参考：<http://blog.csdn.net/bluishglc/article/details/31052749>

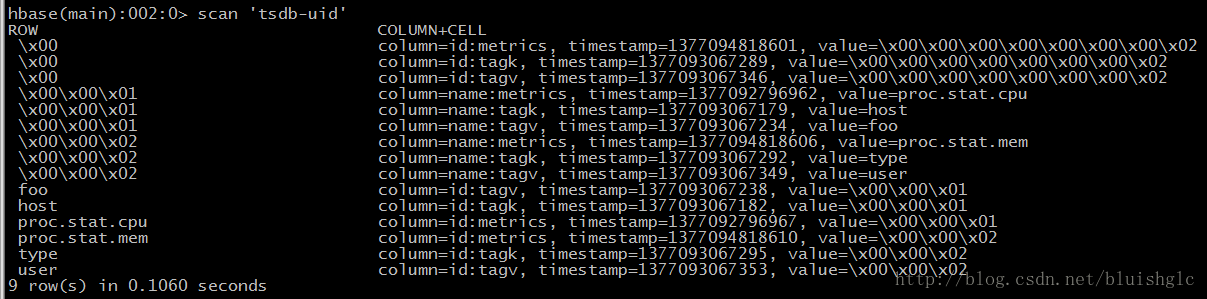
OpenTSDB是基于HBase存储时间序列数据的一个开源数据库，确切地说，它只是一个HBase的应用而已，其对于时间序列数据的处理可以供其他系统参考和借鉴。本文会针对它在数据库的设计方面展开一些探索和讨论。本文原文链接：<http://blog.csdn.net/bluishglc/article/details/31052749>，转载请注明出处！

本文基于的是OpenTSDB最早的一个稳定版本1.0.0进行讲解的，下载部署完成之后，我们首先需要了解的是它的数据库Schema， 它主要有两个表：tsdb-uid和tsdb. 前者描述指标（metrics）相关的元数据，后者存储时间序列数据。首先我们来了解一下“指标”（metrics）的概念，简单讲一个指标就是一个需要收集的数据项，但是只有指标是不能全面地描述出一条数据产生的相关背景信息的，比如：如果我们要统计cpu的使用率，我们可以建立一下名为**proc.stat.cpu**的metrics,如果我们从不同的机器和用户下收集了大量的cpu信息，如果没有对一条信息进行一定地标识，我们是无法区分出哪些数据来自哪台机器的哪个用户，所以我们还需要建立一些“标签”（Tag）来标识一条数据。严格地说，指标和标签之间并没有必然的从属关系，就像两个不同的指标的数据可能都有指示其来自哪台主机的host标签一样，但是有一点是确定的，即：**对于一条数据来说，应该至少含有一个指标和一个标签**，这样的数据才是有意义的，因此，在OpenTSDB的表设计上，就把“指标”（metrics）和“标签”（Tag）统一放在了tsdb-uid表中存储，格式为：**RowKey(自增ID，3字节数组）：name:metrics,name:tagk,name:tagv，**同时对它们之间的**反向关联关系**也作了展开存储。

实际上我们看以看到，对于数据来说，指标到数据是一对多的父子关系，标签对数据也是一对多的父子关系，OpenTSDB在**这里的设计是非常具有典型性的，实际上这也是HBase表设计上的一种常见的“Pattern”**：**把表间关联关系展开，以JOIN的结果为RowKey存储数据！包括正向关联和反向关联两类数据！（请仔细参考图1理解）**

下面让我们插入2个metrics：**proc.stat.cpu**和**proc.stat.mem，**以及一条记录: **proc.stat.cpu 1297574486 54.2 host=foo type=user**来观察一下数据表结构**：**

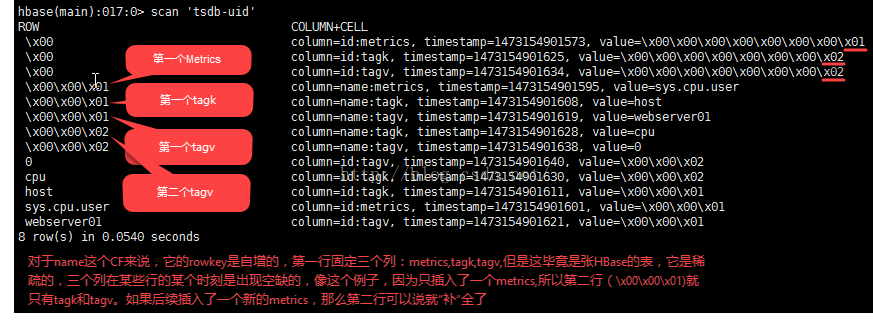
首先是**tsdb-uid**表：



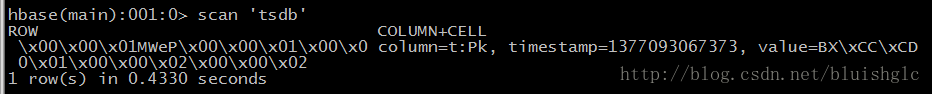
图一

从表中的记录可知:

1. 第一条记录：rowkey为\x00，含3个字段：metrics,tagk,tagv， 其值分别是已经添加的所有指标、标签名和标签值的数量。这一条数据是系统生成和维护的。这里有两个metrics:cpu和mem,两个key:host和type,两个value:foo和user,所以 rowkey为\x00的三个数据的value都是2

2. 在OpenTSDB中，每一个metric、tagk或者tagv在创建的时候被分配一个唯一标识叫做UID，他们组合在一起可以创建一个序列的UID或者TSUID。在OpenTSDB的存储中，对于每一个metric、tagk或者tagv都存在从0开始的计数器，每来一个新的metric、tagk或者tagv，对应的计数器就会加1。当data point写到TSD时，UID是自动分配的。你也可以手动分配UID，前提是auto metric被设置为true。  
  
关于UID，我们再看一张图解：  
  


然后我们看**tsdb**表：



图二

我们看重点看一下纪录表的rowkey:

指标UID(指标+标签的某个组合）+ 数据生成时间（取整点时间）+标签1-Key的UID+标签1-Vlaue的UID+...+标签N-Key的UID+标签N-Vlaue的UID

让我们以图纪录为例，重点看一下时间的处理：

1297574486 = 2011-02-13 13:21:26

MWeP = 01001101 01010111 01100101 01010000 = 1297573200 = 2011-02-13 13:00:00 (截取整点小时位)

PK = 01010000 01101011 = 1286 (从整点小时到记录时间的秒偏差,1286秒正是21分钟26秒)

1297573200+1286=1297574486

**PK，即小时内秒数被当作了Column**

**一些设计技巧：**

**1. 针对Hot Spot的应对策略**

OpenTSDB处理的是典型的时间序列化数据，必然面临“热点”问题，关于它对热点问题的处理，HBase的官方文档 <http://hbase.apache.org/book/rowkey.design.html> 中专门提到过：

 However, the difference is that the timestamp is not in the *lead* position of the key, and the design assumption is that there are dozens or hundreds (or more) of different metric types. Thus, even with a continual stream of input data with a mix of metric types, the Puts are distributed across various points of regions in the table.

一般来说，如果使用时间做rowkey,那么前面就必须加“哈希”字段（也就是salted处理)。但是OpenTSDB并没有特别的哈希字段，它的处理比较聪明：首先,时间字段不会放在rowkey的开始位置，其次，rowkey开始位置挑选了自身的一个理想的业务字段“metrics"来替代了“哈希”字段。

从OpenTSDB的处理上我们可以总结出一点：在处理时间序列数据时，如果系统中存在“理想的”“天然的”起哈希作用的字段应该优先考虑其作为rowkey的起始组成部分，后接时间字段，但如果找不到这样的字段再设置人工的哈希字段

**2. rowkey的设计思想**

一.为了能够检索特定的metrics,tag name,tag name的data point, 将 metrics,tag name,tag name编入rowkey是显然的事情,但是直接使用它们来组成rowkey有两个明显的问题:

1. 会占用大量的存储空间(因为这些值会大量重复地出现在很多的rowkey中)

2. 由于每一个metrics,tag key,tag value的长度都是不固定的,这不利于通过字节偏移量来直接定位它们.(否则需要使用特定的分隔符,而且为了避免输入信息中可能存在特定的分隔符导致解析出错,还要对所有输入信息的分割符进行转义处理)

围绕一个性能指标,会有多种附加"属性"(或者说"标签")对其进行说明与描述, 那么对指标的查询也自然是以这些标签或标签值展开的,因此一条指标记录的rowkey必然要包含这些标签和标签值.但是由于标签和标签值是不定长的,这为rowkey的设计带来麻烦,所以需要为这些标签和标签值分配一个定长的ID,在rowkey中使用它们的ID来指代它们,这样rowkey就可以规范化,方便从rowkey中直接通过偏移截取需要的"部分".

二.Tall-Narrow和Wide-Flat两种表设计风格相结合