lab

实验一 系统搭建

> 搭建对象存储系统，熟悉对象存储系统的操作。

minio

minio server

```
$ wget https://dl.min.io/server/minio/release/linux-amd64/minio
$ chmod +x minio
$ export MINIO_ROOT_USER=liaoyujian && export MINIO_ROOT_PASSWORD=nian1113 && minio
server /data
```

```
root@iZbp1dhxop9xyp51nal11dZ:~# export MINIO_ROOT_USER=liaoyujian && export MINIO_ROOT_PASSWORD=nian1113 &&
 minio server /data

 You are running an older version of MinIO released 2 weeks ago
 Update: Run `mc admin update`

API: http://172.27.193.88:9000  http://127.0.0.1:9000
RootUser: liaoyujian
RootPass: nian1113

Console: http://172.27.193.88:46421 http://127.0.0.1:46421
RootUser: liaoyujian
RootPass: nian1113

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
   $ mc alias set myminio http://172.27.193.88:9000 liaoyujian nian1113

Documentation: https://docs.min.io
```

minio client

```
$ go install github.com/minio/mc@latest
$ mc alias set myminio http://172.27.193.88:9000 liaoyujian nian1113
```

```
root@iZbp1dhxop9xyp51nal11dZ:~# mc alias set myminio http://172.27.193.88:9000 liaoyujian nian1113
Added `myminio` successfully.
```

- 创建桶 `mc mb myminio/document`

```
root@iZbp1dhxop9xyp51nal11dZ:~# mc mb myminio/document
Bucket created successfully `myminio/document`.
```

- 拷贝移动对象 `mc cp[mv] time.txt myminio/document/`

```
root@iZbp1dhxop9xyp51nal11dZ:~# mc mv time.txt myminio/document/
time.txt:          18 B / 18 B ┃▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓┃ 1.38 KiB/s 0s
root@iZbp1dhxop9xyp51nal11dZ:~# mc cp time.txt myminio/document/
time.txt:          4 B / 4 B ┃▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓┃ 377 B/s 0s
```

- 显示对象 `mc ls myminio/document/`

```
root@iZbp1dhxop9xyp51nal11dZ:~# mc ls myminio/document/
[2021-12-17 14:40:29 CST]    18B time.txt
```

- 删除对象或者bucket `mc rm -r --force myminio/document/`

实验二 性能观测

> 熟悉性能指标：吞吐率、带宽、延迟
>
> 分析不同负载下的指标、延迟的分布

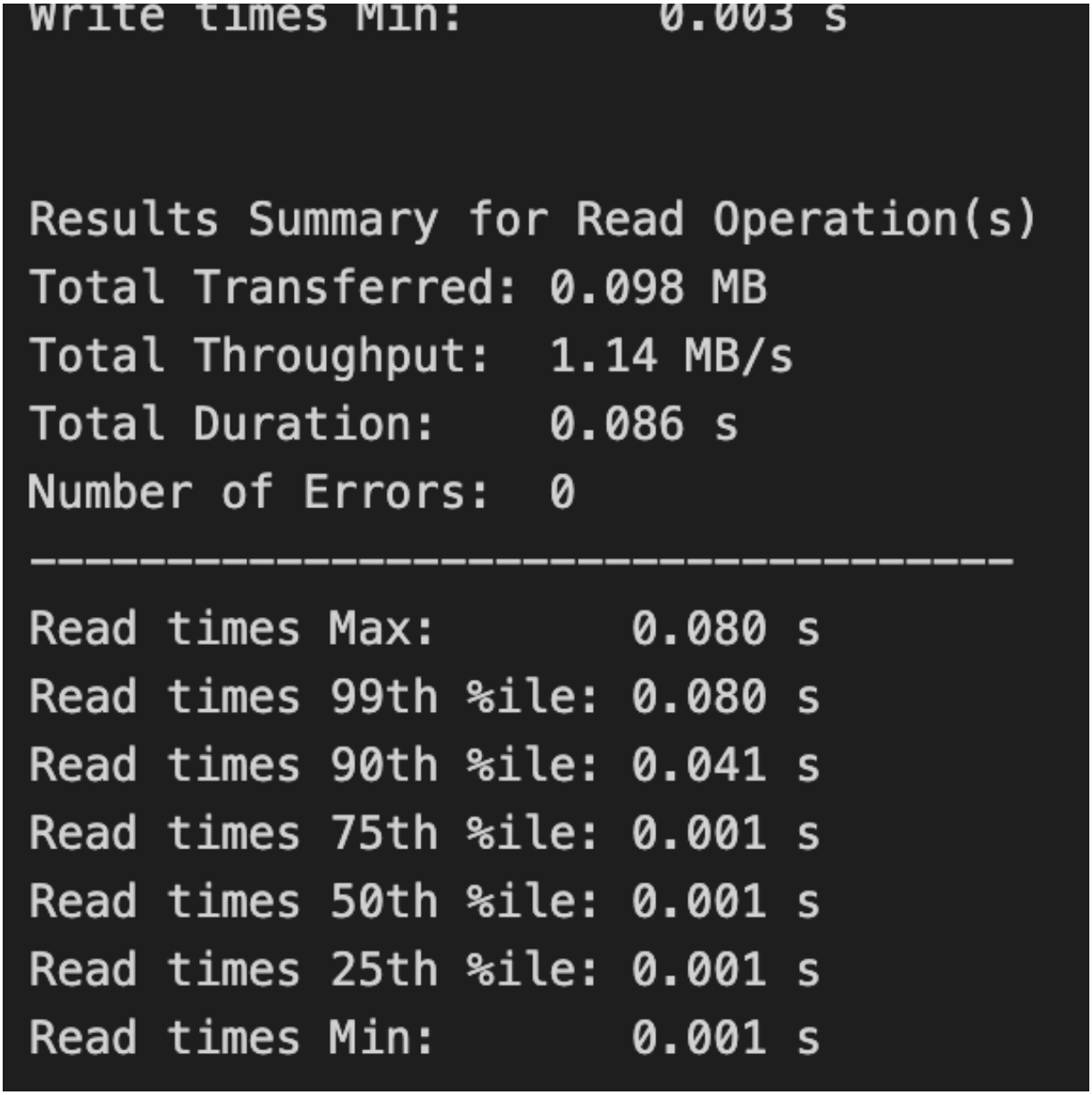
选择方案S3 Bench

```
$ go get -u github.com/igneous-systems/s3bench
```

- 命令行测试 **线程*10* 每个对象大小*1KB***

```
s3bench \
  -accessKey=liaoyujian -accessSecret=nian1113 \
  -endpoint=http://172.27.193.88:9000 \
  -bucket=loadgen -objectNamePrefix=loadgen \
  -numClients=10 -numSamples=100 -objectSize=1024
```

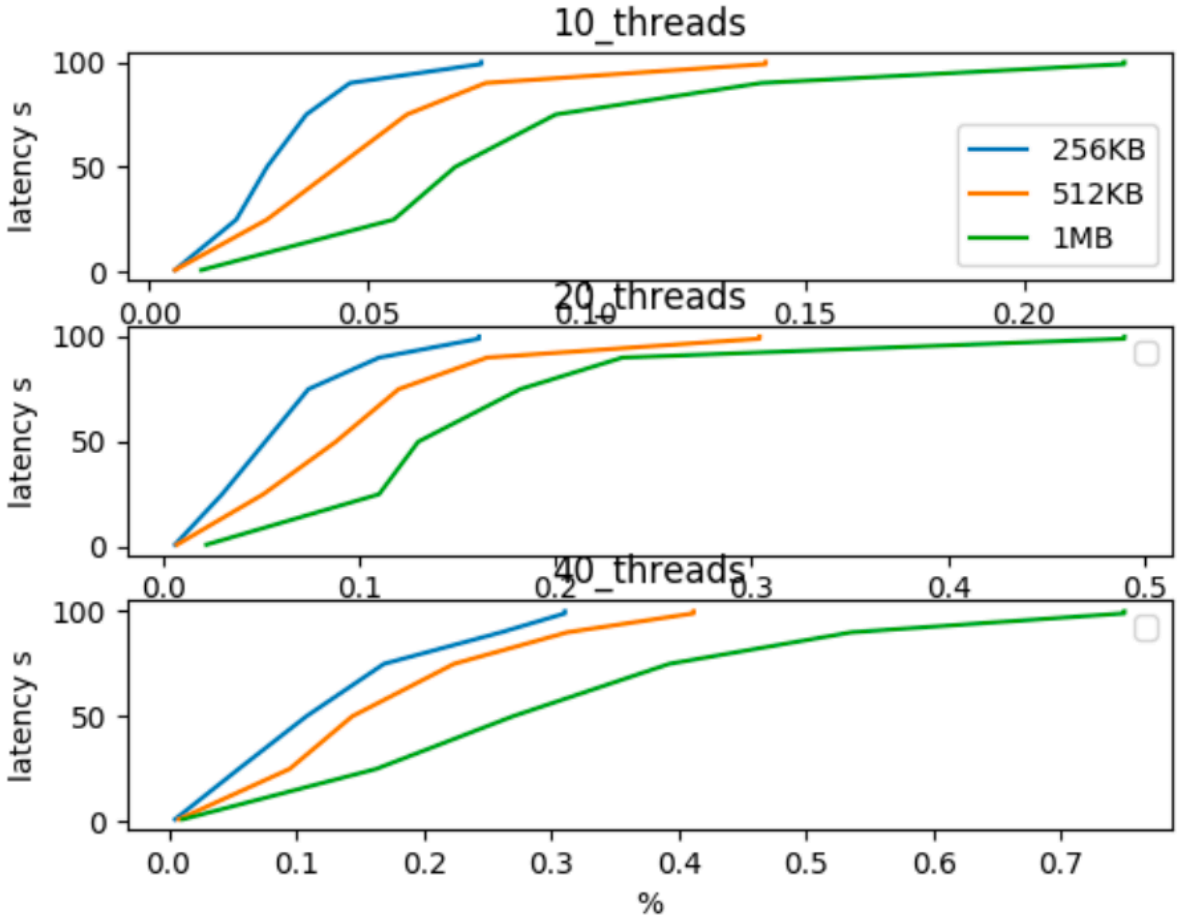
```
Results Summary for Write Operation(s)
Total Transferred: 0.098 MB
Total Throughput:  0.73 MB/s
Total Duration:    0.134 s
Number of Errors:  0
-------------------------------------
Write times Max:       0.029 s
Write times 99th %ile: 0.029 s
Write times 90th %ile: 0.021 s
Write times 75th %ile: 0.016 s
Write times 50th %ile: 0.012 s
Write times 25th %ile: 0.009 s
```



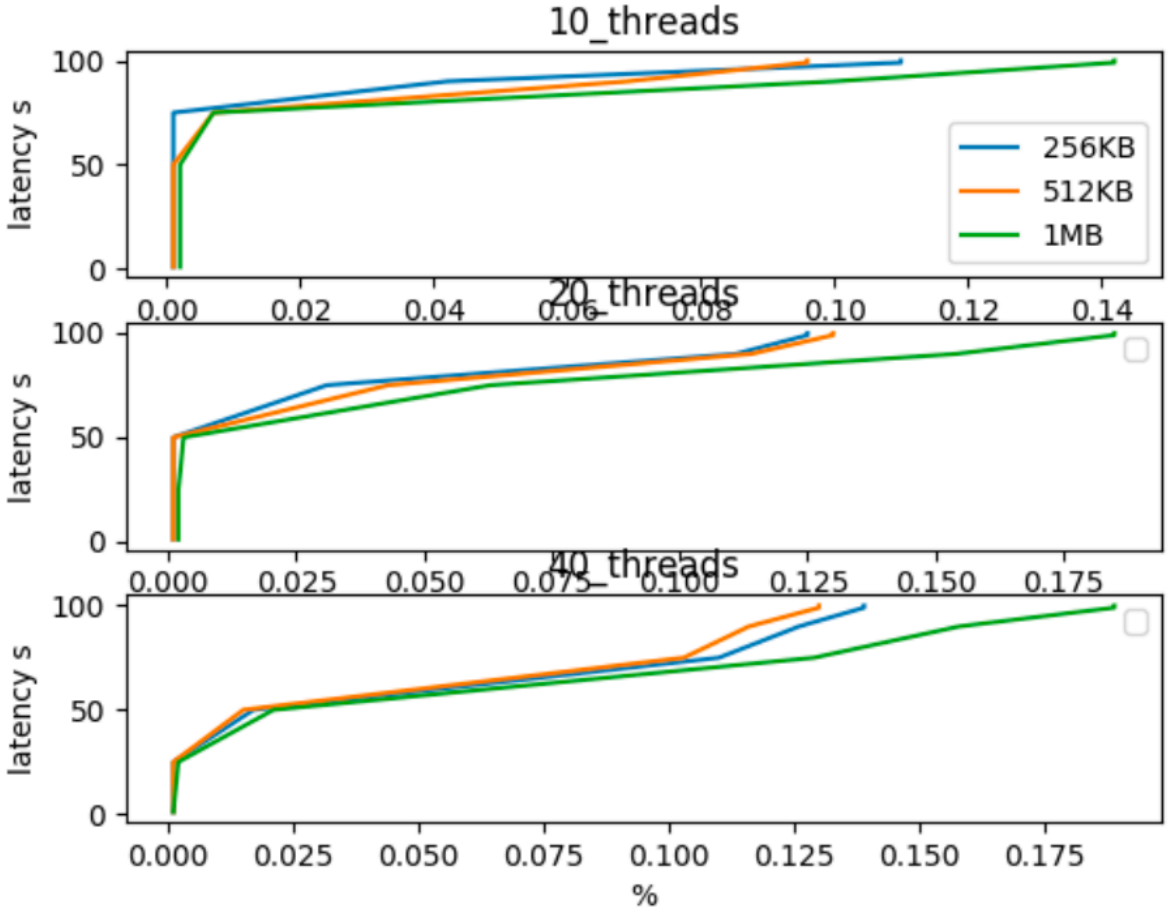
- 不同线程，不同大小负载测试

threads	10	20	40
Size (KB)	256,512,1024	256,512,1024	256,512,1024

    - ***Write***


10_threads
latency s
100
50
0
0.00
0.05
0.10
0.15
0.20
256KB
512KB
1MB
20_threads
latency s
100
50
0
0.0
0.1
0.2
0.3
0.4
0.5
40_threads
latency s
100
50
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
%

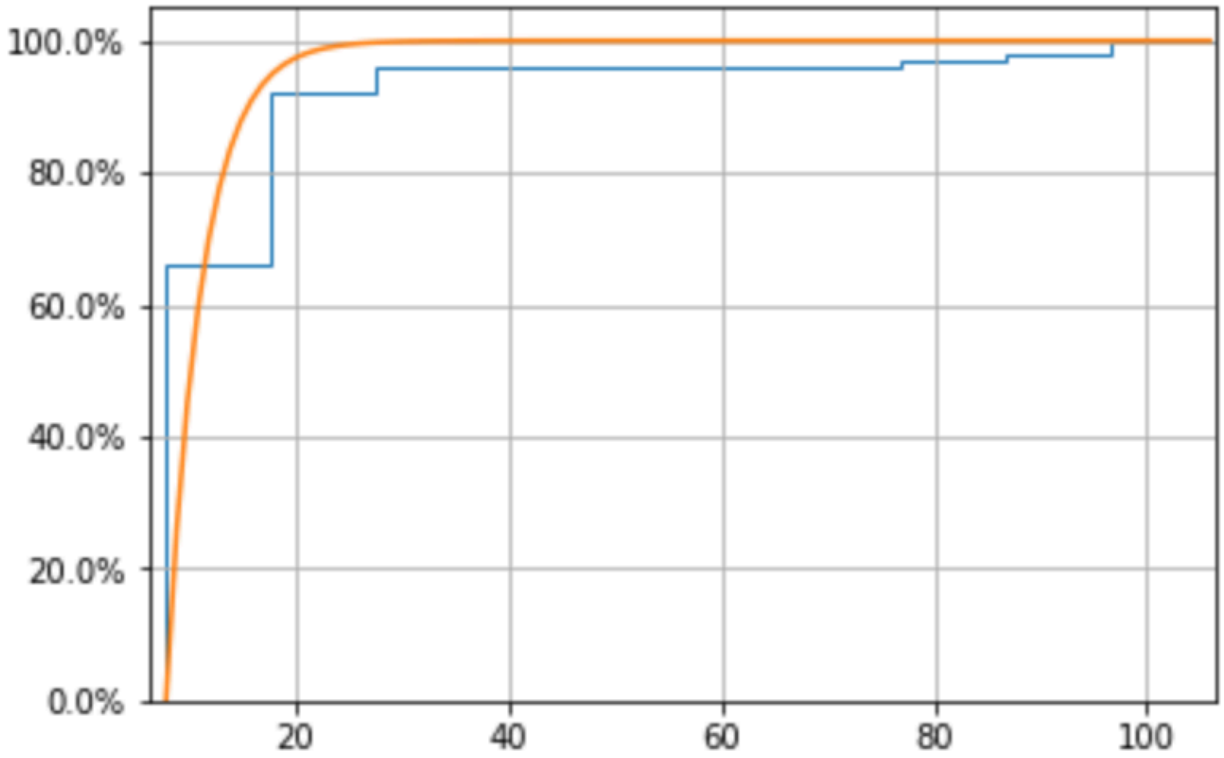
- ***READ***


10_threads
latency s
100
50
0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
256KB
512KB
1MB
20_threads
latency s
100
50
0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
40_threads
latency s
100
50
0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
%

> 分析Write和Read曲线，大体上，线程越多，对象的大小越大，尾延迟现象越明显

实验三 尾延迟应对

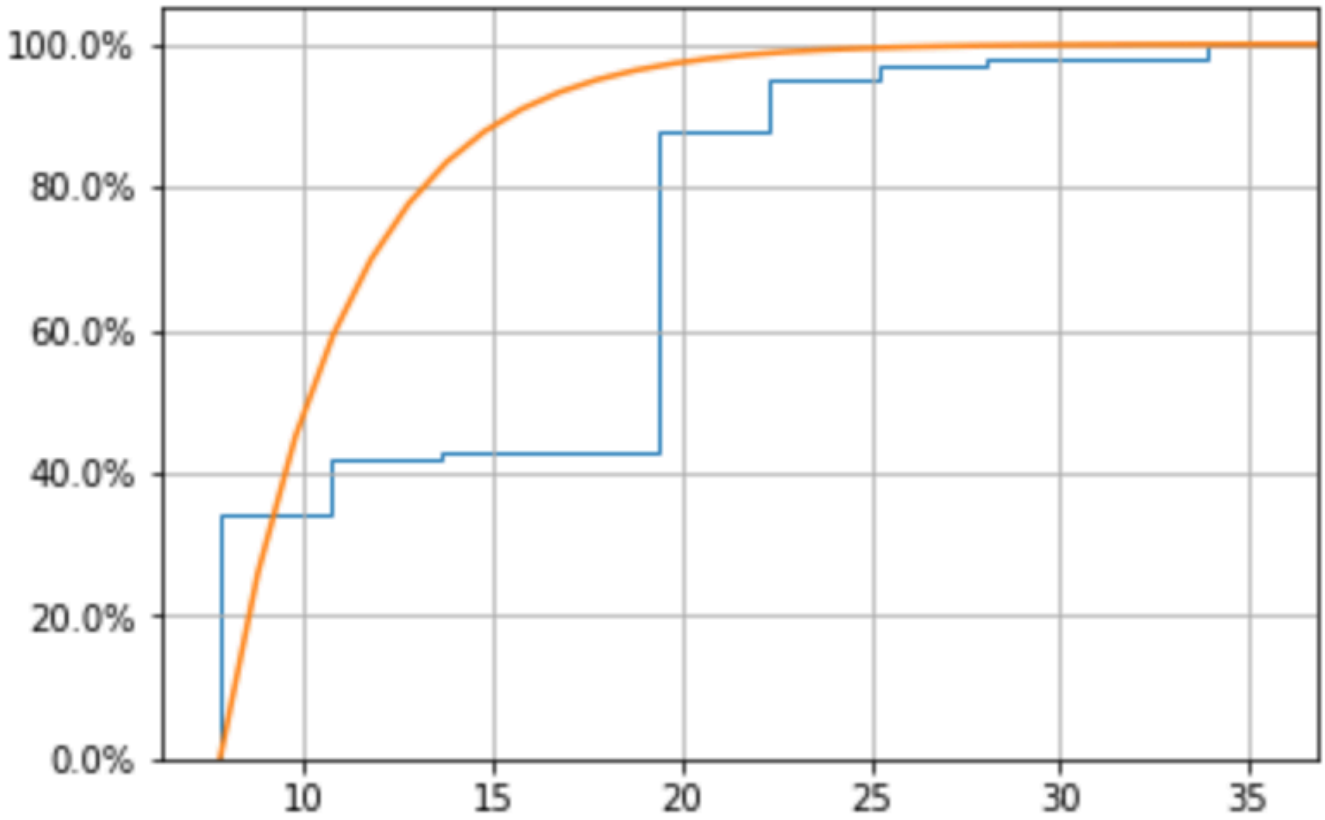
- 原始尾延迟现象



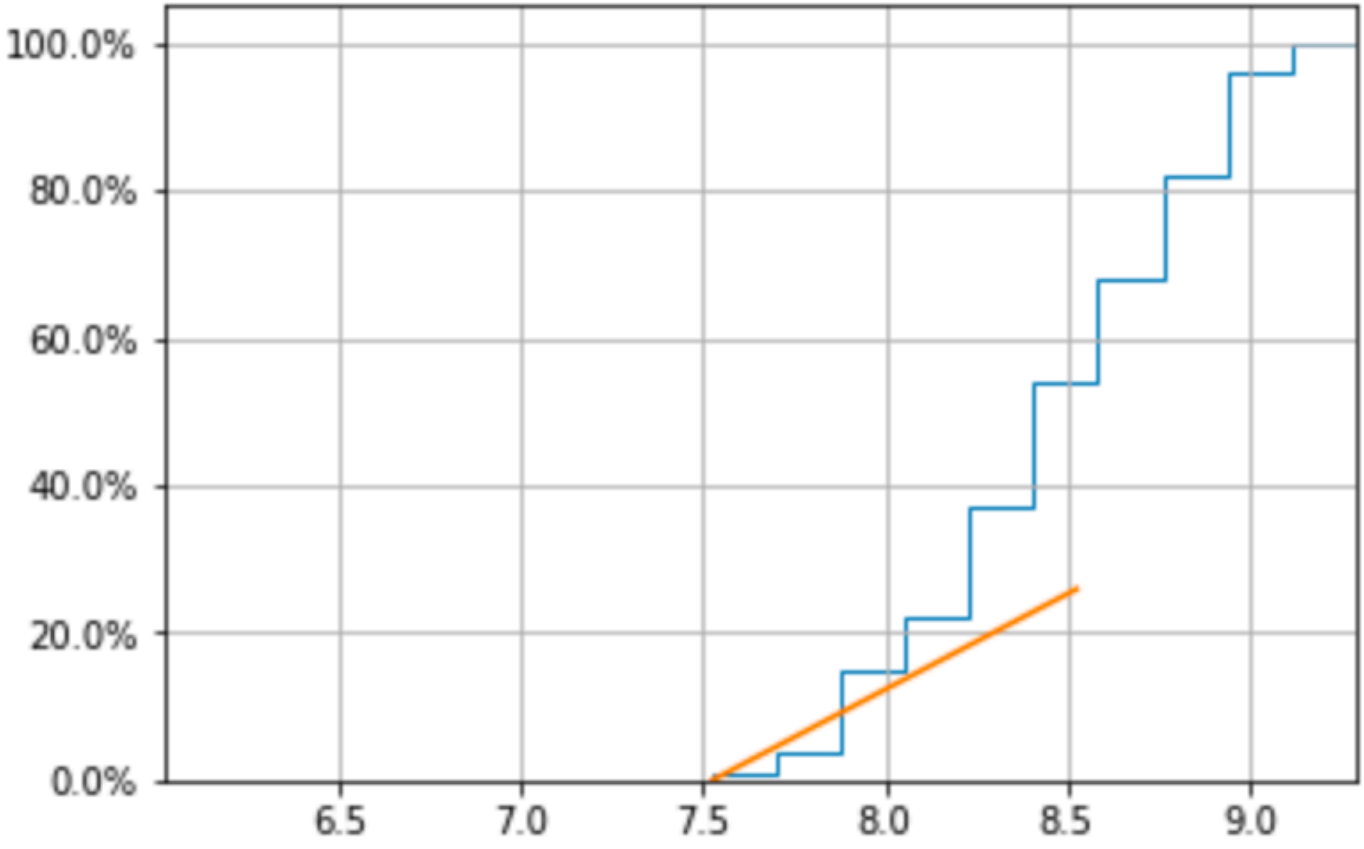
关联请求Tied requests

> 同时发给多个副本，但告诉副本还有其它的服务也在执行这个请求，副本任务处理完之后，会主动请求其它副本取消其正在处理的同一个请求。

- 同时发起两个请求， **尾延迟时间降低明显，95%的请求都能在20ms以内响应**



- 理想情况下，**发送200个请求，取前100个请求的响应时间，延迟大大减少**



对冲请求Hedged requests

> 抑制延迟变化的一个简单方法是向多个副本发出相同的请求，并使用首先响应的结果。一旦收到第一个结果，客户端就会取消剩余的未处理请求。不过直接这么实现会造成额外的多倍负载。
>
> 一个方法是推迟发送第二个请求，直到第一个请求到达 95 分位数还没有返回

95分位，大概为20ms左右

- 延迟对冲，对延迟超过20ms的请求进行对冲