# Microservice Architecture of Alibaba

Abhishek Tiwari

Citation: *A. Tiwari*, ”Microservice Architecture of Alibaba”, Abhishek Tiwari, 2024. doi:10.59350/4fttv-pt832

Published on: December 22, 2024

Alibaba has built a complex system of microservices to support its large user base and manage its diverse business operations. This article explores key learning from Alibaba's microservices architecture, presenting critical observations from its design, scalability, performance optimisation, and resource allocation model. The article draws from research published by Luo et al., which characterises the structural properties of call graphs within Alibaba's large-scale microservices ecosystem, uncovering several graph-based properties and insights (see [1] and [2]).

## Hyperscale

Alibaba's microservices ecosystem operates at an unprecedented scale. The Alibaba System Infrastructure (ASI) supports over 10 million cores across 50 large-scale clusters, hosting approximately 100,000 business pods. These clusters serve diverse workloads, from real-time streaming and e-commerce to complex machine-learning tasks. During the Double 11 Shopping Festival in 2020, Alibaba's system managed a peak of 580,000 transactions per second, a 1,400% increase compared to ten years earlier. This scale shows how strong and flexible Alibaba's microservices model is. These systems can handle changes in demand throughout the day and during special events. For instance, backend services can see up to ten times more queries during busy times compared to quieter periods.

## Anatomy of Microservice Call Graphs

A call graph represents the interactions and dependencies between different microservices within a system (see [3]). It maps the sequence of calls one service makes to others, capturing upstream and downstream interactions. This structure allows developers and system architects to analyse the data flow, discover performance bottlenecks, and optimise resource allocation.

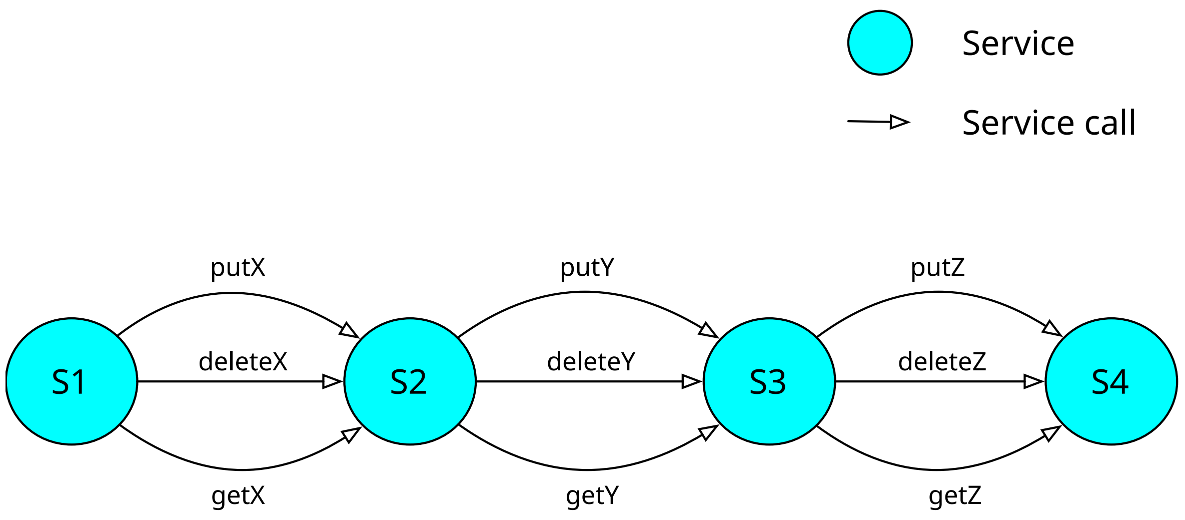

**Figure 1:** An example of service call graph

The following illustrates how Alibaba classifies its microservices into stateless and stateful categories. Stateless services operate independently of state data, while stateful services, like databases and Memcached, store and manage data. Stateful services usually offer a few uniform interfaces, such as reading or writing data, whereas stateless services provide numerous evolving interfaces. Communication between microservices follows three paradigms: inter-process communication (IPC), remote invocation, and indirect communication. IPC typically occurs between stateless and stateful services. Remote invocation, such as Remote Procedure Call (RPC), involves two-way communication where downstream services return results to upstream services. Indirect communication, like Message Queue (MQ), enables one-way messaging, where a third entity stores messages for reliability, and downstream services retrieve them as needed without a reply. While remote invocation offers high efficiency, indirect communication provides greater flexibility for managing asynchronous tasks.

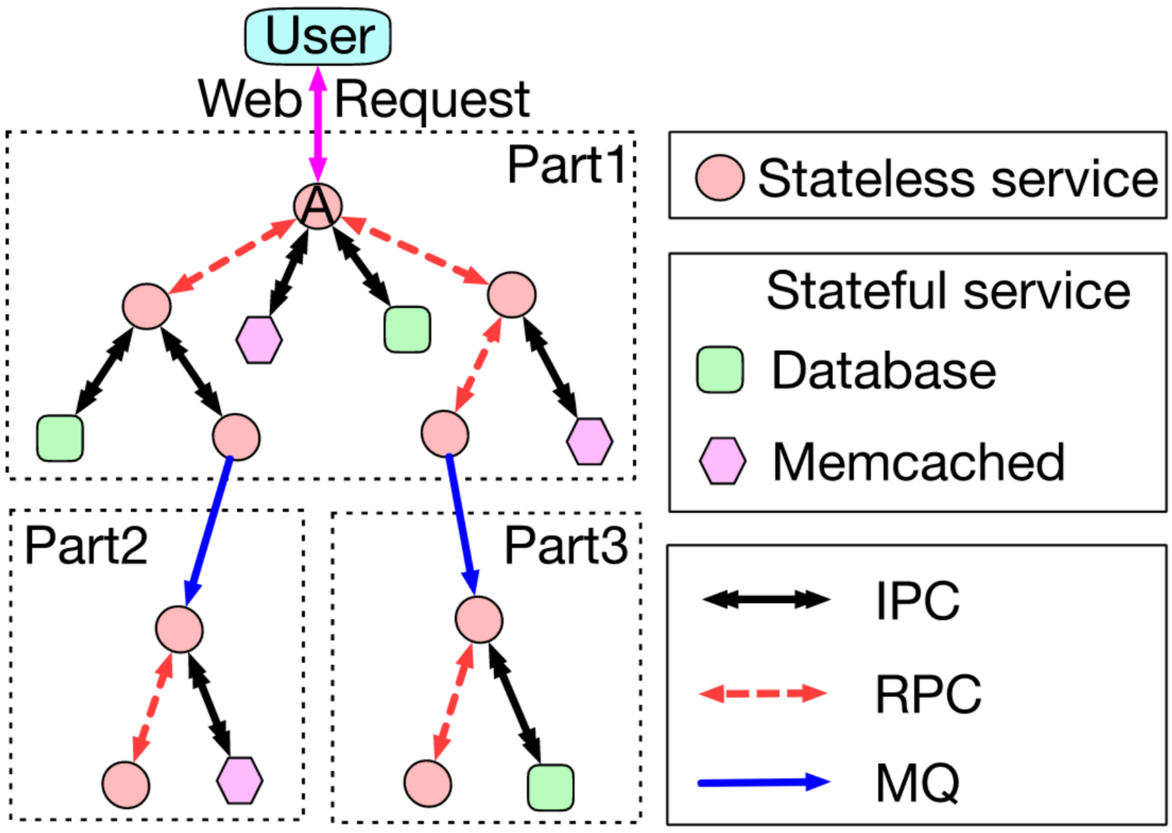

**Figure 2:** Call graph components of Alibaba's microservices ecosystem. Image credits Luo et al.

In the context of Alibaba's ecosystem, call graphs provide crucial insights into the inner workings of thousands of microservices, enabling the company to ensure efficiency and reliability at scale. These graphs exhibit several distinct features and highlight the unique challenges Alibaba's microservices ecosystem faces.

#### Heavy-Tail Distribution

About 10% of call graphs include more than 40 unique microservices; some even have over 1,500 services, which shows how complex Alibaba's service connections can be. Larger graphs often include numerous Memcached components, which expedite data retrieval for frequently accessed information, further optimising performance.

### Tree-Like Structures

Alibaba's microservice graphs tend to branch out rapidly, resembling tree-like structures. For example, stateless microservices frequently initiate a cascade of calls to downstream services, often ending with stateful components such as databases or caches. This tree-like branching facilitates parallel processing and serves requests more efficiently.

### Dynamic Topologies

The same services can show multiple topological variants based on runtime conditions, making resource management and performance optimisation tasks harder. For example, a single service may transition between nine classes of topologies depending on user requests, seasonal trends, or operational scenarios.

### Hotspot Services

Approximately 5% of microservices handle over 90% of invocations, so they are deemed critical for performance and resource allocation. These services are dubbed hotspots, such as payment processing and user authentication services. Such microservices provide functions that many online services need and address cross-cutting concerns.

### Critical Path Complexity

Some services have up to 70,000 critical paths, requiring targeted strategies for optimising resource allocation. The length and variability of these critical paths directly influence end-to-end latency, with longer paths causing notable performance bottlenecks.

### Call Dependency Fluctuations

Call graphs often experience periodic fluctuations due to diurnal user behaviour or business cycles. For instance, food delivery services peak during meal times, while e-commerce services show surges during promotional events.

## Key Insights

### Application Evolution

Alibaba's experience highlights significant differences between long-term and short-term developed applications. Long-term applications optimised over the years exhibit simpler call graph structures with fewer tiers and a more streamlined design. For example, such applications often provide 12 times more services than their short-term counterparts while maintaining higher throughput and reduced latency. By contrast, newer applications often feature deeper and more complex call graphs, with up to 10 tiers, leading to increased request latency and resource demands.

### Microservice Call Rate (MCR)

Alibaba's microservices handle an exceptionally high number of requests from both upstream services and users. Call rates exhibit considerable variability, with diurnal and event-driven patterns influencing traffic loads. For example, services experience up to a tenfold difference in call rates between peak and off-peak hours. These spikes can reach unprecedented levels during events like the Double 11 Shopping Festival. This variability requires predictive and adaptive strategies, including machine learning-based forecasting, to ensure resources are available when needed.

### Response Time Sensitivity

The sensitivity of microservices to CPU and network resource availability is a critical performance factor. Even moderate levels of CPU contention can result in a 20% degradation in response time. Alibaba mitigates this with fine-grained resource management policies and advanced job schedulers that spread workloads evenly across hosts. Network optimisation, such as leveraging field-programmable gate arrays (FPGAs) in Alibaba's X-Dragon infrastructure, further reduces latency by enhancing data transmission efficiency.

### Resource Utilisation Mismatch

Resource usage across services exhibits distinctive patterns. In Alibaba's clusters, CPU utilisation follows a heavy tail distribution: 95% of microservices consume less than 40% CPU utilisation, while a small number of services consume significantly more. Memory utilisation shows a different pattern—about 40% of services require more than 80% memory utilisation and can demand more than 60% heap utilisation.

## Memory Utilisation Stability

Unlike the highly dynamic CPU and network resources, memory usage in Alibaba's microservices remains relatively stable. This stability is maintained by emphasising Java Virtual Machine (JVM) heap usage instead of container-level metrics. Alibaba has observed that container memory usage often exceeds JVM heap usage by approximately 20% due to garbage collection processes. The system minimises out-of-memory issues and delivers consistent performance by managing memory through JVM metrics.

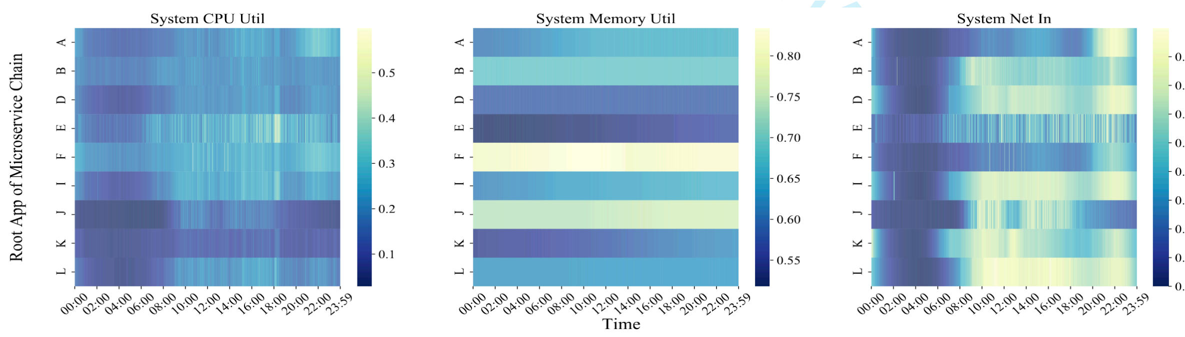

**Figure 3:** CPU, memory, and network usage fluctuations with time of day. Image credits Xu et al.

## Topology-Aware Allocation

Alibaba's microservice call graphs exhibit significant topological diversity, with the same service sometimes generating multiple distinct graph structures based on runtime conditions. By classifying these graphs, Alibaba tailors resource provisioning to the specific needs of each topology. This approach ensures that services with complex or deep call graphs receive the resources necessary to maintain performance, while simpler graphs allocate resources more conservatively.

## Impact of Dynamic Scaling

Dynamic scaling plays an essential role in maintaining runtime performance. Alibaba employs horizontal and vertical scaling techniques to address fluctuating workloads. Horizontal scaling adds additional containers to handle increased demand, while vertical scaling enhances the capacity of existing containers. These approaches are supported by Advanced Horizontal Pod Autoscaler (AHPA), which dynamically adjusts resource allocation based on real-time monitoring data.

### Proactive and Reactive Strategies

Alibaba combines proactive and reactive resource management strategies to address variability and ensure low latency. Proactive measures include historical data analysis and traffic forecasting to anticipate demand. Reactive strategies, such as burst-aware autoscaling, respond to sudden surges in traffic, allocating resources in real time to prevent performance degradation. These strategies ensure that Alibaba's microservices maintain high availability and responsiveness, even under extreme load conditions.

### Enhanced Network Resource Management

Alibaba improves network performance to avoid slowdowns in communication between services. It uses advanced message queuing systems to reduce delays in asynchronous tasks. Direct communication methods like Remote Procedure Calls (RPC) are employed for tasks requiring immediate responses. Additionally, intelligent load balancing helps evenly distribute network traffic across services.

## Comparing with Meta

A similar analysis of Meta's microservices architecture was previously reviewed (see [4]). We find fascinating similarities and differences when we analyse Meta's and Alibaba's microservices architectures.

### Scale and Service Count

Both companies operate on a massive scale but with notable differences. Meta's architecture consists of approximately 18,500 active services, managing over 12 million service instances as of late 2022. Alibaba's ecosystem comprises over 20,000 microservices, though their total instance count is not directly comparable due to different reporting methodologies. Both architectures demonstrate that production microservices deployments can be orders of magnitude larger than typical benchmarks or test environments.

### Topology Characteristics

Meta's topology is highly heterogeneous, containing three distinct types of software entities: well-scoped business services, multi-purpose services deployed as single units, and services that do not cleanly fit the microservices paradigm. Alibaba's topology reveals similar complexity but manifests differently, with about 5% of services being "hot spots" used by 90% of online services.

While Meta's services are sparsely interconnected with around 180,000 communication edges, Alibaba's topology shows a distinctive tree-like structure, with most nodes having an in-degree of one.

While both architectures show some power law/heavy-tail behaviours, they manifest in different ways. Meta sees power law behaviour primarily in service complexity (endpoint count), whereas Alibaba sees heavy-tail distributions in call graph characteristics and service usage patterns.

This difference might reflect the two organisations' different architectural choices and optimisation priorities. Meta's architecture consolidates complexity into a few highly complex services, while Alibaba's architecture distributes complexity across service interaction patterns.

### Dynamic Behaviour

Both architectures exhibit highly dynamic behaviour but in different ways. Meta's topology sees constant flux with daily service creation and deprecation fluctuations. Alibaba's dynamism manifests more in request patterns, where a single service can generate up to nine classes of topologically distinct graphs. These observations suggest that while both architectures are dynamic, they express dynamism at different system layers. It is important to note that Meta and Alibaba are very different businesses - one is a social media platform, and the other is an e-commerce service, so differences in dynamic behaviour are well expected.

### Underlaying Infrastructure

Finally, some key infrastructure differences between Alibaba and Meta's microservices architectures likely influence design choices and behaviours. For example, Alibaba uses a cloud-based infrastructure with Kubernetes managing containerised microservices on bare-metal servers. In contrast, Meta has its own geo-distributed on-premises data centres that are purpose-built for its specific needs.

## Conclusion

Alibaba uses a microservices architecture to manage its large operations. This system supports over 20,000 microservices and allows quick adjustments to meet changing demands. It relies on complex call structures and can scale dynamically to handle various tasks efficiently. By improving memory stability, response time, and resource management, Alibaba effectively handles large events like the

Double 11 Shopping Festival. A comparison with Meta's microservices architecture shows the importance of designing systems based on specific business needs. This comparison provides valuable insights for organisations implementing large-scale microservices architecture.

## References

[1] S. Luo *et al.*, "An In-Depth Study of Microservice Call Graph and Runtime Performance," 2022, *IEEE*. doi: 10.1109/TPDS.2022.3174631.

[2] M. Xu *et al.*, "Practice of Alibaba cloud on elastic resource provisioning for large-scale microservices cluster," 2024, *Wiley*. doi: 10.1002/spe.3271.

[3] A. Tiwari, "Unveiling Graph Structures in Microservices: Service Dependency Graph, Call Graph, and Causal Graph," 2024, *Abhishek Tiwari*. doi: 10.59350/hkjz0-7fb09.

[4] A. Tiwari, "Microservice architecture of Meta," 2024, *Abhishek Tiwari*. doi: 10.59350/7x9hc-t2q45.