分布式系统中唯一 ID 的生成方法

2017/02/05 · IT技术 · 分布式

分享到： 3

原文出处： darktea

本文主要介绍在一个分布式系统中, 怎么样生成全局唯一的 ID

一, 问题描述

在分布式系统存在多个 Shard 的场景中, 同时在各个 Shard 插入数据时, 怎么给这些数据生成全局的 unique ID?

在单机系统中 (例如一个 MySQL 实例), unique ID 的生成是非常简单的, 直接利用 MySQL 自带的自增 ID 功能就可以实现.

但在一个存在多个 Shards 的分布式系统 (例如多个 MySQL 实例组成一个集群, 在这个集群中插入数据), 这个问题会变得复杂, 所生成的全局的 unique ID 要满足以下需求:

1. 保证生成的 ID 全局唯一
2. 今后数据在多个 Shards 之间迁移不会受到 ID 生成方式的限制
3. 生成的 ID 中最好能带上时间信息, 例如 ID 的前 k 位是 Timestamp, 这样能够直接通过对 ID 的前 k 位的排序来对数据按时间排序
4. 生成的 ID 最好不大于 64 bits
5. 生成 ID 的速度有要求. 例如, 在一个高吞吐量的场景中, 需要每秒生成几万个 ID (Twitter 最新的峰值到达了 143,199 Tweets/s, 也就是 10 万+/秒)
6. 整个服务最好没有单点

如果没有上面这些限制, 问题会相对简单, 例如:

1. 直接利用 UUID.randomUUID() 接口来生成 unique ID (http://www.ietf.org/rfc/rfc4122.txt). 但这个方案生成的 ID 有 128 bits, 另外, 生成的 ID 中也没有带 Timestamp
2. 利用一个中心服务器来统一生成 unique ID. 但这种方案可能存在单点问题; 另外, 要支持高吞吐率的系统, 这个方案还要做很多改进工作 (例如, 每次从中心服务器批量获取一批 IDs, 提升 ID 产生的吞吐率)
3. Flickr 的做法 (http://code.flickr.net/2010/02/08/ticket-servers-distributed-unique-primary-keys-on-the-cheap/). 但他这个方案 ID 中没有带 Timestamp, 生成的 ID 不能按时间排序

在要满足前面 6 点要求的场景中, 怎么来生成全局 unique ID 呢?
Twitter 的 Snowflake 是一种比较好的做法. 下面主要介绍 Twitter Snowflake, 以及它的变种

二, Twitter Snowflake

https://github.com/twitter/snowflake
Snowflake 生成的 unique ID 的组成 (由高位到低位):

- 41 bits: Timestamp (毫秒级)
- 10 bits: 节点 ID (datacenter ID 5 bits + worker ID 5 bits)
- 12 bits: sequence number

一共 63 bits (最高位是 0)
unique ID 生成过程:

- 10 bits 的机器号, 在 ID 分配 Worker 启动的时候, 从一个 Zookeeper 集群获取 (保证所有的 Worker 不会有重复的机器号)
- 41 bits 的 Timestamp: 每次要生成一个新 ID 的时候, 都会获取一下当前的 Timestamp, 然后分两种情况生成 sequence number:
- 如果当前的 Timestamp 和前一个已生成 ID 的 Timestamp 相同 (在同一毫秒中), 就用前一个 ID 的 sequence number + 1 作为新的 sequence number (12 bits); 如果本毫秒内的所有 ID 用完, 等到下一毫秒继续 (这个等待过程中, 不能分配出新的 ID)
- 如果当前的 Timestamp 比前一个 ID 的 Timestamp 大, 随机生成一个初始 sequence number (12 bits) 作为本毫秒内的第一个 sequence number

整个过程中, 只是在 Worker 启动的时候会对外部有依赖 (需要从 Zookeeper 获取 Worker 号), 之后就可以独立工作了, 做到了去中心化.
异常情况讨论:

- 在获取当前 Timestamp 时, 如果获取到的时间戳比前一个已生成 ID 的 Timestamp 还要小怎么办? Snowflake 的做法是继续获取当前机器的时间, 直到获取到更大的 Timestamp 才能继续工作 (在这个等待过程中, 不能分配出新的 ID)

从这个异常情况可以看出, 如果 Snowflake 所运行的那些机器时钟有大的偏差时, 整个 Snowflake 系统不能正常工作 (偏差得越多, 分配新 ID 时等待的时间越久)
从 Snowflake 的官方文档 (https://github.com/twitter/snowflake/#system-

clock-dependency) 中也可以看到, 它明确要求 “You should use NTP to keep your system clock accurate”. 而且最好把 NTP 配置成不会向后调整的模式. 也就是说, NTP 纠正时间时, 不会向后回拨机器时钟.

三, Snowflake 的其他变种

Snowflake 有一些变种, 各个应用结合自己的实际场景对 Snowflake 做了一些改动. 这里主要介绍 3 种.

1. Boundary flake

http://boundary.com/blog/2012/01/12/flake-a-decentralized-k-ordered-unique-id-generator-in-erlang/

变化:

- ID 长度扩展到 128 bits:
- 最高 64 bits 时间戳;
- 然后是 48 bits 的 Worker 号 (和 Mac 地址一样长);
- 最后是 16 bits 的 Seq Number
- 由于它用 48 bits 作为 Worker ID, 和 Mac 地址的长度一样, 这样启动时不需要和 Zookeeper 通讯获取 Worker ID. 做到了完全的去中心化
- 基于 Erlang

它这样做的目的是用更多的 bits 实现更小的冲突概率, 这样就支持更多的 Worker 同时工作. 同时, 每毫秒能分配出更多的 ID

2. Simpleflake

http://engineering.custommade.com/simpleflake-distributed-id-generation-for-the-lazy/

Simpleflake 的思路是取消 Worker 号, 保留 41 bits 的 Timestamp, 同时把 sequence number 扩展到 22 bits;

Simpleflake 的特点:

- sequence number 完全靠随机产生 (这样也导致了生成的 ID 可能出现重复)
- 没有 Worker 号, 也就不需要和 Zookeeper 通讯, 实现了完全去中心化
- Timestamp 保持和 Snowflake 一致, 今后可以无缝升级到 Snowflake

Simpleflake 的问题就是 sequence number 完全随机生成, 会导致生成的 ID 重复的可能. 这个生成 ID 重复的概率随着每秒生成的 ID 数的增长而增长. 所以, Simpleflake 的限制就是每秒生成的 ID 不能太多 (最好小于 100次/秒,

如果大于 100次/秒的场景, Simpleflake 就不适用了, 建议切换回 Snowflake).

3. instagram 的做法

先简单介绍一下 instagram 的分布式存储方案:

- 先把每个 Table 划分为多个逻辑分片 (logic Shard), 逻辑分片的数量可以很大, 例如 2000 个逻辑分片
- 然后制定一个规则, 规定每个逻辑分片被存储到哪个数据库实例上面; 数据库实例不需要很多. 例如, 对有 2 个 PostgreSQL 实例的系统 (instagram 使用 PostgreSQL); 可以使用奇数逻辑分片存放到第一个数据库实例, 偶数逻辑分片存放到第二个数据库实例的规则
- 每个 Table 指定一个字段作为分片字段 (例如, 对用户表, 可以指定 uid 作为分片字段)
- 插入一个新的数据时, 先根据分片字段的值, 决定数据被分配到哪个逻辑分片 (logic Shard)
- 然后再根据 logic Shard 和 PostgreSQL 实例的对应关系, 确定这条数据应该被存放到哪台 PostgreSQL 实例上

instagram unique ID 的组成:

- 41 bits: Timestamp (毫秒)
- 13 bits: 每个 logic Shard 的代号 (最大支持 8 x 1024 个 logic Shards)
- 10 bits: sequence number; 每个 Shard 每毫秒最多可以生成 1024 个 ID

生成 unique ID 时, 41 bits 的 Timestamp 和 Snowflake 类似, 这里就不细说了.

主要介绍一下 13 bits 的 logic Shard 代号 和 10 bits 的 sequence number 怎么生成.

logic Shard 代号:

- 假设插入一条新的用户记录, 插入时, 根据 uid 来判断这条记录应该被插入到哪个 logic Shard 中.
- 假设当前要插入的记录会被插入到第 1341 号 logic Shard 中 (假设当前的这个 Table 一共有 2000 个 logic Shard)
- 新生成 ID 的 13 bits 段要填的就是 1341 这个数字

sequence number 利用 PostgreSQL 每个 Table 上的 auto-increment sequence 来生成:

- 如果当前表上已经有 5000 条记录, 那么这个表的下一个 auto-increment sequence 就是 5001 (直接调用 PL/PGSQL 提供的方法可以获取到)
- 然后把 这个 5001 对 1024 取模就得到了 10 bits 的 sequence number

instagram 这个方案的优势在于:

- 利用 logic Shard 号来替换 Snowflake 使用的 Worker 号, 就不需要到中心节点获取 Worker 号了. 做到了完全去中心化
- 另外一个附带的好处就是, 可以通过 ID 直接知道这条记录被存放在哪个 logic Shard 上

同时, 今后做数据迁移的时候, 也是按 logic Shard 为单位做数据迁移的, 所以这种做法也不会影响到今后的数据迁移