如何构建大规模互联网服务？

2018/08

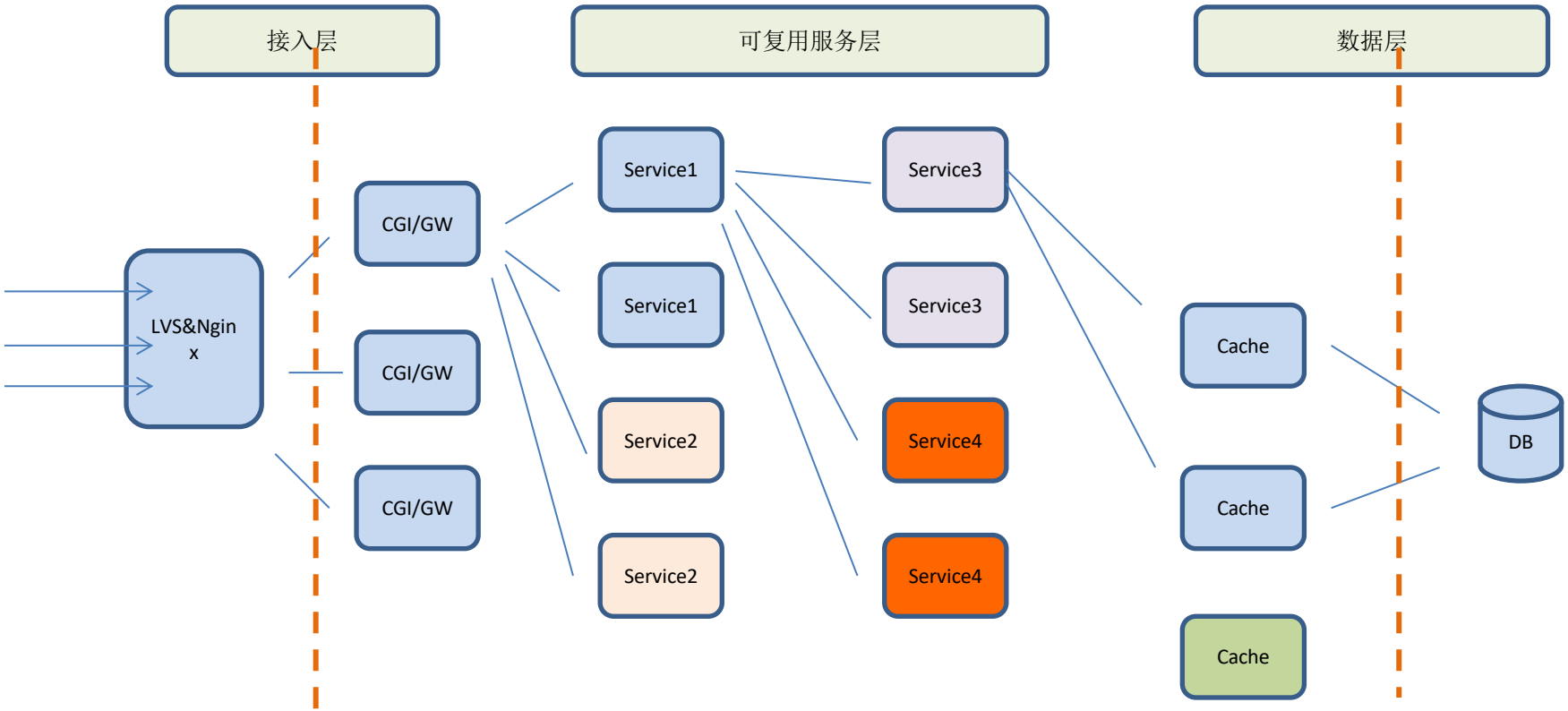
What？

- 7 * 24 Stable
- High QPS & Low Latency
- Large-Scale User & Data(需要分布式架构)
- Massive in-parallel development
- Highly efficient testing, deployment, maintenance

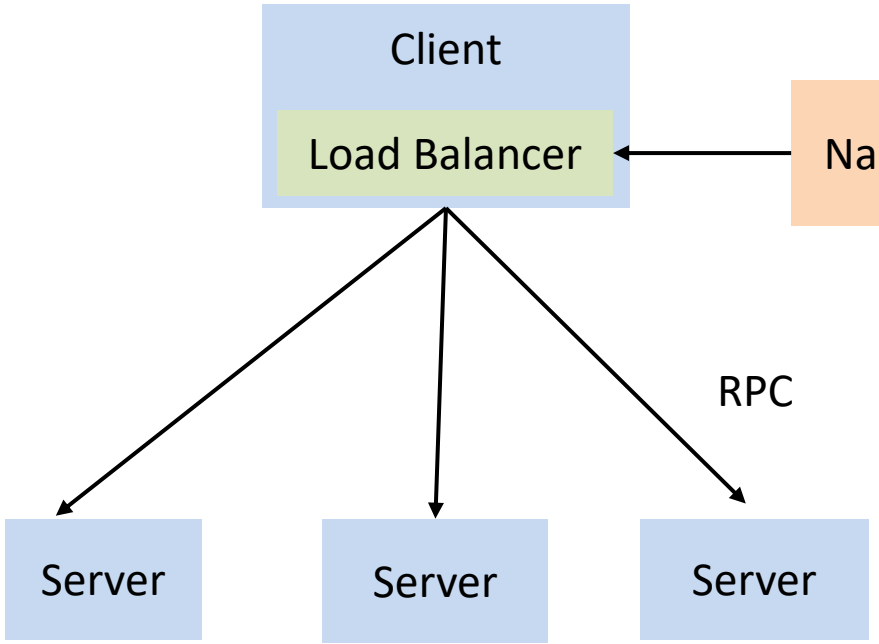
设计目标

- Availability：服务无中断的、在合理的时间内返回合理的结果（no error，no timeout）的能力
- Scalability：不需要改变系统软硬件设计，仅靠改变机器数量就可以增大缩小服务的处理能力
- Extensibility：系统对业务发展变化的适应性能力，业务变化时系统是否需要大范围改动
- Manageability：全方位监控、部署，运维，故障处理自动化
- Cost：机器成本、运维成本

基本分层架构模板


接入层
可复用服务层
数据层
LVS&Ngin
x
CGI/GW
CGI/GW
CGI/GW
Service1
Service1
Service2
Service2
Service3
Service3
Service4
Service4
Cache
Cache
Cache
DB

要解决的基本问题？



- 服务注册与发现？
- 怎么扩容/缩容？
- 服务宕机？
- 数据传输格式(序列化)/协议?
- 怎么和服务集群交互、负载均衡?
- 服务不响应怎么办? 重试？幂等f(f(x)) = f(x) ？
- 跨语言调用？

Availability – 怎么定义可用性？

- MTBF : Mean Time Between Failure (系统平均故障间隔时间)
- MTTR : Mean Time To Repair(系统平均修复时间)
- Availability = MTBF / (MTBF + MTTR) 提高可用率则要降低故障概率，降低故障恢复时间
- 5个9：(1-99.999%)*365*24*60=5.26分钟，表示在连续运行1年时间里最多可能的业务中断时间是5.26分钟

Availability – 为什么需要高可用？

- 服务器故障
- 网络故障
- 服务变更故障
- 电力故障
- 自然灾害

Availability – 怎么实现高可用？

本质是：　多副本（Replica）冗余和故障转移（Fail-Over）

- 多副本
  - 机器级副本
  - 集群级副本（单元化/SET）
  - 机房级副本：异地多活
- 负载均衡
  - Random
  - RR
  - WRR
  - LALB：Latency-Aware LB
  - Consistent Hash
  - 按业务切分：按UID范围切分、地址位置切分
- 服务治理
  - 过载保护
  - 柔性可用（降级、熔断、限流、排队）
  - 灰度发布、快速回滚
  - 自动化测试
  - 立体监控
- 存储层
  - Master-Slave
  - Paxos、Raft

Scalability

- 入口层：DNS、ECMP
- 逻辑层：无状态+负载均衡
- 存储层
  - 分库分表
  - Sharding：Range based、余数Hash、Consistent Hash、Route Table

Performance – 评测公式

- 串行调用：Latency = SUM(Latency(1), Latency(2), …)
- 并发调用：Latency = MAX(Latency(1), Latency(2), …)
- QPS = WorkerNum * 1000 / avg.latency
- TP99, TP999 Latency = the minimum latency under which 99.9% of requests has been served
- Concurrent Requests = QPS * Latency(in seconds) 表示服务端需要开启多少个并发Worker？使用同步还是异步？

Performance – 怎么实现高性能？

- 单机高性能
  - Reactor / Proactor
  - Memory pool、thread pool、object pool、lock-free / wait-free、Zero-Copy
  - Asynchronous、Coroutine
  - Batch Request，Pipeline Request
  - 缓存
- 集群高性能
  - 负载均衡算法
  - Backup Request

Extensibility

- 单机可扩展性
  - Interface based programming
  - MVC
  - 设计模式
- 服务可扩展性
  - 微服务：基本思想是“拆分”
  - 分布式事务（补偿、可靠消息、TCC）
  - 消息队列

Security

- XSS
- CSRF
- SQL注入

基础设施

- RPC框架（网络模型、线程模型、任务模型）
- 注册中心、配置中心
- 服务网关
- 监控及分布式跟踪
- 自动化测试、部署、运维
- 数据库平台、缓存平台
- 大/小文件存储平台、图片平台

参考资料

- RPC框架：https://github.com/brpc/brpc
- Raft：https://github.com/brpc/braft
- Paxos：https://github.com/tencent/phxpaxos