

hw9:LSMT & TS DB

请你按照你的理解，用Word文档答复下列问题：

1. 请阐述日志结构数据库适合什么样的应用场景？(1分)
2. 请阐述日志结构数据库中的读放大和写放大分别是什么意思？(1分)
3. 日志结构合并树中，WAL的作用是什么？(1分)
4. 请你在自己的机器上安装 InfluxDB，并像课程上所演示的一样监控你的笔记本电脑的状态，在Web界面的Explore中截图贴在Word文档中，并根据截图简要说明一下你的笔记本电脑的运行状态。(2分)
 - 请提交包含上述问题答案的文档

评分标准：

- 上述问题答案不唯一，只要你的说理合理即可视为正确。

Q1

Log以Append的模式追加写入，不存在删除和修改。这种结构虽然大大提升了数据的写入能力，却是以牺牲部分读取性能为代价，故此这种结构通常适合于**写多读少**的场景



In computer science, the **log-structured merge-tree** (also known as **LSM tree**, or **LSMT**[1]) is a data structure with performance characteristics that make it attractive for providing indexed access to files with **high insert volume**, such as transactional log data.
(适合大量插入数据的场景) —— from Wikipedia Log-structured merge-tree

https://en.wikipedia.org/wiki/Log-structured_merge-tree

Q2

读放大(Read Amplification)问题：log-structure db一般使用层次化的存储。不同层级存储着不同版本的数据，需要一层一层（从上到下）查找，直到找到需要的数据。这个过程可能需要不止一次的I/O，有可能寻找一份数据需要遍历所有的数据文件。特别是range query，影响特别明显。

eg, 在LSMtree中，如果查询一个没有在数据库的值，需要遍历完所有的文件才知道这个数据不在数据库中。

写放大 (Write amplification) 问题：实际写入 HDD/SSD 的数据大小和程序要求写入数据大小之比。正常情况下，HDD/SSD 观察到的写入数据多于上层程序写入的数据。由于写数据可能触发合并，压缩操作，写一个数据可能导致很多层的合并压缩，甚至可能到最后一层的合并压缩。

Q3

WAL : Write Ahead Log

WAL主要作用是用来恢复发生 crash时 memtable中的未committed中的数据。 所以WAL 的写入需要优先于memtable，且每一次写入都需要flush，这也是write head的由来。

对RocksDB 的每次写操作都必写到两个地方：

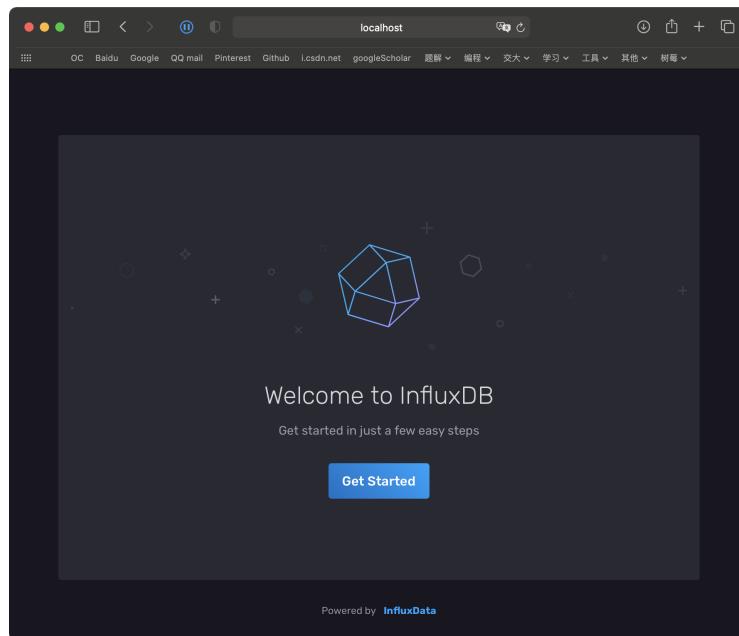
- 1) 基于内存的数据结构memtable（达到quota 后会flush 至SST file）。
- 2) 预写日志-Write Ahead Log (WAL)。

如果出现异常情况，WAL 可以用来完整恢复memtable 中的数据，恢复db 的原有的状态。通过每次用户写之后flush WAL，来保证进程crash 后的一致性。

Q4

brew services restart influxdb

localhost:8086

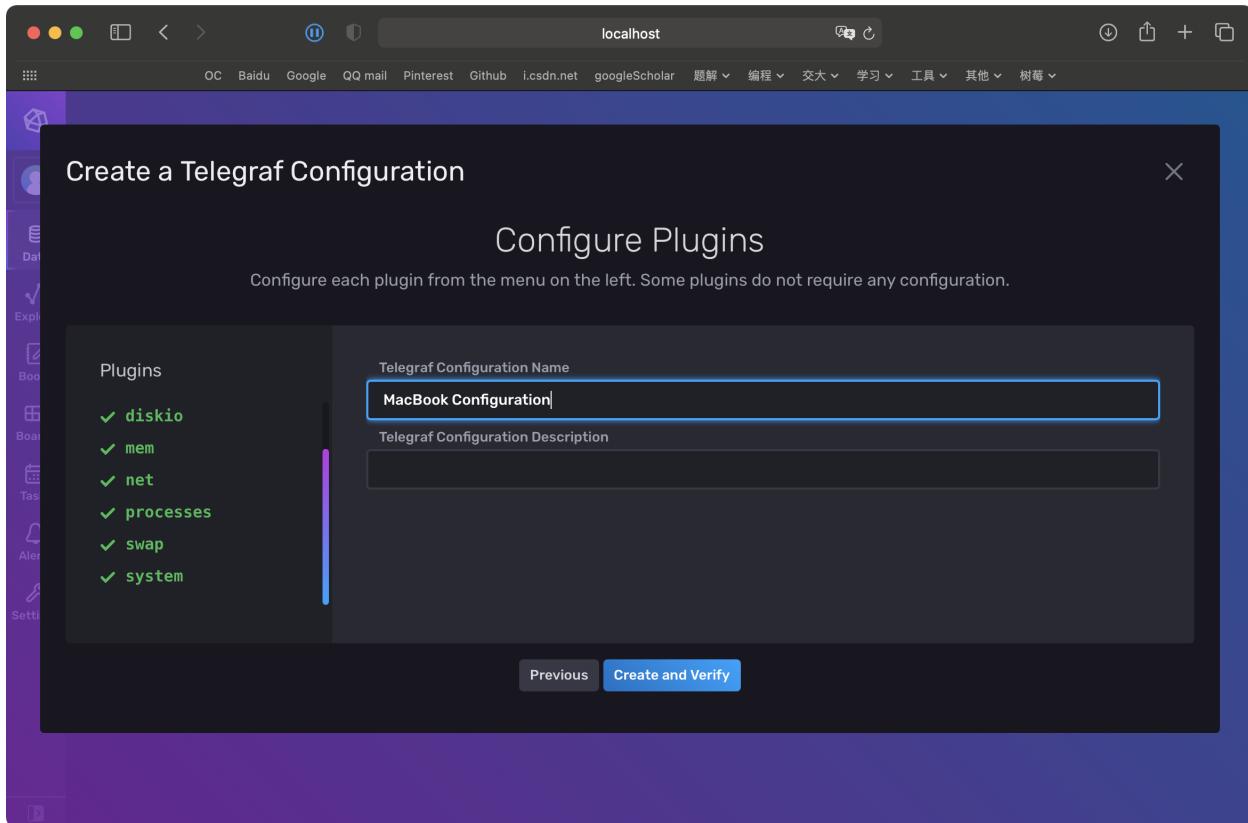


name: root

password: password

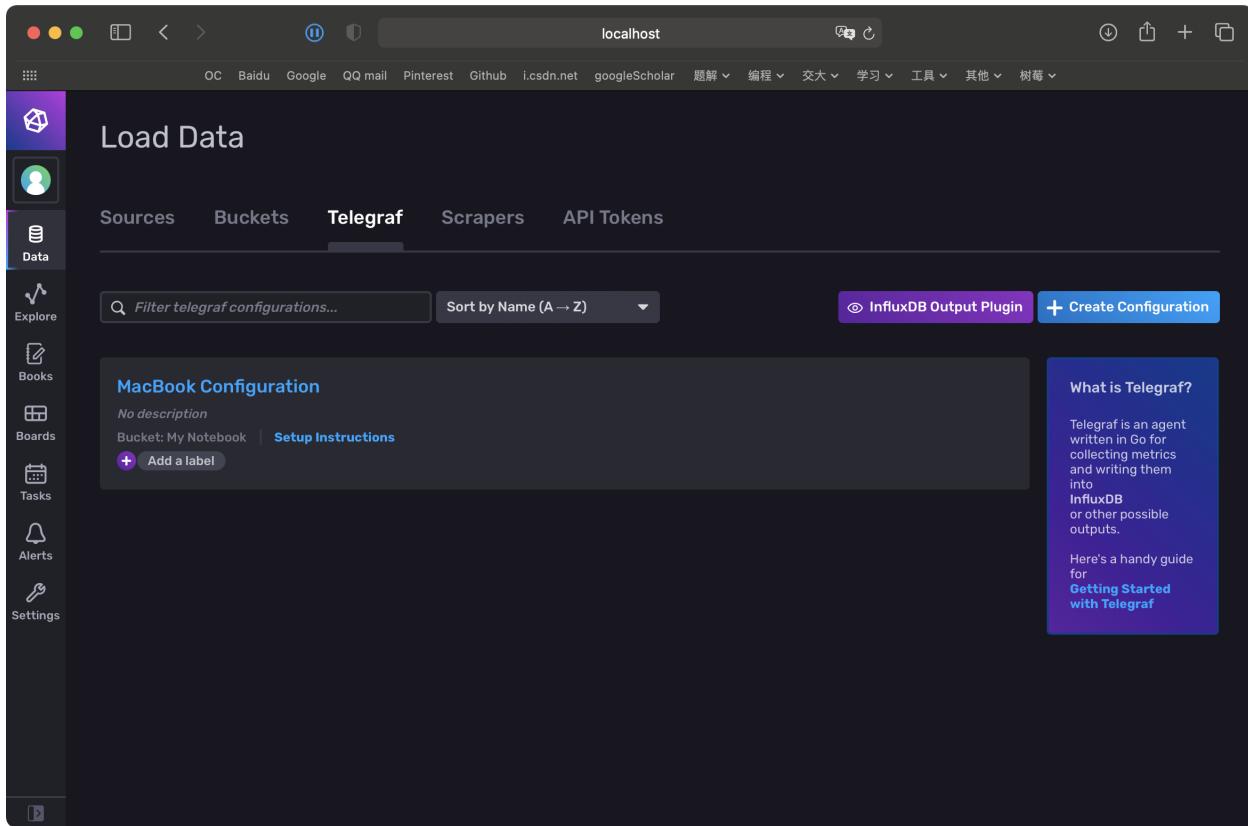
bucket: My Notebook

org: SJTU



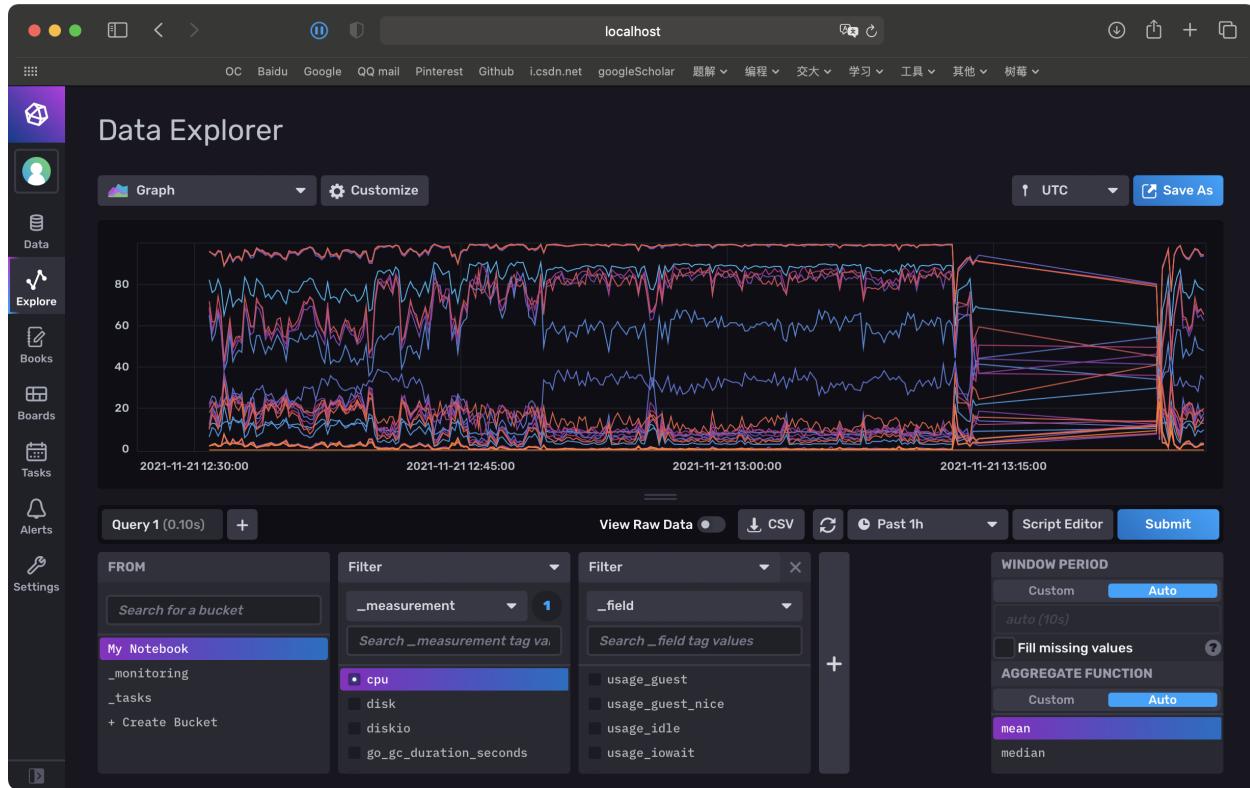
生成telegraph configuration

```
export INFLUX_TOKEN=IzTEgVDLlv1zdyPr_ocS0tpfiUprYKDNdkkCvlhyX41y_00axqbAg8bQgQxCPLNIE_bMezgX3Iq5c3_b_BfMA==
```

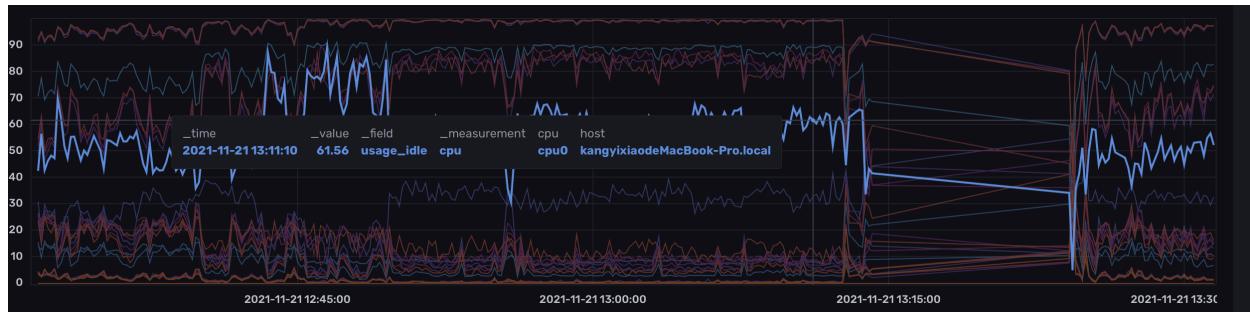


```
● ● ● telegraf --config http://localhost:8086/api/v2/telegrafs/087c99ed8fe4b000 — 106×34
(base) ~ export INFLUX_TOKEN=IzTEgVDLlv1zdyPr_ocS0tpfiUprYKDNdkkCvlhyX41y_00axqbAg8bQgQxCPLNIE_bMezgX3
Iq5c3_b_BfMA==
(base) ~ telegraf --config http://localhost:8086/api/v2/telegrafs/087c99ed8fe4b000
zsh: command not found: telegraf
(base) ~ brew install telegraf
Updating Homebrew...
==> Downloading https://ghcr.io/v2/homebrew/core/telegraf/manifests/1.20.4
#####
==> Downloading https://ghcr.io/v2/homebrew/core/telegraf/blobs/sha256:bab8e8a7e0a31e12
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sha256:ba
#####
==> Pouring telegraf--1.20.4.big_sur.bottle.tar.gz
==> Caveats
To restart telegraf after an upgrade:
  brew services restart telegraf
Or, if you don't want/need a background service you can just run:
  /usr/local/opt/telegraf/bin/telegraf -config /usr/local/etc/telegraf.conf -config-directory /usr/local/etc/telegraf.d
==> Summary
  /usr/local/Cellar/telegraf/1.20.4: 9 files, 173.4MB
(base) ~ telegraf --config http://localhost:8086/api/v2/telegrafs/087c99ed8fe4b000
2021-11-21T12:30:24Z I! Starting Telegraf 1.20.4
2021-11-21T12:30:24Z I! Loaded inputs: cpu disk diskio mem net processes swap system
2021-11-21T12:30:24Z I! Loaded aggregators:
2021-11-21T12:30:24Z I! Loaded processors:
2021-11-21T12:30:24Z I! Loaded outputs: influxdb_v2
2021-11-21T12:30:24Z I! Tags enabled: host=kangyixiaodeMacBook-Pro.local
2021-11-21T12:30:24Z I! [agent] Config: Interval:10s, Quiet:false, Hostname:"kangyixiaodeMacBook-Pro.local",
", Flush Interval:10s
```

安装并运行telegraph



每一条线显示第几个CPU以及field,如下图：

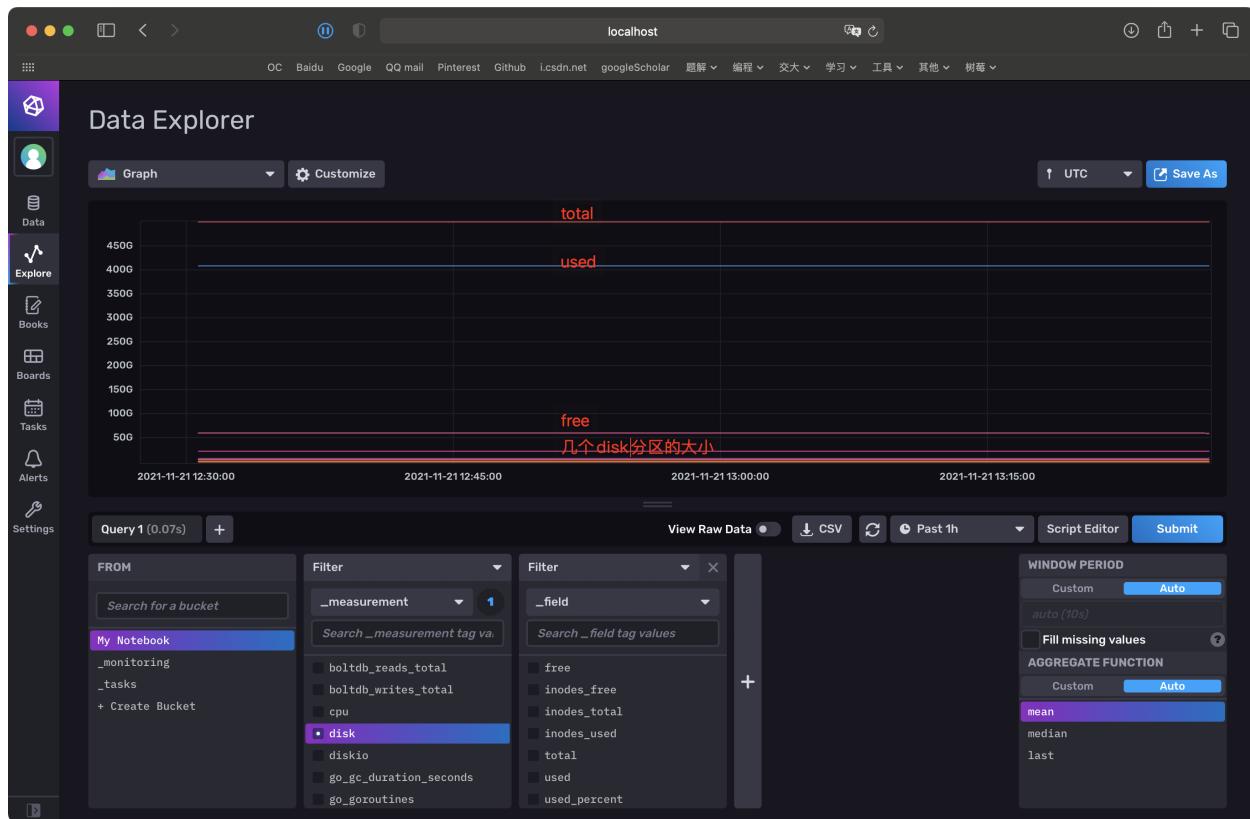


第0个CPU的空闲 (idle)情况

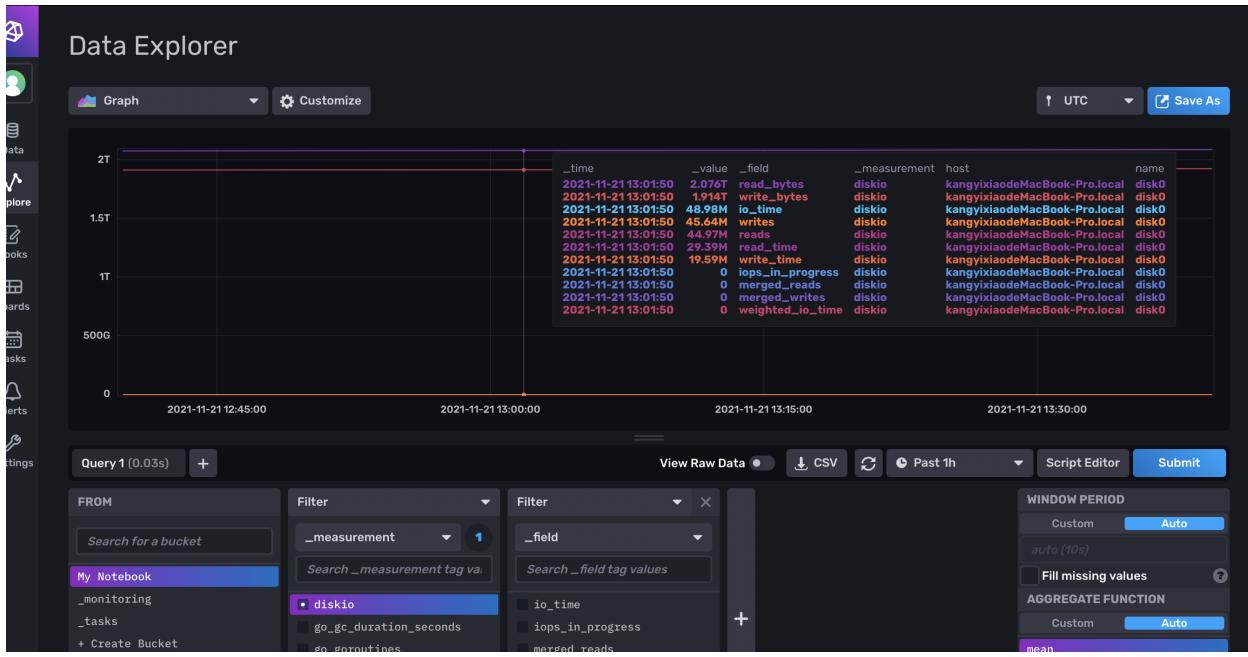


0号CPU的工作情况 (system), 可以看见和上图Idle的情况基本是相反的

中间有一段直线，是将电脑休眠测试的结果，显示CPU的运行情况变化平缓，这个和预期符合。同时看出CPU的运行变化每一个核负载不一样，单个核的变化有波动但是核与核之间的负载大小排序基本不变



监测disk使用情况，最上面是total的disk占用，第二条是使用的disk,第三条是free的disk，下面几条是disk1f1, disk1f2.. disk2f1等分区的占用情况，可以看出变化比较小



可以检测每一个时间点上的diskIO 操作

reference

安装

这篇将会介绍怎么安装、运行和配置InfluxDB。安装InfluxDB包需要 root 或是有管理员权限才可以。InfluxDB默认使用下面的网络端口：TCP端口8086用作InfluxDB的客户端和服务端的http api通信 TCP端口8088给备份和恢复数据的RPC服务使用 另外，InfluxDB也提供了多个可能需要自定义端口的插件，所以的端口映射都可以通过配置文件修改，对于默认安装的InfluxDB，这个配置文件位

👉 <https://jasper-zhang1.gitbooks.io/influxdb/content/Introduction/installation.html>

LSM-Tree 的写放大写放大、读放大、空间放大RockDB 写放大简单分析参考文档

本文缺少实际的实践经验。全部来自在网上的“道听途说”和自己的“胡思乱想”。基于 LSM-Tree 的存储系统越来越常见了，如 RocksDB、LevelDB。LSM-Tree 能将离散的随机写请求都转换成批量的顺序写请求（WAL + Compaction），以此提高写性能。但也带来了一

<https://cloud.tencent.com/developer/article/1352666>



Rocksdb WAL实现及源码详解

作为通用的单机存储引擎，一个基本的crash safe功能是需要提供的，用来保证异常时的数据一致性。像我们rocksdb所在节点出现断电异常，节点死机等情况，内存中的数据是会丢失的。此时，需要rocksdb提供一种机制，能够在这种异常情况下尽可能地保证数据的一致性。

知 <https://zhuanlan.zhihu.com/p/343323703>

