27 | 如何设计计算高可用架构?

2018-06-28 李运华

27 | 如何设计计算高可用架构?
朗读人：黄洲君　10′52″ | 4.98M

计算高可用的主要设计目标是当出现部分硬件损坏时，计算任务能够继续正常运行。因此计算高可用的本质是通过冗余来规避部分故障的风险，单台服务器是无论如何都达不到这个目标的。所以计算高可用的设计思想很简单：通过增加更多服务器来达到计算高可用。

计算高可用架构的设计复杂度主要体现在任务管理方面，即当任务在某台服务器上执行失败后，如何将任务重新分配到新的服务器进行执行。因此，计算高可用架构设计的关键点有下面两点。

1. 哪些服务器可以执行任务

第一种方式和计算高性能中的集群类似，每个服务器都可以执行任务。例如，常见的访问网站的某个页面。

第二种方式和存储高可用中的集群类似，只有特定服务器（通常叫“主机”）可以执行任务。当执行任务的服务器故障后，系统需要挑选新的服务器来执行任务。例如，ZooKeeper 的 Leader 才能处理写操作请求。

2. 任务如何重新执行

第一种策略是对于已经分配的任务即使执行失败也不做任何处理，系统只需要保证新的任务能够分配到其他非故障服务器上执行即可。

第二种策略是设计一个任务管理器来管理需要执行的计算任务，服务器执行完任务后，需要向任务管理器反馈任务执行结果，任务管理器根据任务执行结果来决定是否需要将任务重新分配到另外的服务器上执行。

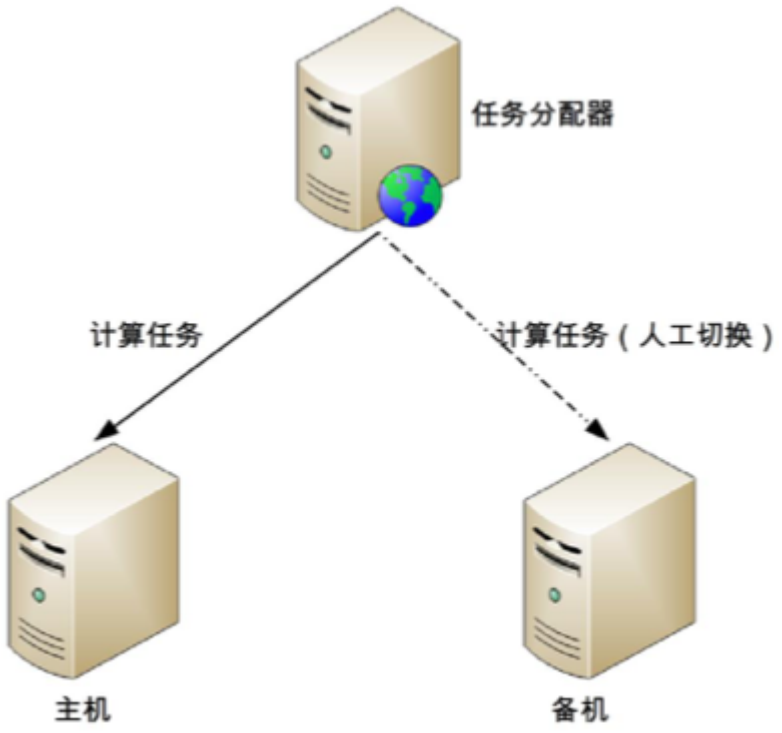
需要注意的是：“任务分配器”是一个逻辑的概念，并不一定要求系统存在一个独立的任务分配器模块。例如：

- Nginx 将页面请求发送给 Web 服务器，而 CSS/JS 等静态文件直接读取本地缓存。这里的 Nginx 角色是反向代理系统，但是承担了任务分配器的职责，而不需要 Nginx 做反向代理，后面再来一个任务分配器。
- 对于一些后台批量运算的任务，可以设计一个独立的任务分配系统来管理这些批处理任务的执行和分配。
- ZooKeeper 中的 Follower 节点，当接收到写请求时会将请求转发给 Leader 节点处理，当接收到读请求时就自己处理，这里的 Follower 就相当于一个逻辑上的任务分配器。

接下来，我将详细阐述常见的计算高可用架构：主备、主从和集群。

主备

主备架构是计算高可用最简单的架构，和存储高可用的主备复制架构类似，但是要更简单一些，因为计算高可用的主备架构无须数据复制，其基本的架构示意图如下：



主备方案的详细设计：

- 主机执行所有计算任务。例如，读写数据、执行操作等。
- 当主机故障（例如，主机宕机）时，任务分配器不会自动将计算任务发送给备机，此时系统处于不可用状态。
- 如果主机能够恢复（不管是人工恢复还是自动恢复），任务分配器继续将任务发送给主机。
- 如果主机不能够恢复（例如，机器硬盘损坏，短时间内无法恢复），则需要人工操作，将备机升为主机，然后让任务分配器将任务发送给新的主机（即原来的备机）；同时，为了继续保持主备架构，需要人工增加新的机器作为备机。

根据备机状态的不同，主备架构又可以细分为冷备架构和温备架构。

冷备：备机上的程序包和配置文件都准备好，但备机上的业务系统没有启动（注意：备机的服务器是启动的），主机故障后，需要人工手工将备机的业务系统启动，并将任务分配器的任务请求切换发送给备机。

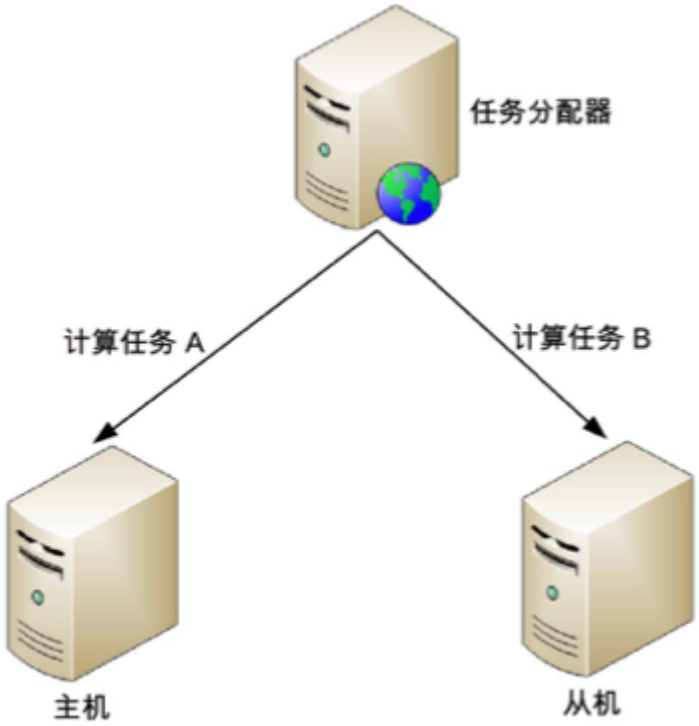
温备：备机上的业务系统已经启动，只是不对外提供服务，主机故障后，人工只需要将任务分配器的任务请求切换发送到备机即可。冷备可以节省一定的能源，但温备能够大大减少手工操作时间，因此一般情况下推荐用温备的方式。

主备架构的优点就是简单，主备机之间不需要进行交互，状态判断和切换操作由人工执行，系统实现很简单。而缺点正好也体现在“人工操作”这点上，因为人工操作的时间不可控，可能系统已经发生问题了，但维护人员还没发现，等了 1 个小时才发现。发现后人工切换的操作效率也比较低，可能需要半个小时才完成切换操作，而且手工操作过程中容易出错。例如，修改配置文件改错了、启动了错误的程序等。

和存储高可用中的主备复制架构类似，计算高可用的主备架构也比较适合与内部管理系统、后台管理系统这类使用人数不多、使用频率不高的业务，不太适合在线的业务。

主从

和存储高可用中的主从复制架构类似，计算高可用的主从架构中的从机也是要执行任务的。任务分配器需要将任务进行分类，确定哪些任务可以发送给主机执行，哪些任务可以发送给备机执行，其基本的架构示意图如下：



主从方案详细设计：

- 正常情况下，主机执行部分计算任务（如图中的“计算任务 A”），备机执行部分计算任务（如图中的“计算任务 B”）。
- 当主机故障（例如，主机宕机）时，任务分配器不会自动将原本发送给主机的任务发送给从机，而是继续发送给主机，不管这些任务执行是否成功。
- 如果主机能够恢复（不管是人工恢复还是自动恢复），任务分配器继续按照原有的设计策略分配任务，即计算任务 A 发送给主机，计算任务 B 发送给从机。
- 如果主机不能够恢复（例如，机器硬盘损坏，短时间内无法恢复），则需要人工操作，将原来的从机升级为主机（一般只是修改配置即可），增加新的机器作为从机，新的从机准备就绪后，任务分配器继续按照原有的设计策略分配任务。

主从架构与主备架构相比，优缺点有：

- 优点：主从架构的从机也执行任务，发挥了从机的硬件性能。
- 缺点：主从架构需要将任务分类，任务分配器会复杂一些。

集群

主备架构和主从架构通过冗余一台服务器来提升可用性，且需要人工来切换主备或者主从。这样的架构虽然简单，但存在一个主要的问题：人工操作效率低、容易出错、不能及时处理故障。因

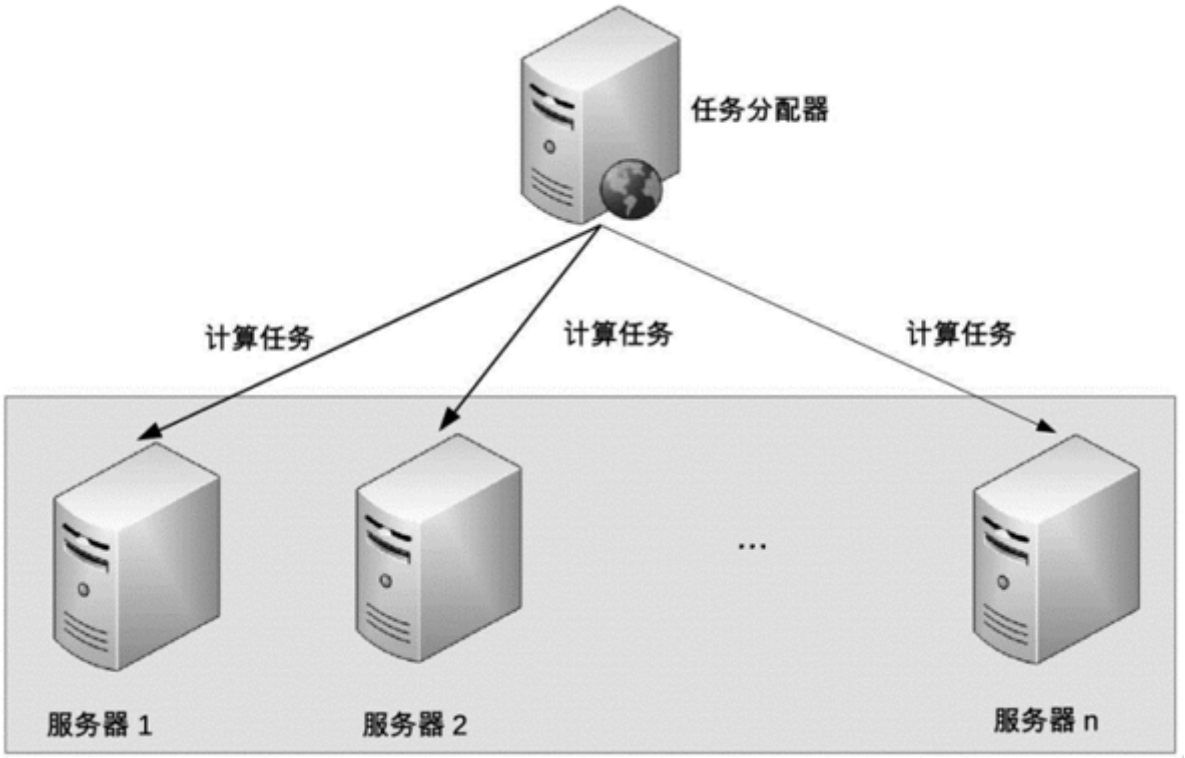
此在可用性要求更加严格的场景中，我们需要系统能够自动完成切换操作，这就是高可用集群方案。

高可用计算的集群方案根据集群中服务器节点角色的不同，可以分为两类：一类是对称集群，即集群中每个服务器的角色都是一样的，都可以执行所有任务；另一类是非对称集群，集群中的服务器分为多个不同的角色，不同的角色执行不同的任务，例如最常见的 Master-Slave 角色。

需要注意的是，计算高可用集群包含 2 台服务器的集群，这点和存储高可用集群不太一样。存储高可用集群把双机架构和集群架构进行了区分；而在计算高可用集群架构中，2 台服务器的集群和多台服务器的集群，在设计上没有本质区别，因此不需要进行区分。

对称集群

对称集群更通俗的叫法是负载均衡集群，因此接下来我使用“负载均衡集群”这个通俗的说法，架构示意图如下：



负载均衡集群详细设计：

- 正常情况下，任务分配器采取某种策略（随机、轮询等）将计算任务分配给集群中的不同服务器。
- 当集群中的某台服务器故障后，任务分配器不再将任务分配给它，而是将任务分配给其他服务器执行。
- 当故障的服务器恢复后，任务分配器重新将任务分配给它执行。

负载均衡集群的设计关键点在于两点：

- 任务分配器需要选取分配策略。
- 任务分配器需要检测服务器状态。

任务分配策略比较简单，轮询和随机基本就够了。状态检测稍微复杂一些，既要检测服务器的状态，例如服务器是否宕机、网络是否正常等；同时还要检测任务的执行状态，例如任务是否卡死、是否执行时间过长等。常用的做法是任务分配器和服务器之间通过心跳来传递信息，包括服务器信息和任务信息，然后根据实际情况来确定状态判断条件。

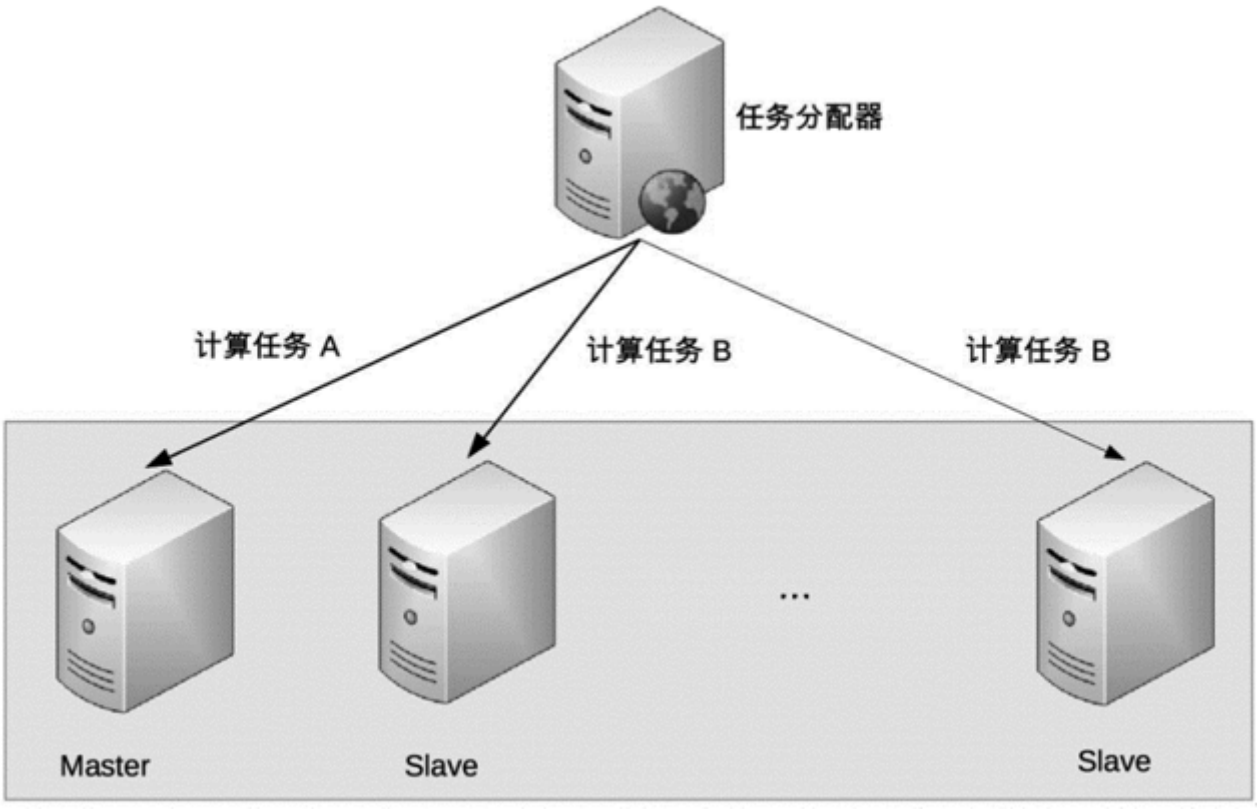
例如，一个在线页面访问系统，正常情况下页面平均会在 500 毫秒内返回，那么状态判断条件可以设计为：1 分钟内响应时间超过 1 秒（包括超时）的页面数量占了 80% 时，就认为服务器有故障。

例如，一个后台统计任务系统，正常情况下任务会在 5 分钟内执行完成，那么状态判断条件可以设计为：单个任务执行时间超过 10 分钟还没有结束，就认为服务器有故障。

通过上面两个案例可以看出，不同业务场景的状态判断条件差异很大，实际设计时要根据业务需求来进行设计和调优。

非对称集群

非对称集群中不同服务器的角色是不同的，不同角色的服务器承担不同的职责。以 Master-Slave 为例，部分任务是 Master 服务器才能执行，部分任务是 Slave 服务器才能执行。非对称集群的基本架构示意图如下：



非对称集群架构详细设计：

- 集群会通过某种方式来区分不同服务器的角色。例如，通过 ZAB 算法选举，或者简单地取当前存活服务器中节点 ID 最小的服务器作为 Master 服务器。
- 任务分配器将不同任务发送给不同服务器。例如，图中的计算任务 A 发送给 Master 服务器，计算任务 B 发送给 Slave 服务器。
- 当指定类型的服务器故障时，需要重新分配角色。例如，Master 服务器故障后，需要将剩余的 Slave 服务器中的一个重新指定为 Master 服务器；如果是 Slave 服务器故障，则并不需要重新分配角色，只需要将故障服务器从集群剔除即可。

非对称集群相比负载均衡集群，设计复杂度主要体现在两个方面：

- 任务分配策略更加复杂：需要将任务划分为不同类型并分配给不同角色的集群节点。
- 角色分配策略实现比较复杂：例如，可能需要使用 ZAB、Raft 这类复杂的算法来实现 Leader 的选举。

我以 ZooKeeper 为例：

- 任务分配器：ZooKeeper 中不存在独立的任务分配器节点，每个 Server 都是任务分配器，Follower 收到请求后会进行判断，如果是写请求就转发给 Leader，如果是读请求就自己处理。
- 角色指定：ZooKeeper 通过 ZAB 算法来选举 Leader，当 Leader 故障后，所有的 Follower 节点会暂停读写操作，开始进行选举，直到新的 Leader 选举出来后才继续对 Client 提供服务。

小结

今天我为你讲了几种常见的计算高可用架构，并分析了不同方案的详细设计，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，计算高可用架构从形式上和存储高可用架构看上去几乎一样，它们的复杂度是一样的么？谈谈你的理解。

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！）

精选留言

feifei 13

计算高可用架构，主要解决当单点发生故障后，原本发送到故障节点的任务，任务如何分发给非故障节点，根据业务特点选择分发和重试机制即可，不存在数据一致性问题，只需要保证任务计算完成即可

存储高可用架构，解决的问题是当单点发生故障了，任务如何分发给其他非故障节点，以及如何保障数据的一致性问题。

所以存储的高可用比计算的高可用的设计更为复杂。

2018-06-28

作者回复

分析正确👍

2018-06-28

李同杰 4

存储高可用需要考虑数据复制的问题，复杂度高于计算高可用架构。

2018-06-28

作者回复

👍👍👍

2018-06-28

oddrock 4

存储高可用比计算高可用要复杂的多，存储高可用是有状态的，计算高可用一般解决的都是无状态问题，有状态就存在着如何保存状态、同步状态的问题了

2018-06-28

作者回复

分析正确

2018-06-28

yungoo 3

计算高可用系统需考虑safety和liveness，而存储高可用除了需考虑safety和liveness，还受CAP约束

2018-06-28

作者回复

你已经融会贯通👍

2018-06-28

成功 1

存储任务要考虑CAP三个方面，肯定比计算任务复杂

2018-07-01

Johnny.Z 1

任务分配器挂了是不是高可用计算就没办法保证了，任务分配器是否也要弄成集群呢？

2018-06-28

作者回复

是的，全流程的高可用要求任务分配器也高可用

2018-06-28

空档滑行 1

计算高可用复杂在选择算法，随着集群规模的扩大，复杂性增加的不明显。比如负载均衡如何判断节点可用，如何判断任务失败还是只是时间较长。

存储高可用除了面临计算高可用同样的问题在，还要考虑数据的同步，异地备灾也比计算高可用复杂，而且随着集群数量增加，整个策略都要做相应的改变

2018-06-28

作者回复

分析正确👍

2018-06-28

星火燎原 1

存储高可用架构的复杂度在于节点数据的一致性上，计算高可用架构复杂度在于主从节点的选举上 不知对不对

2018-06-28

作者回复

基本正确，存储高可用选举也比较复杂

2018-06-28

大光头 0

不一样，存储高可用更复杂。计算高可用要考虑任务调度，但是存储高可用不光要考虑这个，还要考虑数据一致性的问题

2018-07-11

Leon Wong 0

往往AP设计倾向的系统，大多数是对称集群;而往往CP设计倾向的系统，大多数都是非对称集群

2018-07-03

阿鼎 0

实际上没有哪个业务同意人工切换主备的，都是需要自动切换。请问老师，热备和温备有什么区别?

2018-07-01

作者回复

有的业务是人工切换的，因为简单，例如一些酒店管理系统

2018-07-02

今夕是何年 0

计算高可用和存储高可用最大的区别就是 存储高可用需要数据复制。

2018-07-01

凡凡 0

1.任务集群主要难点在任务和资源的分配，对于共有计算资源一般需要一个任务分配器的角色，做统一分配和管理，比如mesos和k8s以及在他们基础上实现的调度器。但是对于自有资源的计算任务，可以没有调度器的绝色，只需要负载均衡器即可。对于第一种类型的计算集群复杂度是比较高的。2．存储集群高可用的难度分为读请求和写请求两个方面，写一般需要写入多个节点才算完成，即存储多份保证高可用。读请求在任何一个数据存储节点都可以读取。存储集群还需要一个索引服务（头节点），写时分配需要写入的节点，读时索引数据存储节点。

2018-06-28

大P 0

看评论人人都是架构师的感觉，老师讲的好，各位学友更是青出于蓝而胜于蓝！佩服

2018-06-28

张立春 0

如果正在执行的任务发生计算节点故障，要继续平滑完成该任务的难度不比存储节点出现故障的数据同步低吧

2018-06-28