

科大讯飞股份有限公司

文档密级：公司内部 A

AIUI 日志系统延迟消费及洪峰内存 激增问题攻关总结

本文件属科大讯飞股份有限公司所有，
未经书面许可，不得以任何形式复印或传播。

文档修改记录

版本	时间	修改人	修改范围
1.0	2021-01-22	吴金福	创建文档

目录

- 1 引言 1
 - 1.1 背景 1
 - 1.2 文档范围 1
 - 1.3 读者对象 1
 - 1.4 参考资料 1
 - 1.5 术语与缩写解释 2
- 2 问题难点 2
- 3 攻关过程 3
 - 3.1 日志系统架构图 3
 - 3.2 问题分析 3
 - 3.2.1 日志丢失 3
 - 3.2.2 延迟消费 4
 - 3.2.3 内存激增 5
 - 3.3 问题复现 5
 - 3.4 ELK 8
 - 3.5 Filebeat 9
 - 3.6 Zap 11
- 4 结论 14
- 5 经验总结 15

1 引言

1.1 背景

AIUI 日志及监控系统是基于 ELK 部署的，主要负责现网问题的排查、业务数据的落盘以及服务监报告警的实施。ELK 是 AIUI 服务稳定及业务支撑非常重要的一环。

近期，在业务高峰期，频频出现日志延迟消费、日志丢失、组件内存激增的现象。这导致在业务高峰期一旦发生告警，则可能会发生告警延迟现象，即服务已经恢复正常，但是告警邮件和短信依然在发送，给运维和开发人员带来了很大困惑。同时日志丢失也影响了问题排查和业务数据落盘。组件内存激增则会导致业务高峰期内存增至 20G 左右，甚至引起组件重启。

基于此问题的严重性，特进行 AIUI 日志系统延迟消费及洪峰内存激增问题的攻关行动，攻关持续时间半个月，已彻底解决问题。

1.2 文档范围

此文档适用于部门和项目内的设计人员、开发人员、测试人员、交付人员，以及系统运维人员查看。

1.3 读者对象

此文档适用于部门和项目内的设计人员、开发人员、测试人员、交付人员，以及系统运维人员查看。

1.4 参考资料

序号	文档名称	文档来源	最后修改日期
1	zap	https://github.com/uber-go/zap	
2	filebeat	https://github.com/elastic/beats/tree/master/filebeat https://www.elastic.co/guide/en/beats/filebeat/current/index.html	
3	Filebeat-Elastic 中文社区	https://elasticsearch.cn/topic/filebeat	

1.5 术语与缩写解释

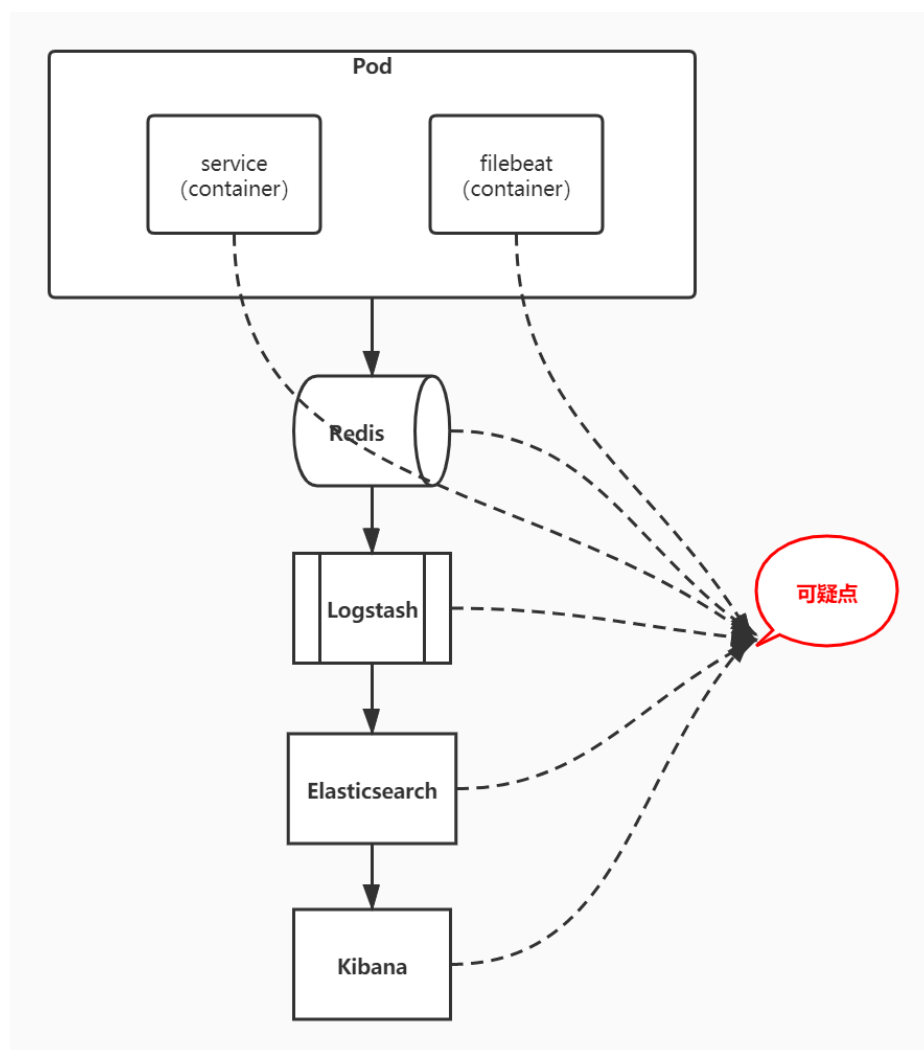
序号	术语/缩写	解释
1	ELK	ELK 是三个开源软件的缩写，分别表示：Elasticsearch , Logstash, Kibana，它们都是开源软件。新增了一个 FileBeat，它是一个轻量级的日志收集处理工具(Agent)，Filebeat 占用资源少，适合于在各个服务器上搜集日志后传输给 Logstash，官方也推荐此工具。

2 问题难点

- 现网日志量巨大（一周 250 亿条，占用磁盘 50T），问题仅在高峰期出现，现网难排查。
- 涉及业务组件及监控组件多，测试环境难复现。
- 问题链路长，疑点多，增加问题定位难度。

3 攻关过程

3.1 日志系统架构图



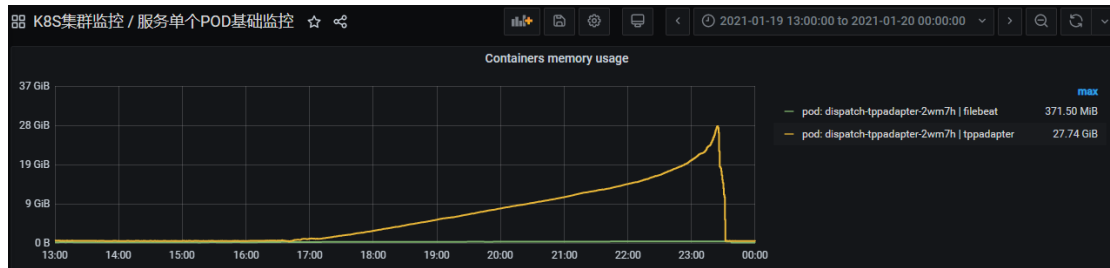
3.2 问题分析

3.2.1 日志丢失

日志丢失一般由业务线反馈问题、排查问题时发现，无法统计具体丢失日志所占比例。从现象来看，大部分日志丢失发生在业务高峰期，因此有理由怀疑日志丢失和延迟消费问题来源是一致的。一旦解决了延迟消费问题，应该也可以解决日志丢失问题。

3.2.3 内存激增

从问题组件的内存监控图上看，内存从晚高峰持续上涨，然后在某个时间点猛跌。猜测是日志积压导致的内存持续上涨。



3.3 问题复现

因为现网日志量巨大，而且为了保证现网稳定性，无法在现网进行问题 debug。因此第一步是编写核心功能模块的独立测试程序，尝试在测试环境复现问题。

```
root@aui-diaodu-001-no:~/zaptest# ./service -h
Usage of ./service:
-i int
    interval (default 10)
-m int
    maxfilenum (default 10)
-n int
    number (default 10)
-p string
    path (default "./zap/session.log")
-s int
    string num (default 1000)
root@aui-diaodu-001-no:~/zaptest# ./service -p ./zap/session.log -s 10000 -n 20 -m 30
```

主要尝试调整以下几个变量来模拟日志积压和内存上涨现象

- 调整 redis 内存，通过设置 `config set maxmemory` 来实现。
- 调整日志写入的时间间隔。
- 调整滚动日志的最大数量。
- 调整协程数量。
- 调整写入的日志长度。
- 调整 filebeat 的配置参数。

调整各个变量，观察日志积压的同时，通过 `golang pprof` 分析内存占用情况。


```
root@aui-diaodu-001-no:~# go tool pprof -inuse_space http://172.31.131.181:8877/debug/pprof/heap
Fetching profile over HTTP from http://172.31.131.181:8877/debug/pprof/heap
Saved profile in /root/pprof/pprof.service.alloc_objects.alloc_space.inuse_objects.inuse_space.017.pb.gz
File: service
Type: inuse_space
Time: Jan 22, 2021 at 2:56pm (CST)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) top 20
Showing nodes accounting for 1034.03kB, 100% of 1034.03kB total
Showing top 20 nodes out of 21
      flat  flat%   sum%        cum   cum%
  517.02kB  50.00%  50.00%    517.02kB  50.00%  bytes.makeSlice
  517.02kB  50.00%  100%    517.02kB  50.00%  go.uber.org/zap/buffer.(*Buffer).Write
    0         0%  100%    517.02kB  50.00%  bytes.(*Buffer).WriteString
    0         0%  100%    517.02kB  50.00%  bytes.(*Buffer).grow
    0         0%  100%   1034.03kB  100%  encoding/json.(*Encoder).Encode
    0         0%  100%    517.02kB  50.00%  encoding/json.(*encodeState).marshal
    0         0%  100%    517.02kB  50.00%  encoding/json.(*encodeState).reflectValue
    0         0%  100%    517.02kB  50.00%  encoding/json.(*encodeState).string
    0         0%  100%    517.02kB  50.00%  encoding/json.arrayEncoder.encode
    0         0%  100%    517.02kB  50.00%  encoding/json.mapEncoder.encode
    0         0%  100%    517.02kB  50.00%  encoding/json.sliceEncoder.encode
    0         0%  100%    517.02kB  50.00%  encoding/json.stringEncoder
    0         0%  100%   1034.03kB  100%  go.uber.org/zap.(*Logger).Info
    0         0%  100%   1034.03kB  100%  go.uber.org/zap/zapcore.(*CheckedEntry).Write
    0         0%  100%   1034.03kB  100%  go.uber.org/zap/zapcore.(*ioCore).Write
    0         0%  100%   1034.03kB  100%  go.uber.org/zap/zapcore.(*jsonEncoder).AddReflected
    0         0%  100%   1034.03kB  100%  go.uber.org/zap/zapcore.(*jsonEncoder).EncodeEntry
    0         0%  100%   1034.03kB  100%  go.uber.org/zap/zapcore.Field.AddTo
    0         0%  100%   1034.03kB  100%  go.uber.org/zap/zapcore.addFields
    0         0%  100%   1034.03kB  100%  main.SessionLogSync
(pprof) █
```

通过实验发现，在短文本的情况下没有出现日志积压现象，内存增长至 500M 左右后稳定。分析发现滚动日志单个文件 50M，一共 10 个文件，因此这部分监控统计到的内存占用主要是虚拟内存占用。

于是对比观察现网出现积压的组件日志，发现问题组件的日志均比较长。因此调整单条日志长度为 2000，增大日志写入频率（日志平均每秒新增 20M 左右），出现积压现象，而且内存出现持续上涨。

```
root@dispatcher-kc-adapter-current-6bf8b89f8-j2bmn:/tmp# cat /proc/5081/status
Name: service
State: S (sleeping)
Tgid: 5081
Ngid: 10796
Pid: 5081
PPid: 12960
TracerPid: 0
Uid: 0 0 0 0
Gid: 0 0 0 0
FDSize: 256
Groups:
NSTgid: 5081
NSpid: 5081
NSpgid: 5081
NSSid: 12960
VmPeak: 1761352 kB
VmSize: 1704012 kB
VmLck: 0 kB
VmPin: 0 kB
VmHWM: 14364 kB
VmRSS: 12900 kB
VmData: 1689900 kB
VmStk: 228 kB
VmExe: 3548 kB
VmLib: 2024 kB
VmPTE: 252 kB
VmPMD: 24 kB
VmSwap: 0 kB
HugetlbPages: 0 kB
Threads: 25
SigQ: 0/515413
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000000000
SigCgt: ffffffffclfeff
CapInh: 00000000a80425fb
CapPrm: 00000000a80425fb
CapEff: 00000000a80425fb
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
Seccomp: 0
Speculation_Store_Bypass: vulnerable
Cpus_allowed: 00ffffff
Cpus_allowed_list: 0-23
Mems_allowed: 00000000,00000003
Mems_allowed_list: 0-1
voluntary_ctxt_switches: 510167
nonvoluntary_ctxt_switches: 6373
```

VmSize(KB)	任务虚拟地址空间的大小 (total_vm-reserved_vm)，其中 total_vm 为进程的地址空间的大小，reserved_vm: 进程在预留或特殊的内存间的物理页
VmLck(KB)	任务已经锁住的物理内存的大小。锁住的物理内存不能交换到硬盘 (locked_vm)
VmRSS(KB)	应用程序正在使用的物理内存的大小，就是用 ps 命令的参数 rss 的值 (rss)
VmData(KB)	程序数据段的大小（所占虚拟内存的大小），存放初始化了的数据； (total_vm-shared_vm-stack_vm)
VmStk(KB)	任务在用户态的栈的大小 (stack_vm)

VmExe(KB)	程序所拥有的可执行虚拟内存的大小，代码段，不包括任务使用的库 (end_code-start_code)
VmLib(KB)	被映像到任务的虚拟内存空间的库的大小 (exec_lib)
VmPTE	该进程的所有页表的大小，单位：kb

通过对内存占用情况的分析，增长内存主要是虚拟内存占用，物理内存占用并不高。这也佐证了前面的猜想，因为日志积压导致监控内存上涨。下一步就是要分析出具体的原因，这需要分别对 ELK、filebeat 和 zap 做具体的了解和分析。

3.4 ELK

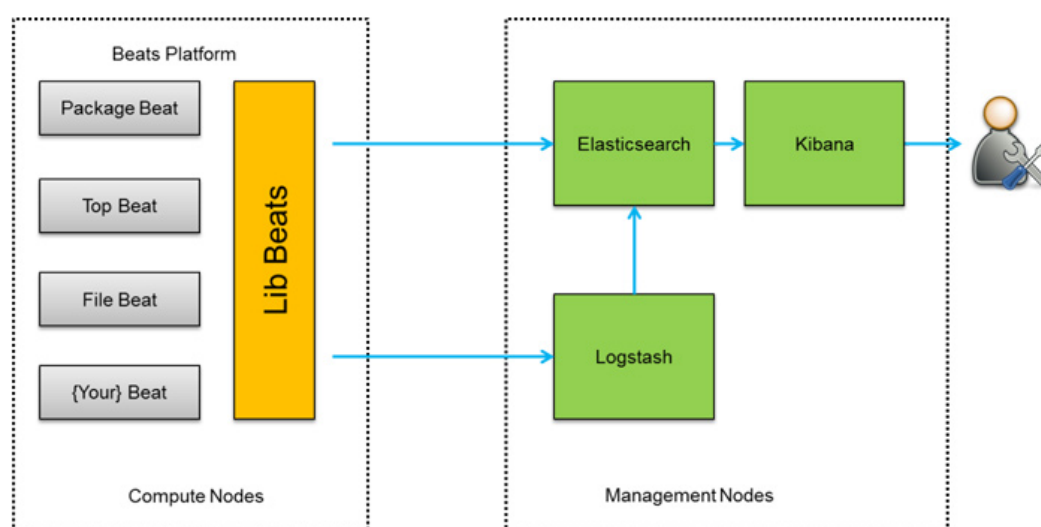
ELK 是三个开源软件的缩写，分别表示：Elasticsearch, Logstash, Kibana，它们都是开源软件。Elasticsearch 是个开源分布式搜索引擎，提供搜集、分析、存储数据三大功能。它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，restful 风格接口，多数据源，自动搜索负载等。

Logstash 主要是用来日志的搜集、分析、过滤日志的工具，支持大量的数据获取方式。一般工作方式为 c/s 架构，client 端安装在需要收集日志的主机上，server 端负责将收到的各节点日志进行过滤、修改等操作在一并发往 elasticsearch 上去。

Kibana 也是一个开源和免费的工具，Kibana 可以为 Logstash 和 ElasticSearch 提供的日志分析友好的 Web 界面，可以帮助汇总、分析和搜索重要数据日志。

Filebeat 隶属于 Beats。目前 Beats 包含四种工具：

- Packetbeat（搜集网络流量数据）
- Topbeat（搜集系统、进程和文件系统级别的 CPU 和内存使用情况等数据）
- Filebeat（搜集文件数据）
- Winlogbeat（搜集 Windows 事件日志数据）



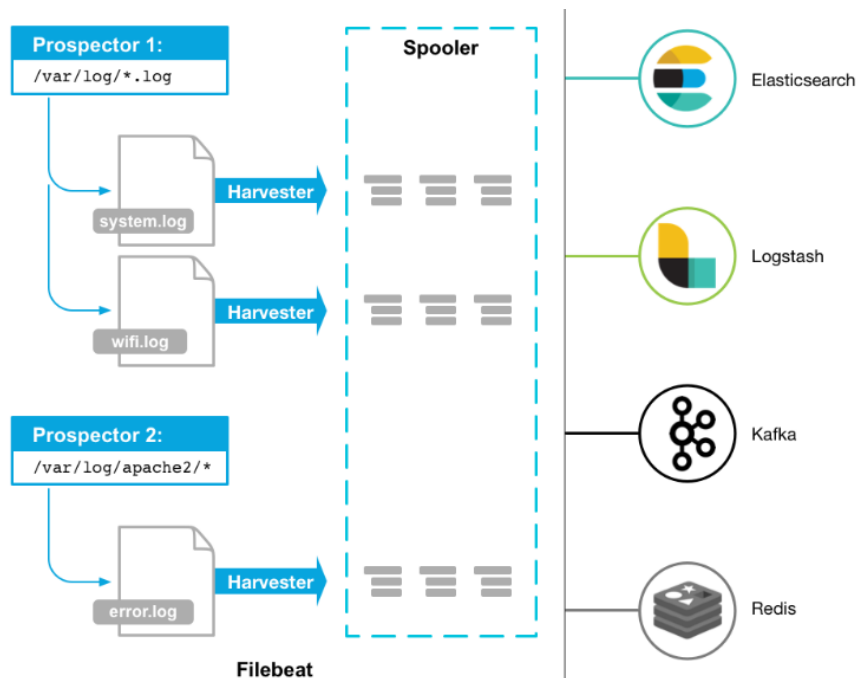
从之前的分析看，问题出在这一块的可能性不大，因为 redis 的性能并未达到瓶颈。

3.5 Filebeat

Filebeat 是本地文件的日志数据采集器。作为服务器上的代理安装，Filebeat 监视日志目录或特定日志文件，tail file，并将它们转发给 Elasticsearch 或 Logstash 进行索引、kafka 等。

Filebeat 由两个主要组件组成：prospector 和 harvester。这些组件一起工作来读取文件（tail file）并将事件数据发送到指定的输出

启动 Filebeat 时，它会启动一个或多个查找器，查看为日志文件指定的本地路径。对于 prospector 所在的每个日志文件，prospector 启动 harvester。每个 harvester 都会为新内容读取单个日志文件，并将新日志数据发送到 libbeat，后者将聚合事件并将聚合数据发送到为 Filebeat 配置的输出。



harvester 负责读取单个文件的内容。读取每个文件，并将内容发送到 the output

每个文件启动一个 harvester, harvester 负责打开和关闭文件，这意味着在运行时文件描述符保持打开状态

如果文件在读取时被删除或重命名，Filebeat 将继续读取文件。

这有副作用，即在 harvester 关闭之前，磁盘上的空间被保留。默认情况下，Filebeat 将文件保持打开状态，直到达到 close_inactive 状态

关闭 harvester 会产生以下结果：

- 1) 如果在 harvester 仍在读取文件时文件被删除，则关闭文件句柄，释放底层资源。
- 2) 文件的采集只会在 scan_frequency 过后重新开始。
- 3) 如果在 harvester 关闭的情况下移动或移除文件，则不会继续处理文件。

prospector 负责管理 harvester 并找到所有要读取的文件来源。

如果输入类型为日志，则查找器将查找路径匹配的所有文件，并为每个文件启动一个 harvester。

每个 prospector 都在自己的 Go 协程中运行。

那么 Filebeat 是如何记录文件状态的呢？filebeat 将文件状态记录在文件中（默认在 `/var/lib/filebeat/registry`）。此状态可以记住 Harvester 收集文件的偏移量。若连接不上输出设备，如 ES 等，filebeat 会记录发送前的最后一行，并再可以连接的时候继续发送。Filebeat 在运行的时候，Prospector 状态会被记录在内存中。Filebeat 重启的时候，利用 registry 记录的状态来进行重建，用来还原到重启之前的状态。每个 Prospector 会为每个找到的文件记录一个状态，对于每个文件，Filebeat 存储唯一标识符以检测文件是否先前被收集。

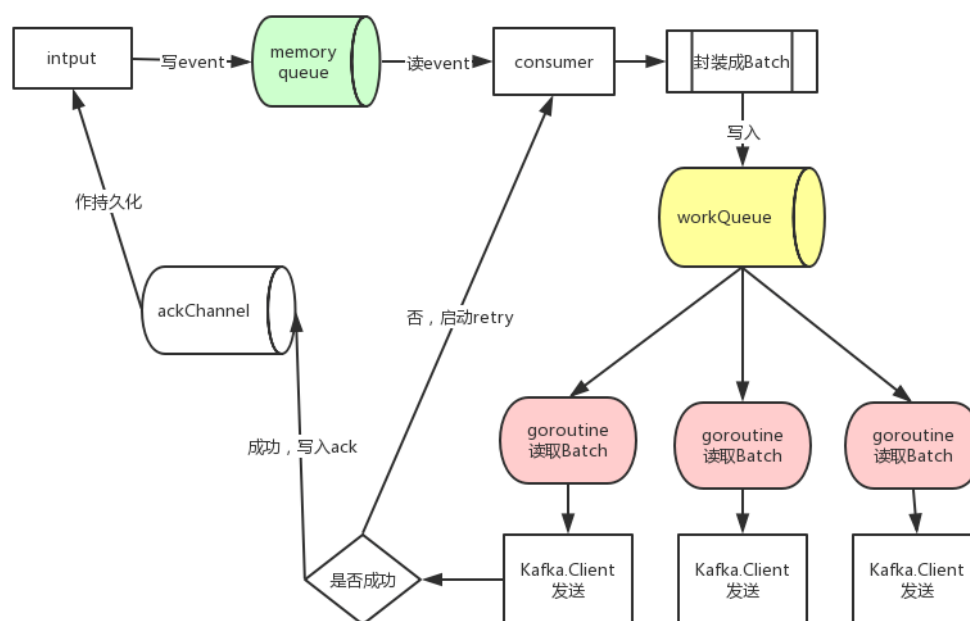
Filebeat 如何保证事件至少被输出一次？Filebeat 之所以能保证事件至少被传递到配置的输出一次，没有数据丢失，是因为 filebeat 将每个事件的传递状态保存在文件中。在未得到输出方确认时，filebeat 会尝试一直发送，直到得到回应。若 filebeat 在传输过程中被关闭，则不会再关闭之前确认所有事件。任何在 filebeat 关闭之前为确认的时间，都会在 filebeat 重启之后重新发送。这可确保至少发送一次，但有可能会重复。可通过设置 `shutdown_timeout` 参数来设置关闭之前的等待事件回应的时间（默认禁用）。

Filebeat retryer 结构体

```

17
18 package pipeline
19
20 import (
21     "sync"
22 )
23
24 // retryer is responsible for accepting and managing failed send attempts. It
25 // will also accept not yet published events from outputs being dynamically closed
26 // by the controller. Cancelled batches will be forwarded to the new workQueue,
27 // without updating the events retry counters.
28 // If too many batches (number of outputs/3) are stored in the retry buffer,
29 // will the consumer be paused, until some batches have been processed by some
30 // outputs.
31 type retryer struct {
32     logger    logger
33     observer  outputObserver
34
35     done chan struct{}
36
37     consumer interruptor
38
39     sig      chan retryerSignal
40     out      workQueue
41     in       retryQueue
42     doneWaiter sync.WaitGroup
43 }
44
```

重试示意图



日志中错误信息

```

[root@red-master ~]# kubectl logs dispatch-tpgadapter-2m7m -n dispatch -c filebeat > filebeaterror
[root@red-master ~]# cat filebeaterror
[root@red-master ~]# grep ERROR filebeaterror
2021-01-20T18:55:01.886+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T18:56:51.149+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T18:57:42.447+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T18:58:13.402+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T18:59:28.838+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T19:00:23.858+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T19:01:18.548+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T19:01:44.138+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T19:01:58.639+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T19:02:14.182+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T19:06:47.423+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again
2021-01-20T19:07:13.898+0800 ERROR log/input.go:460 Harvester could not be started on new file: /zap/session.log. Err: Error setting up harvester: Harvester setup failed. Unexpected file opening error: file info is not identical with opened file. Aborting ha
rvesting and retrying file later again



```

由此可见，Filebeat 出现积压时，retryer 会将需要重新发送的 events 再次写入 workQueue 中，重新进入发送流程，这个文件占用的空间就无法释放。因为 service 和 filebeat 通过共享目录的方式共同操作日志文件，而日志文件由 service 创建，所以长时间的积压就会导致文件持续无法释放，内存持续增长。展示的现象就是 service 内存持续增长，而 filebeat 内存正常。而从日志错误信息看，Harvester 处理能力不足也是造成日志积压的原因。



3.6 Zap

Zap 是 uber 开源的高性能日志库，面向高性能并且也确实做到了高性能。性能数据如下



Log a message and 10 fields:

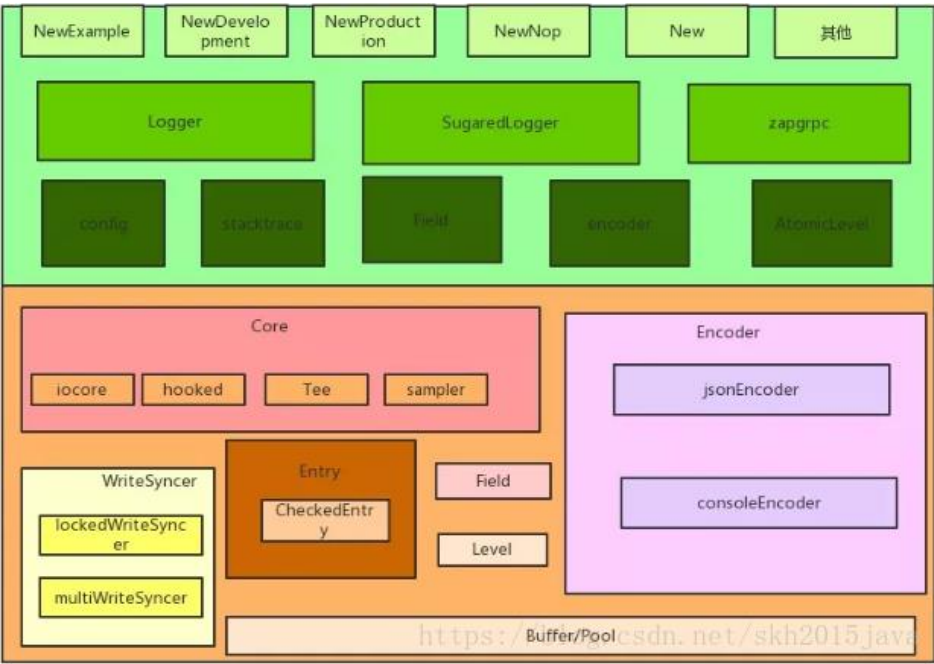
Package	Time	Time % to zap	Objects Allocated
 zap	862 ns/op	+0%	5 allocs/op
 zap (sugared)	1250 ns/op	+45%	11 allocs/op
zerolog	4021 ns/op	+366%	76 allocs/op
go-kit	4542 ns/op	+427%	105 allocs/op
apex/log	26785 ns/op	+3007%	115 allocs/op
logrus	29501 ns/op	+3322%	125 allocs/op
log15	29906 ns/op	+3369%	122 allocs/op

Log a message with a logger that already has 10 fields of context:

Package	Time	Time % to zap	Objects Allocated
 zap	126 ns/op	+0%	0 allocs/op
 zap (sugared)	187 ns/op	+48%	2 allocs/op
zerolog	88 ns/op	-30%	0 allocs/op
go-kit	5087 ns/op	+3937%	103 allocs/op
log15	18548 ns/op	+14621%	73 allocs/op
apex/log	26012 ns/op	+20544%	104 allocs/op
logrus	27236 ns/op	+21516%	113 allocs/op

Log a static string, without any context or `printf`-style templating:

Package	Time	Time % to zap	Objects Allocated
 zap	118 ns/op	+0%	0 allocs/op
 zap (sugared)	191 ns/op	+62%	2 allocs/op
zerolog	93 ns/op	-21%	0 allocs/op
go-kit	280 ns/op	+137%	11 allocs/op
standard library	499 ns/op	+323%	2 allocs/op
apex/log	1990 ns/op	+1586%	10 allocs/op
logrus	3129 ns/op	+2552%	24 allocs/op
log15	3887 ns/op	+3194%	23 allocs/op



Package	Synopsis
zapcore	zapcore 定义了低级接口，这些接口是 zap 所依赖的核心接口。接口的实现和依赖分离，这样最大化的降低了代码之间的耦合。而且可以直接对 zapcore 进行封装，便于二次开发与封装。
zapgrpc	grpc logger 的封装实现，便于 grpc go 用户添加 log。
internal/bufferpool & buffer	buffer 提供了 append field 功能，通过 append 把基础类型添加到 buffer 中。同时使用了 sync.Pool 提供的对象池技术,通过对象复用，减少内存分配。
internal/ztest & zaptest	zap test warp，提供了 mock 接口便于测试。zap 代码整体单元测试覆盖到了 98.9%，真的是非常惊人，这是十分良好的习惯，值得我们尊重和学习。

通过 zap 打印一条结构化的日志大致包含 5 个过程：

1. 分配日志 Entry: 创建整个结构体，此时虽然没有传参(fields)进来，但是 fields 参数其实创建了。
2. 检查级别，添加 core: 如果 logger 同时配置了 hook，则 hook 会在 core check 后把自己添加到 cores 中。
3. 根据选项添加 caller info 和 stack 信息: 只有大于等于级别的日志才会创建 checked entry。
4. Encoder 对 checked entry 进行编码: 创建最终的 byte slice，将 fields 通过自己的编码方式(append)编码成目标串。
5. Write 编码后的目标串，并对剩余的 core 执行操作，hook 也会在这时被调用

通过对 zap 的了解，性能应该不是问题。但是流量洪峰期日志生产速度过快，如果滚动文件数量太少，可能会导致部分日志丢失，因此需要调整滚动最大文件数。

4 结论

1、日志丢失：

原因：滚动更新设置的 `MaxBackups` 数为 10，流量洪峰时日志生产过快，导致部分未进入队列就已经删除的日志丢失。

解决方法：使用 `zap` 接口，同时调大 `MaxBackups`。

2、延迟消费

原因：`harvester_buffer_size` 使用默认值，导致每个 `harvester` 监控文件时，使用的 `buffer` 的太小，当日志较大时，通道出现阻塞。同时 `output.redis` 中 `worker` 使用默认值，处于单进程处理模式，输出能力不足。综合上述二者影响，日志积压在本地，等待 `filebeat` 的 `retry` 机制进行重发，因此导致了日志延迟发送，从 `kibana` 端上看到的现象就是延迟消费，从而告警出现延迟。

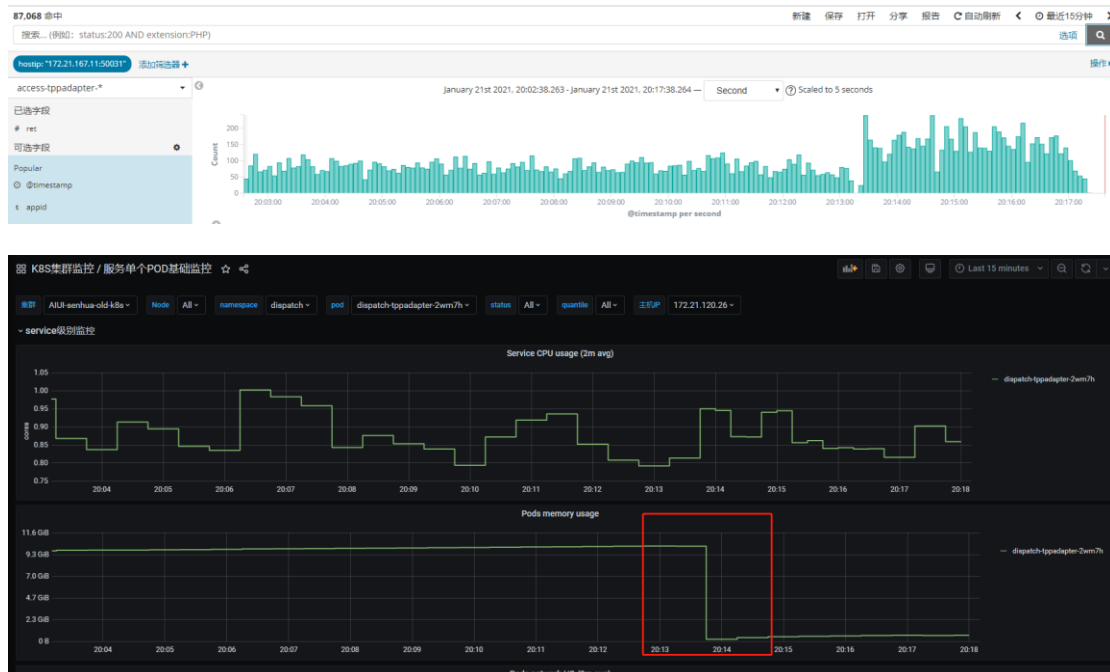
解决方法：调大 `harvester_buffer_size`，调大 `worker` 值。

3、内存激增

原因：`Filebeat` 出现积压时，`retryer` 会将需要重新发送的 `events` 再次写入 `workQueue` 中，重新进入发送流程，这个文件占用的空间就无法释放。因为 `service` 和 `filebeat` 通过共享目录的方式共同操作日志文件，而日志文件由 `service` 创建，所以长时间的积压就会导致文件持续无法释放，内存持续增长。展示的现象就是 `service` 内存持续增长，而 `filebeat` 内存正常。

解决方法：调大 `harvester_buffer_size`，调大 `worker` 值。

优化后的效果很显著，`Kibana` 中日志大幅增加，内存从 10G 下降至 300M。



5 经验总结

- 大胆猜测，小心验证（提出假设，倒推验证）。
- 化整为零，各个击破（拆分问题模块，各个子模块深入分析）。
- 磨刀不误砍柴工（编写问题复现或者验证工具是非常重要的手段）。
- 多一个思路，多一个可能（尝试从不同角度分析问题，不能钻牛角尖）。
- 了解它，才能战胜它（了解工作原理有助于问题根因的查找）。
- 坚持才能胜利。