kafka Index service 最佳实践(Druid 0.9.1.1)

TalkingData
移动 · 数据 · 价值

- 需求场景
- 选择过程
- Kafka Indexing Service介绍
- 性能测试
- 碰到的坑

需求场景

- 实时查询最新数据
- 自动处理延迟到来的非实时数据
- 分布式可扩展

选择过程

- Tranquility ×

窗口外数据丢弃

数据量过大时OOM，task仍然pending

- Kafka Indexing Service √

支持实时查询

按时间分segment,非实追加到对应时间的segment

通过【equalDistribution】把Peon分配到【 Middle Managers】上实现分布式

加大对应kafka的topic的partition数量，加大taskCount的值，产生更多的Peon

kill peon、kill overload、kill middleManager、生成segment过程中kill进程 几种情况下程序都能不丢失数据

Kafka Indexing Service介绍

- 官方文档地址

  http://druid.io/docs/0.9.1.1/development/extensions-core/kafka-ingestion.html

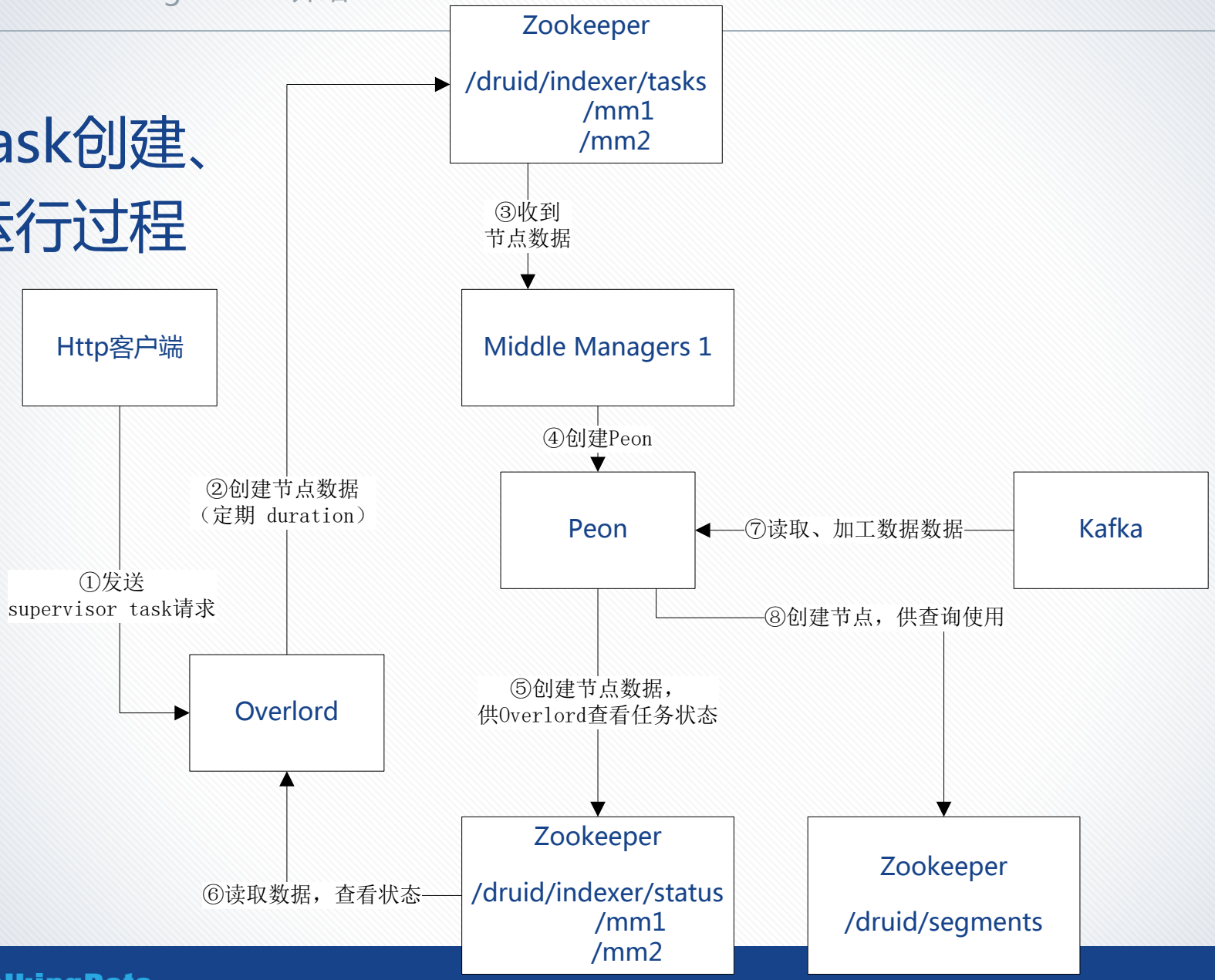
- 主要注意参数说明

  replicas设置为1，加大不会起到replica的作用

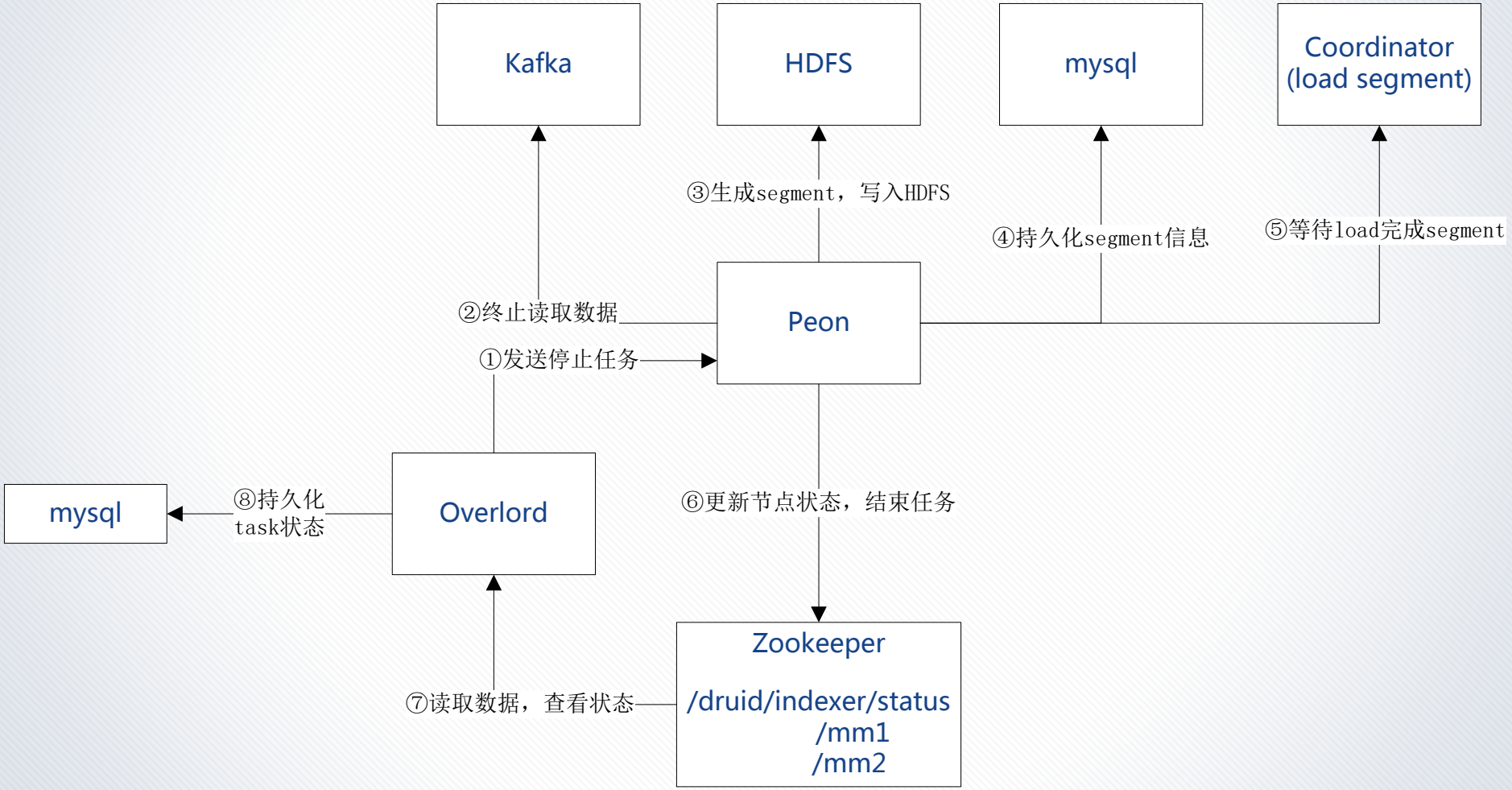
  taskDuration：目前使用PT1H，即一小时

  completionTimeout：设置与taskDuration一致

Task创建、运行过程



Task正常 终止过程



性能测试

性能测试基于Tracking从【2016-9-13】到【2016-10-8】的数据，共25天。

测试分为【查询全部】、【查询千万级别】、【查询百万级别】三个部分

查询全部数据性能如下：

序号	查询类型		所用时间

1	count	[{
		"timestamp" : "1970-01-01T00:00:00.000Z",
		"result" : {
		"rows" : 8533145983,
		"loginCount" : 626537993
		}
		}]
			冷数据查询：
			0m 19.296s
			
			热数据查询：
			0m0.174s

2	hyperUnique	[{
		"timestamp" : "1970-01-01T00:00:00.000Z",
		"result" : {
		"userHyperUnique" : 9.807841445878088E7
		}
		}]
			冷数据查询：
			0m22.376s
			
			热数据查询：
			
			0m0.125s

3	atomCube	[{ "test_size" : 97790882 }]	冷数据查询： 7m6.809s 热数据查询： 2m34.292s

查询千万级别性能如下：

序号	查询类型		所用时间

1	count	[{
		"timestamp" : "1970-01-01T00:00:00.000Z",
		"result" : {
		"rows" : 315711057,
		"loginCount" : 11705279
		}
		}]
			冷数据查询：
			0m3.919s
			
			热数据查询：
			0m0.230s

2	hyperUnique	[{
		"timestamp" : "1970-01-01T00:00:00.000Z",
		"result" : {
		"userHyperUnique" : 1.1680285387818966E7
		}
		}]
			冷数据查询：
			0m19.552s
			
			热数据查询：
			0m0.201s

3	atomCube	[{ "test_size" : 11687873 }]	冷数据查询： 1m2.711s 热数据查询： 0m11.243s

查询百万级别性能如下：

序号	查询类型		所用时间

1	count	[{
		"timestamp" : "1970-01-01T00:00:00.000Z",
		"result" : {
		"rows" : 18243951,
		"loginCount" : 1022415
		}
		}]
			冷数据查询：
			0m2.569s
			
			热数据查询：
			0m0.071s

2	hyperUnique	[{
		"timestamp" : "1970-01-01T00:00:00.000Z",
		"result" : {
		"userHyperUnique" : 105181.35039583383
		}
		}]
			冷数据查询：
			0m21.085s
			
			热数据查询：
			0m0.100s

3	atomCube	[{ "test_size" : 102111 }]	冷数据查询： 0m32.763ss 热数据查询： 0m0.578s

碰到的坑

historical节点宕掉

问题描述：
数据导入持续进行一段时间之后，个别hisotrical节点报OOM、进而导致其他节点压力增大、最终导致historical节点全部宕掉。

结论:
操作系统默认的max_map_file为65530 而测试数据特别离散、导致segment数量增长快速。historical节点会把每个segment通过mmap映射到虚拟内存中。按照操作系统的配置一个进程最多mmap 65530个文件，超过这个文件就会报上面的错误

解决方案
执行命令 sysctl -w vm.max_map_count=655350，加大参数值。

segment handoff假死

问题描述：

所有的数据导入任务全部卡死等待segment hand off，coordinator日志不滚动、CPU占用率超高

结论：

由于大批量的segment merge完成提交了非常多的segment，导致coordinator分配segment计算cost的时间过长，block住了其他的操作

采用的基于cost策略（CostBalancerStrategyFactory）计算负载，由于shard数量太多，计算时间太长，一直未能生成新的shard,导致task不结束。

解决方案：

1.暂时采用随机策略（RandomBalancerStrategy）计算负载；（需要修改代码）DruidCoordinator#690(BalancerStrategyFactory factory = new RandomBalancerStrategyFactory())

2.合并shard，减少分片。

Kafka index Task不生成segment、任务直接SUCCESS

问题描述：
pen日志中出现

问题原因：
completionTimeout设置的过小

问题解决：
completionTimeout设置要与taskDuration一致。
例子如下：

```
"taskDuration": "PT1H",
"completionTimeout": "PT1H"
```

Druid supervisor任务一直运行

问题描述：

发起supervisor shutdown停止后，supervisor已经正常停止，但是supervisor启动的tasks一直在运行。

问题分析：

supervisor会按taskduration启动新的任务，如果在新任务启动不久后发起的shutdown请求，新启动的任务不会被终止。本次案例osuer的taskduration是PT1H,上次task启动时间是2017-01-24T08:07:56.672+08:00，发起新的任务的时间是2017-01-24T09:08:10.951+08:00,shutdown的时间是2017-01-24T09:08:10.963+08:00。在新任务启动12ms后发起了shutdown请求，导致新任务不能终止。

问题解决：

修改supervisor机制的成本较高，因此通过判断supervisor的任务的启动时间和taskduration到期时间都要大于120s，才能shutdown supervisor，否则等待满足条件再shutdown supervisor。

时区问题

时区需要按照【-Duser.timezone=GMT+8】格式设置，其他格式（ +8，Asia/Shanghai ）运行会报错。

线上运行【Middle Manager】的配置如下图：

```
druid.service=druid/middleManager
druid.port=8091
druid.host=bj-xg-app-druid-001.tendcloud.com

# Number of tasks per middleManager
druid.worker.capacity=40

# Task launch parameters
druid.indexer.runner.javaOpts=-server -Xms1g  -Xmx6g -XX:MaxDirectMemorySize=4g -Duser.timezone=GMT+8 -Dfile.encoding=UTF-8 -Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager
druid.indexer.task.baseTaskDir=var/druid/task

# HTTP server threads
druid.server.http.numThreads=25

# Processing threads and buffers
druid.processing.buffer.sizeBytes=536870912
druid.processing.numThreads=2

# Hadoop indexing
druid.indexer.task.hadoopWorkingPath=var/druid/hadoop-tmp
druid.indexer.task.defaultHadoopCoordinates=["org.apache.hadoop:hadoop-client:2.6.0-cdh5.4.5"]
```

- 修复druid cluster第一次启动bug

参照：Kafka supervisor kills unresponsive tasks too quickly refer to [https://github.com/druid-io/druid/issues/3276]

- 解决NoneShardSpec分区问题

问题描述：

hadoop merge后，总条数 <= targetPartitionSize时，把ShardSpec置为了NoneShardSpec，导致流数据时无法确定分区数，因此不能确定SegmentIdentifier，返回null,从而导致程序跑出 Could not allocate segment for row with timestamp 异常。

解决方案：

把【new NoneShardSpec()】替换为【NumberedShardSpec(0, 0)】。

- 修复transaction failure的bug

参照：

https://github.com/druid-io/druid/pull/3728

https://github.com/druid-io/druid/commit/a8069f2441b6c11f11306b3bc5b34cb9841098ff

TalkingData

THANKS!

官网 / www.talkingdata.net
微博 / @TalkingData
微信 / TalkingData
服务支持 / support@tendcloud.com