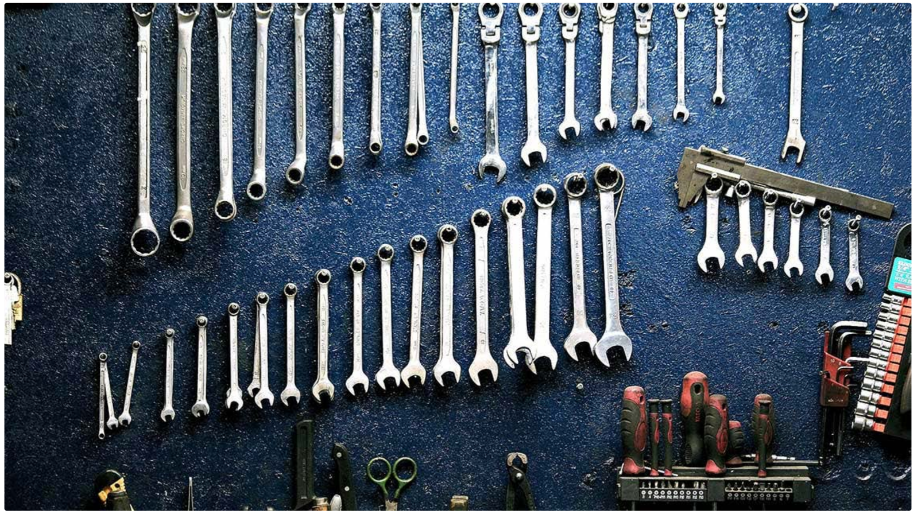
05 | 复杂度来源：高可用

2018-05-08 李运华



05 | 复杂度来源：高可用
李运华
- 00:00 / 12:21

今天，我们聊聊复杂度的第二个来源高可用。

参考维基百科，先来看看高可用的定义。

> 系统无中断地执行其功能的能力，代表系统的可用性程度，是进行系统设计时的准则之一。

这个定义的关键在于“**无中断**”，但恰好难点也在“无中断”上面，因为无论是单个硬件还是单个软件，都不可能做到无中断，硬件会出故障，软件会有bug；硬件会逐渐老化，软件会越来越复杂和庞大……

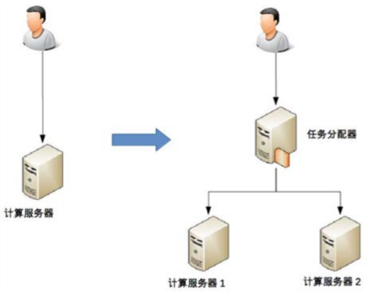
除了硬件和软件本质上无法做到“无中断”，外部环境导致的不可用更加不可避免、不受控制。例如，断电、水灾、地震，这些事故或者灾难也会导致系统不可用，而且影响程度更加严重，更加难以预测和规避。

所以，系统的高可用方案五花八门，但万变不离其宗，本质上都是通过“**冗余**”来实现高可用。通俗点来讲，就是一台机器不够就两台，两台不够就四台；一个机房可能断电，那就部署两个机房；一条通道可能故障，那就用两条，两条不够那就用三条（移动、电信、联通一起上）。高可用的“冗余”解决方案，单纯从形式上来看，和之前讲的高性能是一样的，都是通过增加更多机器来达到目的，但其实本质上是有根本区别的：**高性能增加机器目的在于“扩展”处理性能；高可用增加机器目的在于“冗余”处理单元。**

通过冗余增强了可用性，但同时也带来了复杂性，我会根据不同的应用场景逐一分析。

计算高可用

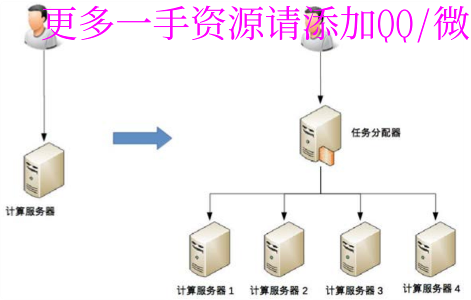
这里的“计算”指的是业务的逻辑处理。计算有一个特点就是**无论在哪台机器上进行计算，同样的算法和输入数据，产出的结果都是一样的**，所以将计算从一台机器迁移到另外一台机器，对业务并没有什么影响。既然如此，计算高可用的复杂度体现在哪里呢？我以最简单的单机变双机为例进行分析。先来看一个单机变双机的简单架构示意图。



你可能会发现，这个双机的架构图和上期“高性能”讲到的双机架构图是一样的，因此复杂度也是类似的，具体表现为：

- 需要增加一个任务分配器，选择合适的任务分配器也是一件复杂的事情，需要综合考虑性能、成本、可维护性、可用性等各方面因素。
- 任务分配器和真正的业务服务器之间有连接和交互，需要选择合适的连接方式，并且对连接进行管理。例如，连接建立、连接检测、连接中断后如何处理等。
- 任务分配器需要增加分配算法。例如，常见的双机算法有主备、主主，主备方案又可以细分为冷备、温备、热备。

上面这个示意图只是简单的双机架构，我们再看一个复杂一点的高可用集群架构。

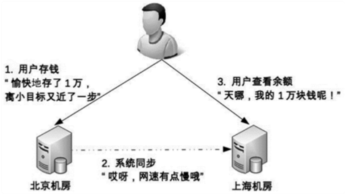


这个高可用集群相比双机来说，分配算法更加复杂，可以是1主3备、2主2备、3主1备、4主0备，具体应该采用哪种方式，需要结合实际业务需求来分析和判断，并不存在某种算法就一定优于另外的算法。例如，ZooKeeper采用的就是1主多备，而Memcached采用的就是全主0备。

存储高可用

对于需要存储数据的系统来说，整个系统的高可用设计关键点和难点就在于“存储高可用”。存储与计算相比，有一个本质上的区别：将数据从一台机器搬到到另一台机器，需要经过线路进行传输。线路传输的速度是毫秒级别，同一机房内部能够做到几毫秒；分布在不同地方的机房，传输耗时需要几十甚至上百毫秒。例如，从广州机房到北京机房，稳定情况下ping延时大约是50ms，不稳定情况下可能达到1s甚至更多。

虽然毫秒对于人来说几乎没有什么感觉，但是对于高可用系统来说，就是本质上的不同，这意味着整个系统在某个时间点上，数据肯定是不一致的。按照“数据+ 逻辑= 业务”这个公式来套的话，数据不一致，即使逻辑一致，最后的业务表现就不一样了。以最经典的银行储蓄业务为例，假设用户的数据存在北京机房，用户存入了1万块钱，然后他查询的时候被路由到了上海机房，北京机房的数据没有同步到上海机房，用户会发现他的余额并没有增加1万块。想象一下，此时用户肯定会背后一凉，马上会怀疑自己的钱被盗了，然后赶紧打客服电话投诉，甚至打110报警，即使最后发现只是因为传输延迟导致的问题，站在用户的角度来说，这个过程的体验肯定很不好。



除了物理上的传输速度限制，传输线路本身也存在可用性问题，传输线路可能中断、可能拥塞、可能异常（错包、丢包），并且传输线路的故障时间一般都特别长，短的十几分钟，长的几个小时都是可能的。例如，2015年支付宝因为光缆被挖断，业务影响超过4个小时；2016年中美海底光缆中断3小时等。在传输线路中断的情况下，就意味着存储无法进行同步，在这段时间内整个系统的数据是不一致的。

综合分析，无论是正常情况下的传输延迟，还是异常情况下的传输中断，都会导致系统的数据在某个时间点或者时间段是不一致的，而数据的不一致又会导致业务问题；但如果完全不做冗余，系统的整体高可用又无法保证，所以存储高可用的难点不在于如何备份数据，而在于如何减少或者规避数据不一致对业务造成的影响。

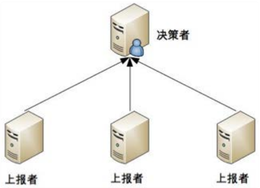
分布式领域里面有一个著名的CAP定理，从理论上论证了存储高可用的复杂度。也就是说，存储高可用不可能同时满足“一致性、可用性、分区容错性”，最多满足其中两个，这就要求我们在做架构设计时结合业务进行取舍。

高可用状态决策

无论是计算高可用还是存储高可用，其基础都是“状态决策”，即系统需要能够判断当前的状态是正常还是异常，如果出现了异常就要采取行动来保证高可用。如果状态决策本身都是有错误或者有偏差的，那么后续的任何行动和处理无论多么完美也都没有意义和价值。但在具体实践的过程中，恰好存在一个本质的矛盾：通过冗余来实现的高可用系统，状态决策本质上就不可能做到完全正确。下面我基于几种常见的决策方式进行详细分析。

1.独裁式

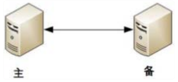
独裁式决策指的是存在一个独立的决策主体，我们姑且称它为“决策者”，负责收集信息然后进行决策；所有冗余的个体，我们姑且称它为“上报者”，都将状态信息发送给决策者。



独裁式的决策方式不会出现决策混乱的问题，因为只有一个决策者，但问题也正是在于只有一个决策者。当决策者本身故障时，整个系统就无法实现准确的状态决策。如果决策者本身又做一套状态决策，那就陷入一个递归的死循环了。

2.协商式

协商式决策指的是两个独立的个体通过交流信息，然后根据规则进行决策，最常用的协商式决策就是主备决策。

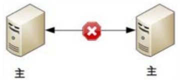


这个架构的基本协商规则可以设计成：

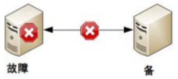
- 2台服务器启动时都是备机。
- 2台服务器建立连接。
- 2台服务器交换状态信息。
- 某1台服务器做出决策，成为主机；另一台服务器继续保持备机身份。

协商式决策的架构不复杂，规则也不复杂，其难点在于，如果两者的信息交换出现问题（比如主备连接中断），此时状态决策应该怎么做。

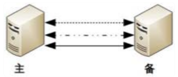
- 如果备机在连接中断的情况下认为主机故障，那么备机需要升级为主机，但实际上此时主机并没有故障，那么系统就出现了两个主机，这与设计初衷（1主1备）是不符合的。



- 如果备机在连接中断的情况下不认为主机故障，则此时如果主机真的发生故障，那么系统就没有主机了，这同样与设计初衷（1主1备）是不符合的。



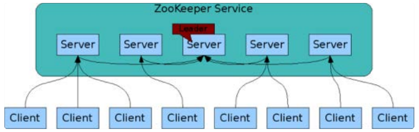
- 如果为了规避连接中断对状态决策带来的影响，可以增加更多的连接。例如，双连接、三连接。这样虽然能够降低连接中断对状态带来的影响（注意：只能降低，不能彻底解决），但同时又引入了这几条连接之间信息取舍的问题，即如果不同连接传递的信息不同，应该以哪个连接为准？实际上这也是一个无解的答案，无论以哪个连接为准，在特定场景下都可能存在问题。



综合分析，协商式状态决策在某些场景总是存在一些问题的。

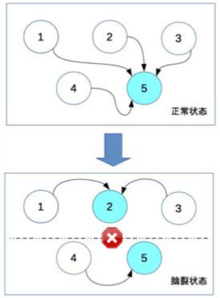
3.民主式

民主式决策指的是多个独立的个体通过投票的方式来进行状态决策。例如，ZooKeeper集群在选举leader时就是采用这种方式。



民主式决策和协商式决策比较类似，其基础都是独立的个体之间交换信息，每个个体做出自己的决策，然后按照“多数取胜”的规则来确定最终的状态。不同点在于民主式决策比协商式决策要复杂得多，ZooKeeper的选举算法Paxos，绝大部分人都看得云里雾里，更不用说用代码来实现这套算法了。

除了算法复杂，民主式决策还有一个固有的缺陷：脑裂。这个词来源于医学，指人体左右大脑半球的连接被切断后，左右脑因为无法交换信息，导致各自做出决策，然后身体受到两个大脑分别控制，会做出各种奇怪的动作。例如：当一个脑裂患者更衣时，他有时会一只手将裤子拉起，另一只手却将裤子往下脱。脑裂的根本原因是，原来统一的集群因为连接中断，造成了两个独立分隔的子集群，每个子集群单独进行选举，于是选出了2个主机，相当于人体有两个大脑了。



从图中可以看到，正常状态的时候，节点5作为主节点，其他节点作为备节点；当连接发生故障时，节点1、节点2、节点3形成了一个子集群，节点4、节点5形成了另外一个子集群，这两个子集群的连接已经中断，无法进行信息交换。按照民主决策的规则和算法，两个子集群分别选出了节点2和节点5作为主节点，此时整个系统就出现了两个主节点。这个状态违背了系统设计的初衷，两个主节点会各自做出自己的决策，整个系统的状态就混乱了。

为了解决脑裂问题，民主式决策的系统一般都采用“投票节点数必须超过系统总节点数一半”规则来处理。如图中那种情况，节点4和节点5形成的子集群总节点数只有2个，没有达到总节点数5个的一半，因此这个子集群不会进行选举。这种方式虽然解决了脑裂问题，但同时降低了系统整体的可用性，即如果系统不是因为脑裂问题导致投票节点数过少，而真的是因为节点故障（例如，节点1、节点2、节点3真的发生了故障），此时系统也不会选出主节点，整个系统就相当于宕机了，尽管此时还有节点4和节点5是正常的。

综合分析，无论采取什么样的方案，状态决策都不可能做到任何场景下都没有问题，但完全不做高可用方案又会产生更大的问题，如何选取适合系统的高可用方案，也是一个复杂的分析、判断和选择的过程。

小结

今天我给你讲了复杂度来源之一的高可用，分析了计算高可用和存储高可用两个场景，给出了几种高可用状态决策方式，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧。高性能和高可用是很多系统的核心复杂度，你认为哪个会更复杂一些？理由是什么？

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。（编辑乱入：精彩的留言有机会获得丰厚福利哦！）

公号-Java大后端

2018-05-08

今日心得

需求驱动驱动；而高可用与高性能，是架构设计中两个非常重要的决策因素。因此，面对不同业务系统的不同需求，对高可用与高性能也会有不同的决策结论，其实现的复杂度也各不相同。支付宝业务，对于可用性和性能就会有很高的要求，在可用性方面希望能提供7 * 24不间断服务，在高性能方面则希望能实时收付款；而对于一个学生管理系统，在可用性与性能方面就不一定要有多高的要求，比如晚上可关机，几秒内能查询到信息也可接受。为此，高可用性与高性能的复杂度讨论需要结合业务需求。

1 WHAT - 什么是可用性?
定义可用性，可以先定义什么是不可用。需要经历若干环节，网站的页面才能呈现在最终的用户面前；而其中的任何一个环节出现了故障，都可能会导致网站的页面不可访问，也就是出现了网站不可用的情况。昨夜iOS版本QQ出现大面积闪退就是一个系统不可用的典型案例。

我们可以利用百分比来对网站可用性进行度量：
网站不可用时间=完成故障修复的时间点 - 故障发现的时间点
网站年度可用时间=年度总时间 - 网站不可用时间
网站年度可用性=(网站年度可用时间/年度总时间) x 100%

举例：一些知名大型网站的可用性可达到99.99%（俗称4个9），我们可以算一下一年下来留给处理故障的时间有多少?
年度总时间=365*24*60=525600分钟
网站不可用时间=525600*(1-99.99%)=52.56分钟
也就是，如果网站要达到4个9的可用性，一年下来网站不可用时间最多53分钟（也就是不足1个小时）。

可见，高可用性就是技术实力的象征，高可用性就是竞争力。

2 WHY - 为什么会出现不可用?
硬件故障。网站多运行在普通的商用服务器，而这些服务器本身就不具备高可用性，再加之网站系统背后有数量众多服务器，那么一定时间内服务器宕机是大概率事件，直接导致部署在该服务器上的服务受影响。

软件BUG或网站更新升级发布。BUG不能消灭，只能减少；上线后的系统在运行过程中，难免会出现故障，而这些故障同样直接导致某些网站服务不可用；此外，网站更新升级发布也会引起相对较频繁的服务器宕机。

不可抗拒力。如地震、水灾、战争等。

3 HOW - 如何做到高可用
核心思想：网站高可用的主要技术手段是服务与数据的冗余备份与失效转移。同一服务组件部署在多台服务器上；数据存储在多台服务器上互相备份。通过上述技术手段，当任何一台服务器宕机或出现各种不可预期的问题时，就将相应的服务切换到其他可用的服务器上，不影响系统的整体可用性，也不会导致数据丢失。

从架构角度看可用性：当前网站系统多采用经典的分层模型，从上到下为：应用层、服务层与数据层。应用层主要实现业务逻辑处理；服务层提供可复用的服务；数据层负责数据读写；在部署架构上常采用应用和数据分离部署，应用会部署到不同服务器上，这些服务器被称为应用层的服务器；这些可复用的服务也会各自部署在不同服务器上，称为服务层的服务器；而各类数据库系统、文件柜等数据则部署在数据层的服务器。

硬件故障方面引起不可用的技术解决措施：(1)应用服务器。可通过负载均衡设备将多个应用服务器构建为集群对外提供服务（前提是这些服务需要设计为无状态，即应用服务器不保存业务的上下文信息，而仅根据每次请求提交的数据进行业务逻辑的操作响应），当均衡设备通过心跳检测手段检测到应用服务器不可用时，则将其从集群中移除，并将请求切换到其他可用的应用服务上。(2)服务层服务器。这些服务器被应用层通过分布式服务框架（如Dubbo）访问，分布式服务框架可在应用层客户端程序中实现软件负载均衡，并通过服务注册中心提供服务的服务器进行心跳检测，当发现有服务器不可用时，立即通知客户端程序修改服务列表，同时移除响应的服务器。(3)数据服务器。需要在数据写入时进行数据同步复制，将数据写入多台服务器上，实现数据冗余备份；当数据服务器宕机时，应用程序将访问切换到有备份数据的服务器上。

软件方面引起不可用的技术解决措施：通过软件开发过程进行质量保证。通过预发布验证、严格测试、灰度发布等手段，尽量减少上线服务的故障。

彡工鸟

2018-05-08

这么多回复里，没有人提到高可用和高性能的量化指标，没有这个指标前提下，无法断定哪个更复杂吧。打个比方，高可用两条99就行了，你觉得会复杂，会难么？高性能要求你在并发百万，千万级调用十几个服务前提下，仍能保持10多毫秒，你觉得简单？复杂与否还是要指标。另外，很多人都关注应用节点和硬件节点高可用，却忽略了业务高可用这个视角，系统全挂了，你人工接入业务，在后台帮用户开通，办理，对业务来说也是高可用吧。以上个人看法

作者回复

2018-05-08

你说的有道理，没有绝对的结论，我的问题只是想引起大家思考，通过思考来更深入理解复杂度。

通常情况下，高可用要复杂一些，因为需要考虑的情景很多，而且没有完美的方案，只能做取舍。

YMF_WX1981

2018-05-08

高可用相对复杂。

高性能，不管通过什么方式，或多或少，性能总获提高，行为上非必须做；高可用必须做，因为系统宕机或数据丢失时，谈高性能也无意义。

高可用涉及分布式存储和分布式计算，这两课题本身就复杂。

高可用涉及的非技术因素，如自然，政治。

So…

bieber

2018-05-25

高可用的解决方法不是解决，而是减少或者规避，而规避某个问题的时候，一般都会引发另一个问题，只是这个问题比之前的小，高可用的设计过程其实也是一个取舍的过程。这也就是为什么系统可用性永远只是说几个九，永远缺少那个一。
而高性能，这个基本上就是定义计算能力，可以通过架构的优化，算法的改进，硬件的升级都可以得到很好的解决，从而达到我们心里对性能的预期...

作者回复

2018-05-25

有道理，没有完美的高可用方案

罗烽

2018-05-08

高性能，高可用，哪个复杂度更高？
我认为高可用更复杂。性能方面，我们可已通过增加机器，拆分服务来提高性能。但是高可用这个不是通过单纯花钱（增加机器）能解决的，但还是必须要花钱����，相比较而言，它更需要一个良好的设计，这个就很复杂了。
关于高可用，我有些自己的想法
1，还是要做小的服务，小的服务稳定性会更高。
2，高可用的监控十分的重要，只有能先发现问题，才能接下来处理问题。
3， 存储高可用（减少和规避数据不一致），这个太复杂的不清楚，我们的业务现在没有那么复杂，数据库用的就是阿里云的主备rds，相比较而言，使用阿里云的服务会让我们的服务保障性更高些，这个只能想到这些

李志伟

2018-05-08

个人觉得根据场景而定，如果一个系统部署结构复杂，组件众多，数据量也很大。那么高可用性的代价就会比较高。因为高可用意味着冗余，
冗余也就意味着要有额外的策略来管理这些冗余的组件。另外大数据量数据服务冗余异地多活也是很有挑战性的。 于此相对如果一个系统他的业务复杂度很高，涉及到很多的复杂计算，但是本身部署结构不复杂，那么这时候高性能的复杂度就会比较大

公众号：歪脖贰点零

2018-05-08

为保证高可用，有时候会引入其他组件，比如keepalive等等，此时keepalive也易容易产生单点问题，于是做主从或其他方案。若其他方案同样存在单点问题，如此往复下去。悲观的看，似乎无止境，更多的时候是个取舍。

夜行观星

2018-05-13

就我一个人注意到ZK的选举算法不是Paxos吗？虽然不是本文重点��

作者回复

2018-05-13

感谢指正，ZK的协议是ZAB，官方文档也解释了ZAB不是Paxos算法，因为两者的设计目标不同，我没有深入研究两者协议，但大部分研究过的人认为ZAB是在Paxos算法上进行了改良和优化，有兴趣的可以深入研究一下。

高歌在羊城

2018-05-08

大神，希望后面多一些落地的案例分析，章节篇幅可以长一点，一次讲一个要点都行��

作者回复

2018-05-08

别急，后面很多案例和模式分析

---

mike

2018-05-08

高可用的关键字通过 “无间断”分析，它与高性能都是通过增加机器的方案来解决问题，两者本质区别在于目的不同，一个是通过 “扩展”处理性能，一个是 “冗余”处理单元。

实现方式，可以是 1 主 3 备、2 主 2 备、3 主 1 备、4 主 0 备，具体场景进行选择，高可用的难点在于如何减少和规避数据不一致对业务造成的影响。

方法论：独裁式，协商式，民主式。分析它们各自的优缺点，结合实际业务场景，选择合适的高可用方案。

如何选取适合系统的高可用方案，也是一个复杂的分析、判断和选择的过程。

水寒山冷

2018-05-08

高可用和高性能分开看其实难度都不大，难就难在有些场景需要二者兼顾。要解决高可用，数据层要做备份要做分区，一旦这么做就面临一致性问题，而在多个分区进行数据一致性同步本身就对高性能冲突，所以大部分设计都设计为最终一致性。毕竟不是每个公司都跟googe一样财大气粗可以把分布式协商的延迟降低到可以接受的级别。

Ivan

2018-05-08

高可用一般会考虑的更多一些，简单点说一个不可以的服务也就不存在性能一说，冗余是高可用的主要手段，高可用的主要复杂度体现在状态监控，服务切换或服务恢复上，为了降低其复杂度，又有无状态设计，熔断设计等等，这里面其实又牵扯到高性能，一个高性能的服务往往是快的小的独立的，相应的其高可用也就较容易实现。感觉最终的落地点还是在业务复杂度上，登录偏向高性能，支付偏向高可用

幸福时光

2018-05-08

架构的问题谈复杂性不如谈重要性来得直接，这个依赖于架构所要解决的业务场景的复杂度是对高性能有更高要求，还是对高可用有更高要求。如果对高性能的要求取舍大于高可用，自然高性能的架构考虑势必会复杂一些。大多数情况下，鱼和熊掌不可兼得，最终架构选择还是要依赖业务场景做出平衡。

---

zeus2_18921421203

2018-05-08

高可用主要难点是数据的高可用，而应用服务 机房 链路目前都比较成熟。

---

ncicheng

2018-05-08

可不可以这样理解，协商式用在只有主备两台机器的简单情况，当机器数量多于两个但少于五个时就采用独裁式，当机器很多时就采用民主式更好？谢谢

孙振超

2018-05-26

相对而言还是高可用更难些，按照作者说的高性能其实就是容量，在负载均衡系统高可用的情况下加机器就可以了，而想做到各个环节的高可用不是靠加机器就能搞定的，通常需要复杂的算法、引入更多的中间件、牺牲一定的性能才能实现，这其中还要进行各种权衡取舍裁剪才可以

作者回复

2018-05-27

确实如此

Joker

2018-05-25

高性能是为了达到一个量化的目标，通常我们会有各种不同的办法去实现，抛开消耗来说，方法有很多种，就像上篇讲到的，粗暴加机器，优雅划分等;但是高可用是为了规避一个非量化的抽象bug场景集合，这些不都是能提前预测到的，所以高可用一般来说都会比高性能复杂！

作者回复

2018-05-27

是的，通俗来讲，高性能是土豪，有钱可以任性；高可用是文豪，需要日积月累修炼��

dbo

2018-05-15

CAP理论的作者已经重新解释了很多人对CAP的误解，C A P 三者并不是互斥关系只能选其二，而是在出现网络分区时，可以选择0.9 + 0.9 + 0.7之类的措施，在三者间进行平衡取舍。

作者回复

2018-05-15

后面会详细阐述CAP

小思绪

2018-05-13

协商式中的双主也是脑裂吧?

作者回复

2018-05-13

表现有点类似，但主备一般不用脑裂，集群才用脑裂这个词

云辉

2018-05-09

高性能可预测，可牺牲，可取舍。而高可用要考虑各种情况，而且是有状态的，对于数据不一致的恢复是一场噩梦。相对来说，还是高可用难一点，高性能在极端情况下也难的。

itperson

2018-05-08

高可用更复杂一些，因为需要考虑很多的异常处理方式。

syldjzl

2018-05-08

醍醐灌顶，讲的非常好

艳姐

2018-07-04

如何做到存储高可用没有讲

作者回复

2018-07-05

后面有很多章节讲各种架构模式

milu

2018-07-01

没有什么是加机子解决不了性能。。如果有。。一定是机子不够

作者回复

2018-07-02

有钱也不能这么任性啊��

DavidTsai

2018-06-28

高可用主要解决的是应对高性能分布式不同机器下最终输出结果的一致问题，同时需要决策如何保证机器之间信息互通正常而做出选择

Cola

2018-06-16

对于普通公司来说，高可用远比高性能复杂的多。现代软件系统由于硬件设备性能大幅提升与一系列成熟稳定的云服务/解决方案，要达到用户层面的高性能其实不难，甚至通过单纯的砸钱于设备质量与数量也能达到普义上的高性能。但是高可用却要考虑一系列负载均衡，多进程集群(无状态设计)，服务治理(熔断设计，失效转移，快速恢复)，监控(状态报警，自动扩展，黑名单)，缓存与存储(数据的一致性，脏数据的处理，冷热备份)，这是一个庞大的技术栈。但是当用户量级达到另一个层面，高性能难度也成指数性增长，这时候这两个命题你中有我我中有你，两者是密不可分的。

作者回复

2018-06-19

是的，高可用异常场景很多

水月洞天

2018-06-12

高性能解决的是计算的速度和计算量问题，高可用解决异常带来的不可控情况。后者不仅要考虑计算的高可用，还要考虑传输的高可用，存储的高可用，考虑的场景更多。

我走我流

2018-06-11

实际项目中独裁式，协商式，民主投票式都用过。。。

作者回复

2018-06-13

哪种效果比较好?

Geek_8242cb

2018-05-29

老师，银行账务类强一致性业务，适用最终一致性方案吗？我们通常要求既要实时看到账务操作结果，又要提供高性能，最终只能用依赖于数据库实现一致性，但性能压力很大

作者回复

2018-05-30

强一致性目前没有太好的方式，目前一般采取用户分区的做法，即：将用户分散在多个数据分区中，每个数据分区中的用户用单点数据库保证强一致性

唐朝首都

2018-05-29

总体而言高可用更复杂一点，因为对于高可用来说，只要我们能够找到服务的性能瓶颈，针对性做出优化，基本上都能解决；但是对于高可用来说，基本是无解的，我们能做的基本上是做选择与取舍，找出更符合我们业务场景的方案。

Tony

2018-05-28

高可用复杂
先从需求取舍来讲，高可用绝大部分情况下都必须做，数据对企业来说是非常重要的
数据+逻辑=业务，没了数据，业务也会瘫痪
通篇文章介绍了高可用的核心方案是通过冗余方案实现，但是没有一个完美的冗余方案能做到100%保证高可用

而反观上一篇的高性能，单机可以通过垂直方案提升服务性能，水平方向可以通过增加服务器扩展性能，尽管性能提高有个逼近值

相比于高可用，高性能实现要容易一些

missa

2018-05-22

高性能，高可用。
高可用会比较复杂，性能这块随意现在硬件的发展，成本越来越低价了，大多数公司也不会专门的对机器进行性能的调优。
高可用需要考虑的存储高可用，和状态高可用就比较复杂多样的场景了。

作者回复

2018-05-22

是的，分布式状态一致性很难，异常场景太多

陈天境

2018-05-17

对于域名，有没有高可用而言呢？比如我们最近域名由于某些原因被撤销备案导致访问不了，而重新备案时间又很长。

作者回复

2018-05-17

被撤销备案这个就不是技术能解决的，如果真要防备这种情况，申请两个或更多域名，但这样做不利于用户认知和SEO推广

日光倾城

2018-05-15

我觉得满足高可用更复杂，因为影响高可用的因素更多也更难以把控，

作者回复

2018-05-16

是的，细节很多，有的互相冲突，需要判断取舍

J

2018-05-15

我认为高可用更复杂一点 高性能和高可用底层上除了设计节点通信外，还多了复杂的选举算法，如paxos

云学

2018-05-15

有些人把高可用与高可靠混淆了，高可用是不要中断服务，高可靠是数据不丢失。

作者回复

2018-05-15

有区别，但实践中一般很难清晰的区分，否则每次都要解释半天，我们一般都是混用，大家都明白是什么意思。

严格来说，高可用是指正常提供服务的概率，主要和故障恢复时间有关；高可靠是指出问题的概率，主要和故障次数有关。大部分情况下其实我们都是说可用性，因为保证系统能够正常提供服务才是我们的首要目标。

---

focus　根

2018-05-15

自己虽然做Android开发，但是也对这个架构的处理产生兴趣 应该如何实践提高自己对架构设计的能力

作者回复

2018-05-15

app的架构主要体现在可扩展性，后面会讲

王磊

2018-05-14

如上的回复'一般不建议集群跨机房，性能太低，你说的这种场景一般不会用异地多活'，是会用还是不会用，如果不会用，那用什么?

作者回复

2018-05-14

单机房高可用+数据分区，后面会讲到

王磊

2018-05-14

关于高可用状态决策，处在一个集群的节点也可能是异地，如北京，上海，深圳，对吗?否则，一个用户在北京取钱后，余额为0，在数据同步到深圳之前，理论上又在深圳取钱，这种情况因为有集群的leader
，以leader的状态为准，来避免这种情况。

作者回复

2018-05-14

一般不建议集群跨机房，性能太低，你说的这种场景一般不会用异地多活

赵志强

2018-05-13

我觉得大多情况不能把高性能和高可用割裂来讲，甚至在很多场合也不能分开，就如同作者所举例，要保证双十一海量并发访问，秒杀某件商品，即需要高性能（即不能卡机），也需要高可用（数据的一致性），很难想象缺失一个后，长时间打不开页面或页面打开很快，但标的显示数量和实际不符。至于比较哪个更复杂，还要看投入多少成本，增加机器就能提高可用性我觉得一点也不复杂。

作者回复

2018-05-14

分开讲不是说每个系统都只能有一个复杂度，而是为了条理性

石同享

2018-05-13

老师想请教个问题。分布式环境下数据同步是常遇到的问题，分布式环境下达到强一致很困难，所以一般考虑最终一致性。绝大部分场景最终一致是可以接受的。但是像支付，交易这种场景最终一致性对用户来说也是不可能忍受的，用户期待的是强一致。 您觉得应该如何去权衡这个最终一致性和用户期待的强一致? 另外最终一致性这块什么时候会讲呀。感谢(数据一致应该也可理解是高可用的一部分吧)～

作者回复

2018-05-13

后面异地多活章节会讲

杰

2018-05-12

这课程再贵10倍我也买，把架构讲的如此通俗易懂，膜拜一下大神!

作者回复

2018-05-12

有钱任性么？��
学到技术最重要，如果觉得超值，可以让更多人来学习��

晴天

2018-05-12

看到现在综合前几章的内容，打破了我对设计的一些固有看法以及相当然的观点，可用性是软件复杂度的来源之一，而并没有银弹，也无一劳永逸的办法，集群增加以及优化策略都是折中主义，所以很符合开篇之词，设计是解决一定程度上复杂度带来的问题。

---

fxp

2018-05-11

解决高性能问题的必经之路就是高可用。

作者回复

2018-05-11

如果你是说要先做到高可用才能做到高性能，那这个说法是不成立的；

如果你说高性能设计涉及到高可用的取舍，那是有道理的，例如mysql的高性能写入是以牺牲一定高可用为代价的

追寻云的痕迹

2018-05-11

化繁为简，通过简单的举例，就把核心概念说说清楚了。架构师的作用就在于各种场景中做出合适的取舍，而不是像普通程序员那样陷入Paxos还是Raft好的争论中。PowerShell之父Jeffery Snover有一句话叫，To ship is to choose，讲的就是取舍的问题。

作者回复

2018-05-11

你已经剔透了��

衣申人

2018-05-10

状态决策属于一致性的范畴？这一节不是谈可用性吗？

---

feifei

2018-05-10

高性能，这个架构的设计更为复杂，更加不可控制

作者回复

2018-05-11

通常情况下高可用复杂一些，异常场景很多

王旭东

2018-05-10

个人认为正常场景下，高可用比高性能更复杂也更重要，复杂在于高可用无法100%保证CAP。
重要在于高性能的前提是系统可用。其实两个都重要和复杂，都需要我们好好权衡������

作者回复

2018-05-10

确实如此

波波安

2018-05-10

从带来问题的严重性来看，一般高可用出现的问题比高性能方面出现的问题更严重。

Seven_dong

2018-05-10

状态是分布式系统的噩梦，然而一个系统又不可能没有状态

作者回复

2018-05-10

还是有无状态的系统呢，nginx负载均衡，redis缓存等

任锋

2018-05-09

您好 华仔，首先我觉得高可用是比高性能更复杂，也看了你的回复，我已确认，我们的架构就是属于独裁模式，但是这还是一方面的弊端，如果链接数超了怎么办呢？双主应该可以解决，但是很少听说数据库双主架构？

作者回复

2018-05-10

数据库一般不用双主，用分库分表

印宏宇

2018-05-09

高可用：

不中断的提供功能执行。

按场景来说分为计算和存储高可用。

计算高可用：

需要一个分配器

分配器需要管理与计算资源的连接

需要根据场景制定不同的分配算法（1主3备，2主2备，3主1备，4主0备）

存储高可用：

需要考虑传输的延迟和中断，分布式领域CAP理论，存储高可用不可同时满足"一致性，可用性，分区容错性"，只能最多满足两种情况。

状态决策是高可用的基础：

独裁，独裁本身就不满足高可用。

协商，当备机和主机连接中断，备机不知道主机当前是否活着，那么备机是否升级到主机，如果引入多个连接，那么多个连接信息传递如果不一致，导致了更多的复杂性。

民主，少数服从多数，但如果分成了两个部分"脑裂"，会选出两个首脑，所以一般情况下民主必须是半数以上。

对于高性能和高可用谁更复杂，还是要看选择的指标和面对的业务场景。

如果每秒处理10000的订单和保持99.999%高可用性都非常复杂。高可用还是要更复杂一些，因为没有完美的解决方案，只能是取舍。

有渔@蔡

2018-05-09

高性能，高可用，采用的语言不同，可收到事半功倍的效果。比如nginx+lua的openresty，性能杠杠的。elixir写的高并发，几年都不会挂。

有渔@蔡

2018-05-09

作者能专门讲下你们实际项目中的异地多活解决方案吗好?

作者回复

2018-05-10

后面有具体的异地多活，你也可以搜索互联网上的案例

Geek_d8f635

2018-05-09

区块链技术如果越来越成熟，是不是对高性能有很大帮助?

作者回复

2018-05-10

据我目前对区块链的理解来看，区块链恰恰是性能低下的实现方案，不但没有帮助，还会存在明显的性能问题

快乐的小傻子

2018-05-09

高可用性是不是应该为高性能的基础呢？服务性能高，可用性差，这个用户体验和公司业务都不易接受吧

作者回复

2018-05-11

有一定关联，但不是基础和上层的关系

郭峰

2018-05-09

看目录里好像没有针对一致性问题专门讲解的章节，会在后面高可用或者CAP那里讲解吗?

作者回复

2018-05-09

会的，到时候交流

海洋

2018-05-09

高可用和高性能哪个更复杂？本身就是一个前提模糊的复杂问句，要看业务类型，业务需求和实际系统的瓶颈。例如我就把zk作为单点运行，因为追求性能，另外实践证明zk单点比zk集群更稳定可靠！

带刺的温柔

2018-05-09

刚上完线有点小兴奋看完老师讲解结合自己遇到的项目我觉得毫无疑问性能与高可用都重要。只是在业务发展的不同时期需要有不同的选择。比如说前期我觉得主要是业务开发性能与高可用可以暂时放一放，中期随着业务的发展量不断增加那么在性能上就要多花些功夫，随着业务不断的发展这个钱包也鼓起来了那这个时候两手都要抓而且要硬。我的问题是老师帮我点评一下有说错的没

作者回复

2018-05-09

后面架构设计原则会解答你的问题

---

fiseasky

2018-05-08

数据库冗余，如何做到数据的一致性呢？就是数据如何同步过来的。我现在的项目是自己写一个定时工具来同步，但是时效性比较差。

Hesher

2018-05-08

业务规模不大的情况下，高可用更加复杂。拆分业务，分库分表拆服务，加机器加实例，加缓存，等等优化手段都可以提升性能。而高可用则需要仔细考虑，根据业务决定使用AP方案还是CP方案，比如绝大多数互联网应用都是保证AP，而银行、证券等系统，数据一致性要求肯定会更高，优先保证CP。

老王

2018-05-08

我还在可用的路上，还没到性能呢..

Smile

2018-05-08

看各位大神的留言 也是一种提升。。。

Kevin

2018-05-08

希望多举例子从例子上分析引入

作者回复

2018-05-09

如果不提炼一些原则和理念，单纯讲例子，难以讲述核心本质，这也是很多朋友看了几十上百个架构案例，每个好像都看懂了，但让自己去设计架构的时候，还是一筹莫展

卡莫拉内西

2018-05-08

请问ap类型的分布式协调服务属于哪种方式呢?在高可用的场景下是否比cp类型的zk更合适，例如eureka。
我觉得高可用的设计应该比高性能更复杂，高可用是普遍互联网软件系统的基础特性（特殊需求除外），设计好高性能大多是基于高可用的，高可用都没有，扯高性能那就是空中楼阁，不知道我的想法对不对

曹铮

2018-05-08

有个问题要和大神探讨，文中说脑裂是靠投票解决的，逻辑上没错。但一般脑裂这名词更多指发生分区后，又选出一个主，原主又不自知。这种情况下要解决的是分区时，原主绝不能响应事务处理，分区恢复后原主要主动切换成从的问题
在raft中这是依靠写事务要多数节点响应，以及心跳信息中带term来解决的
原谅我抠字眼

作者回复

2018-05-09

脑裂发生时，两个主都不知道，因此原主不能按照你说的方式不响应

武坤

2018-05-08

老师好，今天讲得计算高可用，和上次讲得“任务分配”两者是不是可以重合，同样的架构，即使高可用，也是高性能?
还有一个问题，上次讲得高性能方式“任务分解”，拆分后的每个子任务也得进行计算高可用，用现在讲的高可用方式合理吗?

作者回复

2018-05-08

看起来一样，但本质不一样，高性能的集群中机器是对等的，高可用集群中机器关系是变化多样的。

高歌在羊城

2018-05-08

高可用比高性能的需求更加强烈，服务可用的前提下才讲高性能才有实际意义。性能开销一般在网络io，数据库读写上，可通过合理的服务化水平扩展进行提升，分布式缓存，内存数据库等，高

可用意味着所有的组件都要冗余，要求在飞速运行中进行升级修理，实时监控到系统组件的运行状态，快速修复，没有百分百的高可用系统存在。

山人

2018-05-08

我觉得高可用和高性能的复杂度比较没有绝对的答案，依据具体的指标来定，比如我们不能说小时级的高可用比上千并发毫秒级的性能要求的复杂度要高，而且还要结合具体的业务场景具体问题具体分析

孙晓明

2018-05-08

不能笼统的说高性能还是高可用的复杂度高，需要具体分析场景。例如，高性能的复杂度中，如果使用任务分解实现高可用，不仅需要考虑集群相应复杂度的问题，还要考虑软件代码如何实现的复杂度问题。高可用的复杂度中，对数据高可用的实现，需要考虑数据一致性如何实现等问题。

野马

2018-05-08

高可用更复杂，高可用需要良好的设计。高性能可以通过增加机器达到高性能。

大光头

2018-05-08

相对而言，高可用会更加复杂些，因为高可用意味着要增加存储资源，即数据冗余，数据冗余就会导致一致性问题。而高性能，一般可以通过增加计算资源实现

彦龍

2018-05-08

个人理解：两者都很重要，根据业务场景不同，侧重点不同，高可用需求场景会更多一些。两者有时是相互关联的，比如为了实现数据处理的速度，使用分布式集群处理，这同样增加了整个系统的可用性；为了实现web系统的可用性，使用lbs，这也间接增强了整个系统处理性能。

成功

2018-05-08

高性能对系统非故障状态下的客户感知更直观，高可用对系统故障下的容灾性和可用性起决定作用，高可用在目前的技术体系中更难实现。

李志博

2018-05-08

高性能吧，不同系统业务场景都不一样，需要从设计，技术选型，架构方面好好取设计，才有可能实现高性能，但是高可用感觉和业务关系不大，不考虑一些极端场景，市面上的方案都够用了

Forrest

2018-05-08

我认为是高可用更复杂，因为难点在于数据的一致性。

淡彩

2018-05-08

我认为高可用更复杂一些。正如上一节课提到的高性能可以通过简单粗暴的增加计算节点(程序已经优化到一定程度的前提)去提升，那么高可用对于高性能的计算节点来说就是为了保证高性能的计算节点能够可靠的使用，当然还包括这节提到的存储的高可用，如果没有了高可用，那么高性能也必然无法得到正常的发挥。对于高可用的多种方案，都没有非常完美的方案，在异常情况下这就得通过人工干预了，这里需要运维监控等一系列保障措施。以上是我的观点，若有不妥之处，望指正。

雪

2018-05-08

高可用更加复杂些
1.没有绝对的高可用因为无法满足CAP
定理
2.高性能可以通过拆分系统和提高硬件性能来完成目前为止软件对性能的要求还可以通过各种方式解决

LEON

2018-05-08

您好，我有一个问题，文中"通过冗余来实现的高可用系统，状态决策本质上就不可能做到完全正确。"这部分不是很理解。主备决策中，如果两台设备的心跳线为多条的话。只要有一个心跳线信号正常在就可以维持主备模式。理论上没有问题，可以做到状态决策完全正确，但是不排除是两台设备的心跳进程全部故障。所以说通过冗余来实现的高可用系统，状态决策本质上就不可能做到完全正确。这样理解是否正确?

作者回复

2018-05-08

是的，心跳线也会故障，多条心跳线还会出现信号不一致的问题

narry

2018-05-08

个人认为如果将两者进行比较，高可用带来的挑战更大，高性能可以依靠扩展来应对，根据AKF扩展立方理论，3个方向上的扩展都涉及到了高可用的问题，X方向与其它两个方向的结合就会带来计算（XY）和数据一致性(XZ)高可用问题,所以解决高可用是解决扩展问题的关键技术挑战，挑战更高

@漆~心endless

2018-05-08

单从考虑的方面来说，高可用需要注意的点比较多，但也不能单纯的说哪一种更复杂，需要根据业务来判断。