第10章 存储I/O专栏

正确描述IO类型

原创 EMC中文技术社区 戴尔易安信技术支持 *2016-06-03*

不同应用通常具有不同的I/O类型，了解应用的I/O类型是为其设计解决方案、排错性能问题的首要工作。那I/O类型通常包括哪些需要考虑的因素？我们今天就来谈一谈I/O类型的几个重要方面。

读 vs. 写

应用程序的读写请求必须量化，了解他们之间的比列，因为读写对存储系统的资源消耗是不通的。了解读写比率直接关系到如何应用缓存、RAID类型等子系统的最佳实践。写通常需要比读更多的资源，SSD的写操作相对读更是慢得多。

顺序 vs. 随机

传统存储系统通常都是机械硬盘，因此整个系统设计为尽可能顺序化I/O，减少由于磁盘寻道所带来的延迟。所以，顺序I/O相对随机I/O的性能会好很多。随机小I/O消耗比顺序大I/O更多的处理资源。随机小I/O更在意系统处理I/O的数量，即IOPS；而顺序大I/O则更在意带宽，即MB/s。因此，如果系统承载了多种不同的应用，必须了解它们各自的需求，是对IOPS有要求，还是对带宽有要求。这往往需要在两种之间进行折衷考虑。闪盘是一个例外，它没有机械寻道操作，因此对随机小I/O的处理是非常迅速的，由此是读操作。

大I/O vs. 小I/O

我们通常把<=16KB的I/O认为是小I/O，而>=32KB的I/O认为是大I/O。就单个I/O来讲，大I/O从微观的角度相比小I/O会需要更多处理资源，不过对于智能存储系统来说，会尽可能把I/O整理为顺序的，以单个操作执行，如此依赖，将多个小I/O整理成单个大I/O处理后，反而会更快。I/O的大小依然取决于应用程序本身，了解I/O的大小，影响到后期对缓存、RAID类型、LUN的一些属性的调优。

位置引用

数据的位置分布影响到后期对二级缓存或存储分层技术的应用，因为这些技术都会根据I/O的位置分布来判断是否将I/O放置到缓存或快速的层级。位置引用是指那些被频繁的存储位置，我们通常认为最新创建的数据以及最近被访问过的数据，它们周围的数据也同时被访问的可能性会比较大。因此，了解应用程序的I/O位置特性，有助于应用正确的性能优化技术。

稳定 vs. 爆发

I/O数量在一天中的不同时段会有不同的表现。例如，早高峰时段的I/O数量相比下班后的I/O会多出许多。如果能准确预测和估计应用的I/O在不同时间段的稳定性和爆发性，可以正确分配资源，提高资源利用率。在前期的设计阶段，就应该考虑系统是否能够处理I/O高峰期。

多线程 vs. 单线程

多线程是实现并发操作的一种方式，同时也意味着对存储系统的资源消耗更多。这种高IOPS的请求方式，在有些情况下会造成磁盘繁忙，进而导致I/O排队，增加了响应时间。因此，适度的调整线程数量，不仅可以实现并发，而且能在不拖累整个存储系统的情况下，达到最优的响应时间。

关于不同应用程序存储IO类型的描述

原创 EMC中文技术社区 戴尔易安信技术支持 *2016-05-11*

存储系统作为数据的载体，为前端的服务器和应用程序提供读写服务。存储阵列某种意义上来说，是对应用服务器提供数据服务的后端“服务器”。应用服务器对存储系统发送数据的“读”和“写”的请求。然而，不同的应用程序对存储的数据访问类型有所不同。本文描述典型的不同应用程序的存储IO类型。帮助读者了解不同应用程序存储IO类型的同时，提供的数据也可以为存储模拟和压力测试的数据参考。

IO类型描述:

描述不同应用的存储IO类型之前，先要描述存储中的定义IO的几个术语：

1. IO大小（IO Size）：IO Size是应用程序发起，经过操作系统的磁盘子系统，向存储系统发送的读写请求的单位大小。不同的应用程序所发送的IO大小都不相同，例如对于数据库应用，它在数据读写的时候IO Size是8KB，而在事务日志的写入的时候可能是512Bytes-64KB不等。所以，通常所说的IO Size都是一个平均的概念。即某一款应用在一段时间内的平均IO大小。
2. 读写比例（Read/Write）：读写比例比较容易理解，就是应用程序读数据和写数据分布。这个在规划存储的时候也至关重要，因为存储系统中的保护级别（RAID）的不同，对写有损失。例如RAID-5单次写入需要分别对数据位和校验位进行2次读和2次写。所以说，如果用RAID-5作为写入比例较高的应用，显然会对性能有很大影响。
3. 顺序与随机读写比例（Random/Sequential）：顺序和随机读写取决与应用的获取数据的方式。通常情况下，如果数据的读取和写入是在连续的磁盘空间上，可以认为是顺序读写。如果应用读取的数据分布在不同磁盘空间，且无固定的顺序，则视为随机读写。由于传统的机械磁盘（闪存盘不再讨论之列）读写数据需要盘面的转动和磁头的移动，这使得随机读写的效率在物理磁盘层面要远小于顺序读写。通常存储系统都都会利用缓存来减少这部分的延迟，减缓因为磁头的移动而带来的性能损失。随机读写的代表的是OLTP的数据库文件，顺序读写的代表则是数据的事务日志。

应用程序存储IO类型：

下面的表中描述的不同的应用程序对应的IO大小、读写比例、随机和顺序比例。表中的比例为一个通用的参考值，比例接近真实各种应用的IO类型。当然不能包含全部的应用类型因为根据不同生产环境，数值也会有很大的差异。这里的数据提供一个参考，可以用于使用压力测试工具，例如IOMeter（IOMeter用法可以参考文章：Iometer学习笔记），模拟不同应用的IO负载。

应用类型	IO大小	读写比例	随机与顺序读写比例
Web File Server	4KB、8KB、64KB	95%读/5%写	75%随机/25%顺序
Web Server Log	8KB	100% Write	100%顺序
OS Paging	64KB	90%读/10%写	100%顺序
Exchange Server	4KB	67%读/33%写	100%随机
Workstation	8KB	80%读/20%写	80%随机/20%顺序
Media Streaming	64KB	98%读/2%写	100%顺序
OLTP - Data	8KB	70%读/30%写	100%随机
OLTP - Log	512bytes - 64KB	100%写	100%顺序

数据库存储I/O类型分析与配置

原创 EMC中文技术社区 戴尔易安信技术支持 *2016-06-11*

存储设备作为数据的容器，为应用提供数据存取服务，而存储系统将数据展现给不同的应用后，应用程序对数据访问不尽相同。简要来说，就是读和写，更加细分的话是以不同的传输单元（I/O大小）进行顺序和随机类型的读写。不同应用的I/O访问差异，直接决定了存储系统对它们的服务方式的不同，了解主流应用的I/O访问类型，是存储性能规划和调优的基础。

本文作主流数据库系统的存储I/O访问类型与配置首篇，罗列了MS SQL Server相关，作者所能收集到的SQL Server I/O访问类型与特点，具体到不同的操作的I/O与配置文档列表，供读者查阅和参考。

SQL Server运行过程中，会不断与存储端做交互，但不是每一个操作都会产生存储I/O。鉴于SQL Server会直接使用主机内存作为缓存，所以在SQL Server中，对数据的读写又分为SQL Server自身缓存中进行的Logical I/O和实际与存储系统交互的Physical I/O。本文所描述的存储I/O，即SQL Server中实际与存储交互的Physical I/O操作。在讨论SQL Server的I/O类型之前，我们先看一下SQL Server在哪些情况下会产生存储I/O（Physical I/O）：

- SQL Server内存中没有缓存的数据，第一次访问数据需要从数据文件中加载的时候。
- 任何数据库插入、更新、删除提交之前，把日志记录写入日志文件的时候。对于Select类似的读操作，如果数据在主机内存中有缓存，则不会产生存储I/O。
- SQL Server进行Checkpoint的时候，将缓存中的数据写入数据文件时候。
- 当SQL Server缓存存在压力，触发Lazy Write将页面刷新到存储设备的时候。
- 执行特殊操作，例如CHECKDB、Reindex、Update Statistics、数据备份的时候。

SQL Server对存储中I/O操作是以8KB的页面为最小处理单位。因为SQL Server只能运行在Windows服务器上，而Windows的I/O大小与类型直接由应用发起来决定。所以，SQL Server的I/O类型与大小会相对比较固定。下表中我们给出SQL Server不同类型的操作所对应的I/O类型与大小：

SQL Server操作	**随机/顺序**	**读/写**	**I/O大小**
OLTP – 日志	顺序	写（除了恢复的时候）	512 bytes – 64KB，平均8-9.5KB
OLTP – 数据	随机	读/写	8KB – 64KB
Bulk Insert	顺序	写	大多数8KB，最大128KB
Read Ahead （DDS、Index Scan）	顺序	读	大多数8KB，最大256KB
备份	顺序	读/写	1MB
恢复	顺序	读/写	64KB
Reindex	顺序	读/写	多数是8KB，最大256KB
创建数据库	顺序	写	512KB

由上表可见，SQL Server的主要操作多以8KB的I/O大小为主，而某些顺序I/O类型的操作，会使用比较大的I/O大小，原因可想而知：为了提供更大的Throughput。SQL Server的读I/O基本只位于数据文件所在的LUN上面。SQL Server中的写I/O分为两种，数据文件与日志文件，这些写I/O的数目，则完全取决于数据更新的频率。

SQL Server对于不同类型的I/O，每次进行实际操作I/O大小会存在少量差异。 例如对于表的索引查询，绝大多数以8KB为单位，而像一些整表查询，则可能以64KB甚至更大的单位比较多。至于何时会使用多大的I/O，没有一个绝对的情况。只能说取决于SQL Server的内部机制，作者也没有找到相应的资料来描述这种行为。

表中的读/写一栏显示了I/O读与写的重点。表中没有列出读/写比例，是因为这个比例则需要根据应用而定，不同的应用之间差别比较大。

根据表中的I/O类型与特点，总结几点存储的配置建议：

- SQL Server的不同数据文件，如果分散存放在不同的LUN上，可以使得SQL Server的读写I/O由若干个LUN共同完成，达到并发处理的效果，提升性能。
- 衡量SQL Server的OLTP和OLAP（DDS）存储性能的指标是不同的，前者看中的IOPS，而后者则是Throughput，所以规划的重点也不尽相同。（参考：论存储IOPS和Throughput吞吐量之间的关系）
- 鉴于SQL Server不同操作，读/写的重点的不同，理解通用RAID的类型的写惩罚机制，对规划数据文件和日志文件存储，选取合适的RAID保护级别会有比较大的帮助。（参考：浅谈RAID写惩罚（Write Penalty）与IOPS计算）
- 根据所使用的存储阵列的特点，参考厂商所提供的存储阵列配置文档，根据情况启用适当的存储功能，例如配置Write Cache/Read Cache的比例，使用FAST等技术实现存储分层都是针对SQL Server存储配置中可以参考的技术实现。
- 配置存储过程中，建议使用合适的工具对磁盘进行压力测试，根据实际应用情况模拟存储I/O，检验存储的性能是否满足需求。

综上所述，本文的内容总结为以下几点：

- 虽然SQL Server主机的内存大小会决定存储I/O的数量，缓存页面的数量越多，就越能减少存储端发生的读I/O数目。但是如果存储读性能够好，同样可以缓解SQL Server端的缓存压力。所以，在SQL Server本身存储I/O数量不能继续优化的情况下，利用存储来提升性能是可选方案。
- SQL Server写I/O的数目，完全取决于数据的更新频率，如果数据更新频繁，日志文件存储的LUN会受到比较大的压力。 同理，使用写入性能够好的存储，用作日志文件的部署，可以很大程度提升整体SQL Server的存储性能。另外，适当RAID配置，以及启用一些存储端功能，例如调整写缓存的比例，FAST分层存储等等，也不同程度帮助写I/O提高存储响应速度。
- 对于SQL Server的存储配置没有一劳永逸的做法，而且数据库系统的高定制化特点决定了服务于不同应用系统的存储I/O访问类型会有很大区别。DBA和存储管理员需要根据自身应用的特点，结合观察数据库系统的存储行为，参考文本中给出的数据与配置文档，结合实际应用的存储负载才能更好的完成存储规划与性能调优，从而满足应用与数据库的应用需求。

最后，作为一篇整理与总结的文章。本文试图收集更多的参考资料，用来描述主流数据库系统 MS SQL Server的I/O类型，并给出配置的建议与参考文档。由于信息来源于网络与其他厂商的官方文档。内容整理可能存在缺失与不准确，如果有任何遗留或者错误，希望广大读者指正。

浅析I/O处理过程与存储性能的关系

原创 EMC中文技术社区 戴尔易安信技术支持 *2016-06-05*

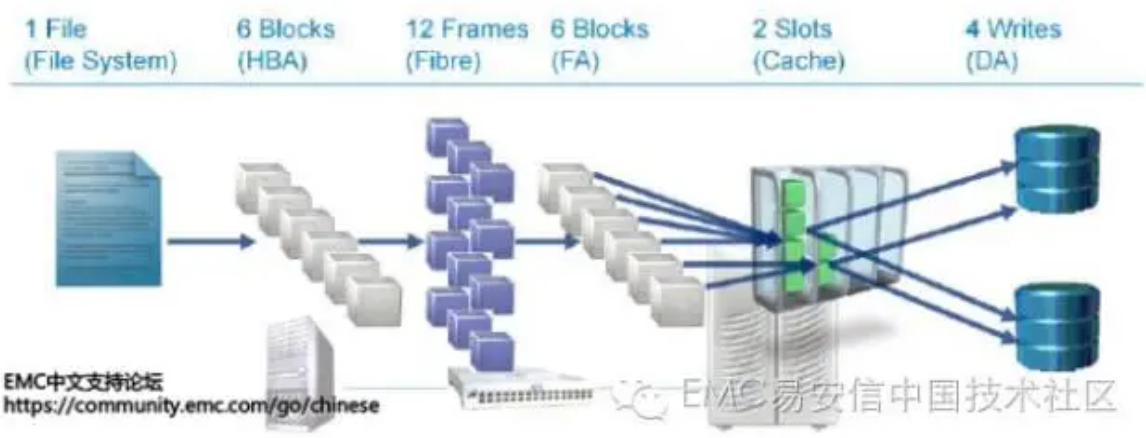
“性能”这个词可以说伴随着整个IT行业的发展，每次新的技术出现，从硬件到软件大多数情况下都围绕着性能提升而展开。“摩尔定理”指出CPU的处理速度每18个月会翻一番，但是进入21世纪的第二个十年来，似乎它的速度慢了下来。但是IT行业的各个行业领导者们，还是不断在计算机的性能寻求突破，继续挑战物理极限。细看存储行业，每款新的存储产品的推出，也围绕着如何更快、更好的服务前端服务器的I/O请求为中心。本文从I/O（Block）的流向介绍，试图解读整个I/O流与存储性能之间的些许联系。

本文作为一篇存储基础的介绍文章，帮助读者了解看似简单的数据读写中的更多细节。

存储I/O流与存储性能:

存储I/O（后文简称I/O）的处理过程就是计算机在存储器上读取数据和写入数据的过程。这种存储器可以是非持久性存储（RAM），也可以是类似硬盘的持久性存储。一个完整的I/O可以理解为一个数据单元完成从发起端到接收端的双向的过程。在企业级的存储环境中，在这个过程会经过多个节点，而每个节点中都会使用不同的数据传输协议。一个完整的I/O在每个不同节点间的传输，可能会被拆分成多个I/O，然后从一个节点传输到另外一个节点，最后再经历相同的过程返回源端。

下图演示了一个文件在经过整个I/O路径中每个节点所进行的变化（以EMC Symmetrix存储阵列为例）：



整个I/O流经历一下几个节点:

- File System – 文件系统会根据文件与Block的映射关系，通过File System Manager将文件划分为多个Block，请求发送给HBA。
- HBA – HBA执行对这一系列的更小的工作单元进行操作，将这部分I/O转换为Fibre Channel协议，包装成不超过2KB的Frame传输到下一个连接节点FC Switch。
- FC Switch – FC Switch会通过FC Fabric网络将这些Frame发送到存储系统的前端口（Front Adapter）。
- Storage FA – 存储前端口会将这些FC 的Frame重新封装成和HBA初始发送I/O一致，然后FA会将数据传输到阵列缓存（Storage Array Cache）
- Storage Array Cache – 阵列缓存处理I/O通常有两种情况：1.直接返回数据已经写入的讯号给HBA，这种叫作回写，也是大多数存储阵列处理的方式。2. 数据写入缓存然后再刷新到物理磁盘，叫做写透。I/O存放在缓存中以后，交由后端控制器（Disk Adapter）继续处理，完成后再返回数据已经写入的讯号给HBA。
- Disk Adapter – 上述两种方式，最后都会将I/O最后写入到物理磁盘中。这个过程由后端Disk Adapter控制，根据后端物理磁盘的RAID级别的不同，一个I/O会变成两个或者多个实际的I/O。

根据上述的I/O流向的来看，一个完整的I/O传输，经过的会消耗时间的节点可以概括为以下几个：

- CPU – RAM， 完成主机文件系统到HBA的操作。
- HBA – FA，完成在光纤网络中的传输过程。
- FA – Cache，存储前端卡将数据写入到缓存的时间。
- DA – Drive，存储后端卡将数据从缓存写入到物理磁盘的时间。

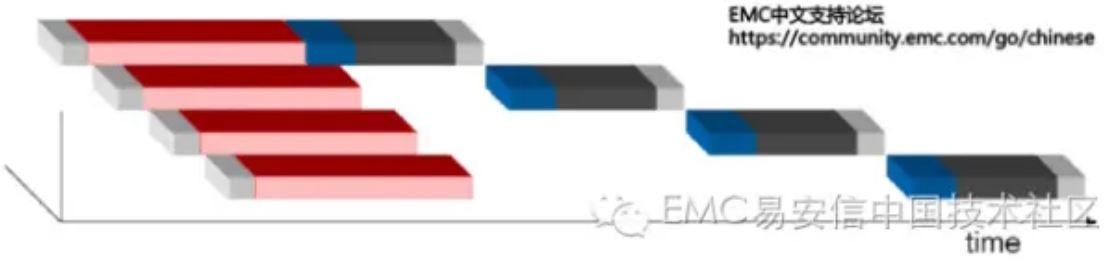
下面的表中根据不同阶段的数据访问时间做了一个比较，一个8KB的I/O完成整个I/O流向的大概耗时。（表中的耗时根据每秒的传输数据整除获得，例如HBA到FA的速度有102,400KB/秒除以8KB得到78 μs）。根据表中的数据显而易见，I/O从主机的文件系统开始传输到存储阵列的缓存在整个这个I/O占比很小，由于机械硬盘的限制，最大的耗时还是在DA到物理磁盘的时间。如果使用闪存盘，那这个数据会大幅缩小，但是与其他几个节点的传输时间相比，占比还是比较大的。

8KB I/O的传输	**耗时（单位：μs）**	**扩大1,000,000耗时**
CPU到RAM的传输，166MHz 64位总线	6 μs	6 sec
HBA到FA，100MB/秒速率	78 μs	1 min 18 sec
FA到Cache（Symmetrix DMX Director 2.5GHz）	26 μs	26 sec
DA到物理磁盘，73GB 15,000 rpm, Seek 3.6ms，latency 2.0ms	5,700 μs	1 hr 35 min

可以看到，存储阵列的缓存在整个I/O流中所起到的作用是至关重要。缓存的处理效率与大小，直接影响到I/O处理的速度。而然，在实际的环境中，即使存储阵列的缓存工作得当，主机的I/O也不会达到100 μs也就是0.1ms的水平，通常在1-3ms左右，就会认为I/O处理处于比较高性能的模式。原因就是因为另外两个因素“数据头处理”和“并发”。

\1. “数据头处理”由于I/O流中每个I/O的数据组成并不是只包含数据，如下图所示，一个I/O除了数据以外还包含了Negotiation，Acknowledgement用来负责在I/O流中的每个节点传输和进行管理的。其中包含和TCP/IP一样的“Handshaking”信息以及流控制的信息，比如初始化传输，结束通讯等等。Header中则会定义一些例如CRC校验的信息，保证数据的一致性。所有这些数据的处理都会耗费一定的处理资源，增加I/O流的耗时。

2.“并发“。由于I/O流整个过程中不可能只同时处理一个I/O，所有的I/O在HBA，FC，FA和DA处理的过程中都是已大量并发的情况下进行。而主要的耗时取决于I/O队列的等待，虽然存储阵列会在并发上进行优化。同一个处理Slice的处理还是会一队列形式进行。入下图所示，当存储同时面对多个I/O的处理的情况，总会有某个I/O会在整个流的最后出来，而增加I/O的耗时。所以说，在I/O流的每个节点出现瓶颈，或者短板的时候。I/O的耗时就会增加。



综上所述，I/O流与存储性能的关系可以总结为以下几点：

- 完成一个I/O流主要经历过的节点有HBA，FC网络，存储前端口FA，存储缓存、存储后端口，物理磁盘。而很个过程中最耗时的是物理磁盘。
- 存储阵列的缓存的大小和处理方式直接影响到I/O流的性能，也是定义一个存储阵列优劣的重要指标之一。
- I/O的处理速度通常会远离理论值，原因多个并发量较大而造成的队列延迟。
- 优化I/O的方式可以从多个节点入手，而最显著的效果是提升物理磁盘的速度。因为存储阵列会把尽可能多的数据放入缓存，而当缓存用满以后的数据交换则完全取决于物理磁盘的速度。
- 适当选用合适的RAID级别，因为不同的RAID级别的读写比例大不相同，可能使得物理磁盘处理耗时几倍增加。参考：浅谈RAID写惩罚（Write Penalty）与IOPS计算

存储系统性能 - 带宽计算

原创 EMC中文技术社区 戴尔易安信技术支持 *2016-06-10*

遇到过很多同行、客户问我：“xxx存储系统究竟最大支持多少【IOPS】？”，这真不好说，因为手里确实没有测试数据。更何况，IOPS与i/o size、random/sequential、read/write ratio、App threading-model、response time baseline等诸多因素相关，这些因素组合起来便可以描述一种类型的I/O，我们称之为【I/O profile】。不同的因素组合得到的IOPS都不一样，通常我们看到的【标称IOPS】都是在某一个固定组合下测得的，拿到你自己的生产环境中，未必能达到标称值。这也是为什么要做前期的performance analysis/sizing的缘故。

直到有人这样问我：“xxx存储系统究竟最大支持多少【带宽】？"我愣了下，仔细想想，硬件性能极限就摆在那，基于bandwidth = Frequency * bit-width，而且很多需要的数据都是公开的，东拼西凑应该可以算出个大概。

我并不是Performance专家，从未做过Performance Consulting/Sizing方面的工作，最多也只是做过性能方面的分析/排错，所以这篇文章的准确性多半存在不靠谱的地方，读者斟酌着看吧。

在读文章之前，建议先看一下如下计算公式和名词。

计算公式：

- Real-world result = nominal * 70% -> 我所标称的数据都是*70%（性能计算：Little Law & Utilization Law）以尽可能接近实际数据，但如果另外提供了由资料获得的更为准确的数据，则以其为准。
- Bandwidth = frequency * bit-width

QPI带宽：假设QPI频率==2.8 Ghz

× 2 bits/Hz (double data rate)

× 20 (QPI link width)

× (64/80) (data bits/flit bits)

× 2 (unidirectional send and receive operating simultaneously)

÷ 8 (bits/byte)

= 22.4 GB/s

术语：

- Westmere -> Intel CPU微架构的名称
- GB/s -> 每秒传输的byte数量
- Gb/s -> 每秒传输的bit数量
- GHz -> 依据具体操作而言，可以是单位时间内运算的次数、单位时间内传输的次数 (也可以是GT/s)
- 1byte = 8bits
- IOH -> I/O Hub，处于传统北桥的位置，是一颗桥接芯片。

- QPI -> QuickPathInterconnect，Intel前端总线（FSB）的替代者，可以认为是AMD Hypertransport的竞争对手
- MCH -> Memory Controller Hub，内置于CPU中的内存控制器，与CPU直接通信，无需走系统总线
- PCI Express(Peripheral Component InteconnectExpress, PCIe) - 一种计算机扩展总线(Expansion bus)，实现外围设备与计算机系统内部硬件（包括CPU和RAM）之间的数据传输。
- Overprovisioning - 比如 48*1Gbps access port交换机，通常只有4*1Gbps uplink，那么overprovisioning比 = 12:1
- PCI-E 2.0每条lane的理论带宽是500MB/s
- X58 – 相当于传统的北桥，只不过不再带有内存控制器，Code name = Tylersburg
- Lane - 一条lane由一对发送/接收差分线(differential line)组成，共4根线，全双工双向字节传输。一个PCIe slot可以有1-32条lane，以x前缀标识，通常最大是x16。
- Interconnect - PCIe设备通过一条逻辑连接(interconnect)进行通信，该连接也称为Link。两个PCIe设备之间的link是一条点到点的通道，用于收发PCI请求。从物理层面看，一个link由一条或多条Lane组成。低速设备使用single-lane link，高速设备使用更宽的16-lane link。

相关术语：

- address/data/control line
- 资源共享 ->资源仲裁
- 时钟方案（Clock Scheme）
- Serial Bus

PCI-E Capacity:

Per lane (each direction):

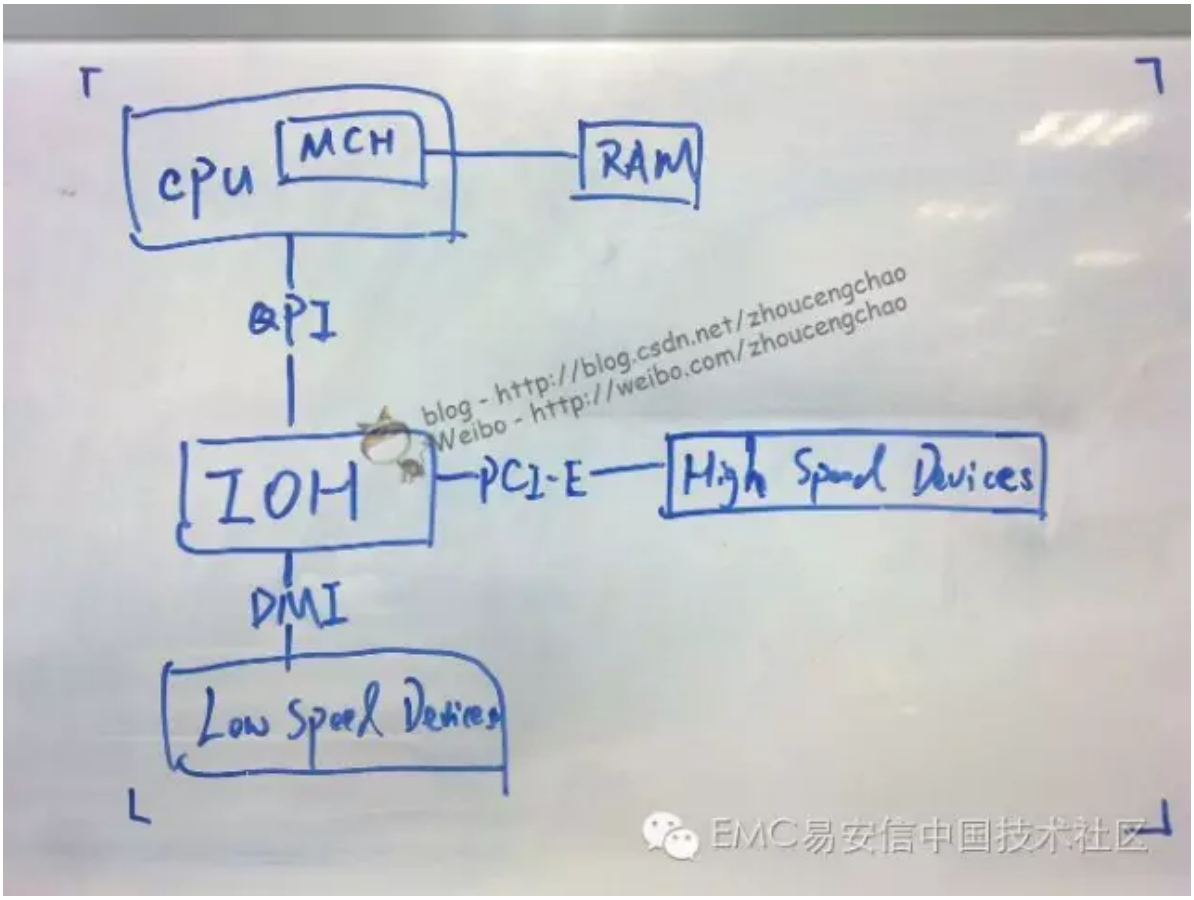
- v1.x: 250 MB/s (2.5 GT/s)
- v2.x: 500 MB/s (5 GT/s)
- v3.0: 1 GB/s (8 GT/s)
- v4.0: 2 GB/s (16 GT/s)

16 lane slot (each direction):

- v1.x: 4 GB/s (40 GT/s)
- v2.x: 8 GB/s (80 GT/s)
- v3.0: 16 GB/s (128 GT/s)

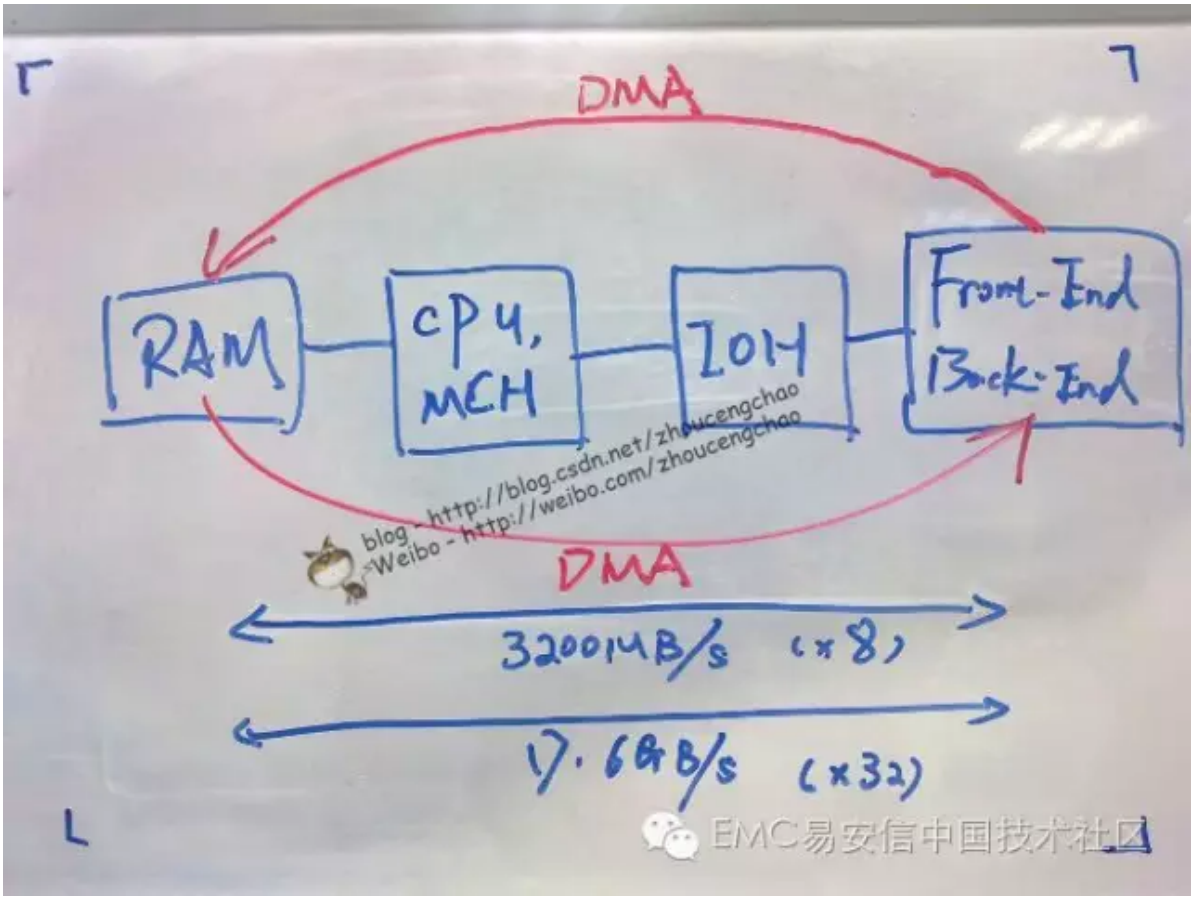
性能是【端到端】的，中间任何一个环节都有自己的性能极限，它并不像一根均匀水管，端到端性能一致。存储系统显然是不均衡的 ->overprovisioning。我将以中端存储系统为例，高端存储过于复杂，硬件结构可能都是私有的，而中端系统相对简单，就以一种双控制器、SAS后端、x86架构的存储系统为例。为了方便名称引用，我们就称他为Mr.Block_SAN吧。

控制器上看得见摸得着，又可以让我们算一算的东西也就是CPU、内存、I/O模块，不过我今天会带上一些极为重要但却容易忽略的组件。先上一张简图（字难看了点，见谅），这是极为简化的计算机系统构成，许多中端存储控制器也就这么回事儿。

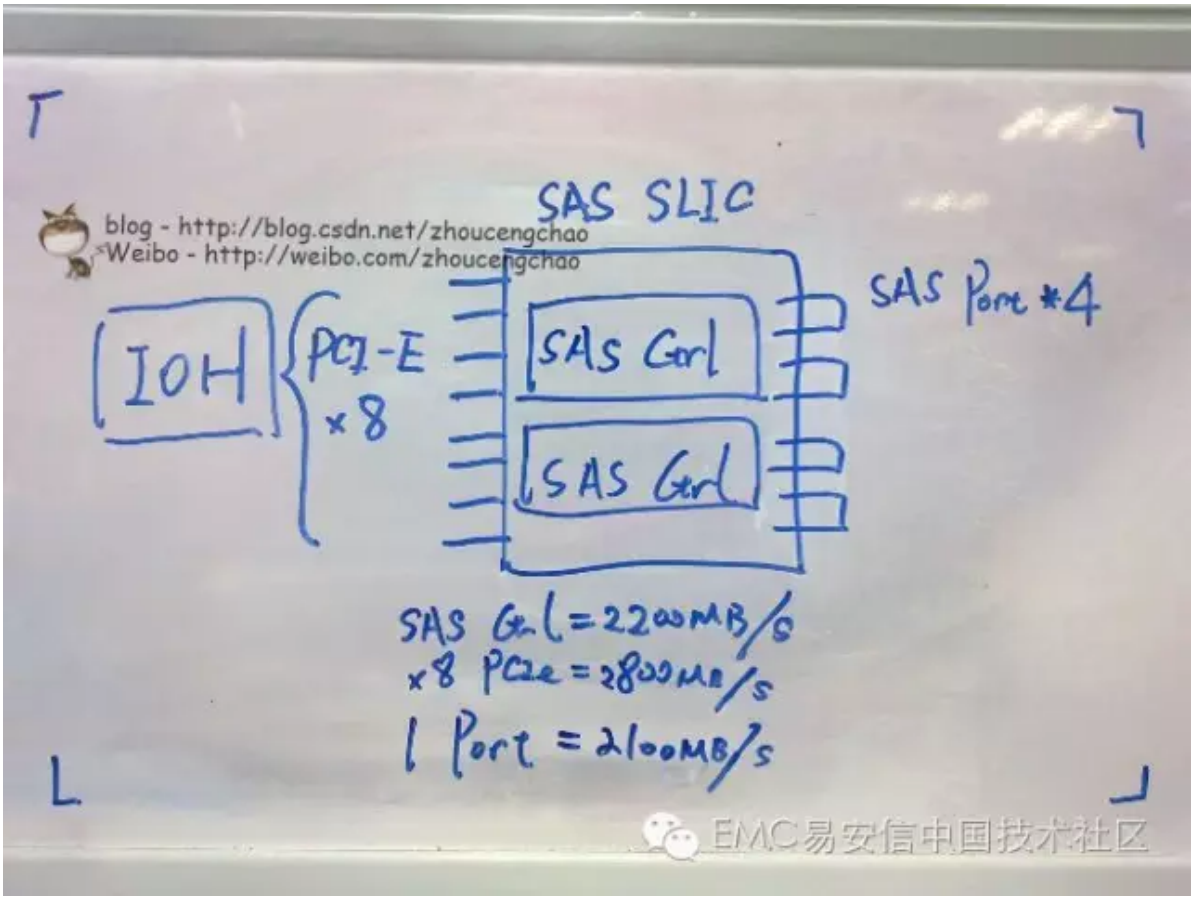


CPU - 假设控制器采用Intel Xeon-5600系列处理器（Westmere Microarchiecture ），例如Xeon 5660，支持DDR3-1333。CPU Bandwidth = 2.8GHz * 64bits / 8 =22.4 GB/s。

内存 – Mr.Block_SAN系统通过DMA (Direct Memory Access) 直接在Front End，内存以及Back end之间传输数据。因此需要知道内存是否提供了足够的带宽。3* DDR3，1333MHz带宽==29GB/s（通常内存带宽都是足够的），那么bit width应该是64bits。Westmere集成了内存控制器，因此极大的降低了CPU与内存通信的延迟。Mr.Block_SAN采用【X58 IOH】替代原始的北桥芯片，X58 chipset提供36 lane PCIe2.0 = 17.578GB/s bandwidth（后面会有更多解释）。



I/O模块 （SLIC）- SLIC是很多人关心的，因为它直接收/发送I/O。需要注意的是一个SLIC所能提供的带宽并不等于其所有端口带宽之和，还要看控制芯片和总线带宽。以SAS SLIC为例，一个SAS SLIC可能由两个SAS Controller组成，假设每个SAS Controller带宽大约2200MB/s realworld，一个SAS port = 4 * 6Gbps /8 * 70% =2100MB/s；一个SAS Controller控制2*SAS port，可见单个SAS Controller无法处理两个同时满负荷运转的SAS port（2200MB/s < 4200MB/s），这里SAS Controller是个瓶颈-> Overprovisioning!整个SAS SLIC又是通过【x8 PCI-E 2.0】 外围总线与【IOH】连接。x8 PCIe bandwidth = 8 * 500MB/s * 70% = 2800MB/s。如果两个SAS Controller满负荷运作的话，即4400MB/s > 2800MB/s，此时x8 PCIe总线是个瓶颈 -> Overprovisioning!



其实还可以计算后端磁盘的带宽和，假设一个Bus最多能连250块盘，若是SAS 15K RPM则提供大约12MB/s的带宽（非顺序随机64KB，读写未知），12 * 250 = 3000MB/s > 2100MB/s -> Overprovisioning!

Tip：一个SAS Controller控制两个SAS Port，所以如果只需要用到两根bus，可以错开连接端口，从而使的得两个SAS Controller都能得到利用。

同理，对任何类型的SLIC，只要能够获得其端口速率、控制器带宽、PCIe带宽，即可知道瓶颈的位置。我选择算后端带宽的原因在于，前端你可以把容量设计的很大，但问题是流量过来【后端】能否吃下来？Cache Full导致的Flush后端能否挡住？对后端带宽是个考验，所以以SAS为例或许可以让读者联想到更多。

PCI-Express – PCIe是著名的外围设备总线，用于连接高带宽设备与CPU通信，比如存储系统的I/O模块。X58提供了36 lane PCIe 2.0，因此36*500/1024 = 17.578125GB/s带宽。

QPI & IOH – QPI通道带宽可以通过计算公式获得，我从手中资料直接获得的结果是19-24GB/s（运行在不同频率下的值）。IOH芯片总线频率是12.8GB/s （List of Intel chipsets这里获得，但不确定总线频率是否就是指IOH本身的运行频率）< 17.578GB/s（36 Lane） -> Overprovisioning!

OK，算完了，能回答Mr.Block_SAN最大能提供多少带宽了吗？看下来CPU、RAM、QPI的带宽都上20GB/s，留给前后端的PCIe总线总共也只有18GB/s不到，即便这样也已经overload了IOH（12GB/s）。所以看来整个系统的瓶颈在IOH，只有12GB/s。当然，你还得算一下Mr.Block_SAN是否支持足够多的外围设备（eg. I/O模块）来完全填充这12GB/s，如果本身就不支持那么多外围设备，那IOH也算不上是瓶颈了。另外，我看到已经有网友提出我的计算存在8b/10b编码换算错误，由于个人

对硬件系统编码尚未透彻研究，理解这部分的读者可以自己对相应组件再乘以80%（我记得应该是）去掉编码转换的开销。

这篇文章更多的是一种举例式的说明，其中的数字和组件存在假设的情况。大家在计算的时候，可以参考这个思路将自己系统的参数和组件套用上去，从而计算出自己系统的带宽瓶颈。

注意 下图有点旧了，我把PCIe 36 Lane框成了MAX Bandwidth，因为那个时候以为IOH应该有足够的带宽，但后来发现可能不是这样，但图已经被我擦了，所以就不改了。