Graphite 系列 #3：Whisper Storage Schemas & Aggregations

whisper graphite

yexiaobai 2015年03月03日发布

在前一篇博客中，我们安装了 Carbon 和 Whisper - Graphite 的后端组件。我们这时使用一个 carbon-cache 进程来监听传入的数据点以及使用 Whisper 存储这些数据点。在这篇博客中我将详细描述 Whisper 怎样把这些数据点存储进文件系统以及你怎样控制这些细节。

为什么这很重要？

当你或你的开发同行以及系统管理员开始发布数据点但没有得到期望的结果的时候，可能会有一些困惑：

- 为什么我的数据点获取的是平均值？
- 我已经间歇的发布了数据点，为什么那里没有数据？
- 我已经发布了多天的数据点，为什么只有一天的数据？

Whisper 怎样存储数据？

我们首先需要明白在 Whisper 文件中数据是怎样存储的。当一个 Whisper 文件被创建，它会有一个不会改变的固定大小。在这个文件中可能有多个 "buckets" 对应于不同分别率的数据点，比如：

- Bucket A: 10-second 精度的数据点
- Bucket B: 60-second 精度的数据点
- Bucket C: 10-minute 精度的数据点

每个 bucket 也有一个保留属性指明数据点应该在 bucket 中应该被保留的时间长度。比如：

- Bucket A: 10-second 精度的数据点保留 6 个小时
- Bucket B: 60-second 精度的数据点保留 1 天
- Bucket C: 10-minute 精度的数据点保留 7 天

有了以上两条信息，Whisper 执行一些简单的数学计算来计算出多少数据点会被实际保存在每个 bucket 中：

- Bucket A: 6 hours x 60 mins/hour x 6 data points/min = 2160 points
- Bucket B: 1 day x 24 hours/day x 60 mins/hour x 1 data point/min = 1440 points
- Bucket C: 7 days x 24 hours/day x 6 data points/hour = 1008 points

如果一个 Whisper 文件被使用 storage schema 配置文件创建，它将有 56 KB 大小。如果你通过 whisper-dump.py 脚本运行它，以下信息将会被输出。注意一个存档对应于一个 bucket ，seconds per point 和匹配我们以上估算的数据点属性。

```
Meta data:
  aggregation method: average
  max retention: 604800
  xFilesFactor: 0.5

Archive 0 info:
  offset: 52
  seconds per point: 10
  points: 2160
  retention: 21600
  size: 25920

Archive 1 info:
  offset: 25972
  seconds per point: 60
  points: 1440
  retention: 86400
  size: 17280

Archive 2 info:
  offset: 43252
  seconds per point: 600
  points: 1008
  retention: 604800
  size: 12096
```

Aggregations 怎么样？

当数据被从一个高精度的 bucket 移动到一个低精度的 bucket 时，Aggregations 就会起作用。让我们使用我们前面示例的 Bucket A 和 B：

- Bucket A: 10-second 精度的数据点保存 6 小时（高精度
- Bucket B: 60-second 精度的数据点保存 1 （低精度）

我们或许有一个应用程序每 10 秒发布一次数据点。任何数据点发布与 6 小时以前的都可以在 Bucket A 发现。尽管如此，如果我发起一个 6 小时以前数据点的查询，它们将被发现在 Bucket B。

数据点是怎样移动到 BUCKET B 的？

低精度的值除以高精度的值来决定需要被聚合的数据点的数量：

- 60 seconds (Bucket B) / 10 seconds (Bucket A) = 6 data points to aggregate

注意：Whisper 需要低精度的值整除高精度的值（除法的结果必须整数），否则聚合不会很精确。

为了聚合数据，Whisper 从 Bucket A 读取 6 个 10-second 数据点并对它们执行一个函数来拿出将被存储在 Bucket B 的单个 60-second 数据点。这里有 5 个可选的聚合函数：average, sum, max, min 和 last。聚合函数的选择依赖于你要处理的数据点。95% 精度的值应该可能使用 max 函数聚合。对于计数器，换句话说，sum 函数会更合适。

当聚合数据点的时候，Whisper 也支持 xFilesFactor 概念，它代表了一个 bucket 中 必须包含的被精确聚集的数据点比例。在我们前面的示例中，Whisper 决定需要聚集 6 个 10-second 数据点。它可能仅仅 4 个数据点有数据，其他两个是 null - 由于网络问题，应用重启等等。

如果我们的 Whisper 文件 xFilesFactor 值是 0.5，这意味着仅仅只有至少 50% 数据点被呈现的时候，它才将聚集数据点。如果超过 50% 数据点是 null， Whisper 将创建一个 null 聚合。在我们的示例中，我们 6 个数据点有 4 个 - 66%。聚合函数将在非空数据点生效来创建聚合值。

你可以给 xFilesFactor 设置 0 -1 之间的任何值。0 表明了即使只有一个数据点非空，聚合就应该被完成。 1 表明了只有所有的数据非空，集合才能完成。

在前一篇博客中，我们拷贝了 `/opt/graphite/conf` 目录下的所有示例配置文件。这些配置文件控制了 Whisper 文件是怎样被创建的：

- /opt/graphite/conf/storage-schemas.conf
- /opt/graphite/conf/storage-aggregation.conf

默认的 STORAGE SCHEMAS

storage-schemas 配置文件是由多个条目包含匹配指标名称和保留时间定义的模式组成的。默认有两个条目：carbon 和 其他的。

carbon 条目匹配以 "carbon" 字符开始的指标名称。Carbon 进程每 60s 发射它们自己内部的指标 - 默认情况下，但是可以被改变。比如，一个 carbon-cache 进程将发射一个它每分钟创建的指标文件数量的指标。保留时间定义指明了数据点每 60s 报告一次，保存 90天。

```
[carbon]
pattern = ^carbon\.
retentions = 60:90d
```

其他的条目度量任何其他的使用星号指定的与 carbon 无关的指标。保留时间定义指明了数据点每 60s 报告一次，将被保存 1 天。

```
[default_1min_for_1day]
pattern = .*
retentions = 60s:1d
```

默认的 STORAGE AGGREGATION

storage-aggregation 配置文件由多个条目包含组成：

- 匹配指标名称的模式
- 一个 xFilesFactor 值
- 一个聚合函数

默认情况下，有 4 个条目：

- 以 .min 结束的指标
    - 使用 min 聚合函数
    - 至少 10% 的数据点应该出现在聚合中
- 以 .max 结束的指标
    - 使用 max 聚合函数
    - 至少 10% 数据点应该出现在聚合中
- 以 .count 结束的指标
    - 使用 sum 聚合函数
    - 至少一个一个数据点聚合
- 任何其他指标
    - 使用 average 聚合函数
    - 至少 50% 数据点出现在聚合中

```
[min]
pattern = \.min$
xFilesFactor = 0.1
aggregationMethod = min

[max]
pattern = \.max$
xFilesFactor = 0.1
aggregationMethod = max

[sum]
pattern = \.count$
xFilesFactor = 0
aggregationMethod = sum

[default_average]
pattern = .*
xFilesFactor = 0.5
aggregationMethod = average
```

默认的 storage schemas 和 storage aggregations 用于测试的时候工作的非常好，但是对于生产环境的指标，你或许需要改变配置文件。

修改 STORAGE SCHEMAS

首先，我将修改 carbon 条目。我想保留 Carbon 每 60s 和 180 天（6个月）的指标报告。180 天后，我想把指标撤收到 10分钟精度并保持 180天。

```
[carbon]
pattern = ^carbon\.
retentions = 1min:180d,10min:180d
```

我生产环境和准生产环境被运行在 Dropwizard 应用程序的 Coda Hale metrics library 每 10秒发布一次。我想保存 10s 的数据 3 天。3 天后，数据应该被聚合到 1-minute 数据并且保存 180天。最后，六个月后，数据应该被聚合到 10-minute 数据并保存 180 天。

> 注意：如果我的 metrics library 发布数据在不同的频率，我的保存定义将需要改变来匹配它。

```
[production_staging]
pattern = ^(PRODUCTION|STAGING).*
retentions = 10s:3d,1min:180d,10min:180d
```

不是 carbon 的指标，生产环境，或者是准生产环境指标可能仅仅测试指标，我将仅仅保存 1天。

```
[default_1min_for_1day]
pattern = .*
retentions = 60s:1d
```

修改 STORAGE AGGREGATION

我将要保持默认的 storage aggregation 条目，但会增加一些以 ratio，m1_rate 和 p95结束的指标。

> 注意：你需要在默认的条目之前增加任何新的条目

```
[ratio]
pattern = \.ratio$
xFilesFactor = 0.1
aggregationMethod = average

[m1_rate]
pattern = \.m1_rate$
xFilesFactor = 0.1
aggregationMethod = sum

[p95]
pattern = \.p95$
xFilesFactor = 0.1
aggregationMethod = max
```

恭喜你，这时候，你应该配置了你的 Graphite 后端来匹配你应用程序发布数据点的频率，并且最后明白了数据点是怎样存储在文件系统的。在下一篇博客中，我将使用使用 graphite-webapp 可视化数据。

Graphite 系列 ：

1. Provision Hardware
2. Carbon & Whisper
3. Whisper Storage Schemas & Aggregations
4. Graphite Webapp
5. Stress Testing Carbon Caches
6. Carbon Aggregators
