分布式系统作业-1

18340013 陈琮昊

1.分布式系统的扩展方式有哪些？各有哪些利弊？

（I）分布式系统的可扩展性体现在如下三个方面：

规模上的扩展：能够根据需要增加更多机器，保存大量数据，提高运算的效率；但一旦出现问题，故障排除便是一大难题。

地理上的扩展：能够在不同地区使用，大大提高了人们的出行效率。但在广域的环境下，延迟就会更加明显；LAN到WAN的扩展也并不是那么轻而易举；缺少多点通信，广播不能执行。

管理上的扩展：可以提高可靠性等。但是会有与使用方法、管理和安全相关的策略冲突。

（II）至于扩展技术则如下：

隐藏通信延迟：原理较为简单，能够避免等待远程服务的请求响应；但对于有些交互式应用来说并不适用。

分布：在多个机器划分数据计算，提高了工作量、效率；但要考虑同步异步、一致性等问题。

复制：使用起来较为简单，让分布在不同地方的数据副本都能够为用户提供服务。但也存在一致性问题。

2.分布式系统设计面临哪些挑战？请举例回答。

第一，异构的机器与网络：

分布式系统中的机器，配置不一样，其上运行的服务也可能由不同的语言、架构实现，因此处理能力也不一样；节点间通过网络连接，而不同网络运营商提供的网络的带宽、延时、丢包率又不一样。

第二，普遍的节点故障：

虽然单个节点的故障概率较低，但节点数目达到一定规模，出故障的概率就变高了。这就需要监控节点的状态，在节点故障的情况下将该节点负责的计算、存储任务转移到其他节点，使得其继续可用而不是宕机。

第三，不可靠的网络：

节点间通过网络通信，而网络是不可靠的。可能的网络问题包括：网络分割、延时、丢包、乱序。相比单机过程调用，网络通信最让人头疼的是超时：节点A向节点B发出请求，在约定的时间内没有收到节点B的响应，那么B是否处理了请求，这个是不确定的。这个不确定会带来诸多问题，最简单的，是否要重试请求、节点B会不会多次处理同一个请求。

3.请从一些开源分布式软件中找出能够体现透明性的样例代码，并解释是何种类型的透明性。

找的是`Hadoop`的部分源代码：

```
public Writable call(Writable param, ConnectionId remoteId)
    throws InterruptedException, IOException {
        Call call = new Call(param); //将传入的数据封装成call对象
        Connection connection = getConnection(remoteId, call); //获得一个连接
        connection.sendParam(call); // 向服务端发送call对象

  boolean interrupted = false;
  synchronized (call) {
   while (!call.done) {
    try {
     call.wait(); // 等待结果的返回，在Call类的callComplete()方法里有notify()方法用于唤醒线程
    }
       catch (InterruptedException ie) {
     // 因中断异常而终止，设置标志interrupted为true
     interrupted = true;
    }
   }
   if (interrupted) {
    Thread.currentThread().interrupt();
   }
   if (call.error != null) {
    if (call.error instanceof RemoteException) {
     call.error.fillInStackTrace();
     throw call.error;
    }
       else { // 本地异常
        throw wrapException(remoteId.getAddress(), call.error);
    }
   }
        else {
        return call.value; //返回结果数据
   }
  }
 }
```

```
private synchronized void setupIOstreams() throws InterruptedException {
  try {
   while (true) {
    setupConnection(); //建立连接
    InputStream inStream = NetUtils.getInputStream(socket); //获得输入流
    OutputStream outStream = NetUtils.getOutputStream(socket); //获得输出流
    writeRpcHeader(outStream);

    this.in = new DataInputStream(new BufferedInputStream
    (new PingInputStream(inStream))); //将输入流装饰成DataInputStream
    this.out = new DataOutputStream
    (new BufferedOutputStream(outStream)); //将输出流装饰成DataOutputStream
    writeHeader();
```

```
    // 更新活动时间

    touch();
    //当连接建立时，启动接受线程等待服务端传回数据，注意：Connection继承了Tread
    start();
    return;
   }
  }
  catch (IOException e) {
    markClosed(e);
    close();
  }
 }
```

上述代码体现了透明性的如下类型：访问、位置、迁移、并发、故障。