<http://www.infoq.com/cn/articles/storage-in-sequential-databases?utm_source=articles_about_imdbs&utm_medium=link&utm_campaign=imdbs>

2017 年时序数据库忽然火了起来。开年 2 月 Facebook 开源了 beringei 时序数据库；到了 4 月基于 PostgreSQL 打造的时序数据库 TimeScaleDB 也开源了，而早在 2016 年 7 月，百度云在其天工物联网平台上发布了国内首个多租户的分布式时序数据库产品 TSDB，成为支持其发展制造，交通，能源，智慧城市等产业领域的核心产品，同时也成为百度战略发展产业物联网的标志性事件。**时序数据库作为物联网方向一个非常重要的服务，业界的频频发声，正说明各家企业已经迫不及待的拥抱物联网时代的到来。**

本文会从时序数据库的基本概念、使用场景、解决的问题一一展开，最后会从如何解决时序数据存储这一技术问题入手进行深入分析。

## 1. 背景

百度无人车在运行时需要监控各种状态，包括坐标，速度，方向，温度，湿度等等，并且需要把每时每刻监控的数据记录下来，用来做大数据分析。每辆车每天就会采集将近 8T 的数据。如果只是存储下来不查询也还好（虽然已经是不小的成本），但如果需要快速查询“今天下午两点在后厂村路，速度超过 60km/h 的无人车有哪些”这样的多纬度分组聚合查询，那么时序数据库会是一个很好的选择。

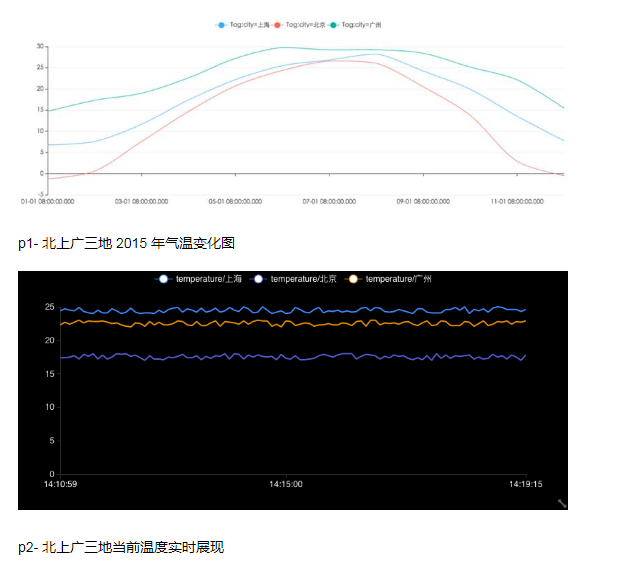
## 2. 什么是时序数据库

先来介绍什么是时序数据。时序数据是基于时间的一系列的数据。在有时间的坐标中将这些数据点连成线，往过去看可以做成多纬度报表，揭示其趋势性、规律性、异常性；往未来看可以做大数据分析，机器学习，实现预测和预警。

时序数据库就是存放时序数据的数据库，并且需要支持时序数据的快速写入、持久化、多纬度的聚合查询等基本功能。

对比传统数据库仅仅记录了数据的当前值，时序数据库则记录了所有的历史数据。同时时序数据的查询也总是会带上时间作为过滤条件。

**时序数据示例**



下面介绍下时序数据库的一些基本概念（不同的时序数据库称呼略有不同）。

**metric**: 度量，相当于关系型数据库中的 table。

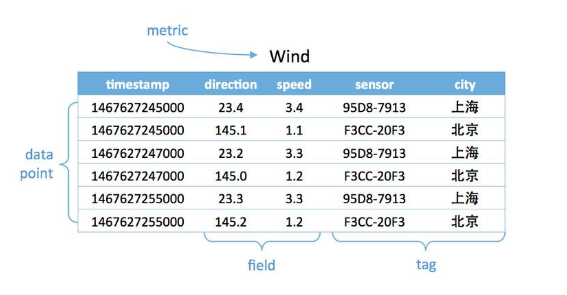
**data point**: 数据点，相当于关系型数据库中的 row。

**timestamp**：时间戳，代表数据点产生的时间。

**field**: 度量下的不同字段。比如位置这个度量具有经度和纬度两个 field。一般情况下存放的是会随着时间戳的变化而变化的数据。

**tag**: 标签，或者附加信息。一般存放的是并不随着时间戳变化的属性信息。timestamp 加上所有的 tags 可以认为是 table 的 primary key。

如下图，度量为 Wind，每一个数据点都具有一个 timestamp，两个 field：direction 和 speed，两个 tag：sensor、city。它的第一行和第三行，存放的都是 sensor 号码为 95D8-7913 的设备，属性城市是上海。随着时间的变化，风向和风速都发生了改变，风向从 23.4 变成 23.2；而风速从 3.4 变成了 3.3。



p3- 时序数据库基本概念图

## 3. 时序数据库的场景

所有有时序数据产生，并且需要展现其历史趋势、周期规律、异常性的，进一步对未来做出预测分析的，都是时序数据库适合的场景。

在工业物联网环境监控方向，百度天工的客户就遇到了这么一个难题，由于工业上面的要求，需要将工况数据存储起来。客户每个厂区具有 20000 个监测点，500 毫秒一个采集周期，一共 20 个厂区。这样算起来一年将产生惊人的 26 万亿个数据点。假设每个点 50Byte，数据总量将达 1P（如果每台服务器 10T 的硬盘，那么总共需要 100 多台服务器）。这些数据不只是要实时生成，写入存储；还要支持快速查询，做可视化的展示，帮助管理者分析决策；并且也能够用来做大数据分析，发现深层次的问题，帮助企业节能减排，增加效益。最终客户采用了百度天工的时序数据库方案，帮助他解决了难题。

在互联网场景中，也有大量的时序数据产生。百度内部有大量服务使用天工物联网平台的时序数据库。举个例子，百度内部服务为了保障用户的使用体验，将用户的每次网络卡顿、网络延迟都会记录到百度天工的时序数据库。由时序数据库直接生成报表以供技术产品做分析，尽早的发现、解决问题，保证用户的使用体验。

## 4. 时序数据库遇到的挑战

很多人可能认为在传统关系型数据库上加上时间戳一列就能作为时序数据库。数据量少的时候确实也没问题，但少量数据是展现的纬度有限，细节少，可置信低，更加不能用来做大数据分析。很明显时序数据库是为了解决海量数据场景而设计的。

可以看到时序数据库需要解决以下几个问题

* **时序数据的写入**：如何支持每秒钟上千万上亿数据点的写入。
* **时序数据的读取**：又如何支持在秒级对上亿数据的分组聚合运算。
* **成本敏感**：由海量数据存储带来的是成本问题。如何更低成本的存储这些数据，将成为时序数据库需要解决的重中之重。

这些问题不是用一篇文章就能涵盖的，同时每个问题都可以从多个角度去优化解决。在这里只从数据存储这个角度来尝试回答如何解决大数据量的写入和读取。

## 5. 数据的存储

数据的存储可以分为两个问题，单机上存储和分布式存储。

### 单机存储

如果只是存储起来，直接写成日志就行。但因为后续还要快速的查询，所以需要考虑存储的结构。

传统数据库存储采用的都是 B tree，这是由于其在查询和顺序插入时有利于减少寻道次数的组织形式。我们知道磁盘寻道时间是非常慢的，一般在 10ms 左右。磁盘的随机读写慢就慢在寻道上面。对于随机写入 B tree 会消耗大量的时间在磁盘寻道上，导致速度很慢。我们知道 SSD 具有更快的寻道时间，但并没有从根本上解决这个问题。

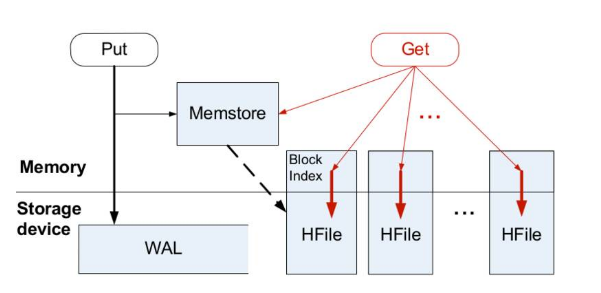
对于 90% 以上场景都是写入的时序数据库，B tree 很明显是不合适的。

业界主流都是采用 LSM tree 替换 B tree，比如 Hbase, Cassandra 等 nosql 中。这里我们详细介绍一下。

LSM tree 包括内存里的数据结构和磁盘上的文件两部分。分别对应 Hbase 里的 MemStore 和 HLog；对应 Cassandra 里的 MemTable 和 sstable。

LSM tree 操作流程如下：

1. 数据写入和更新时首先写入位于内存里的数据结构。为了避免数据丢失也会先写到 WAL 文件中。
2. 内存里的数据结构会定时或者达到固定大小会刷到磁盘。这些磁盘上的文件不会被修改。
3. 随着磁盘上积累的文件越来越多，会定时的进行合并操作，消除冗余数据，减少文件数量。



p4-Hbase LSM tree 结构介绍（注 1）

可以看到 LSM tree 核心思想就是通过内存写和后续磁盘的顺序写入获得更高的写入性能，避免了随机写入。但同时也牺牲了读取性能，因为同一个 key 的值可能存在于多个 HFile 中。为了获取更好的读取性能，可以通过 bloom filter 和 compaction 得到，这里限于篇幅就不详细展开。

### 分布式存储

时序数据库面向的是海量数据的写入存储读取，单机是无法解决问题的。所以需要采用多机存储，也就是分布式存储。

分布式存储首先要考虑的是如何将数据分布到多台机器上面，也就是 分片（sharding）问题。下面我们就时序数据库分片问题展开介绍。分片问题由分片方法的选择和分片的设计组成。

### 分片方法

时序数据库的分片方法和其他分布式系统是相通的。

**哈希分片**：这种方法实现简单，均衡性较好，但是集群不易扩展。

**一致性哈希**：这种方案均衡性好，集群扩展容易，只是实现复杂。代表有 Amazon 的 DynamoDB 和开源的 Cassandra。

**范围划分**：通常配合全局有序，复杂度在于合并和分裂。代表有 Hbase。

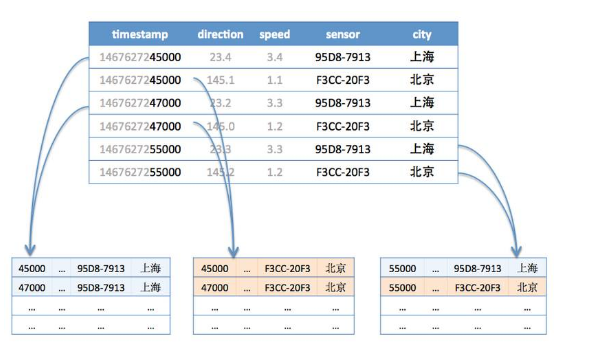
### 分片设计

分片设计简单来说就是以什么做分片，这是非常有技巧的，会直接影响写入读取的性能。

结合时序数据库的特点，根据 metric+tags 分片是比较好的一种方式，因为往往会按照一个时间范围查询，这样相同 metric 和 tags 的数据会分配到一台机器上连续存放，顺序的磁盘读取是很快的。再结合上面讲到的单机存储内容，可以做到快速查询。

进一步我们考虑时序数据时间范围很长的情况，需要根据时间范围再将分成几段，分别存储到不同的机器上，这样对于大范围时序数据就可以支持并发查询，优化查询速度。

如下图，第一行和第三行都是同样的 tag（sensor=95D8-7913;city= 上海），所以分配到同样的分片，而第五行虽然也是同样的 tag，但是根据时间范围再分段，被分到了不同的分片。第二、四、六行属于同样的 tag（sensor=F3CC-20F3;city= 北京）也是一样的道理。



p5- 时序数据分片说明

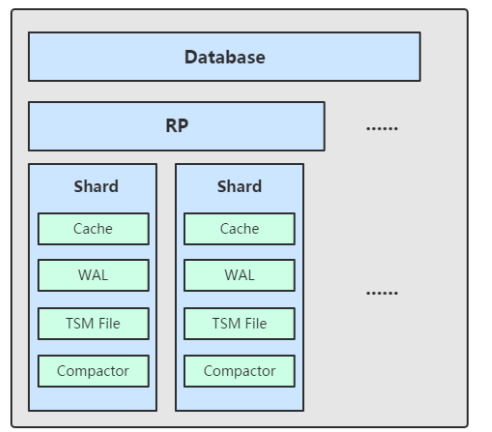
## 6. 真实案例

下面我以一批开源时序数据库作为说明。

### InfluxDB:

非常优秀的时序数据库，但只有单机版是免费开源的，集群版本是要收费的。从单机版本中可以一窥其存储方案：在单机上 InfluxDB 采取类似于 LSM tree 的存储结构 TSM；而分片的方案 InfluxDB 先通过+（事实上还要加上 retentionPolicy）确定 ShardGroup，再通过+的 hash code 确定到具体的 Shard。

这里 timestamp 默认情况下是 7 天对齐，也就是说 7 天的时序数据会在一个 Shard 中。



p6-Influxdb TSM 结构图（注 2）

### Kairosdb:

底层使用 Cassandra 作为分布式存储引擎，如上文提到单机上采用的是 LSM tree。

Cassandra 有两级索引：partition key 和 clustering key。其中 partition key 是其分片 ID，使用的是一致性哈希；而 clustering key 在一个 partition key 中保证有序。

Kairosdb 利用 Cassandra 的特性，将++<数据类型>+作为 partition key，数据点时间在 timestamp 上的偏移作为 clustering key，其有序性方便做基于时间范围的查询。

partition key 中的 timestamp 是 3 周对齐的，也就是说 21 天的时序数据会在一个 clustering key 下。3 周的毫秒数是 18 亿正好小于 Cassandra 每行列数 20 亿的限制。

### OpenTsdb:

底层使用 Hbase 作为其分布式存储引擎，采用的也是 LSM tree。

Hbase 采用范围划分的分片方式。使用 row key 做分片，保证其全局有序。每个 row key 下可以有多个 column family。每个 column family 下可以有多个 column。



上图是 OpenTsdb 的 row key 组织方式。不同于别的时序数据库，由于 Hbase 的 row key 全局有序，所以增加了可选的 salt 以达到更好的数据分布，避免热点产生。再由与 timestamp 间的偏移和数据类型组成 column qualifier。

他的 timestamp 是小时对齐的，也就是说一个 row key 下最多存储一个小时的数据。并且需要将构成 row key 的 metric 和 tags 都转成对应的 uid 来减少存储空间，避免 Hfile 索引太大。下图是真实的 row key 示例。



p7-open tsdb 的 row key 示例（注 3）

## 7. 结束语

可以看到各分布式时序数据库虽然存储方案都略有不同，但本质上是一致的，由于时序数据写多读少的场景，在单机上采用更加适合大吞吐量写入的单机存储结构，而在分布式方案上根据时序数据的特点来精心设计，目标就是设计的分片方案能方便时序数据的写入和读取，同时使数据分布更加均匀，尽量避免热点的产生。

数据存储是时序数据库设计中很小的一块内容，但也能管中窥豹，看到时序数据库从设计之初就要考虑时序数据的特点。后续我们会从其他的角度进行讨论。

* 注 1：来源 http://tristartom.github.io/research.html
* 注 2：来源 http://blog.fatedier.com/2016/08/05/detailed-in-influxdb-tsm-storage-engine-one/
* 注 3：来源 http://opentsdb.net/docs/build/html/user\_guide/backends/hbase.html

感谢[杜小芳](http://www.infoq.com/cn/author/%E6%9D%9C%E5%B0%8F%E8%8A%B3)对本文的审校。