Building Distributed Computation Framework

Yibo Yan

I. Introduction

Distributed systems have constantly been evolving for many years by far. With the emergence of containers and Kubernetes, industries have been developing distributed software that runs in the form of microservice. Kubernetes provides a well-organized way to run microservices with scalability and fault-tolerant in mind, but it still requires a fair amount of expertise in managing and maintaining the cluster status manually. Meanwhile, dividing software into many pieces of small microservices creates a mental overhead and maintainability issue when it comes to making cross-binary changes, such as flag-gate new features in each binary of every microservice. Google has been developing a framework for writing distributed applications named Weaver, abstracting out the layer of manual management of microservices. It provides a runtime on which the distributed applications will run. Within the runtime, it defines the smallest unit of execution, which can be automatically scaled and managed by the runtime.

II. Review

Zhenyuan et al. discussed the design and implementation of Nu, which proposed a runtime framework supporting writing distributed applications in a single binary. [8] The major contribution made by this work is that it proposed a smaller unit logical process named "Proclet", enabling cheaper migrations between different machines. The Nu runtime makes building distributed applications with one single binary possible, and the user-level applications have the discretion to divide the software into different "Proclet". The runtime is designed to manage the "Proclet" and assign these units to processing units when it seems appropriate. Migration will be triggered if the runtime perceives the imbalanced computation. Due to the design nature of the Proclet, it can be easily and quickly migrated to other servers, which also indicates a good feasibility of service duplication.

In the industry, people usually build distributed services as a pack of microservices that communicates with each other. However, the ideology of microservices is tempting but hard to design right. For example, if microservices heavily rely on their own communication mechanism to chat with other microservice, then they can easily become a deeply coupled service which hard to debug and modify. Besides that, microservices can become hard to deploy and roll out updates due to the breaking changes that can involve multiple parts of the system. On the contrary, the monolithic design can be easier to debug and deploy as the compiler can perform the check for the whole system, and the interface remains clear to developers. Therefore, a distributed computation framework built in the manner of a monolithic design is desired to solve this problem.

III. Design Challenge

A. Migration Latency and Throughput

Migration is the key to this distributed computation framework. Therefore, authors borrow ideas from the "IX OS" to achieve low latency and high throughput even during the Proclet migration [1].

The authors also learned from the live migration of virtual machines [4] to further fine-tune the Proclet migration. Migration performance can be hurt by interference due to resource allocation and data dependency. Therefore, the authors utilized "Caladan" to further improve the migration performance [5].

B. Memory Allocation for Proclet

Memory allocation also plays an important role in the Nu system as it needs to provide an abstraction where each individual Proclet can have its own memory space but can be easily migrated and distributed across the machine boundary. The "Hoard" memory allocator provides some insights into solving this problem [2]. Memory allocators in the context of distributed systems also need to take advantage of many CPUs scenarios and arbitrary resources, and Jeff Bonwick et al. provide some insights in their work [3].

C. Scheduling

Scheduling is always an important topic in the task execution. The authors borrow some ideas from work by Key Ousterhout et al. [7]. The "Sparrow" is designed to prefer batch work in the context of distributed workload. It also provides fine-grained task scheduling, which is complementary to the functionality provided by the cluster resource managers. Sparrow prefers simplicity over over-sophisticated features, which trades for a better performance.

IV. Future Improvement and Possible Solution

A. Faster Memory Allocation for Proclet

The memory allocation for the individual Proclet can be simple enough just to use Arena allocation, which is a suitable choice for the design purpose of Proclet — being small and easy to migrate. Arena allocation can also be quick to allocate and deallocate by doing one chunk of stack reclamation. In this context, a fancier memory allocator might not be required to achieve the performance gain, considering the Proclet can be rapidly created and migrated across machines.

B. Local Stack and Heap Protection

As discussed above, the Arena allocation can be enough for the Proclet's design purpose. However, the runtime library will expose the entire memory space to the developers, which is undesired, considering such an approach loses the protection of the memory access. Fortunately, we might be able to borrow the power of the modern compiler. Rust can be a good choice in this context. The runtime library can be written in Rust and use a verifiable and modified version of the Rust compiler which will only allow the safe code block and reject all unsafe usage. In this case, the compiler can prevent the cross-memory-boundary check and reject any dangerous action upfront. However, the modification to the Rust compiler can be tricky to implement. Fortunately, Chinmay Kulkarni et al. already tried a similar approach when designing "Splinter" [6]. "Splinter" provides a rust-based runtime that can dynamically load safe Rust extension on the KV store and utilize such trusted Rust extension to run some calculations locally and avoid extra and unnecessary network round-trips.

References

[1] Adam Belay et al. "The IX Operating System: Combining Low Latency, High Throughput, and Efficiency in a Protected Dataplane". In: *ACM Trans. Comput. Syst.* 34.4 (Dec. 2016). ISSN: 0734-2071. DOI: 10.1145/2997641. URL: https://doi.org/10.1145/2997641.

[2] Emery D. Berger et al. "Hoard: A Scalable Memory Allocator for Multithreaded Applications". In: ASPLOS IX. Cambridge, Massachusetts, USA: Association for Computing Machinery, 2000, pp. 117–128. ISBN: 1581133170. DOI: 10.1145/378993.379232. URL: https://doi.org/10.1145/378993.379232.

[3] Jeff Bonwick and Jonathan Adams. "Magazines and Vmem: Extending the Slab Allocator to Many CPUs and Arbitrary Resources". In: *Proceedings of the General Track: 2001 USENIX Annual Technical Conference*. USA: USENIX Association, 2001, pp. 15–33. ISBN: 188044609X.

[4] Christopher Clark et al. "Live Migration of Virtual Machines". In: *2nd Symposium on Networked Systems Design & Implementation (NSDI 05)*. Boston, MA: USENIX Association, May 2005. URL: https://www.usenix.org/conference/nsdi-05/live-migration-virtual-machines.

[5] Joshua Fried et al. "Caladan: Mitigating Interference at Microsecond Timescales". In: *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*. OSDI'20. USA: USENIX Association, 2020. ISBN: 978-1-939133-19-9.

[6] Chinmay Kulkarni et al. "Splinter: Bare-Metal Extensions for Multi-Tenant Low-Latency Storage". In: *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*. OSDI'18. Carlsbad, CA, USA: USENIX Association, 2018, pp. 627–643. ISBN: 9781931971478.

[7] Kay Ousterhout et al. "Sparrow: Distributed, Low Latency Scheduling". In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP '13. Farminton, Pennsylvania: Association for Computing Machinery, 2013, pp. 69–84. ISBN: 9781450323888. DOI: 10.1145/2517349.2522716. URL: https://doi.org/10.1145/2517349.2522716.

[8] Zhenyuan Ruan et al. "Nu: Achieving Microsecond-Scale Resource Fungibility with Logical Processes". In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 1409–1427. ISBN: 978-1-939133-33-5. URL: https://www.usenix.org/conference/nsdi23/presentation/ruan.