2.5 WEBSERVER、负载均衡、服务器架构

NGINX 与负载均衡

- 反向代理
    1. 将用户请求转发给内部服务器，保护内网拓扑结构

```
                  / static file               /cache
hit->Redis/NoSQL
                /                            /
               /       /Gunicorn + Django-1->cache
miss->MySQL/PgSQL
client->NginX->HaProxy-Gunicorn + Django-2
                       \Gunicorn + Django-3
```

    2. 可以解析用户请求，代理静态文件
- NginX负载均衡
    - 轮询:

```
upstream backserver {
  server 192.168.0.14;
  server 192.168.0.15;
 }
```

    - 权重: weight

```
upstream backserver {
  server 192.168.0.14 weight=10;
  server 192.168.0.15 weight=10;
}
```

    - IP哈希: ip_hash

```
upstream backserver {
  ip_hash;
  server 192.168.0.14:88;
  server 192.168.0.15:80;
}
```

    - fair

```
upstream backserver {
  server server1;
  server server2;
  fair;
}
```

    - url_hash

```
upstream backserver {
  server squid1:3128;
  server squid2:3128;
  hash $request_uri;
  hash_method crc32;
}
```

    - 最小连接数: least_conn

- 其他负载均衡
    - F5: 硬件负载均衡设备, 性能最好, 价格昂贵(30-50一个，买一对)
    - LVS: 工作在 2层 到 4层 的专业负载均衡软件, 只有 3 种负载均衡方式, 配置简单，中国人（章文嵩）开发
        - 七层网络模型：ISO标准
        - 四层网络模型：TCP工业实现
- HAProxy: 工作在 4层 到 7层 的专业负载均衡软件, 支持的负载均衡算法丰富
    - 性能比较: F5 > LVS > **HAProxy > Nginx**
- LVS 的优势
    - 常规负载均衡

      进出都要经过负载均衡服务器. 响应报文较大, 面对大量请求时负载均衡节点本身可能会成为瓶颈

```
发送请求：User -> LoadBalancer -> Server
接收响应：User <- LoadBalancer <- Server
```

    - LVS DR 模式

      LoadBalancer 与 Server 同在一个网段, 共享同一个公网 IP, 响应报文可以由 Server 直达 User

```
发送请求：User -> LoadBalancer -> Server
接收响应：User <———————————— Server
```

- 可以不使用 Nginx, 直接用 gunicorn 吗?
    - 测试可以，正式不可以。

- Nginx 相对于 Gunicorn 来说更安全
    - Nginx 可以用作负载均衡.
- 处理静态文件相关配置

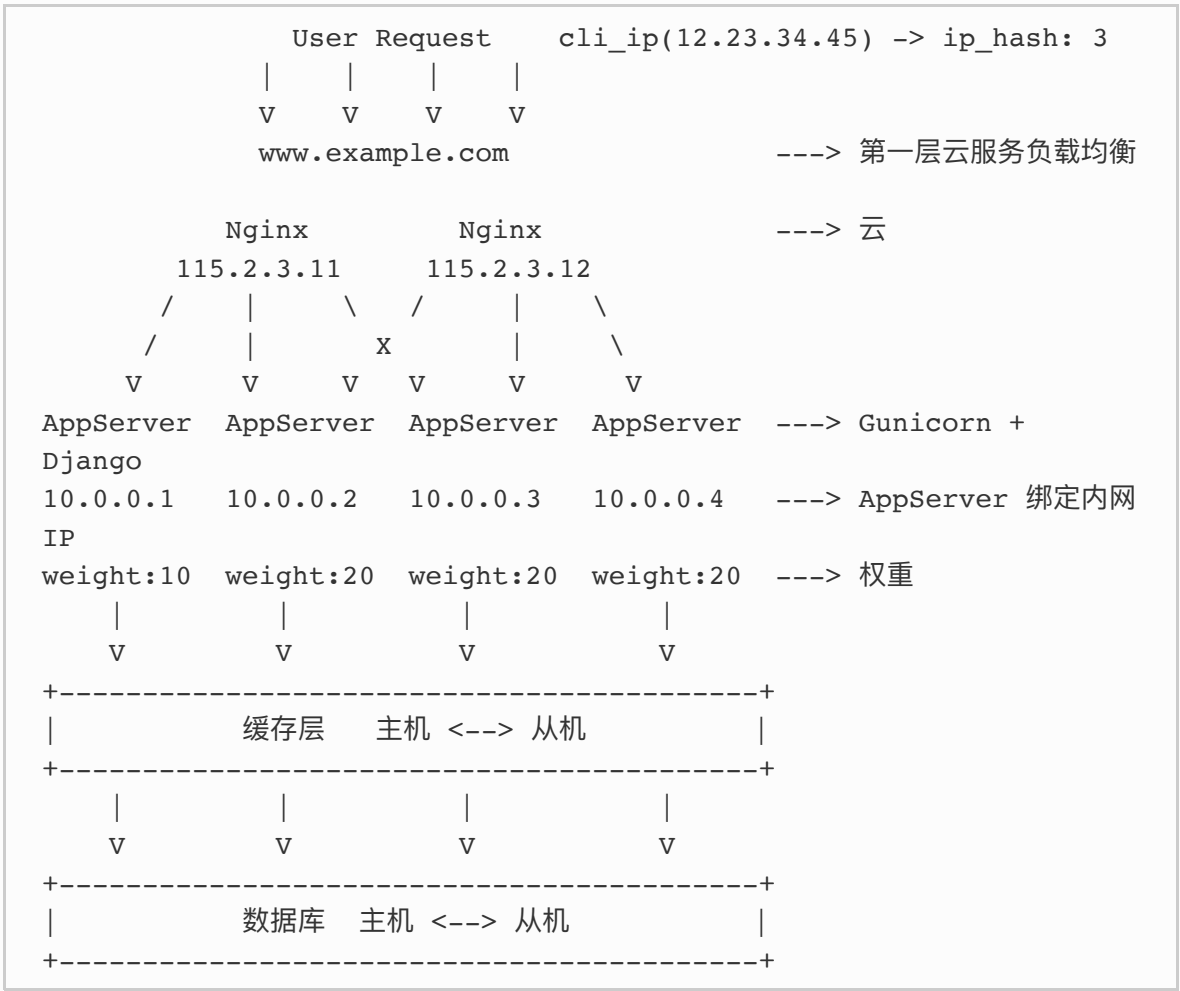
```
location /statics/ {
    root   /project/bbs/;
    expires 30d;
    access_log off;
}

location /medias/ {
    root   /project/bbs/;
    expires 30d;
    access_log off;
}
```

- 面试题：
    - Nginx如何做负载均衡?

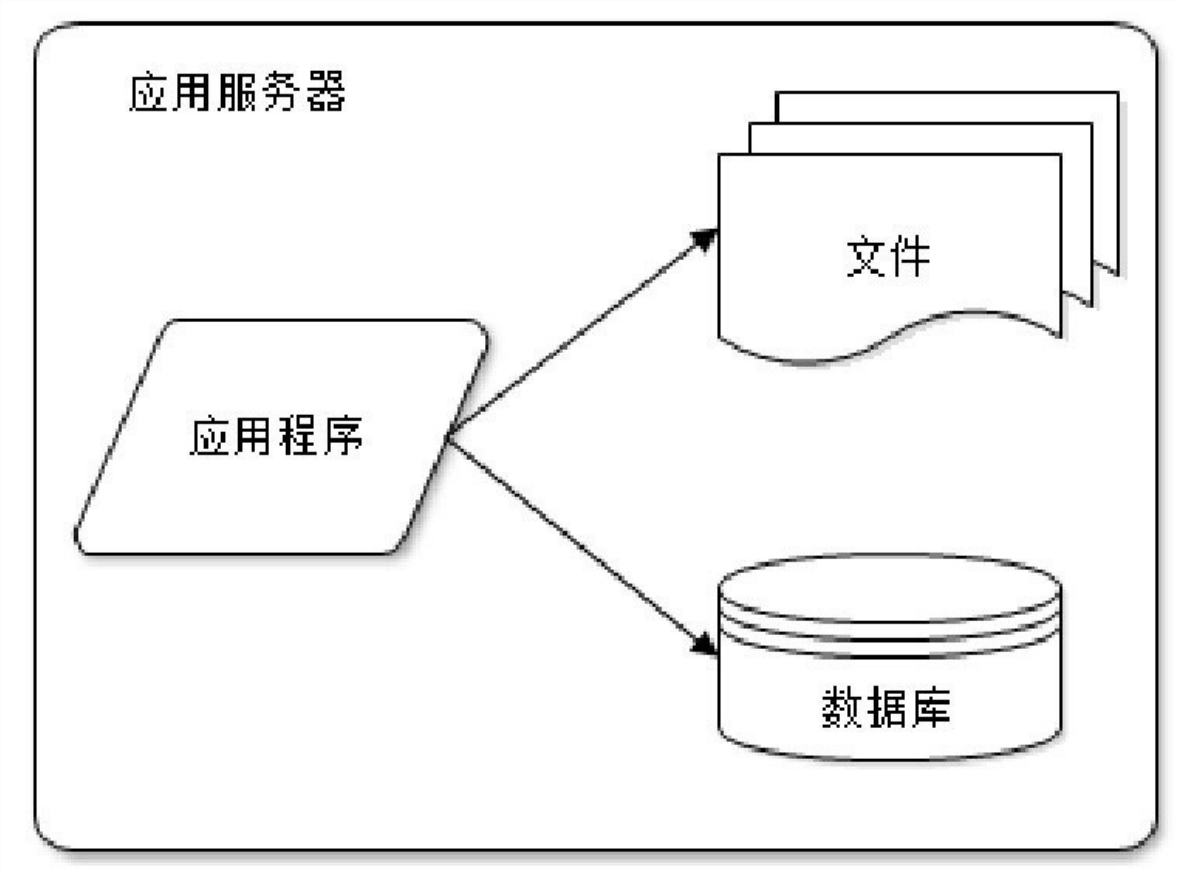
服务器架构

1. 架构研究的 5 个方面
    - 高性能：请求再多，也能服务
    - 高可用：情况再糟，也能服务
    - 可伸缩：请求多了能服务，请求少了也能节省下来资源
    - 可扩展：能不断的加新的服务
    - 安全性：完成以上能力的前提是，不出安全问题
2. 简单、实用的服务器架构图
    - 分层结构: 功能模块解耦合
    - 每层多台机器: 有效避免单点故障
    - 每层均可扩容: 能通过简单的方式提升服务器的性能、可用性、扛并发能力

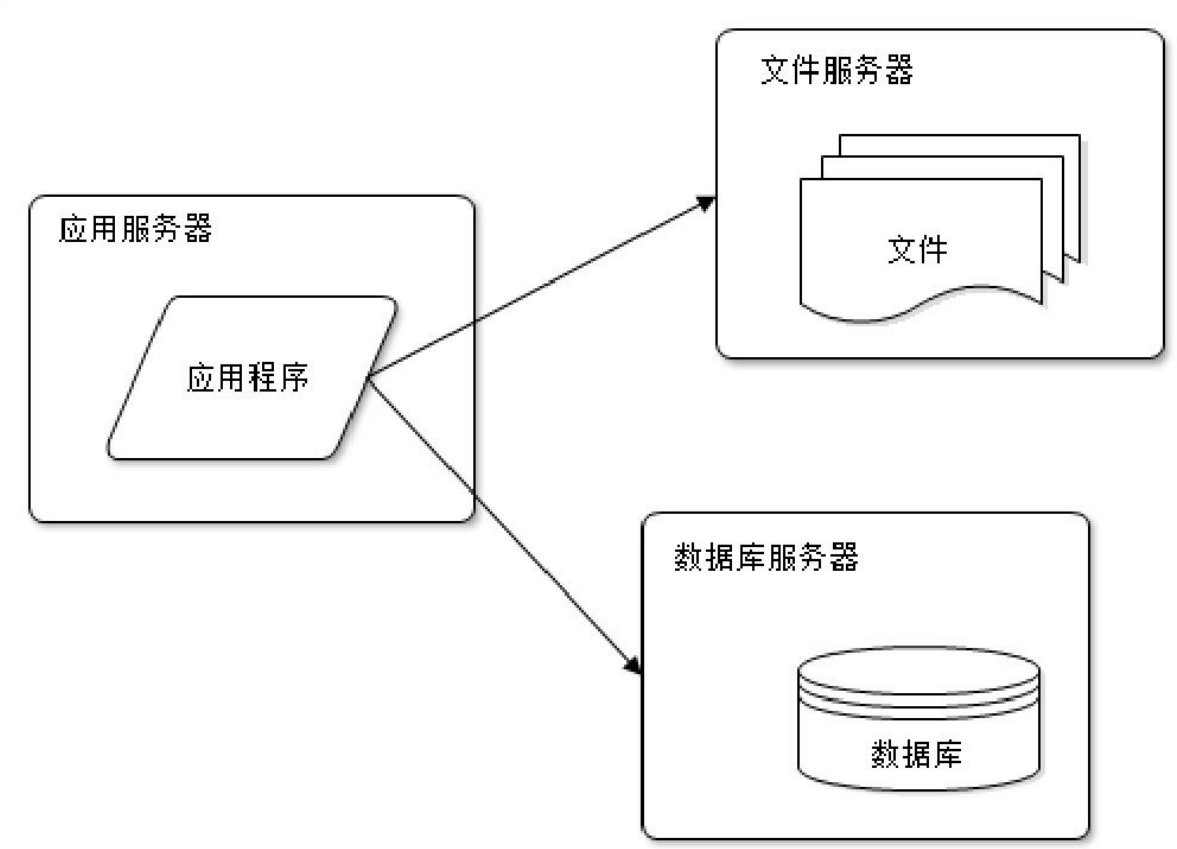


服务器架构的发展

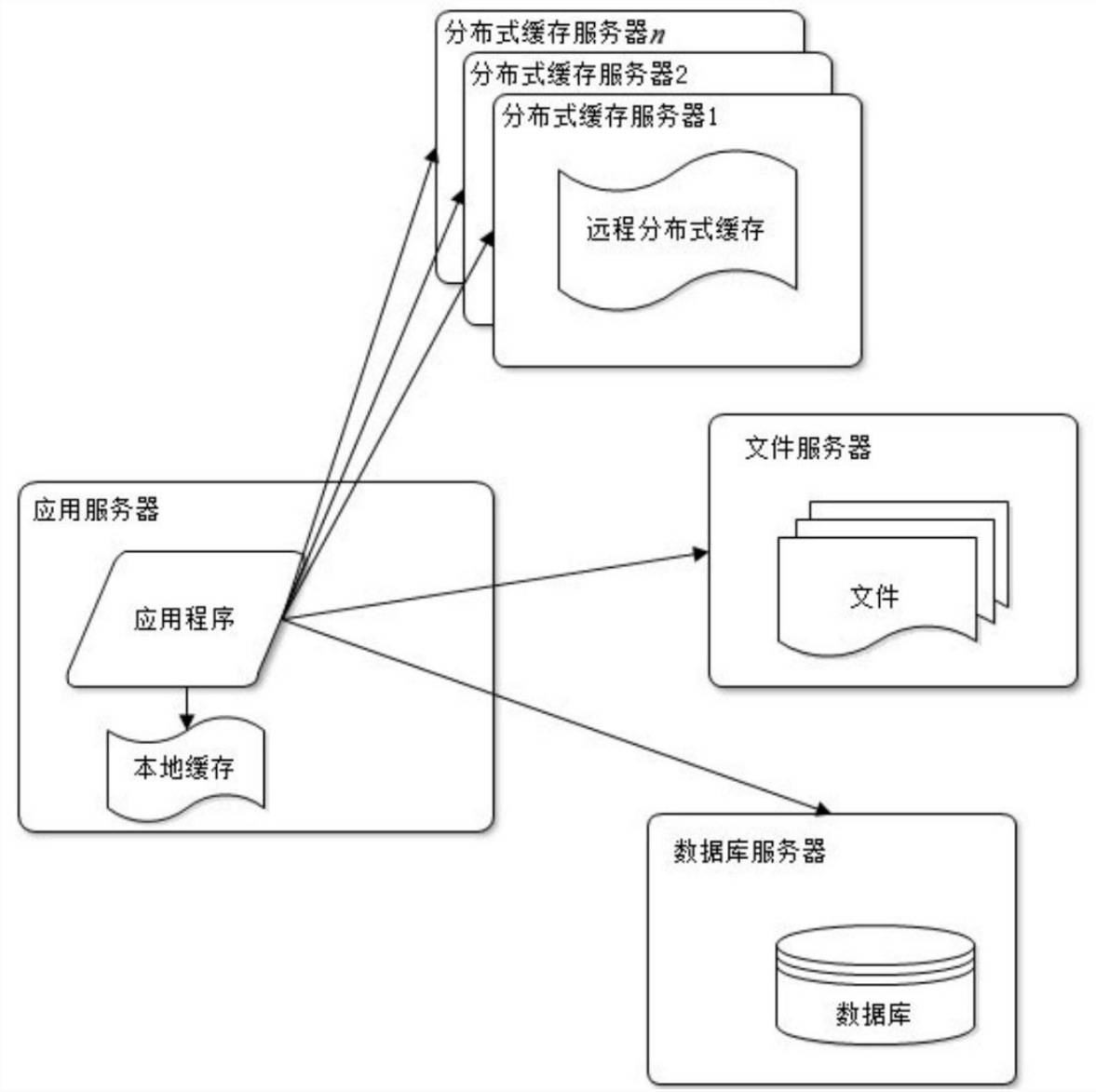
- 早期服务器, 所有服务在一台机器



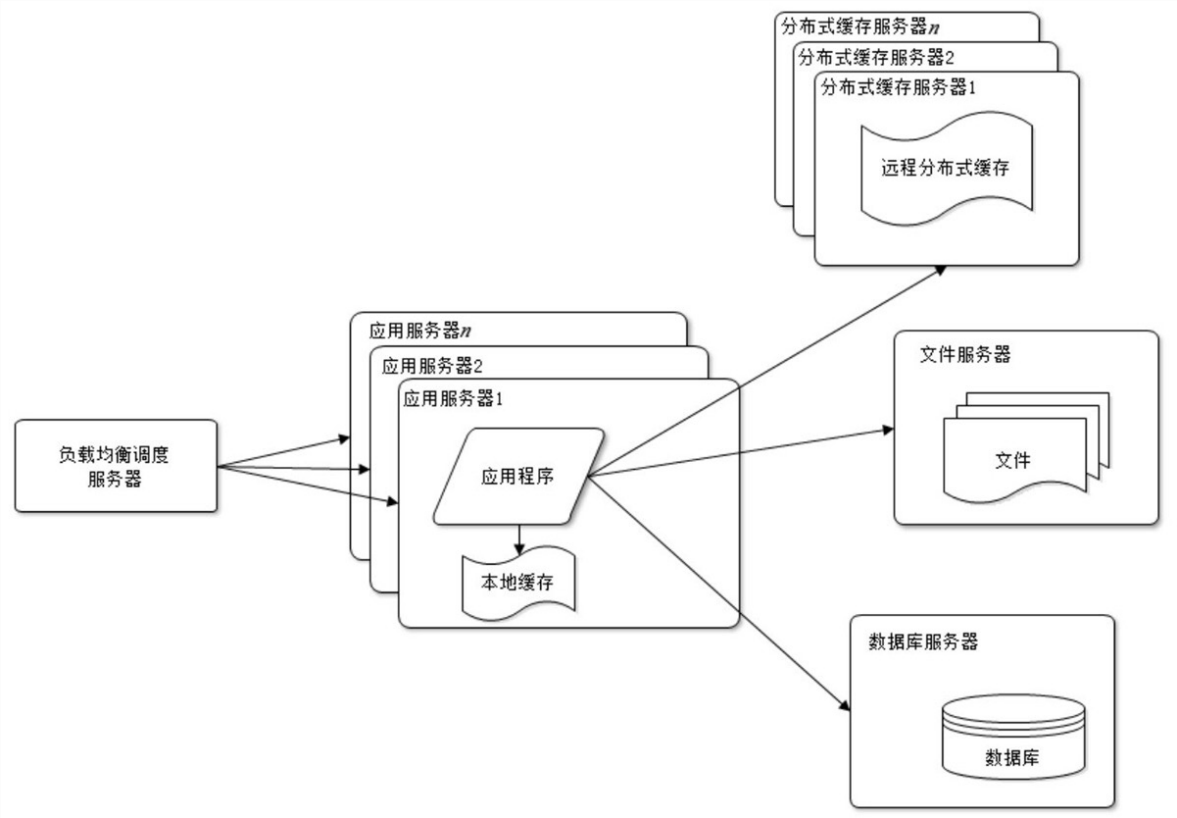
- 服务拆分，应用、数据、文件等服务分开部署



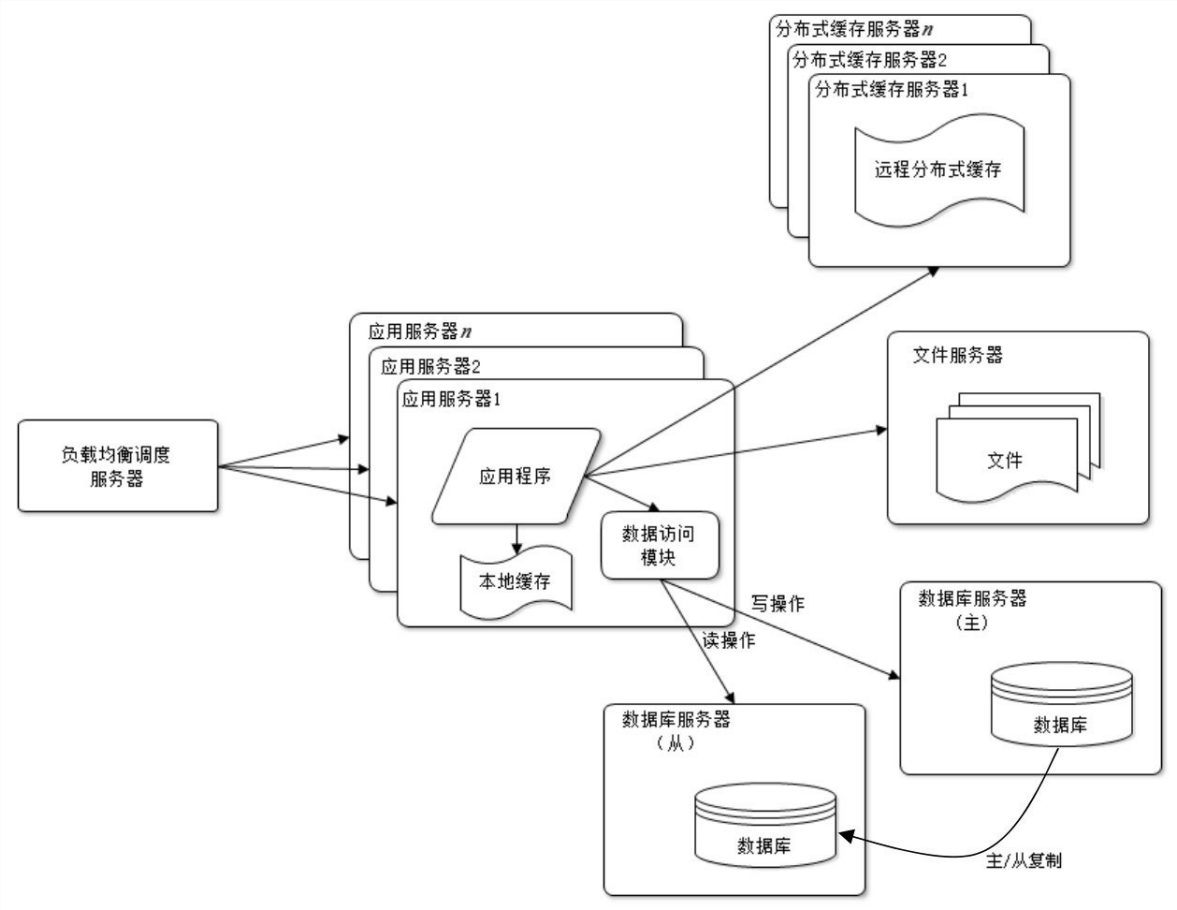
- 利用缓存提升性能


分布式缓存服务器n
分布式缓存服务器2
分布式缓存服务器1
远程分布式缓存
文件服务器
文件
应用服务器
应用程序
本地缓存
数据库服务器
数据库

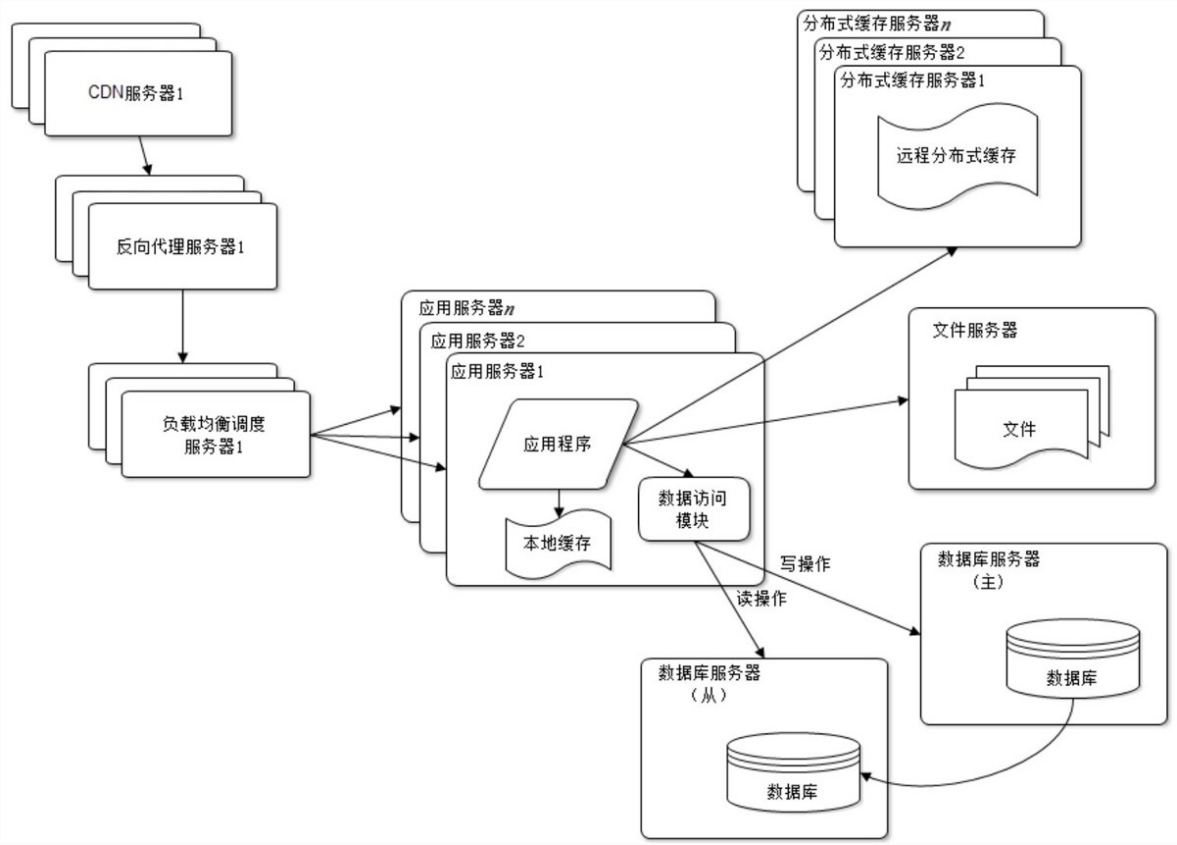
- 应用服务器分布式部署，提升网站并发量和吞吐量



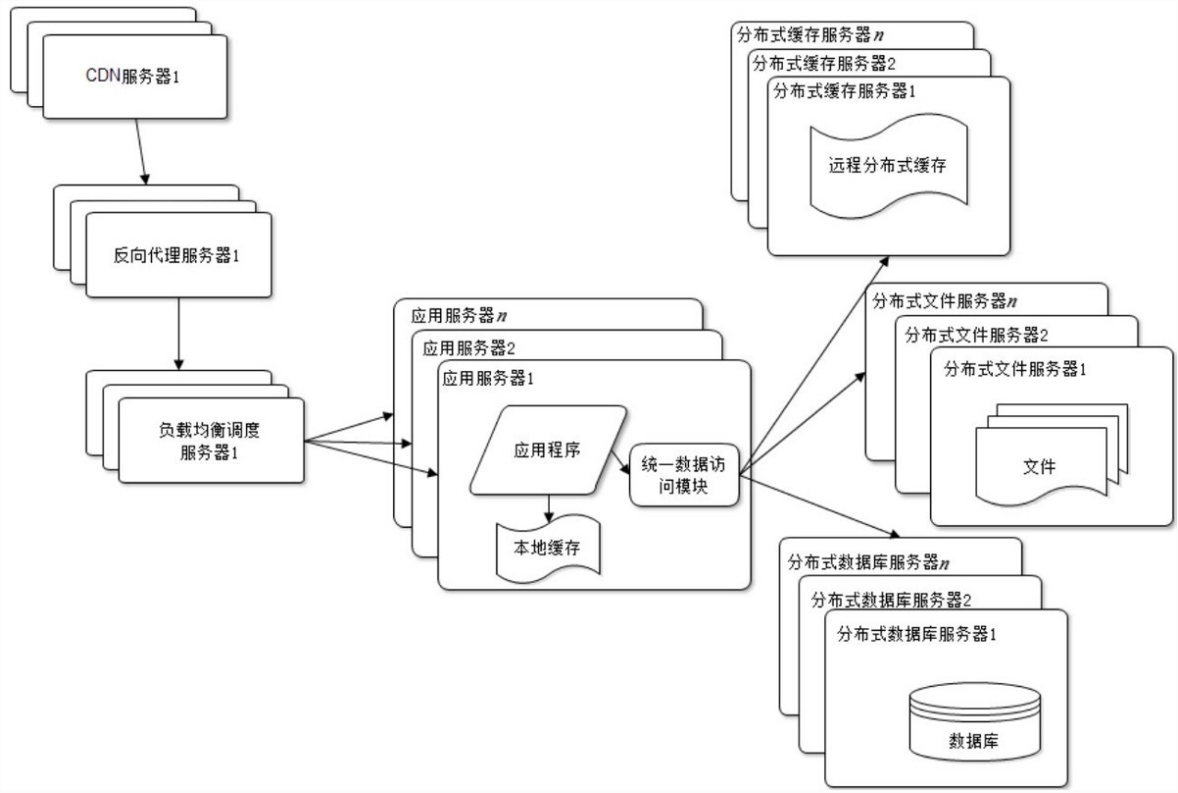
- 通过读写分离读写分离提升数据库性能和数据可靠性



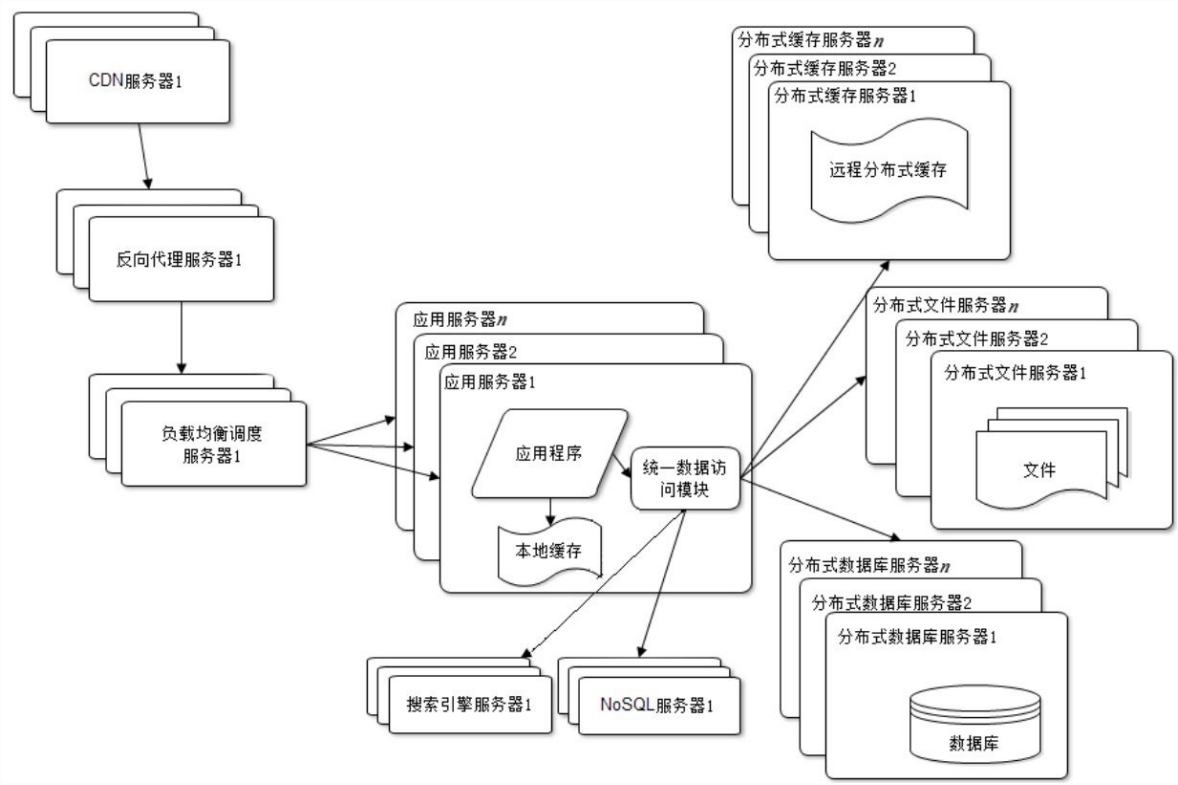
- 使用反向代理、CDN、云存储等技术提升静态资源访问速度，并能有效提升不同地域的访问体验



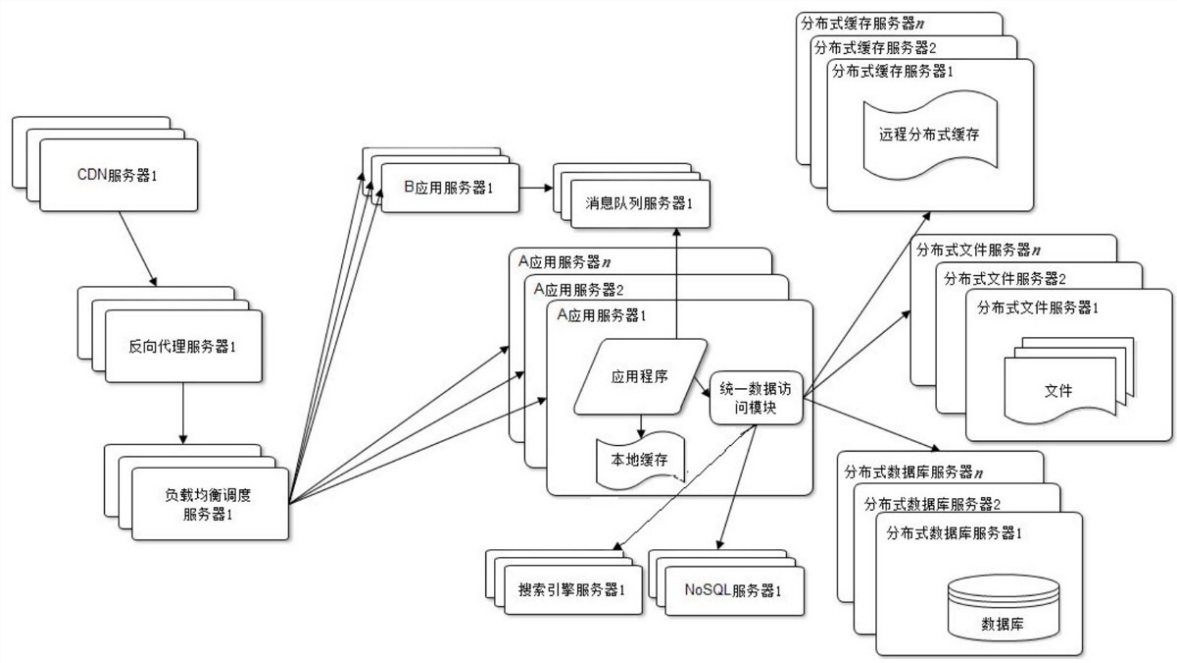
- 通过分布式数据库和分布式文件系统满足数据和文件海量存储需求，并进一步提升数据可靠性



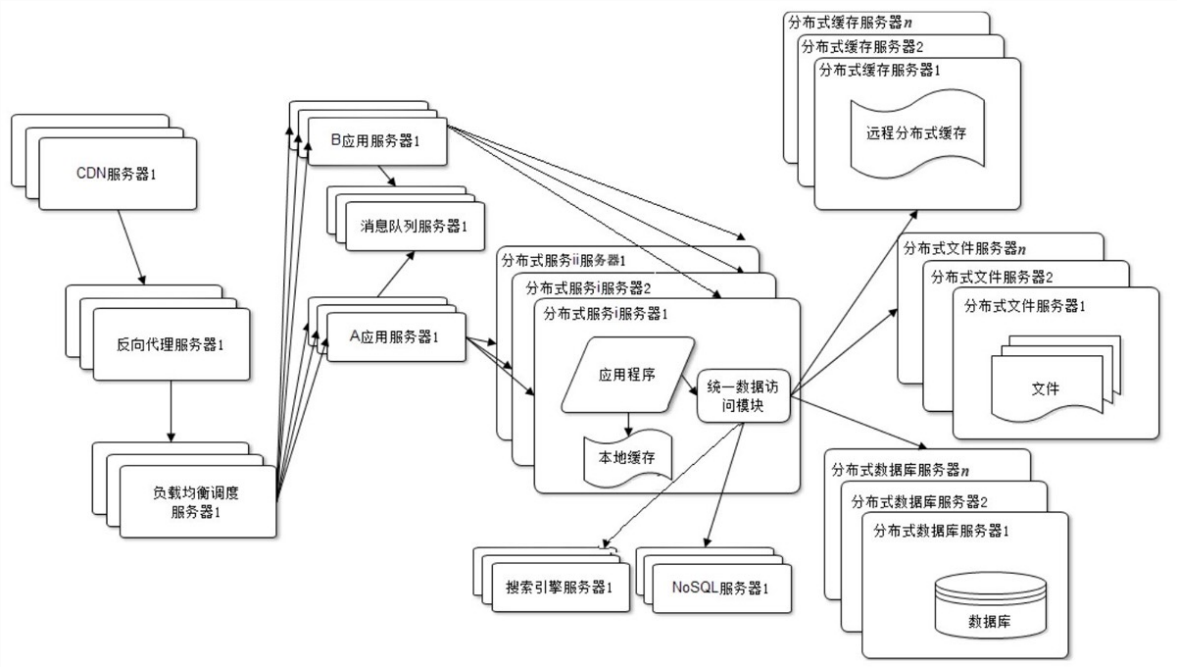
- 增加搜索引擎 和 NoSQL



- 增加消息队列服务器，让请求处理异步化



- 拆分应用服务器与内部服务



- 微服务
    - flask restful

服务器计算

- 费米预测
- 服务器性能预估
    1. 首先需知道网站日活跃 (DAU) 数据
    2. 按每个活跃用户产生 100 个请求计算出“每日总请求量”

       不同类型的网站请求量差异会很大, 可以自行调整一个用户产生的请求数

       ```
       每日总请求量 = DAU x 单个用户请求量
       ```

    3. 有了总请求量便可计算“每日峰值流量”, 流量一般单位为 rps (requests per second)

       根据经验可知: 每天 80% 的请求会在 20% 的时间内到达

       由此可知:

       $$每日峰值流量 = \frac{每日总请求量 \times 80\%}{86400 \times 20\%}$$

    4. 一般带负载的 web 服务器吞吐量约为 300rps, 所以:

       ```
       WebServer 数量 = 每日峰值流量 / 300
       ```

    5. 得到 WebServer 数量以后, 再根据用户规模和请求量估算 Nginx、Cache、Database 等服务器的数量
- 真实工作中服务器分配情况
    - 业务类型:
        1. 新闻网站：单用户产生请求：30 - 50
        2. 游戏网站：单用户请求可能：100 - 200
        3. 社区网站：单用户请求可能：50 - 100
        4. 平均可以取 100
    - 每日总请求 = DAU * 单用户每日请求
        1. DAU 20万：200000 * 100 = 20,000,000 两千万
        2. DAU 50万：5000 万
        3. DAU 100万：1亿
        4. DAU 200万：2亿
    - 峰值请求:
        1. 按经验所有请求在10小时内发生

1. 峰值请求 = 每日总请求 / 10 / 3600
    1. 20万：20000000 / (10 * 3600) = 556
    2. 100万：556 * 5
    3. 200万：
2. 按80%的请求在20%的时间内发生
    1. 峰值请求 = 每日总请求 * 80% / (24 * 20%)
        1. 20万DAU：926次/秒

- 服务器预估
    1. 单 Web 服务器并发 500 rps
        1. 20万：峰值请求 / 500 = 2台
        2. 100万：556 * 5 / 500 = 6台
- 相册服务
    1. 业务限制用户照片，或者等用户上传后自动帮用户压缩
        1. 5M = 500kB
    2. 预估每个用户多少张照片：
        1. 500张
        2. 2000张
    3. 每张照片多大 2-9
        1. 平均5M
    4. 一个用户总空间多少?
        1. 500 * 5MB = 2500MB = 2.5GB
        2. 2000 * 5MB = 10GB
    5. 每个服务器存多少张照片
        1. 10TB = 1000GB = 1000个用户
        2. 20万 200台服务器
    6. 用户20万
    7. 用户100万
    8. 用户200万

- 面试题：
    - 你们有多少用户?
    - 多少日活?
    - 多少服务器?
    - 服务器并发有多高?