# Kubernetes 常见开源分布式存储选型对比

## 功能需求对比

### ceph

- ceph支持三种存储接口：块存储、文件存储、对象存储，且能够支持上千个存储节点的规模，支持 TB 到 PB 级的数据。

- ceph采用 CRUSH 算法，数据分布均衡，并行度高，不需要维护固定的元数据结构，数据具有强一致性，确保所有副本写入完成才返回确认

- 适合读多写少场景

### openebs

- OpenEBS主要关注块存储，如果只需要基本的块存储，OpenEBS更简单

### longhorn

- longhorn 在小规模集群使用时，性价比较高，配置简单，但是兼容性会存在一定问题，比如没用正式支持xfs，默认支持ext4并在此条件下运行可靠

## 2. 性能对比测试

测试结果参考：<https://medium.com/volterra-io/kubernetes-storage-performance-comparison-v2-2020-updated-1c0b69f0dcf4>

参照清单:

- GlusterFS（开源）

- Ceph（开源）

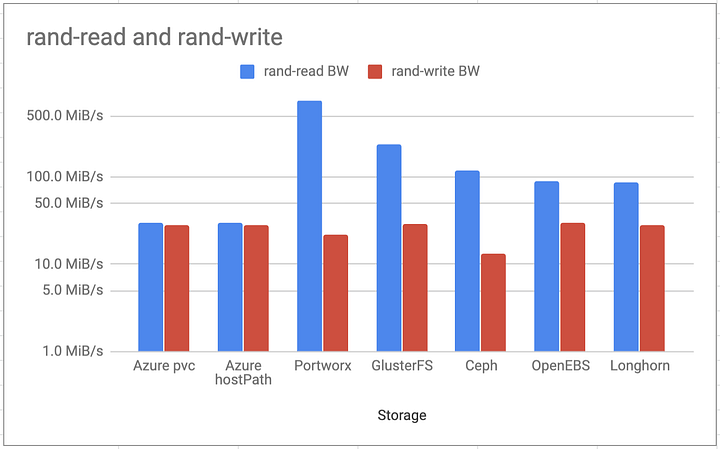
- OpenEBS （开源）

- Longhorn（开源）

### 随机读写带宽

表格

描述已自动生成



结果：

- 随机读BW：Portworx > GlusterFS > Ceph 1.3x> OpenEBS > Longhorn > Azure hostPath > Azure pvc

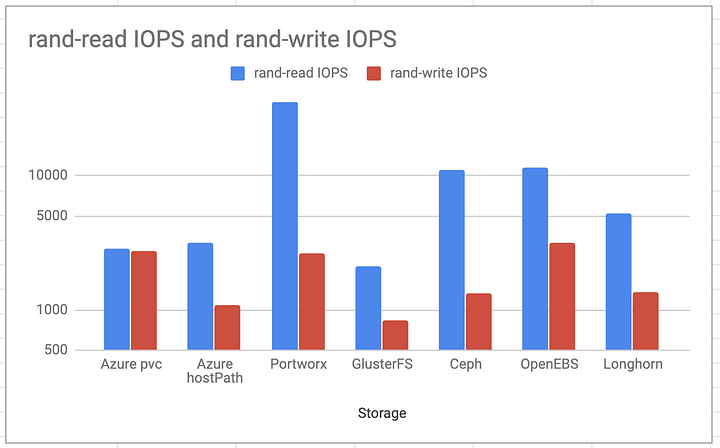
- 随机写BW: Ceph 2x> OpenEBS

- OpenEBS和Longhorn的性能几乎是本地磁盘的两倍。原因是读取了缓存

### 随机读写IOPS

表格

描述已自动生成



结果：

- 随机读IOPS：Portworx 3x > OpenEBS = Ceph 2x> Longhorn

- 随机写IOPS: OpenEBS 2.3x > Portworx > Ceph > Longhorn

### 读写延迟

表格

描述已自动生成

图表, 条形图

描述已自动生成

结果：

- OpenEBS和Longhorn上写入的延迟更好。 GlusterFS仍然比其他存储更好。

### 顺序读/写

表格

描述已自动生成

图表, 条形图

描述已自动生成

结果：

- 顺序读/写测试显示的结果与随机测试相似

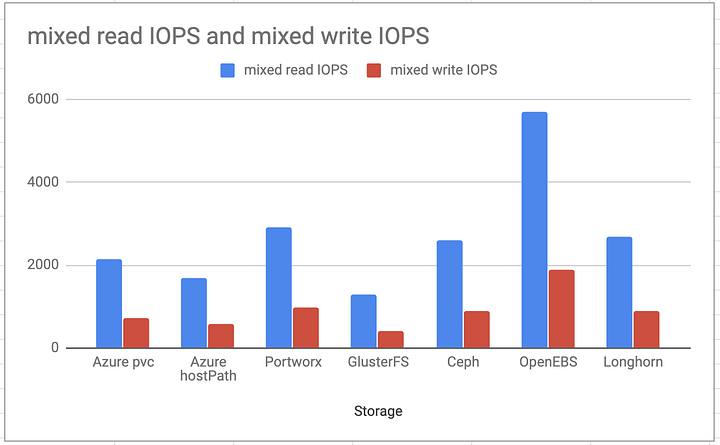
- 顺序读：Longhorn ≈ OpenEBS ≈ Ceph

- 顺序写：Longhorn ≈ OpenEBS 1.5x> Ceph

### 混合读/写IOPS

表格

描述已自动生成



结果：

- 在读写方面，OpenEBS交付的速度几乎是PortWorx或Longhorn的两倍。

- 混合读写 ：OpenEBS 2x> Ceph = Longhorn

## 3. 社区支持和生态系统

- Ceph，2004年发行，是一个成熟的开源项目，有庞大的社区和广泛的应用场景， github 12k ☆

- OpenEBS，2016年10月发行，也有活跃的社区，但相对较新 github 8.2k ☆

- Longhorn，2019年4月发行，github 4.8k ☆

## 4. 部署和管理复杂度

- Longhorn，配置最简单，可靠性高，支持快照、备份

- OpenEBS相对简单，可通过Helm Chart和Operator进行快速部署和管理，

- Ceph的硬件要求和复杂性较高，在管理部署方面较为困难，Ceph的一个很大的优点是只需要维护一个系统，可以把所有的东西放在一个盒子里:块存储、对象存储、文件存储

## 5. 总结

- OpenEBS：围绕NVMe的稳健设计，OpenEBS似乎已成为最快的开源容器存储选项之一。OpenEBS的Jiva存储引擎是基于块设备的存储解决方案，它直接利用Kubernetes节点上的本地块设备。由于直接访问本地块设备，Jiva可以提供较低的延迟和较高的IOPS性能。

- Ceph：Ceph的设计目标是提供高可用性和数据冗余，而不是追求最佳的单个节点性能。由于Ceph涉及复杂的数据分布和网络通信，它的性能可能会受到一些额外的开销。

- Longhorn：对于简单的块存储用例，绝对是有效的选择，它与OpenEBS Jiva非常相似。

- GlusterFS在某些场景中可能遇到一些挑战，例如对于大规模、高吞吐量和低延迟的工作负载来说，可能需要进行一些优化和调整。

附录

1. 性能对比测试

测试结果参考：<https://medium.com/volterra-io/kubernetes-storage-performance-comparison-v2-2020-updated-1c0b69f0dcf4>

参照清单:

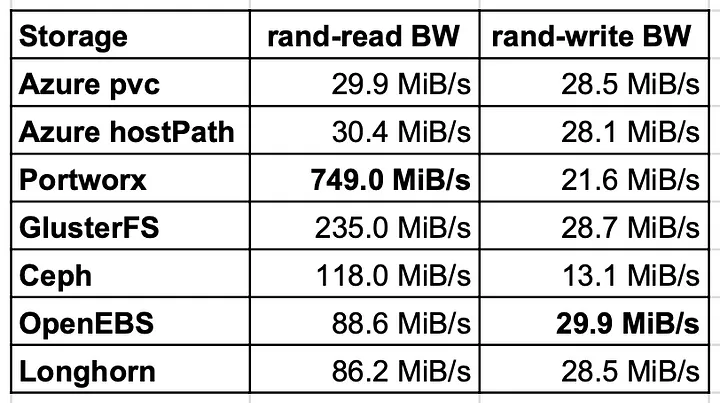
- GlusterFS（开源）

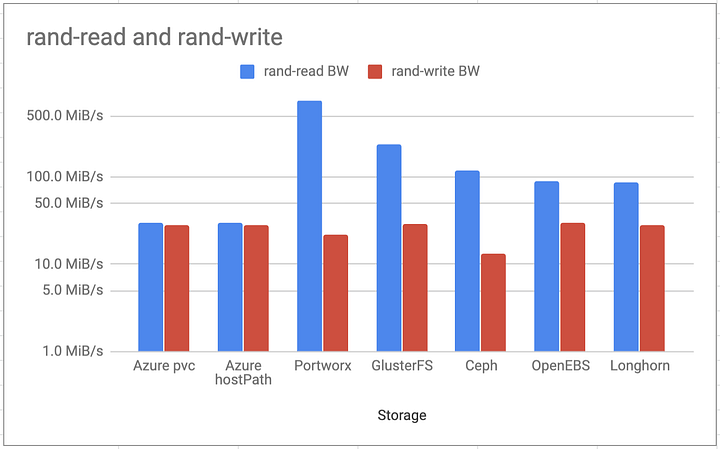
- Ceph（开源）

- OpenEBS （开源）

- Longhorn（开源）

随机读写带宽





结果：

- 随机读BW：Portworx > GlusterFS > Ceph 1.3x> OpenEBS > Longhorn > Azure hostPath > Azure pvc

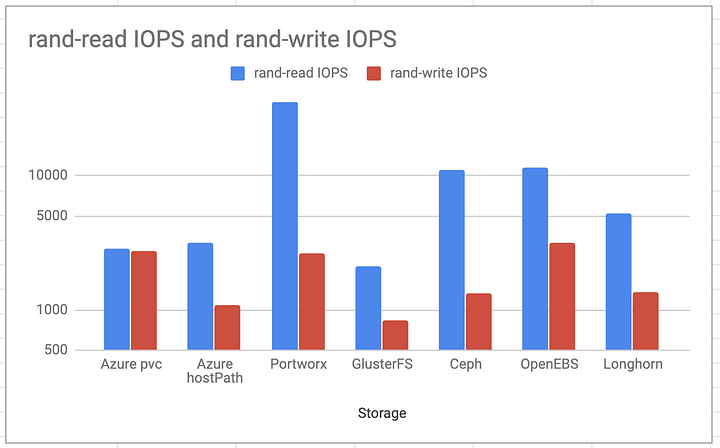
- 随机写BW: Ceph 2x> OpenEBS

- OpenEBS和Longhorn的性能几乎是本地磁盘的两倍。原因是读取了缓存

随机读写IOPS

表格

描述已自动生成



结果：

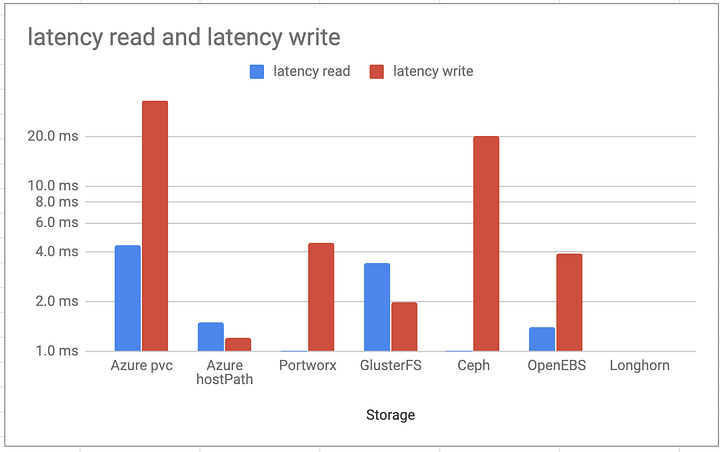
- 随机读IOPS：Portworx 3x > OpenEBS = Ceph 2x> Longhorn

- 随机写IOPS: OpenEBS 2.3x > Portworx > Ceph > Longhorn

读写延迟

表格

描述已自动生成



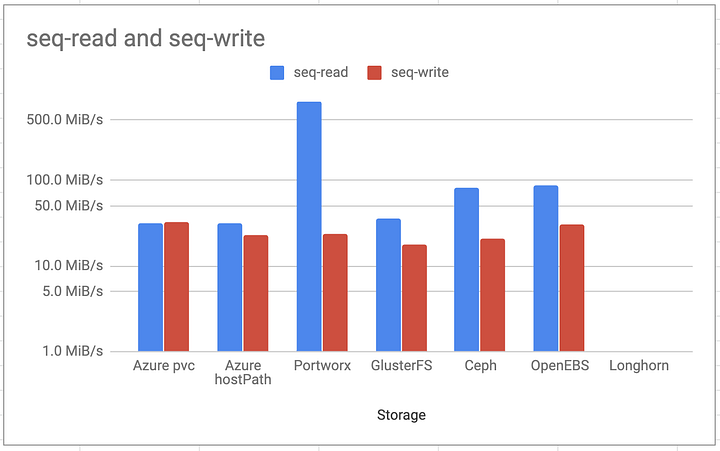
结果

- OpenEBS和Longhorn上写入的延迟更好。 GlusterFS仍然比其他存储更好。

顺序读/写

表格

描述已自动生成

结果：

- 顺序读/写测试显示的结果与随机测试相似

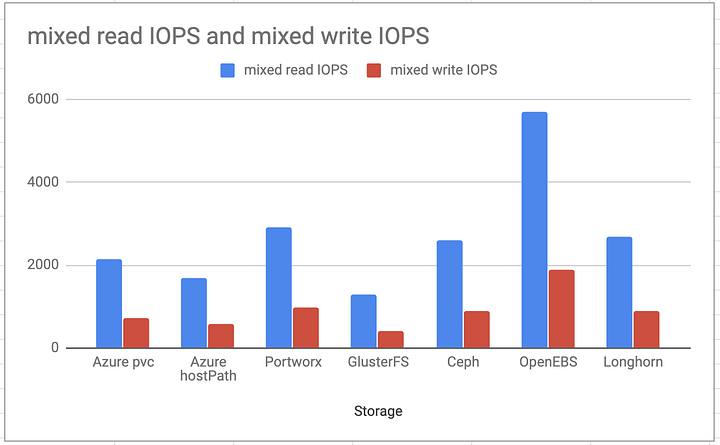
- 顺序读：Longhorn ≈ OpenEBS ≈ Ceph

- 顺序写：Longhorn ≈ OpenEBS 1.5x> Ceph

混合读/写IOPS

表格

描述已自动生成

结果：

- 在读写方面，OpenEBS交付的速度几乎是PortWorx或Longhorn的两倍。

- 混合读写 ：OpenEBS 2x> Ceph = Longhorn

建议在为该类型的守护进程配置的主机上运行特定的Ceph守护进程。我们建议使用其他主机来处理使用您的数据集群的进程（例如OpenStack、CloudStack）

**1.1 CPU**

- Ceph 元数据处理器应该有相当大的处理能力（四核心或更高的CPU）

- Ceph OSDs 运行RADOS服务，用CRUSH计算数据放置、复制数据，并维护自己的集群地图副本。因此，OSD应该有合理的处理能力（例如双核处理器）

- Ceph monitor 守护进程维护了 clustermap，它不给客户端提供任何数据，因此它是轻量级的，并没有严格的处理器要求，普通的单核服务器处理器即可。但必须考虑机器以后是否还会运行 Ceph 监视器以外的 CPU 密集型任务。例如，如果服务器以后要运行用于计算的虚拟机（如 OpenStack Nova ），你就要确保给 Ceph 进程保留了足够的处理能力，所以我们推荐在其他机器上运行 CPU 密集型任务。

一个 Ceph OSD 守护进程需要相当数量的处理性能，因为它提供数据给客户端。要评估 Ceph OSD 的 CPU 需求，知道服务器上运行了多少 OSD 上非常重要的。通常建议每个 OSD 进程至少有一个核心。可以通过以下公式计算 OSD 的 CPU 需求：

（（CPU sockets \* CPU cores per socket \* CPU clock speed in GHz ）/ No.of OSD ） >= 1

case 1: Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz 6 core --> 1 \* 6 \* 2.40 = 14.4 适合多达 14 个 OSD 的 Ceph 节点

case 2: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz 12 core --> 1 \* 12 \* 2.50 = 30 适合多达 30 个 OSD 的 Ceph 节点

# 如果打算使用 Ceph 纠删码特性，最好能有一个更高性能的CPU ，因为运行纠删码需要更强的处理能力。

```

**1.2 RAM**

**| Monitors and managers (ceph-mon and ceph-mgr)**

- 监视器和管理器守护进程的内存使用量一般会随着集群的大小而变化。

- 元数据服务器和监视器必须可以尽快地提供它们的数据，对于小型集群，一般来说，1-2GB就足够了。对于大型集群，应该提供更多（5-10GB）。

- 你可能还需要考虑调整设置，如mon\_osd\_cache\_size或 rocksdb\_cache\_size

**| OSDs (ceph-osd)**

OSD 的日常运行不需要那么多内存（如每进程 500MB ）差不多了；然而在恢复期间它们占用内存比较大（如每进程每 TB 数据需要约 1GB 内存）。通常内存越多越好。

**1.3 Memory**

- 通常不建议将osd\_memory\_target设置为2GB以下，可能会将内存保持在2GB以下，同时也可能导致性能极慢。

- 将内存目标设置在2Gb和4Gb之间通常有效，但可能会导致性能下降，因为元数据可能在IO期间从磁盘读取，除非活动数据集相对较小。

- 4GB是目前默认的osd\_memory\_target大小，这样设置的目的是为了平衡内存需求和OSD的性能，以满足典型的使用情况

- 设置osd\_memory\_target高于4GB时，当有许多（小的）或大的（256GB/OSD)数据集被处理时，可能会提高性能。

重要：

OSD的内存自动调整是“尽力而为”。虽然OSD可能会解除内存映射，让内核回收内存，但不能保证内核会在任何特定的时间框架内实际回收释放的内存。

这在旧版本的Ceph中尤其如此，因为透明的巨页会阻止内核从碎片化的巨页中回收内存。

现代版本的Ceph在应用级禁用透明巨页以避免这种情况，但这仍然不能保证内核会立即回收未映射的内存。OSD有时仍然可能会超过它的内存目标。

tips: 我们建议在系统中保留20%左右的额外内存，以防止OSD在临时高峰期或由于内核延迟回收空闲页而导致的OSD出现OOM。这个值可能会比需要的多或少取决于系统的具体配置。

在使用传统的FileStore后端时，页面缓存是用来缓存数据的，所以一般不需要调优，OSD的内存消耗一般与系统中每个守护进程的PG数量有关

**1.4 Data Storage**

要谨慎地规划数据存储配置，在规划数据存储时，需要考虑较高的成本和性能权衡。来自操作系统的并行操作和到单个硬盘的多个守护进程并发读、写请求操作会极大地降低性能。

重要提示：因为 Ceph 发送 ACK 前必须把所有数据写入日志（至少对 xfs 和 ext4 来说是），因此均衡日志和 OSD的性能相当重要。

**| HDD**

- Ceph 最佳实践指示，你应该分别在单独的硬盘运行操作系统、 OSD 数据和 OSD 日志。

- 推荐独立的驱动器用于安装操作系统和软件，另外每个 OSD 守护进程占用一个驱动器

**| SDD**

- 使用固态硬盘（ SSD ）可以降低随机访问时间和读延时，同时增加吞吐量

- 在大量投入 SSD 前，强烈建议核实 SSD 的性能指标，并在测试环境下衡量性能。

- SSD 很适合 Ceph 里占存储空间较少的的数据部分

- 用于日志和 SSD 时还有几个重要考量：

- 写密集语义：日志记录涉及写密集语义，所以要确保选用的 SSD 写入性能和好于或等于HDD。廉价 SSD 可能在加速访问的同时引入写延时，有时候高性能HDD的写入速度可以和便宜 SSD 相媲美。

- 顺序写入：在一个 SSD 上为多个 OSD 存储多个日志时也必须考虑 SSD 的顺序写入极限，因为它们要同时处理多个 OSD 日志的写入请求。

-分区对齐：采用了 SSD 的一个常见问题是人们喜欢分区，却常常忽略了分区对齐，这会导致 SSD 的数据传输速率慢很多，所以请确保分区对齐了。

-成本：通过将OSD的日志存储在固态硬盘上，并将OSD的对象存储存储在独立的机械硬盘上，可能会看到性能的显著提升，并降低成本

**| 其他考虑因素**

- 确保你的OSD硬盘的总吞吐量之和不超过服务于客户端读取或写入所需的网络带宽

- 应该考虑集群在每台主机上存储的数据占整体数据的百分比。如果某个特定主机上的百分比很大，而该主机出现故障，可能会导致超过 full ratio等问题，从而导致Ceph停止工作，作为防止数据丢失的安全规范措施。

- 每个主机上运行多个OSD时，你还需要确保内核是最新的。以确保你的硬件在每个主机上运行多个OSD时，能按照预期的方式执行。

- OSD 数量较多（如 20 个以上）的主机会派生出大量线程，尤其是在恢复和重均衡期间。很多 Linux 内核默认的最大线程数较小（如 32k 个），如果您遇到了这类问题，可以把 `kernel.pid\_max` 值调高些。理论最大值是 4194303 。例如把下列这行加入 `/etc/sysctl.conf` 文件：

kernel.pid\_max = 4194303

- 通常，大量的小容量OSD节点比少量的大容量OSD节点要好，但这并不是定论，应该选择适当的 Ceph 节点密度，使得三个节点容量小于总容量的 10%。例如：在一个 1PB的ceph集群中，应该避免使用 4 个 250TB 的 OSD 节点，因为每个节点占用了 25% 的集群容量。相反，可以使用 13 个 80TB 的 OSD节点，每个节点容量小于集群容量的 10%。

**1.5 故障域**

故障域是指任何阻止一个或多个OSD的故障。这可能是主机上的守护进程停止；硬盘故障、操作系统崩溃、网卡故障、电源故障、网络中断、断电等等。在规划硬件需求的时候，你必须平衡一下，把大部分的职能集中于低故障域中来降低成本，以及隔离每个潜在故障域所带来的额外成本