

system design, 请设计一个分布式系统，用来产生 GUID， without duplication

ChatGPT said:

设计一个分布式系统来生成全局唯一标识符（GUID）且无重复，是很多分布式系统的核心需求。下面是一个经典的设计方案和思路。

## 1. 需求分析

- 唯一性：生成的 GUID 全局唯一，无重复。
- 高可用：系统可以高并发、分布式部署且不影响生成效率。
- 高性能：生成速度快，低延迟。
- 可扩展性：节点可横向扩展。
- 容错性：节点宕机时不影响整体系统可用性。

## 2. 设计方案

方案核心：分布式 ID 生成器（类似 Twitter Snowflake）

Snowflake 算法思想：

- 64位二进制ID分为几个部分组成：
  - 时间戳部分（毫秒级）
  - 机器节点ID（机器或数据中心唯一标识）
  - 序列号（同一毫秒内自增，解决高并发冲突）

### 2.1 ID 结构设计（示例，64位）

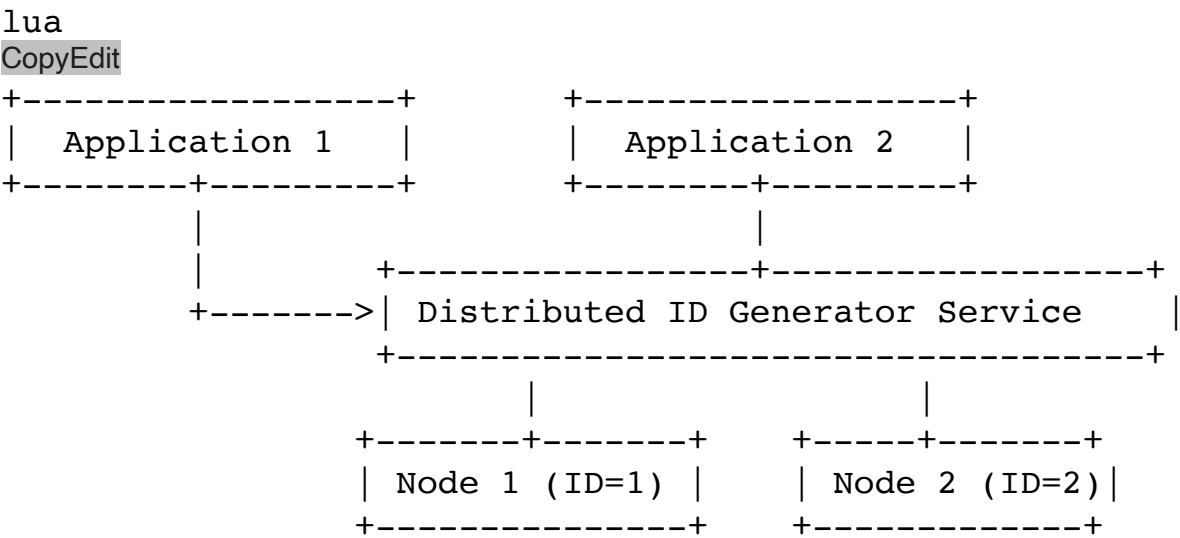
位数	内容	说明
----	----	----

41 bits	时间戳	从某个固定时间点开始的毫秒数
10 bits	机器ID	集群中机器唯一标识
12 bits	序列号	毫秒内自增序列号

2.2 组成部分详细解释

- 时间戳 (41 bits)  
支持约69年 ( $2^{41} / 1000 / 60 / 60 / 24 / 365 \approx 69$ 年)，保证时间单调递增。
- 机器ID (10 bits)  
支持最多1024台机器。
- 序列号 (12 bits)  
同一毫秒内最多产生4096个ID，防止并发冲突。

3. 系统架构设计



- ID 生成服务部署多个实例，每个实例分配一个唯一机器ID。
- 应用请求分布到多个生成节点。
- 每个节点内部维护序列号及时间戳，保证ID不重复。

## 4. 关键实现点

### 4.1 机器ID分配

- 可以由Zookeeper/Etcd或配置中心动态分配唯一机器ID。
- 避免两个实例使用相同机器ID，防止冲突。

### 4.2 时间回退问题

- 如果机器时间回退，需抛出异常或等待，避免生成重复ID。
- 或设计逻辑允许等待到时间回到上次时间后再继续。

### 4.3 序列号溢出

- 同一毫秒内序列号溢出时，阻塞等待下一毫秒。

## 5. Python 简单示例代码（Snowflake 核心逻辑）

```
python
CopyEdit
import time
import threading

class IdWorker:
    def __init__(self, worker_id, datacenter_id=0):
        self.worker_id_bits = 5
        self.datacenter_id_bits = 5
        self.sequence_bits = 12

        self.max_worker_id = -1 ^ (-1 <<
self.worker_id_bits)
        self.max_datacenter_id = -1 ^ (-1 <<
self.datacenter_id_bits)

        if worker_id > self.max_worker_id or worker_id < 0:
            raise ValueError("worker_id out of range")
```

```

        if datacenter_id > self.max_datacenter_id or
datacenter_id < 0:
            raise ValueError("datacenter_id out of range")

        self.worker_id = worker_id
        self.datacenter_id = datacenter_id
        self.sequence = 0
        self.sequence_mask = -1 ^ (-1 <<
self.sequence_bits)

        self.worker_id_shift = self.sequence_bits
        self.datacenter_id_shift = self.sequence_bits +
self.worker_id_bits
        self.timestamp_left_shift = (self.sequence_bits +
self.worker_id_bits + self.datacenter_id_bits)

        self.last_timestamp = -1
        self.lock = threading.Lock()

        # Twitter Epoch: 2010-11-04 01:42:54 UTC
        self.twepoch = 1288834974657

    def _time_gen(self):
        return int(time.time() * 1000)

    def _til_next_millis(self, last_timestamp):
        timestamp = self._time_gen()
        while timestamp <= last_timestamp:
            timestamp = self._time_gen()
        return timestamp

    def get_id(self):
        with self.lock:
            timestamp = self._time_gen()

            if timestamp < self.last_timestamp:
                raise Exception("Clock moved backwards.
Refusing to generate id")

            if self.last_timestamp == timestamp:

```

```

        self.sequence = (self.sequence + 1) &
self.sequence_mask
        if self.sequence == 0:
            timestamp =
self._til_next_millis(self.last_timestamp)
            else:
                self.sequence = 0

        self.last_timestamp = timestamp

        id = ((timestamp - self.twepoch) <<
self.timestamp_left_shift) | \
            (self.datacenter_id <<
self.datacenter_id_shift) | \
            (self.worker_id << self.worker_id_shift) |
\
            self.sequence
        return id

# 示例使用
if __name__ == "__main__":
    worker = IdWorker(worker_id=1, datacenter_id=1)
    for _ in range(10):
        print(worker.get_id())

```

## 6. 总结

- 雪花算法 (Snowflake) 是分布式 GUID 生成的经典方案。
- 关键是时间戳 + 机器ID + 序列号的组合。
- 需要保证机器ID唯一，时间单调递增。
- 结合服务注册和配置中心保证节点ID唯一分配。
- 生产环境可以用 Redis、Zookeeper 等辅助做节点ID分配和健康检测。
- 带服务注册的节点ID分配方案

- Redis/MQ 作为分布式锁/节点协调
- 支持多机房跨地域生成方案