# Assignment #2:  MicroService System Troubleshooting

## Introduction

Troubleshooting of a microservice-based large software system is very challenging due to the large number of underlying microservices and the complex call relationships between them.  In this assignment, you are asked to analyze the root cause and its propagation of the failures in microservices system.

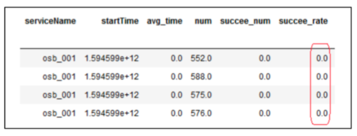
Recently, microservice architecture has become more and more popular for large-scale software systems in web-based services. This architecture decouples a web service into multiple microservices, each with well-defined APIs. There are complex call relationships between different microservices. Each microservice can be individually upgraded and maintained in microservice system. The above figure shows a specific user request to a microservice-based web service, in which the request is completed through several calls between microservices.

In this project, microservices are deployed in virutal machines(host) and you need to deal with 3 kinds of data sources. All these data are time series data and will be introduced in the subsequent part.

## TroubleShooting

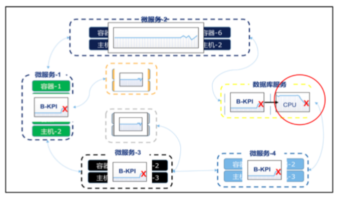
A failure in the microservice system can lead to many anomalous behaviors of different KPIs on different components due to the system's complex interactions between microservices. The root cause of a failure needs to be located as soon as possible, otherwise the failure will cause huge loss.

In this project, the task is to find out system nodes (vm or docker) and KPIs where the root cause occurs when a failure happens. More concretely, a feasible process of troubleshooting in this project is shown in the figure below:

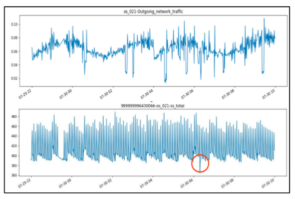


| serviceName | startTime | avg_time | num | succee_num | succee_rate |
|---|---|---|---|---|---|
| osb_001 | 1.594599e+12 | 0.0 | 552.0 | 0.0 | 0.0 |
| osb_001 | 1.594599e+12 | 0.0 | 588.0 | 0.0 | 0.0 |
| osb_001 | 1.594599e+12 | 0.0 | 575.0 | 0.0 | 0.0 |
| osb_001 | 1.594599e+12 | 0.0 | 576.0 | 0.0 | 0.0 |

Downgrade of success rate in ESB business indicator



Find anomalous component in trace around the downgrade time



Find the anomalous KPI in the abnormal node

The throubleshooting process in the figure has 3 steps:

1. Find the time point *t* when the business success rate was significantly lower than 1.
2. Around the time point *t*, look into the anomalous behaviors of microservices and record containers or hosts where the microservices are deployed. For example, microservice `csf_001` in docker node `docker_003` had very long response time around the time point *t*. Typically, this step can be finished by analysing the trace data.
3. In step2, the abnormal nodes, hosts or containers, are found. And then, you can detect which KPIs of the nodes perform anomalously.

## Data

In this project, you will get 3 types of KPI data sources. In this section, the format of all these data will be introduced.

## ESB business indicator(ESB)

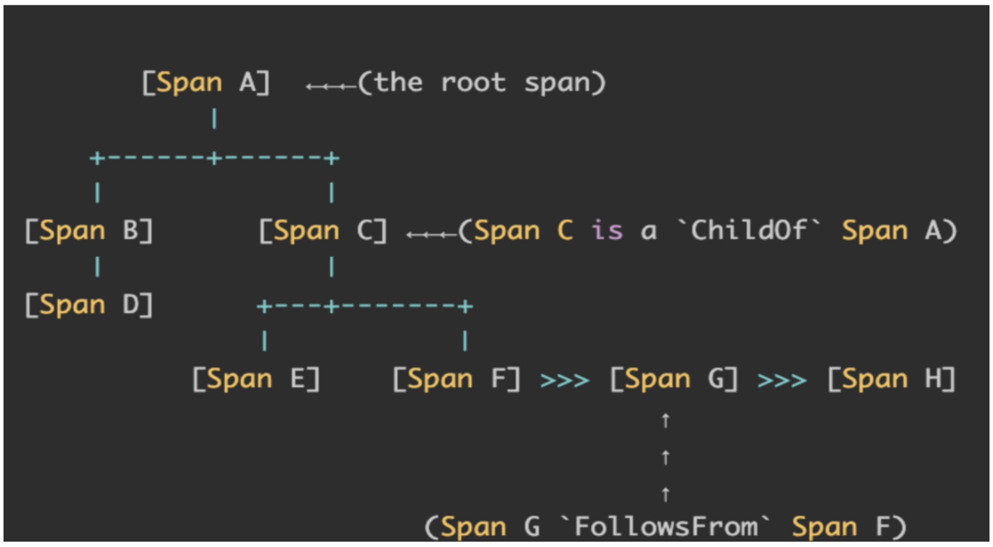
In this project, the microservice system deals with ***only 1*** kind of service request `osb_001`. The ESB data records the request information **every minute**:

| serviceName | startTime | avg_time | num | succee_num | succee_rate |
|---:|---:|---:|---:|---:|---:|
| osb_001 | 1588262400000 | 0.4718 | 361 | 361 | 1.0 |
| osb_001 | 1588262460000 | 0.4915 | 343 | 343 | 1.0 |
| osb_001 | 1588262520000 | 0.4901 | 359 | 359 | 1.0 |
| osb_001 | 1588262580000 | 0.5824 | 359 | 359 | 1.0 |
| osb_001 | 1588262640000 | 0.4923 | 385 | 385 | 1.0 |

- serviceName: service name, always `osb_001` in this project.
- startTime: information of request [startTime, startTime + 1min) is recorded in this row.
- avg_time: average time spent processing a request.
- num: the number of submitted requests.
- succee_num: the number of submitted requests which are successfully completed.
- succee_rate: *#succee_num / #num.*

## Trace

A trace corresponds to a user request and has a unique traceID. A trace consists of several microservice call records, called as `span`. A trace example is shown as the below figure:



As shown in the picture, the spans forms a tree structure, which means that every span except the root has a parent span. The parent relationship represents the call relationship between two microservices. You can refer to https://opentelemetry.io/docs/reference/specification/compatibility/opentracing/#span-references for more information on this.

Span mainly records 4 useful attributes about a microservice call:

- start time
- elapsed time
- host
- microservice name

It is important to notice that span is divided into 2 categories(**inside span** and **outside span** ) according to where the span is recorded. A Example will be given to explain how spans make up a traces.

```
def foo():
    print('foo begin time is: ', time.now) # t1
    ...
    print('call bar begin time is:', time.now)   # t3
    bar()
    print('call bar end time is:', time.now) # t4
    ...
    print('foo end time is :', time.now)   # t2


def bar():
    print('bar begin time is: ', time.now) # t5
    ...
    print('bar end time is: ', time.now) # t6

foo() # user request
```

In the above example, the user calls the function `foo` and there is a call statement in `foo` to call the function `bar`. Intuitively, the user request (calling `foo`) can be considered as a trace: foo -> bar consisting of 3 spans:

| id | parent id | start_time | elapsed_time | service | host | category |
|---|---|---|---|---|---|---|
| span1 | None | t1 | t2 - t1 | foo | host of foo | inside |
| span2 | span1 | t3 | t4 - t3 | bar | host of foo | outside |
| span3 | span2 | t5 | t6 - t5 | bar | host of bar | inside |

As shown in the code block and table, a span corresponds to 2 logs (print statements) recording start time and end time respectively. The inside span(span1 and span3) records the service being processed while the outside span(span2) records the service that will be called. It is important to notice that the **host** column is **where the span is generated**.

To be more specific, a real span has following attributes:

- id: unique id of this span.
- pid: id of its parent span.
- traceId: id of trace the span belongs to. spans with the same `traceId` make up a trace.
- startTime: `start_time` in table.
- elapsedTime: `elapsed_time` in table.
- serviceName: `service` in table.

- cmdb_id: `host` in table.
- callType: there is six calltypes in the trace data: `osb`, `remoteprocess`, `flyremote`, `csf`, `local` and `jdbc`. spans in `osb`, `remoteprocess` and `flyremote` are inside span and the others are outside span.
- success: `True` or `False` representing whether the service is processed successfully.
- dsName: There is a column named `dsName` in `trace_local` and `trace_jdbc`, which is the database accessed by microservice. And in `jdbc`, you can regard accessing databases as the microservice.

## Host KPIs data

Host KPIs data are the time series data with the format of (timestamp, value). There are multiple KPIs in each host (db, linux vm, docker...) and the host name is consistent with the `cmdb_id` column in the trace data and KPIs data.

| itemid | name | bomc_id | timestamp | value | cmdb_id |
|---|---|---|---|---|---|
| 999999998651280 | CPU_free_pct | ZJ-002-056 | 1588521600000 | 98.119746 | db_008 |
| 999999998650980 | CPU_free_pct | ZJ-002-056 | 1588521600000 | 98.837793 | db_003 |
| 999999998650680 | CPU_free_pct | ZJ-002-056 | 1588521600000 | 98.994754 | db_001 |
| 999999998651100 | MEM_real_util | ZJ-002-053 | 1588521600000 | 81.740000 | db_007 |
| 999999996381601 | CPU_free_pct | ZJ-002-056 | 1588521601000 | 95.593747 | db_009 |

There are 6 attributes in the KPIs data:

- (itemid, name, bomc_id): **the type of KPIs**. For example, `itemid` values are different between the 1st line and the 2nd line in the above table, thus, these 2 lines describe different KPIs.
- timestamp, value: the KPI is the `value` at the time of `timestamp`.
- cmdb_id: the host name of KPI. It is the same as `cmdb_id` in the trace data.

## What you need to do

In this assignment, you are required to **give explanations** for several failures.

Overall, we give the dataset and information about 5 failures, one of them:

```
[
 2220,  # failure time, Seconds past 00:00
 [
     ["os_021", "Sent_queue"]
 ] # 2 root causes: [cmdb_id, key_name]
]
```

The example means that there is something wrong within the system at 00:37. And you need to figure out in the system the specific performance that can indicate the failure and the corresponding root causes. For instance, you can give your answer from the following respects:

How the ESB data behave abnormally?

How the root causes affects the related Trace and Host KPIs?

Let's take the above failure as a example, you can easily find the there are some very high values of `avg_time` in ESB data around the failure time:

For trace data, you need to read carefully the introduction of trace data. The components in a trace will influence each other through their call relationships, So, you need to think about what kind of abnormal performance can indicate that there are root causes.

## Files given

One-day-long data of normal period: https://cloud.tsinghua.edu.cn/f/7eece510dc784e70a083/

Test data where failures happen: https://cloud.tsinghua.edu.cn/f/0593e1aa85144cf2b745/

System structure: cmdb.xlsx, where column "name" is deployed on column "host".

Failures: failures.json

## Submission

Submit a report that contains your explanations for the failures.

Reports can be submitted in pdf, doc, ipynb, etc, but all content must be in English.