高技术通讯 2012 年 第22 卷 第10 期:1026 ~ 1034

doi:10.3772/j.issn.1002-0470.2012.10.005

网络相关的存储系统服务质量集成调度机制①

刘川意②*** 王春露**** ,王彦丞****

(*北京邮电大学 软件学院 北京 100876)
(**北京邮电大学 计算机学院 北京 100876)
(***北京邮电大学可信分布式计算与服务教育部重点实验室 北京 100876)
(****中国工商银行数据中心网络部 北京 100096)

摘 要 分析了典型网络存储系统的特点及其在提供服务质量(QoS)保证上面临的技术挑战,定量研究了存储请求、系统负载等因素对网络存储系统 QoS 参数的影响。在此基础上,提出了一种网络相关的 QoS 集成调度(NAIQS)策略,该策略综合考虑了会话的网络条件、硬盘负载条件以及存储请求的大小和请求的紧急程度等,将大请求分割成合适大小的请求分块,并根据请求的松弛时间对其进行排序和调度。实验结果证明,NAIQS 可以较大提高优先级请求的 QoS,并降低大请求的平均响应时间。

关键词 网络存储,服务质量保证,请求调度,存储请求,平均响应时间

0 引 言

近年来,随着网络存储系统的发展,QoS(服务质量)保证日益显现其重要性。传统存储系统的 QoS 保证主要关注在磁盘或磁盘阵列内部如何对 I/O 请求队列进行相应的调度来提供相应的 QoS 保证[1]。网络存储系统往往是分布式共享基础设施,不同的应用、不同的用户需要通过网络并发对其访问,因此 QoS 保证是网络存储系统中一个关键的问题。由于不同的应用或不同的用户对 QoS 有不同的要求,给 QoS 调度带来了新的技术挑战。针对这些问题,本文提出了一种网络相关的集成调度(network aware integrated QoS scheduling, NAIQS)机制,NAIQS 调度可以较大提高高优先级请求的服务质量,并降低由数据密集型应用产生的数据传输量较大的网络存储请求(大请求)的平均响应时间。

1 相关工作

随着存储系统规模的不断扩大以及数据中心等共享存储基础设施的逐渐兴起,相关的工作考虑在对 I/O 请求进行调度之前首先加以分类和区分,为来自不同应用的 I/O 请求提供不同的 QoS 保证。基于 QoS 保证的思路的典型算法有 Cello[2] 和 YFQ[3]等,典型的系统有 Facade[4] 和 Zygaria[5] 等。另外,研究人员还使用来自企业存储服务器的真实的 Workload 进行了详尽的测试,其测试方案的设计和思想等对本文工作有重要的参考价值。明尼苏达大学的 Lu 等[6]和加州大学圣克鲁兹分校的 Wu 等研究了智能存储设备的 QoS 控制问题[7]。

在网络 QoS 保证问题上人们往往注重于网络传输[8],而忽略了端处理。之前也有相关研究的思路是分割请求为小分块。如 Daigle 和 Strosnider 等[9]讨论了"分割一个请求为更小的请求分块"的用途以便别的实时的更高优先的请求在分块访问完成后就能够优先被调度;Dimitrijevic 等[10]扩充了这项工作,并且提出了相应的方式使高级别请求被优先调度。除了请求分割之外,他们也考虑了寻道和旋转的分割。

本研究通过对典型的网络存储系统[11,12]进行观察和分析,以及对比以前的网络 QoS 相关研究[13]或者存储子系统的 QoS 相关研究[14-19],得出如下一些结论:QoS 需求应该是端到端的;各种 QoS 需求是

① 国家自然科学基金(60273006)和国家教育部博士点专项基金(20100470256)资助项目。
② 男,1982 年生,博士;研究方向:信息处理;联系人,E-mail: cy-liu04@mails.tsinghua.edu.cn
(收稿日期:2010-10-10)

有区别的;数据访问模式和特性可能是很不相同的;每个客户端或会话都有其自身的网络条件或特性。在此基础上,提出一个针对网络存储系统的 QoS 调度方案——网络相关的 QoS 集成调度(NAIQS)。NAIQS 的主要想法包括:根据客户端会话的网络条件和特性,分割大请求为合适的相对更小的请求,以适应应用对 QoS 的要求以及当前会话的网络条件和共享存储系统的资源使用情况;并且基于请求的紧急程度和当前的工作负荷条件进行调度请求。该 QoS 调度方案的优点包括:

- 平滑会话通道中的数据传输。通过分割大请求,一个会话所接收的数据变得没那么突变。这将会使网络通道受益,从而使得网络流量更加平稳和缓和。
- 减少资源需求,如存储服务器缓冲的使用。因为来自于远端客户的请求(往往需要穿越广域网),其拥有的网络带宽相比于存储服务器所能提供的访问带宽而言,是非常有限的。特别是对于大请求,在数据传输期间,往往需要消耗很大容量的存储服务器内存缓冲区,而后者是很稀缺的资源。
- 允许紧急请求优先抢占。通常,存储请求是没有优先级的,即一旦一个请求被递交,则随后的请求需要等待直到该请求被执行完毕。而在网络存储系统中,由于其体系结构是分布式共享存储系统,因此不同的应用可能要求不同的带宽、请求优先级等,正如在上面分析的。为了满足服务质量需求,一个后到的请求可能需要优先于当前的请求被处理,因此,分割大请求会提供对紧急请求的优先处理的机会。

2 数据访问特点与分析

本节主要介绍典型网络存储系统的内部组成和数据访问模式,并讨论几个典型因素对数据传输的影响。

2.1 系统组成

图 1 展示了典型的网络存储系统的内部组成。多个会话(S_1 到 S_n)从多个客户端连接到系统的相应存储节点上。每个会话代表了一个具有一定的网络带宽 Rn_i 的可用网络通道。访问请求在每个会话相对应的会话队列中排队。每个会话也有一个相应的数据缓冲来接收请求的访问数据。

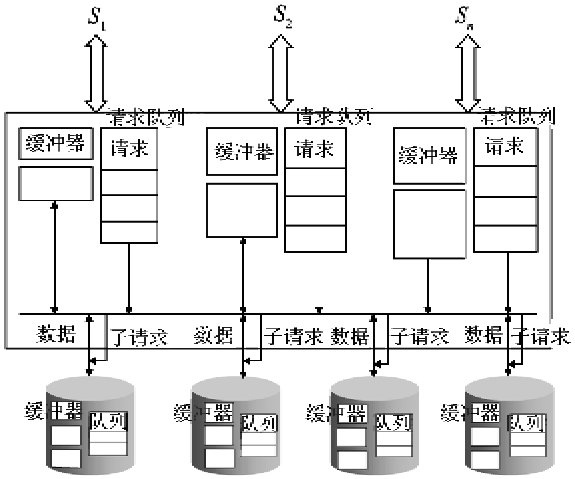


图 1 服务多个应用会话的网络存储系统内部组成

因为存储数据块状分布于多个硬盘设备,每个请求要被分割成多个子请求然后发送到相应的设备。起始的设备,包括硬盘数量和每个子请求的大小由请求的逻辑地址和数据大小所决定。子请求随后被传输到物理硬盘。数据从硬盘的内部缓冲区取得,并且被存储在会话缓冲区中与最初请求对应的缓冲区中。

2.2 大数据请求的影响

数据访问硬盘设备有一定的开销,如硬盘寻道时间、转动时间和切换时间等。这种开销不能通过一个先验的模型来决定,而是取决于当前一个和上一个访问之间的相对位置,如请求的顺序,它是通过负载和底层的硬盘调度策略来决定的。为了达到更大的硬盘访问吞吐量,大的请求将会更好。

图 2 展示了希捷 SC39102FC1 型号硬盘的访问性能。在这个测试中,请求随机分布于整个硬盘空间中。很明显,在小请求时,性能是很差的。例如,当请求大小是 4kB 时,所达到的性能是0.5MB/s。这是因为硬盘访问开销决定了访问时间,而与开销相比数据传输时间是很小的。当请求大小增加时,传

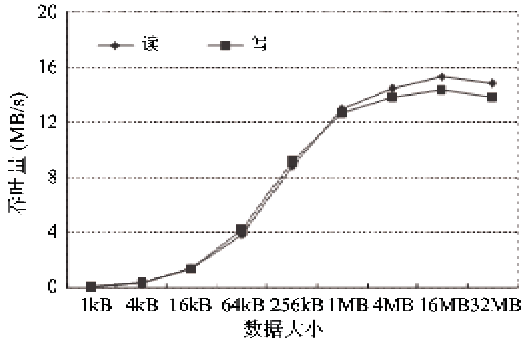


图 2 数据请求大小对硬盘访问性能的影响

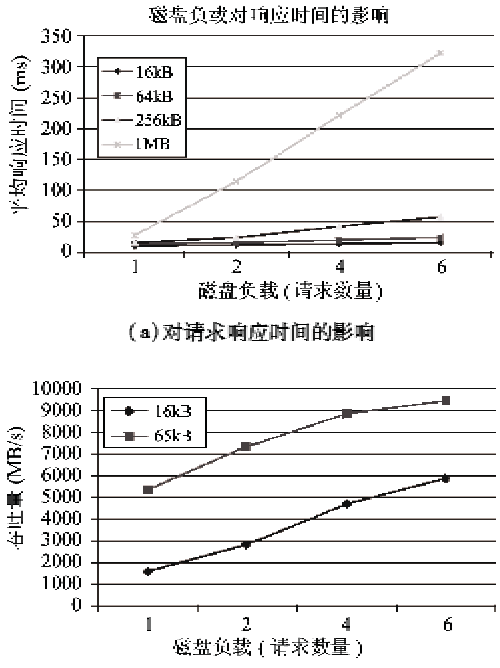
输时间也在增加，而硬盘开销保持不变。当请求大小接近 4MB 时，内部的传输时间决定了总体时间。结果，吞吐量很接近地反映了内部硬盘带宽。

网络存储系统作为分布式共享设施，对于某一些应用，像数据备份应用或者科学图像处理应用，数据请求的大小可能是巨大的。对于大数据请求，在硬盘访问的请求将会占据一段长的时间，就像网络传输一样。如图 2 所示大请求的一个优点就是硬盘利用率和吞吐量的提高，因而这种类型的请求很适合于对吞吐量要求迫切的应用。但是另一方面，大的请求也会带来一些缺点，这将在 2.3 节和 3.2 节中给予进一步定量说明和分析。因此，为每个对话保持合理的请求大小和请求速率是非常重要的。

2.3 磁盘阵列负载的影响

通常总的吞吐量伴随请求数量的增加而提高，因为大部分现在的 SCSI 硬盘有内部的调度算法，通过这些调度算法来对到达的请求重新排序以减少寻道和转动时间。另外，通过磁盘条带化，更多的并行访问增加了磁盘的利用率。

然而，平均响应时间随着未完成请求数量的增长而相应地增加。图 3(a)展示了不同负载和不同大小的请求对于请求响应时间的影响。请求大小的范围从 16kB 到 1MB，块单元大小是 64kB，硬盘采用希捷 SC39102FC。从图 3(a)可以看到，大请求(1MB)的平均响应时间随着硬盘负载的增加而明显增加。当硬盘负载从 1 变化到 6 个请求，响应时间从 28ms 增加到 322ms，增加了 11 倍以上。这意味着小请求的响应时间发生波动的可能性要小于大请求。换句话说，为了能更好地控制响应时间，大请求的负载应该受到限制。



(a)对请求响应时间的影响

(b)对系统吞吐率的影响

图 3 磁盘阵列负载对服务质量参数的影响

另一方面，如图 3(b)所示，吞吐量随着硬盘负载的增加而相应地受益。对于 16kB 大小的请求，从 1 个到 6 个请求的提升是 350%。对于 64kB 大小的请求，提升也达到 80%。因为我们不能预测物理硬盘中请求调度的顺序，不仅新的请求会被增加的访问时间所影响，以前的请求也会受到影响。因此，如果现存请求的处理时间已经很紧凑，再递交新的请求是不明智的。但如果现存请求有充足的时间来处理，增加更多的请求到硬盘对于达到更好的吞吐量是有益的。

3 NAIQS 机制

满足来自不同客户端的请求面临一些困难和挑战。首先，每个会话(通道)以及这个会话上的请求有它自己的特性。一个通道可能拥有充足的带宽，而另一个通道可能带宽很有限。其次，请求的大小可能是很不相同的。正如所讨论的，大请求对于另外那些紧急的请求来说可能会有不利的影响。最后，当前的 SCSI 和 SATA 硬盘支持多个未处理的命令，其使用一些调度策略来重新排列这些未处理的请求，使一些调度策略作为如何重新排列这些请求而存在。因为我们不能先验地确定调度策略，所以很难估计硬盘访问时间的偏差。

为了应对这些挑战，我们提出了一种网络存储相关的 QoS 集成调度(NAIQS)策略。这个调度策略综合考虑了网络模式和特性、硬盘负载条件以及请求的大小和请求的紧急程度。

假设有 n 个会话，每个回话的可利用带宽是 Rn_i。每个会话有一个数据大小为 S_i 的请求以及所需要的来回延迟时间，其中来回延迟时间包括硬盘访问时间和网络传输时间，它是在 T_i 范围内的。换句话说，这个请求的时限是 $T_{departure} + T_i$。所要求的等待时间可以通过客户端的应用来指定，例如，一个应用的用户或者系统管理员能够为应用配置期望的响应时间，或者它可以通过中间件设备来转化。中间件设备基于更高级别的 QoS 目标，如服务水平协议、带宽要求等。

3.1 请求的松弛时间(Slack Time)

定义请求的松弛时间是一个时间间隔,在这个时间间隔内请求应该得到调度。如果请求在这个松弛时间段以外被调度,时限将会错过。为了计算松弛时间,我们首先计算硬盘访问时间和网络传输时间。

前面已经讨论过,磁盘阵列中的请求被进一步分割成子请求,这些子请求被同时分派到各个硬盘同时并行处理。因而,硬盘访问时间通过硬盘访问时间中最长的那个访问时间来决定。假定

$$(ld+k)S_{\text{stripe}} < S_i < (ld+k+1)S_{\text{stripe}} \quad (1)$$

在这里,$0 \leqslant k \leqslant d-1$,$d$ 是硬盘的数量,l 每个硬盘包括的数据块的数量,S_{stripe} 是块的大小,S_i 是请求的大小,则 $l = S_i/(S_{\text{stripe}} \times d)$,$k = (S_i\%(S_{\text{stripe}} \times d)/S_{\text{stripe}}$。对于这个请求,最大的子请求是 $(l+d) \times S_{\text{stripe}}$。

花费在存储检索和网络传输上的平均时间是:

$$\begin{aligned} T_{\text{disk}} &= T_{\text{overhead}} + ((l+1) \times S_{\text{stripe}})/R_d \\ T_{\text{net}} &= S_i/Rn_i + Tn_i \\ T_{\text{total}} &= T_{\text{disk}} + T_{\text{net}} \\ &= T_{\text{overhead}} + ((l+1) \times S_{\text{stripe}})/R_d \\ &\quad + S_i/Rn_i + Tn_i \end{aligned} \quad (2)$$

在这里,T_{disk} 是花费在磁盘阵列检索上的时间,T_{net} 是网络传输大小为的 S_i 数据请求的时间,T_{overhead} 是硬盘定位时间(包括寻道时间和转动时间),R_d 是硬盘访问速率,Rn_i 和 Tn_i 分别是网络可利用的带宽以及第 i 个会话的传输延时。因此,这个请求的松弛时间可由式

$$\begin{aligned} TS_i &= T_i - T_{\text{total}} \\ &= T_i - T_{\text{overhead}} - ((l+1) \times S_{\text{stripe}})/R_d \\ &\quad - S_i/Rn_i - Tn_i \end{aligned} \quad (3)$$

计算出,其中,T_i 是请求能容忍的最大等待时间。

3.2 存储请求的分割

本文已经阐述了大请求所带来的好处,这种好处在网络环境中更突出[20]。但实际上由于网络存储系统是共享性基础设施,因此它的客户端的多样性就导致了请求大小类型的多样性。在备份、科学计算以及仿真、数据流等应用中常常包括很多大数据请求。这些大请求如果不能恰当处理就会引起以下问题:

- 到达相应会话的数据常常具有突发性。就像网络流量,不是平滑的,存储访问数据到达会话缓冲区时有时数据量会很大,造成的一个直接的缺点就是对缓冲区的资源占用很高。
- 大请求可能使随后的高优先权的请求错过其时限。因为存储访问请求是非抢占式的,大请求可能消耗一段较长的硬盘访问时间,这阻碍了随后的请求的执行和处理,从而降低了满足请求的服务质量要求的概率。

为了应对上述问题,NAIQS 方案的一个主要观点即是将大请求分割成合适的小数据请求,在本文中将其称为数据分块(segment)。

图 4 展示了在使用 NAIQS 方案前后,数据的获取和传输过程。在图 4 的 NAIQS 实现中,简单把大小为 S_i 的大请求均匀分割成两个分块,则每个分块的大小是:$S_1 = S_2 = S_i/2$。在时刻 T_1,第一个分块从硬盘检索出来传送到缓冲区。缓冲区大小是:$B_1 = (T_1 - T_{\text{data}}) \times R_d = S_1 = S_i/2$。同时,网络通道此刻开始发送数据到它的客户端。在时刻 T_2,第二个分块开始获取数据并传送到目标缓冲区。则此刻可体现数据传输的并行性,即从磁盘设备中访问存储数据和将获取的数据传输到发起客户端是同时进行的。磁盘设备的数据访问曲线($B\_d(t)$)以及数据传输曲线($B\_n(t)$)如下:

$$\begin{aligned} B\_d(t) &= \begin{cases} R_d \times (t - T_{\text{data}}) & , T_{\text{data}} \leqslant t < T_1 \\ S_i/2 & , T_2 \leqslant t < T_3 \\ S_i/2 + R_d \times (t - T_2) & , T_1 \leqslant t < T_2 \\ S_i & , t > T_3 \end{cases} \\ B\_n(t) &= \{Rn_i \times (t - T_1) \quad , T_1 \leqslant t < T_{\text{end}} \end{aligned} \quad (4)$$

因此,缓冲区的消耗为

$$\begin{aligned} B(t) &= B\_d(t) - B\_n(t) \\ &= \begin{cases} R_d \times (t - T_{\text{data}}), & T_{\text{data}} \leqslant t < T_1 \\ S_i/2 - Rn_i \times (t - T_2), & T_1 \leqslant t < T_2 \\ S_i/2 - R_n \times (T_2 - T_1) + R_d \times (t - T_2), & \\ \qquad\qquad T_2 \leqslant t < T_3 & \\ S_i - Rn_i \times (t - T_1), & T_3 \leqslant t \leqslant T_{\text{end}} \end{cases} \end{aligned} \quad (5)$$

在时刻 T_2,缓冲数据量是:$B_2 = B_1 - (T_1 - T_2) \times Rn_i$。如果 T_2(即设备访问第二个分块时)时刻太迟或者网络传输带宽太大,则会话在 T_2 时刻之前完成网络传输是可能的。如果是那样的话,网络传输在重新开始传输之前将会等待,直到足够量的数据从硬盘获取出来。在时刻 T_3,即从硬盘获取第二个数据分块完成,缓冲区的使用量将变成数据分块的大小,即 $B_3 = S_2 = S_i/2$。此后请求将需要时间 $B_3/Rn_i = S_i/(2 \times Rn_i)$ 来完成数据的传输。

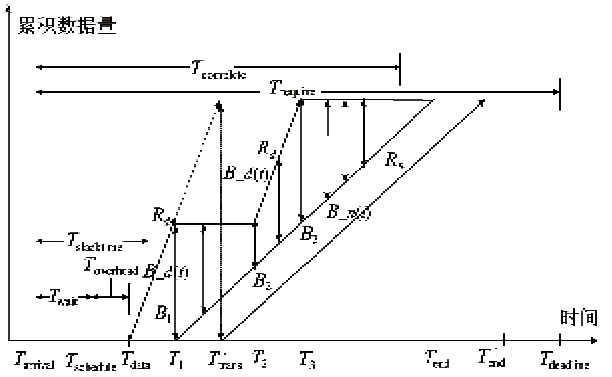


图 4 在使用 NAIQS 前后,数据获取过程中所占用的数据缓冲区

从图 4 可以看到,由于 NAIQS 方案充分利用了硬盘访问与网络传输的并行性。因此,缓冲区的使用减少了(图 4 的虚线部分是未使用 NAIQS 方案时,数据访问和网络传输的情况),同时响应时间也降低了。此外,优先抢占所需时间减少了,例如,现在一个高优先级请求能在一个上一个请求的请求第一个分块完成之后就可以进行抢占了。

3.3 NAIQS 算法流程

NAIQS 策略采用可变大小的大请求分割,请求分块的大小取决于三个因素,即原始大请求的大小、请求的紧急级别以及请求所在会话可利用的网络带宽。请求分块的大小对于不同的大请求可以是不同的。

在这些大请求被分割以后,形成一组由不确定的未分割大请求以及分割请求分块所组成的请求集合。然后根据其松弛时间来对其进行排序并且相应地对其进行调度。算法首先检查每个活动会话的第一个请求(因为我们假设对于每个会话,请求是按顺序进行处理的),如果需要的话,分割该请求。然后查找到最短松弛时间的请求 S_j,并检查存储系统当前的负载情况,判断该请求是否可以被发送。如果可以,则根据相应磁盘阵列的情况和配置(磁盘数目以及数据块单元大小),把该请求进一步分裂成子请求,发送给相应的磁盘设备进行执行。调度算法通过负载改变来触发,即请求在磁盘阵列中完成或者有新的请求到达。

这在这样的调度策略下,请求是以时间周期为粒度进行调度的。每个时间周期为 T_c。在这个周期内,能够估计从磁盘设备获取到的数据总量

$$S_{total} = T_c \times d \times r_d \tag{6}$$

其中,d 是硬盘的数量,r_d 是硬盘的平均带宽。

NAIQS 算法首先基于请求的松弛时间将活动会话的请求分成不同等级。不妨设定,请求的松弛时间在范围 $[0, 2 * T_c)$ 内属于等级 0;请求的松弛时间在范围 $[2 * T_c, 4 * T_c)$ 内属于等级 1;请求的松弛时间在 $[4 * T_c, 8 * T_c)$ 范围内属于等级 3 等。

分割策略同时负责调度请求或请求分块。这些选择的请求和片段放置在等待序列中等待调度。搜索从等级 0 的请求开始。0 等级的请求实际上不会有分割,因为它们是很紧急的请求。所有这个等级的请求都会被选择。在第一个等级的选择结束后,剩余请求的数据量为

$$S_{left} = S_{total} - \sum_{i \in \{L_i = 0\}} S_i \tag{7}$$

如果 $S_{left} \leqslant 0$,那么调度过程停止;否则,继续查找下两个等级的所有请求。例如,如果等级 1 和等级 2 的请求都存在,那么下两个等级就是等级 1 和等级 2。如果在等级 1 没有请求存在,则下两个等级就是等级 2 和等级 3,依此类推。需要确保当前被调度的请求数目要少于 S_{left}/S_{seg},其中 S_{seg} 是每个请求或请求分块的最小大小。

对于请求 S_k,其权重 W_k 以及分割出的请求分块的大小 S_k^c 用式

$$\begin{aligned} W_k &= Rn_k / L_k \\ S_k^c &= S_{left} \times W_k / (\sum\nolimits_j W_j) \end{aligned} \tag{8}$$

表示。其中,Rn_k 是请求可利用的网络带宽,L_k 是请求的紧急级别。最终实际分割的请求分块大小为

$$S_k^a = \begin{cases} S_k, & S_k \leqslant S_k^c + S_{seg} \\ ((S_k^c + S_{stripe} - 1)/S_{seg}) \times S_{seg}, & \text{其他情况} \end{cases} \tag{9}$$

即实际的分割是标准化分割:如果计算出的请求分块的大小接近于请求 S_k 大小,则不必进一步分割请求;否则,请求分块的大小取最接近 S_{seg} 的整数倍的值。所有的待调度请求(或请求分块)放置在一个列表返回。图 5 对上述分割算法进行了形式化描述。

NAIQS 算法基于上述分割算法。首先计算本时钟周期内的预计数据传输量;然后调用分割算法进行请求分割并且按照松弛时间递增的顺序排序等待队列中的请求。调度算法在发送一个请求之前,检查系统当前的负载情况,判断是否把请求发送到相应的磁盘设备。

NAIQS _ breakdown ***algorithm***(*S*)
Input: *S* 为等待队列中的请求集合
1. $S_{left} = S_i$;
2. *Reqs* = *NULL*;
3. **For** 所有的等级为 0 的请求 S_i
   a) 移除原始请求 S_i;
   b) 增加 S_i 到 *Reqs*……// 不需要分割;
   c) $S_{left} = S_{left} - S_i$;
4. **If** ($S_{left} <= 0$) *return Reqs*;
5. 查找等待的下两个等级请求, *level i*,*j*…// *i*,*j* 可以是 1,2
6. **For** 下两个等级每个请求或请求分块 S_k
   a) 计算权重 $W_k = Rn_k / L_k$ //*Lk* 是请求 S_k 的等级;
   b) 基于权重计算请求分块的大小,参见公式(5-8)
   c) 计算实际的请求分块大小 S_k', 参见公式(5-9)
   d) **If** ($S_k' > S_k$)
      i. $S_{left} = S_{left} - S_k$;
      ii. 从会话队列中移除 S_k;
      iii. 增加 S_k 至 *Reqs*;
e) **Else**
      i. 创建一个新的请求(请求分块),大小为 S_k';
      ii. 计算该请求的松弛时间;
      iii. 增加该请求到 Reqs;
      iv. 更新请求 S_i 的大小;
7. **Return** Reqs;
End Algorithm

图 5 请求分割算法

一个到来的紧急请求,如一个交互式请求,能够放置在等待队列中并且更早地调度。一个请求被发送结束后即能够触发等待队列中的请求处理。最后,当等待请求队列为空时,则开始下一轮调度。NAIQS 算法的形式化描述如图 6 所示,从中可以看到,NAIQS 算法综合考虑了网络条件,请求大小以及紧急程度,同时它也考虑到了磁盘阵列负载的情况。

3.4 控制磁盘负载

正如在 2.3 节所分析的,负载(磁盘阵列中待发送请求的数目)对磁盘吞吐率以及单个请求的访问延时都有重要影响。请求大小越大,响应时间的波动也就越大。因此,当一个新的请求在发送到相应的磁盘之前,检查它是否影响当前请求所需的 QoS 要求是必要的。

NAIQS _ sched ***algorithm***( )
1. 计算 $S_{total} = T_c * d * r_d$…// 这个时钟周期共传输的数据量
2. **For** 所有的活动会话
   a) 会话队列中的第一个请求设置为紧急等级 0;
3. *reqs* = *NAIQS _ breakdown*(S_{total});
4. 根据请求的松弛时间对 *reqs* 分类;
5. **For** 每个 *reqs* 中的请求(请求分块)
   a) 检查当前磁盘阵列负载;
   b) 发送该请求是否安全?
      i. **If** 是
         发送子请求到相应的磁盘;
      ii. **Else return**;
6. **Return**
End Algorithm

图 6 NAIQS 算法

在 NAIQS 的负载控制算法中,每个即将发送的请求将被检查是否存在破坏 QoS 需求的可能。检查首先估计这个新的请求引入的额外时间代价,然后把这个时间代价与先前估计的波动时间相加,并且将得到的时间代价之和与请求的松弛时间进行比较,判断该请求是否安全。只有当发送队列的所有请求对于新请求都是安全的时候,新的请求才被允许发送。在检查之后,所有请求预期的波动时间将被更新并且反映新请求的影响。

4 实验和评价

为了实验和评价 NAIQS 策略的有效性,我们实现了一个嵌入 NAIQS 机制的原型系统。本节描述实验环境,测试结果以及对结果的分析。

4.1 实验环境

图 7 展示了实验及测试环境的情况,主要包括:16 个通过光纤环 FC-AL 连接的 FC 硬盘(磁盘型号是 Seagate 39102FC),其中 FC-AL 环为 1Gb 带宽;服务器采用戴尔 PowerEdge 6350,并通过 QLA 2200 HBA 连接到 FC-AL 环。NAIQS 策略原型系统部署在该服务器中,并仿真磁盘阵列(RAID)控制器。磁盘阵列部署是现代存储系统中是很受欢迎的一种方案。它能够通过并行访问提供高性能,通过参与硬盘中的负载共享提供负载平衡,以及通过冗余信息提供容错功能。另外,负载发生器也部署在服务器中,模拟多客户端工作负载流。

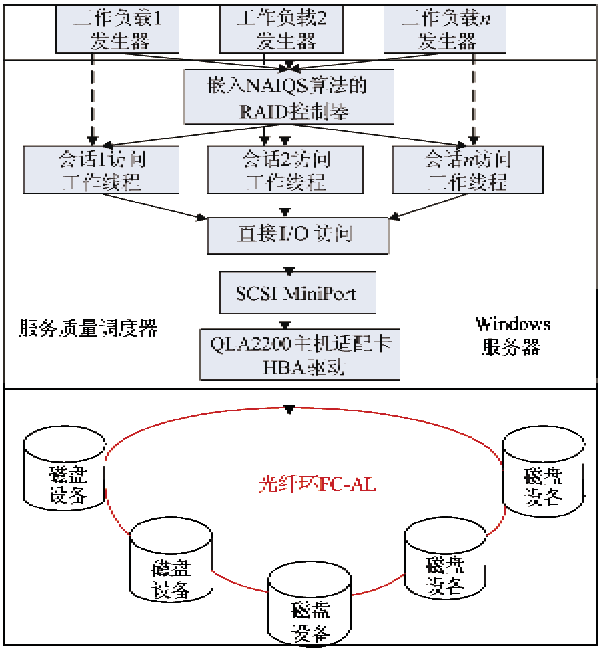


图 7 实验和测试环境

4.2 高优先级请求的服务质量

本节说明在不同的 QoS 调度算法下，高优先级请求（如交互式命令）的服务质量满足情况的比较。在这个测试中，最初有两个会话：一个会话代表了小的交互式请求，请求平均大小为 4KB，请求时限短，截止响应时间为 60ms，到达速率为每秒 2 个请求；另一个会话主要是大请求，请求的平均大小为 1MB，到达速率为每秒 1 个请求。我们假设两个会话的网络带宽都为 100Mbps。磁盘阵列的条带单元大小为 64KB。经过 90s 之后，第 3 个会话加入了。它的请求平均大小也是 1MB，到达速率也为每秒 1 个请求。这 3 个会话再持续 60s。

图 8 展示了高优先级请求的响应时间比较。在图上能清楚地看到 3 个带。最低的带，大约 10 到 13ms，代表了没有队列延迟的实际硬盘设备访问时间。大部分请求的访问时间落在这个带里。对于那些落在这个带里的请求来说，3 个调度策略之间没有区别。第二个带大约是 25ms 至 35ms。NAIQS 算法中其余的请求落入这个带。这个带中的响应时间包括实际硬盘访问时间（第一个带）加上等待大请求的某一个或某些请求分块完成的排队延迟。从图 8可以看到，通过分割大请求，NAIQS 算法确保高优先级请求的及时响应。第三个带在 35ms 至 48ms 之间。这个带的延时包括实际访问时间以及等待一个大请求完成处理的时间。Normal 算法（一般调度算法）以及 Priority 算法（基于优先级的调度算法）中请求的一部分落入这个类别，因为它们不得不等待更早的请求完成处理。最后一个带大于 50ms。这个带当第三个会话加入时发生（90ms 以后）。第三个会话使得交互式请求可能需要等待两个大请求，所以到 90ms 之后，一些 Normal 算法以及 Priority 算法的请求会落入这个带中。

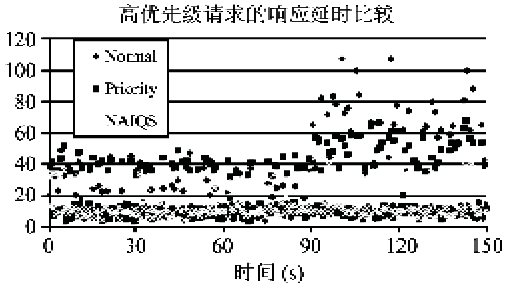


图 8 不同的 QoS 调度算法对高优先级请求的响应时间比较

表 1 展示了请求的平均响应时间、标准方差以及小请求（即高优先级请求）的最大响应时间。Normal 算法以及 Priority 算法具有最坏的平均响应时间，他们的方差也更大，最大响应时间也更大。Normal 算法的最大响应时间为 106ms，比 NAIQS 算法最大响应时间多了 2.5 倍。

表 1 请求响应时间统计

调度算法	平均响应时间（ms）	标准差（ms）	最大响应时间（ms）
Normal	23.4	23.4	106.8
Priority	21.6	17.78	68.83
NAIQS	16.2	10.1	43.47

表 2 显示了请求因为等待大请求的完成而错过截止时间的概率。在本测试中，截止时间设置成 55ms，包括硬盘访问时间以及网络传输时间。可知，若最多只等待一个大请求，则上述 3 个算法都是“安全”的。然而，若需要等待两个或两个以上的大请求完成执行的话，Normal 算法以及 Priority 算法则有比较大比例的错失率。从这些数据能够观察到，如果请求具有突发性，多个请求可能碰撞到一起。

表 2 请求错过截止时间的概率

调度算法	等待 1 个大请求	等待 2 个大请求
Normal	0%	27.5%
Priority	0%	12.5%
NAIQS	0%	0%

在这种情况下,NAIQS 算法很清楚地展示了它的优点。

4.3 大请求的响应时间

上一节分析了 NAIQS 策略使得小的高优先请求拥有更快的响应时间。另一方面,大请求也得益于 NAIQS 策略带来的 I/O 与网络的并行处理。图 9 展示了大请求(2MB)在轻负载情况(三个数据流)下的响应时间。大请求流的请求平均大小为 2MB,到达速率为每秒 1 个请求;另外两个数据流的请求平均大小分别为 64KB 和 256KB 大小的请求,到达速率分别为每秒 2 个请求和每秒 1 个请求,到达间隔按照指数分布。我们假设网络带宽为 200Mbps,网络传输延迟为 5ms,响应时间为请求传递和响应被接收的时间差。因此它包括网络传输时间以及存储访问时间(包括队列等待时间)。

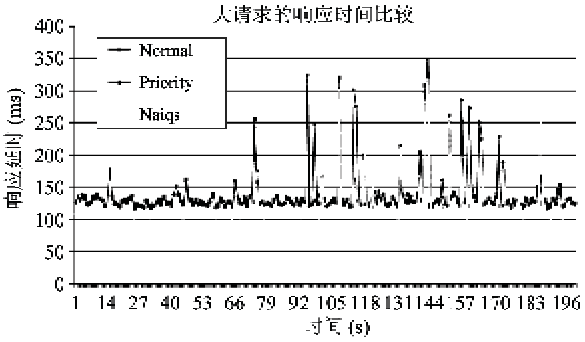


图 9 不同的 QoS 调度算法下,大请求的响应时间比较

从图 9 中可以观察到,大请求响应时间主要分为两个基本带。Normal 算法以及 Priority 算法调度的大请求有相似的响应时间,并且主要落入响应时间较慢的一个带中。使用 NAIQS 算法调度的大请求响应时间主要落入更低的带中,这意味着 NAIQS 算法能够降低大请求的响应时间。在图中有一些高峰,那是因为两个或者更多的请求在十分接近的时间里面到来,因此队列延迟导致了更长的响应时间。

5 结 论

针对共享网络存储系统,如云存储和数据中心存储系统,所需要面对不同用户通过网络的并发访问,本文提出了一种网络相关的服务质量调度机制——NAIQS。该机制的主要思路和特色是:通过考虑会话的网络条件、当前系统的负载情况等,将大请求分割成合适大小的请求分块,并根据请求的松弛时间对其进行排序和调度。实验证明,这种调度机制可较大提高高优先级请求的服务质量,并降低大请求的平均响应时间。

参考文献

[1] Worthington B, Ganger G, Patt Y. Scheduling algorithms for modern disk drives. In: ACM-Sigmetrics. Proceedings of the 1994 Association for Computing Machinery Special Interest Committee on Measurement and Evaluation Conference on Measurement and Modeling of Computer Systems, New York, USA: ACM Press, 1994. 241-251

[2] Shenoy P, Vin H. Cello: a disk scheduling framework for next generation operating systems. In: Proceedings of the Conference on Measurement and Modeling of Computer Systems, New York, USA: ACM Press, 1998. 44-55

[3] Bruno J L, Brustoloni J C, Gabber E, et al. Disk scheduling with quality of service guarantees. In: Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Washington DC, USA: IEEE Computer Society Press, 1999. 400 - 405

[4] Lumb C, Merchant A, Alvarez G. Facade: virtual storage devices with performance guarantees. In: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, San Francisco, USA: The USENIX Association, 2003. 131-144

[5] Wong T M, Golding R A, Lin C, et al. Zygaria: storage performance as a managed resource. In: Proceedings of the 12th IEEE Real Time and Embedded Technology and Applications Symposium, San Jose, California, USA: IEEE Computer Society Press, 2006. 125-134

[6] Lu Y P, Du H C, Ruwart T. QoS provisioning framework for OSD-based storage. In: Proceedings of IEEE Conference on Mass Storage Systems and Technologies, Minnesota University, MN, USA: IEEE Computer Society Press, 2005. 28-35

[7] Wu J C, Brandt S A. QoS support for intelligent storage devices. In: 2nd Intelligent Storage Workshop, Minneapolis, USA, 2004. 110-115

[8] Guerin R, Peris V. Quality-Of-Service in packet networks: basic mechanisms and directions. *Computer Networks Journal*, 1999, 31(3):169-189

[9] Daigle S J, Strosnider J K. Disk scheduling for multimedia data streams. *Proceedings of the Information Solution and Technology/ The International Society for Optical Engineering*, 1994, 2188: 212-223

[10] Dimitrijevic Z, Rangaswami R, Chang E. Design and implementation of semi-preemptible IO. In: proceedings of 2nd USENIX Conference on File and Storage Technolgies, San Francisco, USA: The USENIX Association, 2003.

145-158

[11] Bandulet C. The storage evolution: from blocks, files and objects to object storage systems. In: Storage Networking Industry Association, 2007

[12] Storage Networking Solutions - Europe. Object storage architecture: defining a new generation of storage systems built on distributed, intelligent storage devices. http://www.snseurope.com/featuresfull.php? id=2193: Angel Business Communications Limited, 2004

[13] Braden R, Clark D, Shenker S. Integrated services in the internet architecture: an overview. In: RFC 1633. Program on Internet and Telecoms Convergence. USA: ACM Press, 1994.200-212

[14] Wijayaratne R, Reddy A L N. Integrated QoS management for disk I/O. In: Proceedings of IEEE International Conference on Multimedia Computing and Systems, Florence, Italy: IEEE Computer Society Press, 1999.487-492

[15] Dimitrijevic Z, Rangaswami R. Quality of service support for real-time storage systems. In: Proceedings of international Irish Progressive Services International-2003 Conference, Stefan, Montenegro, 2003.143-150

[16] Guerin R, Peris V. Quality-Of-Service in packet networks: basic mechanisms and directions. *Computer Networks Journal*, 1999, 31(3):169-189

[17] Chuang J C I. Resource allocation for StorServ: network storage services with QoS guarantees. In: Proceedings of NetStore' 99 Symposium, Seattle, USA, 1999.45-56

[18] Bosch P, Mullender S J. Real-time disk scheduling in a mixed-media file system. In: Proceedings of Sixth IEEE Real Time Technology and Applications Symposium, Washington D C, USA: IEEE Computer Society Press, 2000.23-32

[19] Brandt S, Banachowski S, Lin C, et al. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In: Proceedings of the IEEE Real-Time Systems Symposium, Washington D C, USA: IEEE Computer Society Press, 2003.396-407

[20] Lu Y, Du D. Performance study of iSCSI-based storage system. *IEEE Communication Magazine*, 2003, 41(8): 76-82

NAIQS: network aware integrated QoS scheduling for storage systems |

Liu Chuanyi*,***, Wang Chunlu**,***, Wang Yancheng****

(*Software School, Beijing University of Posts and Telecommunications, Beijing 100876)

(**School of Computer Science and Technology, Beijing University of Posts and Telecommunications, Beijing 100876)

(***Key Laboratory of Trustworthy Distributed Computing and Service(BUPT), Ministry of Education, Beijing 100876)

(****Network Department of Data Center Consolidation (Beijing) of Industrial and Commercial Bank of China, Beijing 100096)

Abstract

The characteristics of a typical network storage system and its technical challenges in providing QoS (quality of service) guarantees are analyzed, and the effects of some major influencing factors such as storage requests and system loads on QoS parameters of a network storage system are quantitatively studied. On the basis of this, the network-aware integrated QoS scheduling (NAIQS), a new strategy for network storage systems' providing of QoS guarantees is proposed. The strategy synthetically considers the network conditions of different sessions, disk array loads, data request size and request emergencies, and then splits a typical large request into sub-requests with suitable sizes, and schedules them according to their respective slack time. The NAIQS was implemented and embedded into a prototype system, and the experimental results show that it can largely improve the QoS parameters of high priority requests and lower the average response time for large requests.

Key words: network storage, QoS guarantee, request scheduling, storage request, average response time