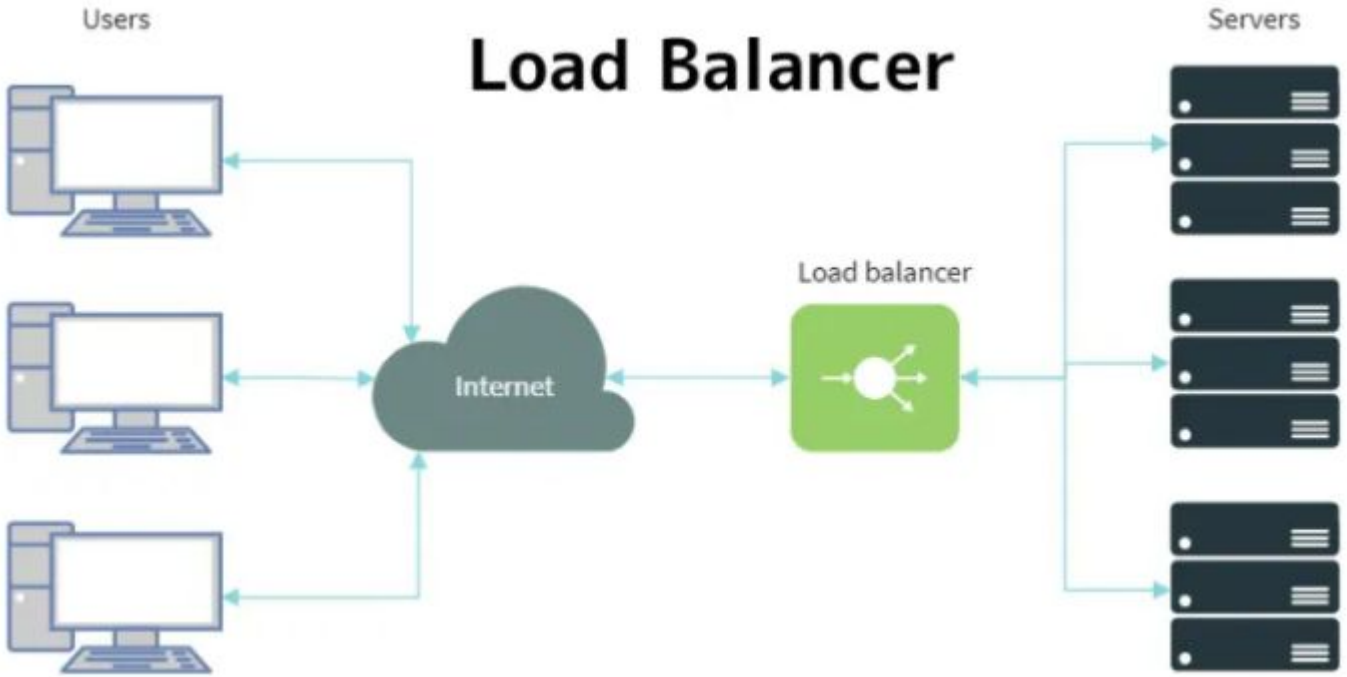
系统设计中
Load Balancer的必知必会

垂直扩展 vs 水平扩展

什么是LB



LB解决的复杂度问题

- 在多台服务器之间有效地分配客户端请求或网络负载，防止资源过载
- 通过仅向在线服务器发送请求来确保高可用性和可靠性，防止请求进入不好的服务器
- 提供根据需求指示添加或减少服务器的灵活性
- 帮助消除单一的故障点

如何实现LB

- 硬件
- 软件
- 客户端

LB一些设计“原则”

- Filter by Availability

  default设计：LB去连接一个主机，如果 连续三次失败，会发生类似“电路跳闸”的行为，一旦跳闸，这个状态持续30s，继续去连接，再跳闸，下次就是1分钟，然后2分钟，以指数形式上升。

- Response time的权重

  每一个机器会根据主机的反应时间，来给主机权重。主机用的时间越长，权重就越小，机器就越忙，可能处理的时间就越长，那就是告诉LB，我活很多了，别发给我了，但是这个是有弊端的，因为LB并不知道这个延迟是发生在哪个部分，可能是数据库，可能是网络，可能本身这个 API 就是很费时。

LB相关的算法

LB在将请求转发到后端服务器之前会考虑两个因素：

1）首先确保它们选择的服务器实际上对请求做出了适当的响应

2）然后使用预先配置的算法从运行状况良好的服务器集合中选择一个

Load Balancer Algorithm - Round Robin

```java
import java.util.List;

public class RoundRobinLB {
    /** 服务器列表 */
    private static List<String> serverList = new ArrayList<>();
    static {
        serverList.add("192.168.1.1");
        serverList.add("192.168.1.2");
        serverList.add("192.168.1.3");
        serverList.add("192.168.1.4");
    }
    private static Integer index = 0;

    public static String getRRServer() {
        // 复制遍历用的集合，防止操作中非线程安全的集合有变更
        List<String> tempList = new ArrayList<>(serverList.size());
        tempList.addAll(serverList);
        String server = "";
        synchronized (index) {
            index++;
            if (index == tempList.size()) {
                index = 0;
            }
            server = tempList.get(index);;
        }
        return server;
    }
}
```

Load Balancer Algorithm - Weighted Round Robin

```java
public class WeightedRoundRobinLB {
    private static HashMap<String, Integer> serverMap = new HashMap<>();
    static {
        serverMap.put("192.168.1.1", 2);
        serverMap.put("192.168.1.2", 2);
        serverMap.put("192.168.1.3", 2);
        serverMap.put("192.168.1.4", 4);
    }
    private static Integer index = 0;
    public static String getWeightedRRServer() {
        List<String> tempList = new ArrayList();
        HashMap<String, Integer> tempMap = new HashMap<>();
        tempMap.putAll(serverMap);
        for (String key : serverMap.keySet()) {
            for (int i = 0; i < serverMap.get(key); i++) {
                tempList.add(key);
            }
        }
        synchronized (index) {
            index++;
            if (index == tempList.size()) {
                index = 0;
            }
            return tempList.get(index);
        }
    }
}
```

Load Balancer Algorithm - Random

```java
import java.util.*;

public class RandomLB {
    /** 服务器列表 */
    private static List<String> serverList = new ArrayList<>();
    static {
        serverList.add("192.168.1.1");
        serverList.add("192.168.1.2");
        serverList.add("192.168.1.3");
        serverList.add("192.168.1.4");
    }
    /**
     * 随机路由算法
     */
    public static String random() {
        // 复制遍历用的集合，防止操作中非线程安全的集合有变更
        List<String> tempList = new ArrayList<>(serverList.size());
        tempList.addAll(serverList);
        // 随机数随机访问
        int randomInt = new Random().nextInt(tempList.size());
        return tempList.get(randomInt);
    }
}
```

Load Balancer Algorithm - Weighted Random

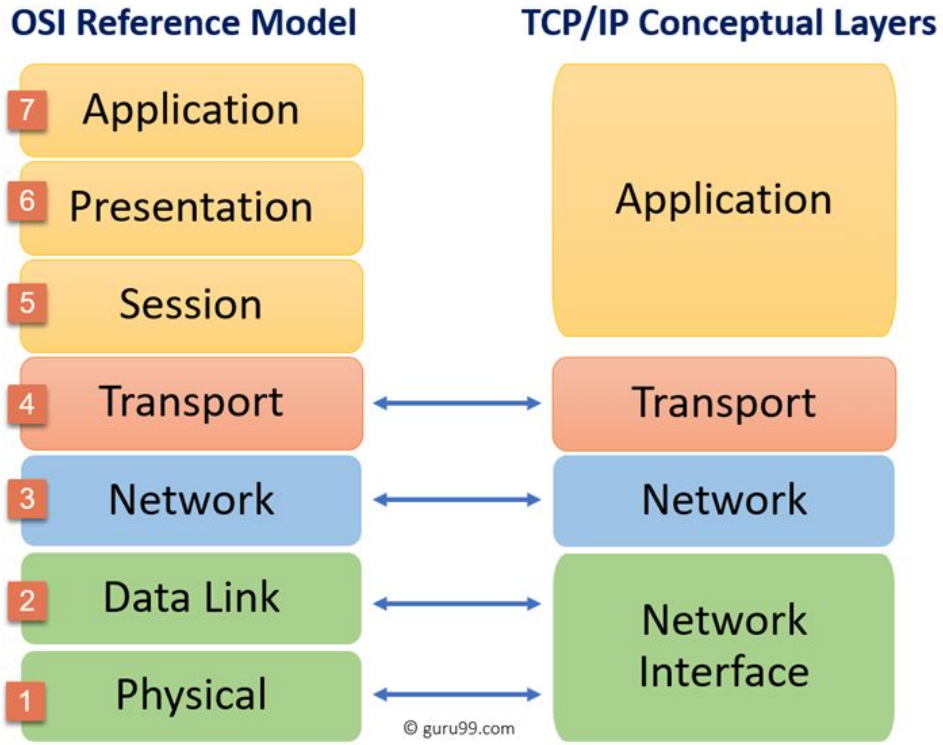
```java
public class WeightedRandomLB {
    private static HashMap<String, Integer> serverMap = new HashMap<>();
    static {
        serverMap.put("192.168.1.1", 2);
        serverMap.put("192.168.1.2", 2);
        serverMap.put("192.168.1.3", 2);
        serverMap.put("192.168.1.4", 4);
    }
    /**
     * 加权路由算法
     */
    public static String randomWithWeight() {
        List<String> tempList = new ArrayList();
        HashMap<String, Integer> tempMap = new HashMap<>();
        tempMap.putAll(serverMap);
        for (String key : serverMap.keySet()) {
            for (int i = 0; i < serverMap.get(key); i++) {
                tempList.add(key);
            }
        }
        int randomInt = new Random().nextInt(tempList.size());
        return tempList.get(randomInt);
    }
}
```

LB相关的算法 - IP哈希

Load Balancer Algorithm - Least Connection/Fastest Response

L4 LB vs L7 LB

OSI: https://blog.csdn.net/yaopeng_2005/article/details/7064869



LB解析HTTPS