

互联网分布式系统架构分享

54chen (陈臻) @ Qcon

czhttp@gmail.com

<http://www.54chen.com>

2011-04-10

互联网在中国

- **1987年第一封电子邮件：穿越长城，走向世界**
- 分布式系统：近几年发展活跃



常见的搭配

- mySQL
- Memcached

- 大量数据放在内存中
- 极致优化mySQL
- 读写数据有套路

Mysql-memcached搭配-读

- 先从memcached读取
- 如果有值，返回
- 如果无值，从mysql读取，返回并写memcache

Mysql-memcached搭配-写

- 写入mysql，产生对应的key后删除memcache的值

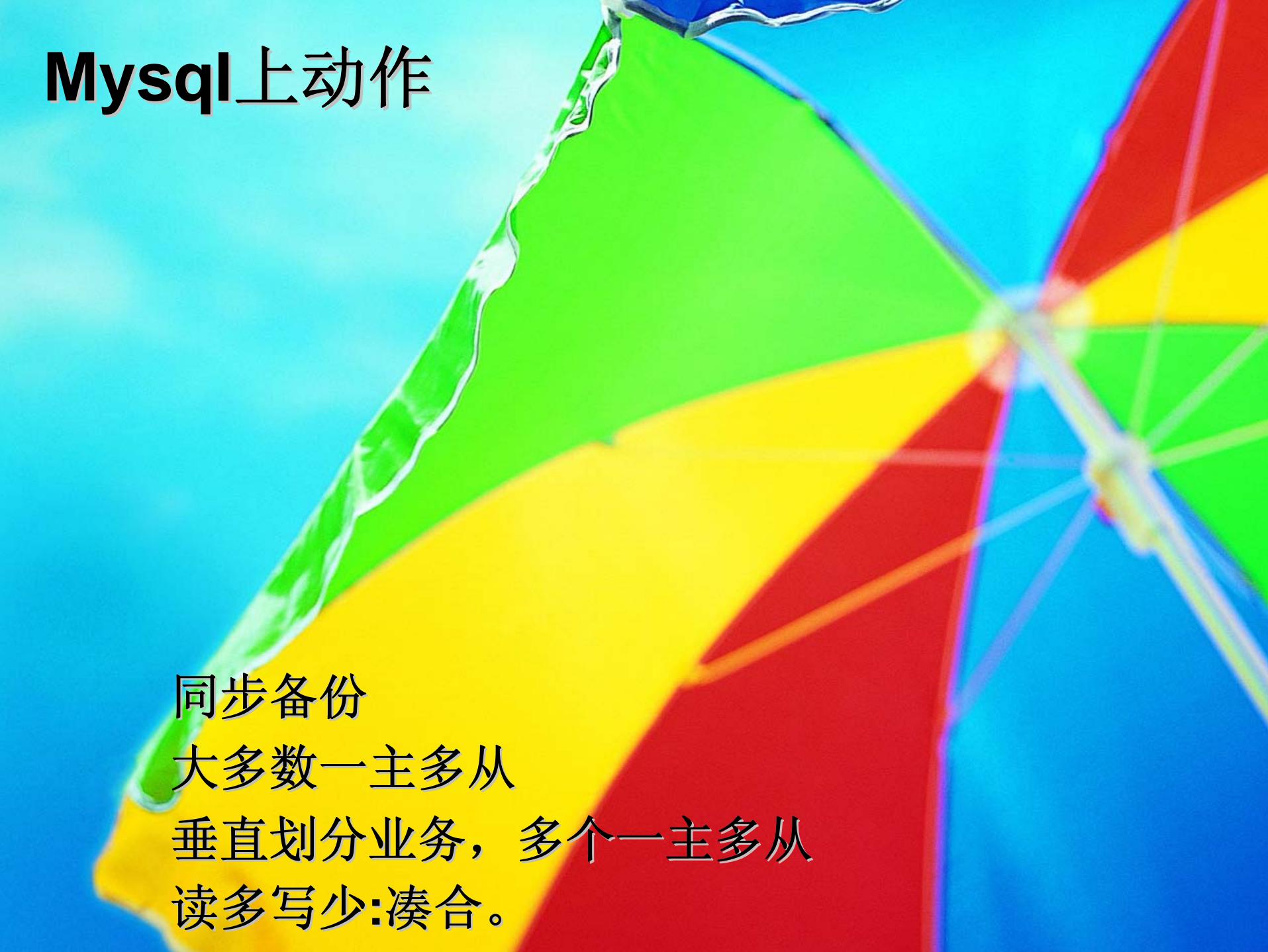
而节点之间的关系又如何呢？

Memcached节点间

读取加速
一致性hash

失效影响只到部分数据
读多写少:凑合

Mysql上动作



同步备份
大多数一主多从
垂直划分业务， 多个一主多从
读多写少：凑合。

即便如此

使用mysql

还需要

在代码中

小心翼翼

请看示例



小心翼翼的使用举例-忘记where

- 举例: **delete from table_name**
- 应该: **delete from table_name where id = ?**

习惯性加上**limit**

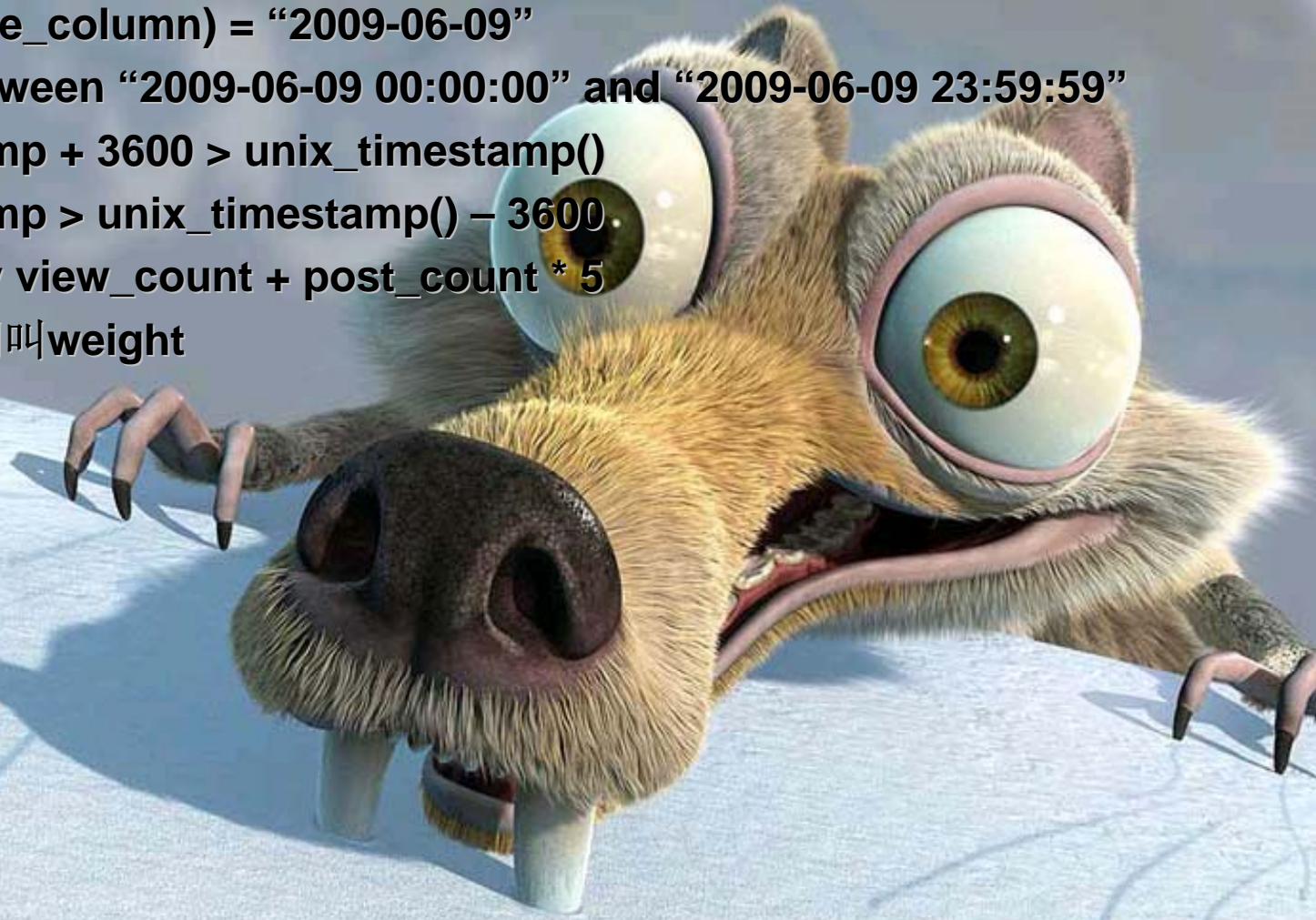
操作之前:

连对机器了么? 写条件了么? 要备份么?



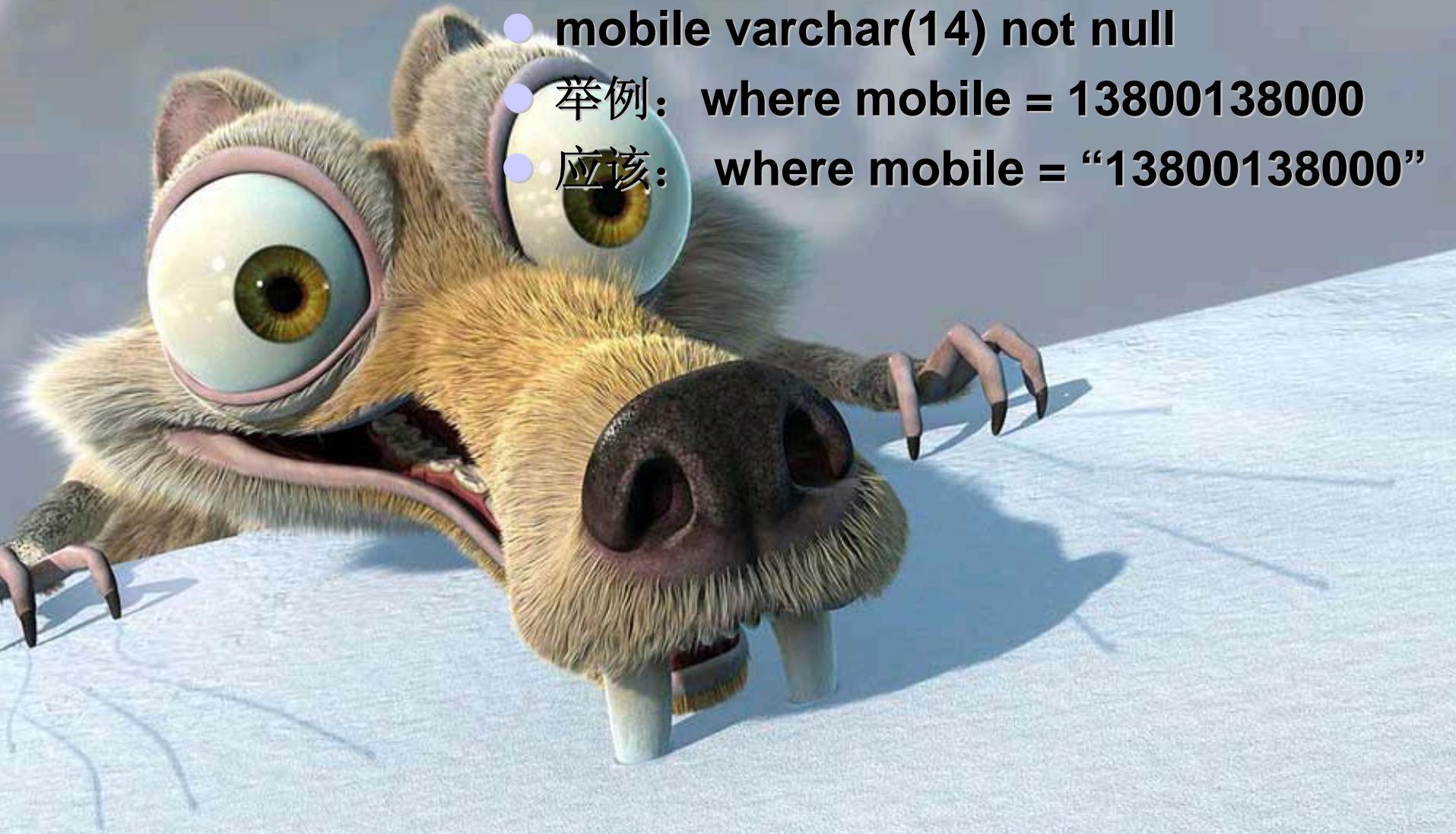
小心翼翼的使用举例-操作列

- 举例: `date(time_column) = "2009-06-09"`
- 应该: `date between "2009-06-09 00:00:00" and "2009-06-09 23:59:59"`
- 举例: `timestamp + 3600 > unix_timestamp()`
- 应该: `timestamp > unix_timestamp() - 3600`
- 举例: `order by view_count + post_count * 5`
- 应该: 单搞一列叫`weight`



小心翼翼的使用举例-忘记引号

- **mobile varchar(14) not null**
- 举例: **where mobile = 13800138000**
- 应该: **where mobile = "13800138000"**



小心翼翼的使用举例-limit的问题

- 举例: `select * from table limit limit 60000000, 100`
- 应该: `select * from table where id>60000000 limit 100`





小心翼翼的使用举例-rand的问题

- 举例: `select * from online_user order by rand() limit 100`
- 应该:
 - `select * from online_user where page = $rand_page`
 - `select * from online_user where id > $rand_pos limit 100`
 - `select * from online_user + memcached ...`



任何一个不小心

- Mysql锁死
- 速度慢
- 负载高

让coding更加轻松

- 不担心
- 够放心
- 睡安心



Dynamo

- 去年很热的no-sql
- Key-value
- 非常不错的各种实现



进入Dynamo的世界

- 一致性哈希
- CAP原则
- Merkle Tree
- Gossip协议
- hinted handoff数据
- 向量时钟

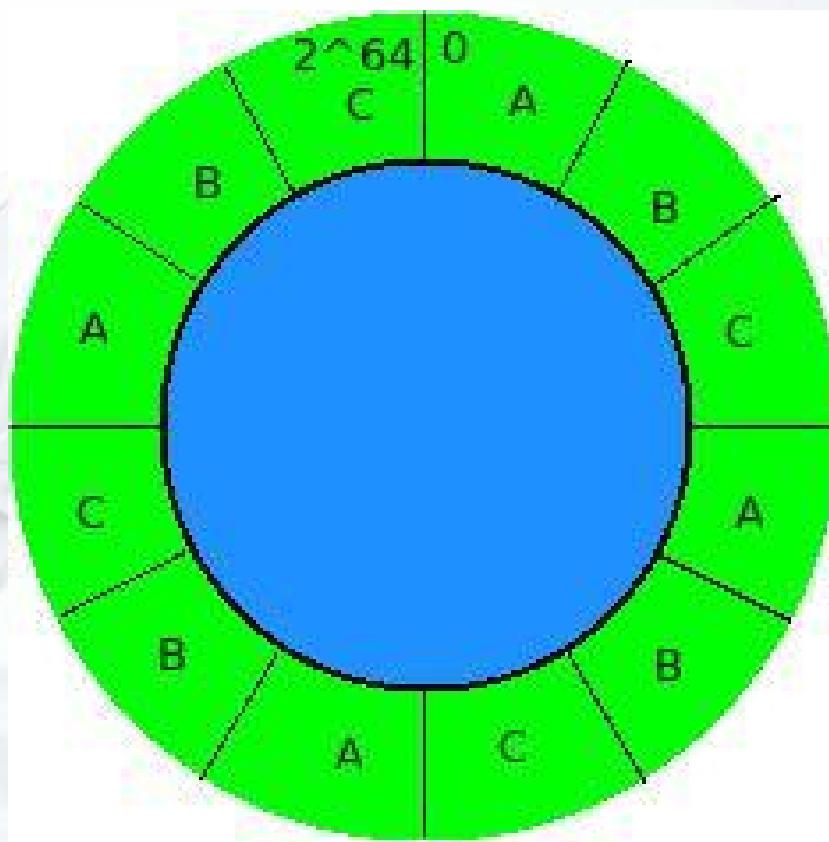


进入Dynamo的世界

目标：99.9%的请求延迟要在**300毫秒**以内
数据同步与负载均衡

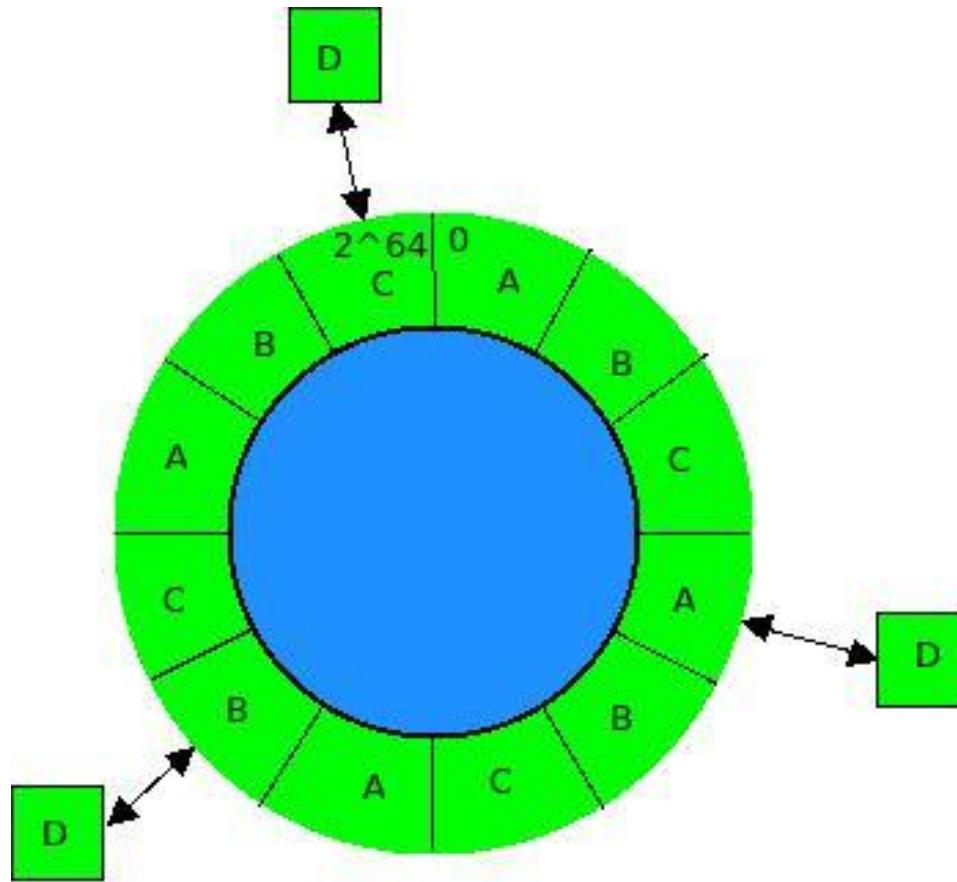
一致性哈希

- 保障数据在有节点变动时影响范围最小
- **Dynamo**设计时，使用了虚拟节点的概念



虚拟节点

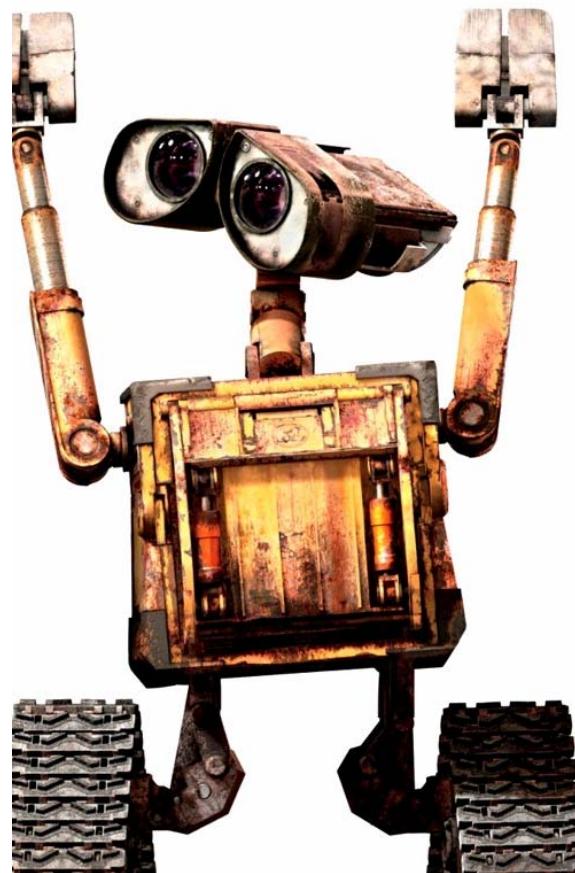
- 虚拟节点让负载均衡
- 即便有节点变动也负载均衡



CAP原则

- Consistency (一致性)
- Availability (可用性)
- Partition tolerance (分区容错性)
- 定理：任何分布式系统只可同时满足二点，没法三者兼顾。
- 忠告：架构师不要将精力浪费在如何设计能满足三者的完美分布式系统，而是应该进行取舍。

● NRW: Dynamo的CAP



NRW

- **N**: 复制的次数;
- **R**: 读数据的最小节点数;
- **W**: 写成功的最小分区数。



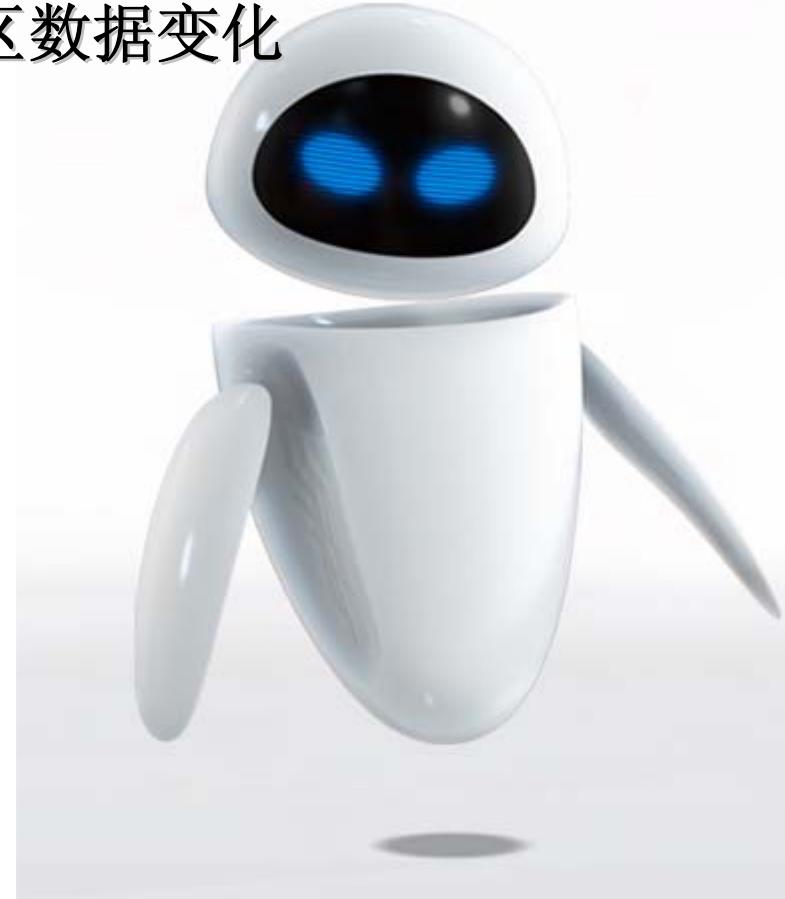
NRW:这三个数的具体作用

- 用来灵活地调整**Dynamo**系统的可用性与一致性。
- 例：
- 如果**R=1**, 表示最少只需要去一个节点读数据即可，读到即返回，这时是可用性是很高的，但并不能保证数据的一致性，
- 如果说**W**同时为1, 那可用性更新是最高的种情况，但这时完全不能保障数据的一致性，因为在可供复制的**N**个节点里，只需要写成功一次就返回了，也就意味着，有可能在读的这一次并没有真正读到需要的数据（一致性相当的不好）。
- 如果**W=R=N=3**, 每次写的时候，都保证所有要复制的点都写成功，读的时候也是都读到，这样子读出来的数据一定是正确的，但是其性能大打折扣，也就是说，数据的一致性非常的高，但系统的可用性却非常低了。
- 如果**R + W > N**能够保证我们“读我们所写”，**Dynamo**推荐使用322的组合。



Merkle Tree

- 将每一个数据对应的**key-value**建立成一个**hash tree**
- 微小的变化也会引发顶层的不一致
- 用来通知各同步节点之间的分区数据变化

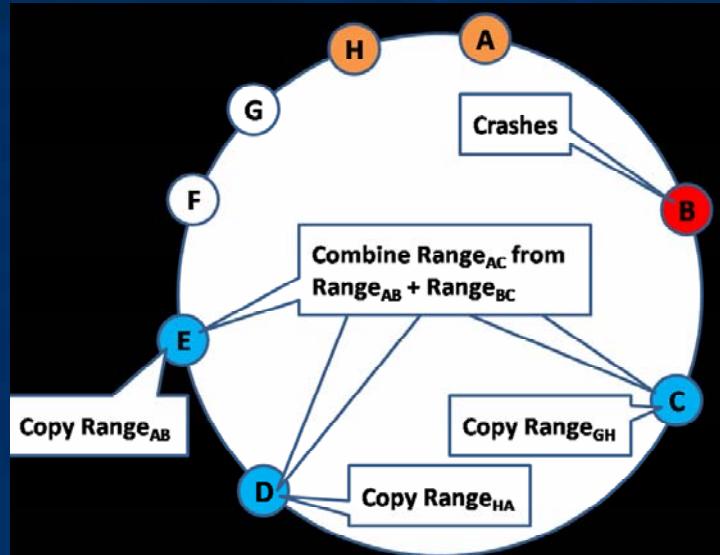


Gossip协议

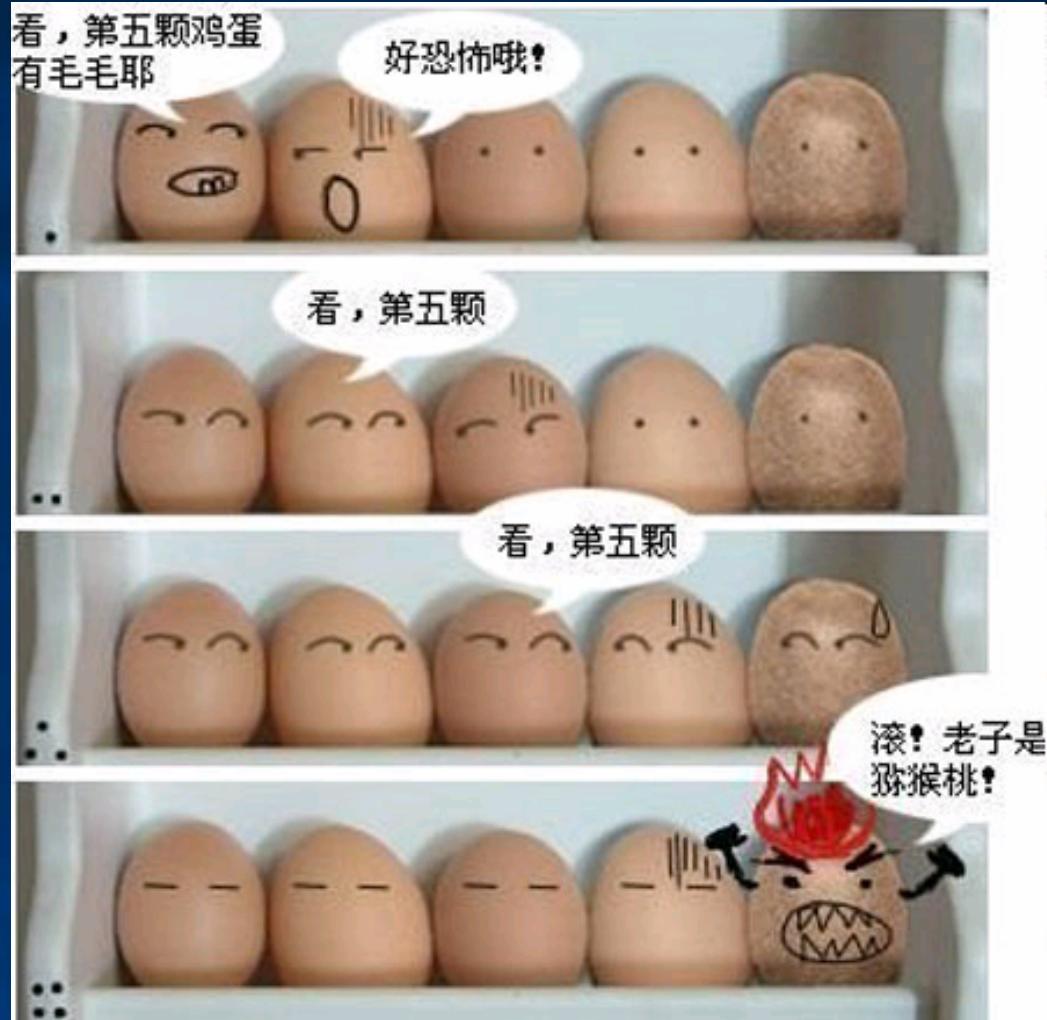
- 当集群里的一个节点出现故障
- 或者增加一个节点
- 周知各节点的协议



什么是Gossip协议？



什么是Gossip协议？



hinted handoff数据

- 当集群里的一个节点出现故障
- 数据自动进入临近的节点**handoff区**
- 收到恢复通知后自动恢复**handoff**数据



向量时钟 vector clock

- 在写入数据时，各个节点对数据本身都一个向量记录
- 在读取多个节点时，进行向量计算从而得出最正确结果

[a,1]

a节点 时钟计数1

[b,2]

b节点 时钟计数2



向量时钟 vector clock

- 在写入数据时，各个节点对数据本身都一个向量记录
- 在读取多个节点时，进行向量计算从而得出最正确结果

[a,1]

a节点 时钟计数1

[b,2]

b节点 时钟计数2



A wide-angle, low-angle shot of a city skyline under a bright blue sky with scattered white clouds. In the foreground, the tops of several skyscrapers are visible, their facades featuring various architectural patterns like horizontal lines and small squares. The sun is positioned in the upper left corner, creating a bright lens flare.

这些，构成Dynamo

然后是cassandra

- Facebook出品 2008年开源
- Cassandra最初作者Avinash Lakshman
(Amazon's Dynamo的作者之一)



Cassandra

Cassandra偷懒的地方



- 大分区
- Cassandra未用**vector clock**, 而只用**client timestamps** 简化了冲突选择

Cassandra牛X的地方

实现了**bigTable**

数据模型

基于列族

(**Column Family**)



然后是**voldemort**

- LinkedIn出品
- 重点在使用**Hadoop**存放收集整理数据
- 定死了节点数量



国内一些dynamo产品

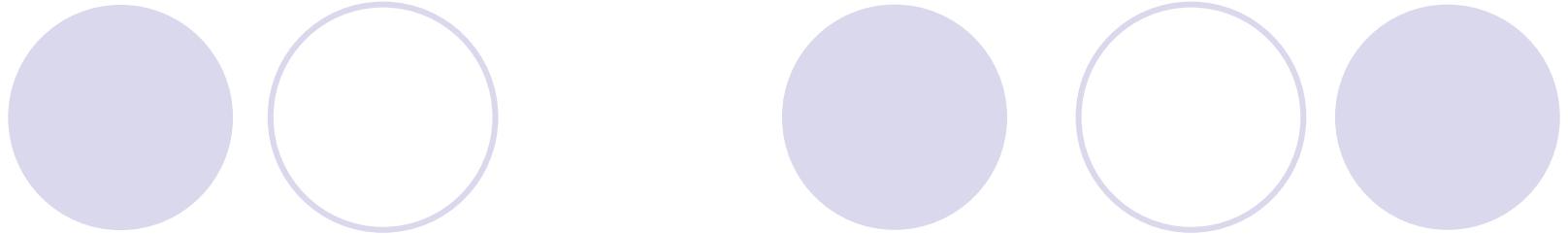
- 人人网 nuclear
- 豆瓣 beansdb
- 淘宝 Tair
- 新浪 sina-sdd
- ...



豆瓣douban!

淘宝网





感谢始祖 amazon.com

Thanks

54chen(陈臻)

czhttp@gmail.com

<http://www.54chen.com>