Microservice Architecture of Meta

Abhishek Tiwari

Citation: *A. Tiwari*, ”Microservice Architecture of Meta”, Abhishek Tiwari, 2024. doi:10.59350/7x9hc-t2q45

Published on: September 08, 2024

Microservices have become the dominant architectural paradigm for building large-scale distributed systems, but until now, their inner workings at major tech companies have remained shrouded in mystery. A recent new paper by Huye et. al - researchers at Tufts University and Meta - provides an unprecedented look under the hood at Meta's massive microservices architecture (see [1]).

Scale is mind-boggling

Let's start with the sheer scale, which is frankly staggering. Meta's microservices architecture encompasses over 18,500 active services running across more than 12 million service instances. Just let that sink in for a moment - 12 million instances! This is orders of magnitude beyond what most organizations are dealing with. What's particularly interesting is that this massive scale isn't primarily driven by replication of existing services. Rather, the number of unique services has been steadily growing, doubling over the 22-month period studied. Just to contrast, based on publicly available information Netflix has about 1000 active microservices. This suggests an ever-expanding set of business functionalities being built out as microservices. The topology is also highly dynamic, with services being created and deprecated on a daily basis. On average, hundreds of new services are added each day while a similar number are deprecated. This speaks to the rapid pace of development and evolution happening continuously.

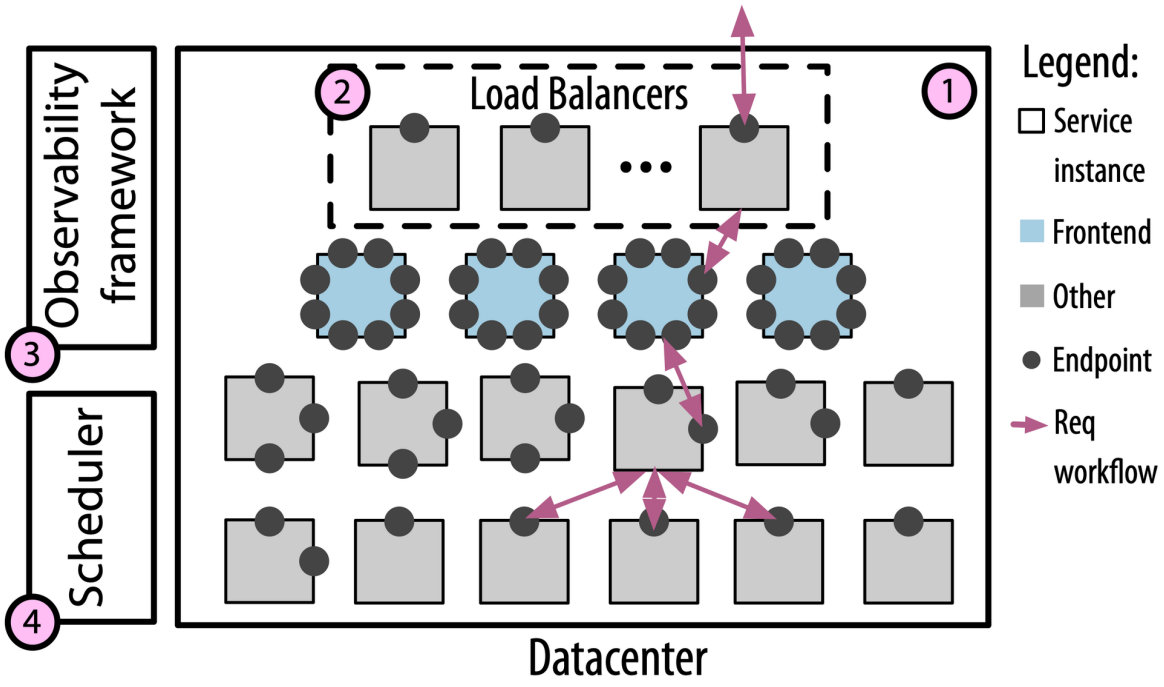

Figure 1: Meta's microservice architecture consists of: (1) a topology of interconnected, replicated software services running in dozens of data centers; (2) load balancers for distributing re- quests amongst service replicas; (3) an observability framework for monitoring the topology and creating traces (graphs) of a sampled set of request workflows; and (4) a globally-federated scheduler for running services on host machines within containers.

Heterogeneity reigns supreme

A key finding that stood out to me is just how heterogeneous Meta's microservices landscape is. The researchers identified three distinct types of software entities deployed as services: traditional microservices representing a single, well-scoped business use case; more monolithic services that serve multiple business cases but are deployed as a single service; and "ill-fitting" entities that don't cleanly align with traditional microservices definition. This last category was particularly eye-opening. It turns out a significant portion of Meta's services (over 60% on some days!) are generated by platforms like their ML inference system that create separate services for each tenant. This allows independent scaling and deployment, but means these platforms are essentially using microservices as a form of multi-tenancy. The paper argues these ill-fitting entities highlight areas where the microservices paradigm falls short, requiring custom solutions for scheduling, scaling, and observability. The per-tenant deployment model used by the Inference Platform suggests that isolation and independent scaling of ML models is a priority, which doesn't fit neatly into the standard microservices paradigm.

Request workflow dynamics

The researchers conducted a deep dive into the characteristics of request workflows (aka dependency graph or call graph) flowing through Meta's microservices. Some of the key findings here really challenged my assumptions. Request workflows are highly dynamic and unpredictable. The same root endpoint can result in dramatically different execution patterns. Traces tend to be wide rather than deep, indicating common use of data sharding. Services often fan out to call many other services in parallel rather than long chains of sequential calls. There's also a high degree of service reuse within traces. The same services are often called multiple times at different depths.

Interestingly, while the overall behavior is quite variable, there are some local patterns that emerge. Parent services tend to have a dominant set of child services they call in over 50% of executions. And the combination of parent service ID and child service set is somewhat predictive of execution concurrency. These insights have major implications for how we model and reason about microservices behavior. Tools that assume deterministic or predictable workflows are likely to fall short in real-world environments like Meta's.

These insights have major implications for how we model and reason about microservices behavior. Tools that assume deterministic or predictable workflows are likely to fall short in real-world environments like Meta's.

Sparsely interconnected

Despite having over 18,500 active services, the researchers found only about 393,622 edges that connect services. This is far fewer than what we might expect in a fully connected topology, which could potentially have over 342 million edges. The overall topology of connected services does not exhibit a power-law relationship typical of many large-scale networks. This sparsity suggests that most services have a relatively limited and specific role within the overall system, interacting with only a small subset of other services.

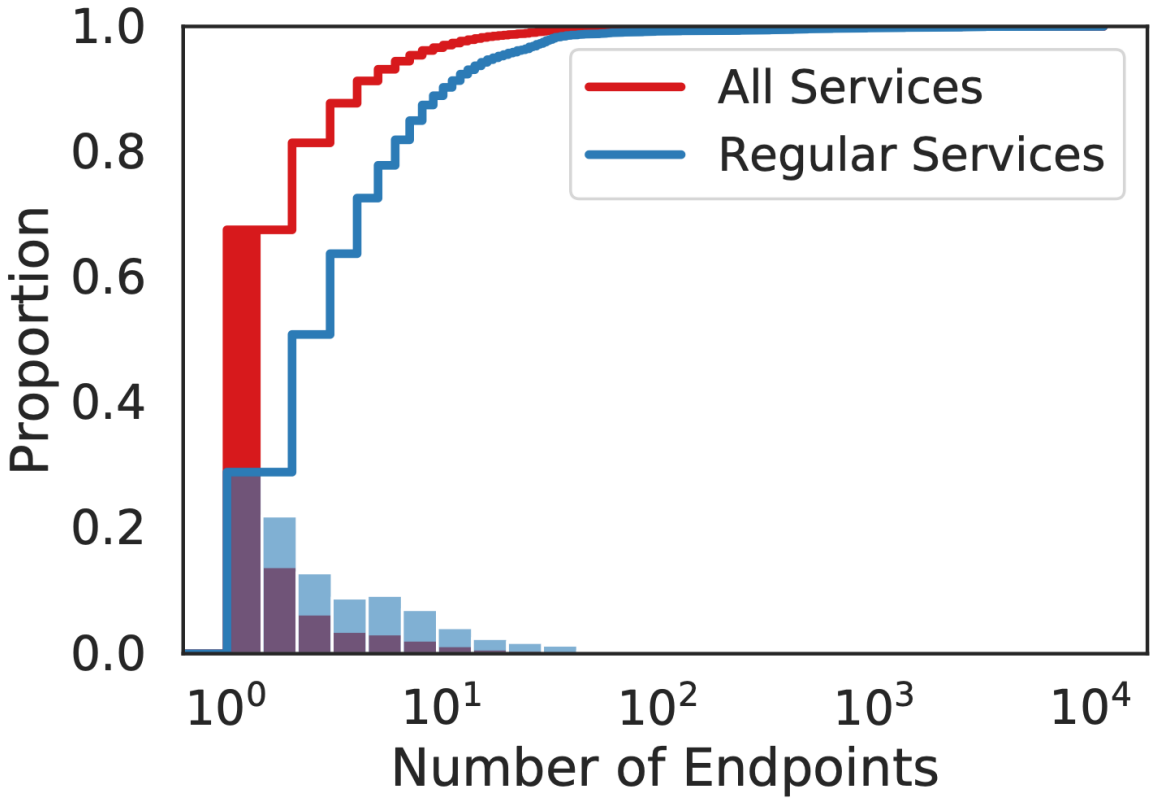

Figure 2: Service complexity measured by number of unique endpoints in a service shows a power-law distribution

Such a structure likely enhances modularity and reduces complexity in some ways, as changes to one service are less likely to have wide-ranging ripple effects across the entire system. However, it also raises interesting questions about how data and functionality flow through the system. Are there key hub services that act as central points of coordination? How does this sparse interconnection impact overall system resilience and performance? This characteristic of Meta's architecture challenges some common assumptions about microservices and highlights the need for specialized tools and approaches for managing and visualizing such sparsely connected large-scale systems. It's a reminder that as we scale microservices to extreme levels, the resulting architectures may look quite different from what we typically envision in smaller-scale deployments.

Observability remains a challenge

One of the more sobering findings relates to observability. Despite Meta's sophisticated distributed tracing infrastructure, a significant portion of traces suffer from incomplete visibility. Many execution

branches are prematurely terminated due to sampling, dropped records, rate limiting, or uninstrumented services.

This “observability loss” tends to disproportionately affect deeper branches in the call graph. For some profiles, up to 80% of calls beyond a certain depth were unobservable. This presents major challenges for accurately understanding end-to-end request behavior and diagnosing issues. It’s a stark reminder that even with best-in-class tooling, achieving comprehensive observability in large-scale microservices remains an unsolved problem.

Implications and open questions

The paper concludes with a thoughtful discussion of the implications of these findings and opportunities for future research. A few key points resonated with me. We need much better artificial topology and workflow generators for microservices research. Current testbeds and synthetic workloads fail to capture the heterogeneity, dynamism, and complexity seen in real-world deployments like Meta’s. Microservices tooling needs to evolve to handle the unpredictability and variability observed. Approaches that rely on static topologies or deterministic workflows are likely to break down at scale. There’s a need to rethink how we incorporate “ill-fitting” entities into microservices architectures. Should infrastructure platforms provide richer interfaces to allow scheduling and observability across multiple dimensions? We also need standardized methods to characterize and compare microservices deployments across organizations. The lack of common definitions and metrics makes it challenging to conduct comparative analyses.

Emergence of “ill-fitting” entities

The emergence of “ill-fitting” entities that don’t cleanly map to microservices paradigm is particularly telling. It’s likely that this is a unique characteristics of ML workloads - such as resource-intensive training, need for specialised hardware (like GPUs), frequent redeployment, and multi-tenancy requirements - are pushing the boundaries of what traditional microservices architectures were designed to handle.

It’s worth noting that this pattern might be particularly pronounced at Meta due to their heavy investment in AI across their products. Other companies with different focuses might see different patterns in their ill-fitting services. Nevertheless, as AI continues to grow in importance across the tech industry, it’s likely that many companies will face similar challenges in integrating ML workloads into their microservices architectures.

Conclusion

This paper opens up a whole host of intriguing questions that I hope researchers and practitioners will dig into. How can we design microservices architectures that gracefully accommodate heterogeneous entities and multi-dimensional scaling requirements? What new approaches to observability can help us achieve more complete visibility into request workflows, especially for deep call chains? How can we build more realistic microservices testbeds and synthetic workloads that capture the complexity seen in production environments? Are there ways to make microservices topologies more predictable and stable without sacrificing agility? Or do we need new programming models altogether for mega-scale distributed systems? How does Meta's microservices architecture compare to other tech giants like Google, Amazon, Netflix, or Microsoft? Are there common patterns and challenges that emerge at that scale?

References

[1] D. Huye, Y. Shkuro, and R. R. Sambasivan, "Lifting the veil on Meta's microservice architecture," 2023, *USENIX Association*. Available: https://www.usenix.org/conference/atc23/presentation/huye