分布式系统的基石

——Id生成器中的学问

大嘴

xuhaifeng@yuewen.com

议题

- id基本概念
- 传统id生成器的类型&优缺点
- 改进型id生成器的类型&优缺点
- 目前较流行id生成器的类型&优缺点
- 我们特有需求id的策略和算法

id基本概念

- id：一个对象的唯一标示。
- 示例：身份证号
- 作用：
  - 标识对象
  - 分库分表
  - 索引、排序

传统id生成办法

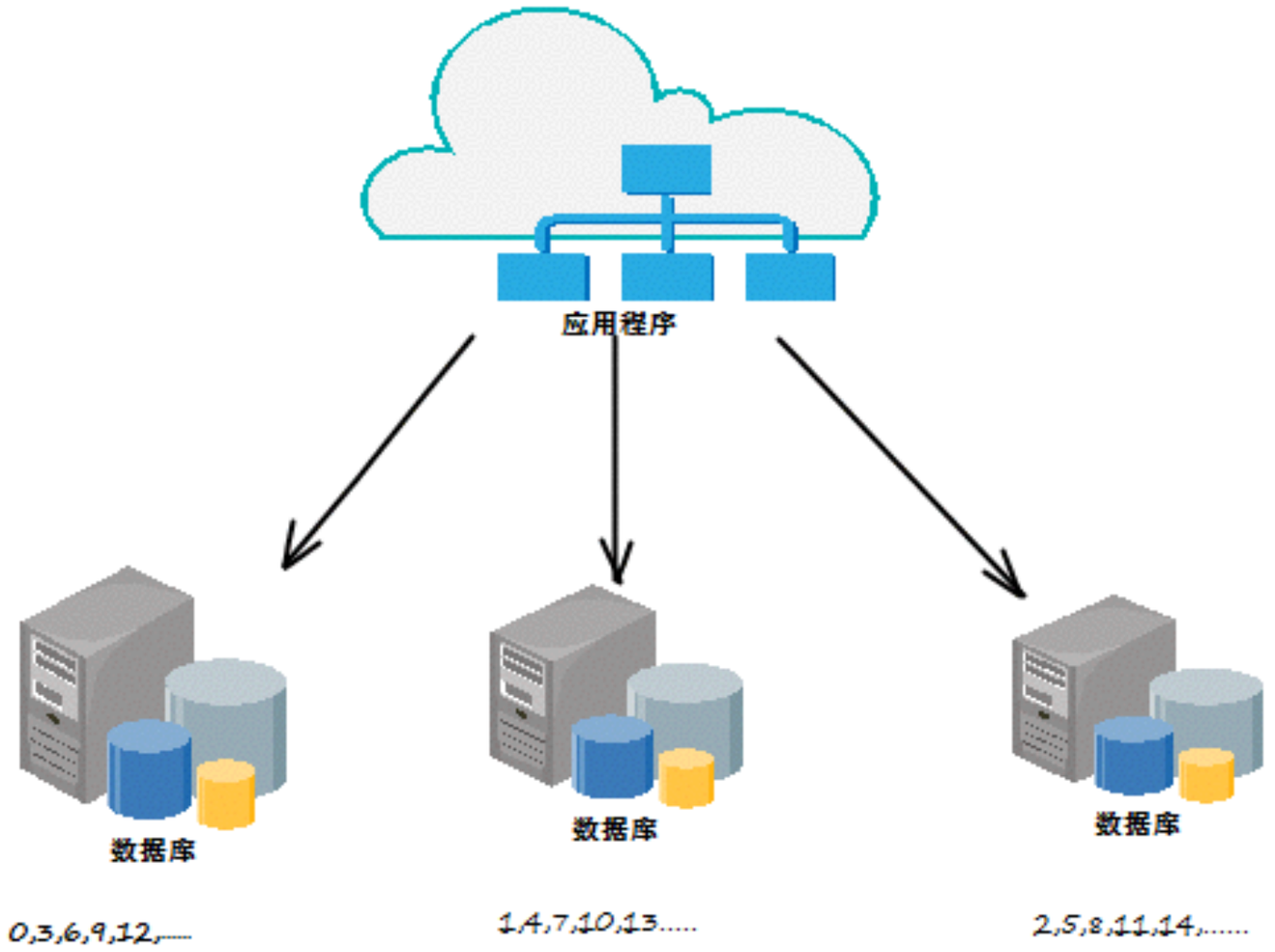
- int型，自增
- 数据库实现
- identity 关键字，记得带上@@
- 和insert必须在一个transaction

传统id缺点

- 数据库扩展性差
- 事务性要求高
- 步长固定
- 无法应对分库分表

数据库改进型id生成办法

- 多数据库
- 自定义步长



数据库改进型id的优缺点

序号	优点	缺点
1	可以简单的分库分表	步长严重依赖于数据库数量
2	数据库压力减小	扩展性：增加数据库除了导数据，还需要更新id字段步长信息
3		分库分表算法限死，为了id的均衡性，貌似只有loop算法
4		几乎无法为id排序

应用程序改进型id的办法

- 典型的 UUID/GUID，宇宙唯一性
- 类UUID，自己算法拼凑String
  - 可以拼你任何想要的内容
  - 典型的：逻辑机器号+类型+时间戳+顺序号

应用程序改进型id的优缺点

序号	优点	缺点
1	自定义性强，可以为所欲为	String太长了
2	可以轻松实现分库分表运算	String截取内容转换成int或者是直接hash都需要额外的运算
3	人类识别度高，后期运维方便	UUID/GUID无业务意义，人类识别度也不高
4	本地生成，无延时	一样无可排序性，对索引不友好

需要什么样的id？

我们使用id的地方

- 书、卷、章节等id
  - 书需要自定义数据库编号
  - 卷、章节需要对应带上数据库编号后取模入表
- 错误码
  - 递增，因按照时间做路由故需要时间信息

id的作用

- 最基本：作为数据库记录的主键
- 加强型：被索引，对索引友好
- 附带价值：作为分库分表的依据
- 扩展功能：对象唯一标识，比如sessionid、批号、错误号

具体分析id

- 分库分表：时间戳、随机数、类型位、库位
- 高可用性：机器位
- 数据可读性必须强
- 递增还是随机？每秒递增？？累加递增？？ 随机碰撞？？

一个id应该具备的条件

- 唯一，必须唯一
- 短，尽可能的短
- 生成速度足够快
- 运算足够简单，快速
- 附带实体业务信息，比如时间、类型等
- 部分信息可以自定义，比如路由信息
- 不仅机器能识别，人类也可以识别
- 对索引友好
- 根据业务规则，能自定义排序等业务规则

确定方向

- 必须足够短，最好是uint32，最长uint64
- 必须系统原生支持，不需要扩展类型
- 比较运算足够快
- 必须递增，可排序并对索引友好
- id必须带业务性质，符合望文生义原则，通过id可以知道这个数据存在的数据库、表等信息，如果是错误号，必须能知道所发生的服务器

snowflake算法—twitter

- 0XFFFF FFFF FFFF FFFF， uint64的最大值
- 组成：时间戳-机器位-随机数
- 41位 + 10位 + 12位
- 9223 3720 3257 7650 688
- 0111 1111 1111 1111 1111 1111 1111 1111 1111 1111
- 0000 0001 0001 0000 0100 0000 0000 0000
- 2199023251472-264-0

snowflake算法—不满意

- 没有类型信息
- 二进制可读性太差
- 分库分表没有数据库定位信息

十进制id

- 1844 6744 0737 0955 1656， uint64的最大值
- 4294967295-0000-01-1-00
- 算法：时间戳-随机数位-类型位-机器位-库位
- 具体位数：10位 + 4位 + 2位 + 1位 + 2位
- 为什么会比最大值少了一位？

递增性

- 随机数变成递增数，随机性无法解决可能的碰撞
- 累加递增
  - 429496-0000-01-1-00
  - 429497-0001-01-1-00
  - 429498-0002-01-1-00
- 每秒递增
  - 429496-0000-01-1-00
  - 429497-0000-01-1-00
  - 429498-0000-01-1-00

递增约束

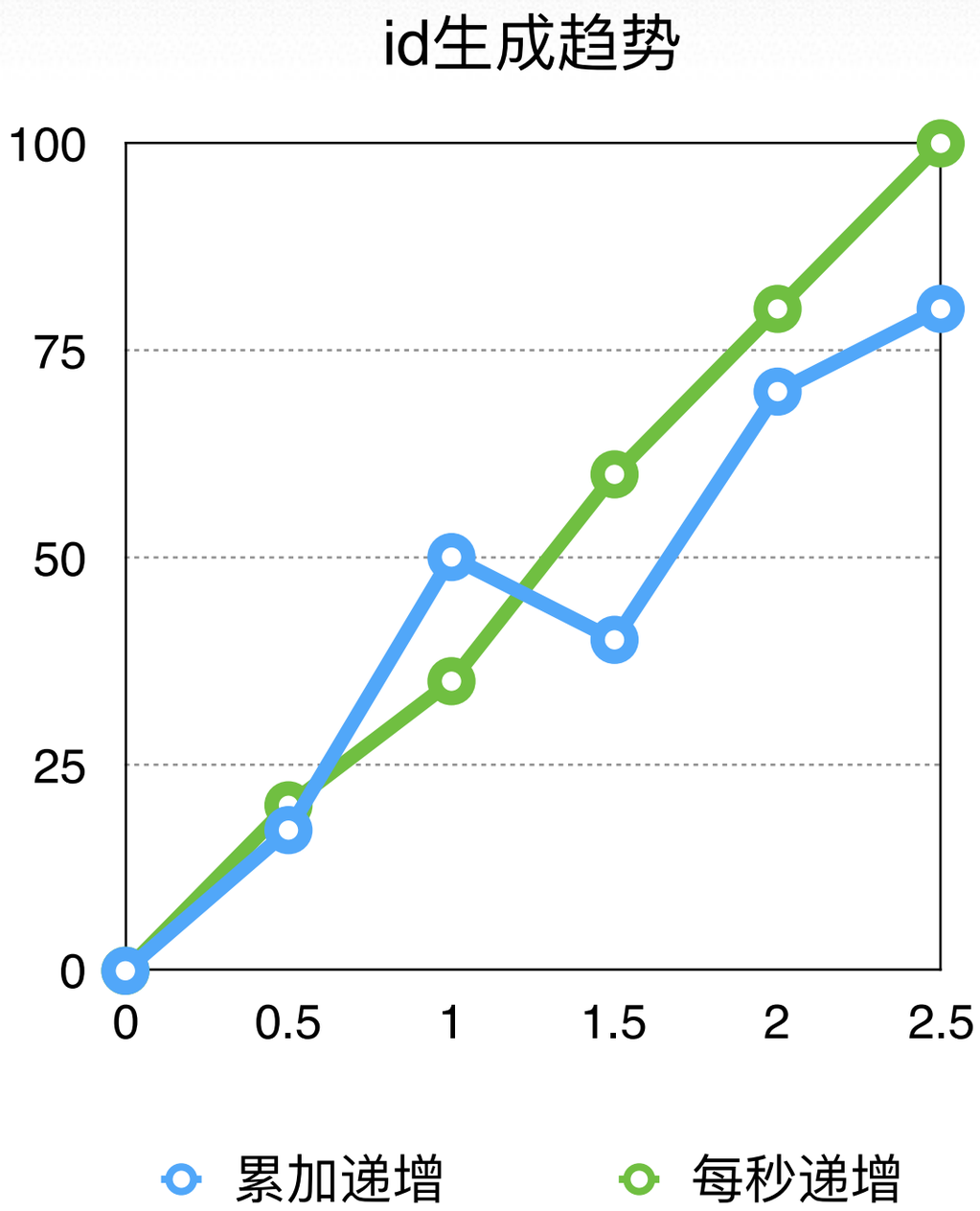
- 累加递增：适合分库分表，可以让每个表比较均衡的存储数据
- 每秒递增：适合做排序，切记不能~~分库分表~~，数据会出现严重的不平衡

真的严格递增吗？

递增id的坑

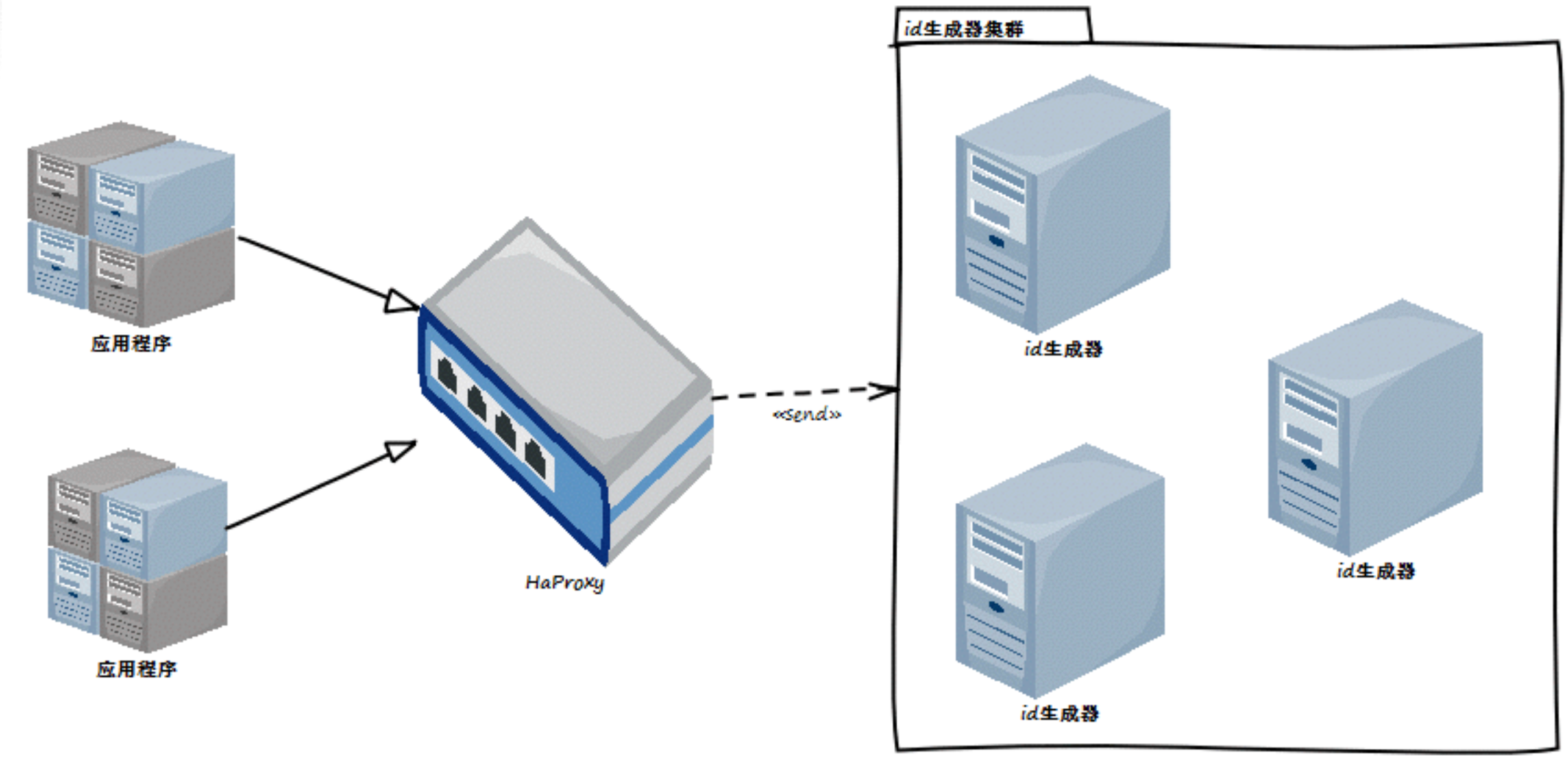
- 累加递增：秒内进位
  - 429497-9998-01-1-00
  - 429498-9999-01-1-00
  - 429498-0000-01-1-00
- 每秒递增：设计成每秒10k个，超过不会放出id
  - 429496-0000-01-1-00
  - 429497-0000-01-1-00
  - 429498-0000-01-1-00

递增规则



- 累加递增：长时间（2s内）内保证单调递增，短时间（1s内）内不保证单调递增
- 每秒递增：它肯定是递增的，因为每秒都会从0开始，单位时间内都是单调递增

服务器模型

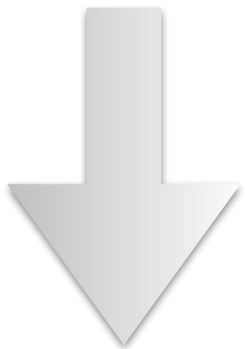


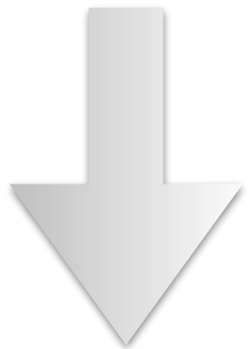
- 服务器无状态，无中心化设计，可以水平扩展
- 最多10台服务器，是不是少了？
  - 10(台) * 100(类型数) * 10000(每秒最大数量)

分库分表使用

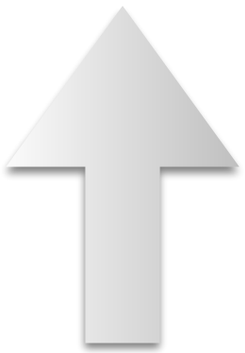
按照时间分

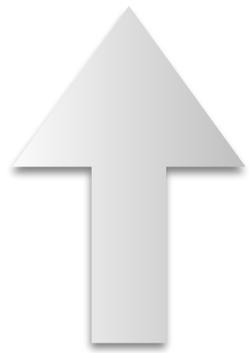
类型





4294967295-0000-01-1-00





分表位

分库位

错误号使用

用户可见 → 映射 → 真实系统错误号

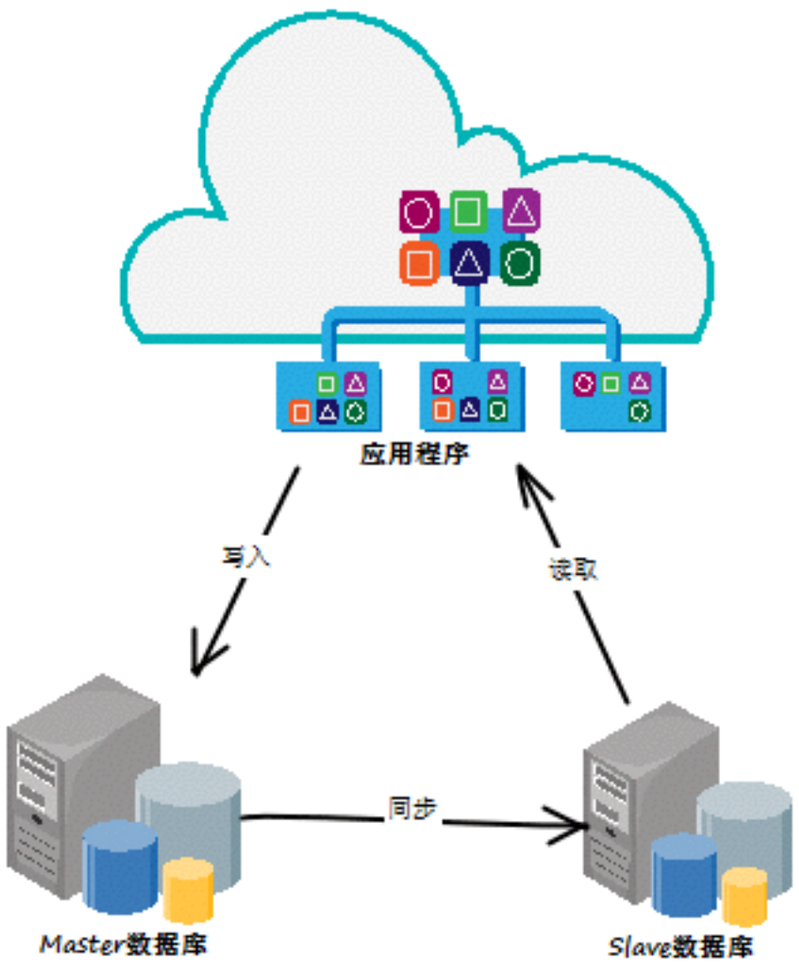
4294967295-00001-0000 → exception number

- 4294967295：时间戳
- 00001：机器号
- 0000：随机号

后来，产品经理来了...

我不管你是不是分库分表，我只要不错序！！

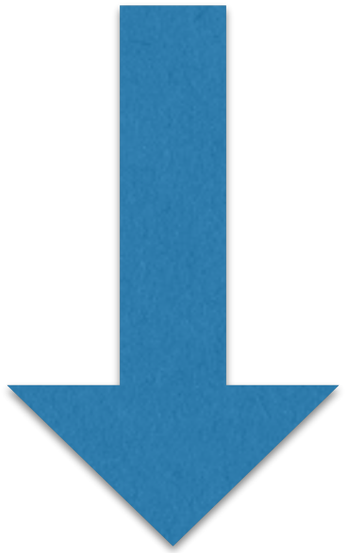
章节编号问题



- 应用程序将章节信息写入master数据库
- 从slaver数据库中读取章节信息
- m->s 数据同步
- 数据同步需要时间，结果章节的id号重叠

章节编号的业务属性

- 章节id必须可按照章节顺序排序，步长固定
- 排序id必须自增，但可以往前跳帧
- 每本书的排序id都独立，不能混淆



id生成器保存章节的最后状态

章节id

按照时间分

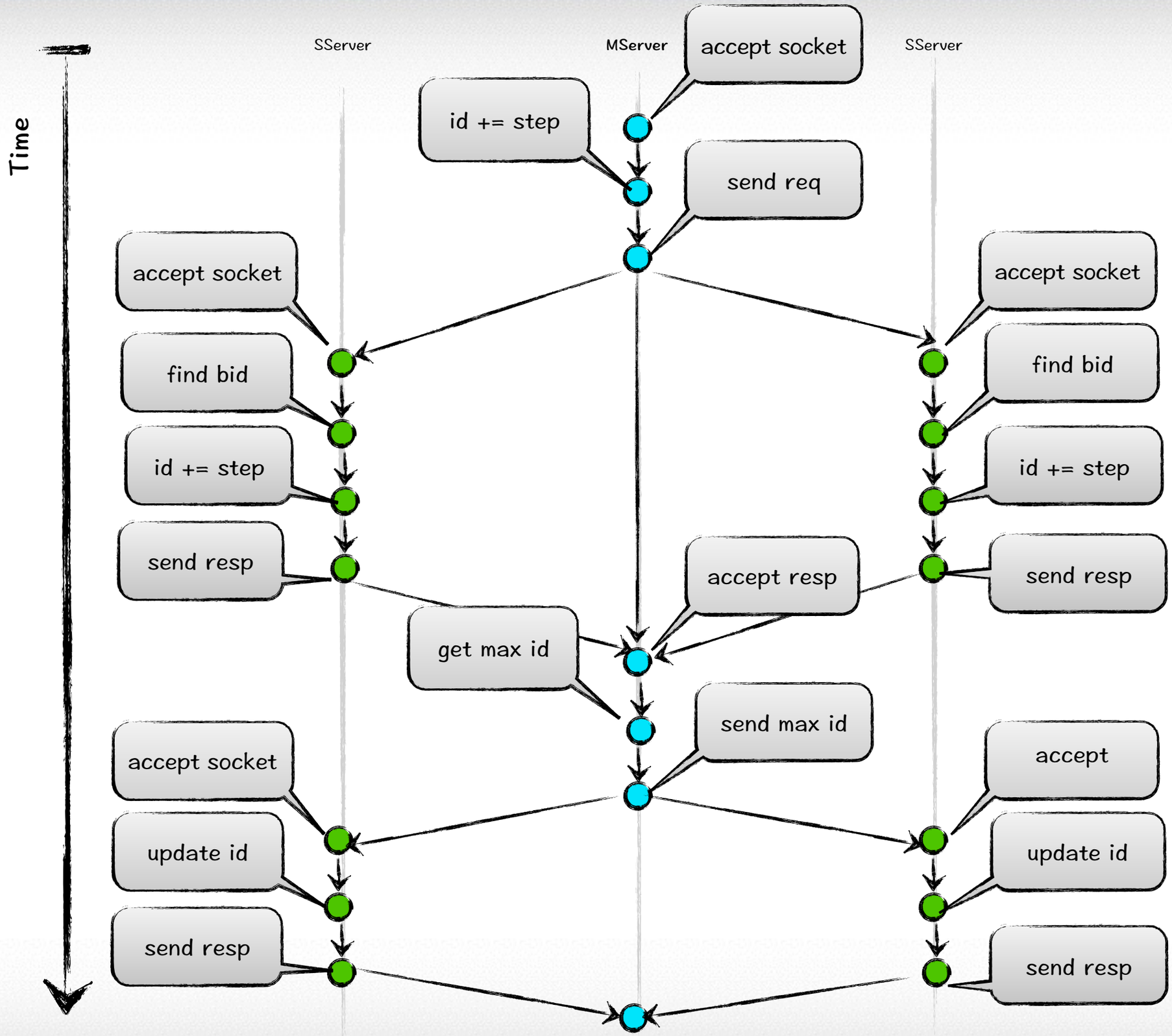
分库位

20160423-000-00-000100

分表位

章节排序号

章节编号过程


Time
SServer
MServer
SServer
accept socket
id += step
send req
accept socket
accept socket
find bid
find bid
id += step
id += step
send resp
send resp
accept resp
send resp
get max id
send max id
accept socket
accept
update id
update id
send resp
send resp

id就是一个19位的数字

技术含量不在于纯技术，
而在于对系统的架构控制

谢谢大家