随着数据量的增大，你可能会增加前端服务器和后端数据库服务器的数量。但不断增加数据库服务器会有别的问题：

1. 有些热 key，无论数据粒度切分的有多细，保存热 key 的服务器还是会过载
2. 成本很高

因此需要通过缓存，缓存速度更快，缓存位于前端服务器和数据库之间

look-asida cache：前端查看缓存是否有数据，若无，则前端查看数据库

look-through cache：通过 memcache 将请求转发给数据库，让数据库来响应

一般来说缓存可以承担 99% 的读操作，但是如果缓存服务器故障了，可能会导致数据库一下承担大量的读操作，我们需要对此进行处理

FaceBook 有两个数据中心，一个在东海岸，一个在西海岸。他们存储的数据完全相同。用户请求会根据距离发到较近的数据中心中。

更新操作：先把更新请求发往数据库服务器，再把 memcache 中的数据删除

> 问题：为什么要选择把数据删除，而不是发送一个 set RPC 将数据发往 memcache 呢?

假设两个 client 对 x 进行加 1 操作，他们都往数据库发送了加 1 的操作，然后发送 set RPC 时，由于网络的问题，第二个执行的 set RPC 比第一个先到达，然后第一个到达的 set RPC 就会把第二个 set RPC 的结果给覆盖了，这样缓存中存储的就是不一致的情况。因此选择用 delete 策略

下面来看看性能问题，在存储上主要就是分区和复制，分区是将数据进行 hash 划分，这样不同数据的读请求能够分散到不同的服务器上，复制就是对相同数据存储多个副本。分区的好处就是能够一定程度缓解服务器的压力，但是如果前端服务器所需的数据来自很多不同的服务器，此时前端服务器就需要和大量的服务器进行通信。复制的好处就是对于一些热 key，其能够很好的分摊读请求，但是这会占用更多的内存，并且需要保证数据的一致性。

现实需要根据具体情况结合复制和分区这两种策略。

东西海岸的数据中心存储相同数据的原因是：如果存储不同的数据，那么位于东海岸的用户想访问的副本在西海岸，那么这段传输时间就会比较久，可能会给用户带来不好的体验，并且，前端服务器要为用户创建一个页面时（可能需要数百个 item），如果可以直接从同一个数据中心的 memcache 中直接获取数据，速度也会快很多。

但这样写操作的成本也会相应增加，执行写操作时，就需要通过网络将数据发送给另一个数据中心。但执行读操作的频率要远大于写操作。并且如果一个数据中心挂掉了，那么另一个数据中心就可以直接接手工作。

FaceBook 将数据中心划分成多个集群，每个集群包含一堆前端服务器和 memcache 服务器，集群间几乎是完全独立的，即集群1前端服务器的所有读请求都会发送给集群内的 memcache 服务器，如果没有命中缓存就需要读取数据库服务器。

> 问题：为什么使用多集群，而不是一个数据中心一个集群呢？

如果一个数据中心一个集群，那么这样对热 key 是没有什么性能提升的，对于不同的集群，热 key 都会有相应的复制，这样可以一定程度的提升性能。另外一个原因就是，如果使用一个集群，前端服务器可能会从所有 memcache 服务器上获取它们所需的数据，这样需要建立的通信连接数量就很多，它们使用 TCP 来进行通信。而且还要维护连接的状态，如果业务繁忙的时候，某个前端服务器可能会收到来自很多不同 memcache 服务器的数据包，此时可能会出现丢包的情况。限制集群大小可以有效缓解这些问题。还有一个原因就是，如果集群很大，要在集群中构建出不同机器都可以通信的网络是不容易的，通过将数据中心拆分成不同的集群，并且让大部分通信都在集群内部，此时我们就只需要在每个集群中有一个规模适中的网络即可。

在不同集群中，它们会对数据进行复制，如果不是经常访问的数据，就不会被复制，region 中还会有一个 regional pool，它会被数据中心中的所有集群共享。前端服务器知道某些 key 使用频率不是很高，就不会将其放入集群中 memcache 服务器里，而是放在 regional pool 的 memcache 服务器上

如果我要在数据中心中加入新集群，用于分摊负载，由于一开始缓存没有数据，那么就会导致读请求全部涌入数据库服务器中，可能会导致数据库崩溃。因此，这里使用一种 cold start 的方法来进行处理。用 cold start 方式时，如果新集群中的服务器在缓存没有找到数据时，其会从别的集群中查找数据，若找到，就会携带相应的数据写入 memcache 服务器上，若在别的集群中也没有找到对应的数据，那么前端服务器才会访问数据库。这个过程可能会持续一到两个小时，直到 memcache 缓存所有的热门数据，之后就关闭 cold start，直接使用本地的 memcache 服务器。

惊群效应

指的是某个 key 非常热门，假设此时某个前端服务器需要修改这个 key，然后使得 memcache 中的对应数据无效，此时大量对该 key 的访问都会导致缓存未命中，使得大量前端服务器都去访问数据库，请求到相应数据后又把相应的数据写入 memcache 中，这样会大量增加负载。

FaceBook 的解决方法为：当一个前端服务器从 memcache 中发现缓存未命中时，其会从数据库获取相应数据，并将其写入 memcache 服务器中，而此时其他前端服务器，如果想访问该 key，它们就得等待该数据被缓存到 memcache 中，再去访问它。

这主要是通过 lease 来实现的。当第一个前端服务器访问该 key，并且该 key 不在 memcache 中时，memcache 会给该前端服务器一个租约（原子的），memcache 会维护一个 key 和 lease 的表，然后前端服务器从数据库获取到相应消息时，其会把相应的数据和 lease 一起发给 memcache，该 lease 表明该前端服务器具有写入的权限，然后 memcache 将该数据写入，并删除该 lease。若其他前端服务器访问 memcache 没有得到相应数据，并且该 key 对应有 lease 时，memcache 会让前端服务器等一会再继续发送请求，如果持有 lease 的前端服务器在一定时间内还没有将数据写入，memcache 会将该 lease 作废，然后用同样的方式进行处理。

如果某个 memcache 服务器故障了，如果我们不做任何处理，那么所有请求都会发往数据库服务器，此时会造成数据库过载。FaceBook 是通过使用 Gutter server 来解决这些问题，如果前端服务器访问 memcache 服务器收到错误，并被告知其无法与 memcache 服务器通信，那么前端服务器就会把请求发往其中一个 Gutter server，这里应该也是通过 hash 来寻找对应的 Gutter server，那么一开始 Gutter server 也是没有任何数据，此时就用解决惊群效应的方法来处理，接着发生故障的 memcache 就被新的服务器替换。

> 问题：闲置的 Gutter server 是否会过于浪费？

回答：因为正常的 memcache 服务器几乎都是满负载的，如果需要新的 memcache 服务器来接替工作的话，新服务器或多或少都需要有一些空闲的内存，以便处理 memcache 满负载的情况

一致性

由于网络问题可能会导致缓存一致性的问题，比如：

```
Q: What is the "stale set" problem in 3.2.1, and how do leases solve it?

A: Here's an example of the "stale set" problem:

1. Client C1 asks memcache for k; memcache says k doesn't exist.
2. C1 asks MySQL for k, MySQL replies with value 1.
   C1 is slow at this point for some reason...
3. Someone updates k's value in MySQL to 2.
4. MySQL/mcsqueal/mcrouter send an invalidate for k to memcache,
   though memcache is not caching k, so there's nothing to invalidate.
5. C2 asks memcache for k; memcache says k doesn't exist.
6. C2 asks MySQL for k, mySQL replies with value 2.
7. C2 installs k=2 in memcache.
8. C1 installs k=1 in memcache.

Now memcache has a stale version of k, and it may never be updated.

The paper's leases would fix the example in the following way:

1. Client C1 asks memcache for k; memcache says k doesn't exist,
   and returns lease L1 to C1.
2. C1 asks MySQL for k, MySQL replies with value 1.
   C1 is slow at this point for some reason...
3. Someone updates k's value in MySQL to 2.
4. MySQL/mcsqueal/mcrouter send an invalidate for k to memcache,
   though memcache is not caching k, so there's nothing to invalidate.
   But memcache does invalidate C1's lease L1 (deletes L1 from its set
   of valid leases).
5. C2 asks memcache for k; memcache says k doesn't exist,
   and returns lease L2 to C2 (since there was no current lease for k).
6. C2 asks MySQL for k, mySQL replies with value 2.
7. C2 installs k=2 in memcache, supplying valid lease L2.
8. C1 installs k=1 in memcache, supplying invalid lease L1,
   so memcache ignores C1.

Now memcache is left caching the correct k=2.
```

注意：delete 操作不会导致过时数据的出现，再访问数据库就好了

这里对 lease 有了新的机制，即当 memcache 服务器从另一个 client 或者数据库收到 delete 操作时，除了删除这个数据，其还会使得该 key 之前的 lease 无效，当过期的 set 命令到达时，memcache 会检查对应的 lease，若发现 lease 过期，则 memcache 会忽略这个 set 请求。下一个读取该 key 的前端服务器会发现缓存未命中，此时会继续颁发 lease 来处理。这就是 FaceBook 保证一致性的方法

> Yahoo 的 PNUTS 系统也有很多不同的细节