浅析下开源时序数据库VictoriaMetrics的存储机制

关注他

20 人赞同了该文章

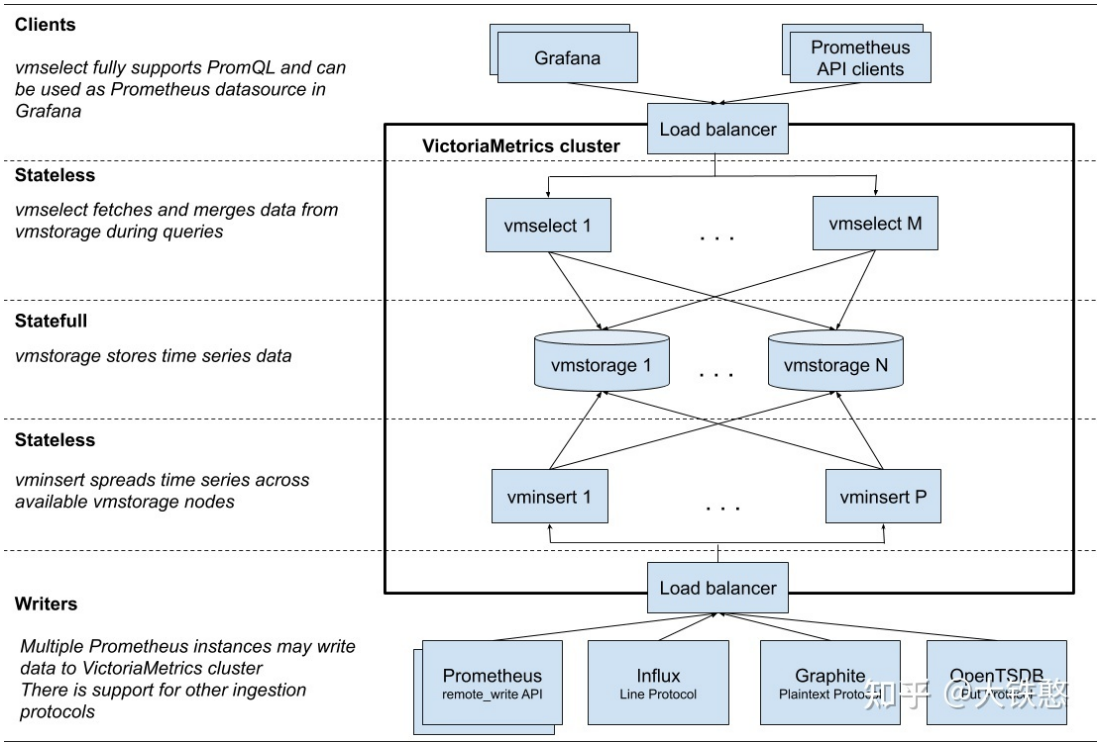
VictoriaMetrics是一个快速高效且可扩展的监控解决方案和时序数据库，可以作为Prometheus的长期远端存储，具备的特性有：
- 支持prometheus查询api，同时实现了一个metricsql 查询语言
- 支持全局查询视图，支持多prometheus 实例写数据到VictoriaMetrics，然后提供一个统一的查询
- 支持集群
- 高性能
- 支持多种协议，包括influxdb line协议，prometheus metrics，graphite ，prometheus远端写api，opentsdb http协议等
- 高压缩比

本文主要分析下VictoriaMetrics的存储机制，包括整体架构、数据模型、磁盘目录、文件格式等部分，对应的源码版本号为v1.45.0。

1.整体架构

VictoriaMetrics的集群主要由vmstorage、vminsert、vmselect等三部分组成，其中，vmstorage 负责提供数据存储服务; vminsert 是数据存储 vmstorage 的代理，使用一致性hash算法进行写入分片； vmselect 负责数据查询，根据输入的查询条件从 vmstorage 中查询数据。

VictoriaMetrics的这个三个组件每个组件都可以单独进行扩展，并运行在大多数合适软件上。vmstorage采用shared-nothing架构，优点是vmstorage的节点相互之间无感知，相互之间无需通信，不共享任何数据，增加了集群的可用性、简化了集群的运维和集群的扩展。



另外，VictoriaMetrics集群支持多个隔离的租户特性，又名命名空间。租户通过accountID (或者accountID:projectID) 来标识，这些ID放在请求URL中。有关VictoriaMetrics租户的一些事实：

- 每个accountID和projectID由[0 .. 2 ^ 32）范围内的任意32位整数标识。如果缺少projectID，则会将其自动分配为0。预计有关租户的其他信息（例如身份验证令牌，租户名称，限制，计费等）将存储在单独的关系数据库中。此数据库必须由位于VictoriaMetrics群集前面的单独服务（例如vmauth）管理。
- 将第一个数据点写入给定租户时，将自动创建租户。
- 所有租户的数据平均分布在可用的vmstorage节点之间。当不同的租户具有不同的数据量和不同的查询负载时，这可以保证在vmstorage节点之间平均负载。

整体上来说，VictoriaMetrics支持多租户，但租户的信息需要使用额外的关系型数据库来存储，且VictoriaMetrics不支持在单个请求中查询多个租户。

2.数据模型

开门见山，**VictoriaMetrics采用的数据模型是单值模型，且只支持浮

点数指标。那么到底什么是单值模型呢？目前，常见的时序数据库的数据模型，主要分成单值模型和多值模型。这里简单说明下单值模型和多值模型，整体上，可以认为单值模型是多值模型的一个特例。

单值模型是根据**业务指标数据**建模，按照单个指标的细粒度进行数据使用和逻辑存储，如下图所示，一行数据只有一个指标值，即value列。目前采用单值模型的时序数据库，有OpenTSDB、 KairosDB、Prometheus等。

metric	appName	region	timestamp	value
cpu	tsdb	shanghai	1613706029	0.5
cpu	tsdb	shanghai	1613706039	0.4
io	tsdb	hangzhou	1613706049	0.7
io	tsdb	hangzhou	1613706059	0.8

多值的模型则是针对**数据源**建模，我们每一行数据针对的是一个数据源，它的被测量的多个指标在同一列上，如下图所示，一行数据有多个指标值，即有cpu和io两列。目前采用多值模型的时序数据库，有InfluxDB、TimescaleDB等。

mesurement	appName	region	timestamp	cpu	io
monitor	tsdb	shanghai	1613706029	0.5	0.7
monitor	tsdb	shanghai	1613706039	0.5	0.5

3.磁盘目录

VictoriaMetrics的根目录下主要包括4个目录或文件，如下图所示。其中，最主要的是数据目录data和索引目录indexdb，flock.lock文件为文件锁文件，用于VictoriaMetrics进程锁住文件，不允许别的进程进行修改目录或文件。

```
├── data
├── flock.lock
├── indexdb
└── snapshots
```


3.1 数据目录

数据目录data的具体结构，如下图所示，在图中使用红色文字，对主要目录或文件做了简单说明，其中最主要的是**big**目录和**small**目录，这两个目录的结构是一样的。其中，在VictoriaMetrics中，使用table来表示的数据或者索引的根目录，而实际上VictoriaMetrics中没有实际的表table级别目录。

在small目录下，以月为单位不断生成partition目录，比如上图中的2020_11目录，对应的实现在源码lib/storage/partition.go中。partition目录包括part目录、临时目录tmp、 事务目录txn等三个目录。

内存中的数据每刷盘一次就会生成一个part目录，如上图中的"708_354_20201103102134.255_20201103102149.255_1643F83394CA24A7"，目录名中的708表示这个目录下包含的数据行数

rowsCount, 目录名中的354表示这个目录中包含的数据块数blocksCount, 20201103102134.255表示目录中包含的数据的最小时间戳，20201103102149.255表示目录中包含的数据的最大时间戳，1643F83394CA24A7是生成这个目录时的系统纳秒时间戳的16进制表示，对应的实现逻辑在源码lib/storage/part.go中；

看到这里，可能会有一些疑问？比如为何要分成big和small目录, 或者说big目录和small中的数据关系是什么？ 这个需要从VictoriaMetrics的compaction机制讲起。

在VictoriaMetrics中，small目录和big目录都会周期性检查，是否需要做part的合并。VictoriaMetrics默认每个10ms检查一次partition目录下的part是否需要做merge。如果检查出有满足merge条件的parts，则这些parts合并为一个part。如果没有满足条件的part进行merge，则以10ms为基进行指数休眠，最大休眠时间为10s。

VictoriaMetrics在写数据时，先写入在small目录下的相应partition目录下面的，small目录下的每个partition最多256个part。VictoriaMetrics在Compaction时，默认一次最多合并15个part，且small目录下的part最多包含1000W行数据，即1000W个数据点。因此，当一次待合并的parts中包含的行数超过1000W行时，其合并的输出路径为big目录下的同名partition目录下。

因此，big目录下的数据由small目录下的数据在后台compaction时合并生成的。 那么为什么要分成big目录和small目录呢?

这个主要是从磁盘空间占用上来考虑的。时序数据经常读取最近写入的数据，较少读历史数据。而且，时序数据的数据量比较大，通常会使用压缩算法进行压缩。

数据进行压缩后，读取时需要解压，采用不同级别的压缩压缩算法其解压速度不一样，通常压缩级别越高，其解压速度越慢。在VictoriaMetrics在时序压缩的基础上，又采用了ZSTD这个通用压缩算法进一步压缩了数据，以提高压缩率。在small目录中的part数据，采用的是低级别的ZSTD，而big目录下的数据，采用的是高级别的

ZSTD。

因此，VictoriaMetrics分成small目录和big目录，主要是兼顾近期数据的读取和历史数据的压缩率。

3.2 索引目录

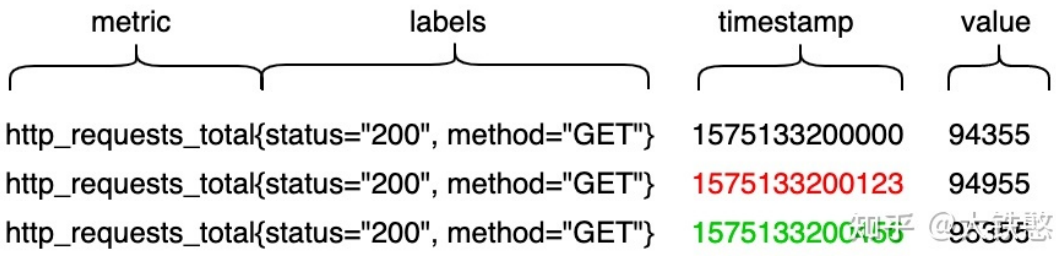
索引目录indexdb的具体结构，如下图所示，在图中使用红色文字，对主要目录或文件做了简单说明。与数据目录不同的是，indexdb目录下由多个table目录，每个table目录代表一个完整的自治的索引，每个table目录下，又有多个不同的part目录，part命名方式比较简单，有文件包含的item数itemsCount和block数blocksCount, 以及根据系统纳秒时间戳自增生成的mergeIdx的16进制表示。


```
├── data
├── flock.lock
├── indexdb
│   ├── 1643F4F397B53DED  tableName #1，命名方式，根据系统的纳秒时间戳自增生成，并格式化16进制形式
│   │   ├── converted-to-v1.28.0
│   │   ├── flock.lock
│   │   ├── tmp
│   │   └── txn
│   ├── 1643F4F397B53DEE  tableName #2
│   │   ├── 1898_2_1643F4F39CF51C01  索引part, 命名方式：itemCount_blocksCount_mergeIdx
│   │   │   ├── index.bin
│   │   │   ├── items.bin
│   │   │   ├── lens.bin
│   │   │   ├── metadata.json
│   │   │   └── metaindex.bin
│   │   ├── 216_1_1643F4F39CF51C02
│   │   │   ├── index.bin
│   │   │   ├── items.bin
│   │   │   ├── lens.bin
│   │   │   ├── metadata.json
│   │   │   └── metaindex.bin
│   │   ├── converted-to-v1.28.0  历史版本兼容
│   │   ├── flock.lock
│   │   ├── tmp  作用与数据文件目录下的tmp相同
│   │   └── txn  作用与数据文件目录下的txn相同
│   └── snapshots
└── snapshots
```

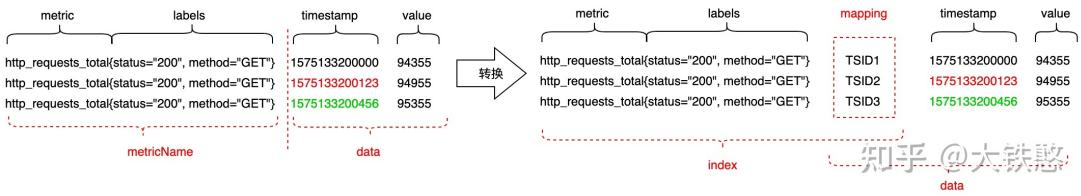

indexdb下面的形如"1643F4F397B53DEE"是怎么生成的，或者什么时候切换新的目录写索引的呢？VictoriaMetrics会根据用户设置的数据保留周期retention来定期滚动索引目录，当前一个索引目录的保留时间到了，就会切换一个新的目录，重新生成索引。

4. 文件格式

在介绍具体的文件格式之前，不得不提下VictoriaMetrics对于写入数据的处理过程。下图是VictoriaMetrics支持的Prometheus协议的一个写入示例。



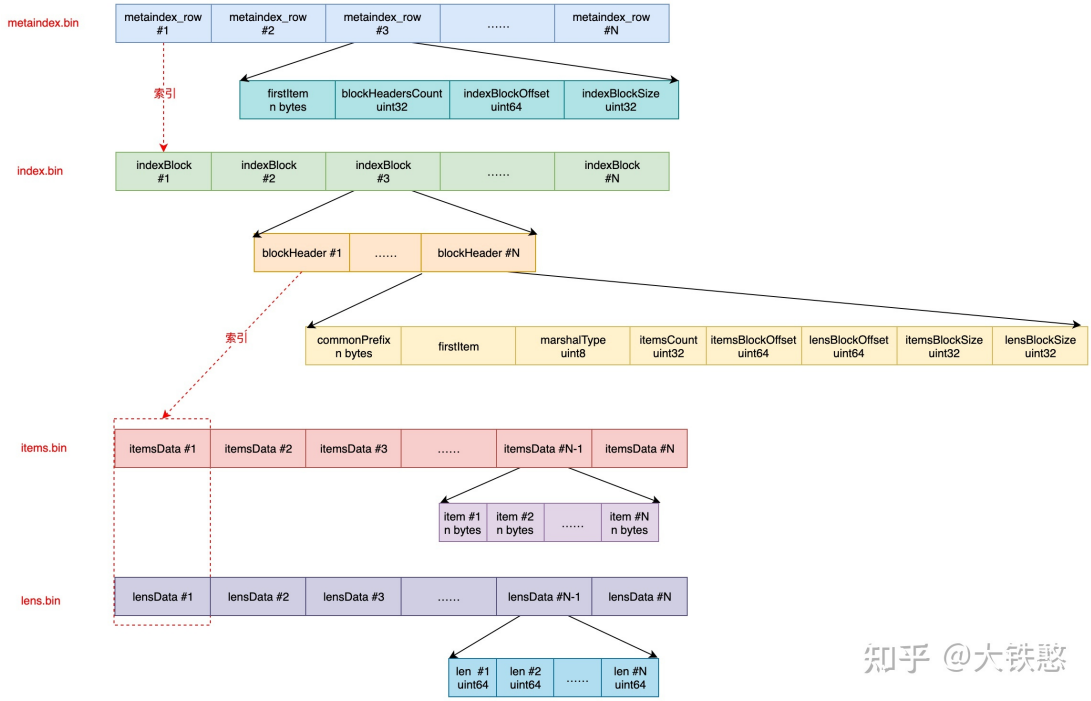
VictoriaMetrics在接受到写入请求时，会对请求中包含的时序数据做转换处理，如下图所示。首先先根据包含metric和labels的MetricName生成一个唯一标识TSID，然后metric + labels + TSID作为索引|index，TSID + timestamp + value作为数据data，最后索引|index和数据data分别进行存储和检索。



因此，VictoriaMetrics的数据整体上分成索引和数据两个部分，因此文件格式整体上会有两个部分。其中，索引部分主要是用于支持按照label或者tag进行多维检索。与大多数时序数据库的数据组织方式一样，比如InfluxDB、Prometheus、OpenTSDB等，VictoriaMetrics也是按时间线来组织数据的，即数据存储时，先将数据按TSID进行分组，然后每个TSID的包含的数据点各自使用列式压缩存储。

4.1 索引文件

VictoriaMetrics每次内存Flush或者后台Merge时生成的索引part，主要包含metaindex.bin、index.bin、lens.bin、items.bin等4个文件。这四个文件的关系如下图所示, metaindex.bin文件通过metaindex_row索引index.bin文件，index.bin文件通过indexBlock中的blockHeader同时索引lens.bin文件和items.bin文件。



metaindex.bin文件中，包含一系列的metaindex_row， 每个metaindex_row中包含最小项firstItem、索引块包含的块头部数blockHeadersCount、索引块偏移indexBlockOffset、索引块大小indexBlockSize。

- metaindex_row在文件中的位置按照firstItem的大小的字典序排序存储，以支持二分检索；
- metaindex.bin文件使用ZSTD进行压缩；
- metaindex.bin文件中的内容在part打开时，会全部读出加载至内存中，以加速查询过滤；
- metaindex_row包含的firstItem为其索引的IndexBlock中所有blockHeader中的字典序最小的firstItem；
- 查找时根据firstItem进行二分检索；

index.bin文件中，包含一系列的indexBlock, 每个indexBlock又包含一系列blockHeader，每个blockHeader的包含item的公共前缀commonPrefix、最小项firstItem、itemsData的序列化类型marshalType、itemsData包含的item数、item块的偏移itemsBlockOffset等内容。

- 每个indexBlock使用ZSTD压缩算法进行压缩；
- 在indexBlock中查找时，根据firstItem进行二分检索blockHeader；

items.bin文件中，包含一系列的itemsData, 每个itemsData又包含一系列的item。

- itemsData会根据情况是否使用ZTSD压缩，当item个数小于2时，或者itemsData的长度小于64字节时，不压缩；当itemsData使用ZSTD压缩后的大小，大于原始itemsData的0.9倍时，则不压缩，否则使用ZSTD算法进行压缩。
- 每个item在存储时，去掉了blockHeader中的公共前缀commonPrefix以提高压缩率。

lens.bin文件中，包含一系列的lensData, 每个lensData又包含一系列8字节的长度len， 长度len标识items.bin文件中对应item的长度。在读取或者需要解析itemsData中的item时，先要读取对应的lensData中对应的长度len。 当itemsData进行压缩时，lensData会先使用异或算法进行压缩，然后再使用ZSTD算法进一步压缩。

VictoriaMetrics索引文件都是围绕着item来组织的，那么item的结构是什么样子的？或者item的种类有哪些？ 在VictoriaMetrics中item的整体上是一个KeyValue结构的字节数组，共计有7种类型，每种类型的item通过固定前缀来区分，前缀类型如下图所示。

```
const (
        // Prefix for MetricName->TSID entries.
        nsPrefixMetricNameToTSID = 0

        // Prefix for Tag->MetricID entries.
        nsPrefixTagToMetricIDs = 1

        // Prefix for MetricID->TSID entries.
        nsPrefixMetricIDToTSID = 2

        // Prefix for MetricID->MetricName entries.
        nsPrefixMetricIDToMetricName = 3

        // Prefix for deleted MetricID entries.
        nsPrefixDeletedMetricID = 4

        // Prefix for Date->MetricID entries.
        nsPrefixDateToMetricID = 5

        // Prefix for (Date,Tag)->MetricID entries.
        nsPrefixDateTagToMetricIDs = 6
)
```


VictoriaMetrics是怎么基于item支持tag多维检索的呢？ 这里就不得不提下VictoriaMetrics的TSID的生成过程。

VictoriaMetrics的MetricName的结构如下所示，包含MetricGroup字节数组和Tag数组，其中，MetricGroup是可选的，每个Tag由Key和Value等字节数组构成。

```
// MetricName reperesents a metric name.
type MetricName struct {
    MetricGroup []byte

    // Tags are optional. They must be sorted by tag Key for canonical view.
    // Use sortTags method.
    Tags []Tag
}
```

```
// Tag represents a (key, value) tag for metric.
type Tag struct {
    Key   []byte
    Value []byte
}
```

VictoriaMetrics的TSID的结构如下所示，包含MetricGroupID，JobID，InstanceID，MetricID等四个字段，其中除了MetricID外，其他三个字段都是可选的。这个几个ID的生成方法如下：

- metricGroupID根据MetricName中的MetricGroup使用xxhash的sum64算法生成。
- JobID和InstanceID分别由MetricName中的tag的第一个tag和第二个tag使用xxhash的sum64算法生成。为什么使用第一个tag和第二个tag？这是因为VictoriaMetrics在写入时，将写入请求中的JobID和InstanceID放在了Tag数组的第一个和第二个位置。
- MetricID，使用VictoriaMetrics进程启动时的系统纳秒时间戳自增生成。

```
// TSID is unique id for a time series.
// Time series blocks are sorted by TSID.
// All the fields except MetricID are optional.
type TSID struct {
    // MetricGroupID is the id of metric group.
    MetricGroupID uint64

    // JobID is the id of an individual job (aka service).
    // JobID must be unique.
    JobID uint32

    // InstanceID is the id of an instance (aka process).
    // InstanceID must be unique.
    InstanceID uint32

    // MetricID is the unique id of the metric (time series).
    // All the other TSID fields may be obtained by MetricID.
    MetricID uint64
}
```


因为TSID中除了MetricID外，其他字段都是可选的，因此TSID中可以始终作为有效信息的只有metricID，因此VictoriaMetrics的在构建tag到TSID的字典过程中，是直接存储的tag到metricID的字典。

以写入http_requests_total{status="200", method="GET"}为例，则MetricName为http_requests_total{status="200", method="GET"}， 假设生成的TSID为{metricGroupID=0, jobID=0, instanceID=0, metricID=51106185174286}，则VictoriaMetrics在写入时就构建了如下几种类型的索引|item，其他类型的索引|item是在后台或者查询时构建的。

- metricName -> TSID, 即http_requests_total{status="200",

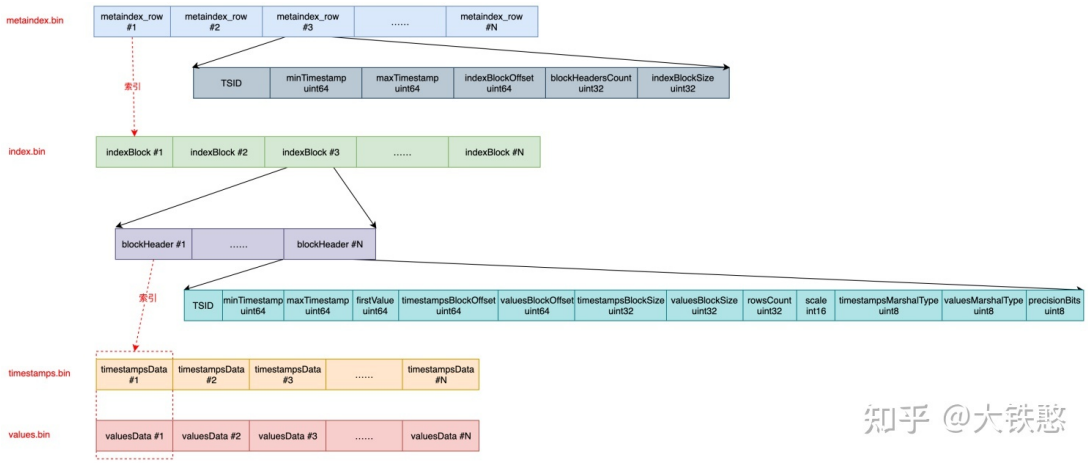
method="GET"} -> {metricGroupID=0, jobID=0, instanceID=0, metricID=51106185174286}
- metricID -> metricName，即51106185174286 -> http_requests_total{status="200", method="GET"}
- metricID -> TSID，即51106185174286 -> {metricGroupID=0, jobID=0, instanceID=0, metricID=51106185174286}
- tag -> metricID，即 status="200" -> 51106185174286, method="GET" -> 51106185174286, "" = http_requests_total -> 51106185174286

有了这些索引的item后，就可以支持基于tag的多维检索了，在当给定查询条件http_requests_total{status="200"}时，VictoriaMetrics先根据给定的tag条件，找出每个tag的metricID列表，然后求所有tag的metricID列表的交集，然后根据交集中的metricID，再到索引文件中检索出TSID，根据TSID就可以到数据文件中查询数据了，在返回结果之前，再根据TSID中的metricID，到索引文件中检索出对应的写入时的原始MetircName。

但是由于VictoriaMetrics的tag到metricID的字典，没有将相同tag的所有metricID放在一起存储，在检索时，一个tag可能需要查询多次才能得到完整的metricID列表。另外查询出metricID后，还要再到索引文件中去检索TSID才能去数据文件查询数据，又增加了一次IO开销。这样来看的话，VictoriaMetrics的索引文件在检索时，如果命中的时间线比较多的情况下，其IO开销会比较大，查询延迟也会比较高。

4.2 数据文件

VictoriaMetrics每次内存Flush或者后台Merge时生成的数据part，包含metaindex.bin、index.bin、timestamps.bin、values.bin等4个文件。这四个文件的关系如下所示。metaindex.bin文件索引index.bin文件，index.bin文件同时索引timestamps.bin和values.bin文件。



metaindex.bin文件中，包含一系列的metaindex_row， 每个metaindex_row中包含时间线标识TSID、最小时间戳minTimestamp、最大时间戳maxTimestamp、索引块偏移indexBlockOffset、索引块大小indexBlockSize、索引块包含的块头部数blockHeadersCount。

- metaindex_row在文件中的位置按照TSID的大小的字典序排序存储；
- metaindex.bin文件使用ZSTD进行压缩；
- metaindex.bin文件中的内容在part打开时，会全部读出加载至内存中，以加速查询过滤；
- metaindex_row包含时间线标识TSID为其索引的IndexBlock中所有blockHeader中的最小时间标识TSID；
- metaindex_row包含最小时间戳minTimestamp为其索引的IndexBlock中所有blockHeader中的最小时间戳minTimestamp；
- metaindex_row包含最大时间戳maxTimestamp为其索引的IndexBlock中所有blockHeader中的最大时间戳maxTimestamp；
- 查找时根据TSID进行二分检索；

index.bin文件中，包含一系列的indexBlock, 每个indexBlock又包含一系列blockHeader，每个blockHeader的包含时间线标识TSID、最小时间戳minTimestamp、最大时间戳maxTimestamp、第一个指标值firstValue、时间戳数据块偏移timestampsBlockOffset、指标值数据块偏移valuesBlockOffset等内容。

- 每个indexBlock使用ZSTD压缩算法进行压缩；
- 查找时，线性遍历blockHeader查找TSID；

timestamps.bin文件中，包含一系列时间线的时间戳压缩块timestampsData; values.bin文件中，包含的一系列时间线的指标值压缩块valuesData。 其中，timestampsData和values.data会根据时序数据特征进行压缩，整体上的压缩思路是：先做时序压缩，然后在做通用压缩。比如，先做delta-of-delta计算或者异或计算，然后根据情况做zig-zag，最后再根据情况做一次ZSTD压缩，VictoriaMetrics支持的压缩算法或者类型主要有6种，如下图所示，压缩编码源码在lib/encoding/encoding.go文件中。

```
const (
        // MarshalTypeZSTDNearestDelta2 is used for marshaling counter
        // timeseries.
        MarshalTypeZSTDNearestDelta2 = MarshalType(1)

        // MarshalTypeDeltaConst is used for marshaling constantly changed
        // time series with constant delta.
        MarshalTypeDeltaConst = MarshalType(2)

        // MarshalTypeConst is used for marshaling time series containing only
        // a single constant.
        MarshalTypeConst = MarshalType(3)

        // MarshalTypeZSTDNearestDelta is used for marshaling gauge timeseries.
        MarshalTypeZSTDNearestDelta = MarshalType(4)

        // MarshalTypeNearestDelta2 is used instead of MarshalTypeZSTDNearestDelta2
        // if compression doesn't help.
        MarshalTypeNearestDelta2 = MarshalType(5)

        // MarshalTypeNearestDelta is used instead of MarshalTypeZSTDNearestDelta
        // if compression doesn't help.
        MarshalTypeNearestDelta = MarshalType(6)
)
```


VictoriaMetrics文档中提及在生产环境中，每个数据点（8字节时间戳 + 8字节value共计16字节）压缩后小于1个字节，最高可达 0.4字节，如下所示。

VictoriaMetrics

Based on production data from our customers, sample size is 0.4 byte That means one metric with 10 seconds resolution will need 6307200 points * 0.4 bytes/point = 2522880 bytes or 2.4 megabytes. Calculation for number of metrics can be stored in 2 tb disk: 2199023255552 (disk size) / 2522880 (one metric for 2 year) = 871632 metrics So in 2tb we can store 871 632 metrics

5. VictoriaMetrics总结

VictoriaMetrics的整体的存储设计还是不错的，比如数据时间分区、数据压缩率高、丰富的生态协议等。但VictoriaMetrics的标签索引、数据

可靠性、支持的数据类型等方面还存在一些不足。比如，标签索引查询多次IO，可能在时间线数量非常多的场景下，其检索效率会比较低，且没有WAL，在写入时可能会存在数据丢失的风险。目前只支持浮点数类型，不支持布尔、字符串、字节数组等类型。

编辑于 2021-11-13 20:42