LECTURE 16 MEMCACHED

1. 前置知识

- over-subscription 过度订阅，指分配比可用的资源更多的计算任务，通常和parallelism一起来实现web server高吞吐量
- Facebook员工解读：传送门

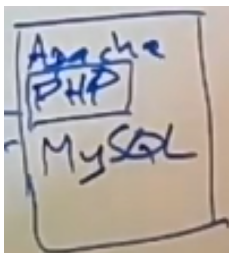
2. 背景

2.1. Facebook流量特征

- 读负载比写负载高一个数量级
- 读操作来自Mysql，HDFS和其他backend service

2.2. 服务端架构演变

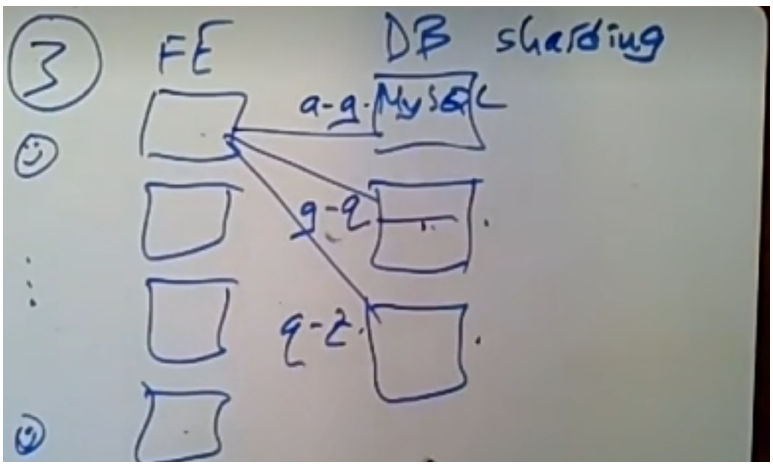
1. 阶段1
    - Web Server和DB不分离
    - 单Web Serber和单DB
    - 如图：



2. 阶段2
    - Web Server和DB分离
    - 多Web Serber和单DB
    - 如图：

3. 阶段3

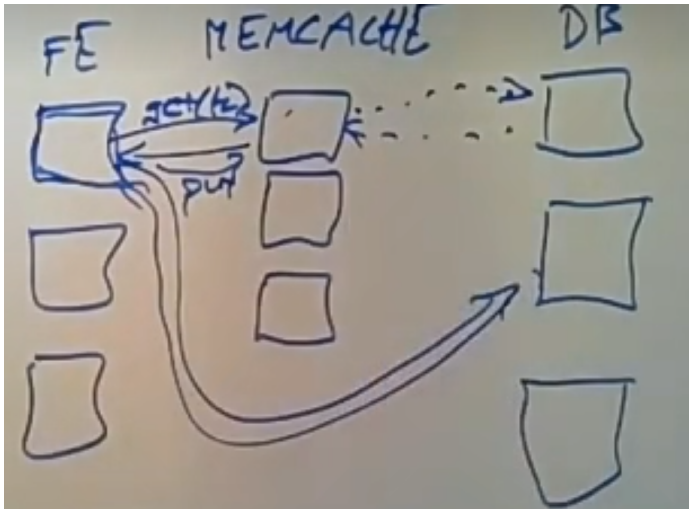
- Web Server和DB分离
- 多Web Server对多DB Server
- DB并非互为replica， 而是使用key进行分片，因此需要修改Web Server的程序能让其找到对应的DB Server（例如一致性哈希算法）
- 并非一味地进行分片就能解决问题，因为有些key作为热点，其所在的DB Server会因为该key承载很大的工作负载
- 如图：



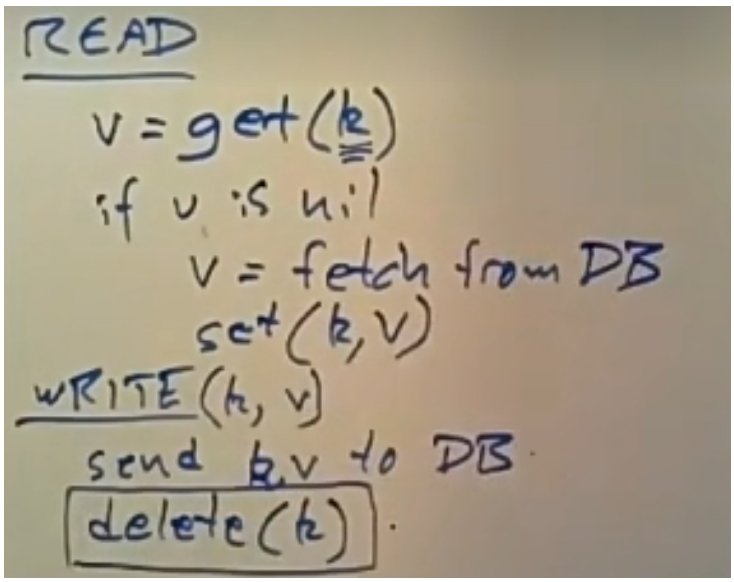
4. 阶段4

- 在**前端**（front end， FE，在memcache中，前端默认指Web Server + Memcache，而在此处的背景中，将Memcache层作为中间层而不是属于前端）和DB之间增加了cache层
- FE和Memcache之间支持的操作：
  - get(k)：获取key为k的value。cache hits时直接返回value；cache miss时，需要FE进行进一步操作
  - set(k, v)：将key为k的value设置为v，用于cache miss时设置key-value tuple

        - delete(k)：删除key为k的key-value tuple
    - 架构图：



    - Web Server programmer视角的变成可能如下：



      注意此处对DB的写实际应该为SQL语句

3. Memcache架构

- 见阶段四关于架构的基本介绍，Memcache由若干个region组成，这些region互为replica。region由如下内容组成：
    - 多个FE clusters，一个cluster由如下内容组成：
        - 多个FE servers

        - 多个Memcache servers，这些Memcache server被划分成为了若干的pools，用于处理不同的负载情况
        - 一个mcrouter
    - 单一storage cluster
    - regional pool
- memcache层的作用：
    - 减轻DB的负担，增加工作负载，进而增大吞吐量。通常DB为了设计的完整性，其性能并不好，而memcache更接近底层硬件，对FE和DB的逻辑和内容完全不关心，因此读取数据更快。如果没有Memcache只用数据库，系统很快会崩溃（MySQL承载不了如此巨大的流量）
    - 在有限的资源下，开发时间短
- 为何不由Memcache层处理cache缺失，如阶段4图示中的虚线部分？（例如硬件cache的look-through方式）

  为了将Memcache与DB解耦，Memcache的下层未必是DB，若由Memcache来处理cache miss，则Memcache必定有与DB交互相关的代码。这也是**look-aside cache**与**look-through cache**的区别。（硬件cache是look-through，Memcache是look-aside）
- 风险： 当某台memcache Server故障时，本该由memcache server承载的负荷转移到了DB上，可能会导致DB server负荷过重也发生故障。（原因在于通常cache命中率很高，但当使用gutter + lease的配合时，可以减轻Memcache server故障带来的危害）

4. Cluster的设计

Region之间互为replica，同一region的FE cluster之间也互为replica。**而在同一cluster内，数据以Consistency Hash 算法对数据对象进行分区！！！！！！！！**，因此Memcache将partition与replication很好地融合

4.1. 降低延迟

- 一个Web server的请求可能和很多个item相关，因此同一时间一个请求可能要访问多个Memcache server（一个cluster内）。（推测：同一个cluster内通过一致性哈希进行分区，存在热点问题。但是region之间和cluster之间有replica，可以将这一个热点key的访问分布在多个cluster甚至多个region之中）
- 数据相关的优化：根据数据之间的关系建立DAG，而后进行批处理

- Memcache之间不会相互通信
- **mcrouter**：是一个中间件/代理（proxy），充当web servers和Memcache之间的代理，用于减少网络通信次数
- UDP/TCP：
  - get操作使用UDP协议（不可靠），注意若出现了乱序/丢包等情况直接向client server报错，由client sever的逻辑进行处理，Memcache针对UDP相关错误没有自动恢复机制，client server会将UDP相关错误认为是cache miss，而后向DB索要数据，但不会回写
  - set/delete使用TCP协议，对于set和delete操作，会经过mcrouter进行转发。若不通过mcrouter 进行转发，而直接使用点对点通信：则通信数目过于庞大，mcrouter可以统一打包、转发，因而减少网络通信数量
  - 滑动窗口：用于避免incast congestion（可能会导致交换机负载过大进而崩溃），只有当一个request收到响应之后，后续request才能进行。当request成功响应时，滑动窗口大小会缓慢增长，若未成功响应则会缩小。
    - 滑动窗口的大小会影响请求响应时间：

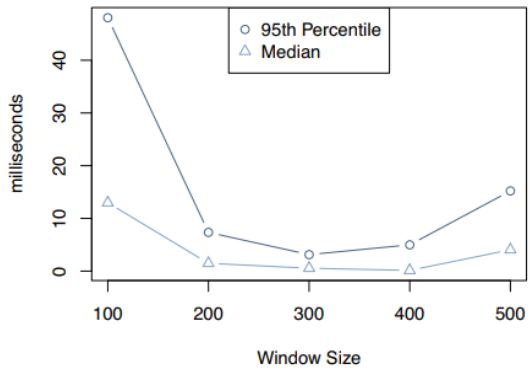


Figure 4: Average time web requests spend waiting to be scheduled

与Little's Law相关

4.2. 降低负载

4.2.1. 两个tricky的问题

- stale sets：

```
C1 get(k),  memcache miss
C2 UPDATE DB(k,v2)
C2 delete(k)
C1 set(k,v1)
```

此时，作为stale数据的v1会长期存在在Memcache中，而v2得不到更新

- **thundering herds，惊群效应：** 一个热点key，遇到invalidate后，很多的get操作必须要向DB索取，此时DB可能会被击穿

4.3. lease机制

- 当get操作发现cache miss时，Memcache给当前进行get操作的FE server一个独一无二的lease。这个lease存在有效期，超过有效期会自动过期（若FE server崩溃，则lease 会自动过期）；若lease存在时，别的FE server发现了get miss，则Memcache会通知FE server过段时间重试，而不是去访问DB，这解决了thundering herds 的问题
- 当前进行get操作的FE server去访问DB，而后进行set操作，当且仅当 FE server的lease号码与Memcache登记的数据一致时，set操作才能成功。
- 当有delete操作到达时，lease会自动失效。则在stale set 的情况中，由于C2 invalidate key为k的数据，则C1进行set时会失败。

4.4. 故障处理

故障有两类：

- 大范围的故障，例如大规模停电。针对大规模故障，只需要将某个slave Region切换到master Region即可
- 小范围故障，例如网络、硬件故障等。存在一些空闲的服务器，称为**gutter**，当故障时负责接力。当FE server没有收到Memcache响应时，会认为该Memcache server故障，直接与gutter通信。（与另一种策略对比，若一个server故障，则用其他server来接替该server的故障（例如通过一致性哈希算法），若故障server存在hot key，则接力server很容易也过载，进而发生故障，这称为**cascading failures**）

5. Region设计

5.1. Invalidation

- 当一条数据被更新时，需要发送delete 给memcache，修改数据的FE server会给自己cluster范围内的memcache server发送delete消息。

- 修改数据的FE server所在的cluster之外的memcache server，由MySQL负责invalidate。在SQL语句中，植入了memcache key，当涉及更新语句时，memcache会向各个cluster的mcrouter发送delete操作，由mcrouter向各个memcache server发送消息，则可以降低网路负载；MySQL内嵌入了一个mcsqueal守护进程，用于监测事务执行过程中对数据对象的修改，mcsqueal存在于每个DB server之中。

5.2. Regional Pools

由于同一个region的cluster之间互为replica，如果一些不经常访问但是缺失代价非常昂贵的数据也被多次复制，则空间会被浪费。因此同一个region之中有regional pool，由一些memcache server组成，用于存储这类数据

5.3. 冷启动问题

- 当启动一个新的cluster时，所有的memcache server都没有数据，则缺失率为100%，此时过大的负载可能会击穿DB，因此冷启动的cluster发生缺失时，从warm cluster中读取数据，而非DB

- 一致性问题：Cluster A某个Server往Storage里更新了一份数据，后端在完成数据更新后会把数据过期的消息发送到其他的Cluster。同时Cluster A里某个Server读这份数据的时候发现cache miss，然后从Cluster B里读；如果恰好Cluster B此时还没有收到数据过期的消息（因为消息传递也是要时间的），这个读会成功返回，然后这个Server拿到事实上已经过期的数据后会往Cluster A的memcached里写。这样子就相当于Cluster A的memcached就存放了一份过期的数据，而且这份过期的数据可能被保留很长甚至无限长的时间。

解决办法是：memcached支持一个功能，在对一个key进行delete操作之后锁住这个key一段时间不允许进行add操作。通过在cold cluster里设置这段hold-off时间为大于0的值（2秒），在上面的场景中，由于前面那个更新的Server是对本地memcached进行了delete操作的，第二个server拿着过期数据往里写的时候就会报错，然后它就知道这个数据在Storage里面有新值，就会去读最新的值再往里写。理论上过期数据还是可能出现的，但是可能性大大减低了。

6. 一致性问题

- Master Region才允许读写，Slave Region只允许读

- 各个Region之间DB的复制通过MySQL的机制自行实现，master region会将数据修改流传递给其他region，也包括delete！因此Master Region的写操作与单Region的操作完全一致，不需要其他操作。

- 对于Slave Region的写：由于slave region只读，因此必须把数据修改信息传递给master region，而后再将该数据的修改重新传递到当前slave region。设想这样的情况：某server先修改了数据x，然后另一台server读取x发现缺失，因此进行set操作，此时delete还未传递到

该slave region，因此lease也未失效，成功写入数据，该stale数据又会存活很久，解决办法如下：

1. 对于该修改了的数据（但是修改还没到达slave），在该region中设置r_k（注意与k相关哦）
2. 写master的DB，并且将k和r_k一起放在SQL语句中（注意单Region为了让DB来发送delete本身就需要在SQL语句中嵌入k）
3. 删除在**local cluster**中key为k的value，而后所有的get都会miss，get操作会被重定向到master region的DB中读取
4. 当delete到达之后，则清除k与r_k（所有cluster！）

7. 相关论文

- 集群拥塞 incast congestion 30
- Little's Law 26