分布式存储技术：总结2023，展望2024

Original 黄岩 zstorage 2024-02-18 15:30 四川

1. 前言

这篇文章对zStorage相关的一些存储技术做一个总结，并对未来发展趋势做一些展望。

本文不关心整个存储行业的技术发展，只讨论跟zStorage有关的技术。实际上，分布式存储是一个相当大的领域，我没精力、也没能力，对整个分布式存储领域做一个全面的技术总结。zStorage的目标是"最好的承载数据库的存储平台"，凡是跟这个目标相关的技术，本文都关心。

2. 全闪分布式

全闪分布式这个概念，逐渐被整个存储行业认可了。国内几个主流厂商，都在开发全新的全闪分布式架构。当然，也有厂商继续基于以前的混合分布式存储架构来包装全闪产品。可以预见，没有全闪分布式架构的厂商，在未来会逐渐被淘汰。当年PureStorage的全闪阵列发布之后，基于混合阵列架构开发的全闪产品，很快就被市场淘汰了。同样的故事，会在分布式存储领域再重复一次。

全闪分布式架构的主要特征在于：

- 异步Event模型
  异步event编程模型。Thread模型的优势是代码结构简单，编码生产效率高，但是性能开销大。在业务逻辑简单，性能要求高的系统中，彻底抛弃thread和mutex是必然选择。
- 用户态驱动
  与内核态驱动相比，用户态驱动省掉了内核与用户态之间的线程上下文切换，也省掉了内核与用户态之间拷贝数据的开销，是提升性能的必选项。
- RDMA和NVMeoF
  100G RoCE或者IB接口，以及NVMeoF，已经成了高性能分布式存储的标准配置。值得提一下的是，很多现存系统的性能瓶颈不在网络通信模块，即便配置了100G IB/RoCE 也无法提升性能，社区版Ceph就存在这个问题。
- 高性能本地存储
  有些分布式存储系统直接采用ext4、xfs等单机文件系统作为本地存储模块，早期的 Ceph也是这么做的，使用了十年之后，Ceph研发团队终于意识到，对于分布式存储的本地存储来说，ext4/xfs/btrfs等单机文件系统并不是最佳选择。

2.1. 存储介质

2.1.1. ZNS

ZNS接口声称可以减少SSD的OP成本，减少GC导致的性能开销，我并不认同。用append-only接口替换掉write-in-place接口，并没有真正省掉OP成本和GC开销，只是把这些成本和开销从SSD盘内部转移到了应用层。使用ZNS接口应用，必须自己设计 类似LSM Tree的数据结构，自己定期对LSM Tree做Compaction以回收无用空间。那些没有被即时回收的空间，跟SSD盘内部配置的OP是等价的。从应用发起的Compaction流程，占用了HOST的CPU处理能力，并且占用了HOST到SSD的总线带宽。

ZNS的真正潜力在于，允许SSD盘配置性能更弱的CPU，以及允许SSD盘配置更少的DRAM， 从而让SSD盘的成本下降。可是，硬件成本不只跟复杂度相关，跟销量本身相关性更强，销量越大成本越低。现在看来，ZNS靠简化SSD盘内部设计来节约掉的成本，并不足以抵消普通NVMe接口SSD由于销量巨大而带来的优势。

重新为SSD盘定义块接口，已经有过很多尝试了。但是，只有NVMe获得了巨大成功。Open Channel算是失败了。NVMe中有一个Streams选项，也没有得到广泛应用。现在看来， ZNS也很难成功。最近两年又有人提出FDP，是另外一个版本的Streams，FDP是否能得到推广，尚未可知。

2.1.2. SCM

随着Intel Optane PM宣布停产，字节级寻址的持久化内存的发展速度慢了下来。无论 如何SCM技术不会消失，很可能在CXL时代大放异彩。现在开发的新架构，必须考虑如何 利用好SCM。如果力量足够强，应该投入一些研发资源持续跟进SCM技术的发展。

2.1.3. HDD

早有预测说，2027年SSD的每TB存储成本会下降到与HDD持平，到时候HDD将彻底失去 生存空间，HDD会像BP机、MP3、傻瓜相机、车载GPS导航仪一样，彻底消失。在我看来， 也许2027年太早，HDD还不至于彻底消失。这个时间点，也许是2030年，或者2035年， 总之，这个时间点一定会到来，不需要等太久。在2023年还继续把研发成本投入 到SSD和HDD的混合存储产品上，是不明智的。

2.2. 重删

全闪阵列支持重删，背后的逻辑是：SSD盘的随机读性能，在全闪阵列中用不完的，富裕很多。CPU性能也是有富裕的，那么，就把这些富裕的SSD和CPU的性能利用起来，用 在重删上面，来降低每TB成本。

主存储的重删，不能只有产品支持，还要配合商业模式，也就是，不再按照SSD盘的物 理容量定价，而是按照客户存入的数据量定价。数据缩减率低的风险，厂商承担。这是 全闪阵列成功的原因。

但是，现在看来，以上这些成功因素，已经发生了变化。牺牲一些性能，换取每TB价格 下降，看起来不再那么有吸引力了。10年前，Pure Storage起步的时候，SSD每TB的价格，是HDD的几倍甚至十几倍。但是，现在SSD每TB成本价格已经跟HDD接近了，再过几年 就跟HDD持平了。10年前，阵列还是以HDD或者混合存储为主，那个时代，阵列的性能都 不高，几万IOPS就可以称霸江湖了。那个年代，人们习惯于低性能的存储，几百上千IOPS的存储，一样可以跑数据库应用。

但是，现在人们已经无法忍受低性能存储了。现在单片SSD的性能就可以达到百万IOPS。另外一方面，有十几个服务器节点，上百个SSD盘的Ceph集群，也是百万IOPS。如果还要牺牲一些性能，换取每TB成本下降，这个逻辑有些讲不通了。

所以，现在搞全闪阵列，或者全闪分布式，是否还应该搞重删，值得重新思考审视。在我看来，把CPU的每一条指令，都用到提升IOPS和带宽上，是更明智的选择。

2.3. 开源软件

现在讨论分布式存储，已经很难避开Ceph、DAOS、SPDK这些开源软件。Ceph已然垂垂老矣，但凡有一点实力的Ceph厂商，已经开始着手开发下一代架构的产品，或者基于 DAOS， 或者全自研的新架构。

DAOS发展迅猛，但是遭遇了Intel Optane停产的打击，基于DAOS开发的产品比原计划 上市时间推迟至少一年，也为此浪费了很多研发成本。对于实力比较强的团队，这不 算致命，但是对于财力不强的企业，打击不小。在Optane停产之后，Intel是否会停止对DAOS的投入，是一个值得关注的风险点。研发是高风险、高投入、高产出的活动，如果 Intel不再投入，基于DAOS做产品化的团队，是否有能力推动DAOS架构继续发展，是值得怀疑的。一般来说，对开源软件做产品化的团队，对开源技术的先进性有强烈的信仰，对开源社区的技术决策有

很强的依赖。团队如何摆脱盲目迷信开源习惯，如何培养独立思考、独立决策的能力，将会是一个非常大的挑战。

SPDK并不是简单包装一下就可以上市销售的完整系统，只是一些零碎的存储相关驱动 和协议的工具模块库。SPDK里面的工具模块性能和质量都还可以，只是必须全流程按 照SPDK设定的轮询模式才能充分发挥出它的性能。

无论Ceph、DAOS还是SPDK，使用这些开源软件的团队都应该有清晰的策略，是选择坚决跟随开源社区版本，还是分叉出自己的版本，从此与开源版本分道扬镳。

zStorage使用了SPDK，我们的策略是后者，维护自己的版本，遇到严重Bug到社区去找一找，如果已经解决了，就把这个Bug的补丁单独合并过来。如果找不到，就自己解决。如果我们需要什么新功能特性，多半是自己开发。例如，我们给增加了SPDK中 断轮询自动切换的特性。还增加很多统计功能，例如，统计哪些poller耗时过长。

2.4. Share Everything架构

share everything架构这个口号是vastdata喊出来的，Vastdata是2023增长最迅猛的 存储初创公司。存储行业的研发人员，都不应该忽视这个公司，以及它的DASE架构。

share everything架构分为两层：下面的存储层是双控的EBOF全闪存储硬盘框，上面业 务层运行在标准服务器节点上的软件集群。所有业务节点通过100G RDMA网络共享访问 EBOF存储节点。EBOF存储节点的特点是双控、双口nvme SSD盘、每个控制器带有多个 100G RDMA网口。Vastdata自己并不生产EBOF硬盘框，委托其他厂商生产。vastdata的 目标是让EBOF硬盘框变成像标准服务器一样的廉价硬件，这样围绕share everything 架构生态系统才能发展起来。

这种架构可以在集群规模不大的情况下支持大比例EC，这样可以降低每TB存储成本。与 基于标准服务器的分布式SDS相比，这种架构把最小故障域从服务器节点缩小到了SSD盘， 一个EBOF框中单个控制器故障，还有另外一个控制可以工作，不会导致整个EBOF框故障 离线，这种故障模型跟双控阵列一样。这种架构的问题是性价比不高，虽然最近20年网 络技术发展很快，网络带宽迅速从千兆发展到100G，但是IB和RoCE网络的带宽成本都比服务器内部PCIe总线的带宽成本高很多。至少现在，跨越100G IB/RoCE访问EBOF硬盘框， 并没有性价比优势。

无论如何，这个技术值得关注。zStorage也会在未来版本中考虑如何适配EBOF硬盘框， 但不是现在。

2.5. 共识协议

2.5.1. 为什么需要共识协议

分布式系统的设计难点是故障处理。如何在某些部件发生故障的情况，保持系统正常工作状态，包括，数据不丢失，并且能够正确处理从主机所收到的IO请求。这就是故障处理流程要考虑的问题。

存储系统有两种类型，分布式和集中式。它们之间最大差别在于，分布式假设系统中所有所有部分都可能发生故障，所有部件都是可以被更换的。

集中式系统，假设存在一个非常可靠的部件，它不会发生故障。例如，系统的背板，或者仲裁节点等等。集中式系统类似一个官僚系统，对某个故障做出决策判断的基本模式是，下级服从上级。在集中式系统中，如果对某个问题的判断产生分歧，那么按照流程上升，交给上级做裁决。例如，硬盘阵列的A控和B控相互认为对方故障了，认为自己还处于可正常工作状态，这个分歧如何解决？一般来说，由背板来解决，背板上有一个锁，谁抢到，谁就是正常工作状态。背板上这个"锁"，很可能就是一个简单的数字逻辑芯片实现的。就是这么一个简单的锁，解决了"发生分歧以谁的判断为准"的问题。

在分布式系统中，没有永远不发生故障部件，所有部件都可能发生故障。在分布式系中，对关键问题的判断，需要采用"投票"的方式，共同做出决策。投票，就需要有一个规则，大家按照这个规则来投票。这个投票的规则，就是Paxos和Raft这类共识协议。在某些部件发生故障的情况下，让系统能够做出正确的反应，对分布式系统来说，并不简单。其中有一个前提条件是，"投票规则"的设计没有漏洞。这一点也不容易做到，因为有很多特殊情况需要考虑。如果每个分布式存储系统都自己设计一套投票规则， 一般来说，总会在运行过程中发现各种规则没考虑到的情况。Paxos和Raft解决了这个问题，这两个共识协议被大家认为是比较完善的，逐渐被行业接受了。一般来说，新的分布式系统没必要再定义自己特殊的"投票规则"了，直接采用Paxos和Raft就好了，这两个共识协议，很多双同行的眼睛仔细检视过了，经过了TLA+这种形式化语言的证明。因此，对分布式存储系统的设计者来说，透彻理解Paxos和Raft很重要。

分布式存储软件的架构，由控制面和数据面两个部分来构成。系统运行的关键决策由控制面负责，例如，集群由哪些节点构成，每个节点负责处理哪部分数据，哪个节点发生了故障等等。控制面由一些控制节点构成，控制节点按照共识协议的规则来共同决策，这保证了决策的连贯和一致，避免了"集群脑裂"。

数据面负责处理IO请求，也需要对一些问题做决策。例如，何时可以给主机回复写成功？数据有三个副本，应该从哪个副本读？等等，如果这些问题全部交给控制面决策，那么控制面就会变成瓶颈，集群的性能会很差。数据面需要自行对某些问题做决策，因此，数据面也需要运行共识协议。

2.5.2. 为什么关注共识协议

虽然Paxos和Raft的设计很严密，投票规则没有漏洞，但是，还不能完全满足实际产品的需求。对Raft的改进和增强主要集中在数据面，etcd可以满足管控面的需求。zStorage使用了Raft协议，我们遇到下面一些问题：

- Multi-Raft
  zStorage在数据面使用Multi-raft，把一个存储池的数据打散(sharding)到多个Raft 组中，每个节点和硬盘承载多个不同Raft组的副本，在扩容或者硬件故障情况下，在节点和盘之间重新分布这些Raft副本，以此达成可弹性伸缩的目标。
  每个节点上有多个Raft组，心跳消息就会很多，会占用很多网络带宽和CPU处理能力，实际上，并不需要在每个Raft组都维持自己的心跳，只需要在节点之间维持心跳就 可以探测到节点（或者进程）故障了。因此，需要对多个Raft组的心跳消息进行合并。
  除了合并心跳消息之外，还需要考虑对多个Raft组的日志合并之后写盘。日志合并之后， 减少了对SSD盘随机写的次数。另一方面，SSD盘写操作的最小单位是512Bytes的数 据块，如果不做合并，要等到Raft日志凑齐一个512B，可能需要很长时间。如果不等凑齐就写盘，那么SSD盘上就会有比较多的空洞，增加了写放大，影响性能，也影响 SSD盘的寿命。
  zStorage是块存储系统，不涉及跨越多个Raft组的分布式事务管理，这方面比分布式数据库简单一些。

- 硬盘状态机
  zStorage是一个分布式存储系统，对它来说，集群所有服务器节点硬盘上的数据全集， 构成了Raft状态机。但是，Raft有一个隐含的基本假设是，状态机是在内存中的，并且尺寸并不很大。虽然Raft的大论文中有一节提到了硬盘状态机，但是篇幅并不长，只给了一些基本的处理原则，并没有具体的设计。
  Raft硬盘状态机，首先要解决的问题是，如何打快照？不可能再像内存状态机一样， 把数据完整复制一份到硬盘上了，因为硬盘上所有存储空间都归属状态机，没有其 他多余空间用来存储快照了。Raft的大论文中给出的建议是，本地存储要支持COW快 照，这个方法其实也不太可行。一方面对于用户数据本身要支持用户级别的快照， Raft的COW快照跟用户级别的快照可能存在相互冲突，不好解决。另外，在发生盘故 障的情况下，对于几个TB的数据执行快照安装流程，可能耗时几个小时。在这几个小 时中保持日志不删除，也不可行。zStorage用了自己的方法，解决Raft在硬盘状态机 情况下如何打快照和安装快照的问题。
  当出现硬盘故障之后，需要对故障硬盘上的副本做数据重构。具体过程是，对跟故障 硬盘相关的Raft组做

配置变更，用正常硬盘上的Raft副本替换掉故障副本，配置变更之后要对新副本执行安装快照的过程，也就是把数据复制到这个新副本上。一般来说，这个数据复制的过程耗时很长，zStorage当前版本规格是4TB/小时，那么一片8TB的 SSD盘发生故障，需要耗时至少2小时。如果在这两个小时过程中，再发生一个SSD盘 故障，那么就会有部分Raft出现两个副本故障，只剩下一个正常副本，那么这个Raft 组就不可用了，不能再继续正常工作。整个系统中有几千个Raft组，只要有1个Raft组 不能正常工作，那么整个存储池就停摆了。

为了解决这个问题，zStorage给Raft增加 了一个logger角色。logger是一个可以在只有日志没有完整状态机数据的情况下工作 的Raft组成员，logger可以投票，但是不能被选为Leader。增加了Logger角色之后，zStorage的三副本可以在一个SSD故障的情况下，在10秒内恢复为三副本工作状态，只 要10秒内没有发生第二个SSD故障，那么就不会出现Raft组不可用的情况。增加了 Logger角色之后，提升了Raft在硬盘状态机情况下的可用性。

zStorage还利用Logger角色实现了基于Raft的二副本模式。

- Erasure Code

  实际上，Raft并不需要每个成员都保持一个完整的状态机，一个Raft组中所有成员加 起来有一份完整的状态机就够了。在分布式存储产品上实现基于Raft的EC，目前可供参考的例子并不多，很多问题需要自行解决，zStorage也正在探索中。

- 实现问题

  - 配置变更

    Raft论文中讲了两种实现成员变更的方法：1、单步骤只变更一个成员，也就是在 一次变更中，要么增加一个成员，要么减少一个成员，不能直接替换掉一个成员。例如，要实现由(A, B, C)==>(A, B, D)的变更必须分为两步走，第一步先做 (A, B, C)==>(A, B, C, D)，第二步再做(A, B, C, D)==>(A, B, D)。这种方法 比较简单，但是有可用性问题；2、第二种做法是joint consensus方法，实现稍复 杂，没有可用性问题。很多开源的Raft都没有实现joint consensus方法。

  - 流程异步化

    SPDK所谓的“线程”，并非线程，而是异步event处理模式。zStorage达成相对比较 高性能的一个重要原因是，避免了pthread、mutex、condition的编程模式，使用 SPDK框架提倡的异步event模式。这就是要求代码中任何位置都不能阻塞，Raft流程中的写日志、发送消息、写元数据等等，都不能阻塞。目前有一些异步实现的开源Raft可供参考，但是，这些开源Raft总有些不经常运行的流程是同步实现的。

  - 读操作

    Raft中实现读的方法有两种：1、读操作其他操作相同，也要走Append Entry和 commit流程；2、不走Append Entry和commit流程，直接在Leader上读状态机；显然第2种方法性能更好，但是要保证新的Leader选出来之前，旧的Leader先退位， 如果旧的Leader等待了一段时间，没有收集到足够的心跳消息，它自行退位。如果 这个旧Leader所在的服务器时钟不准，那么“等待一段时间”就不可靠，就可能出 现新旧两个Leader同时工作，Client从旧Leader读出了旧数据的情况，从而导致了 数据不一致问题。为了避免这种问题，zStorage会每隔一段时间，对所有服务器的 时钟频率做一次校对，并且让“等待一段时间”这个时长足够容忍不同服务器节点 之间的物理时钟频率的偏差。

  - 测试用例

    zStorage没有设计自己的共识算法，而是直接使用了Raft，这个相对成熟的算法。这个技术决策客观上缩短了zStorage的开发周期，节约了开发成本。但是，即使完全按照Raft论文描述的原理去写代码，也不能完全避免代码中的Bug。找出并改 掉这些代码中Bug，最关键的是测试，要有一套完备的测试用例。etcd一套已经 得到广泛应用的Raft开源实现，etcd有一套针对Raft的测试用例集，可用参

考。但是etcd的测试用例是用golang写的，zStorage是C语言实现的，etcd用例无法直 接使用，zStorage用C语言重写了etcd所有用例，并且每晚例行运行这些测试用例 对zStorage的raft实现进行测试。

未来zStorage还会持续在Raft协议上投入研发资源，我们目标让Raft更好地为存储业务服务。zStorage对于改进Raft协议本身没兴趣，例如，如何让raft支持乱序提交，这类 改进收益有限，但是它修改了raft核心工作流程，风险比较高，投入产出比不高。

2.6. 存储层接口

一般来说，分布式存储系统至少有两层：协议网关层、数据存储层，有些系统还有一个 中间的业务层。数据存储层的基本功能是对数据进行持久化，有些还会实现一些业务功 能，例如：加密、压缩、块存储的逻辑卷、快照、克隆等等。存储层实现功能不同，对 上层提供的接口也不同。总的来说，存储层接口有两类：1、追加接口；2、原地更新接口。根据存储层接口不同，可以把分布式存储系统的架构分为两类。

2.6.1. 追加接口（Append-Only）

谷歌的GFS、微软的WAS(Windows Azure Storage)存储层的接口是追加接口。一般来说， 提供接口的分布式存储系统，存储层不会实现很复杂的业务，例如：逻辑卷、快照、克隆 等等，都不会在存储层实现。存储层最多只实现压缩、加密等简单的数据处理服务。另外，这种类型的系统一般也不需要实现Multi-Paxos和Raft等共识协议，只需要实现 相对比较简单的seal-and-new流程，就可以保证故障场景中数据一致性。由于存储层 的数据一旦写入就不再改变，扩容和故障恢复的数据重构流程都相对简单。

这种架构有两个主要缺点：1、业务层实现复杂；2、处理流程长，性能开销大。由于存储 层只支持追加写，因此业务层的实现更复杂，一般来说，业务层需要实现一套LSM Tree的 处理逻辑，包括compaction流程。这种架构简化了存储层空间管理，也减轻了SSD盘 内部的GC压力，但是需要业务层发起compaction流程，并且需要跨网络进行数据合并， 这使得空间回收的处理流程更长，性能开销更大。

2.6.2. 原地更新接口（Write-in-place）

zStorage和Ceph的存储层对外提供原地更新接口。与Append-Only接口的架构对比， Write-in-place接口可以在存储层实现更多业务功能，也不需要跨越网络做compaction， 性能更好。缺点是模块的功能边界不够清晰，不太适合规模很大的开发团队。zStorage 团队规模不算很大，而且对性能要求很高，这种架构更合适。

3. 数据库的存储平台

3.1. 软件定义存储（SDS）

zStorage平台坚持走纯软件路线，不会与某种类型的特殊硬件强绑定。zStorage并不排 斥利用特殊硬件，但是要考虑投入产出比，要考虑特殊硬件对软件架构发展的影响。基 于zStorage平台的产品可以走软硬结合的一体化路线，提升性能、改善产品稳定性等等。

DAOS强绑定Intel Optane PM，由于Optane停产对DAOS的发展造成了不小的冲击。强绑某 种硬件，对于分布式软件来说，有很大风险。这种强绑定硬件的设计模式，也没有创造 很多新价值。

全闪阵列是采用专用硬件的典型代表，每个厂商都设计生产自己的专用硬件，得不到服务器通用硬件生态系统的支持，因此成本比较高。

3.2. 存算分离

分布式（集群）数据库有两种架构模式：1、每个数据库节点各自管理自己的持久化存储， 节点之间不共享数据，如TiDB、OceanBase、ClickHouse等，就是这一类；2、数据库节点 与底层存储设备是分离的，底层存储在

所有数据库节点之间共享，例如Oracle RAC、 PolarDB等，都是这一类数据库。第1类是存算一体的Share nothing架构，第2类是存算 分离的架构。第1类的优点是，不需要专门的存储系统，数据库自己管理存储系统，数据 库自己负责把数据复制为几份，自己管理冗余数据副本。

存算分离的主要意义在于，可以独立对存储和计算进行扩容和缩容。其次，存算分离架 构可以充分利用专业存储系统的能力，提升系统整体的可靠性、可用性和易用性。

应用对于存储和计算需求，并不总是确定不变的，有时候应用需要更多的计算资源，有 时又需要更多存储资源，如果存储和计算之间的比例不可弹性变化，那么就有很大概率 存在浪费。一般来说，share nothing的存算一体分布式数据库集群更适合OLAP应用。但是Snowflake采用了存算分离架构，并且表现非常出色。

既然分布式数据库自己就能管理硬盘，能做三副本，硬盘故障之后也能做数据重构，为 什么还需要数据专业的数据存储系统？对存储数据来说，专业的数据存储系统，比兼职的数据库做得更好。存储系统一般都支持快照、克隆等特性，并可以可以基于快照实现 块存储层的备份，速度快，数据一致性有保障。专业存储会定期读出数据做校验，如果发现错误，则用冗余副本来恢复。专业存储支持纠删码等冗余方式，每TB存储成本更低。

存算分离架构已经成了比较明显的趋势。甚至连TiDB、ClickHouse等Share Nothing架构 的分布式数据库也准备推出存算分离的版本。

如何更好支持存算分离的数据库架构，是zStorage的一个重要课题。

3.3. 平衡OLTP和OLAP

数据库应用分为交易型（OLTP）和分析型（OLAP）两种类型。银行存款取款，查询某人 的账户余额等，对数据库来说，算是交易型应用。数据库的交易型操作需要存取的数据量 不大，但是对时延要求比较苛刻。如果存储系统时延太高，或者时延不稳定，时高时低， 那么用户的感觉就不好。如果银行要统计最近一年每天上午9:00 - 10:00时间段的存款 总金额，那么就需要把所有存款记录读出来，过滤出9:00 - 10:00的存款记录，然后再 把这些记录中的金额加起来。分析型操作需要访问的数据量很大，一般来说，分析型 任务的执行时间都很长，因此对时延不是特别敏感，但是对存储系统的吞吐量要求 比较高。

一般认为，Share nothing的分布式数据库更适合分析型应用，也就是TiDB、OceanBase、 ClickHouse这一类。因为可以多个节点同时并行做分析任务，然后再对结果进行汇总。基于共享存储的集群数据库更适合交易型应用，Oracle RAC就是这一类。生产环境的实际情况是，很难找到纯粹的交易型应用。更普遍的需求是， 多数情况下是交易型操作，也夹杂少量的报表类分析型业务。交易型的数据库， 也需要满足少量分析型任务的需求。

软件定义的分布式存储系统，跟Share nothing分布式数据库架构类似。如果基于共享的 分布式存储系统构建Oracle RAC类似的交易型数据库，同时让分布式存储帮助完成少量 的分析型任务，充分利用多节点的并行能力。实际上Oracle的数据库一体机ExaData就是 这么做的，它也在市场上取得了商业成功。所以，我认为“分布式存储”、“共享存储集群”、 “计算下推”同时具备这三个要素的架构，是理想的数据库架构。

关于zStorage

zStorage是针对数据库应用开发的高性能全闪分布式块。三节点zStorage集群可以达到200万IOPS随机读写性能，同时平均时延<300us、P99时延小于800us。zStorage支持多存储池、精简配置、快照/一致性组快照、链接克隆/完整克隆、NVMeoF/NVMeoTCP、iSCSI、CLI和API管理、快照差异位图（DCL）、慢盘检测、亚健康管理、16KB原子写、2副本、强一致3副本、Raft 3副本、IB和RoCE、TCP/IP、后台巡检、基于Merkle树的一致性校验、全流程TRIM、QoS、SCSI PR、SCSI CAW。

(结束)

个人观点，仅供参考

People who liked this content also liked

一个TCP性能优化点

zstorage

zStorage的Raft快照

zstorage