Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis

Hailiang ZHAO @ ZJU-CS

http://hliangzhao.me

May 23, 2024

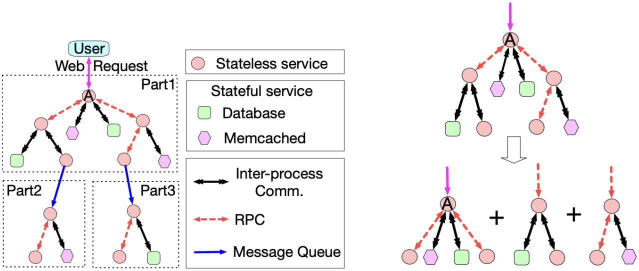
A lecture slide for the paper —— Luo et al., Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis, in: SoCC '21.

Outline

Outline

Call Graph

Call graph. The request from users is called *an origin request* and this request is first sent to an *Entering Microservice*, which then triggers a series of calls between related microservices.



(a) Components of call graphs **(b)** Two-tier invocations

Figure 1: Illustrations of microservices.

- **Stateless services** are isolated from state data
- **Stateful services**, e.g., databases and Memcached, need to store data

Call Graph

Communication Paradigms. There exist three types of communication paradigms between a pair of microservices:

- **Inter-process communication (IP)** happens between stateless and stateful microservices
- **Remote invocation**, such as RPC, is a two-way communication under which a DM must return a result to its corresponding UM ▷ *high efficiency*
- **Indirect communication** such as Message Queue (MQ) is one-way only (publish & subscribe) ▷ *good flexibility*

Hierarchical Call Dependencies

Hierarchical Call Dependencies. The call graph can be divided into several parts according to the edge of *indirect communication*. Each part can consist of multiple *two-tier invocations* with each consisting of a UM and all the DMs it calls (3 parts for the example in Fig. 1).

The call depth (a.k.a. the number of tiers) is the length of the *longest* path (5 for the example in Fig. 1).

Response Times

RTs. The response time (RT) of a call is the length of the interval from UM calling its DM to it receiving the response.[1]

- Since an indirect communication does not need to return a result, RT of an origin request is dominated by the part associated with its user (e.g., Part 1 in Fig. 1(a))
- The same class of user requests can trigger different microservice call procedures and thus incur heterogeneous RTs

[1] There is a place for call graph aware response time modeling!

Physical Running Environment

The authors analyze more than ten billion call traces among nearly twenty thousand microservices in 7 days from Alibaba cluster.

- **Physical running environment.** Alibaba clusters adopt Kubernetes to manage the bare-metal cloud.
  - *Online services* are running in containers and managed by Kubernetes directly
  - *Batch jobs* are running in secure containers and delivered to Fuxi for further scheduling

$$\textit{bare-metal} + \textit{secure containers} \rightarrow \textit{minimize interference}$$

  - Stateful services are deployed in a dedicated cluster which is not shared with other batch applications or stateless services

Microservices System Metrics

The monitoring system collects several system metrics *for each container* produced in every minute and takes the average to record.

- **Hardware.** Cache misses per kilo instructions, CPI
- **OS.** CPU & memory utilization
- **Application.** JVM heap utilization, JVM GC

Note that a microservice usually runs in hundreds of containers.

A data sample looks like:

$$TimeStamp, Metrics, Values, Microservices, PodIP$$

Microservices Invocations in a Call Graph

The monitoring system also records *the call dependency* between related microservices within a call graph.

- *TraceID*. All invocations triggered by the same user request share the same *TraceID*
- *Interface*, through which an UM calls a DM
- *UM Pod_IP* and *DM Pod_IP*
- *RT*
- *rpcID*, which contains the ID information of a pair of microservices
- *Communication Paradigm*, which includes IP, RPC, or indirect communication, e.g., MQ

An interesting data —— Among all the calls that happened between two *stateless* microservices in the traces, RPC, MQ and IP account for 76%, 23% and 1% of communication paradigms respectively.

Aggregate Statistics

The monitoring system also records *all* the calls (received from UMs or sent to DMs) related to each individual microservice.

- *Provided/Consumed Interface*. A microservice contains multiple provided interfaces to be called by its UMs. It calls DMs via different consumed interfaces.
- *call_times* quantifies the number of calls generated from each interface *in one minute* with the time recorded by *TimeStamp*
- *RT* characterizes the average response time among all these calls within one minute for each interface

Outline

Characterizes of Microservice Call Graphs

Observation 1

The size of call graphs follows a heavy-tail distribution.

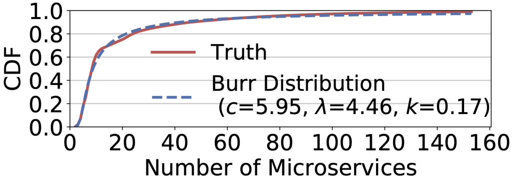


Figure 2: The number of microservices in a graph follows a Burr distribution (within 99th percentile).

- The largest call graph can even consist of hundreds to thousands of microservices
- For these call graphs of large size (containing more than 40 microservices), about 50% of their microservices are Memcacheds (faster than from DB)

Characterizes of Microservice Call Graphs (Cont'd)

Observation 1

The size of call graphs follows a heavy-tail distribution.

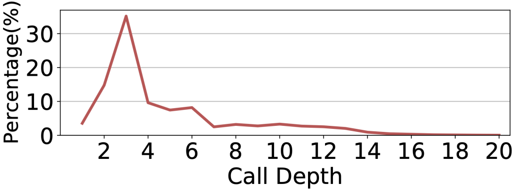


Figure 3: The distribution of call depth in all call graphs.

- A common graph depth in Alibaba traces is 3
- The call graphs have an average depth of 4.27, with a standard derivation of 3.25

Thus, it is extremely challenging to *configure the right number of containers* for all microservices in production clusters.

Characterizes of Microservice Call Graphs (Cont'd)

Observation 2

Microservice call graph behaves likes a tree and many of them only contain a long chain.

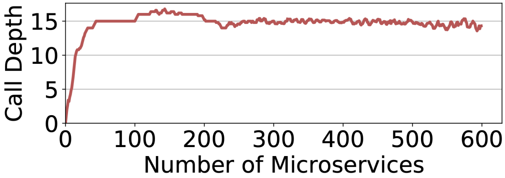


Figure 4: The maximum call depth (95th percentile) under a fixed number of microservices.

- The call depth stagnates when the number of microservices increases. This is due to that a microservice graph tends to branch out quickly like a tree to include more two-tier invocations (significantly different from DAG graphs from batch applications)

Characterizes of Microservice Call Graphs (Cont'd)

Observation 2

Microservice call graph behaves likes a tree and many of them only contain a long chain.

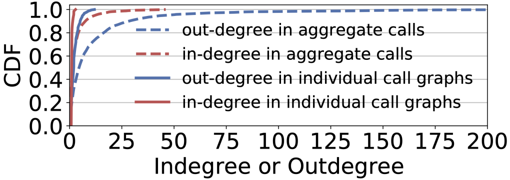


Figure 5: Distribution of the degree of stateless microservices in individual graphs and aggregate calls.

- More than 10% of stateless microservices have an out-degree of at least 5, while most of them have an in-degree of 1
- Note that a call is sent to a stateful microservice, it will not incur further calls[2]

[2] This characterize can be used in modeling.

Characterizes of Microservice Call Graphs (Cont'd)

Observation 2

Microservice call graph behaves likes a tree and many of them only contain a long chain.

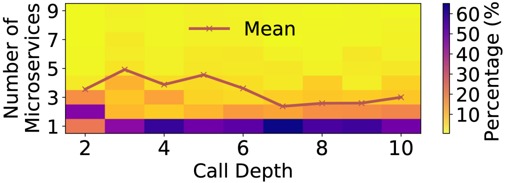


Figure 6: The distributions of the number of microservices in different tiers.

- As long as the depth becomes larger than two, the corresponding tier includes only one microservice with a high probability (above 60%) $\Longrightarrow$ As such, many deep graphs can be represented by one long chain

Characterizes of Microservice Call Graphs (Cont'd)

Observation 3

Many stateless microservices are hot-spots.

- As depicted in Fig. 5, more than 5% of microservices have in-degrees of 16 in aggregate calls
- These super microservices appear in nearly 90% of call graphs and handle 95% of total invocations in Alibaba traces

This result implies that, the loosely-coupled microservice architecture leads to a significant *unbalance* of workload across different microservices.

It indicates how should we do the scaling.

Characterizes of Microservice Call Graphs (Cont'd)

Observation 4

Microservice call graphs are highly dynamic.

- Microservice call graphs present significant topological differences between each other even among all the graphs generated by the *same* online services
- Once a call is sent to an entering microservice, the subsequent calls can be quite complicated depending on the status of a user

What about clustering the call graphs into clusters based on their topology?

Two-Tier Invocation Analysis

Observation 1

The call patterns of stateless microservices vary a lot over different tiers.

There are three types of stateless microservices: *normals*, *relays*, and *blackholes*.

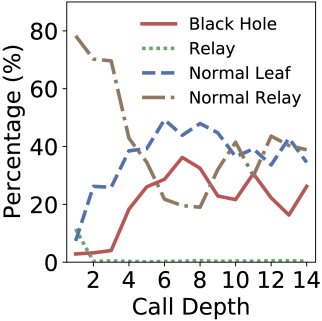


Figure 7: The percentage of black holes (relays) increases (decreases) with the call depth growing (call graphs with long depth take small portion).

Two-Tier Invocation Analysis (Cont'd)

Observation 1

The call patterns of stateless microservices vary a lot over different tiers.

- The probability that whether a normal microservice will call other microservices is still tier specific
- In expection, normal relays decrease over tiers. However, as shown in Fig. 7, when the call depth is above 8, such a probability increases over tiers

It is quite challenging to simulate production call graphs using simple mathematical models.

Two-Tier Invocation Analysis (Cont'd)

Observation 2

MQ contributes greatly to reducing the e2e RT in deep graphs.

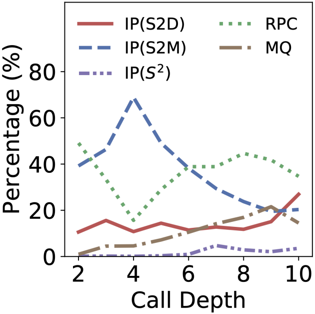


Figure 8: The distributions of communication paradigms in different tiers.

- The percentage of S2M reduces linearly in call depth when the depth is above 3 ▷ *increased cache miss*
- The percentage of S2D increases sublinearly ⇒ The left are filled by MQ ▷ *help reducing the e2e RT*

Outline

Cyclic Dependency

When a DM replies to its UM immediately without involving other microservices, a cyclic dependency exists.

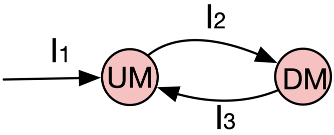


Figure 9: Cyclic dependency between a pair of microservices.

- **Strong dependency.** The entering interface of UM is the same as the reply interface for DM calls to call ($I_1 = I_3$)
- **Weak dependency.** Those two interfaces are different

Cyclic Dependency

Observation

Cyclic dependency makes up a non-negligible fraction among all dependencies.

UM \ DM	RPC	MQ
RPC	7.6%	0.0016%
MQ	0.00167%	0.22%

- As shown in the table, cyclic dependency contributes to more than 7.8% of the total microservice dependencies and most are via RPC calls (2.7% are strong dependencies)
- The number of cyclic calls involving three microservices is relatively small ($<$ 200 among billions of calls)

Hints. Confirm whether there is a strong need to combine the two interfaces into a single one to avoid deadlocks.

Coupled Dependency

A UM can repeatedly call the same DM multiple times in a two-tier invocation.

$$\text{Call Probability}(Y2X) = \text{Count}(X)/\text{Sum}, \tag{1}$$

$$\text{Call Time}(Y2X) = \text{Count}(X)/N. \tag{2}$$

When both are large, they form a strong coupled dependency.

- Count(X): the number of times Y calls X (X can be called multiple times by Y within the same two-tier invocation)
- Sum: The number of two-tier invocations triggered by Y in all call graphs
- N: The number of those two-tier invocations in which X was called

Coupled Dependency (Cont'd)

Observation

A noticeable fraction of pairs have strong coupled dependency and their interfaces could be coupled together for performance optimization.

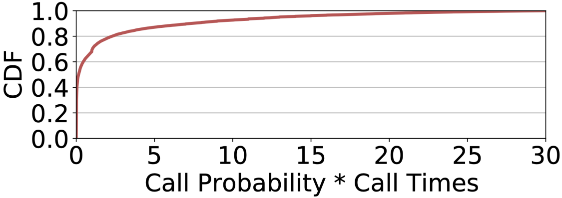


Figure 10: A high product means the DM will be called by UM within the same pair repeatedly with a high probability.

- More than 10% of pairs of microservices have a product of no less than 5. 17% of pairs with strong coupled dependency do not share DM with any other microservice

Couple the called interface of DM with that of UM together!

Parallel Dependency

In a two-tier invocation, a UM calls its multiple DMs either in a sequential manner or in parallel. Parallel dependency can help to greatly reduce the RT of upstream microservices (couple the two called interfaces into one).

Observation

Strong parallel dependency rarely exists in Alibaba traces.

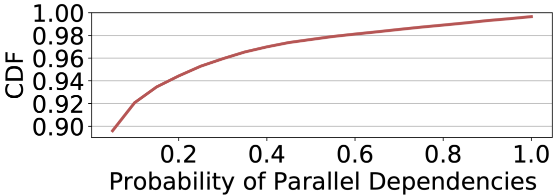


Figure 11: Cumulative distribution of the probability of parallel dependency between all pairs of microservices.

- 10% of pairs of microservices have a parallel dependency with probability larger than 0.05. Only 0.6% of pairs' probability larger than 0.9

Outline

Microservice Call Rate

Microservices run in hundreds to thousands of containers in a *hybrid* cluster and serve *time-varying* requests with *highly dynamic* call dependencies.

Microservice call rate (MCR) measures the number of calls received by a microservice in each minute per container.

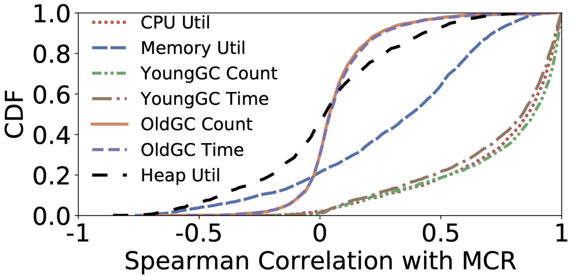


Figure 12: The correlation between MCR and diff. performance metrics.

We expect that *a large MCR leads to a high resource pressure.*

Microservice Call Rate

Observation

MCRs highly correlate with CPU utilization and Young GC but not with memory utilization.

- All microservices show a positive correlation **between the CPU utilization and MCR** and more than 80% of them yield a strong correlation ($SC > 0.6$)
- YongGC Count and YoungGC Time also show a strong correlation with MCR
- More than 20% of microservices have a **negative** correlation between MCR and memory utilization (memory utilization is almost stable at runtime in most containers in Alibaba microservice traces)

Microservice RT Performance

We cluster all the call graphs of each service into multiple classes (by the *InfoGraph* algorithm). Each class contains graphs of similar topology and call paths.

- *Within each class*, we compute both the standard derivation and the mean of the end-to-end RTs (i.e., *RTs of the Entering Microservice*) and then take the ratio between them as a measurement of the **intra-cluster-variance**.
- Similarly, we collect all end-to-end RTs *from all classes* of a service to measure the **inter-cluster-variance**.

Microservice RT Performance

Observation 1

End-to-End RTs of an online service are stable among call graphs of similar topologies but vary significantly across different topologies.

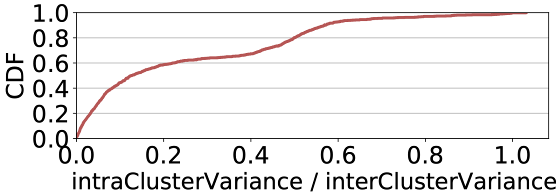


Figure 13: Cumulative distribution of RT intra-cluster variance to RT inter-cluster variance ratios.

- More than 90% of online services have a small ratio (< 0.6), indicating the RTs within each cluster are much more stable than that across different clusters

Microservice RT Performance

Online microservices usually co-exist with batch processing jobs on the same physical host to improve cluster utilization.
$\Rightarrow$ *resource interference!*

To measure its impact,

- We take the RT under a low host utilization (i.e., 10%) as a baseline and measure the normalized RTs under different host utilization for a fixed MCR.
- We then average all the normalized values across different MCRs under each host utilization. Finally, we compute the 75th percentile normalized RT among all microservices.

Microservice RT Performance

Observation 2

RT performance can be greatly degraded due to a high host CPU utilization.

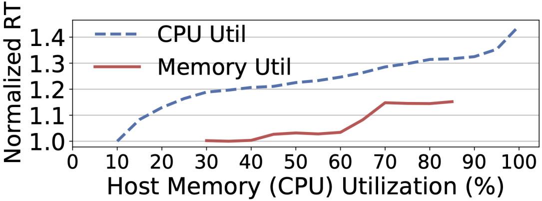


Figure 14: Performance degradation due to resource inter- ferences on the same physical host.

- When the host CPU utilization exceeds 40% (80%), the RT of a microservice can be degraded by more than 20% (30%) in average
- When the host memory utilization is below 60%, the interference can be ignored

Microservice RT Performance

Observation 3

RTs of a microservice are stable when the call rate varies.

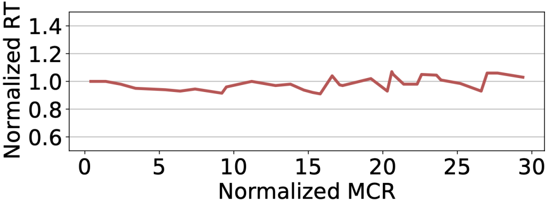


Figure 15: RT performance under different normalized microservice call rates —— Most calls in Alibaba clusters can be processed immediately without any queueing delay.

There is a large room to improve the resource utilization of microservices by resizing a proper number of running containers.

Outline

Stochastic Graph Model

A graph $G_s = (V_s, E_s)$ is initialized by a starting service s and can grow with more and more two-tier invocations.

- Each microservice $v \in V_s$ has a tier number (call depth) $d(v)$ ($d(s) = 1$). The largest tier number in G_s is denoted by h_s.
- Each edge $e(v_i, v_j)_{i<j} \in E_s$ is directed and formed by one parent v_i and one child v_j.
- Each $v \in V_s$ has a label $l(v)$ that denotes its type: $l(v) \in \mathbf{L} := \{db, Memcached, blackhole, replay, normal\}$.[3]
- We consider only two adjcent tiers, i.e., for each $e(v_i, v_j) \in E_s$, we have $d(v_j) = d(v_i) + 1$.

[3] Add *req*?

Stochastic Graph Model

Let $C(v) = |v_c : (v, v_c) \in E_s|$ denote the children set of v. For $v \in V_s$ with $d(v) = h$, $|C(v)|$ follows a random distribution given by

$$\Pr(|C(v)| = j) = F_n(j),$$

where F_h is the distribution of the number of microservices in a two-tier invocation starting from tier h in Alibaba traces.

Each child $v \in C(v)$ takes a label from $\mathbf{L}$ randomly based on the following distribution:

$$\Pr(l(v) = \phi) = G_{h+1}(\phi), \quad \forall \phi \in \mathbf{L},$$

where $G_{h+1}(\phi)$ can be simply derived by combining results from Fig. 7 and Fig. 8.