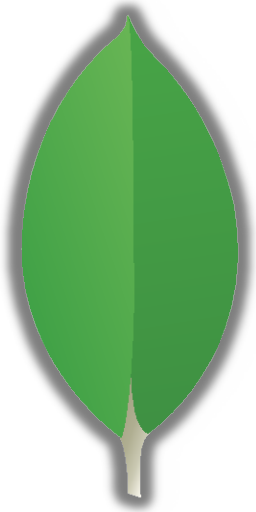


mongoDB

IoT系统中的MongoDB设计

- 张耀星
- 首席咨询顾问
- yaoxing.zhang@mongodb.com

开篇

- 通过一个实际IoT案例的分析，讨论
  - 如何使用MongoDB解决一个实际的需求？
  - 需要考虑哪些方面的问题？

理论篇

IoT系统的特点

海量数据入库

- 采样点越密，数据量越大
- 设备越多，数据量越大

灵活的数据模型

- 设备类型越多，模型越多变
- 软件迭代越快，模型变化越多

高并发访问

- 设备/用户越多，反馈数据越密集

区域分散性

- 设备分散在各地但需要集中处理

分析需求

- 数据的价值来自分析

数据乱序

- 设备网络问题导致的数据乱序

MongoDB的应对

海量数据入库

- 水平扩展

灵活的数据模型

- 原生支持

高并发访问

- 水平扩展

区域分散性

- 区域分片

分析需求

- Aggregation
- Spark Connector
- BI Connector

数据乱序

- 数据模型+应用端解决

结论

MongoDB非常适合支撑IoT系统

实践篇

案例需求介绍

- 涉及的IoT设备包括但不限于：
  - 电灯，门禁，插座等室内设备；
  - 员工笔记本定期上报数据；
  - 厂区穿梭车定期上报数据；
- 规划设备数量：1000万；
- 数据上报频率：平均5分钟，但依设备不同，上报频率为1分钟至1小时不等；
- 数据大小：0.2KB/条；
- 数据存储时间：1年以上；
- 数据使用：
  - 提取某设备某时间段内的所有数据供下游系统分析。50tps，2s；
  - 给定位置范围，给定时间段，获取所有数据。10tps，4s；

设计目标

1.MongoDB架构
2.满足需求的数据模型
3.MongoDB容量规划

一点数学计算

插入性能	**1000万/5分钟/60秒 ≈ 30000 次/秒**
文档总量	1000万 * 365天 * 24小时 * 60分钟 / 5分钟 = 10512亿 / 年
数据容量	10512亿 * 0.2KB ≈ 210TB
压缩后容量	210TB * 30%[1] ≈ 63TB

数据模型

- Q：什么是分桶模式？
- A：把n条数据聚合后放在一个文档里。

```
{
  _id: ObjectId("..."),
  sensorId: "传感器ID",
  region: "CN",
  groupId: 100,
  bucketTime: ISODate("2019-01-01T00:00:00+0800"),
  data: [
    {time: "00:00", value: {...},
    {time: "00:30", value: {...},
    {time: "01:00", value: {...},
    ...
  ]
}
```

假设每小时1个桶，
每30秒1条数据

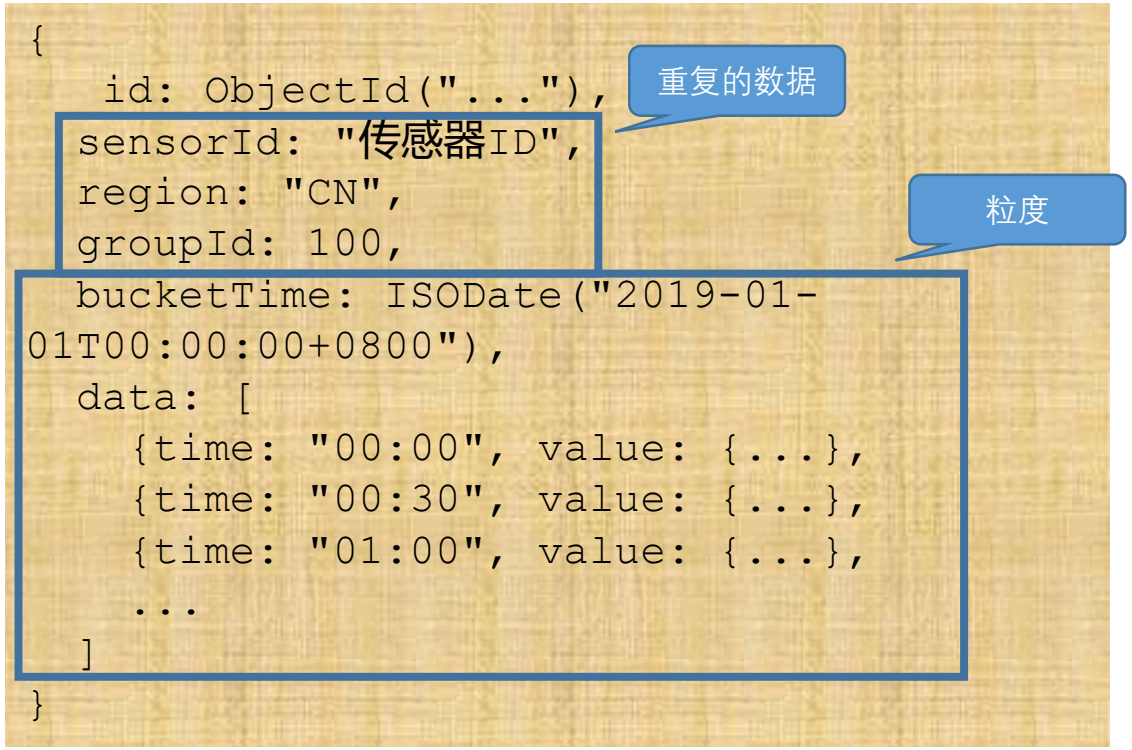
总共120条

分桶后数据量 = 总数据量 / 120

更深入的理解分桶

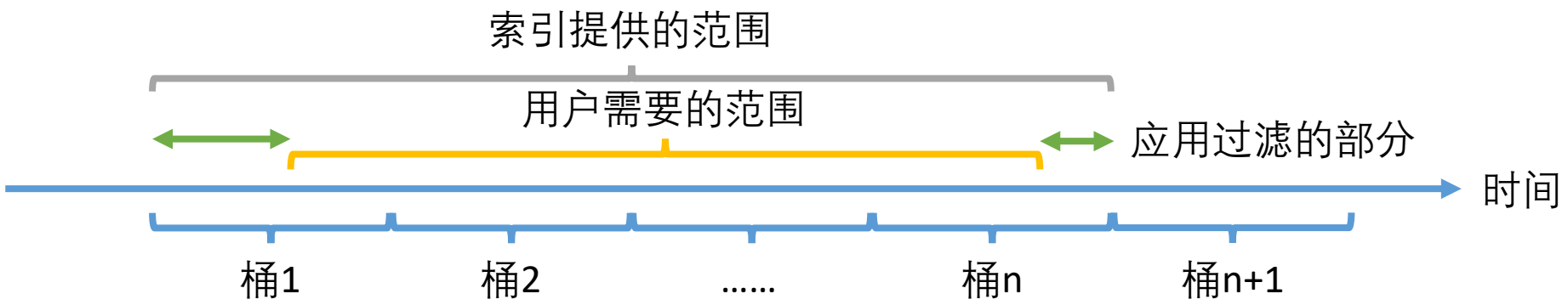
- 分桶模式背后的意义是什么？
  1. 大幅度减少重复数据带来的额外负担；
     1. 数据本身的IO消耗
     2. 修改索引带来的IO消耗
  2. IoT系统中的海量数据大部分时候没有必要进行细粒度的查询；

- 在高并发、海量数据应用场景中，每条数据减少一点开销，最后累计都将带来可观的收益！
- 土豪随意



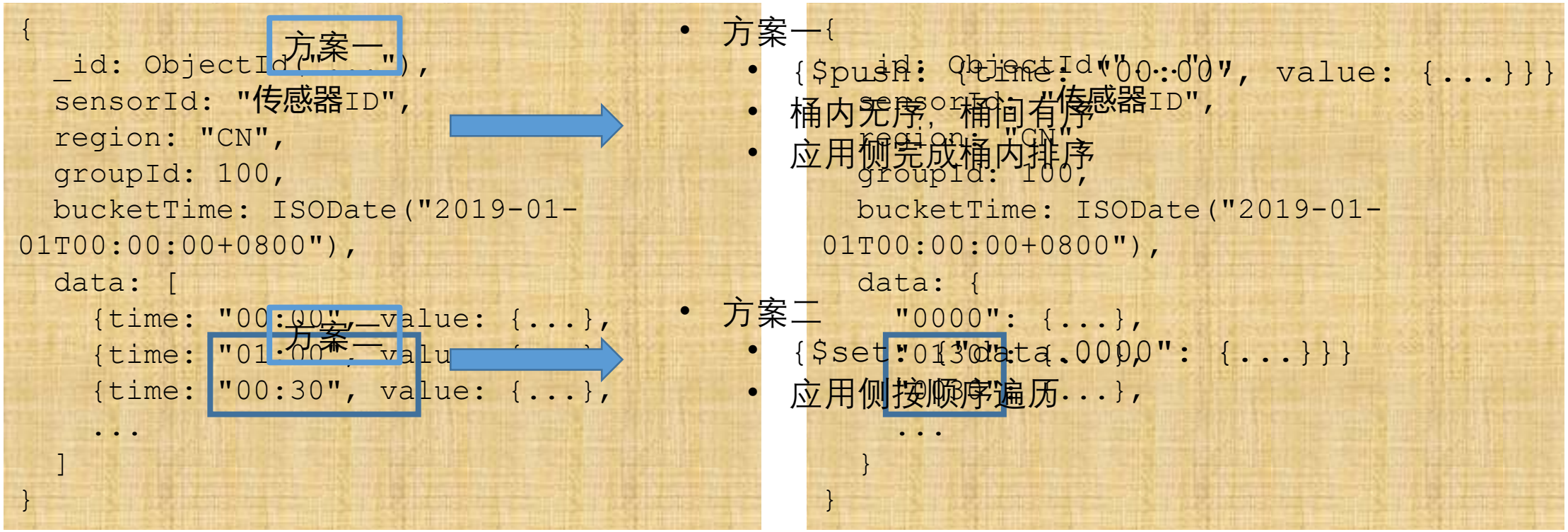
一个典型场景

- 本例中的场景：
  - 提取某设备某时间段内的所有数据供下游系统分析。50tps，2s；



还有一个问题

- IoT设备因为网络不稳定造成的数据乱序问题怎么解决？



另一个场景

- 本例中的另一个场景
  - 给定位置范围，给定时间段，获取所有数据。10tps，4s；

```
{
  _id: ObjectId("..."),
  sensorId: "传感器ID",
  region: "CN",
  groupId: 100,
  bucketTime: ISODate("2019-01-
01T00:00:00+0800"),
  data: [
    {time: "00:00", loc: {...}},
    {time: "01:00", loc: {...}},
    {time: "00:30", loc: {...}},
}
```

db.coll.createIndex({"data.loc": "2d"})

```
{
  _id: ObjectId("..."),
  sensorId: "传感器ID",
  region: "CN",
  groupId: 100,
  bucketTime: ISODate("2019-01-
01T00:00:00+0800"),
  data: {
    "0000": {loc: {...}},
    "0130": {loc: {...}},
    "0030": {loc: {...}},
}
```

db.coll.createIndex(???)

小结

- Step 1: 了解MongoDB的优势
- Step 2: 列出可选解决方案
- Step 3: 根据实际情况选择最佳方案

下一步该干什么？

- 如何写数据？
  - update({sensorId: "...", bucketTime: ISODate("...")}, {$push: {}}, {upsert: true});
- 如何读数据？
  - find({sensorId: "...", ISODate("...")}})
  - find({bucketTime..., loc: {$geoWithin: {...}}})
- 如何聚合数据？
  - 按何种粒度预处理？

索引怎么建？

然后？

造假！

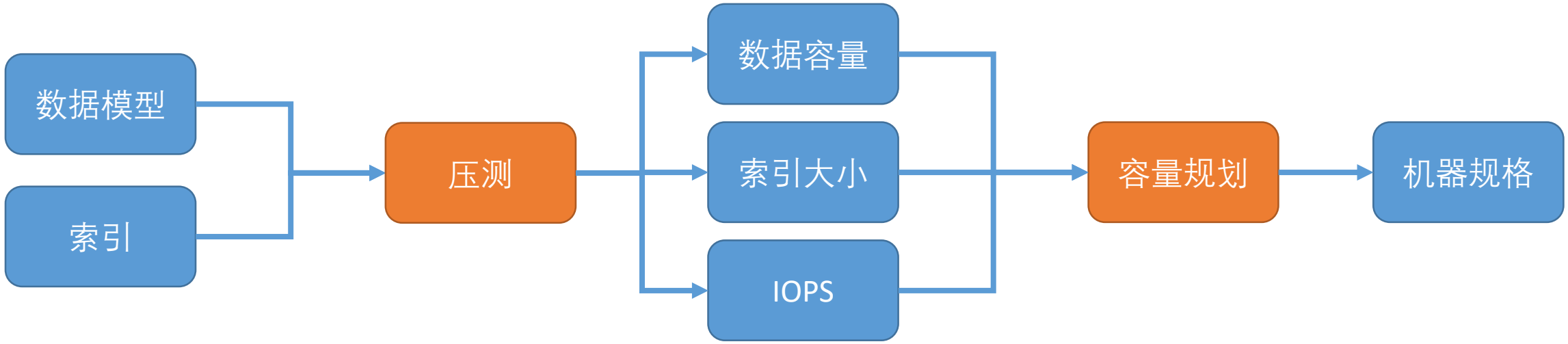
造假的方式

- benchRun: mongo shell集成的压测工具
  - https://github.com/mongodb/mongo/wiki/JavaScript-Performance-Testing-Harness
- POCDriver: MongoDB工程师编写的压测工具
  - https://github.com/johnlpage/POCDriver/
- Java Faker: Ruby Faker的Java移植版，帮助你更精准地造假
  - https://github.com/DiUS/java-faker
- YCSB: (略)

造假的目的

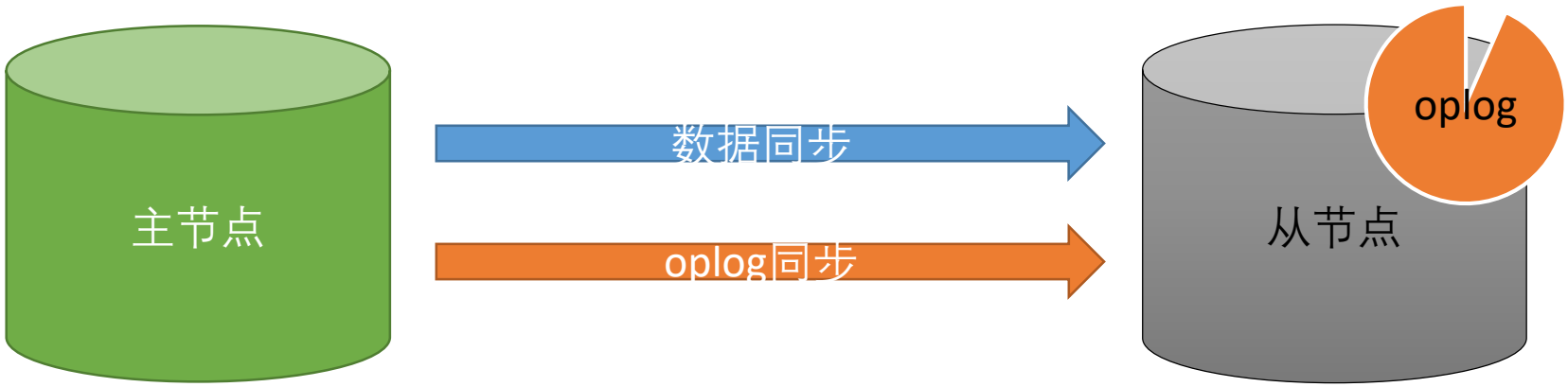
- 取得压测结果，评估性能指标
- 取得参考压缩率，评估容量
- 取得索引大小，评估内存容量

目前得到的信息



容量规划的原则——分片容量

- 视节点硬件性能，每个分片数据量建议不超过3TB
  - 压缩前
  - 不含索引



210TB / 3TB = 70 片

容量规划的原则——内存

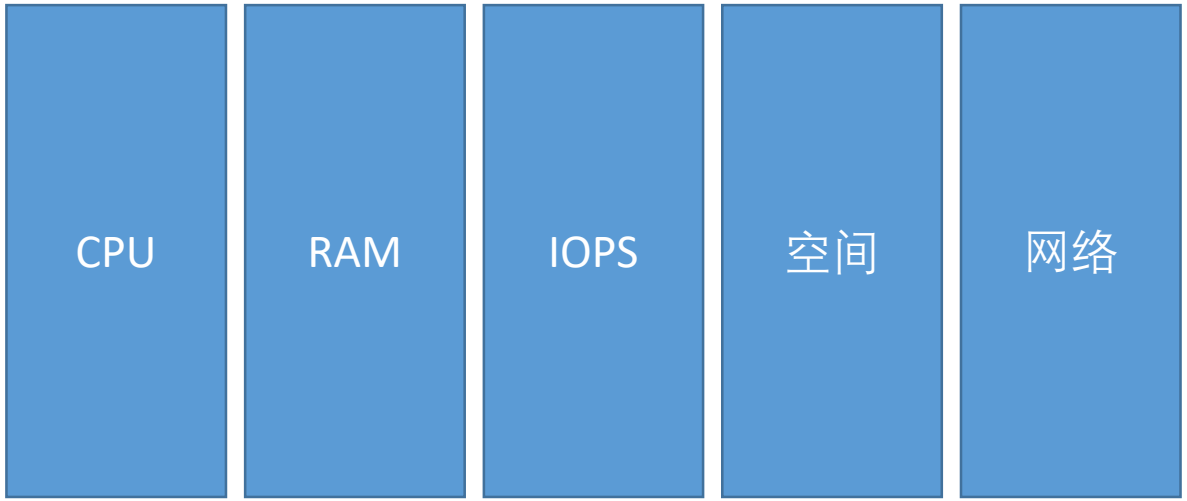
- 内存必须容纳热数据的索引
  - 索引在磁盘和内存中都是前缀压缩的
- CacheSizeGB = (RAM - 1GB) * 50%

- 70片共有多少内存？

容量规划的原则——IOPS

- 分片IOPS总和大于测试推算出的总IOPS
  - 建议使用SSD
  - 重读写时建议RAID 10
  - 重写时建议RAID 0
  - 强烈不推荐RAID 5/RAID 6

木桶原理



综合以上信息

- 70个分片是否足够？
- 70个分片应该分布在多少台物理机上？
- 如何分布以避免单点失败？

最后一点

- 真的需要70个分片吗？
- 按需增加！
- 提前扩容！

Q&A