第二十讲：分布式系统

第 1 节：分布式系统概述

向勇、陈渝、李国良

清华大学计算机系

xyong,yuchen,liguoliang@tsinghua.edu.cn

2021 年 5 月 10 日

提纲

1. 第 1 节：分布式系统概述
   - 通信基础
   - 通信抽象

通信基础

分布式系统改变了世界的面貌
关键问题：如何构建在组件故障时仍能工作的系统

通信基础

不可靠的通信层

- 现代网络的核心原则是，通信基本是不可靠的。
- 丢包是网络的基本现象
- 应该如何处理丢包？

通信基础

一个基于 UDP/IP 构建的简单客户端和服务器

```
// client code
int main(int argc, char *argv[]) {
    int sd = UDP_Open(20000);
    struct sockaddr_in addr, addr2;
    int rc = UDP_FillSockAddr(&addr, "machine.cs.wisc.edu", 10000);
    char message[BUFFER_SIZE];
    sprintf(message, "hello world");
    rc = UDP_Write(sd, &addr, message, BUFFER_SIZE);
    if (rc > 0) {
        int rc = UDP_Read(sd, &addr2, buffer, BUFFER_SIZE);
    }
    return 0;
}
```

通信基础

一个基于 UDP/IP 构建的简单客户端和服务器

```
// server code
int main(int argc, char *argv[]) {
    int sd = UDP_Open(10000);
    assert(sd > -1);
    while (1) {
        struct sockaddr_in s;
        char buffer[BUFFER_SIZE];
        int rc = UDP_Read(sd, &s, buffer, BUFFER_SIZE);
        if (rc > 0) {
            char reply[BUFFER_SIZE];
            sprintf(reply, "reply");
            rc = UDP_Write(sd, &s, reply, BUFFER_SIZE);
        }
    }
    return 0;
}
```

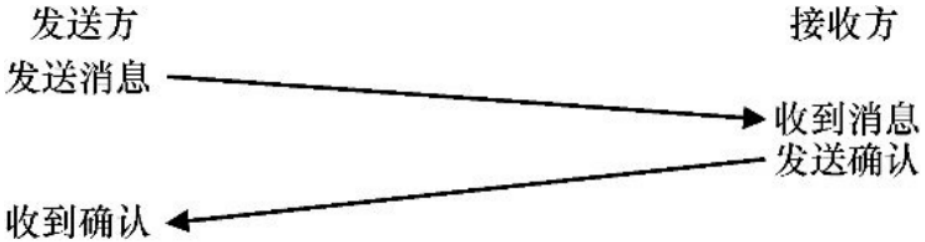
可靠的通信层

问题：发送方如何知道接收方实际收到了消息？

可靠的通信层

问题：发送方如何知道接收方实际收到了消息？

- 确认（acknowledgment），或简称为 ack
- 发送方向接收方发送消息，接收方然后发回短消息确认收到



图：消息加确认

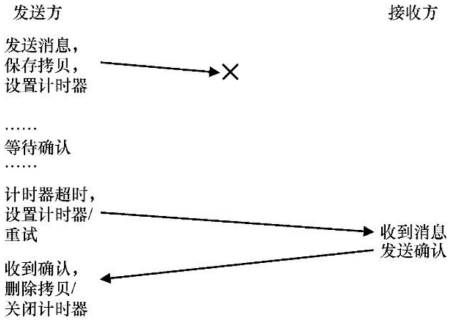
可靠的通信层

问题：如果没有收到确认，发送方应该怎么办？

可靠的通信层

问题：如果没有收到确认，发送方应该怎么办？

- 额外的机制，称为超时（timeout）+ 重试（retry）
- 当发送方发送消息后，如在一定时间内未收到确认，则断定该消息已丢失
- 发送方然后就重试（retry）发送，再次发送相同的消息，希望这次它能送达



图：消息加确认：丢失的请求

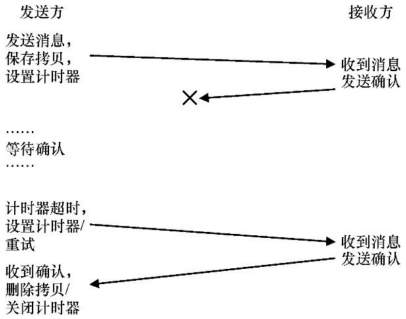
可靠的通信层

问题：如果没有收到确认，发送方应该怎么办？

可靠的通信层

问题：如果没有收到确认，发送方应该怎么办？

- 额外的机制，称为超时（timeout）+ 重试（retry）
- 当发送方发送消息后，如在一定时间内未收到确认，则断定该消息已丢失
- 发送方然后就重试（retry）发送，再次发送相同的消息，希望这次它能送达



图：消息加确认：丢失回答

可靠的通信层

问题：如何保证接收方每个消息只接收一次（exactly once）？

可靠的通信层

问题：如何保证接收方每个消息只接收一次（exactly once）？

- 发送方可以为每条消息生成唯一的 ID，接收方可以追踪它所见过的每个 ID
- 顺序计数器（sequence counter）
  - 无论何时发送消息，计数器的当前值都与消息一起发送。此计数器值（N）作为消息的 ID。发送消息后，发送方递增该值（到 N + 1）。
  - 接收方使用其计数器值，作为发送方传入消息的 ID 的预期值。如果接收的消息（N）的 ID 与接收方的计数器匹配（也是 N），它将确认该消息，将其传递给上层的应用程序。在这种情况下，接收方断定这是第一次收到此消息。接收方然后递增其计数器（到 N + 1），并等待下一条消息。
  - 如果确认丢失，则发送方将超时，并重新发送消息 N。这次，接收器的计数器更高（N+1），因此接收器知道它已经接收到该消息。因此它会确认该消息，但不会将其传递给应用程序。

提纲

1. 第 1 节：分布式系统概述
   - 通信基础
   - 通信抽象

通信抽象

问题：有了基本的消息传递层，在构建分布式系统时，应该使用什么抽象通信？

- 操作系统抽象：分布式共享内存（Distributed Shared Memory，DSM）

通信抽象

问题：有了基本的消息传递层，在构建分布式系统时，应该使用什么抽象通信？

- 操作系统抽象：分布式共享内存（Distributed Shared Memory，DSM）
  - 使不同机器上的进程能够共享一个大的虚拟地址空间
  - 通过操作系统的虚拟内存系统来实现
  - DSM 最大的问题是它如何处理故障
  - 另一个问题是性能
  - 这种方法今天并未广泛使用

通信抽象

问题：有了基本的消息传递层，在构建分布式系统时，应该使用什么抽象通信？

- 编程语言（PL）抽象：远程过程调用（Remote Procedure Call），或简称 RPC

通信抽象

问题：有了基本的消息传递层，在构建分布式系统时，应该使用什么抽象通信？

- 编程语言（PL）抽象：远程过程调用（Remote Procedure Call），或简称 RPC
  - 目标：使在远程机器上执行代码的过程像调用本地函数一样简单直接。
  - RPC 系统通常有两部分：存根生成器（stub generator，有时称为协议编译器，protocol compiler）和运行时库（run-time library）。

通信抽象 – 存根生成器

存根生成器 (Stub Generator)：存根生成器接受接口 (interface) 代码，并生成客户端存根（client stub）和服务端代理 (server proxy)
客户端存根（client stub）

- 创建消息缓冲区; 将所需信息打包到消息缓冲区中;
- 将消息发送到目标 RPC 服务器; 等待回复;
- 解包返回代码和其他参数; 返回调用函数。

```
interface {
  int func1(int arg1);
  int func2(int arg1, int arg2);
};
```

通信抽象 – 存根生成器

存根生成器 (Stub Generator)：存根生成器接受接口 (interface) 代码，并生成客户端存根（client stub）和服务端代理 (server proxy)
服务端代理 (server proxy)

- 解包消息;
- 调用实际函数;
- 打包结果；发送回复。

```
interface {
  int func1(int arg1);
  int func2(int arg1, int arg2);
};
```

通信抽象 – 运行时库

运行时库处理 RPC 系统中的大部分繁重工作

- 如何找到远程服务？
- 如何构建 RPC 的传输级协议？

```
interface {
  int func1(int arg1);
  int func2(int arg1, int arg2);
};
```

通信抽象 – 需思考的问题

值得进一步思考的问题

- 调用中的复杂参数，即一个包如何发送复杂的数据结构？
- 并发性的服务器组织方式？
- 当远程调用需要很长时间才能完成时，会发生什么？
- 是否向客户端暴露通信的异步性质，从而实现一些性能优化？

```
interface {
  int func1(int arg1);
  int func2(int arg1, int arg2);
};
```