IoTDB C#客户端性能分析报告

问题提出

在C#客户端中对IoTDB服务器端执行插入操作时，时延大致由三部分组成，它们分别为：

- 对插入的数据进行序列化的时延
- 服务器端处理插入操作的时延
- 网络传输时延

在实际情景中，对不同规模和组织方式的数据，如何选择接口是一个值得考量的问题，结果相同的情况下，选择不同的接口有时会带来极大的性能差异。我们接下来会从IoTDB C#客户端微信群中客户提出的一个问题入手，来探讨在某种特定情景下的接口选择问题，为后续用户在插入数据的场景下的选择提供参考

问题描述

该用户的需求在于，需要针对某个device建立500个timeseries，每个timeseries以秒为单位写入数据，一天的原始数据转化为3600 * 24行的TS数据，总体写入1-2年的数据。最终形成的表结构大致如下所示

timestamp	ts1	ts2	...	ts500
1	*	*	...	*
2	*	*	...	*
3	*	*	...	*
4	*	*	...	*
...	...	...	...	...

在写入的过程中用户发现使用insert_tablet接口插入这些数据的速度比较反常，速度较慢，同时速度在下降，于是更改了对数据的组织形式，每个设备一个timeseries，采用并行的策略来插入，最终形成的结构大致如下

timestamp	ts1	timestamp	ts2	...	timestamp	ts500

1	*	1	*	...	1	*
2	*	2	*	...	2	*
3	*	3	*	...	3	*
4	*	4	*	...	4	*
...	...	...	...	...	...	...

这种结构带来了并行编程的困难，同时由于目前我们的客户端尚未支持并行，这种方法的速度会慢于第一种组织结构。我们集中讨论第一种组织形式，在用户的这种需求下，我们需要探讨使用哪种接口会适合于这种组织形式的数据的插入及接口的某些设计是否有待改进

实验与分析

在对问题进行解读后，我们从插入操作的三种时延入手，来确定延迟大的原因，同时使用了不同接口来进行测试，以期找到这种情况下延迟最小的方案

对于批处理操作，通过test_insert * 接口，能测量网络延时和序列化的总时间，在接口内部测试出序列化时间。能够测量出三个部分的具体时延。实验结果如下所示，由于是在PC上测试，复现结果可能因为网络波动和硬件差异与以下结果有不同

insert_record接口

总共插入20000行数据，每行数据有两个timeseries，类型分别为boolean和32位的int，用时如下表所示（调用20000次insert_record接口）：

	本地用时(ms)	远端用时(ms)
insert_record	2200ms	1255000
test_insert_record	1600ms	1208000

- 本地吞吐量：9100 p/s
- 远端吞吐量：16 p/s

insert_records接口

总共插入20000行数据，每行数据有两个timeseries，类型分别为boolean和32位的int，用时如下表所示：

	本地用时(ms)	远端用时(ms)
insert_records	35（序列化）+ 448（task等待）	44（序列化）+ 20000（task等待）
test_insert_records	43（序列化）+ 128（task等待）	41（序列化） + 23000（task等待）

- 本地吞吐量：41000 p/s
- 远端吞吐量：1000 p/s

insert_tablet接口

总共插入20000行数据，每行数据有两个timeseries，类型分别为boolean和32位的int，形成大小为20000行的tablet，用时如下表所示：

	本地用时(ms)	远端用时(ms)
insert_tablet	6800（序列化）+ 120（task等待）	6600（序列化）+ 250（task等待）
test_insert_tablet	6500（序列化）+ 1（task等待）	6800（序列化）+ 130（task等待）

- 本地吞吐量：2900 p/s
- 远端吞吐量：2850 p/s
- 可以看到，序列化时间过长，insert_tablet接口在本地服务器中插入20000条数据时，序列化时间占比超过98%，绝大部分时间花费在数据的序列化上。经过检查我们发现在tablet序列化的过程中有频繁的内存申请和释放操作，致使整体序列化耗时大大增加，详见ByteBuffer介绍，由此我们对序列化的函数进行一定程度的优化，优化后得到的测试结果如下所示：

	本地用时(ms)	远端用时(ms)
insert_tablet	23（序列化）+ 210（task等待）	26（序列化）+ 250（task等待）
test_insert_tablet	36（序列化）+ 2（task等待）	33（序列化）+ 220（task等待）

- 本地吞吐量：86000 p/s
- 远端吞吐量：72000 p/s
- 优化后，序列化时间占据整体时间的7%~10%，延时减少，吞吐量大幅度提升

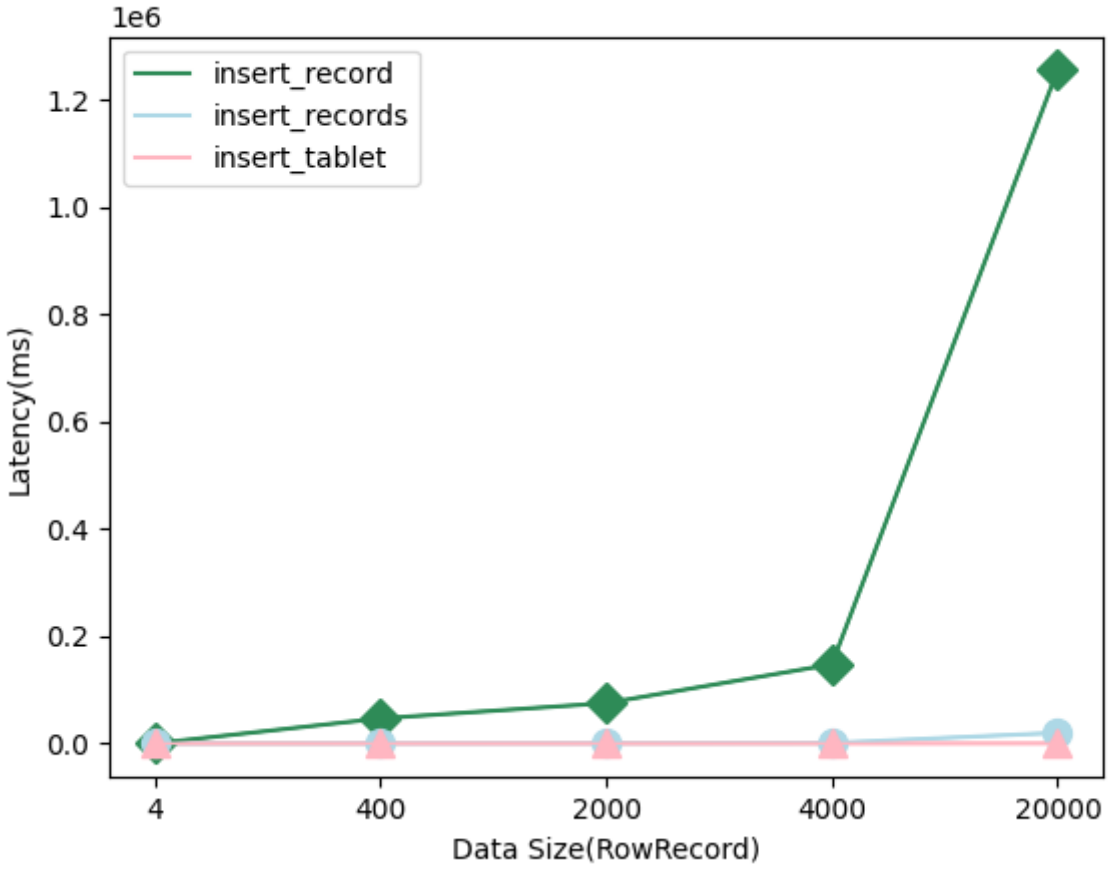
分析

在实验过程中，每次结果有较大的偏差，原因是网络不稳定，网络延时的波动导致整体的耗时波动。对于固定的批处理操作，序列化时间和服务器端处理请求（进行插入操作）的整体是稳定的。

在三种接口的耗时在改进前均不理想，对于各个接口耗时长的原因对比分析如下：

- 对于insert_record接口和insert_records，前者一共调用了20000次，每次都需要等待网络延迟的时间，频繁的请求占据了绝大部分时间，而后者是一次性的发送数据和请求，大大减少了网络延时的占比，因此用时也大幅减少，远端吞吐量提升约60倍
- 对比insert_records和优化后的insert_tablet接口，序列化的时间差距不大。服务器端在处理records时需要按行处理，而对于结构化的tablet处理速度更快，因而insert_tablet的用时相对于insert_records大幅减少，远端吞吐量提升70~80倍
- 对比优化前和优化后的insert_tablet接口，后者的序列化时间相对前者大幅减少，原因是在序列化时内存的申请释放操作减少，速度增加，因而优化后用时大幅减少，远端吞吐量提升25~30倍

远端服务器不同数据规模下各接口耗时如下图所示：



可以看到，随着数据规模的增加，insert_record的时间消耗大幅度增长，远高于insert_records和insert_tablet

结论

经过上述实验，我们得出如下结论：

- 在插入与该用户类似的结构化较强、没有空值、规整、每行的column固定的的数据时，建议使用insert_tablet接口，经过改善后的insert_tablet接口具备较好的性能，能满足该用户的需求
- 数据量较大，但数据整体不规整或者有空值，每行数据的column数不定时建议使用insert_records接口，该接口对record数据的插入速度较为可观
- 数据量小，需要对原有数据做出一定的修正时，使用insert_record接口