Journal of Artificial Intelligence Practice (2023)
Clausius Scientific Press, Canada

DOI: 10.23977/jaip.2023.060605
ISSN 2371-8412 Vol. 6 Num. 6

DeeTune: Design and Application of an eBPF-based Network Framework for Baidu

Bo Li[1,a,*], Shiwei Ma[1,b]

[1]*Baidu APP Technology and Platform R&D Department, Baidu Inc, Beijing, China*
[a]*libo15@baidu.com*, [b]*mashiwei@baidu.com*
[*]*Corresponding author*

Keywords: eBPF, Performance, Profiling, Tracing, Cloud computing

Abstract: With the development of cloud computing and the continuous development of infrastructure, architecture upgrades and other technologies, Baidu's internal services are gradually moving to the cloud environment. Although the efficiency of services has increased significantly, some shortcomings and deficiencies of the basic capabilities of the cloud environment have gradually become apparent, resulting in the inability to meet some reasonable requirements of the enterprise, such as building the topology relationship between different microservices and conducting the test session for The traditional way of implementation is to record the real traffic to reflect and verify the function and so on. The traditional way of implementation is often to implant the code into the business system to make changes. However, given the diversity of business forms and technologies, the conventional way has a lot of problems in terms of business intervention, communication and coordination, performance, stability, and other aspects. In this paper, we introduce Baidu's eBPF-based network framework: DeeTune, which provides the ability to create service topology, record traffic, monitor non-intrusive metrics, etc., further improve the efficiency of SRE and quality assurance.

1. Background

The scale of Baidu's microservices is huge and growing, and the dependency relationships between services are also very complex. The service topology can show the global service and the calling relationship between services, and can also contain monitoring information to show the golden index between service links, so the service links are important for system observability, stability assurance and infrastructure construction.

Due to the high cost of manual maintenance, the traditional way based on SDK and framework has many problems, such as multiple technology stacks, business interventions, etc. The lack of a global service topology will result in the inability to meet some actual business requirements:

▪ Stability assurance: lack of a service and traffic topology that supports rapid positioning and accurate stop loss in the event of outages. When stability issues occur, the hope is that the service topology can quickly locate the problem service and server space, efficiently inform and notify dependent and dependent parties of the failed service, and assess the impact of failure analysis.

▪ Infrastructure development: there is a lack of a service topology to guide the relocation and reconstruction of server rooms. In the past, sorting and confirming services when relocating server rooms was an important but tedious task. Each server room relocation required a long time to sort services and their dependencies, which not only required manpower, but also required reserving machine resources, in addition to introducing stability risks.

▪ System reconfiguration and upgrade: lack of relationships between upstream and downstream services to evaluate the impact of system reconfiguration and upgrades on upstream and downstream services. How to accurately notify trusted and dependent parties during the upgrade process, and how to obtain the data to assess the complexity of services such as fan-in and fan-out during the refactoring process to assess whether services need to be split and merged, etc;

The huge scale of services, complex business types, and numerous technology stacks pose challenges for quality engineers, such as building integration test environments, writing test cases, and assessing the completeness of tests. Traffic Playback is one of the most advanced and mature solutions for automated testing that enables rapid code regression capability by recording traffic online and replaying it offline. It can significantly improve the efficiency of project iterations, accelerate the progress of code regression testing, and ensure the quality of enterprise development.

There is no single schema and tools for recording traffic, and the capabilities and implications of replaying traffic are severely limited:

▪ The enterprise technology stack is complex, and the supporting base libraries and frameworks are also numerous, making it impossible to perform unified traffic recording through frameworks or business transformation;

▪ Even a single technology stack and framework will face business interventions and consistency issues, and stability may also be affected to some extent

▪ There are many access paths between services, such as gateway access, virtual IP access, direct connection between services, etc., and it is not possible to record traffic through a unified data ingress and egress;

Metrics (e.g., traffic, time consumption, etc.) and tracking calls between services is an important foundation used in enterprises for stability issues and service performance optimization, but currently all observation solutions are intrusive and require redesign of the framework or the enterprise itself. On the other hand, in host and container monitoring, in addition to some static counters provided by the operating system, there is a need to collect and aggregate data from various data sources to support some deeper observation capabilities that help analyze and locate system problems. However, the development difficulties and resource consumption for this type of capability are very large.

2. Introduction to eBPF

The root cause of the above technical challenges is the diversity of business forms and technology packages selected for the enterprise. Horizontal, cross-business, and cross-technology stack requirements impact specification, business intervention, communication and coordination, performance, stability, and other aspects.

With the rapid development and application of eBPF technology in recent years, it can provide us with new solutions and ideas to solve the above problems[1]:

▪ eBPF is a kernel-related technology, which has nothing to do with the technology stack and the framework of the user state.

▪ In business non-intrusive way to get more kernel status and user status information, can to a great extent to provide us with help and even solve the problem.

eBPF full name: extended Berkeley Packet Filter[2], is the Linux kernel state introduced a set of general-purpose execution engine, which can trigger the Linux kernel based on the event to run custom code logic: eBPF, known as extended Berkeley Packet Filter, is a set of general-purpose execution engines introduced in the Linux kernel state that allow the execution of custom code logic based on event-driven triggers in the Linux kernel:

- eBPF provides a software-defined kernel approach that can be used to implement logic such as Linux dynamic tracing and Linux high-speed network packet processing;
- eBPF can insert specified hook code into the kernel without modifying the kernel source code or loading kernel modules, and can be executed when the kernel or application is running at a specified hook point (predefined hooks include system calls, function inputs and outputs, kernel tracepoints, network events, etc.);

3. Features of eBPF

The eBPF technology has the following features:

- Safe and stable: by strictly limiting access to function sets, memory addresses, loop counts, and code path triggers, the kernel has a built-in stable API that ensures that only eBPF instructions verified as safe are executed by the kernel;
- Efficient: eBPF instructions continue to execute in the kernel without copying data to userland. Using the Just-In-Time (JIT) compiler, which converts bytecode to machine code, execution efficiency is equivalent to that of the kernel and execution is more efficient;
- Hot loading (continuous deployment): eBPF programs are loaded and unloaded without rebooting the Linux system;
- Data interoperability: maps enable interoperability of user and kernel state data;
- Compatibility: eBPF provides a stable API that can run on old kernels, then it must continue to run on new kernels;

eBPF technology can provide new ideas and solutions in security, tracking and performance analysis, networking, observation and monitoring, etc:

- Security: security testing can be done from the system call, packet, and socket layers, e.g., writing firewall programs, developing a DDOS protection system, etc...[3]
- Tracing & Performance Analysis: the kernel provides many types of probes (probe points), such as kernel probes, perf events, tracepoints, user space probes, XDP, etc., and eBPF programs can be written to collect the information from these probes and in this way trace the program and analyze program performance. to analyze program performance[4];
- Networking: powerful packet processing programs can be developed at the kernel level, such as Cilium, to provide load balancing at the kernel level, to bring the service mesh to a deeper level, and to solve the performance problem of Sidecar[5];
- Observation and monitoring: continuous observation and monitoring of these test points can enrich the scope and depth of the metrics data. And more importantly, this work can be done without changing the premise of the established procedures[6].

4. Best Practices

- Facebook: Katran open source load balancer, L4LB, DDoS, tracing
- Netflix: BPF frequent users, e.g. production environment tracing, profiling
- Google: Android, server security, observability and more, GKE uses Cilium as network foundation by default
- Apple: uses Falcon to detect security risks
- AWS: using eBPF as RPC observability tool, etc.
- Alibaba: extensions to Terway, a container network plugin, extensions to ilogtail, an observation tool.
- Bpftrace: Provides a quick way to implement dynamic tracing with eBPF, and can be used as a

simple command line tool or as a programming tool for beginners[7];

▪ BCC: BCC is a Python wrapper around the eBPF peripheral toolset for creating efficient kernel tracing and manipulation routines that can greatly improve BPF program development[8];

▪ Cilium: Cilium is an eBPF-based networking, observability and security solution. It can completely replace kube-proxy and provides deep network and security visibility and monitoring[9];

▪ DeepFlow: is a highly automated observability platform provided by a Chinese company as open source, using new technologies such as eBPF, WASM, OpenTelemetry, etc., largely avoiding the insertion of hidden code[10];

▪ Coroot: is an open source observability tool based on eBPF that converts collected data into visual and actionable metrics to quickly identify and resolve application problems; modules that can be executed when the kernel or application is running at a specific hook point (predefined hooks include system calls, function inputs and outputs, kernel tracepoints, network events, etc.)[11];

5. Method And Application

5.1. Method

Although eBPF technology can better solve the problems of technology stack dependence and business intervention, there are still many difficulties to be solved in Baidu's complex environment. The environment in which Baidu deploys its microservices uses multiple PaaS platforms, multiple container types, multiple kernel versions, and multiple CPU architectures. These are issues that need to be considered when landing the eBPF agent. At the same time, the question of how to efficiently implement the multiple tracepoints in kernel state and support logic in user state has further increased the complexity and difficulty of system implementation.

By exploring various technologies and combining them with Baidu's actual business requirements, we developed and implemented a series of eBPF-based network frameworks to meet Baidu's requirements and scenarios: DeeTune, which consists of five subsystems, as shown in Figure 1:

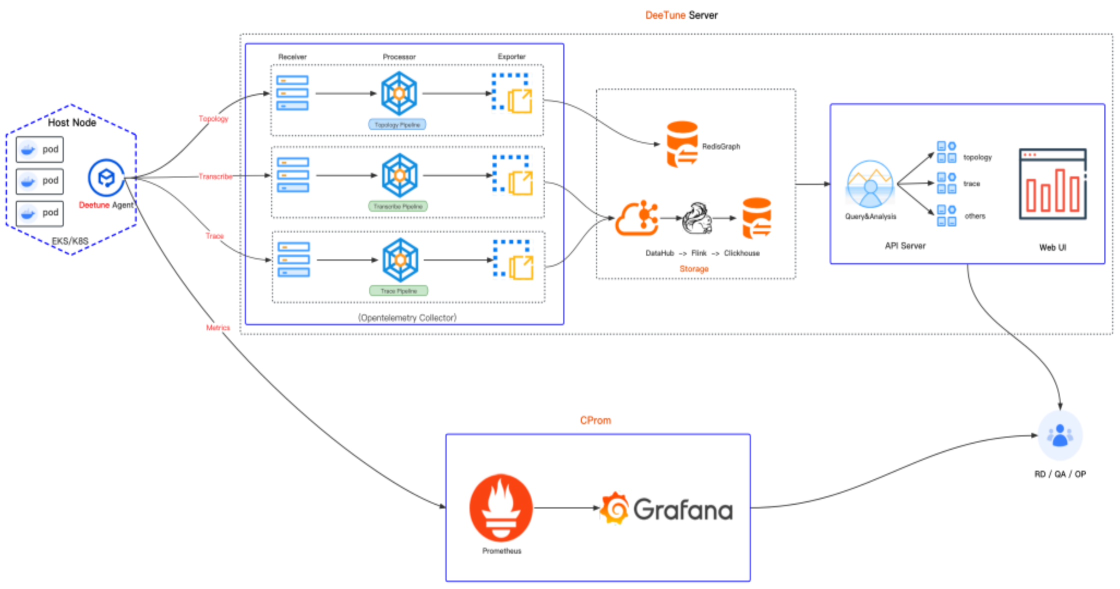

Figure 1: DeeTune System Architecture.

▪ Agent: deployed on the physical machine in the form of a host agent. It loads and executes the eBPF program, listens for the creation and destruction of processes in the kernel, the establishment and closure of TCP connections, the reading and writing of sockets, etc., and processes the corresponding events in the user state, obtaining data such as topology, metrics, and trace.

▪ Server: an independently deployed Otel Collector service that receives observation data reported by the agent, analyzes and processes it, and then stores it in the appropriate memory for later retrieval and analysis.

▪ Memory: used to store topology, record traffic and trace system data, providing different types of memory for different types of data to ensure efficient storage and query analysis.

▪ CProm: Baidu's internal Prometheus and Grafana integration services, providing services for querying large data sets via standard interfaces.

▪ API and Web UI: Provide OpenAPI and web visualization access to enable users in different roles to access and use the platform's data and functionality in appropriate ways.

The service information is the most important basic data of the platform. All functions (such as topology, monitoring, traffic recording) depend heavily on the service information, which is mainly obtained and analyzed by the agent, including the service name, business line, IDC, BNS (Baidu Naming Service), etc. The information can be obtained from three sources: Naming Service, container runtime information, and container daemon. There are three sources of information: Naming Service, Container Runtime Information, and Container Daemon.

Baidu's complex internal infrastructure environment makes obtaining service information complex and difficult:

▪ Baidu has complex internal business scenarios, and different PaaS platforms are able to customize the best online and deployment methods for each business scenario.

▪ Different PaaS platforms have different definitions of service information, registration content, and formats in the naming service, so the platform agent needs to do a lot of compatibility work to get the current service and service information of the invoked service.

▪ Baidu mainly uses three types of containers internally: matrix (a container type developed by Baidu), container, and Docker.

▪ When PaaS deploys service instances, it writes some service information into container daemons such as containererd, dockererd, etc., and different PaaS platforms may use more than one container type, which makes it even more difficult for the agent to obtain service information.

The solution to the above problems mainly relies on the agent's compatibility and processing of different scenarios, and where possible, communication and collaboration with different PaaSs to inject service information in a unified and standardized way.

Baidu's internal CPU has the ARM A64 architecture in addition to the X86 architecture. Therefore, we need to analyze whether each of the above tracepoints can be executed on both CPU architectures, e.g., we need to perform compatibility processing for the sys_enter_open/sys_exit_open tracepoint on ARM CPUs;

Agent requires certain host resources to run, so too much resource usage will affect the system:

▪ One part of the agent system is internal CPU -intensive computational logic to process kernel events and eventually output data such as topology, metrics, trace, etc;

▪ The other part is memory-intensive logic used to store intermediate and run-time computation data and provide data support for the computation logic;

In the production environment, the kernel can generate several K~hundred K events per second. Therefore, the agent must have an efficient event processing function and an efficient data access structure that reduces the use of locks and conflicts, etc. After several performance optimizations, the DeeTune agent can now control the CPU /MEM in the production environment on 1.3Core/1G.

On the other hand, due to the existence of the eBPF tracepoint, all network requests on the machine are processed by the kernel's eBPF program, so eBPF also has an impact on network latency for calls between services. After testing in a real environment, it shows that triggering events in the tracepoint, adding, deleting, modifying and checking the eBPF map, calling the bfp helper function and judging the unpacking of Layer 7 logs takes about 30 µs, which is negligible for most services.

5.2. Application

5.2.1. Service Topology

The eBPF-based service topology capability can provide highly accurate and complete service topology data that can support efficient fault localization and analysis, strong and weak service dependencies, fault rehearsal, service impact area analysis, and cross-computer space invocation. In addition to visual topology diagrams, the service topology also provides an OpenAPI for the demand side to develop and customize its own functions.

5.2.2. Traffic Recording

Traffic recording is a non-static requirement, and the agent must be able to perform the logging task dynamically according to the recording policy based on the configuration. Traffic recording can be created as a task on the platform and can specify the recording of traffic between any two services that have a connection relationship based on the topology information. At the same time, you can specify the recording policies, such as the recording time, the number of entries, the interface to record, etc. QA via the callback when the task is registered, or via OpenAPI to get the task and recorded traffic. As shown in the figure 2:

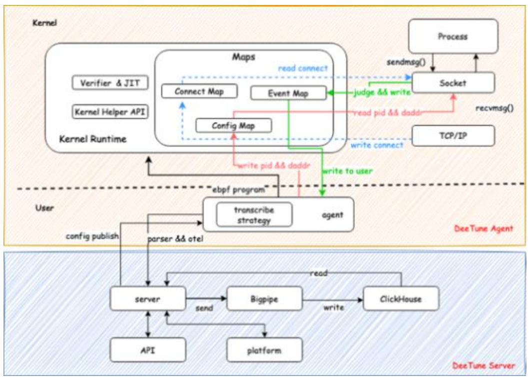

Figure 2: DeeTune Traffic Recording Architecture.

5.2.3. Monitoring metrics

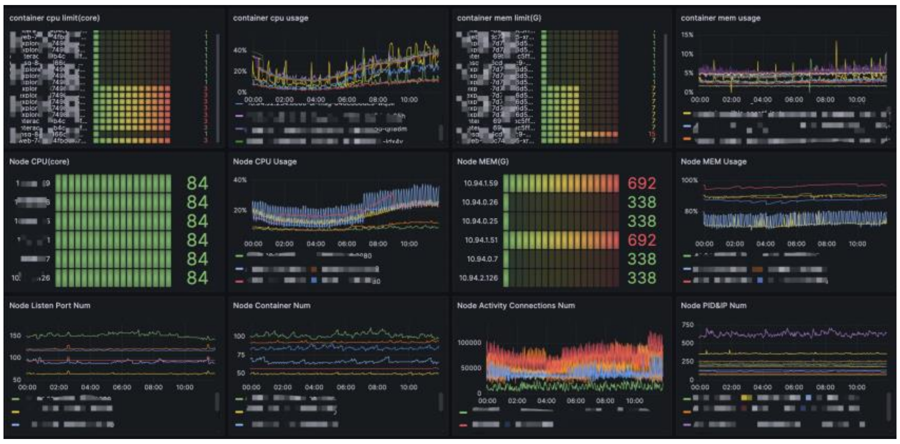

Figure 3: DeeTune Monitoring Metrics Panel.

By default, DeeTune collects and aggregates resource metrics across hosts and containers, such as

CPU, MEM and so on. In addition to basic resource metrics, with eBPF technology we can theoretically collect almost all metrics in the system, such as active and failed connections, network retries, traffic statistics, process and container counts, kernel key metrics, and so on. By more closely monitoring the system and containers, operations and maintenance engineers can more efficiently pinpoint resource-related issues: PaaS platforms, for example, are prone to irrational resource usage, high container deployment density, heavy overselling of resources, network health diagnostics, and so on. As shown in the figure 3:

6. Conclusion

DeeTune is an eBPF-based networking framework implemented by Baidu that provides basic functionality for cloud-native scenarios such as service topology, traffic recording, opentracing, container resource and business traffic monitoring, and cross-computer space call detection. Some of the features have already been deployed for pilot operation and have achieved initial success. This paper analyzes the real-world problems and provides a comprehensive introduction to the design methodology and application of DeeTune with respect to the technical benefits and applications of eBPF.

In the future, it will be further optimized and extended in some aspects:

▪ Multi-protocol support: the current link metrics and traffic logging mainly support HTTP1, Redis and MySQL protocols, and in the future, gRPC, bRPC, HTTP2 and other internal homegrown protocols will also be supported

▪ Multisystem linking: linking information with the deployment desk, monitoring system, code and continuous integration system, and quality and efficiency system can help the company find and solve online problems and code issues more efficiently.

Acknowledgements

This work is supported by Baidu Inc. We sincerely thank the reviewers for their insightful comments and suggestions to improve the quality of the paper.

References

[1] Miano S, Bertrone M, Risso F, et al. Creating complex network services with ebpf: Experience and lessons learned[C]//2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR). IEEE, 2018: 1-8.

[2] Kehoe M. eBPF: The next power tool of SRE's [J]. USENIX Association, 2022.

[3] Deri L, Sabella S, Mainardi S, et al. Combining System Visibility and Security Using eBPF[C]//ITASEC. 2019.

[4] Cassagnes C, Trestioreanu L, Joly C, et al. The rise of eBPF for non-intrusive performance monitoring[C]//NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2020: 1-7.

[5] Sedghpour M R S, Townend P. Service mesh and ebpf-powered microservices: A survey and future directions [C]//2022 IEEE International Conference on Service-Oriented System Engineering (SOSE). IEEE, 2022: 176-184.

[6] Liu C, Cai Z, Wang B, et al. A protocol-independent container network observability analysis system based on eBPF[C] //2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2020: 697-702.

[7] Kwak J H. A study on how to build a supercomputer monitoring and performance analysis system based on Performance Co-Pilot, Bpftrace and Grafana[C]//Proceedings of the Korea Information Processing Society Conference. Korea Information Processing Society, 2021: 118-121.

[8] Gregg B. Linux performance [J]. Online]. http://www. brendangregg. com/linuxperf. html, 2018.

[9] Qi S, Kulkarni S G, Ramakrishnan K K. Understanding container network interface plugins: design considerations and performance [C]//2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN. IEEE, 2020: 1-6.

[10] Zhang QX, Wu YF, Yang Y, et al. Survey on Service Dependency Discovery Technology for Microservice Systems[J]. Journal of Software, 2023: 1-18.

[11] https://github.com/coroot/coroot