EasyChair Preprint

№ 4707

Data Transformation: An Overview

Deni Daja

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

December 7, 2020

Data Transformation: An Overview

Deni Daja

December 4, 2020

Abstract

With the accelerated need for scalability and reliability in todays growing systems, most modern services are shifting towards a microservice-based approach as opposed to a monolithic approach. While the benefits of this new paradigm are undeniable, we also have to be aware of a new set of challenges that microservices bring. To support the need for microservice deployment in scale, data centers and their internal network (DCN) are usually the preferred approach. This network can become especially a bottleneck in microservices deployed in a data center, as their communication needs to be as real-time as possible. In order to improve network latency, data transformation capabilities are needed to convert messages back and forth between microservices from their internal formats. This paper discusses the current approaches on data transformation, the bottlenecks that come with current solutions and then explore a new paradigm for Data Transformation, namely Optimus Prime, that introduces a new approach on Data Transformation by using a new transformation schema and a hardware accelerator for optimal optimal cross-microservice communication.

Contents

1 Introduction of the Research Field . . . **2**

2 Basics . . . **3**
2.1 General Terms . . . 3
2.2 OS Concepts . . . 3

3 Previous and Related Work . . . **4**
3.1 Popular Data Transformation Frameworks . . . 4
3.2 New Paradigms on Data Transformation . . . 5

4 Optimus Prime's Approach . . . **5**
4.1 Motivation . . . 5
4.2 The Current Bottleneck . . . 5
4.3 Software Component: Schema Abstraction . . . 6
4.4 DTA General Architecture . . . 6
4.5 Optimus Prime Implementation . . . 7
4.6 Optimizations . . . 8

5 Evaluation . . . **8**
5.1 Evaluation Basis . . . 8
5.2 Single Pipeline Results . . . 9
5.3 Multi Pipeline Results . . . 9

6 Discussion . . . **10**

7 Conclusion . . . **10**

1 Introduction of the Research Field

A microservice architecture is a style of building applications that functionally decomposes an application into a set of services. Each of these services might be written in a different programming language and can be deployed, scaled and maintained independently. This architectural style is undeniably a perfect candidate for tackling issues that come with the ever-increasing demands on scalability, fast iterations, agility, reliability and cross-functional teams. But every silver lining has a cloud, the microservice approach brings a whole new set of problems to the table for engineers. Managing a growing number of services, monitoring, tracing, testing and handling cyclic dependencies are only a set of problems that need to be handled. Another critical concern is raised by the question: **How will the services communicate?**

In a monolithic paradigm, network is not an issue to be handled. On the other hand, in microservices, network and the accompanying latency might become a huge pain point for companies that aim to provide the best service to their customers. For this reason, an ever-growing approach is to deploy every service in the same Data Center. The Data Center Network (DCN) has tremendous bandwidth capabilities, reaching up to 100Gbps and currently underway to support up to 1Tbps [AGM+10, inf18, eth18]. This means that the services deployed in a Data Center have the potential to communicate in a lightning-fast manner, of course, if they are able to capitalise and make use of this tremendous bandwidth.

Unfortunately, this is not the case. While microservice communication in a data center is indeed fast, it is still far away from the real potential offered. And because networks and protocols have already been refined to a point of almost no further possible improvements, the only remaining candidate to be improved is the server-side itself, and more precisely, the data transformation protocol between each machine before they send the data via the network.

As it stands, there are currently several ways that services use to communicate with each other, with the most prevalent methods being REST and gRPC. REST uses string-based serialization or message format, while gRPC uses binary-based serialization, also called protocol-buffer or Protobuf. As string-based serialization usually comes with performance bottlenecks (text encode/decode, complex parse code, more bandwidth consumption) - gRPC is a better candidate for achieving higher transfer rates between microservices. A typical RPC communication process is also illustrated on Figure 1.

But apparently, even the most optimized data transformation frameworks like Protobuf do not give us the desired communication speed that a data center network can handle. Given the improvements of networks and protocol, the next step to look for improvements is obviously the Data Transformation frameworks and see why they are currently failing to achieve higher rates, and how can these barriers be broken.

Today's research has been showing that the issue with the current data transformation frameworks is mostly the fact that the current frameworks delegate the transformation process to the CPU, which has inherent limitations related to implicit instruction-level parallelization and high instruction count per serialized field. New approaches like Optimus Prime [PGK+20], Intel Data Streaming Accelerator [Cor19] and even new CPU ISA [isa16] instructions are bringing a new paradigm to the table, which integrate new hardware-related methods that are specialized in transforming data in parallel and acheiving maximal rates. *In this paper, we will take a closer look at Optimus Prime and its innovative approach to Data Transformation.*

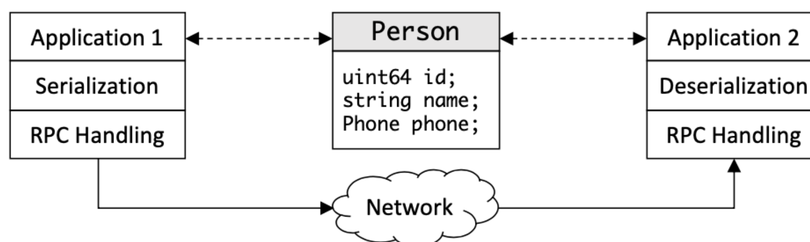

Figure 1: RPC communication protocol

2 Basics

The upcoming topic of hardware-accelerated data transformation touches on several critical concepts that need to be understood before diving deeper.

2.1 General Terms

A Data center is a pool of resources (computational, storage, network) interconnected using a communication network. This network, called *Data Center Network (DCN)* , holds a pivotal role in a data center as it interconnects all of the data center resources together. DCNs need to be scalable and efficient to connect tens or even hundreds of thousands of servers to handle the growing demands of Cloud Computing [dat20a].

Data Transformation is the process of changing the format, structure, or values of data [dat20b]. Because code written in microservices is often in different languages with their own data formats, data transformation is needed to convert to and from the desired format. This process is often called serialization.

Accelerators in general have the goal to help in boosting the overall performance of a computer. Hardware accelerators are used to enhance the speed and performance of the computer as it works by performing functions faster than the central processing unit (CPU). So basically, a hardware accelerator, is an optimized and specialized hardware device that intends to perform a specific action faster than the CPU.

Network terms like network speed, bandwidth and throughput will be used in the upcoming evaluations. Network speed measures the transfer rate of data from a source system to a destination system, while network bandwidth is the amount of data that can be transferred per second. Combine the two, and you have what is known as network throughput - which is the main evaluation criteria used in this paper.

2.2 OS Concepts

A clock cycle indicates a single tick of the internal system clock. Cycles per second are also called Hertz. Some instructions on a CPU take multiple cycles to execute, and optimization means in most cases multiple instructions are executed in a single cycle.

ISA (Instruction Set Architecture) is the list of commands that are hard-wired in to the CPU. It is part the processor that the programmer or compiler writer leverages to run programs.

Speculative execution is an optimization technique where a computer system performs some task that may not be needed to improve concurrency [spe20].

A data buffer (or just buffer) is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.

Time-sharing is the sharing of a computing resource among many users at the same time by means of multiprogramming and multi-tasking [tim20].

Virtual Memory is a memory management technique that provides an idealized abstraction of the storage resources that are actually available on a given machine which creates the illusion to users of a very large (main) memory [vir20].

3 Previous and Related Work

When talking about data transformation performance, a number of papers have compared the different existing approaches, mostly focusing on the string-based and binary-based protocols like XML, JSON, Apache Thrift and Protobuf. Newer approaches that support data transformation on a hardware level like Optimus Prime are quickly gaining attraction, but not much related work is available given that it is a relatively new plane of research and development. Nevertheless, some upcoming projects for transformations accelerated by hardware will be discussed.

3.1 Popular Data Transformation Frameworks

The usual discussions on data transformation mention some of the technologies that have been circulating for a considerable amount of time. The common factor among these methods is that they all leverage the CPU and ISAs to perform the transformations. Let's have a quick overview of these protocols.

The first line of solutions that leverage the CPU for transformations are **string-based serialization protocols like XML and JSON** . These two frameworks are the most common modern data serialization formats. XML came prior to JSON but it is not the protocol of choice when it comes to serialization. The reason for this being that XML is extremely verbose and heavyweight. This makes JSON a more preferred approach to modern string-based serialization. With that being said, these two formats still have several advantages, with ease of use on the developer's side and human-readability being the most attractive feature. One of the main disadvantages that they possess are the fact that they are text-based, meaning they will always need to be parsed character by character, thus imposing a limit on deserialization speed. For this reason, binary serialization protocols were implemented.

The next solutions that make use of ISAs are the **binary-based serialization frameworks like Protobuf and Thrift** . These binary formats are not subject to the disadvantage of parsing as the JSON and XML formats. They make use of 'positional binding' and shared files that are described as *schemas* between server/client that define the message format and greatly decrease the size of data being transmitted. So in general, Protobuf is a simpler, faster and smaller protocol for data transfer.

In a strict comparison between these formats, the binary serialization protocols have an expected lead in transmission time. In an experiment performed in [SM12], Protobuf and Thrift scored almost 10x faster times for serialization than XML and 2x faster times than JSON. The deserialization experiment led to even greater differences between the protocols, with Protobuf and Thrift achieving 20x faster times than XML and x5 times faster than JSON.

What we can deduce from this is that binary protocols are clearly faster. But with that being said, unfortunately, they still are the biggest overhead that block microservices from communicating with the potential speed that modern NICs offer. To tackle this issue, a new outlook currently being explored is hardware-accelerated data transformation frameworks and protocols. In the next section we will have a deeper look into this new paradigm shift.

3.2 New Paradigms on Data Transformation

Hardware-assisted Data Transformation is a new and innovative approach that is beginning to emerge. As this remains a relatively new topic, not much related work is present, but let's quickly go over some promising solutions that already exist or are on the pipeline.

ISA Extensions for DT. The CPU vendors are the first ones that have realized the issues of using current ISAs to express data transformations in an explicitly parallel form. For this reason, Intel has already been granted a patent for ISA extensions to x86-64 which provide dedicated support for specific DT operations [isa16].

Intel Integrated Data Streaming Accelerator (DSA) is a recently released specification for an high-performance data copy and transformation accelerator that will be integrated in future Intel® processors, targeted for optimizing streaming data movement and transformation operations common with applications for high-performance storage, networking, persistent memory, and various data processing applications [Cor19]. The goal is to provide higher overall system performance for data mover and transformation operations, while freeing up CPU cycles for higher level functions.

UDP [FZEC17] applies the Finite-Automata - an emerging computational model that promises orders of magnitude better performance than CPUs in executing Finite State Machines (FSMs) model to a coarse-grained class of workloads such as data mining and CSV file parsing. Their architecture is targeted towards bulk loading and cleaning of batches of data.

4 Optimus Prime's Approach

4.1 Motivation

Going through the current state of data transformation frameworks, we reached to the conclusion that because networks and protocols have already taken great strides forward and indeed approaching electrical limitations of propagation and switching, Data Transformation is the logical next step to optimize. As it currently stands, Data Transformations are performed exclusively by software. That is, each transformation is expressed in long sequences of instructions in the CPU's ISA. Due to limitations of ISA's to express transformations, this confirms that Data Transformations are the bottleneck of inter-microservice RPC.

The first approach is, of course, to improve Data Transformation software, but this proves to be difficult. The paper [PGK+20] mentions that even in optimal scenarios of 5 instructions/output byte and perfect speculation control, it means *transforming a 300B message will take 700 nanoseconds*. Even if the fields of a message can be operated in parallel, synchronization costs limit any benefits from parallelization using threads.

Claim: This paper alleviates the Data Transformation problem by introducing two main innovations, the first being an in-memory schema for explicitly representing parallel transformation tasks, and the second one is architecting a dedicated hardware unit - a data transformation accelerator to unpack the schema and perform the tasks. It is also proposed that this accelerator can be used by any Data Transformation framework that generate that schema. So in a sense, it is designed as a general purpose accelerator, making it an ideal candidate for inclusion in future server chips.

4.2 The Current Bottleneck

The paper argues that performing Data Transformations on the CPU entails a high instruction count per serialized field and relies on implicit instruction-level rather than explicit field-level

parallelism. Additionally, the Data Transformation is so fine-grained that is unable to benefit from parallelization with software threads due to synchronization costs. To take the example of Protobuf, to serialize an object, each field is individually transformed based on the type. The output of each field contains a key (tag) acting as identifier for the field and the serialized bytes, ready for wire transfer. Unfortunately, neither software threads nor CPU ISAs are the right form to represent the parallelism between fields. In principle, each field in the object to be transformed could and should be independently transformed if the hardware is made aware of each field's type and location location. Designing an effective abstraction that explicitly represents this parallelism is key to accelerating Data Transformation. Serial instructions are the wrong abstraction to expose these types of independent operations, because the problem is inherently parallel.

Summary: Because CPU-centric Data Transformation continues to be bound by the limitations imposed by the ISA, the paper argues that accelerating Data Transformations requires both hardware and software to be co-designed around a new parallel abstraction that replaces the CPUs ISA.

4.3 Software Component: Schema Abstraction

The schema introduced in the paper solves the bottlenecks of expressing transformations in traditional ISA. In order to create a meaningful schema that explicitly expresses field-parallelism, the paper makes the following observation: **The transformation on each field is completely described by its type and the address of the input data. Therefore, a data structure containing these two info for each field is the leanest abstraction required to express all of a message's transformations** [PGK+20]**.**

This abstraction is called schema (Figure 2) , and holds the type and memory address of each field. This schema is generated by the app and passed to the accelerator to invoke a new transformation. The paper also argues that any Data Transformation framework (i.e Protobuf) can use the accelerator as long as their setter method is modified to create the schemata during the process of creating the message.

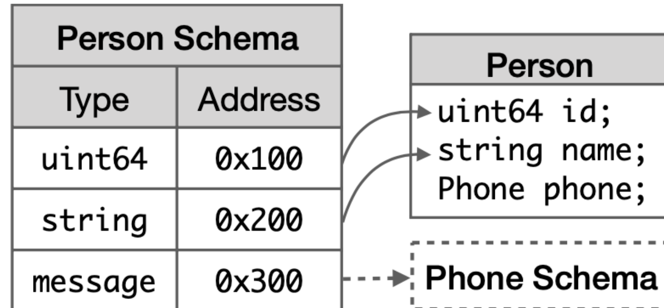

Figure 2: Example Schema

4.4 DTA General Architecture

The next step that this paper proposes is the actual hardware implementation that will unpack the abstraction schema and perform the transformation tasks in parallel. This hardware unit is called Optimus Prime and it follows the building blocks of a traditional data transformation accelerator. The main components of a data transformation accelerator (DTA) are:

1. **Dispatcher** - Responsible for interacting with the server's cores. It contains the accelerator's internal registers which are read/written by the cores when invoking new transformations.

2. **Converters** - Specialized hardware converters that can perform data transformations in a handful of cycles.

3. **Reader and writer** - Access data and stream it to and from the programmable converter.

These components are explained further on the next section, which describes the concrete implementation by Optimus Prime.

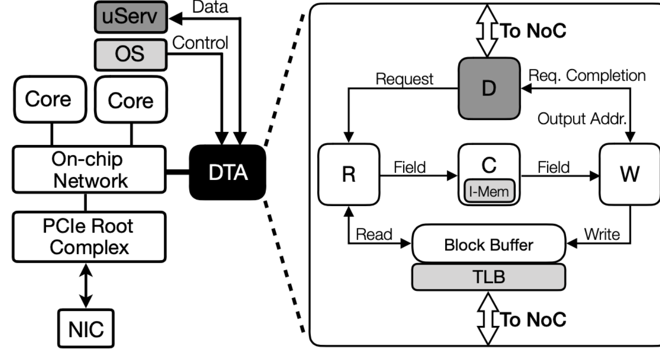

Figure 3: Architectural Overview of a DTA

Another trade-off that needs to be made is related to the physical location of the hardware unit. One option is to have a dedicated unit for each core, this would mean that it would share the core's L1 cache and TLB, eliminating the need for the Block Buffer. But the paper decides to go with the other approach, which is a shared unit for all cores and located physically at the NoC's edge. The reason for this being that if many cores were present in the server, the silicon cost would quickly add up and outweighing the benefits. The general architecture can be visualized on Figure 3.

4.5 Optimus Prime Implementation

Optimus Prime is a concrete DTA implementation which comprises of 4 main components and a buffer, as described below:

1. **Dispatcher** - The Dispatcher's main responsibility is receiving transformation requests from the cores and notifying the corresponding core upon completion. It contains a set of dedicated control registers accessed by CPU cores to request new transformations. Each request contains the following information: a pointer to the schema to be transformed, a pointer to output buffer where result is to be written, a pointer to serialized buffer and a valid bit. On new requests, the dispatcher controller passes the schema to the reader and the output pointer to the writer. When the transformation is complete, the valid bit is cleared. It is worth noting that transformations are synchnronous - that is, a core waits for a request completion before issuing another.

2. **Reader** - The reader gets the schema pointer from the dispatcher and issues a memory request for that address to the block buffer. The block buffer returns a cache line containing schema fields, which the reader proceeds store in a dedicated field buffer. Then, the reader fetches the fields one-by-one from the field buffer, extracts the data pointer and issues a read request to the block buffer. The block buffer returns a cache line containing the field's raw data, which the reader stores in the Data Buffer. The reader then extracts required data from the data buffer based on field's type and forwards to the converter to carry the transformation. It also calculates the offset where the Writer should place the transformed data.

3. **Converter** - The converter takes chunks from the reader and performs the actual data transformation. The chunk contains information that identifies the field's type and therefore what operation to perform. After data is transformed, the converter passes the converted bytes into the writer to be written in output buffer. The field type indicates the entry in the instruction memory that the converter executes. For common data types, a single instruction usually is enough to perform the conversion.

4. **Writer** - The writer receives transformed data from the converter and writes to the appropriate location in the output buffer, which is identified by a base-offset pair. Base is supplied by the requested core and passed by the Dispatcher, while the offset is calculated by the reader and passed to the writer. On write completed, it notifies the dispatcher.

The whole process described above can also be visualized in the microarchitecture Figure 4

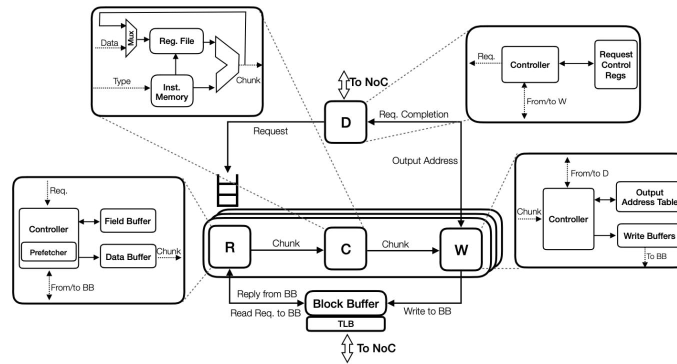

Figure 4: Microarchitecture of Optimus Prime

4.6 Optimizations

Optimus Prime is architected as a decoupled access-execute pipeline and includes a single Reader, Converter and Writer. When having the field buffer, the reader can do prefetching on the data buffer with 100% accuracy. This is a very helpful optimization because before any converter can begin transforming data, a reader must perform the memory accesses. But, even with prefetching, the pipeline spends the majority of it's cycles waiting for memory access. In order to increase utilization, the pipeline can be time-shared among multiple requests. This requires keeping multiple request contexts per Reader which can be rotated in one cycle. So these two optimization techniques can be used to hide memory latency.

5 Evaluation

The evaluation focuses on the ability of Optimus Prime's to transform data at modern NICs potential bandwidth level. The experiments that authors undertake include a single-pipeline, multi-pipeline, time-sharing optimization scenario and also microservices scenario. Here, we take a look at the first two experiments.

Note: All experiments and images discussed below are taken from [PGK+20].

5.1 Evaluation Basis

Three types of messages were used to test the transformation:

- **Flat Message** - Every field of the message is a primitive type (i.e string , integer)

- **Mixed Message** - Consists of some primitive types and some reference types (fields that point to another object).
- **Nested Message** - Every field of the message a reference type.

OP(n, m): n denotes the number of parallel pipelines, and m denotes the time-sharing degree, meaning how many pipelines are time-shared.

5.2 Single Pipeline Results

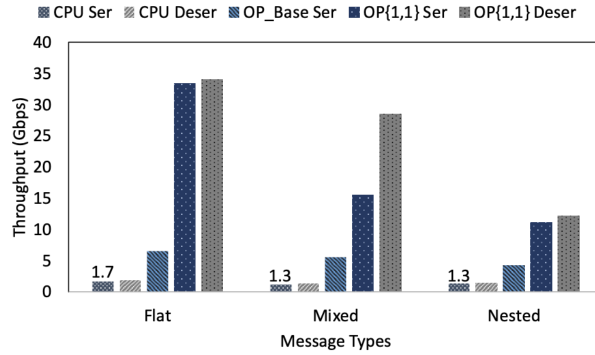

Figure 5: Results of OP(1,1)

As seen in Figure 5, a non-optimized single-pipeline OP(1,1) denoted as *OP_Base Ser* is 5x faster than baseline CPU without prefetching and time-sharing, and has another 2-4x improvement with prefetching and time-sharing (denoted as *OP 1,1 Ser/Deser*). The bottleneck in OP(1,1) is the serial processing of messages by a single transformation pipeline. Given that messages are naturally independent of each other, next configurations with multiple pipelines are evaluated. Also worth noting that flat messages perform much better than nested ones, as expected.

5.3 Multi Pipeline Results

Even though OP(1,1) reaches an almost 20x better throughput than a single core, there is still some room left to attain the 40Gbps sustainable by modern NICs. The next step is to test transformation pipelines which operate in parallel.

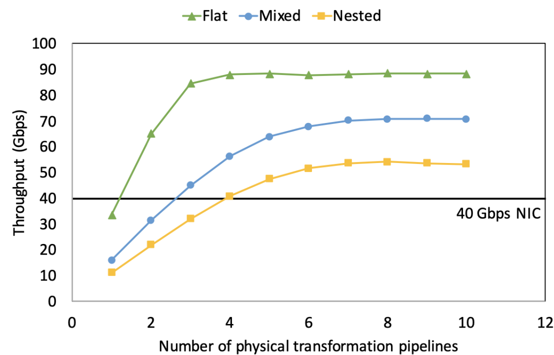

Figure 6: Results of OP(n,1)

As seen on FIgure 6, the 40Gbps can be easily achieved by using 2 parallel pipelines in flat messages, 3 in mixed messages and 4 in nested. The throughput plateaus beyond a certain pipeline number because they exhaust the available NoC bandwidth in aggregate.

In a microservices scenario, these improvements lead to a reduced Data Transformation latency by up to 10× and service latency by up to 30% compared to the CPU baseline.

6 Discussion

This paper discusses the topic of data transformation for cross-microservice communication. We started by exploring popular choices for data transformation like REST with JSON-string based serialization and gRPC with Protobuf-binary based serialization.

After taking a closer look at these approaches, it was understood that even though good progress has been made in the data transformation realm, with the advances of networking protocols, the speed of inter-service communication is still nowhere near the potential of 40Gbps that NICs offer.

The next step was to explore a new and innovative approach to Data Transformation, namely Optimus Prime. This paper argued that the current bottleneck with Data Transformation comes from the current implementations of transforming the data in the CPU, where both instruction-level parallelism and thread-level parallelism are not of much help. Optimus Prime "re-invented" the transformation process by expressing the parallelism explicitly via an in-memory schema and a hardware unit to unpack and perform transformations.

This paper does a good job at covering at a high level the need for data transformation, the status quo of popular Data Transformation frameworks and how a new perspective like Optimus Prime breaks the barriers of Data Transformation and leads to arriving at the physical barriers of modern NICs.

On the other hand, this paper does not go into much low-level details on how exactly Protobuf serializes the message fields based on type or how Optimus Prime implements the Conversion step of the pipeline to achieve single-instruction data transformations for most types. The reason for this is twofold. Firstly, this would lead us to a very low-level description of the process, mostly dealing with bit manipulation operators and hardware knowledge. Going into that much detail, would require a much deeper knowledge of computer architecture and OS, something that is out of scope for this paper. The second reason is that even the original Optimus Prime paper tends to jump past the Converter steps quickly and not get into much detail.

While the original paper did a wonderful job at painting the full picture of the Data Transformation issue, one potential point for improvement is not skipping ahead in a few sections i.e: The Converter component. If explained in more detail, it would provide the reader a much more enriching experience overall.

7 Conclusion

Data transformation is the process of converting messages to the appropriate format in inter-microservice communication. Several types of data transformation frameworks exist, from string-based serialization like JSON/XML to binary-based serialization protocols like Protobuf which offer a better performance overall. But with improvements in network technology and protocol processing, data transformation still causes a significant portion of end-to-end communication latency. This paper looks into an innovative approach, namely Optimus Prime, that introduces a new abstraction schema and the accompanying hardware accelerator to transform data at network line-rate. Optimus Prime achieves 60× higher throughput than traditional CPU cores and shortens the service latency of evaluated microservices by up to 30%.

References

[AGM+10] Mohammad Alizadeh, Albert G. Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *In Proceedings of the ACM SIGCOMM 2010 Conference. 63–74.*, 2010. 1

[Cor19] Intel Corp. Intel data streaming accelerator preliminary architecture specification. https://software.intel.com/en-us/download/intel-data-streaming-accelerator-preliminary-architecture-specification/, 2019. 1, 3.2

[dat20a] Data center network architectures. https://en.wikipedia.org/wiki/Data_center_network_architectures/, 2020. 2.1

[dat20b] What is data transformation: definition, benefits, and uses. https://www.stitchdata.com/resources/data-transformation/, 2020. 2.1

[eth18] The ethernet alliance. https://www.infinibandta.org/infiniband-roadmap/, 2018. 1

[FZEC17] Yuanwei Fang, Chen Zou, Aaron J. Elmore, and Andrew A. Chien. Udp: a programmable accelerator for extract-transform-load workloads and more. *In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). 55–68.*, 2017. 3.2

[inf18] Infiniband trade association. https://www.infinibandta.org/infiniband-roadmap/, 2018. 1

[isa16] Instruction set for variable length integer coding. https://patents.google.com/patent/US20180095760A1/en, 2016. 1, 3.2

[PGK+20] Arash Pourhabibi, Siddharth Gupta, Hussein Kassir, Mark Sutherland, Zilu Tian, Mario Paulo Drumond, Babak Falsafi, and Christoph Koch. Optimus prime - accelerating data transformation in servers. *ASPLOS*, 2020. 1, 4.1, 4.3, 5

[SM12] Audie Sumaray and Shamila Kami Makki. A comparison of data serialization formats for optimal efficiency on a mobile platform. *ICUIMC '12: Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, 2012. 3.1

[spe20] Speculative execution. https://en.wikipedia.org/wiki/Speculative_execution/, 2020. 2.2

[tim20] Time sharing. https://en.wikipedia.org/wiki/Time-sharing/, 2020. 2.2

[vir20] Virtual memory. https://en.wikipedia.org/wiki/Virtual_memory, 2020. 2.2