基于 Petri 网模型的磁盘日志记录算法与性能分析*

周枫[1], 林闯[1], 郑纬民[1]

[1](清华大学 计算机系, 北京 100084);
E-mail: zhoufeng00@mails.tsinghua.edu.cn

摘要: 磁盘日志是数据库系统、文件系统中常用的数据结构，同步操作的特性使日志经常成为整个系统的性能瓶颈。日志操作的磁盘调度算法是提高日志操作性能的一条途径，本文提出了一种新的磁盘日志记录算法——延迟最优化的日志记录算法。在建立磁盘硬件的高级 Petri 网模型的基础上，建立了连续扇区的日志记录、基于磁道的日志记录、延迟最优化的日志记录三种算法的模型并进行性能分析。分析和模拟结果显示后两者都可以提高日志性能近一个数量级，同时新算法在高负载情况下性能大大优于原有算法。

关 键 词: 磁盘日志，Petri 网，性能分析

中图法分类号: *分类号;分类号* **文献标识码**: A

1 引言

磁盘是目前计算机系统中主要的外存设备，由于其机械运动的本质，磁盘的速度与内存、处理器相差数个数量级，因此往往成为系统性能的瓶颈。日志是一种常用的盘上数据结构，在操作系统及应用程序都有应用，例如数据库系统的事务日志、文件系统的元数据操作日志和科学计算应用的检查点日志等。日志本质上是持久的线性表，一般允许从一端插入数据，从另一端读取数据或者随机读取数据。日志一般用来保存关键的中间数据，比如在数据库中用来保存一个事务进行中修改的数据。对日志的操作往往很频繁，因此日志操作的性能很重要。例如在很多商用数据库系统中，如何分配日志文件或分区，对系统的性能影响非常大。本文就主要讨论日志记录操作（Logging）的算法和性能问题。

响应时间和带宽是日志记录的主要性能参数。由于日志操作在整个系统范围内一般是串行的，所以日志操作的响应时间与带宽基本成反比关系，其中原因首先是因为日志操作必须是同步的，以保证可靠，因此在单个进程中无法并发进行，其次在多个进程间往往也没有并发性，例如一个数据库一般只有一个事务日志，所以尽管对访问是并发的，但日志记录却是串行的。所以在本文中我们将响应时间作为主要考察的性能参数。

日志可以利用普通的 I/O 操作实现，例如文件操作。但是日志写入包含大量的小块数据同步写操作，直接通过文件系统进行日志写入操作，将得到非常差的性能（可能仅有磁盘最大带宽的 1/100-1/1000）。

另一种方案是越过文件系统，直接将日志按优化的组织方式写入磁盘，以缩短操作的响应时间。主要的思路是将数据写在离磁头当前位置较近的位置，以缩短磁头运动和等待旋转的时间，采用这样方法的设计如 TRAIL[1，2]，DCD[3]。文献[2]中的结果表明这可以使日志操作的速度提高一个数量级以上，我们的模拟和实验结果也类似，而且在我们的实验系统上某个数据库系统的事务操作性能提高了 5 倍以上。

在本文中，我们提出了一个新的日志记录算法——延迟最优化的日志记录，与 TRAIL 中使用的基于磁道的日志记录算法不同，新算法在某些情况下在一个磁道上写多个记录，力图通过减少寻道操作，进一步降低

* **收稿日期**: 2001-??-??; **修改日期**: 2001-??-??

基金项目: 国家自然科学基金(69873012); 国家重点基础研究发展规划项目(G1999032702; G1999032707)

作者简介: **周枫**(1977－),男,江苏无锡人,硕士生,主要研究领域为计算机体系结构; **林闯**(1948－),男,辽宁沈阳人,博士,教授,,博士生导师,研究方向是系统性能评价,计算机网络,随机 Petri 网,逻辑推理模型等; **郑纬民**(1946－),男,浙江宁波人,教授,博士生导师,研究方向有并行处理与分布计算机系统,面向 AI 体系结构以及说明性语言的编译方法和程序开发环境等.

操作的延迟时间。为了分析算法的性能，我们首先使用 Petri 网构造一个磁盘存取的简单模型，然后以此模型为基础，分别构造扇区连续的日志记录、基于磁道的日志记录（TRAIL 系统中使用的算法），以及延迟最优化的日志记录的高级 Petri 网模型。

利用 Petri 网工具 RENEW[4，5]进行模拟分析的结果说明，两种优化的磁盘日志记录方法比直接通过文件系统进行日志都有 1 个数量级以上的性能提高，同时显示了新算法在较高的负载下有更好的性能，因此能使整个系统日志记录的带宽有较大的提高。

本文同时显示了高级 Petri 网对 I/O 系统建模与分析有相当的帮助， Petri 网能较好地表达 I/O 系统的并发、随机等特性，同时具有综合控制流与数据流的能力，可以使得系统中资源的分配、回收等过程得到清楚显示[6，7]。利用一定的 Petri 网工具，可以在算法模型的设计过程中模拟运行网，快速检验其正确性和合理性，加速了算法的设计与改进过程。设计完成后，使用模拟分析功能，可以得到算法在各种负载下的性能参数，以比较不同算法的优劣。

2 建模工具

算法建模的目的是在着手实现之前分析算法的行为，确认算法的正确性，比较不同算法的性能。I/O 系统具有并发性、随机性的特点，简单的流程图、块图等工具不足以清楚反映系统的动态行为。我们选择高级 Petri 网作为建模工具，考虑的主要需求有：1.清楚反映算法的控制语义与数据流；2.可执行，可观察算法的动态行为；3.可以提取统计信息，以便分析算法性能；4.能够模拟并发、随机的系统行为，并有时间度量；5.易于构造，易于修改和重用，具有层次性和面向对象的能力。

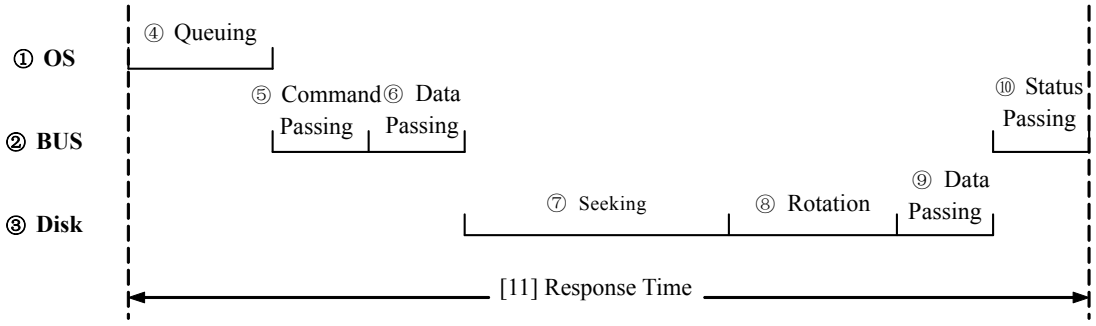
高级 Petri 网能很好满足这些要求，我们考虑的其它选择有：(1) P/T 网，清楚简单，但不满足条件 4；(2) 状态机，能够反映系统的控制流，但缺乏数据流的表现能力，而且随着算法复杂程度增加，状态数爆炸。

我们使用的软件工具是 RENEW[6]，RENEW 使用的网模型是参考网（Reference Net），参考网是与 Java 语言集成的一种高级 Petri 网，除一般高级 Petri 网的特点之外，还可以直接调用外部 Java 类的方法，便于表达复杂的判断和计算步骤，RENEW 可以对网模型进行单步或连续的模拟运行，得到模型的性能参数，同时具有方便的图形用户界面。

3 磁盘 I/O 系统的性能模型

我们在整个磁盘 I/O 系统模型构造中采用层次式的方法，先构造磁盘硬件系统的存取模型，然后在经过验证的磁盘硬件系统模型上构造各个算法的模型，这样保证了结果的正确性，同时使模型模块化。

我们主要关心 I/O 性能，因此读写的内容不是模型的一部分，模型主要描述的是操作的时间特性，要考虑磁盘 I/O 系统各主要组成部分（操作系统、总线、磁盘）的延迟。图 1 显示了在 SCSI 总线上一次磁盘操作的各部分时间组成：



①操作系统, ②总线, ③磁盘, ④排队, ⑤命令传输, ⑥数据传输, ⑦寻道, ⑧旋转, ⑨数据传输, ⑩状态传输, [11]响应时间

Fig.1 Layered Time Composition of Disk Operations

图1 磁盘操作的分层时间组成

我们首先建立磁盘硬件系统（包括上图中总线和磁盘两层）模型，后文将此模型简称为“磁盘模型”。我们模型一个 IBM DDYS-T36950 磁盘，这是一个 10000RPM 的 SCSI 接口磁盘，容量约 35G，有 3 个盘片，6 个磁头。为了避免模型过于复杂，我们作了一些简化：

1. 假设只有一个盘片，这样跨超一个磁道的延迟（track skew）将是恒定值；
2. 假设每个磁道的扇区数相等，而实际上盘片外部的每个磁道扇区数要多一些。

这两个简化对模型的准确程度将没有太大的影响，但却能大大简化模型，模型的其它参数都依据 DDYS-T36950 的手册和实测结果，具体如表 1：

Table 1 Disk Model Parameters

表1 磁盘模型参数

Name①	Value②	Comment③
Nc	15000	Number of cylinders④
Ns	500	Sectors per track④
Tr	6.0 ms	Time per round⑥
Tskew	1.6 ms	Track skew⑦
Tseek	10.6 ms	Full stroke seek⑧
Tbus	0.5 ms	Other latencies（Bus latency， SCSI overhead）⑨

①名, ②值, ③说明, ④柱面数, ④每磁道扇区数, ⑥每圈旋转时间,
⑦道间延迟, ⑧全程寻道时间, ⑨其它延迟(总线延迟, SCSI 开销)

根据这些参数，可以得到“寻道”和“读写”（忽略读、写两种操作时间特性的微小差别）两种基本操作的服务时间：

$$T_{seek}(c',c)=\begin{cases}T_{skew}+\dfrac{(T_{seek}-T_{skew})(c'-c)}{N_c} & c'\neq c\\ 0 & c'=c\end{cases},$$

$$T_{readwrite}(c',s',n,c,s)=T_{bus}+T_{seek}(c',c)+T_{rotate}(s',s'')+T_{access}(n)$$

$$\text{Where: } s''=\left(s-\frac{|c'-c|\cdot T_{skew}\cdot N_s}{T_r}+\frac{T_{seek}(c',c)\cdot N_s}{T_r}\right)\bmod N_s,\qquad T_{rotate}(s',s)=\begin{cases}\dfrac{(s'-s)T_r}{N_s} & s'\geq s\\ \dfrac{(s'+N_s-s)T_r}{N_s} & s'<s\end{cases}$$

$$T_{access}(n)=\frac{nT_r}{N_s}$$

Fig.2 Service Time of *Seek* and *Read/Write* Operations

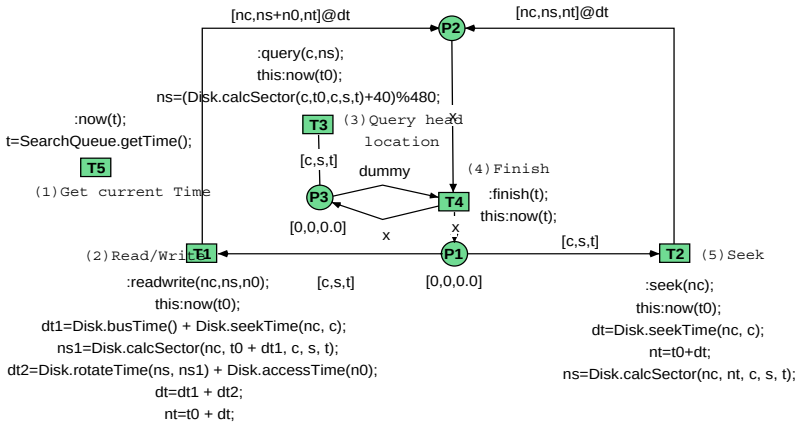
图 2 “寻道”和“读写”操作的服务时间

参数 c,c’分别表示当前磁头所在柱面号和目的柱面号，s,s’分别表示当前磁头所在扇区号和目的扇区号，n 表示读写的扇区数。中间结果，s’’，Trotate，Taccess 分别是寻道结束时的磁头扇区位置、旋转延迟和数据传输时间。

根据以上结果，我们可以建立磁盘模型如图 3：

简单介绍参考网的语法：与 P/T 网类似，矩形表示变迁，圆形表示处所，作为高级网的特性，变迁上可以有铭记（Inscription），以表示这个变迁实施的进一步条件，以及进行必要的计算，还可以调用外部方法完成较复杂的计算，例如 T1 的铭记中，调用 Disk.busTime(), Disk.seekTime()等外部方法，分别计算总线延迟和寻道延迟，这即是图 2 中的公式，具体在外部程序 Disk.java 中实现（具体实现略）。标记除了可以是整数、实数等原子类型外，还可以是元组，用[a,b,…]表示，如 P1 中用三元组[c,s,t]表示 t 时刻，磁头位置是 c 柱面、s 扇区。变迁的实施时间在弧上用 token@time 表示，这与一般的高级网在变迁上表示时间不同，输入弧上的时间表示变迁的可实施延迟，输出弧上的时间表示输出的标记的存在延迟。

模型的主要变迁有：T1（读写），T2（寻道）和 T3（查询当前状态），这些同时也是此模型与外部的接口，接口使用“同步通道（Synchronous channel）”来实现，在图中即是“:readwrite(nc,ns,n0)”等变迁铭记，上层模型调用这此同步通道的时候（相应的语法是 disk:readwrite(…)），相应的变迁将实施，同步通道是参考网实现模型的模块化的主要手段。



(1)取得当前时间,(2)读/写,(3)查询磁头位置,(4)完成,(5)寻道

Fig.3 High Level Petri Net Model of a Disk System

图 3 磁盘系统的高级网模型(disk.rnw)

请注意此模型中使用的一个具有普遍意义的技巧，即如何用 Petri 网模型象磁盘这样状态连续变化的系统，思路是在连续变化的状态中找出一段时间内的“不变量”，然后以此“不变量”为状态进行建模。

具体来说，磁盘自身的运动是连续时间过程，其状态（磁头位置）是不断变化的，而 Petri 网表示的是离散时间过程，因此要对状态进行一些转化，使其成为离散变化的过程，才能使用 Petri 网表示。如前所述，我们使用 [柱面号 c，扇区号 s，时间 t] 三元组来表示磁盘状态，就是为了使状态离散化，在 t 时间后如果没有事件（寻道、读写）发生，那任何时间的磁头位置都可通过此三元组确定，因此可以认为此三元组表示了磁盘在 t 时刻到下一次事件发生时刻之间的状态。

4 日志记录算法的 Petri 网模型

日志记录算法属于磁盘 I/O 系统分层模型中 OS 层，针对日志这一特写操作实现。我们对三种不同的日志记录算法进行建模。为了问题分析的简便，有如下模型假设：

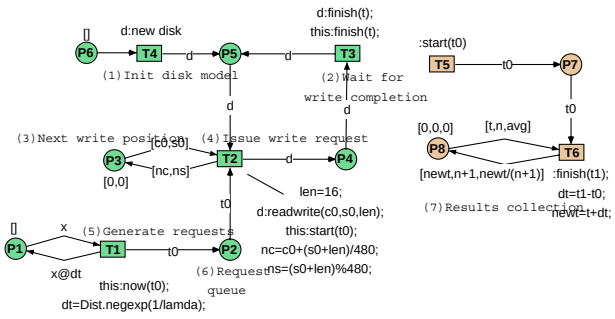
1. 请求的到达是速率为 λ 的泊松过程；
2. 每次日志长度固定，为 16 个扇区（8K 字节）；
3. 使用上文定义的磁盘模型。

评价算法优劣的主要标准是在不同的请求到达速率下的平均响应时间和吞吐率，响应时间越短越好，吞吐率越大越好。

4.1 算法1——连续扇区的日志记录

连续扇区的日志记录指的是将日志写在磁盘上连续的位置上，通过文件进行日志（不考虑文件碎片的影响）或在应用程序中直接操作磁盘设备进行日志，都可以认为是连续扇区的。这是最常用的日志方法，因此其性能是我们讨论改进算法的基准。

图 4 是算法 1 的参考网模型。图中左方是算法部分，右方是结果收集部分，记录总计响



(1)初始化磁盘模型,(2)等待写操作完成,(3)下次写位置,
(4)发出写操作,(5)生成请求,(6)请求队列,(7)结果收集

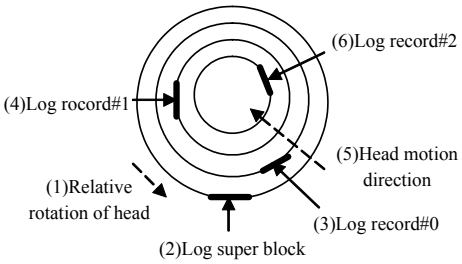
Fig.4 Algorithm 1 – Sector-Continuous Logging

图 4 算法 1 - 连续扇区的日志记录

应时间、完成请求次数和平均响应时间。图中 P1,T1,P2 是请求的产生和请求队列，T1 的铭计中，调用外部方法 Dist.negexp()以产生负指数分布的请求标记（参数为 lamda），放入队列 P2 中。P6 和 T4 使用 new disk 调用，初始化一个磁盘模型，放入 P5 中，可以看到，模型中所有名为 d 的标记都是此磁盘模型。T2 主要调用 d:readwrite()通道，进行写操作，更新 P3 中的下一次写位置，并调用本模型中的 start 通道（this:start()，就是右方的 T5），记录操作的开始。T3 使用 d:finish()同步通道，等待写操作的完成，并调用 this:finish()记录这次操作的响应时间。

4.2 算法2——基于磁道的日志记录

分析表 1 中的磁盘时间参数即可发现，对于小块数据的读写，数据传输的时间很短，而寻道时间和旋转等待时间是主要的。对日志记录操作，因为写入一般是连续的，因此寻道时间较小，主要的时间是旋转延迟。如果我们在一个磁道上只记录一条或少数几条日志条目，将数据就近写入磁头的下方，就可以减少旋转延迟。写完一条或数条日志记录后，在合适的时机将磁头移动到下一磁道，以便进行后面的操作。这是算法 2 和算法 3 的基本思路。



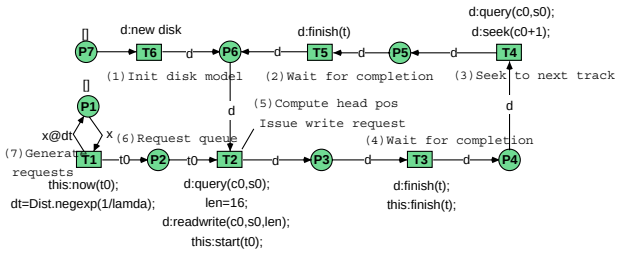
(1)磁头相对旋转,(2)日志超级块,(3)日志记录#0,
(4)日志记录#1,(5)磁头运动方向,(6)日志记录#2

Fig.5 Track-based Logging

图 5 基于磁道的日志示意图

这样的算法在实现时，需要与磁盘硬件进行直接的通信，因此最好在操作系统中实现。因为目前的磁盘接口中，写操作都必须给出具体地址，因此要将数据写入磁头的下方，需要查询或预测磁头的位置，这可通过 SCSI 命令、微基准测试等数种方式得到磁盘物理参数后完成；写入不确定位置的日志记录经过简单的编码，以便读取时识别；磁头的移动等物理操作都可通过 SCSI/IDE 命令集来直接或间接实现。这些问题本文不予讨论。

基于磁道的日志记录（Track-based Logging），在文献[2][3]中首次提到，此算法将磁道作为日志操作的基本单位，每次将一个记录写在磁头的下方，写完后立即将磁头移到下一个磁道。如此重复。



(1)初始化磁盘模型,(2)等待写操作完成,(3)寻道到下一磁道,(4)等待操作完成,(5)计算磁头位置 发出写操作,(6)请求队列,(7)生成请求

Fig.6 Algorithm 2 – Track-based Logging

图 6 算法 2 – 基于磁道的日志记录

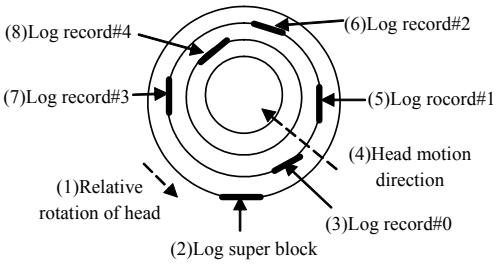
图 5 显示这样记录的日志在磁盘上的大致分布：

算法 2 的参考网模型如图 6 所示。结果的收集部分与算法 1 类似，因此略去。请求的生成、排队与模型的初始化与算法 1 也类似。主要的不同在写操作地址的确定，在 T2 中，通过调用 d:query()通道，查询到下一次读写操作最接近的地址（这其中考虑了总线延迟），然后根据此地址开始写操作。等待写操作完成后，T4 将磁头移动到下一道。

4.3 算法3——延迟最优化的日志记录

可以预计，算法 2 比算法 1 将有很大的性能提高，因为算法 2 将旋转延时降到了几乎为 0。但是算法 1 还有可以改进的地方，直观地看，在一个道上写多个记录，在某些情况下将可以提高性能。因为当负载较大时，连续的几次日志操作，如果写在同一道上，则明显可以减少寻道的次数，因为将磁头移动到下一磁道的时间不能忽略（对参考磁盘来说是 1.6ms）。图 7 显示了这样的日志在磁盘上的大致分布：

根据这个想法我们提出算法 3-延迟最优化的日志记录，力图在负载较低的情况下得到与算法 2 类似的行为，而在负载较高的情况下达到更高的吞吐率。算法 3 的参考网模型见图 8。



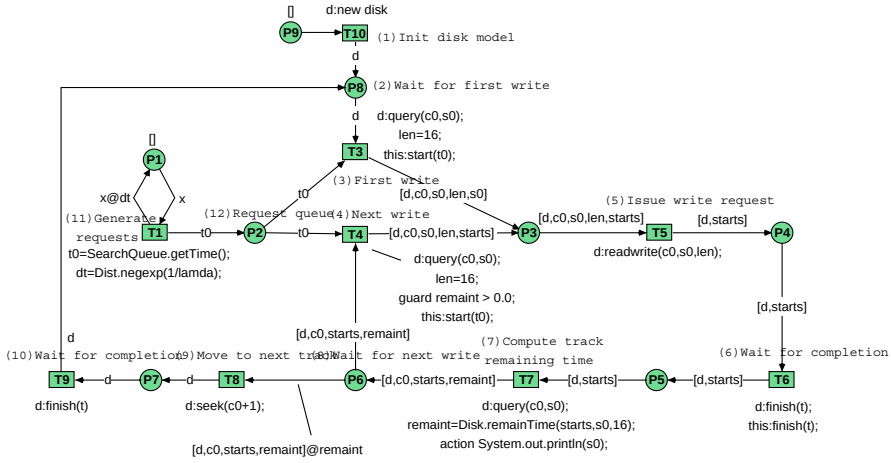
(1)磁头相对旋转,(2)日志超级块,(3)日志记录#0,(4)磁头运动方向, (5)日志记录#1, (6)日志记录#2,(7)日志记录#3,(8)日志记录#4

Fig.7 Latency-Optimized Logging

图 7延迟最优化的日志示意图

算法 3 比算法 2 要复杂一些，但基本结构是类似的。磁盘同样是每个请求完成过程中必须占用的资源，不同的是在算法 3 中，每次请求由 T3 或 T4 两个变迁之一开始执行，两个变迁分别表示当前磁道的第一次写操作和后序的所有写操作（重复写），相应的 P8 和 P6 分别是磁盘资源的等待位置，P2 则是请求队列。与算法 2 类似，T5 发出写操作命令，T6 等待写操作完成，而 T7 则利用记录下的此道第一次写的扇区位置 starts 和当前扇区位置 s0，调用外部方法 Disk.remainTime()计算当前道从 starts 开始到转满一圈的剩余时间。注意 T8 与 T4 的竞争关系，因为 T4 是瞬时变迁而 T8 是时间变迁，故 T4 具有比 T8 高的优先级，这保证了如果剩余时间足够，算法就会在一道上写多个记录（在 T4-T5-T6-T7 间循环），如果剩余时间用光，则 T8 实施，寻到下一磁道，然后返回 P8 等待下一次请求。

与算法 2 的行为对比，可以发现，如果请求到达速率很小，那么每道上将只写一条记录，只要磁盘标记及时回到 P8，每次请求到来时将不需要等待而立即执行，这时算法沿 T3-T5-T6-T7-T8-T9 路线循环运行。而当请求到达速率很大时，算法将基本沿 T4-T5-T6-T7 路线循环运行，这时与算法 2 相比，省去了每次操作间寻道的时间。



(1)初始化磁盘模型,(2)等待首次写,(3)首次写,(4)重复写,(5)发出写操作,(6)等待操作完成,(7)计算此道剩余时间,(8)等待重复写,(9)移动到下一道,(10)等待操作完成,(11)生成请求,(12)请求队列

Fig.8 Algorithm 3 – Latency-Optimized Logging

图 8 算法 3 - 延迟最优化的日志记录

因此从 Petri 网模型的直观分析上看，算法 3 在较低负载情况下具有与算法 2 类似的低延迟特性，而在较高负载下又能达到比算法 2 更高的吞吐率。下面我们将用定量方法来验证这一结论。

5 模型求解及结果分析

要得到网的性能参数，一般有分析求解与模拟两大类方法，由于磁盘系统中非线性的因素相当多，因此分析求解是比较困难的。同时参考网本身是侧重于可视化与模拟，而不是分析的，这从本文中的模型也可以看出，模型中涉及的计算较多，而且有外部程序的辅助。因此下面的模型求解中，我们以模拟为主，分析为辅，来考察各算法的性能。

首先我们可以发现算法 2 是非常简单的，在请求大小固定的情况下，服务时间可以认为就是“写入时间+寻道时间”，是一个固定值，因此是 M/D/1 的排队系统。稍有不同的是，我们关心的请求平均完成时间中不

包括寻道时间，但这并不影响系统的性质。

利用 M/D/1 的性能公式，得到：$\rho = \frac{\lambda}{\mu} = \lambda T_s, T_q = \frac{T_s(2-\rho)}{2(1-\rho)}$。

其中服务时间 Ts=Tbus+Tr*n/Ns+Tskew=2.3ms，我们所求的请求完成时间 Treq=Tq-Tskew=Tq-1.6，因此得到 Treq-λ 关系图如图 9 所示：

图 9 中的实线是按公式计算的准确结果，而交叉点是使用 RENEW 对算法 2 的 Petri 网模型进行模拟得到的结果(每个数据点都模拟了 2000 个请求的完成)。模拟结果是比较准确的，这从一方面验证了算法 2 模型的正确性。

算法 1 和算法 3 无法用简单的排队模型来描述，我们对这两个算法的模型也使用 RENEW 进行模拟，即可得到这三个算法在不同请求到达速率下的延迟与吞吐率关系比较，如图 10 和图 11 所示：

可见首先与传统的算法 1 相比，算法 2、3 的响应时间在大部分的负载情况下都要低一个数量级左右，这表明，利用磁盘的物理运动特点，对日志操作进行优化，可以大大降低操作响应时间，提高系统的最大吞吐率。同时，算法 3 在所有负载情况下性能都要比算法 2 好，特别是允许的最大负载几乎是算法 2 的 3 倍，因此算法 3 在我们讨论的典型磁盘参数下具有比算法 2 好得多的性能。改进的幅度与寻道、旋转和总线延迟时间都有关系，但算法 3 总是具有比算法 2 好的性能是肯定的。

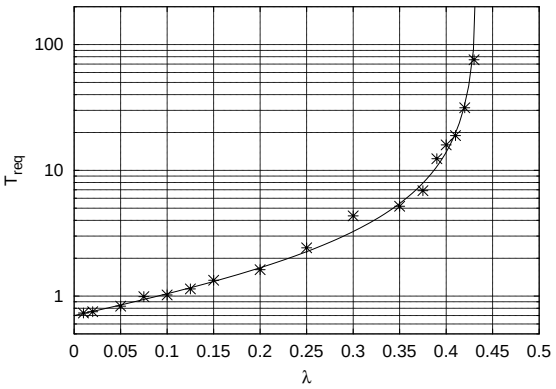
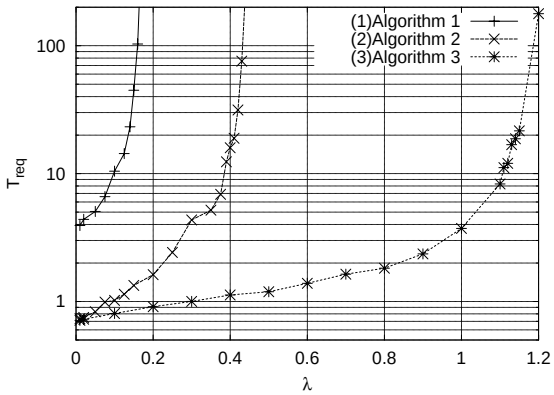


Fig.9 Treq-λ Graph of Algorithm 2

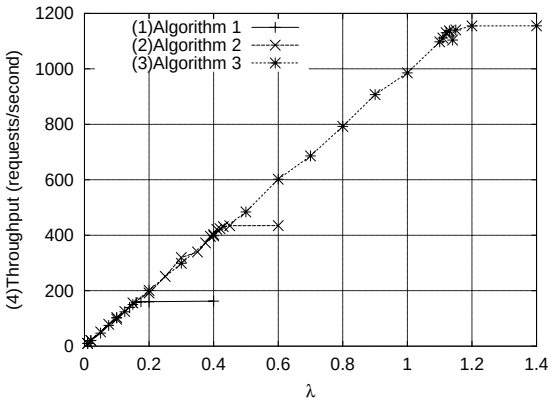
图 9 算法 2 的 Treq-λ 关系图



(1)算法 1,(2)算法 2,(3)算法 3

Fig.10 Average Response Time

图 10 平均响应时间



(1)算法 1,(2)算法 2,(3)算法 3,(4)吞吐率(请求/秒)

Fig.11 Throughput

图 1011 吞吐率

6 结论

本文的主要贡献是：(1) 构造了一个磁盘硬件系统的简化高级 Petri 网模型；(2) 提出了一个新的磁盘日志记录调度算法（算法 3）；(3) 对传统的扇区连续的日志记录、基于磁道的日志记录和我们提出的延迟最优化的日志记录等三个算法，分别建立了高级 Petri 网模型；(4) 通过对三个模型的模拟分析，证明了通过使用优化的日志记录算法，能提高日志操作效率近一个数量级，这将会使数据库等应用性能有相当大的提高；(5) 证明了我们提出的算法 3 具有比算法 2 更好的性能，特别是能够承担较大的工作负载。

延迟最优化的日志记录算法在我们的原型系统 FastSync 中得到了应用，FastSync 是一个利用日志作为磁盘写缓存的 I/O 系统，利用了日志写低延迟的特点。FastSync 的文件写操作延迟是普通 Unix 磁盘写操作延迟的 1/8 以下，FastSync 对上层应用是完全透明的，在其上运行一般的 SQL 数据库系统，事务操作的速度即可达到原来的 1.5-8 倍。

除了作为性能评价的工具，Petri 网在我们的算法设计过程中也起到了相当大的作用。首先高级网具有很强的表达能力，我们所使用 RENEW 还具有与外部语言（Java）交互的能力，这使得我们一方面能够以可视的方式快速的表达算法或系统，同时也能用 Java 类来实现一些比较复杂的计算或逻辑。同时使用模拟功能，可以清楚地察看模型的工作过程，发现有错误时，可以非常方便地进行修改。

注：本文中的 Petri 网模型文件可以在http://hpc.cs.tsinghua.edu.cn/~zf/petrinets/下载。

References:

[1] Lan Huang Tzi-Cker, Trail: Track-Based Logging in Stony Brook Linux, http://citeseer.nj.nec.com/210226.html

[2] Tzi-Cker Chiueh Lan, Trail: A Fast Synchronous Write Disk Subsystem Using Track-Based Logging, http://citeseer.nj.nec.com/295845.html

[3] Y. Hu and Q. Yang, DCD---disk caching disk: A new approach for boosting I/O performance, Proceedings of the 23rd International Symposium on Computer Architecture, (Philadelphia, Pennsylvania), pp. 169-- 178, May 1996

[4] University of Hamburg, Department of Informatics, Theoretical Foundations Group, Distributed Systems Group, Reference Network Workshop(RENEW), http://www.renew.de/

[5] Kummer, Olaf; Wienberg, Frank, Renew - the Reference Net Workshop, Petri Net Newsletter No. 56, pages 12-16. April 1999

[6] Mattias Gries, Modeling a Memory Subsystem with Petri Nets: a Case Study, Workshop Hardware Design and Petri Nets HWPN98, Lisbon, Portugal, pp. 186—201

[7] Lin Chuang, Stochastic Petri Nets and System Performance Analysis, Tsinghua University Press, January 2000

附中文参考文献:

[7] 林闯，随机 Petri 网和系统性能评价，清华大学出版社，2000 年 1 月出版

Disk Logging Algorithms and Performance Analysis Based on Petri Net Models*

ZHOU Feng[1], LIN Chuang[1], ZHENG Wei-min[1]

[1](*Department of Computer Science and Technology*, *Tsinghua University*, *Beijing 100084*, *China*);

E-mail: zhoufeng00@mails.tsinghua.edu.cn

Abstract: Disk log is an important type of data structure used in databases and file systems. The synchronous nature of logging often renders it the performance bottleneck of the whole system. One way to solve this problem is using disk-scheduling algorithms optimized for logging. A new disk logging algorithm – latency-optimized logging - is proposed in this paper. A High Level Petri Net (HLPN) model of a disk subsystem and subsequently HLPN models of three logging algorithms (sector-continuous logging, track-based logging and latency-optimized logging) were built and performance of the algorithms analyzed. The results show that both of the latter two can improve logging performance by a factor of ten and the new algorithm we propose performs much better under heavy load.

Key words: Disk logging, Petri nets, performance analysis

---

* Received Month Data, 2001; accepted Month Data, 2001

Supported by the National Natural Science Foundation of China under Grant No. 69873012; the National Grand Fundamental Research 973 Program of China under Grant No. G1999032702 and No. G1999032707.