Pangu 2.0: 如何打造一个高性能的分布式存储系统

> Pangu存储系统是阿里云存储服务的底层存储系统，大家熟知的OSS/NAS/EBS等服务都是基于Pangu系统开发，其重要性无需多言。这篇论文是阿里巴巴在FAST` 23上发表的和盘古系统相关的三篇论文之一，论文里比较详细的介绍了盘古系统的架构以及演进过程，笔者之前在字节也是参与类似系统的开发，因此看到阿里将盘古系统2.0架构第一次系统地以论文形式发表出来，第一时间拿来拜读

摘要

本文介绍了Pangu存储系统如何不断地随着硬件技术和业务模型的发展而不断进化，提供具有**100us级I/O延迟**的高性能、可靠的存储服务。Pangu的进化包括两个阶段：

- 第一阶段，Pangu利用新的SSD硬件和RDMA网络技术重新设计了一套用户空间存储系统，降低了其I/O延迟，同时提高了吞吐和IOPS；
- 第二阶段，Pangu从面向容量的存储服务转变为面向性能的存储服务。为了适应这种业务模型的变化，Pangu将服务器升级为拥有更高容量的SSD和RDMA带宽的存储服务器，网络从25 Gbps升级到100 Gbps。同时还引入了一系列关键设计，包括减少网络流量放大、远程直接缓存访问和CPU计算卸载，以确保Pangu充分利用硬件升级带来的性能提升

除了技术创新，论文里还分享了Pangu进化过程中的运营经验，并讨论了其中的重要经验教训。

概述

Pangu系统从2009年开始开发部署，提供了一个高扩展、高性能和高可靠性存储系统，已经成为了阿里巴巴集团和阿里云底层统一的存储服务。目前服务了诸如淘宝、蚂蚁金服、阿里妈妈等众多核心业务，同时也是阿里云块存储(EBS)、对象存储(OSS)、NAS(Network Attatched Storage)、PolarDB和MaxCompute等云服务的底层存储系统，管理者EB级别的数据和上万亿的文件。

Pangu 1.0(2009 ~ 2015)

1.0阶段的主要目标是能够利用通用的服务器资源(cpu和hdd磁盘)构建一个大规模的分布式文件系统，提供毫秒级别的IO延迟。当年磁盘还普遍都是HDD磁盘，网络还都是几Gbps级别的，Pangu 1.0存储引擎使用的是Linux自带的ext4内核文件系统，网络使用TCP网络，可以认为当时对标的主要是HDFS，从业务视角来看当时面临的主要问题是如何存储大规模的数据，对性能的要求并不是很高。Pangu 1.0时期为了适应业务需求，也提供了多种文件类型，比如Append-only的LogFile，支持随机读写的Random Access File和类似文件系统的临时文件需求的TempFile，可以看出这个时期从Pangu团队来看主要目标是能够接入更多的业务，积极配合满足业务的需求。

Pangu 2.0(2015 ~ now)

随着高性能SSD硬件和RDMA高性能网络技术的出现和成熟，2015年开始，Pangu团队开始积极拥抱这些新兴技术，将其引入到Pangu系统，这也开启了Pangu 2.0阶段。Pangu团队总结了1..0版本中的一些洞察和思考：

- 1.0提供了各种文件类型，尤其是支持随机读写的文件类型，随机读写相比顺序读写并不能发挥SSD硬件的性能，提供极致的吞吐和IOPS
- 1.0单机存储使用的是内核文件系统，IO链路上存在大量的数据复制和中断，这些都影响了IO性能，相比SSD和RDMA消耗的延迟，已经成为了大头
- 数据中心架构已经有服务器为中心演变为了资源为中心，如何适配这种数据中心架构的演进从而提供低IO延迟的服务带来了新的挑战

2.0 第一阶段： 用户态文件系统

为了充分发挥SSD和RDMA的性能，在单机存储引擎的设计上Pangu 2.0做了如下改进：

- 重新设计了文件系统，提供了一个用户态的文件系统(USSOS)，提供了append-only的语义，并重新设计了一种自包含的Chunk Layout避免了文件系统一次io产生两次IO的情况(一次数据io和一次meta io)。
- 2.0在线程模型上使用`run-to-completion`的模型，避免了额外的线程切换开销，实现了用户态存储栈和用户态网络栈的高效协同，同时也提供了一个用户态的CPU/MEM资源调度机制。
- 2.0也提供了一些机制去应对这种变化的环境从而保证服务的SLA(到底是啥机制，没说...)

该阶段实施效果也是相当显著的，2018年双十一期间，Pangu 2.0为阿里巴巴数据库服务提供了280us的优异延迟性能；对于写敏感的EBS/Drive服务，其PCT999延迟低于1ms；对于读敏感的如在线搜索业务，其PCT999延迟低于11ms

2.0 第二阶段：升级基础设施适应面向性能的业务模型和打破CPU/Mem的瓶颈

2018年开始，越来越多的企业开始使用阿里云服务，他们的业务场景对Pangu的延迟性能提出了更高的要求，为了更好满足业务的需求，阿里巴巴开始自研存储服务器，提供了单机97TB SSD和100Gpbs网卡的高性能服务器。硬件升级了，软件栈同样也要进行适配，从而打破CPU/MEM的瓶颈，软件栈主要做了如下优化：

- 通过技术手段减少网络写放大，并且提供动态调整流量优先级的策略
- 提供远端缓存访问技术(RDCA)
- 降低序列化反序列化开销，减少CPU消耗
- 引入了`CPU wait`指令同步超线程

通过使用自研的存储服务器，先比之前网络从2x25Gbps升级到2x100Gbps，并且解决了硬件升级造成软件层面存在的CPU/MEM瓶颈，Pangu 2.0的吞吐提升了6.1倍

Pangu架构

图1是Pangu的整体架构，整体看包含了三个部分：Pangu Service、Pangu Core和Pangu Monitor。

- Pangu Service：这一部分严格说不算是Pangu系统的范畴，指的Pangu系统的用户，有OSS/NAS/EBS等面向普通用户的产品服务。这里值得一提的是，除了传统的存储服务（OSS/EBS/NAS等），Pangu Service提供一个一个面向云原生的文件系统服务Fisc[1]，这也是现在云原生大潮下比较重要的一个服务，也发表在今年的FAST上。
- Pangu Core: 该部分是本文关注的Pangu系统部分，包含了Client、Masters和ChunkServers。对外提供了Append-Only的持久化文件系统。其中Client是提供给Pangu Service的SDK，负责接收调用方的文件读写请求，然后和Maters和Chunkservers交互完成数据的读写，Client是一个重SDK，其在整个系统中扮演着重要的角色，诸如多副本的写入，数据的一致性等都由其负责

- Pangu Mornitor: 整个系统的监控系统，对于任何系统都是必备组件，提供了实时的监控和基于AI的根因分析服务

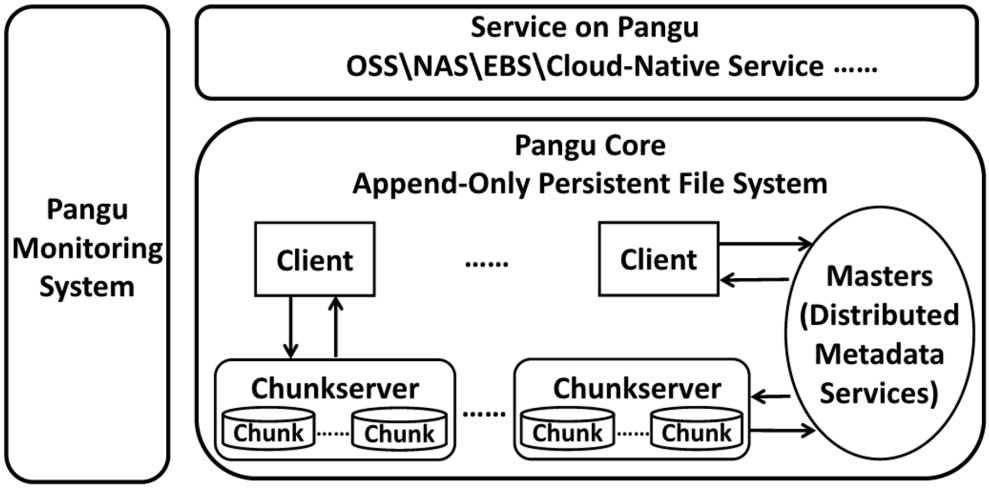

Figure 1: The architecture of Pangu.

这里重点介绍一下Pangu Core里的Masters和ChunkServer，也是Pangu系统里最重要的两个组件。

Masters主要负责管理所有元数据，使用Raft的协议维护元数据一致性。为了更好的水平扩展性和可扩展性（例如数百亿个文件），Pangu主节点将元数据服务分解为两个独立的服务：命名空间服务和Stream元数据服务。

- 命名空间服务提供有关文件的信息，例如目录树和命名空间
- Stream是Pangu中文件的代称有一些列的Chunk组成，一个Stream可以认为是可以无限append写入的大文件，Stream元数据服务提供文件到块的映射(即块的位置)，这个和微软的Azure Storage系统是相似的(传闻Pangu早期的操刀者也是由微软过来的同学完成)

ChunkServer则负责将数据的单机存储，其组织形式也是有一个一个的Chunk组成，并且使用用户态文件系统(USSFS)对Chunk进行管理。USSFS为不同硬件提供高性能的追加式存储引擎。在Pangu 2.0的第一阶段中，每个文件都存储在具有三个副本的块服务器中，后来垃圾回收工作程序（GCWorker）执行垃圾回收并使用纠删码（EC）存储文件。在Pangu 2.0的第二阶段中，逐渐将3倍复制替换为EC在关键业务（例如EBS）中，以减少Pangu中的流量放大

Pangu 2.0

设计目标：

- 低延迟：2.0旨在利用高性能的的SSD和RDMA硬件在存算分离架构下提供100us级别的IO延迟，并且能够提供ms级别的SLA保证，即便是存在网络抖动和服务器故障
- 高吞吐：充分发挥存储服务器的吞吐能力，提供极限吞吐
- 一致的高性能：为所有使用Pangu的服务，如在线搜索、数据分析、EBS、OSS和数据库，都能够提供一致的高性能服务

第一阶段：拥抱SSD和RDMA

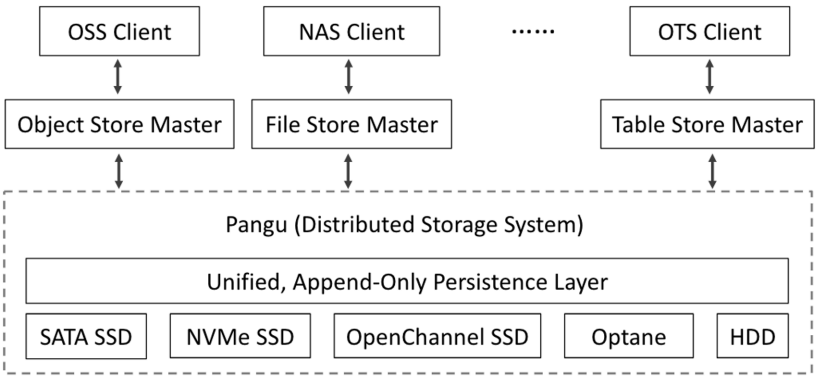

Figure 2: Various businesses are based on the unified persistence layer.

Append-only文件系统

Pangu 1.0为了满足业务的需求提供了多种不同语义的文件类型，如果Log类型、随机读写类型，后来团队观测到这些类型并不能充分发挥SSD的性能，而且每个服务都定制化的实现不同类型的接口，这是非常消耗人力而且极易出错的，所以在2.0的设计了简化了该设计只提供一个统一的Append-only的FlatLogFile类型，如图2。如此，通过顺序写入可以充分发挥SSD的吞吐和IOPS能力。

Heavyweight client

Pangu Client是一个很重的SDK，负责了从Maters获取元数据信息，然后从ChunkServer读写数据。从Masters获取到元数据后，就得到了Chunkserver的位置信息，然后还要负责复制协议，比如讲用户数据通过EC的方式写入到多个ChunkServer上，除此之外还提供了重试的机制以及backup read的能力。Client还提供了后台探测机制，周期性的探测Chunkserver的状态，这样在写入和读取的时候就可以有选择的规避有故障的ChunkServer。

Append-only Chunk Management

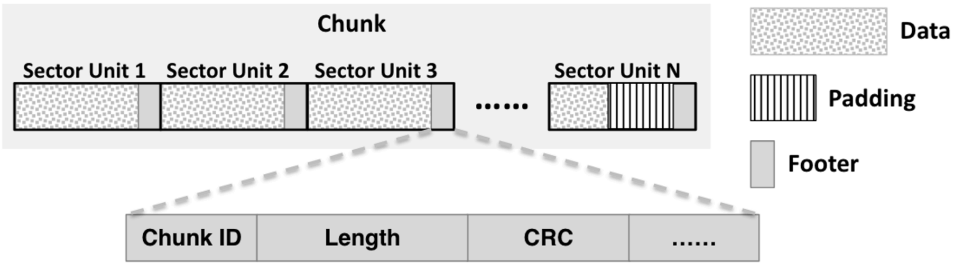

Figure 3: The self-contained chunk layout.

传统的文件系统，如Ext4，存储文件是以Block为单元，一个文件的数据和元数据分两次IO写入到存储介质，这样不仅增加了写入的延迟也缩短了SSD的使用寿命，因此Pangu 2.0设计了图3所示的一种自包含的Chunk Layout，每个Chunk自己保存了数据和元数据，这样在写入的时候只需要一次IO即可，这种设计还有一个好处是在故障的时候，可以自行恢复，因为元数据包含了ChunkID、Length和CRC等。同时，ChunkServer在内存还保留一份所有Chunks的元数据，并且周期性的进行Checkpoint到持久化存储，当故障发生的时候，Chunkserver可以对比Checkpoint中的元数据和Chunk Layout里保存的元数据，通过比对CRC从而恢复到一个正确的状态。

Metadata Operation Optimization

Masters提供了两种元数据服务：目录树和Stream管理。目录树比较好理解，就是文件系统的目录结构，即用户含有目录的文件和底层文件的映射，底层文件Pangu将其抽象为一个Stream，每个Stream有一系列Chunks组成，这些Stream与Chunks信息映射关系组成一个元数据服务。两层元数据服务使用不同的分片策略（比如目录树通过子目录划分，然后stream则通过哈希划分）划分为不同的分片，每个分片由raft协议保证一致性的元数据服务管理，这样保证了元数据服务的扩展性。在该架构下，Pangu 2.0还做了很多优化保证了元数据访问的性能和可用性：

- **元数据并行处理：**目录树服务和Stream服务都是用并行化处理的方式满足元数据访问的低延迟，这里可以参考阿里在FAST`22年发表的InifiFS[2]。同时Pangu也会将元数据尽可能得Hash到不同的元数据服务处理，除此之外，Pangu 2.0还是用了 一种可预测的目录ID的数据结构使其能够支持高效的并行处理路径解析（这里不是太懂？）
- **长度可变的Chunk大小：**Pangu 2.0选择使用增大Chunk的大小，他们认为这样有几个好处，第一可以减少元数据的数量；第二能够避免Client频繁的请求元数据；第三有利

于提高SSD的寿命。但是如果单单增大Chunk的大小就会导致产生很多碎片，因此他们引入了可变长的Chunk大小，支持1MB到2GB大小的Chunk。比如EBS服务Chunk大小的95分位是64MB，99分位是286.4MB。这种变长的设计可以有效降低碎片的概率，同时也和Pangu 1.0保持了兼容。（**这里有个疑问如果增大Chunk大小对数据的可靠性有什么影响，理论上这会影响存储的持久性SLA，因为Chunk变大了，那么其或者的副本分布的机器数就变少了，挂一块盘需要修复一个Chunk的时间就会变长，即修复一整块盘数据的时间会变长**）

- **在Client端缓存Chunk的信息：**每个Client都会维护一个本地的Meta缓存池，使用LRU的策略进行淘汰
- **Chunk请求批处理：**每个Client都会讲Chunk请求进行Batch处理，这样单个请求发送到Master，master可以并行处理然后将结果一并返回给Client
- **Chunk信息预取：**Pangu设计了一个贪婪、概率性的预取策略从而减少Chunk请求。当master收到Chunk请求的时候，除了返回请求的Chunk之外也会返回其他相关的Chunk信息(这个策略这里解释根据什么策略返回的)。如果收到写请求，则会返回比客户端请求更多的可用的Chunk，这样客户端写入失败的时候就不用去请求Master可以直接切换一个Chunk写入
- **Data Piggybacking减少一次RTT:** 合并Create Chunk和Write Chunk请求，Create Chunk直接携带用户写入的数据，通过一次请求完成Chunk的创建和数据的写入，这样可以减少一次RTT

ChunkServer 用户态文件系统(USSOS)

Pangu 2.0使用了高性能的SSD和RDMA网络，1.0使用的内核文件系统成为了短板，因为会带来大量的中断以及内核用户态的数据复制，因此利用kernel-bypass技术设计实现了一套用户态的文件系统，在线程模型上也使用了Run-to-Completion模型。

用户态的内存管理

Chunkserver的用户态文件系统基于RDMA的网络栈(DPDK)和SPDK的存储栈。在两种协议的基础上进行了如下的优化：

- 使用Run-to-Completion线程模型，减少上下文的切换和线程间的通信以及同步
- 通过用户态的共享内存减少内存拷贝。一个线程请求一块大内存后，这个共享内存会在网络栈和存储栈共享。比如用RDMA协议从网络收到数据后会放到一块共享大页内存

中，然后发送内存的地址和大小后，数据就可以通过SPDK直接写入到存储媒介中。通过这种方式可以减少数据的拷贝，同时一些后台的任务比如GC一样可以共享这块内存实现数据的零拷贝

用户态的调度机制

线上环境经常会遇到因为任务调度导致各种问题，Pangu 2.0引入了一个用户态的调度机制去进一步提高性能

- **防止一个任务阻塞后续的任务。**使用共享的大页内存可以实现用户空间网络和存储堆栈之间的零拷贝，但每个chunkserver只有固定数量的工作线程，新的请求根据请求中文件的哈希值分配到工作线程。如果一个任务需要太多时间（例如表查找、遍历、内存分配、监控和统计），它会占用资源并阻塞后续任务，降低请求的性能并导致其他请求饿死。为了解决这个问题，对不同情况采取不同措施，例如对于一些长任务，引入心跳机制监控任务的执行时间并设置警报，如果一个任务运行时间太长，那么就会被放入后台线程执行，当前线程就被出让出来执行其他任务
- **优先级调度保证QoS。**对不同的请求赋予不同的优先级，通过优先级队列保证高优先级的任务能够早于低优先级的任务执行
- **Polling和evnt-drivent切换(NAPI)。**USSOS在Polling和event-driven模式之间提供了一种切换机制从而减少中断处理带来的CPU损耗。具体讲就是应用默认是event-driven模式，当收到一个通知后就会切换到polling模式，polling一段时间没有新的IO请求后会再次切换到event模式

Append-Only USSFS

为了充分发挥SSD的吞吐和延迟，USSFS只提供了Append-Only语义的FlagLogFile。不同于POSIX的Ext4，其只提供Chunk级别的语义，如open, close, seal, format, read, and append。这样可以充分利用SSD的顺序写优势，同时也提供了随机读的能力。同时也使用了一些其他机制最大化其性能：

- 通过使用自包含的Chunk Layout可以减少数据操作的次数，也去除传统文件系统Page Cache和Journal等开销
- 去除层次化的Inode和文件目录语义，所有的操作的操作都记录在日志文件中，通过日志文件的会发可以重建元数据
- 使用polling模式替代ext4使用的中断模式，最大化ssd的性能

高性能SLA保证

Pangu引入很多机制去确保高性能以及ms级别的P999 SLA：

- **Chasing机制。**这个机制可以减少系统抖动对写延迟的影响，在写入数据副本的时候设置MinCopy和MaxCopy，通常2xMinCopy > MaxCopy（不如MaxCopy为3，MinCopy为2），假设其中一个副本发生了异常，那么写入只需要写入MinCopy就返回用户成功。假设两个正常副本写入成功时的时间为T，Client已经返回用户写入成功，但是依然会将该Chunk保存在内存中，等待时间`t`(ms级别)，如果异常的副本在`T+t`前返回成功，那么Client就会从内存中将该chunk释放。如果没有，但是没有写入成功的数据大小小于`k`，那么client就会重试写入，如果没有写入成功的数据大于`k`，那么Client就会将失败的副本Seal掉(禁止后续写入)，然后通知masters修复该副本。**通过他们分析发现小心的选择`t`和`k`可以额在减少写入尾延迟的同时不带来任何数据丢失的风险。**
- **Non-stop write。**客户端在写入Chunk失败时，会seal掉该Chunk，并把已经写成功的数据长度上报给Masters，重新选择一个新的Chunk进行写入。如果sealed的chunk数据有损坏，那么会有后台修复将该Chunk从没有损坏的Chunk修复一份出来
- **Backup Read。**这个机制是为了减少系统抖动情况下的读延迟。一个chunk有多个副本，当下发读请求到一个副本时，等待一段时间`t`，如果请求还没有成功，那么会马上发送`n`个请求到其他副本，哪个请求先回来就返回给用户，这些后发出去的请求称为backup read。`t`和`n`的选择Pangu会根据不同类型的磁盘和io大小动态进行计算和调整，也会根据系统的负载限制其数量
- **Backlisting。**为了避免请求下发到异常的Chunkserver导致延迟增加，引入了确定性黑名单和非确定性黑名单。当一个chunkserver被认为无法提供服务时（例如，chunkserver的SSD已损坏），该服务器将被添加到确定性黑名单中。如果一个chunkserver可以提供服务，但其延迟超过一定阈值，它将被添加到非确定性黑名单中，并且加入黑名单的概率随着其服务延迟的增加而增加。客户端定期向黑名单中的服务器发送I/O探测请求以释放它们。同时也限制了黑名单上服务器的总数以确保系统的可用性。对于每个服务器，它引入了一个宽限期来添加/删除它们到/从黑名单中以维护系统的稳定性。此外，由于TCP和RDMA链接中的故障服务器可能不同，因此Pangu分别维护TCP和RDMA链接的黑名单，并在两个链接上进行I/O探测以更新它们

第二阶段：适应面向性能的业务模型

Pangu 2.0阶段转向了性能的优化，为了适应业务对性能的需求，自研了名为泰山的存储服务器，每台泰山服务器配备了2x24C的Skylake CPU，12x8TB的商用SSD，128GB的DDR内存以及2x 100Gbps网卡。并且通过SSD制造的优化(缓存和channel等)，单台泰山服务器的SSD吞吐能够达到20GB/s。

升级了硬件后，CPU、Memory和网络成为了新的瓶颈，因此2.0阶段主要是进行软件栈的优化打破这些瓶颈。

网络瓶颈

从25Gbps升级到100Gbps增大带宽容量，同时在硬件和软件上也要做出相应优化：

- **硬件上：**采用高性能的NIC/RNIC、光模块（QSFP28 DAC、QSFP28 AOC、QSFP28等），单模/多模光纤和高性能交换机
- **网络软件栈：**早期为了大规模部署RDMA，采用了关闭NIC端口或在RDMA网络上有太多暂停帧时（例如几秒钟）临时切换到TCP等各种机制，但是这些机制不能处理基于暂停帧的流量控制的其他问题（例如死锁和头部阻塞，因此又升级到有损RDMA，禁用暂停帧，以避免这些问题并提高性能

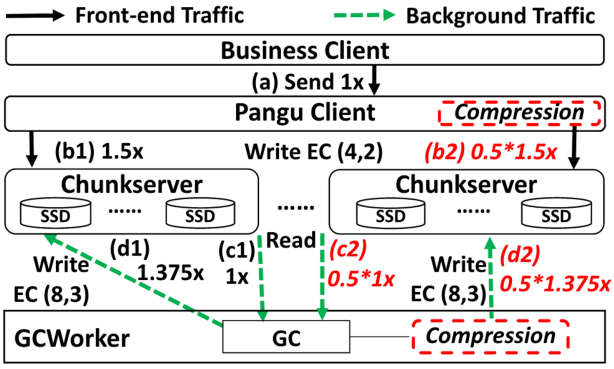

Figure 9: Pangu optimizes network traffic with 3 techniques: EC, compression, and balance between background and front-end traffic. The entire life cycle of a file we define is as follows. First, the business client sends a file (step a) to the Pangu client. Second, the Pangu client writes the file to the chunkserver in way b1 (b2 after compression). Third, the GCWorker reads the file from the chunkserver in way c1 (c2 after client compression) and performs garbage collection. Finally, the GCWorker writes the file in way d1 (d2 after GCWorker compression).

流量优化，减少网络流量的放大，如图9所示是EBS场景下数据写入的放大：

(a) 客户端发送数据到Pangu Client ----- -------- 1x

(b) Pangu Client写三副本到ChunkServer ------- 3x

(c) 后台GC读数据进行处理 ---------- 1x

(d) 然后GC后将数据EC(8.3)写入ChunkServer --- 1.375x

最终写入一份数据的流量放大为6.375x (1x+3x+1x+1.375x)，也就是说对于100Gbps，其实际网络带宽为16Gbps。为了解决这个问题，Pangu 2.0通过EC和数据压缩的方式减少网络流量的放大

- **使用EC替代3副本。**用EC(4,2)替代第(b)步的三副本，放大从3x减少到1.5x，最终放大为4.875x。EC对于小IO不是很友好，存在zero-padding的问题，为了解决这个问题Pangu允许小IO聚合写入以及动态切换EC和3副本。除此，Pangu将EC库Jerasure替换为Intel ISA-L，延迟也降低了2.5~3倍
- **压缩FlatLogFile。**在GC和Client写入数据的时候使用LZ4对数据进行压缩，线上数据分析平均压缩率可以到50%，这样(b)和(c)步放大分布变为0.5*1.5x和0.5*1.375x，整体的放大就降为了2.9375x
- **动态为前端和后台流量分配网络带宽。**如果集群还有很多空闲流量，那么可以调低后台GC的带宽占用阈值，还有比如夜间调大GC带宽使用阈值，白天减小阈值等

内存瓶颈

内存瓶颈发生在网络进程和应用进程对内存带宽的竞争，如网卡不能够申请到足够的内存带宽，就会对PCIe产生严重的反压，最终导致网卡buffer积压网络包，最终丢弃溢出的包，进而触发拥塞控制机制导致系统整体性能恶化，如吞吐降低30%，延迟增加5~10%，IOPS也会降低10%。针对这个问题，Pangu采用下面三个措施：

- **增加小容量的DRAM扩充内存带宽。**因为内存容量不是瓶颈，因此可以增加小容量的DRAM去充分利用内存通道增加单台机器的内存带宽
- **后台流量从TCP切换到RDMA。**早期因为25Gbps RDMA网络只有一个硬件队列，因此后台流量都是使用了TCP，随着网络升级到100Gbps，后台流量也可以切换打RDMA，相比TCP，网络带宽需求可以降低75%。为了保证前端流量的Qos，设计了一种类似Linux tc的机制控制后台流量
- **远端缓存直接访问。**Pangu团队发现数据离开网卡后在内存中时间非常短（平均几百微秒）。假设在内存中停留的时间平均为200微秒的，对于双端口100 Gbps的网卡，只需要5 MB的临时存储来存储离开NIC的数据，因此他们使用英特尔的DDIO在商用硬件上

实现了RDCA(Remote Direct Cache Access)使得发送这可以绕过接受者的内存直接访问Cache。在盘古的一些集群上的广泛评估结果表明，对于典型的存储工作负载，RDCA每台服务器消耗12MB LLC缓存(占总缓存的20%)，平均内存带宽消耗减少了89%，网络吞吐量提高了9%。RDCA在非存储工作负载中也很有效，例如，在延迟敏感的HPC应用程序中，它将集合通信的平均延迟降低了多达35.1%

CPU瓶颈

对网络和内存优化后，100Gbps网卡下吞吐依然只达到了理论值的80%。主要是因为序列化/反序列化、数据压缩和CRC计算都消耗了大量的CPU资源，使得CPU称为了瓶颈。为了解决这个问题，Pangu从下面三个方面进行了优化：

- **混合RPC。**Pangu团队发现RPC请求中序列化/反序列化消耗了30%的CPU，因此他们对数据链路使用类似FlatBuffer替代Protobuf，但是在控制链路依然使用ProtoBuf，最终优化后单核CPU网络吞吐增加了59%
- **使用`CPU wait`指令支持超线程。**引入cpu `wait`指令解决超线程存在的同一个物理核线程切换的开销以及两个线程同时执行相互干扰带来的性能退化，最终单核CPU网络吞吐增加了31.6%
- **软硬件系统设计。**引入可编程FPGA硬件，将数据压缩和CRC计算卸载到FPGA进行，同样的网络吞吐下，可以节省30%的CPU开销[3]

其他优化案例

- 进行大量的数据完整性校验。通过CRC确保数据的完整性，比如端到端的CRC、不同副本间CRC、随机副本CRC等
- 处理一些影响SLA的异常抖动。比如TCMalloc在申请内存时候如果触发了从全局内存申请则会产生延迟抖动，因此他们引入了用户空间的内存分配池，优化RDMA驱动使用匿名页。其次一些周期性的任务放到后台执行，比如打印日志，还有比如内存占用过高时候进行内存回收会导致很高的CPU使用率，因此调整了内存回收阈值减少回收的机会
- 处理USSOS中的可纠正机器检查异常(MCE)以提高可用性。增加后台守护进程去检测和处理MCE错误，一旦发现就会通知内核迁移故障的内存
- 对内存性能进行评估，不同供应商的内存带宽差异很大，如果都加入同一个集群可能带来延迟的抖动

经验教训

用户态系统

- 用户态系统的开发和运营比内核空间系统更简单。在Pangu的数据中，内核空间文件系统的错误修复需要约两个月的时间，而在USSFS中只需要几周的时间
- 开发用户空间系统应该借鉴内核空间的设计。为了构建高性能的USSOS，不仅需要统一存储和网络堆栈，还需要设计用户空间模块来管理内存、CPU调度和硬件故障处理
- 用户空间系统的性能提升不仅适用于高速存储设备，如SSD。Pangu的USSFS中提供了一系列机制来加速HDD的性能，例如利用自包含的块布局来减少元数据操作次数，利用磁盘内部和外部轨道的差异来提高写入效率

性能和成本

- 为了满足新的业务需求，Pangu通常首选添加更多硬件来提高性能，基于总拥有成本（TCO）平衡考虑（例如，将网络从25 Gbps升级到100 Gbps，通过增加更多小容量DRAM和升级更强大的CPU来增加内存通道数量）
- 硬件扩展有效地提高了Pangu的性能，但由于产生的成本不可持续，因此Pangu还花费了大量精力，例如流量优化，提高其资源利用率和效率
- 在性能和成本之间进行权衡是必要的，Pangu通过定期评估业务需求和硬件资源的使用情况来确保平衡

PMem

Pme确实有很多优点，Pangu基于Pmem也开发了30us的EBS服务，但是PMem最终还是被Intel砍掉了，所以在开发新服务时，需要更加深入地考虑替代性、可持续性和成本权衡等因素

硬件卸载

- 硬件卸载的成本与收益权衡。Pangu硬件卸载压缩的整个开发过程用了20人团队两年时间，期间也解决了许多问题，如FPGA硬件成本、压缩数据的完整性以及与硬件中的其

他功能共存等问题

- 硬件卸载的结果收益大大超过了成本。硬件卸载显著降低了压缩的平均延迟和尾延迟，有效降低了低延迟内的网络流量，服务的整体能力提高了约50%
- 内部先使用，然后逐步扩展到核心服务，整个过程中为了防止可能存在的硬件错误损害数据完整性，在硬件上执行数据解压缩和CRC检查，定期的进行软件CRC检查
- Pangu的所有200 Gbps集群默认启用硬件压缩，事故发生的次数越来越少

参考

1. Fisc: A Large-Scale Cloud-Native-Oriented File System
2. InfiniFS: An Efficient Metadata Service for Large-Scale Distributed Filesystems
3. OLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database