.Prometheus之存储及WAL

一、整体介绍

Prometheus 2.x 采用自定义的存储格式将样本数据保存在本地磁盘当中。如下所示，按照两个小时（最少时间）为一个时间窗口，将两小时内产生的数据存储在一个块(Block)中，每一个块中包含该时间窗口内的所有样本数据(chunks)，元数据文件(meta.json)以及索引文件(index)。

```
[mingming.chen@m162p65 data]$ tree
.
├── 01E2MA5GDWMP69GVBVY1W5AF1X
│   ├── chunks              # 保存压缩后的时序数据，每个chunks大小为512M，超过会生成新的chunks
│   │   └── 000001
│   ├── index               # chunks中的偏移位置
│   ├── meta.json           # 记录block块元信息，比如 样本的起始时间、chunks数量和数据量大小等
│   └── tombstones          # 通过API方式对数据进行软删除,将删除记录存储在此处（API的删除方式，并不是立即将数据从
chunks文件中移除）
├── 01E2MH175FV0JFB7EGCRZCX8NF
│   ├── chunks
│   │   └── 000001
│   ├── index
│   ├── meta.json
│   └── tombstones
├── 01E2MQWYDFQAXXPB3M1HK6T20A
│   ├── chunks
│   │   └── 000001
│   ├── index
│   ├── meta.json
│   └── tombstones
├── lock
├── queries.active
└── wal                     #防止数据丢失(数据收集上来暂时是存放在内存中,wal记录了这些信息)
    ├── 00000366            #每个数据段最大为128M，存储默认存储两个小时的数据量。
    ├── 00000367
    ├── 00000368
    ├── 00000369
    └── checkpoint.000365
        └── 00000000
```

二、block

TSDB将存储的监控数据按照时间分成多个block存储,默认最小的block保存时间为2h,后台程序还会将小块合并成大块，减少内存中block的数量，便于索引查找数据，可以通过meta.json查看，可以看到01E2MA5GDWMP69GVBVY1W5AF1X被压缩1次，source有3个block，那么2*3=6小时的数据量。

关于**block**压缩：

1. 最初的两个小时的块最终会在后台压缩为更长时间的块；
2. 压缩的最大时间块为数据保留时间的10%或者31天，取两者的较小者。

2.1 head block

1.head block中的数据是被存储在内存中的并且可以被任意修改;
2.head block和后续的block初始设定保存2h数据，当head block超过3h时，会被拆分为2h+1h，2h block会变成只读块写入磁盘.(通过观察服务器上prometheus存储目录，每次压缩合并小块时间都比块内部时间多三个小时,为head block)

三、WAL(Write-ahead logging, 预写日志)

Prometheus为了防止服务器在崩溃后重新启动时，有些丢失暂存在内存中的还未被写入磁盘的监控数据，引入了write-ahead-log (WAL)机制来防止崩溃。WAL被分割成默认大小为128M的文件段（segment）,这些文件包含尚未压缩的原始数据，因此比常规块文件大得多。

3.1 数据流向

prometheus将周期性采集到的数据通过Add接口添加到head block，但是这些数据暂时没有持久化，TSDB通过WAL将数据保存到磁盘上(保存的数据没有压缩，占用内存较大)，当出现宕机，启动多协程读取WAL，恢复数据。

四、和存储相关的启动参数

五、总结

需要解决的几个问题：

1.远程存储节点长时间挂掉（默认blocK大小为2小时，实际大于六小时，prometheus2.15经测试验证非官方文档说的两个小时），刷盘到prometheus的数据库中的数据还能不能同步到远程?

解答：

1.根据以上内容和3.远程写参数优化可知，prometheus本地存储和远程存储并无影响。因为远程存储是通过将WAL中的数据缓存到多个内存队列（shards）中，然后写到远程存储设备，其直接与WAL打交道。而prometheus只是用WAL来防止数据丢失，其存储的一系列动作都与WAL没关系。所以当内存中缓存的数据达到刷盘的阈值，WAL中没有写到远程存储的数据就会丢失，当重新启动远程存储服务，原来那部分没有写入远程存储服务的数据已经丢失，只能从最新的数据开始写入远程存储。