doi:10.11959/j.issn.1000-436x.2015018

基于负载特性和服务时间评估改进的 AS 调度算法

陈金忠，姚念民，蔡绍滨，孙美玲
(哈尔滨工程大学 计算机科学与技术学院，黑龙江 哈尔滨 150000)

摘 要：Linux I/O 调度层的预期调度算法 AS(anticipatory scheduling)对所有的负载分配相同的预期周期，如果等待的 I/O 请求没有及时到达，将会带来额外的时延。针对 AS 算法的不足，提出了一种基于负载特性和服务时间评估改进的 AS 算法(WPCAS, workload characteristic and service time evluation AS)。WPCAS 分为进程归类模块(PC)和服务时间评估模块(STE)两部分。PC 模块根据负载特性为每类进程指定不同的预期周期。STE 模块根据服务时间决定是否预期下一个请求。通过对比实验表明，WPCAS 在吞吐量、预期成功平均等待时延和伪空闲周期方面都优越于 95%-Heuristic 和 AS 算法。

关键词：存储系统；I/O 调度层；AS；WPCAS；服务时间
中图分类号：TP333.35 **文献标识码**：A

Improved anticipatory scheduling algorithm based on workload characteristic and service time evaluation

CHEN Jin-zhong, YAO Nian-min, CAI Shao-bin, SUN Mei-ling
(College of Computer Science and Technology, Harbin Engineering University, Harbin 150000, China)

Abstract: Anticipatory scheduling (AS) of Linux I/O schedule layer assigns fixed anticipation period length to each process. It will introduce extra delay if I/O request does not arrive in time. The improved anticipatory scheduling (WPCAS) algorithm is proposed which is based on workload characteristic and service time. WPCAS includes process classifier(PC) module and request service time evaluation (STE) module. PC module assigns different anticipation period lengths in terms of workload characteristic. STE module decides whether to anticipate the coming request according to service time. Experimental results show that WPCAS algorithm is superior to 95%-Heuristic and AS in terms of throughput, average waiting time of anticipation success and deceptive idleness.

Key words: storage system; I/O schedule layer; AS; WPCAS; service time

1 引言

随着 CPU 与内存速度的不断提高，存储系统成为计算机性能的瓶颈。I/O 调度层已经成为存储系统不可或缺的一部分[1,2]。I/O 调度算法大致分为 3 大类。

第 1 类是实时任务 I/O 调度算法，这类算法主要是在保证任务实时性的条件下，最大化任务的吞吐量。SCAN-EDF[3]将相同截止时间的请求划分到同一组，然后在每一组内应用 SCAN 算法进行扫描。SCAN-EDF 只对相同截止时间的任务按磁头移动方向调度，其性能取决于具有相同截止时间任务的数目。为了对更多任务进行优化，DM-SCAN (deadline-modification-SCAN)[4]将任务的截止时间修改为最短任务的截止时间 $d_{\min}$，将任务的完成时间不超过截止时间 $d_{\min}$ 的所有任务划分到同一组，然后在组内应用 SCAN 算法调度提高任务的吞吐量。RG-SCAN[5](reschedulable-group-SCAN)引用了

收稿日期：2013-06-29；**修回日期**：2013-08-13
基金项目：国家自然科学基金资助项目（61103209, 61170227）；浙江省自然科学基金资助项目（LZ12F02005）；浙江省教育厅科学基金资助项目（Y201222977）
Foundation Items: The National Natural Science Foundation of China (61103209, 61170227); The Natural Science Foundation of Zhejiang Province (LZ12F02005); Education Department Foundation of Zhejiang Province (Y201222977)

RG-group 的概念。它首先寻找包含最多任务数的集合，集合中所有任务的完成时间不超过任务的截止时间，此集合称为 RG-group，然后对 RG-group 应用 SCAN 算法来提高任务的吞吐量。RG-SCAN 不需要修改任务的截止时间。SCAN-EDF、DM-SCAN 和 RG-SCAN 都是在组内进行任务调度，属于局部调度算法。GSR (global seek-optimizing real-time)[6]根据磁头移动方向将任务分组，然后在保证实时性的前提下，在组与组之间调度任务，以达到吞吐量的最大化。GSR 是全局调度算法。

第 2 类是预定带宽或时延的 I/O 调度算法，这类算法用于满足用户的预定带宽或者预定时延，如 DVT(differential virtual time)[7]、Hybrid[8]、Argon[9]、pClock[10]、adaptive DRR(deficit round-robin)[11]、MBAC(measurement-based admission control) [12]和 BFQ(budget fair queuing)[13]等算法。这些算法在满足预定带宽或时延的条件下，提高任务的吞吐量。

第 3 类调度算法是为了提高吞吐量而设计的算法，主要有 SCAN、LOOK、SSTF(shortest seek time first)、NOOP(no operation)、Deadline、AS(anticipatory scheduling)和 CFQ(completed fairness queuing)等调度算法[14~17]。这些算法旨在减少磁头移动的距离。NOOP、CFQ、Deadline 和 AS 是当前 Linux I/O 调度层常用的调度算法。NOOP 实现了最简单的 FIFO 队列，所有的请求除了合并之外，采用先来先服务的策略。CFQ 为每个进程分配一个队列，按照 I/O 请求的扇区地址进行排序，每个进程的 I/O 请求以循环的方式服务。CFQ 对于每个进程都是完全公平的。相对于 NOOP 来说，先到来的请求不一定先服务，CFQ 可能出现 I/O 请求饿死的现象。Deadline 在 CFQ 的基础上，为每个 I/O 请求分配了截止时间，解决了饿死的问题。CFQ 和 Deadline 考虑的焦点在于满足零散 I/O 请求上，对于连续的 I/O 请求，比如顺序读，并没有做优化。并且这些调度算法都是持续性工作的[18]，一旦服务完 I/O 请求，立即调度当前队列 I/O 请求。为了满足随机 I/O 和顺序 I/O 混合的场景。预期调度算法(AS)在请求提交完之后并不立即处理当前队列中的 I/O 请求，而是等待一段空闲时间，以等待下一个邻近的请求被提交，等待周期为 6 ms。这样给应用程序提供了提交相邻磁道位置的 I/O 请求的机会，从而减少磁头寻道的距离。但 AS 存在两点不足。第一，不同负载特性的 I/O 请求对 AS 的性能产生很大影响。对于 I/O 密集型的应用程序，2 个 I/O 请求到达时间间隔可能很小(<6 ms)，使用 6 ms 的预期周期，显然浪费了时间。对于 I/O 请求时间间隔较大的应用程序(>6 ms)，使用 6 ms 的预期周期，必然会增加预期失败的次数。第二，AS 只根据当前队列中 I/O 请求的磁道位置而没有根据 I/O 请求的服务时间来决定是否预期下一个请求，这就导致预期不准确。

针对 AS 算法存在以上 2 个方面的不足，提出了一种改进的基于负载特性和服务时间评估的 AS 调度算法(WPCAS)，主要分为进程归类模块(PC)和服务时间评估模块(STE)。PC 模块通过分析负载的访问特性，为不同类型的负载设置不同的预期周期。STE 模块根据 I/O 请求的服务时间决定是否预期下一个 I/O 请求。相比于 AS，WPCAS 不仅适应了负载的动态变化，而且使预期更加准确。因此，本文提出的 WPCAS 调度算法在吞吐量和伪空闲周期等方面的性能都优于传统的 AS 调度算法。

2 相关工作

Linux I/O 子系统主要包括虚拟文件系统、页面高速缓存、通用块设备层和 I/O 调度层。Linux 内核的调度算法集中于 I/O 调度层，主要有以下 4 种调度算法：NOOP、Deadline、CFQ 和 AS。

2.1 I/O 请求服务时间模型

假设请求序列 $R=\{R_1,R_2,\cdots,R_n\}$，R_i 是 R_j 的前一个请求。$d_{i,j}$ 为请求 R_i 与请求 R_j 的磁道的距离。b_j 为 R_j 的数据块长度。l_j 为请求 R_j 的扇区位置。$c_{i,j}$ 为服务请求 R_j 的时间，则 $c_{i,j}$ 可用下式计算

$$c_{i,j}=\text{seek}(d_{i,j})+\text{rot}(l_j)+\text{trans}(b_j) \tag{1}$$

如式(1)所示，其中，seek 、rot 和 trans 分别表示 R_j 的寻道时延，旋转时延和数据的传输时延。根据文献[19]的磁盘模型，寻道时延可以表示为

$$\text{seek}(d_{i,j})=\begin{cases}3.24+0.4\sqrt{d_{i,j}}, & d_{i,j}\leqslant 383\\ 8.00+0.008d_{i,j}, & d_{i,j}>383\end{cases} \tag{2}$$

其中，旋转时延与磁盘转速相关，传输时延等于 I/O 请求除以数据传输率。结合式(1)和式(2)，可计算 I/O 请求的服务时间。

2.2 AS 预期策略

2.2.1 AS 预期原理

AS 执行完成当前读请求，并不立即执行队列的下一个请求，而是预留 6 ms 的时间窗口去等待下

一个即将到来的请求。如果下一个到来的请求与当前已完成请求的磁道位置是邻近的，那么将会极大提高 I/O 的性能。为了预期下一个请求，AS 算法维护每类进程历史 I/O 请求的平均到达时间间隔和磁道位置信息[20]。

AS 预期原理：AS 根据 I/O 请求的磁道位置判断从当前队列挑选的请求是不是“最优”请求。如果挑选出 I/O 请求的磁道位置与当前磁头位置的距离小于进程历史相邻 I/O 请求磁道位置之差的平均值，就认为此请求是“最优”请求，从而立即服务该请求。否则，启动定时器，预期下一个请求[20]。

2.2.2　AS 预期机制状态转换

图 1 显示了 AS 的预期机制状态转换图。当上一个 I/O 请求完成时，处于 FINISHED 状态，I/O 调度层根据上一节提到的预期原理判断是否预期下一个请求。如果预期下一个 I/O 请求，那么启动定时器，并将定时器的超时时间设置为 6 ms，进入了 WAITING 状态。否则，直接调度当前队列中的 I/O 请求，进入 SCHEDULING 状态。当定时器超时，表示预期失败，那么从当前队列选取 I/O 请求进行服务，进入 SCHEDULING 状态。当预期成功时，服务预期到的 I/O 请求，也进入 SCHEDULING 状态。此后，将 I/O 请求分发到通用块设备层中进行调度，进入 RUNNING 状态。当执行完 I/O 请求时，又回到了 FINISHED 状态。

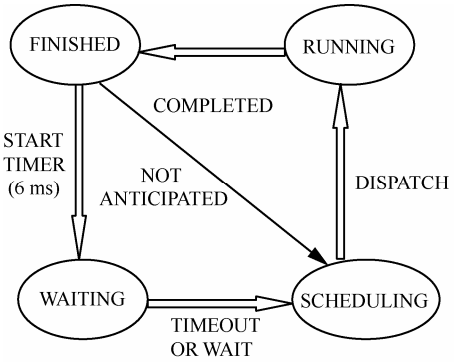


图 1　预期机制状态转换

2.3　AS 执行流程

Linux I/O 调度层的 AS 算法 具有以下几个特征。

1) 近似单向电梯算法,可以向后寻道的最大磁道距离为 1 024×1 024 个扇区。

2) 为读写操作指定不同的超时时间，读超时时间为 125 ms，写超时时间为 250 ms。

3) 读写操作采用批处理的策略。为读 batch 与写 batch 分配不同的超时时间，读 batch 超时时间为 500 ms，写 batch 超时时间为 125 ms。

4) 预期周期：AS 为 I/O 请求指定 6 ms 的预期周期，期望下一个请求邻近于当前请求。

图 2 描述了 AS 算法的执行流程。AS 维护 FIFO 队列和 SORT 队列。FIFO 队列是按照 I/O 请求到达时间进行排序的队列，SORT 队列是按照 I/O 请求的磁道位置与当前磁头位置的距离远近进行排序的。首先，如果 FIFO 队列中有超过截止时间的请求，就马上服务这个请求，并从 FIFO 队列和 SORT 队列删除该请求。否则，根据预期原理决定是否预期下一个 I/O 请求。然后, AS 调度算法将当前队列中的 I/O 请求或者是预期到的 I/O 请求分发到通用块设备层的调度队列中去。最后，由磁盘驱动器来完成数据的实际传输。

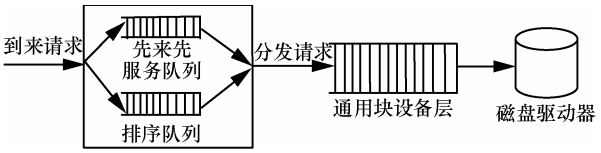


图 2　预期调度算法的处理流程

3　WPCAS 调度策略

综上所述，AS 有 2 个特点。第一，预期周期为固定的 6 ms。第二，根据当前队列中 I/O 请求的磁道位置与当前磁头位置的距离是否小于进程历史相邻 I/O 请求磁道位置之差的平均值，决定是否预期下一个 I/O 请求。AS 有 2 点不足。首先，不同负载特性的 I/O 请求会对 AS 的性能产生很大影响。对于 I/O 密集型的应用程序，2 个 I/O 请求的到达时间间隔可能很小(<6 ms)，使用 6 ms 的预期周期，显然浪费了时间。对于 I/O 请求时间间隔较大的应用程序(>6 ms)，使用 6 ms 的预期周期，必然会增加预期失败的次数。其次，根据 I/O 请求的磁道位置来判断是否预期，不够准确。为了更好地适应负载变化和更准确的预期 I/O 请求，提出了一种改进的基于负载特性和服务时间评估的 AS 调度算法(WPCAS)，它包括进程归类模块(PC)和服务时间评估模块(STE)。PC 模块主要功能是根据 I/O 请求的负载特性，将进程归类并设置为不同的预期周期。STE 模块的主要功能是根据 I/O 请求的服务时间来判断是否预期下一个 I/O 请求。

3.1　进程归类模块(PC)

AS 对所有的负载都使用固定的预期周期

(6ms)，不能适应负载的动态变化。本文提出了进程归类的方法，其基本思想是根据负载特性决定预期周期的大小，将不同类别的进程映射为不同的预期周期。进程归类模块首先找出 I/O 请求到达时间间隔最密集的区域，然后将这个区域对应的到达时间间隔作为预期周期。下面给出一些定义。

定义 1 T_{think}：I/O 请求的到达时间间隔。

定义 2 概率密度函数 $f(T_{\text{think}})$ 表示 T_{think} 的概率密度。

定义 3 分布函数 $F(T_{\text{think}})$ 表示 T_{think} 的分布。

定义 4 $\text{Map}(i)$ 表示进程 i 映射的预期周期。

假设 I/O 请求的到达时间间隔在 0 到 20 ms 之间。由定义可知

$$F(T_{\text{think}})=\int_0^T f(T_{\text{think}})\,\mathrm{d}T_{\text{think}}\ (0\leqslant T\leqslant 20) \tag{3}$$

当 $f'(T_{\text{think}})=0$ 时，落入此区域的 I/O 请求数最多。因此，得到最佳的预期周期 T_b 的取值范围，即

$$T_b\in[a,b]\,(0\leqslant a\leqslant b\leqslant 20) \tag{4}$$

另外，记 $p_{a,b}$ 为 I/O 请求到达时间间隔在区域 $[a,b]$ 的概率，由概率分布函数知

$$p_{a,b}=F(b)-F(a) \tag{5}$$

结合式(3)~式(5)，可为进程 i 指定最佳的预期周期，即

$$\text{Map}(i)=\begin{cases} j,\ j\in[a,b], p_{a,b}\geqslant\gamma, 0<\gamma\leqslant 1\\ T_{as}, p_{a,b}<\gamma\end{cases} \tag{6}$$

其中，T_{as} 表示预期成功的平均等待时间，定义为预期成功所花费的总时延除以预期成功的 I/O 请求数。γ 是 $p_{a,b}$ 的门限值。当 $p_{a,b}\geqslant\gamma$，那么选择整数 $j\in[a,b]$，作为进程的预期周期。当 $p_{a,b}<\gamma$，选择 T_{as} 作为预期周期。因此，进程归类模块根据 Map 函数动态调整预期周期的长度，适应了负载的动态变化。

3.2 服务时间评估模块(STE)

针对 AS 根据 I/O 请求的磁道位置来判断是否预期下一个 I/O 请求，提出了更精确的基于服务时间的预期策略。其基本思想是比较当前队列 I/O 请求的服务时间和预期情况下 I/O 请求服务时间的大小，决定是否预期。

定义 5 S_i 表示刚服务完的第 i 个 I/O 请求的磁道位置，S_{i+1}^s 表示从当前队列挑选的 I/O 请求的磁道位置，此请求是离当前磁头位置最近的。S_{i+1}^a 表示预期成功的 I/O 请求的磁道位置。

定义 6 一个 I/O 请求的三元组 (T_i^{sk},T_r,T_t)，T_i^{sk} 指 I/O 请求 i 的寻道延迟，T_r 指 I/O 请求的旋转延迟，取决于磁盘转速。T_t 指 I/O 请求的传输延迟，取决于数据传输率。

定义 7 进程历史相邻 I/O 请求的磁道位置之差的平均值 $a_{\text{seek\_dist}}$ 定义如下：

$$a_{\text{seek\_dist}}=\frac{1}{n}\sum_{i=1}^{n}\left|S_i-S_{i-1}\right|$$

其中，$S_0=\left|S_1-\text{head\_s}\right|$，head_s 表示磁头的初始位置，$n$ 表示 I/O 请求的个数。

定义 8 预期周期 P 指等待下一个 I/O 请求的时间长度。

定义 9 预期成功率 p_s、预期失败率 p_f 定义如下。

p_s：预期成功请求数占总预期请求数的比例。

p_f：预期失败请求数占总预期请求数的比例。

显然 $p_s+p_f=1$。

首先计算从当前队列挑选的 I/O 请求的服务时间。记为 T_{i+1}^s。则 T_{i+1}^s 可表示为

$$T_{i+1}^s=T_{i+1}^{sk}+T_r+T_t=T_{\text{seek}}(S_{i+1}^s,S_i)+T_r+T_t \tag{7}$$

其中，$T_{\text{seek}}(S_{i+1}^s,S_i)$ 表示寻道时间函数。

接下来，计算在预期情况下 I/O 请求的服务时间，记为 T_{i+1}^a。则 T_{i+1}^a 可表示为

$$T_{i+1}^a=p_sT_s+p_fT_f \tag{8}$$

其中，T_s 表示预期成功的服务时间，T_f 表示预期失败的服务时间。假设 T_{as} 表示预期成功平均等待时间。T_{af} 表示预期失败的平均等待时间，预期周期为 P ms，则 T_{af} 等于预期周期 P。显然，预期成功的服务时间 T_s 等于预期到 I/O 请求的寻道时延 $(T_{\text{seek}}(S_{i+1}^a,S_i))$，旋转时延，传输时延和预期成功的平均等待时延 (T_{as}) 之和，如式(9)所示

$$T_s=T_{\text{seek}}(S_{i+1}^a,S_i)+T_{as}+T_r+T_t \tag{9}$$

在预期失败时，需要执行当前队列的 I/O 请求，所以预期失败的服务时间 T_f 等于从当前队列挑选出的 I/O 请求的服务时间 T_{i+1}^s 与预期失败的等待时延 T_{af} $(T_{af}=P)$ 之和，如式(10)所示

$$T_f=T_{i+1}^s+P \tag{10}$$

根据式(9)和式(10)，式(11)表示为

$$T_{i+1}^{a}=p_s(T_{\text{seek}}(S_{i+1}^{a},S_i)+T_{as})+p_f\cdot P+p_f(T_{\text{seek}}(S_{i+1}^{s},S_i))+T_r+T_t \tag{11}$$

根据 T_{i+1}^{s} 与 T_{i+1}^{a} 的大小判断是否预期下一个请求，计算 T_{i+1}^{s} 与 T_{i+1}^{a} 的差值，记为 Δ，则有

$$\Delta=p_s(T_{\text{seek}}(S_{i+1}^{s},S_i)-T_{\text{seek}}(S_{i+1}^{a},S_i)-T_{as})-p_fP \tag{12}$$

当 $\Delta>0$，则有

$$\frac{P}{T_{\text{seek}}(S_{i+1}^{s},S_i)-T_{\text{seek}}(S_{i+1}^{a},S_i)-T_{as}}<\frac{p_s}{p_f} \tag{13}$$

在式(13)中，S_{i+1}^{s} 和预期周期 P 是已知的。其中 P 从进程归类模块得到。p_s、p_f 和 T_{as} 可通过进程历史信息计算得到。S_{i+1}^{a} 可由当前磁头位置向前或者向后偏移 $a_{\text{seek\_dist}}$ 来代替，即 $S_{i+1}^{a}=S_i\pm a_{\text{seek\_dist}}$。

如果式(13)成立，表示调度下一个 I/O 请求的时间小于调度当前队列 I/O 请求的时间。此时预期下一个 I/O 请求。否则，调度当前队列的 I/O 请求。而在 AS 调度中，如果 $\left|S_{i+1}^{s}-S_i\right|<a_{\text{seek\_dist}}$ 就立即服务当前队列的 I/O 请求。可以看出，服务时间评估模块将磁道位置转换成服务时间，根据 I/O 请求的服务时间来判断是否预期下一个 I/O 请求，使预期更加准确。

3.3　WPCAS 算法设计与复杂性分析

3.3.1　WPCAS 算法设计

WPCAS 算法主要包括 2 个部分，分别是进程归类模块和服务时间评估模块。进程归类模块为不同类型的负载设置不同的预期周期，并输出预期周期大小。服务时间评估模块比较 SORT 队列当前 I/O 请求的服务时间和预期情况下 I/O 请求的服务时间，决定是否预期下一个 I/O 请求。如果预期下一个 I/O 请求，设置预期标志(anticipated_flag=1)，否则，设置预期标志(anticipated_flag=0)。WPCAS 算法的形式化描述如下。

算法 1　WPCAS 调度算法

输入：I/O 请求状态信息

输出：预期周期大小

step1　统计进程 I/O 请求的历史状态信息，包括磁道位置和到达时间间隔、预期失败的请求数、预期失败所花费的总时延、预期成功的请求数和预期成功所花费的总时延。

step2　计算预期成功率(p_s)、预期失败率(p_f)和预期成功的平均等待时间(T_{as})。

step3　根据式(3)和式(4)，计算最佳的预期周期范围，再由式(6)，得到每类进程的最佳预期周期大小。然后，设置并输出每类进程的预期周期大小。

step4　计算调度 SORT 队列当前 I/O 请求的服务时间 T_{i+1}^{s} 和预期情况下 I/O 请求的服务时间 T_{i+1}^{a}，并进行比较。如果 $T_{i+1}^{s}>T_{i+1}^{a}$，设置预期标志(anticipated_flag=1)，表示预期下一个 I/O 请求。否则，设置预期标志(anticipated_flag=0)，表示调度 SORT 队列的当前 I/O 请求。

step5　若 FIFO 队列有超过截止时间的 I/O 请求，则将该请求分发到通用块设备层的调度队列，进行处理。否则，转入 step6。

step6　如果 anticipated_flag=0，则将 SORT 队列的当前 I/O 请求插入通用块设备层的调度队列，等待磁盘驱动器完成数据传输。否则，根据 step3 得到预期周期的长度, 设置定时器的超时时间，等待下一个 I/O 请求到达。如果定时器超时，预期失败请求数加 1，服务 SORT 队列的当前 I/O 请求。否则，预期成功请求数加 1，服务预期到的 I/O 请求。

3.3.2　WPCAS 算法复杂性分析

假设进程类别为 m，每类进程历史 I/O 请求数为 n,那么 WPCAS 需要记录 mn 个 I/O 请求，即空间复杂度为 $O(mn)$。PC 模块的时间复杂度为 $O(m)$，STE 模块的时间复杂度为 $O(mn)$。所以 WPCAS 算法的时间复杂度为 $O(m+mn)$。因此，WPCAS 算法是可行的。

4　性能测试

在 Linux 2.6.20 内核对 AS 算法进行修改，增加了 PC 和 STE 2 个模块。分别对吞吐量，预期成功的平均时延和伪空闲周期[18]进行了测试。为了测量真实的磁盘性能，绕过了 Linux 页面高速缓存，使用直接 I/O 测试。主要包含以下几种负载。

1) IOzone 生成的顺序负载。

2) IOzone 生成的随机负载。

3) Postmark 生成的 Web 服务器负载。

通过测试对比了 NOOP、CFQ、Deadline、AS、WPCAS 和 95%- Heuristic[18]等调度算法的性能。

4.1　测试工具

采用 IOzone 和 Postmark 作为负载测试工具。IOzone[20]是一个文件系统的 benchmark 工具，可以

测试不同的操作系统中文件系统和磁盘的读写性能。 Postmark[21] 主要用于测试文件系统在邮件系统或电子商务系统中的性能，其特点是：频繁、大量地存取小文件，可生成 Web 服务器负载、数据库负载和 Email 负载等。

4.2 性能指标

实验性能评价参数主要有：

1) 吞吐量；

2) 预期成功的平均等待时延；

3) 伪空闲周期的长度。

伪空闲周期长度定义为预期失败的总时延除以总的预期请求数。显然，这个值越小，平均响应时间越小。

4.3 实现方法

实验测试中，在 Linux 2.6.20 内核 block/blkdev.h 头文件的 io_context 结构体加入了链表 inv_d。在 block/as-iosched.c 文件的 as_update_iohist 记录最近 *n* 个请求的到达时间间隔，并存入链表 inv_d。在 as_antic_update_rq 加入了 2 个计数器 as_scount 和 as_stime，记录预期成功的请求数和预期成功所花费的总时延。在 as_antic_timout 函数中加入了 2 个计数器 as_fcount 和 as_ftime，记录预期失败的请求数和预期失败所花费的总时延。然后添加函数 as_antic_dertermined 和 as_class_proccess。as_class_proccess 根据进程归类模块为不同负载指定的预期周期。as_antic_determined 根据服务时间评估模块决定是否预期下一个请求。

4.4 顺序负载和随机负载测试

使用 IOzone 生成 512 KB 大小的文件，测试 NOOP、Deadline、CFQ、AS 和 WPCAS 在 10、40、80 类进程下的吞吐量。以进程类别数等于 40 为例，顺序负载配置为: #root/iozone *–i* 0 *–i* 1 *–r* 4096 *–t* 40 *–s* 100 *–I –R* result.*xls*。其中，*–i* 0 表示顺序写、*–i* 1 表示顺序读、*–r* 表示测试块大小、*–t* 表示进程类别数、*–s* 表示测试文件大小、*–I* 表示绕过文件系统缓存，直接对磁盘进行读写。*–R* 表示产生 excel 的输出日志。随机负载配置为：#root/iozone *–i* 2 *–r* 4096 *–t* 40 *–I –R* result.*xls*。其中，*–i* 2 表示随机读写负载。实验中分别对 10、40、80 类进程下的顺序负载和随机负载测试了 50 次，对 3 种情况下的测试结果分别求平均值并进行比较。

4.4.1 顺序读写 512 KB 文件的吞吐量

图 3~图 5 显示了 I/O 调度算法在顺序负载下的吞吐量。图 3 和图 4 表明，当进程类别较少时，NOOP 和 Deadline 在顺序负载下的吞吐量比其他 I/O 调度算法都高。图 5 表明，当进程类别数较多时，由于不同进程的 I/O 请求持续交错的到达，I/O 请求的连续性遭到破坏，所以 NOOP 和 Deadline 的吞吐量比其他 I/O 调度算法低。对于顺序负载，WPCAS 的吞吐量比 AS 提高了 10.9%左右。AS 算法为每类进程分配固定的预期周期（6 ms），对于顺序请求的负载，AS 虽然能够预测到下一个到达的连续请求，但也造成了某些密集 I/O(到达时间间隔小于 6 ms)长时间的等待。而 WPCAS 算法不但能够准确的预

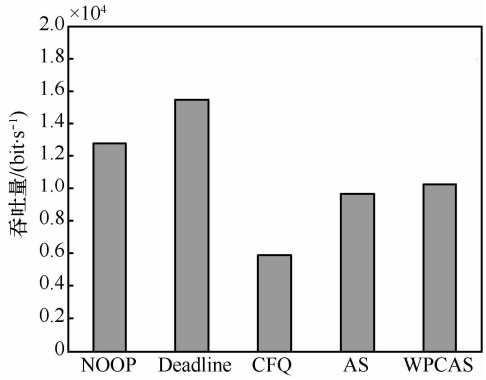


图 3 进程类别数为 10，顺序负载下各种 I/O 调度算法的吞吐量

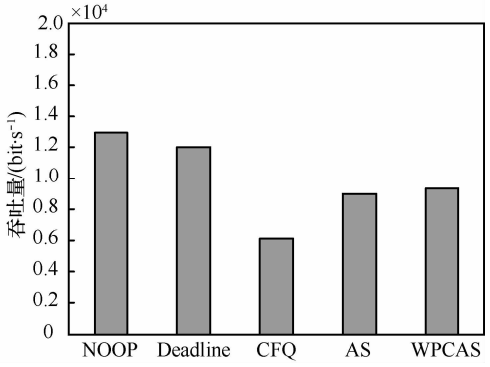


图 4 进程类别数为 40，顺序负载下各种 I/O 调度算法的吞吐量

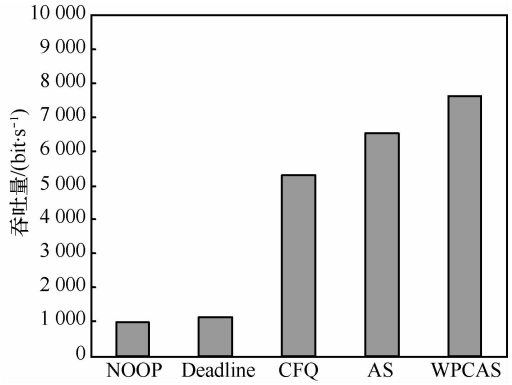


图 5 进程类别数为 80，顺序负载下各种 I/O 调度算法的吞吐量

测下一个连续的请求，而且根据进程特性为每类进程分配不同的预期周期，能够适应负载的动态变化。因此，WPCAS 算法的吞吐量比 AS 高。

4.4.2　随机读写 512 KB 大小文件的吞吐量

图 6~图 8 显示了 I/O 调度算法在随机负载下的吞吐量。可以看出，CFQ 算法对于随机读写吞吐量高于其他调度算法。这是由于 CFQ 为每个进程维护一个 I/O 队列，各个进程发出的 I/O 请求以轮循的方式处理。因此，CFQ 非常适合于随机、零散的 I/O 请求。由于 NOOP 算法按照先来先服务进行调度，对于随机负载，会造成磁头的频繁移动，增加寻道时延。因此，NOOP 的随机读写性低于 AS 和 WPCAS。在随机负载下，WPCAS 比 AS 吞吐量提高了 5%左右。这是由于在随机负载下，每类进程的 I/O 请求的到达时间间隔差别很大，使采用固态预期周期长度的 AS 预期失败或等待时间过长。而 WPCAS 根据负载特性动态改变预期周期的长度，使预期更加准确和高效。

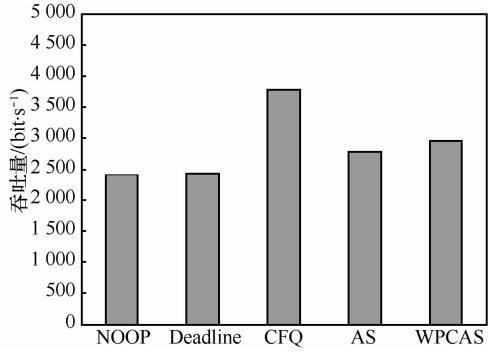


图 6　进程类别数为 10，随机负载下各种 I/O 调度算法的吞吐量

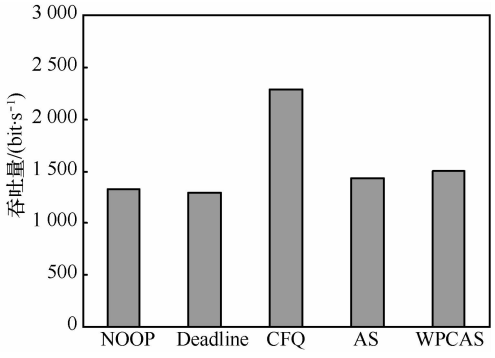


图 7　进程类别数为 40，随机负载下各种 I/O 调度算法的吞吐量

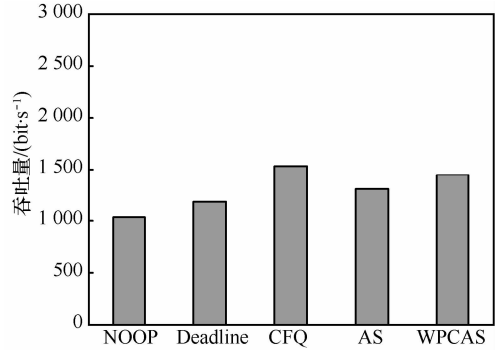


图 8 进程类别数为 80，随机负载下各种 I/O 调度算法的吞吐量

4.5　Postmark 生成的 Web 服务器负载

为了进一步评估 WPCAS 算法的性能，使用 Postmark 模拟真实的 Web 服务器负载，比较了 95%-Heuristic、AS 和 WPCAS 的性能。如表 1 所示，A-T、A-S 和 A-F 分别表示总的预期请求数、预期成功的请求数和预期失败的请求数。A-ST 和 A-FT 分别表示预期成功所花费的总时延和预期失败所花费的总时延。95%-Heuristic、AS 和 WPCAS 的伪空闲周期长度分别为 0.238 ms、0.113 ms 和 0.047 ms。WPCAS 的吞吐量比 AS 和 95%-Heuristic 分别提高了 24%和 40%。这是由于访问 Web 服务器的负载类型是动态变化的。例如，Web 服务器上的某一“热点”新闻，可能引起短时间内大量的 I/O 请求到达，I/O 非常密集。在一段时间之后，这些“热点”的新闻逐渐成为“冷门”，可能很长时间才会有 I/O 请求到达。AS 算法对所有的负载采用固态的预期周期(6 ms)，增加了预期失败的次数和预期成功的所花费的总时延。WPCAS 针对不同类型的负载动态的调度预期周期的长度，适应了负载的动态变化。因此，WPCAS 非常适用于 Web 服务器负载、网站访问负载等。

表 1　**Postmark 生成的 Web 服务器负载**

调度算法	95%-Heuristic	AS	WPCAS
总预期数	384 630	371 464	361 268
预期成功数	362 706	364 590	356 694
预期失败数	21 924	6 874	4 574
成功时延	335 140 ms	214 014 ms	140 768 ms
失败时延	86 536 ms	41 244 ms	16 250 ms
伪空闲周期长度	0.238 ms	0.113 ms	0.047 ms
吞吐量	27 415 kbit/s	34 231 kbit/s	45 238 kbit/s

5　结束语

针对传统 AS 调度算法的不足，提出了一种基于负载特性和服务时间评估改进的 AS 算法(WPCAS)。WPCAS 分为进程归类模块(PC)和服务时间评估模块(STE)两部分。PC 根据负载特性，动态调整预期周期的长度，从而适应负载动态变化。

STE 根据 I/O 请求的服务时间决定是否预期下一个 I/O 请求，使得预期更加的准确。通过对比实验证明了在吞吐量、预期成功的平均时延和伪空闲周期长度等方面，WPCAS 算法优越于 95%-Heuristic 和 AS 算法。特别地，WPCAS 算法非常适用于 Web 网络服务器负载。

参考文献：

[1] WORTHINGTON B, GANGER G, PATT Y. Scheduling algorithms for modern disk drives[A]. In ACM Sigmetrics[C]. 1994. 241-251.

[2] HUANG L, CHIUEH T. Implementation of a Rotation Latency Sensitive Disk Scheduler[R]. Technical Report ECSL-TR81, SUNY, Stony Brook, 2000.

[3] REDDY A L N, WYLLIE J. Disk scheduling in a multimedia I/O system[A]. Proc First ACM Int'l Conf Multimedia (MULTIMEDIA'93)[C]. 1993.225-233.

[4] CHANG R I, SHIH W K, CHANG R C. Deadline-modification-scan with maximum scannable-groups for multimedia real-time disk scheduling[A]. Proceedings of the 19th IEEE Real-Time Systems Symposium[C]. 1998.40-49.

[5] CHANG H P, CHANG R I, SHIH W K, *et al*. Reschedulable-group-SCAN scheme for mixed real-time/non-real-time disk scheduling in a multimedia system[J]. J Syst Software, 2002, 59(2): 143-152.

[6] CHANG H P, CHANG R I , SHIH W K, *et al*. GSR: a global seek-optimizing real-time disk-scheduling algorithm, the journal of transactions in firm real-time database systems[J]. Systems and Software, 2007, 80(2):198-215.

[7] KESAVAN M, GAVRILOVSKA A, SCHWAN K. Differential virtual time (DVT): rethinking I/O service differentiation for virtual machines[A]. Proc of the 1st ACM Symposium on Cloud Computing, SoCC'10[C]. ACM, New York (2010), 2010. 27-38.

[8] RIZZO L, VALENTE P. Hybrid: achieving deterministic fairness and high throughput in disk scheduling[A]. Proc CCCT '04[C]. 2004.

[9] WACHS M, ABD-EL-MALEK M, THERESKA E, *et al*. Argon: performance insulation for shared storage servers[A]. Proc of the 5th USENIX Conference on File and Storage Technologies (FAST'07)[C]. San Jose, CA, USA, 2007.61-76.

[10] GULATI A, MERCHANT A, VARMAN P J. Pclock: an arrival curve based approach for QoS guarantees in shared storage systems[J]. SIGMETRICS Performance Evaluation Rev, 2007, 35(1):13-14.

[11] GULATI A, MERCHANT A, UYSAL M, *et al*. Efficient and Adaptive Proportional Share I/O Scheduling[R]. Hewlett- Packard Pdf, 2009.

[12] GANG P, CKER CHIUEH T. Availability and fairness support for storage qos guarantee in distributed computing systems[A]. ICDCS'08. The 28th International Conference on 2008[C]. 2008. 589-596.

[13] VALENTE P, CHECCONI F. High throughput disk scheduling with fair bandwidth distribution[J]. IEEE Transactions on Computers, 2010, 59(9): 1172-1186.

[14] http://mirror.linux.org.au/pub/linux.conf.au/2007/video/talks/123.pdf [EB/OL]. 2010.

[15] LOVE R. Linux Kernel Development[M]. Developers Library Sams Publishing/Novel, 2005.

[16] STOUPA K, VAKALI A. QoS-oriented negotiation in disk subsystems[J]. Data and Knowledge Engineering Journal, 2006, 58 (2): 107-128.

[17] TANENBAUM A S. Modern Operating Systems[M]. Second Ed Prentice Hall, Upper Saddle River, NJ, 2001.

[18] LYER S, DRUSCHEL P. Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous I/O[A]. Proceedings of the 18th ACM Symposium on Operating Systems Principles[C]. New York, NY, 2001.

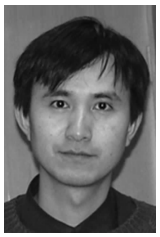
[19] RUEMMLER C, WILKES J. An introduction to disk drive modeling[J]. IEEE Comput, 1994, 27(3): 16-28.

[20] IOzone file system benchmark[EB/OL]. http://www.iozone.org.

[21] KATCHER J. Postmark: a New File System Benchmark[R]. Technical Report TR 3022, Network Appliance, 1997.

作者简介：

陈金忠（1986-），男，安徽巢湖人，哈尔滨工程大学博士生，主要研究方向为 Linux 存储系统、I/O 调度算法和固态硬盘。



姚念民（1974-），男，黑龙江大庆人，博士，哈尔滨工程大学教授、博士生导师，主要研究方向为网络存储、信息安全、无线传感器网络以及高可信计算。

蔡绍滨（1973-），男，黑龙江哈尔滨人，博士，哈尔滨工程大学教授、博士生导师，主要研究方向为无线传感器网络、水声传感器网络。

孙美玲（1985-），女，黑龙江五常人，哈尔滨工程大学硕士生，主要研究方向为信号与信息处理、语音识别。