**Hbase与opentsdb读写测试分析报告**

**1 测试环境**

**1.1硬件环境**

|  |  |  |  |
| --- | --- | --- | --- |
| 主机名 | IP | 内存 | 硬盘 |
| bigdata-1 | 172.17.81.238 | 32G(8G DDR3\*4) | 1T |
| bigdata-2 | 172.17.81.235 | 8G(2G DDR3\*2) | 500G+3T |
| bigdata-3 | 172.17.81.240 | 4G(2G DDR2\*2) | 500G |
| bigdata-4 | 172.17.81.237 | 4G(2G DDR2\*2) | 500G |
| bigdata-5 | 172.17.81.236 | 4G(2G DDR2\*2) | 500G |
| bigdata-6 | 172.17.81.239 | 16G(8G DDR3\*2) | 500G+3T |

表1 服务器硬件环境

**1.2软件环境**

**Hadoop 2.7.3**

**Hbase 1.2.4**

**Java 1.8.0\_111**

**Opentsdb 2.3.0**

**Spark 2.0.0**

**2 opentsdb读写分析**

我们在opentsdb下设计了两种存储方式（见表2），一种是以“台站\_测点\_测项”为metric（后称metric\_1），以“采样率”为tag进行存储。另一种是以“测项”为标签metric（后称metric\_2），以“台站”、“测点”、“采样率”为tag进行存储。下面我们对于两种不同的存储方案进行测试，根据其读写的用时，分析不同存储方式的优劣。

|  |  |  |  |
| --- | --- | --- | --- |
| Row Key | Column Family:t | | |
| 0 | ... | 3600 |
| metric|timestamp|tagk:tagv | value | value | value |
| 台站\_测点\_测项|时间戳|'采样率':采样率 | value | value | value |
| 测项|时间戳|'台站':台站 '测点':测点 '采样率':采样率 | value | value | value |

表2 两种数据存储方式

**2.1 opentsdb写入数据分析**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 台站 | 测项 | 测点 | 数据个数 | 数据大小 | metric | 用时(ms) |
| 11004 | 4253 | 5 | 5052 | 73.1 KB (74,859 字节) | 台站\_测点\_测项 | 1500 |
| 11004 | 4253 | 5 | 5052 | 73.1 KB (74,859 字节) | 测项 | 1364 |
| 11009 | 4214 | 1 | 122704 | 2.04 MB (2,141,535 字节) | 台站\_测点\_测项 | 27416 |
| 11009 | 4214 | 1 | 122704 | 2.04 MB (2,141,535 字节) | 测项 | 27656 |
| 11004 | 4112 | 4 | 8327520 | 169 MB (177,270,136 字节) | 台站\_测点\_测项 | 1821486 |
| 11004 | 4112 | 4 | 8327520 | 169 MB (177,270,136 字节) | 测项 | 2470766 |

表3 opentsdb写入数据用时(ms)

由上表可以看出，当数据量较小时，两种存储方式的写入时间相差不大（见图1），而当数据量逐渐增加到百万个时，metric\_1的存储方法明显要比metric\_2的存储方法写入时间要少的多（见图2）。

图1 小数据写入时间对比

图2 百万级数据写入时间对比

**2.2 opentsdb 读取数据分析**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 台站 | 测项 | 测点 | 数据个数 | 数据大小 | metric | 用时(ms) |
| 11004 | 4253 | 5 | 5052 | 73.1 KB (74,859 字节) | 台站\_测点\_测项 | 151 |
| 11004 | 4253 | 5 | 5052 | 73.1 KB (74,859 字节) | 测项 | 190 |
| 11009 | 4214 | 1 | 122704 | 2.04 MB (2,141,535 字节) | 台站\_测点\_测项 | 625 |
| 11009 | 4214 | 1 | 122704 | 2.04 MB (2,141,535 字节) | 测项 | 1945 |
| 11004 | 4112 | 4 | 8327520 | 169 MB (177,270,136 字节) | 台站\_测点\_测项 | 11942 |
| 11004 | 4112 | 4 | 8327520 | 169 MB (177,270,136 字节) | 测项 | 21103 |

表4 opentsdb读取数据用时(ms)

根据表4，可以看出随着数据量的增加，metric\_1的读取时间要小于metric\_2的读取时间（见图3），尤其当数据量达到百万级时，metric\_1的读取时间只有metric\_2读取时间的56%（见图4）。

图3 小数据读取时间对比

图4 八百万级数据读取时间对比

**2.3 opentsdb读写测试结果分析**

根据对比结果显示，metric\_1和metric\_2的写入效率在小数据方面没有太大差异，但是当数据量达到百万级时，metric\_2的存储时间更长。在读取数据方面，metric\_1无论在小数据还是百万级数据方面，都要比metric\_2读取速度更快。由此可得，无论是写入数据还是读取数据，metric\_1的性能较metric\_2来说都较为优秀。所以我们决定选择metric\_1作为我们opentsdb的数据存储方式。

在同样的数据量的情况下，metric\_1的读写效率都要比metric\_2要优秀一些，对这种情况我们通过查询tsdb表和uid表进行推测分析：

首先，我们查询数据的要求是精确查找，即知道一组数据的所有属性，从而得到某一段时间某台站某一测项的某一测点的一组数据。我们对于不同台站，不同测项或者不同测点的数据查询的速度并没有要求。

第二，对于读取数据两种存储方式速度差异较大的原因，我们认为是由于记录表里row key的设计引起的。纪录表的row key存储方式是“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”。可以看出row key把数据生成时间放在了tag之前，数据聚合过程中就会把时间属性的聚合优先级放在tag之前，对于metric\_1来说，唯一标识某一组数据的要素都放在了metric之中（台站\_测点\_测项），所以当我们查询某一组数据时，metric\_1的记录表里该组数据都聚合在一块区域，查找速度快。而对于metric\_2来说，只将测项作为metric，把台站和测点都放在了tag之中。导致记录表内会优先将同一测项相同时间的不同台站或测点的数据聚合在一起，当我们查找时，由于我们想要的数据是分散的，所以需要花费更多的时间才能找到我们需要的数据。

第三，对于写入数据两种存储方式速度存在差异的原因，我们认为是由于opentsdb数据是独立写入的，而每行会存多条数据，当前数据写入时如果已经存在row，就需要将其加入之前的row。而这种情况就会在加入的时候对原表进行查询操作，由第二点分析可以看出，查询在不同的metric设计下效率会存在差异，因此会造成写入数据时，两种存储方式速度不同。

1. **hbase 读写分析**

我们在hbase中也设计了两种存储方式，一种是将“台站测点测项采样率时间”为row key（后称hbase\_1），将value作为列，每组数据存多行，只有一列（见表5）。另一种是将“台站测点测项采样率”作为row key（后称hbase\_2），将时间作为列，没组数据存一行多列（见表6）。下面我们对于两种不同的存储方案进行测试，分析不同存储方式的优劣。

|  |  |
| --- | --- |
| Hbase | 列族：列 |
|
| Row key | data:data |
| 台站测点测项采样率时间 | value |
| 台站测点测项采样率时间 | value |
| 台站测点测项采样率时间 | value |
| 台站测点测项采样率时间 | value |
| 台站测点测项采样率时间 | value |
| 台站测点测项采样率时间 | value |
| 台站测点测项采样率时间 | value |
| 台站测点测项采样率时间 | value |
| 台站测点测项采样率时间 | value |
| 台站测点测项采样率时间 | value |
| 台站测点测项采样率时间 | value |

表5 hbase\_1 存储方式

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hbase | 列族：列 | 列族：列 | 列族：列 | 列族：列 |
|
| rowkey | date:时间 | date:时间 | date:时间 | date:时间 |
| 台站测点测项采样率 | value | value | value | value |
| 台站测点测项采样率 | value | value | value | value |

表6 hbase\_2 存储方式

**3.1 hbase写入数据分析**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 台站\_测点\_测项 | 数据个数 | 数据大小 | 存储方式 | 用时(ms) |
| 11004\_4\_4112+11009\_1\_4214+11004\_5\_4253 | 8455276 | 179486530 | hbase\_1 | 403272 |
| 11004\_4\_4112+11009\_1\_4214+11004\_5\_4253 | 8455276 | 179486530 | hbase\_2 | 113355 |

表7 hbase 写入数据用时

由表7可得，hbase\_1的用时比hbase\_2的用时要长很多（如图5）。

图5 hbase 写入时间对比

**3.2 hbase读取数据分析**

|  |  |  |
| --- | --- | --- |
| 存储方式 | 读取数据量级 | 用时(ms) |
| hbase\_1 | 万级 | 142.044 |
| hbase\_2 | 万级 | 89.922 |
| hbase\_1 | 百万级 | 13319.046 |
| hbase\_2 | 百万级 | 7732.01 |

表8 hbase读取数据用时

由表8可得，无论对于万级数据还是百万级数据，hbase\_2读取数据的用时都要比hbase\_1要少（见图6，图7）。

图7 hbase万级数据读取时间对比

图8 hbase百万级数据读取时间对比

**3.3 hbase读写测试结果分析**

根据结果显示，我们可以看出hbase\_2的效率要完全碾压hbase\_1的效率。

写入方面，hbase\_2的速度是hbase\_1的速度的近3倍。读取方面，hbase\_2的速度大约比hbase\_1要快50%。

但是，hbase\_2存在设计缺陷，假如某一组数据量超过3000万，hbase会出现OOM（内存不足）的bug。所以就现实情况而言，我们采取综合两种存储方式分别对不同采样率进行优化，得到第三种存储方式（见表9）。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hbase | 列族：列 | 列族：列 | 列族：列 | 列族：列 |
|
| Row key | date:时间 | date:时间 | date:时间 | date:时间 |
| 台站测点测项采样率时间索引 | value | value | value | value |
| 台站测点测项采样率时间索引 | value | value | value | value |

表9 新存储方式

新的存储方式是将“台站测点测项采样率时间索引”作为row key，其中时间索引是取时间精确到天的值，列存数据的时间，在行和列两个方向共同存储一组数据。这种存储方法的好处是充分利用hbase\_2的时间优越性，又增加了限制，不至于引发OOM的bug（这种存储方式的最大列数是24\*60\*60=86400）。

1. **hbase与opentsdb间的比较**

**4.1 hbase与opentsdb写入数据比较**

hbase写入数据用时113355ms（见图5），opentsdb写入完全相同的数据用时1850402ms（见表3，metric\_1三组数据相加）。可以看出hbase写入数据的速度比opentsdb要快十倍还多，hbase在写入数据方面有压倒性的优势。

**4.2 hbase与opentsdb读取数据比较**

由于hbase并没有与opentsdb读取相同的数据，就以数据量来进行计算。

Hbase读取百万级数据用时最短7732.01ms（见图8）。

Opentsdb读取8百万数据用时最短11942ms（见图4）。

可以看出opentsdb的读取要比hbase的读取速度快很多。

**4.3 结果分析**

由上面的数据我们可以得出，hbase的写入数据要比opentsdb强很多，但opentsdb的读取数据的速度也要比hbase强很多。我们系统的主要要求就是对读取数据的要求，由此看来应该选择opentsdb作为我们数据的存储结构。但是，opentsdb的写入数据的时间太慢，我们所有数据全部写入数据库可能需要近1个月的时间（即使多线程优化以后也需要半个月），且opentsdb没有针对我们后续的数据处理的API。 由于opentsdb其实就是基于hbase的一种存储方式，我们决定利用hbase模拟opentsdb的存储方式进行存储，然后用spark对数据进行分析。