parsed independently of the host GPL and independent from the rest of the program. On contrast, internal DSL is implemented inside GPL. Giving up the flexibility of custom syntax of external DSL, internal GPL takes advantages in the learning curve and performance. Generally, internal DSL is easier to write because the language can be tailored to the idioms of the domain. In some cases, the code generator part is omitted in internal DSL.

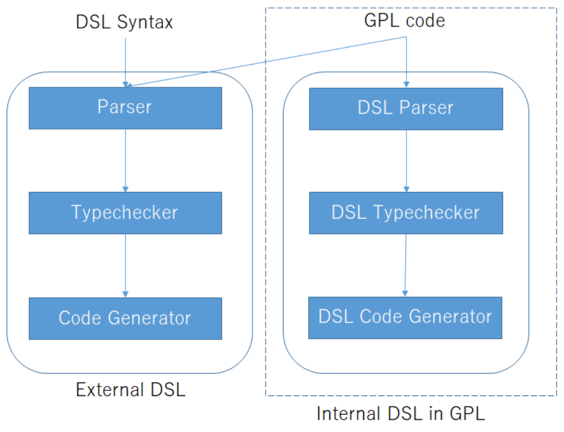

Figure 2. Internal/External DSL

B. DSEL

There is yet another representation of DSL - a domain-specific embedded language (DSEL). DSEL is almost the same meaning of internal DSL. However, DESL is the language which consists of many small internal DSLs. Strictly, DESL is a concept where small internal DSL interoperate with one another. Boost Spirit and YACC could be good examples of DSEL. The Boost Spirit parser framework is designed for recursive descent parser generation based on template metaprogramming techniques. One of Boost Spirit's core techniques is expression templates, enabling users to approximate the Syntax of ENBF (Extended Backus Naur Form) like grammar.

In Boost Spirit, parser object is a backtracking LL(∞) parser capable of parsing rather ambiguous grammars.

C. Higher-order programming

Function objects. A function object, which is also called a functor, allows the persistent object to operate functions like variables during execution. To put it simply, the main purpose of function objects is implementing callback functions.

Listing.1. Function Objects

```
1: bool is_substr_of( const string& sub, const string& all )
2: {
3:   return all.find( sub ) != string: npos;
4: }
5: int main()
6: {
7:    function<bool (const string&, const string&)> f;
8:    f = &is_substr_of;
9:    cout << f( "a", "abc" )  << endl;
10: }
```

At line 1-4 in Listing 1, the function object of is_substr_of() is defined. At line 7-8, the function object is generated and pointed to the variable f.

Binding Functions. Function objects become more effective with binding functions. To name a few, binding functions are lambda expressions, Boost. Phoenix and Boost.Bind. Compared with a straight function call, function objects have two thrusts (advantages). At first, a function object can hold state. Secondly, a function object is a type that results in it being utilized as the template parameters.

Linear static analysis C++ Boost provides Boost: bind, which is a generalization of the standard functions of std:bind1st and std:bind2nd. Bind supports arbitrary function objects, pointers, and member function pointers. Bind can connect any arguments or route input arguments in an arbitrary position. Also, the purpose of Bind is not placing any requirements with the function object. In particular, the result_type, first_argument_type and second_argument_type standard typedefs are not necessary in using Bind.

Lambda expression is an anonymous function utility provided by C++, Java, and so on. Broadly, the anonymous function is defined at the site where it is called. Lambda expression is originated from Alonzo Church's λ-calculus. The concept of anonymous comes from a function body but not bound to a function name. It takes advantages in generating a function definition at any point in the program's lexical scope, where you would expect to pass a function object.

Proposal method

Our DSEL requires the polymorphism corresponding to the data format of items are shown in TABLE I. For example, our parser needs to switch template functions by formats such as X.X.X.X (IP address) and YYYY-MM-DD (timestamp). In this case, function templates are not always the best way to handle polymorphism. Instead, we apply type erasure to handle several callback functions for each data items (address, timestamp, application, and so on).

A. Type erasure

Type erasure is a technique for removing explicit type annotations from a program in the load-time process. It is executed in compile-time (before run-time). Instead of type-passing semantics, type erasure adopts operational semantics which does not require programs accompany by types. In the view of the abstraction principle, type erasure ensures that the run-time program execution is independent of type information.

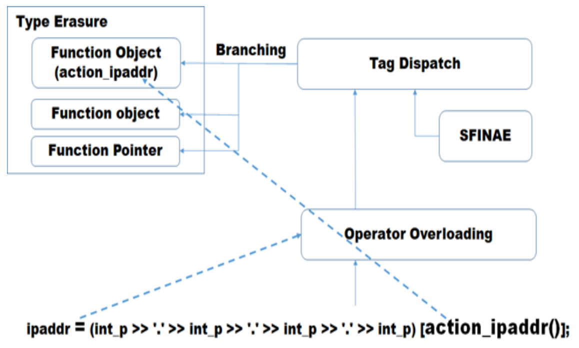
Type Erasure
Function Object (action_ipaddr)
Function object
Function Pointer
Branching
Tag Dispatch
SFINAE
Operator Overloading
ipaddr = (int_p >> '.' >> int_p >> '.' >> int_p >> '.' >> int_p) [action_ipaddr()];

Figure 3. Proposal Method Using Type Erasure

Listing.2. Type Erasure

```
union any_pointer {
 void (*func_ptr)();
 void* obj_ptr;
};
template <class Func, class R>
struct function_ptr_manager {
 static R invoke(any_pointer function_ptr)
 {}
};

template <class Func, class R>
struct function_obj_manager {
 static R invoke(any_pointer func_obj)
 {}
};
```

Figure 3 depicts our method using type erasure. In the upper-left side of Figure 3, function objects (including action_ipaddr) are stored with type erasure. Then, our program selects an appropriate function object by adopting tag dispatch. Further, tag dispatch is based on SFINAE(Substitution Failure Is Not an Error). SFINAE is the technique to avoid the compilation abort even if the substitution of the deduced type arguments in the template's argument list or function parameter list causes an error. Listing 2 shows the example of type erasure. Types of a function pointer and function object are erased and store in *void of the union at line 1-4.

B. Choice of parallelism

Generally, parallelism is divided into two categories: data parallelism and task parallelism. Data parallelism is more popular for scalable parallelism. Broadly, data parallelism is a design pattern that scales as the data set grows, broadly, as the problem size grows. Typically, the data is split into chunks and each chunk processed with a separate (and independent) task. In some cases, the splitting is recursive; in other cases, it is recursive. In the view of mechanisms which enables parallel computation, the two most important mechanisms are thread parallelism and vector parallelism:

Thread parallelism: Thread parallelism adopts a separate data flow of each worker. Thread parallelism also supports functional decomposition.

Vector parallelism: A mechanism for implementing parallelism directly on the hardware using the same data flow of control on multiple data chunks. Usually, vector parallelism supports regular parallelism. Vector parallelism also can be used for coping with irregular parallelism with some limitations.

In our system, thread parallelism is adopted. Another set of design patterns we applied is map and folk-join.

The map pattern, which is also called embarrassing parallelism, divides data into lots of uniform parts and represents a regular parallelization. The fork-join pattern adopts recursion for subdividing data into several chunks for both regular and irregular parallelization. Both patterns are used to achieve scalability of parallelism.

C. Scatter/gather

As we discussed before, we apply a scatter/gather pattern. We use Pthreads for reading chunks and writing a truncated line of session data. In scatter phase, we adopt Intel TBB's hashmap, which is a highly concurrent container. In the gather phase, multiple Pthreads runs in task decomposition.

Task decomposition: If we want to transform code into a concurrent version, there are two ways. First one is data decomposition, in which the program cope with a large collection of data and can compute every chunk of the data independently. The second one is task decomposition, in which the process is partitioned into a set of independent tasks that threads can execute in any order. Data decomposition has some drawbacks. For example, the size of split session data files varies according to the situation in which the data is retrieved.

More specifically, the master thread traverses session data file directory and enqueue the file name. When the queue is full, the master thread waits until the worker thread processing packets consumes a file name and removes it from the queue.

Highly concurrent hashmap: The interval length of aggregation of our system is millisecond. Approximately, the granularity of histogram Ming is around 86,000,000 (60 * 60 * 24 * 1000 = 86,400,000). It is difficult to evade lock contention to store 86,000,000 key-value into a hash map parallel from our experience. We give up using a concurrent hash map which is affected 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.

Intel TBB: We use Intel Threading Building Blocks (TBB) for scatter/gather pattern. TBB is designed on a tasking model for providing features of parallel patterns (map and folk-join) and a collection of thread-safe data structures such as concurrent hashmap. The TBB is implemented to avoid global locks. For example, there is no global task queue and memory allocator, causing locks contention.

The TBB implementation generally evades global lock contention in its implementation. In particular, there is no global task queue, and the memory allocator is lock-free. Listing 3 shows the code for inserting key-value into a concurrent hashmap.

Listing.3. Inserting key-value into hashmap

```
1: iTbb_Map_map::accessor t;
2: TbbVec_Boost_Map.insert(t, timestamp_tmp);
3: t->second = addrString3;
```

Generally, high concurrent containers (including hashmap) are more expensive than STL containers. Highly concurrent containers have more overheads and take longer time than STL containers. Therefore, highly concurrent containers are recommended to speed up from the additional concurrency, which can outperform their slower sequential performance.

B. Parsing expression grammar

Parsing Expression Grammars (PEG) [2] is a derivative of the Extended Backus-Naur Form (EBNF) [3]. PEG is implemented to cope with a decent recursive parser. In other words, a PEG can be interpreted in a recursive-descent parser's manner like EBNF in a more direct manner. PEG is designed to describe a formal language representing a set of rules applied for parsing and recognizing strings and tokens. Another advantage of PEG over EBNF is that it performs with an exact interpretation. In each PEG grammar, only one valid parse tree is determined.

Listing.4. Paring Expression Grammar

```
1: definition( const AddrParse& self )
2: {
3: ipaddr = (int_p >> '.' >> int_p  >> '.' >> int_p >> '.' >> int_p)
[Action_ipaddr()];
4:
5: timestamp = (int_p >> '-' >> int_p >> '-'  >> int_p >> 'T' >>
int_p >> ':' >> int_p  >> ':' >> int_p >> 'Z')
6: [Action_timestamp()];
7
8: r = timestamp | ipaddr;
9: }
```

Listing 4 shows the example of PEG of Boost Spirit. At line 3, the program defines the format of an IP address (X.X.X.X). The timestamp format (YYYY-MM-DDThh:mm: ss) is defined at line 5-6.

EXPERIMENTAL RESULT

In experiment, we use workstation with Intel(R) Xeon(R) CPU E5-2620 v4 (2.10GHz) and 251G RAM. Figure 1 depicts the elapsed time in parsing session data log. The X-axis is the number of lines of session data log file. Y-axis is the elapsed time. The parsing log file's elapsed time increases linearly corresponding to the file size except for some spikes such as around 45,000. Figure 2 depicts the elapsed time in inserting key-value pair data into multi-index. The X-axis is the number of lines of session data log file. Y-axis is the elapsed time.

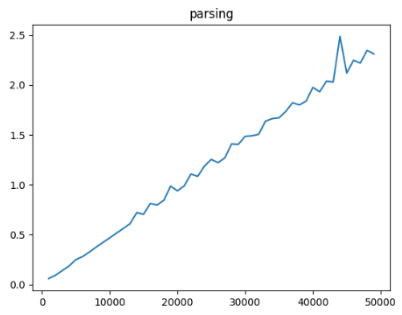

Figure 3. Elapsed Time in Parsing Session Log File

Results in Figure 3 and 4 have been obtained in the same execution of our parse. Also, the elapsed time of parsing log file increases linearly corresponding to the file size. It has been turned out that the proposed method can work in feasible computing time. We have also measured CPU idle time with multiple threads parsing DSEL. Table II shows the comparison of CPU idle time is running 5, 10 and 20 threads. It has become clear that we can speed up by increasing threads from 5 to 10. However, there is no difference between 10 and 20 threads in the measurement of CPU idle time. Figure 6,7 and 8 are time-series data of CPU idle time with multiple threads 5, 10 and 20. The X-axis is second. Y-axis is CPU idle time. In three figures, we have observed two plateaus. The first plateau finishes around 4800 (5 threads), 2800 (10 threads) and 1800 (20 threads).

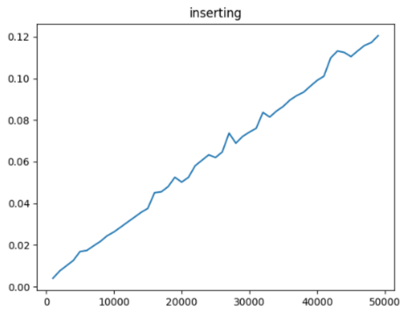

Figure 4. Elapsed Time in Inserting Key-Value into Multi-Index

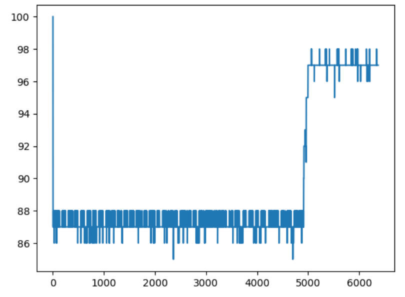

Figure 5. CPU idle time with 5 threads parsing DSEL.

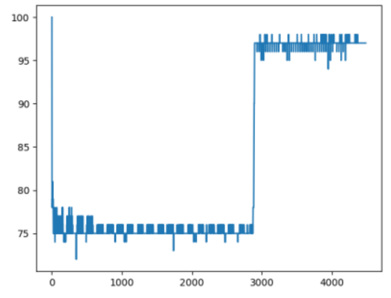

Figure 6. CPU idle time with 10 threads parsing DSEL.

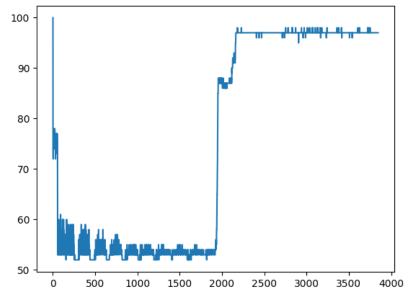

Figure 7. CPU idle time with 20 threads parsing DSEL

It is reasonable in the view of the number of multiple threads. However, the length of the second plateau differs much in three cases. The third case of 20 threads shown in Figure 8 has the longest plateau ranging from about 2000 to 4000. As a result, the third case is the worst in the view of effective resource utilization. We can conclude that the second case with 10 threads shown in Figure 7 is the best configuration.

RELATED WORK

A. Metaprogramming

Various approaches of metaprogramming [4] have been proposed to cope with the embedding of DSLs. One of these approaches is code generation, where code is converted to AST representation for processing the embedded DSL program at compile-time. The multi-state programming approach [5] is proposed to interpret a program in several multiple phases. In [5], the compiler operates at run-time, but at a different phase than the actual processing block statements. Another distinguished research about applying metaprogramming for DSL is proposed by Seefried et al. [6]. In [6], Template Haskell [7] is used to implement PanTHeon and Pan [8].

B. Functor

One of the concepts of functor [9] [10] [11] is originated from the module systems of SML [10] and OCaml [11]. By using functor, [10] and [11] can abstract over required statically type-checked manner. Functor still imposes severe restrictions on module systems in the case of structuring components. Accordingly, a functor is used with function-binding utilities such as Lambda expression and Boost. Bind.

C. Pattern matching

Various techniques of pattern matching in object-oriented programming have been proposed to message exchange in distributed systems [13], semi-structured data [14], and UI event handling [15]. Moreau, Ringeissen and Vittek [16] propose using pattern matching code into existing languages, without any requirement extensions. For Java, Liu and Myers [17] add a pattern matching construct using a backward execution mode. An alternative technique of multi-methods [18] is proposed. In [18], pattern matching is unified with method dispatch. Also, [19] [20] extends multi-methods to predicate-dispatch. In [19] [20], functional programming languages are proposed to convert from one data type to another in pattern matching.

D. Parser expression grammar

Parser expression grammar is inspired by Birman's TS/TDPL and GTS/GTDPL systems [23] [24] [25]. Adams [26] adopts TDPL in a modular language prototyping framework. Also, various practical top-down parsers such as ANTLR [27], PARSEC combinator library for Haskell [28] are available. These top-down parsers provide backtracking capabilities that conform to the model in practice.

E. Time-series log analysis

Another important topic of time series analysis is outlier/anomaly detection. In [29], a data structure called k-ary sketch is proposed for efficient utilization of memory. Also, k-ary sketch enables a constant, per-period update and reconstruction cost. Popular algorithms of anomaly detection of temporal data are ARIMA, HMM and SVM. Pandu [30] et al. proposes a sequence-based analysis using SVM and HVM for anomaly detection of time-sequence of instrumentation data of VMM (virtual machine monitor).

F. DSL

Configurable language for network traffic analysis and intrusion detection is a promising application of DSL. PADS [31] is a declarative data description language for describing both the physical layout and semantic properties of ad hoc data traffic. As an extension of PADS, Fisher [32] proposes an automated inference algorithm of ad hoc data source structure and a format specification in the PADS. Chimera [33] provides a declarative query language for intrusion detection systems with a platform-independent SQL syntax. SQL [34] is a stream-based query system for incorporating expert knowledge to perform timely anomaly detection in large scale traffic data.

G. DSL for concurrency

There have been many research efforts on DSL for concurrent computing. DiLorenzo proposes Transactional forest for concurrent file stores using serializable transactions[37]. Chiw proposes Diderot, a parallel domain-specific language for biomedical image analysis and visualization[38].

Anderson [38] presents Parallel Accelerator, a library and compiler for high-performance scientific computing in Julia. Waltz and Pollack [40] present description research in developing a natural language processing system with modular knowledge sources but strongly interactive processing.

CONCLUSION

As attacks are increasing in sophistication, analytics should also be sophisticated that detect them. Network traffic, in general, has become more invisible. To name a few, major cloud vendors such as Cloudflare recently deploy DNS over TLS/HTTPS. Also, with the spread of TLS 1.3, the middlebox appliances become less effective. Consequently, current Botnet running over the cloud platform is harder and harder to detect and analyze.

In this paper, the parallelizing DSEL (Domain Specific Embedded Language) processing for huge time-series session data has been proposed. In our DESL, the function object is implemented by type erasure for constructing internal DSL for processing time-series data. Type erasure enables our parser to store function pointer and function object into the same *void type with class templates.

Also, as the Internet traffic keeps increasing rapidly, we should cope with hundreds of gigabytes session data ranging from 500 to 1000 million lines. To cope with this challenge, we apply a scatter/gather pattern for concurrent DSEL parsing. Each thread parses DSEL to extract the tuple timestamp, source IP, and destination IP in the gather phase. In the scattering phase, we use a concurrent hash map to handle multiple thread outputs with our DSEL.

In the experiment, we have measured the elapsed time in parsing and inserting IPv4 address and timestamp data format ranging from 1,000 to 50,000 lines with 24-row items. We have also measured CPU idle time in processing 100,000,000 lines of session data with 5, 10 and 20 multiple threads. It has been turned out that the proposed method can work in feasible computing time in both cases.

In the current situation, declarative languages maintain as much expressive power as possible while not imposing the significantly impacting intrusion detection systems' performance. Also, the more flexible framework is necessary for providing logical construction of the expression of sophisticated attacks.

REFERENCES

[1]. I. Brcic: "Ideally Fast" Decimal Counters with Bistables. IEEE Trans. Electron. Comput. 14(5): 733-737 (1965)

[2]. Bryan Ford: Parsing Expression Grammars: A recognition-based Syntactic Foundation, http://pdos.csail.mit.edu/~baford/packrat/popl04/

[3]. Richard E. Pattis: EBNF: A Notation to Describe Syntax, http://www.cs.cmu.edu/~pattis/misc/ebnf.pdf

[4]. K. Czarnecki, J. T. O'Donnell, J. Striegnitz, and W. Taha. DSL implementation in MetaOCaml, Template Haskell, and C++. In Domain-Specific Program Generation, volume 3016 of LNCS, pages 51-72. Springer, 2003.

[5]. W. Taha. A gentle introduction to multi-stage programming. In Domain-Specific Program Generation, Springer LNCS 3016, pages 30-50, 2003.

[6]. S. Seefried, M. M. T. Chakravarty, and G. Keller. Optimising Embedded DSLs using Template Haskell. G. Karsai and E. Visser, editors, GPCE, volume 3286 of Lecture Notes in Computer Science, pages 186-205. Springer, 2004.

[7]. T. Sheard and S. Peyton Jones. Template metaprogramming for Haskell. In M. M. T. Chakravarty, editor, ACM SIGPLAN Haskell Workshop 02, pages 1-16. ACM Press, Oct. 2002.

[8]. C. Elliott. Functional images. In the Fun of Programming, "Cornerstones of Computing" series. Palgrave, Mar. 2003.

[9]. D. MacQueen, "Modules for Standard ML", in Proceedings of the 1984 ACM Symposium on Lisp and Functional Programming, Papers Presented at the Symposium, August 1984, pages 198-207, New York, August 1984. Association for Computing Machinery.

[10]. R. Harper and M. Lillibridge, "A Type-Theoretic Approach to Higher-Order Modules with Sharing", In Proceedings of 21st ACM Symposium on Principles of Programming Languages, January 1994.

[11]. X. Leroy. Manifest Types, Modules and Separate Compilation. In Proceedings of 21st ACM Symposium on Principles of Programming Languages, pages 109-122, January 1994.

[12]. K. Fisher and J. H. Reppy. "The Design of a Class Mechanism for Moby", In proc of SIGPLAN Conference on Programming Language Design and Implementation, pages 37-49, 1999.

[13]. Lee, K., LaMarca, A., Chambers, C.: HydroJ: Object-oriented Pattern Matching for Evolvable Distributed Systems. In: Proc. of Object-Oriented Programming Systems and Languages (OOPSLA). (2003)

[14]. Gapeyev, V., Pierce, B.C.: Regular Object Types. In: Proc. of European Conference on Object-oriented Programming (ECOOP). (2003)

[15]. Chin, B., Millstein, T.: Responders, "Language Support for Interactive Applications", In Proc of European Conference on Object-Oriented Programming (ECOOP). (2006)

[16]. Moreau, P.E., Ringeissen, C., Vittek, M.: A Pattern Matching Compiler for Multiple Target Languages. In: In Proc. of Compiler Construction (CC), volume 2622 of LNCS. (2003) 61-76

[17]. Liu, J., Myers, A.C.", JMatch: Iterable Abstract Pattern Matching for Java", In proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages (PADL). (2003) 110-127

[18]. Clifton, C., Millstein, T., Leavens, G.T., Chambers, C., "Multi Java: Design Rationale, Compiler Implementation, and Applications" in Prof of ACM Transactions on Programming Languages and Systems 28(3) (May 2006) 517-575.

[19]. Ernst, M., Kaplan, C., Chambers, C.: Predicate dispatching: a unified theory of dispatch. In Proceedings of Euro-pean Conference on Object-Oriented Programming (ECOOP). Volume 1445 of Springer LNCS. (1998) 186-211

[20]. Millstein, T.: Practical Predicate Dispatch. In: Proc. of Object-Oriented Programming Systems and Languages (OOPSLA). (2004)'

[21]. Wadler, P., "Views: A way for pattern matching to cohabit with data abstraction", In Proceedings of Principles of Programming Languages (POPL). (1987)

[22]. Okasaki, C.: Views for Standard ML. In: In SIGPLAN Workshop on ML, pages 14-23. (1998)

[23]. Alfred V. Aho and Jeffrey D. Ullman. The Theory of Parsing, Translation and Compiling - Vol. I: Parsing. Prentice-Hall, Englewood Cliffs, N.J., 1972.

[24]. Alexander Birman. The TMG Recognition Schema. PhD thesis, Princeton University, February 1970.
[25]. Alexander Birman and Jeffrey D. Ullman. Parsing algorithms with backtrack. Information and Control, 23(1):1-34, August 1973.
[26]. Stephen Robert Adams. Modular Grammars for Programming Language Prototyping. PhD thesis, University of Southampton, 1991.
[27]. Terence J. Parr and Russell W. Quong. ANTLR: A Predicated LL(k) parser generator. Software Practice and Experience, 25(7):789-810, 1995
[28]. Daan Leijen. Parsec, a fast combinator parser. http://www.cs.uu.nl/?daan.
[29]. Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, Yan Chen: Sketch-based change detection: methods, evaluation, and applications. Internet Measurement Conference 2003: 234-247
[30]. Ady Wahyudi Paundu, Takeshi Okuda, Youki Kadobayashi, Suguru Yamaguchi: Sequence-Based Analysis of Static Probe Instrumentation Data for a VMM-Based Anomaly Detection System. CSCloud 2016: 84-94
[31]. Kathleen Fisher, Robert Gruber: PADS: a domain-specific language for processing ad hoc data. PLDI, 2005: 295-304.
[32]. Kathleen Fisher, David Walker, Kenny Qili Zhu, Peter White: From dirt to shovels: fully automatic tool generation from ad hoc data. POPL 2008: 421-434
[33]. Kevin Borders, Jonathan Springer, Matthew Burnside: Chimera: A Declarative Language for Streaming Network Traffic Analysis. USENIX Security Symposium 2012: 365-379
[34]. Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R. Kulkarni, Prateek Mittal: SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection. USENIX Security Symposium, 2018: 639-656.
[35]. Vern Paxson: Bro: a system for detecting network intruders in real-time. Comput. Networks 31(23-24): 2435-2463 (1999)
[36]. Martin Roesch: Snort: Lightweight Intrusion Detection for Networks. LISA 1999: 229-238
[37]. Jonathan DiLorenzo, Katie Mancini, Kathleen Fisher, Nate Foster: TxForest: A DSL for Concurrent File stores. APLAS 2019: 332-354
[38]. Charisee Chiw, Gordon L. Kindlmann, John H. Reppy, Lamont Samuels, Nick Seltzer: Diderot: a parallel DSL for image analysis and visualization. PLDI 2012: 111-120
[39]. Todd A. Anderson, Hai Liu, Lindsey Kuper, Ehsan Totoni, Jan Vitek, Tatiana Shpeisman: Parallelizing Julia with a Non-Invasive DSL. ECOOP 2017: 4:1-4:29
[40]. Waltz, D.; Pollack, J. (1985). "Massively parallel parsing: A strongly interactive model of natural language interpretation". Cognitive Science. 9: 51–74.

BIOGRAPHIES

Ruo Ando has received PhD from Keio University in 2006. He is now an associate professor by special appointment of National Institute of Informatics since 2016. Before joining NII, he worked as a senior security researcher of the National Institute of Information and Communication Technology since 2006. He received the Outstanding Leadership Award in the 8th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC-09) at China in 2009. He is the member of Trusted Computing Group JRF (Japan Regional Forum) in 2008-2015. He served as a reviewer of Springer Journal PPNA, Willey Journal of Security and Communications Networks and IEEE transactions of Information Forensics and Security. He worked in project Next-Generation Security Info-Security R & D METI (FY2008-10). He was engaged in “Unknown malware detection using incremental malware detection” MEXT FY(2012-2015). He served as a reviewer of Springer Journal, Willey Journal of Security and Communications Networks and IEEE transactions of Information Forensics and Security.

Youki Kadobayashi works for NICT Japan as an invited expert, where his role as the Rapporteur of ITU-T Study Group 17 Question 4 (Cybersecurity) has been supported since 2008. At SG17, he has been working with other experts to gather insights on cybersecurity information exchange for more than five years – results of the study activities have been made available under ITU-T Recommendation series of X.1500 known as CYBEX. He also conducts lectures and hands-on exercises on cybersecurity at the Nara Institute of Science and Technology, Japan, for which he has been working as an Associate Professor since 2000. Since last year, he has participated in several international research collaboration programs; he has been coordinating the FP7 NECOMA project, which has been jointly funded by the European Commission and MIC of Japan. At the NECOMA project, he leverages big data, SDN, and cloud computing technologies for improving cyber-resilience. He is deeply committed to the Asia-Pacific region; he has delivered talks on cybersecurity standards at past Asia-Pacific Tele community Cybersecurity Forum meetings. He also delivers guest lectures on cybersecurity and cloud computing security at the Unitec Institute of Technology in New Zealand as an Adjunct Professor.

Hiroki Takakura is Director of Cybersecurity Research and Development, National Institute of Informatics. In that capacity, he takes the direction on its cybersecurity research. He has also supervised NII Security Operation Collaboration Services (NII-SOCS) to detect, analyze and pursue cyberattacks against 100 national universities since 2017. With his experiences and knowledge on cybersecurity, he is one member of the Ministry of Health, Labor and Welfare advisory group to support the reform in data health/examination and payment agency. He makes an effort to realize robust and resilient digital healthcare services. He received his B.S. and

M.S degrees from Kyushu University in1990 and 1992 respectively. In 1995, he received Dr Eng. degree from Kyoto University in 1995. After research activities at the University of Illinois at Urbana-Champaign (Visiting Scholar), Nara Institute of Science and Technology (Research Associate), Kyoto University (Lecturer and Associate Professor) and Nagoya University (Professor), he is currently a professor at National Institute of Informatics since 2015. Since 2016 he has become a director of the Center for Cybersecurity Research and Development, NII.