

遗传算法在 I/O 调度策略中的应用与研究

吕燕彬, 秦品乐

(中北大学 计算机与控制工程学院, 山西 太原 030051)

摘 要: 将遗传算法的思想应用到 I/O 调度策略的多参数调整过程中, 使策略参数根据存储系统 I/O 操作的阶段性统计量进行多轮迭代, 最终筛选出能够最优适应存储系统实时运行环境的策略参数, 发挥出最优的 I/O 调度性能. 改进后的 I/O 调度策略能够在较短时间内使策略参数收敛. 实验结果表明, 在长时间、重负载运行的存储系统中, 使用所提出的调度策略比使用原有调度策略有明显的 I/O 性能提升.

关键词: I/O 调度; 遗传算法; 存储系统

中图分类号: TP302.7

文献标识码: A

文章编号: 1000-7180(2016)03-0115-05

Research and Application of Genetic Algorithm in I/O Schedule Strategy

LV Yan-bin, QIN Pin-le

(School of Computer Science and Control Engineering, North University of China, Taiyuan 030051, China)

Abstract: In this paper, by applying the conception of genetic algorithm into the parameters turning process of I/O schedule strategy, it takes strategy parameters adapting the storage system according to the periodical statistic of I/O operation, and got a group of strategy parameters, which have the best adaption in real-time storage environment, and makes the I/O schedule strategy have a better performance. In the new way of tuning strategy parameters, it makes the strategy parameters resuming its convergence in a short time. The experimental results show that, in the storage system which is running long term and have a heavy load, the efficiency of computer's I/O operation which applies this paper's strategy has an obvious performance promotion than applying others.

Key words: I/O schedule; genetic algorithm; storage system

1 引言

近年来,随着各类大数据应用场景的不断涌现,海量数据的读写性能也越来越吸引各界学者的关注. 提高一个存储系统的性能,不仅要关注其存储容量,还要关注其读写效率. 在提高读写效率方面,针对随机发起的读写请求, Linux 内核通过 I/O 调度层对这些请求进行重新排序, 合并位置相邻的请求, 一次处理尽可能多的位置相邻 I/O 请求, 从而达到提高读写效率的目的. 目前, 有四种通用的 I/O 调度算法, 分别是: Noop 调度算法、Deadline 调度算法、Anticipatory 调度算法、CFQ 调

度算法^[1-3].

Noop 调度策略, 顾名思义, 在真正将请求发送到物理执行层之前, 它只对 I/O 请求队列做最基本的操作. 该算法将忽视 I/O 请求在系统上层的排序修改, 也将忽略底层存储硬件是否为真正的随机访问设备, 尽可能合并新到达的请求与已存在的请求.

事实上, 即使是同一种调度策略, 在不同的应用环境中也有着较大的性能差异. 如 Marcus Dunn 等人所展示的, Noop 调度策略对 I/O 请求只做简单的入列和合并操作, 这种策略环境下系统的读写性能很大程度上取决于读写进程的数量和进程访问数据的随机程度, 无法量化地概括其性能^[4]. Deadline、

Anticipatory、CFQ 三种调度策略各自性能的发挥依赖于参数设定是否合理,如使用 Deadline 策略时,对 deadline 队列的检查频度在机械臂磁盘和固态硬盘环境中是不一样的最佳适配值^[5]. 使用 Anticipatory 策略时,处理完每个 I/O 请求后到找到可合并的下一请求之间的等待时间,在实时性要求不同的环境中也有不同的最佳适配值. 使用 CFQ 策略时,依据进程对 I/O 依赖程度不同,每个 I/O 队列分配到的时间片长度应当依系统环境设置不同的值^[6]. 那么,如何确定这些参数的最佳取值成为了一个值得研究的课题.

为了解决上述问题,本文引入了遗传算法的思想.

2 遗传算法及其在参数调整中的应用

2.1 遗传算法

遗传算法的目的是通过种群迭代,最终筛选出拥有优质基因的个体. 交叉算子和变异算子对优质基因的“启发”作用是有限的^[7],变异算子的随机性较大,优质基因的产生具有不确定性,而交叉算子产生新基因的过程可以表示为

$$x'_1 = x_1 + \text{rand} \times (x_1 - x_2) \quad (1)$$

式中, x_1 和 x_2 是上一代个体中的两个基因,且 x_1 比 x_2 的适应度好(本文约定:适应度值越接近0,则有更好的适应度),若 x_1 和 x_2 不在同一局部最小点时,交叉算子产生的下一代新基因将会变得无意义,甚至可能破坏了上一代有较好适应度的基因. 由此,选择算子就成了关键环节,它起到了基因的“质量控制”作用^[8].

梁昔明教授等人提出了一种新的约束处理策略^[9],设适应度函数为

$$F_1(x) = f(x) + \lambda \left(\sum_{i=1}^m g_i(x) + \sum_{i=m+1}^l |h_i(x)| \right) \quad (2)$$

式中, $f(x)$ 是目标函数, λ 是惩罚因子, $g_i(x)$ 是不等式约束条件, $h_i(x)$ 是等式约束条件. 在基于基本约束处理函数的基础上,依据局部最小点的明确的等式型特征和可浮动的范围型特征,及加权惩罚因子 λ ,构造出接近较好适应度的基因,这种设计在一定程度上能使种群沿着约束值减小的方向进化.

惩罚因子 λ 按照惩罚项的平均增量不高于目标函数的平均增量设计,可表示为

$$\lambda = \begin{cases} \frac{\frac{1}{p_1} \sum_{x \in \Gamma} f(x) - \frac{1}{p - p_1} \sum_{x \notin \Gamma} f(x)}{\frac{1}{p - p_1} \sum_{x \notin \Gamma} \left(\sum_{i=1}^m g_i(x) + \sum_{i=m+1}^l |h_i(x)| \right)}, & p_1 > p \times 0.1 \\ \frac{\text{std}(f(x))}{\text{std} \left(\sum_{i=1}^m g_i(x) + \sum_{i=m+1}^l |h_i(x)| \right)}, & x \notin \Gamma, \\ & p_1 \leq p \times 0.1 \end{cases} \quad (3)$$

式中, p 为个体总数, p_1 为拥有优质基因的个体的数量, Γ 标记为拥有优质基因的个体集合,std 为标准差函数.

2.2 遗传算法在调度策略参数调整中的应用

Deadline 调度策略的实现,依赖四个参数:fb,ws,re,we.

待处理的请求被划分为若干批次,每批次处理的 I/O 请求数量由参数 fb 定义.

选择下一个处理的请求时优先选择读请求,阻塞写请求,当阻塞次数达到上限时,转而处理一次写请求,并清零阻塞次数计数器,重新优先处理读请求. 这个阻塞次数上限由参数 ws 定义.

基于以上 I/O 请求处理的优先策略,取得下一读请求或写请求之前,检查是否有超时未处理的请求,若有则优先处理超时的请求. 每个读请求和写请求的超时时限分别有参数 re 和 we 定义,单位为系统节拍.

本文环境中,决策变量 x 定义为由上述四个参数组成的策略参数向量,即:

$$x = [\text{fb}, \text{ws}, \text{re}, \text{we}] \quad (4)$$

顺序执行选择算子、交叉算子、变异算子为一轮迭代,每轮迭代的输出将作为下一轮迭代的输入.

存储系统的读请求压力占总 I/O 压力的百分比在特定应用环境中通常是总体稳定的,为了避免由策略参数设置不当引起 I/O 请求被长时间阻塞,有必要为实际响应的请求类型比率设置合理区间,以读请求响应比率 $g_1(x)$ 为参考项,每轮迭代结束时判断是否落在该范围内,设则可得到如下不等式约束:

$$P_1 < g_1(x) \leq P_2 \quad (5)$$

式中, P_1 为该范围的上限, P_2 为下线, $g_1(x)$ 以运行时实际测量值为准.

通用存储系统中,读写请求的发起时刻具有随机性,两种类型的 I/O 请求在时间维度上均匀分布,则写请求被读请求阻塞次数随时间的统计量 $g_2(x)$ 也应符合具体应用场景要求,可得如下不等式约束:

$$K_1 < g_2(\mathbf{x}) = \frac{ws}{\frac{s_s}{s_w}} \leq K_2 \quad (6)$$

式中, K_1 和 K_2 分别为上下限, s_w 为某一迭代轮回中处理写请求的数量, s_s 为某一迭代轮回中处理的 I/O 请求总数。

每次迭代中被分批处理的 I/O 请求批次长度在最佳状态下, 应当尽可能接近被阻塞的写请求过期时已处理的 I/O 请求总量, 这样可以减少因频繁地检查是否有写请求超时而未命中所带来的时间消耗, 则得到:

$$g_3(\mathbf{x}) = \left| \frac{t_c}{re \times fb + we} - fb \right| \leq L \quad (7)$$

式中, t_c 为迭代周期, L 为可接受的误差。

写请求到达的时间点是均匀分布的, 这种均匀分布在特定的存储系统环境中存在一个稳定的频率, 一旦策略参数调整过程中出现实际处理写请求的频率吻合该存储系统环境的写请求频率, 就要尽力维持这个参数向量。由于参数调整过程初始阶段的四个参数都是随机生成, 则对写请求频率的约束在初始阶段对参数约束的效果将会有明显的约束效果。由此可以得到以下等式约束:

$$h_1(\mathbf{x}) = E(\mathbf{x}) = \frac{f(\mathbf{x})}{0!} + \frac{f'(\mathbf{x})}{1!}(F_r - ws) + \frac{f''(\mathbf{x})}{2!}(F_r - ws)^2 + \dots + \frac{f^{(n)}(\mathbf{x})}{n!}(F_r - ws)^n + R_n(F_r) \quad (8)$$

式中, $E(\mathbf{x})$ 为 $(\mathbf{x})$ 的期望, $f(\mathbf{x})$ 为适应度函数, F_r 为运行时读请求频率测量值, 余项 $R_n(F_r)$ 可表示为

$$R_n(F_r) = f^{(n+1)}(ws + \theta(F_r - ws)) \cdot \frac{(1 - \theta)^n (F_r - ws)^{n+1}}{n!} \quad (9)$$

遗传算法的选择算子从上一代种群和通过交叉算子和变异算子产生的新个体中选出下一代个体的过程, 应当是随时间平稳演变的, 即使是变异算子产生的新个体, 如果对调度策略没有起到十分明显的性能提升, 那么也应归纳到前几代种群经过多轮演变形成的最优参数取值范围中, 避免对优质种群产生扰动。因此有必要针对每轮迭代产生的新个体做噪声约束, 可得如下等式约束:

$$h_2(\mathbf{x})_i = \text{cov}(\mathbf{x}_i, \mathbf{x}_{i+1}) \times (\mathbf{x}_{i+1} - \mathbf{x}_i) = C \quad (10)$$

$$\text{cov}(\mathbf{x}_i, \mathbf{x}_{i+1}) = \sqrt{\frac{\mathbf{x}_i^2}{\mathbf{x}_i^2 + \mathbf{x}_{i+1}^2}} \quad (11)$$

式中, $\mathbf{x}_i$ 代表第 i 轮遗传迭代, C 为常数,

策略参数本身根据其代表的意义不同, 还应当被约束在一个合理的取值范围当中。如 re 定义为每个读请求的超时期限, 存储系统的这个期限在大于 500 ms 时将对用户变得不可接收, 尤其是在重负载的存储系统中, 该期限会更小。因此有必要设置一个体现取值范围的等式约束:

$$h_3(\mathbf{x}_i) = \begin{cases} fb_i = fb_i \bmod M_1 \\ ws_i = ws_i \bmod M_2 \\ re_i = re_i \bmod M_3 \\ we_i = we_i \bmod M_4 \end{cases} \quad (12)$$

式中, M_1, M_2, M_3, M_4 分别为对应各个参数的最大取值范围。

本文中目标函数是求取个体适应情况的适应度函数, 即 Deadline I/O 调度策略在指定 $\mathbf{x}$ 参数的情况下, 存储系统的 I/O 性能体现。

I/O 性能主要体现在两个方面: 一方面, I/O 请求的处理效率, 相同时间内能够处理更多的 I/O 请求则表明调度策略可以提供更高的 I/O 请求处理效率; 另一方面, 相同时间内处理的指定 I/O 请求类型所占比率与实际发起的 I/O 请求类型比率吻合程度也体现了存储系统的 I/O 性能, 因为即使处理再多的 I/O 请求, 但这些请求可能是由众多请求少量数据的读请求组成的, 而实际上写请求可能已经被较长时间阻塞, 反之亦然。

依据上述原因, 可以设计如下适应度函数:

$$f(\mathbf{x}_i) = (1 - Q) \left((S_i)^2 - \left(\sum_{j=1}^{i-1} S_j \right)^2 \right)^2 + Q \left((T_i)^2 - \left(\sum_{j=1}^{i-1} T_j \right)^2 \right)^2 \quad (13)$$

式中, S_i 为运行环境中 I/O 请求处理总量的测量值, T_i 为运行环境中读请求处理数量占总 I/O 请求处理总量的比重, Q 为参考权重。依据式(2)加约束后, 得:

$$F(\mathbf{x}_i) = (1 - Q) \left((S_i)^2 - \left(\sum_{j=1}^{i-1} S_j \right)^2 \right)^2 + Q \left((T_i)^2 - \left(\sum_{j=1}^{i-1} T_j \right)^2 \right)^2 + \lambda \left(\sum_{i=1}^3 g_i(\mathbf{x}_i) + \sum_{i=1}^3 |h_i(\mathbf{x}_i)| \right) \quad (14)$$

式中, $F(x_i)$ 即为遗传算法在本文中选择算子的适应度函数, 适应度值越接近零, 说明策略参数向量越能使得调度算法发挥出更好的调度性能, 每个个体据此判断是否加入下一轮迭代.

3 实验结果与分析

本文实验方案流程如图 1 所示, 实验环境是一个提供开放数据服务的随机存储系统, 运行在该系统上的应用有较强的数据依赖性, 且存取数据的位置随机. 该存储系统的目的是为上层应用提供高性能数据 I/O, 其中单机 I/O 调度策略采用 Deadline 调度算法, 该算法的策略参数经过本文算法优化.

值设置如下:

$$\begin{cases} P_1 = 0.8, P_2 = 0.9; \\ K_1 = 0.8, K_2 = 1.0; \\ L = 0.2; \\ C = 0.2; \\ M_1 = 40, M_2 = 50; \\ M_3 = 500, M_4 = 5\,000; \\ Q = 0.3 \end{cases} \quad (15)$$

在此基础上, 设 F_{fb} 、 F_{ws} 、 F_{re} 、 F_{we} 分别为 fb、ws、re、we 的最终稳定值, F 为适应度, U 为 1 s 内处理的 I/O 请求总数, D 为 1 s 内处理的读请求数量占总 I/O 请求数量的比重. 采集 20 组实验数据, 实验结果如表 1 所示.

表 1 调度参数稳定后 I/O 性能测试结果

No.	F_{fb}	F_{ws}	F_{re}	F_{we}	F	U	$D/\%$
1	10	35	295	1 632	0.072 2	734	81
2	12	34	432	2 185	0.074 3	726	82
3	13	35	444	2 133	0.072 9	742	85
4	13	35	122	829	0.070 4	718	80
5	15	33	461	2 255	0.074 8	736	71
6	18	29	432	2 121	0.071 5	739	81
7	21	26	341	1 671	0.071 5	742	79
8	22	27	231	1 239	0.072 0	729	81
9	26	28	292	1 422	0.072 7	733	77
10	29	25	432	2 083	0.071 8	739	83
11	31	22	267	1 359	0.069 0	745	83
12	31	20	289	1 611	0.072 3	729	78
13	36	19	439	1 982	0.073 8	745	80
14	37	17	467	2 301	0.073 4	719	82
15	38	12	346	1 713	0.075 4	737	79
16	40	10	239	1 277	0.068 8	728	80
17	41	8	205	1 143	0.070 9	730	81
18	42	7	165	1 009	0.072 3	729	81
19	44	6	187	1 088	0.071 1	736	85
20	46	3	387	1 648	0.071 2	723	78

图 1 方案流程图

每轮种群中单个个体即为决策变量 x_i , 个体的四个参数应用到 Deadline 调度策略中, 时长为 1 s, 计算 $F(x_i)$ 得到该个体的适应度, 当种群中所有个体完成适应度计算时, 对个体排序, 去除适应度最差的 20%, 剩余个体进入下一轮迭代. 种群稳定的条件是种群中所有个体的适应度均相等, 这时就得到了 Deadline I/O 调度策略的最佳策略参数.

交叉算子通过从选择算子得到的按适应度有序的个体中相邻两个元素的参数取中值得到新个体, 加入到种群中, 去除相同个体.

变异算子从种群中随机挑选 5% 的个体, 将随机数与这些个体的四个参数相加, 经过 $h_3(x)$ 约束, 替换原有个体.

本文的应用环境中, 读请求与写请求发生的概率比例为 4: 1, 据此, 前文中提到的几个约束的关键

从表 1 中可以看出, F_{ws} 随 F_{fb} 的增大而增大, 说明处理 I/O 请求的批次长度越长, 则相同超期时限情况下, 写请求越容易超时, 产生写请求饥饿的可能性越大; F_{we} 随 F_{re} 的增大而增大, 说明读请求与写请求是相互制约的关系, 不能频繁处理一种请求, 却长时间忽略另一种请求; 每组实验结果的 I/O 请求处理总数都确定在 718 至 745 之间, 说明这个范围就是设备所能发挥的 I/O 性能极限; 同时, 可以看到处理的 I/O 请求类型比例也很好地回归到了 4: 1 上.

图 2 展示了第 7 组实验中从开始至迭代结束的过程中, 种群中个体适应度的抽样分布情况, 可以看出, 经过 7 次迭代后, 参数值趋于稳定. 由于每次迭

代都会有被去除的最差个体,且有变异算子对种群的扰动,所以多次迭代的个体之间关联性不确定,不适合画出折线图。

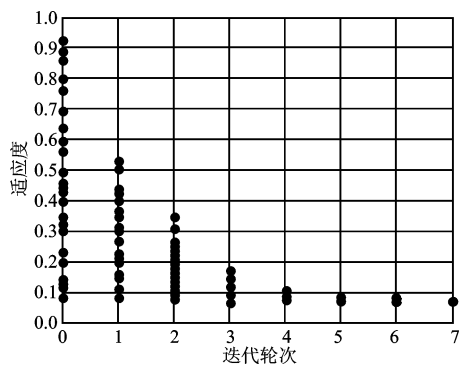


图2 单次实验迭代过程

可以看出,随着迭代过程的进行,参与迭代的个体的适应度逐渐收敛于0.071 5,符合优质参数筛选的预期。

图3展示了采用经过参数优化的Deadline I/O调度算法与采用原有四种I/O调度算法时存储系统I/O性能的对比。其中,已知读写请求发生概率比例为4: 1。可以看出,经过参数优化的Deadline I/O调度算法在相同时间内处理的I/O请求数量最多,且处理的请求类型更接近读写请求比例4: 1。

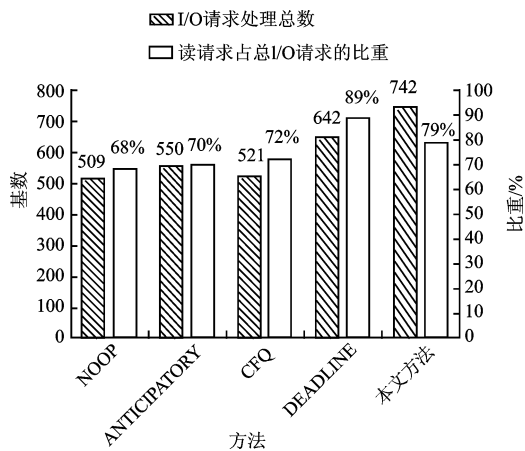


图3 不同调度策略的I/O性能对比

4 结束语

本文利用遗传算法的思想对Deadline I/O调度算法的参数做了优化,通过多轮迭代筛选,得出最优参数组合,使得I/O调度算法能够达到最优的性能。结合计算机存储系统的实际使用环境,分析了上层应用对存储系统I/O性能的通用需求,设置了多个约束条件,使得策略参数种群能够向优质方向进化,

最终得出稳定的优质策略参数。实验结果表明本文所采用的方法可以有效地提升存储系统I/O性能,对于I/O调度算法的参数,可以由多组随机参数进化出对特定运行环境中最适用的参数组合,使得存储系统发挥出最优I/O性能。

参考文献:

[1] Kim J, Seo S, Jung D, et al. Parameter-aware I/O management for solid state disk[J]. IEEE Trans on Computers, 2012, 61(5): 636-649.

[2] Park S, Shen K. FIOS: A fair, efficient flash i/o scheduler[C] // Proceedings of the 10 th USENIX Conference on File and Storage Technologies C. Berkeley, CA, USA: ACM, 2012: 13-13.

[3] 刘硕. 高清视频服务器磁盘I/O调度算法的研究[D]. 厦门: 厦门大学, 2014.

[4] Dunn M, Reddy A L N. A new I/O scheduler for solid state devices. Technical Report TAMU-ECE-2009-02 of Electrical and Computer Engineering[R]. 2009.

[5] Schindler J, Shete S, Smith K A. Improving throughput for small disk requests with proximal i/o[C] // Proceedings of the 9 th USENIX Conference on File and Storage Technologies(FAST' 2011) San Jose, CA. IEEE, 2011: 133-147.

[6] Zhang X, Davis K, Jiang S. Itransformer: Using SSD to improve disk scheduling for high-performance I/O[C] // Proceedings of the 26 th IEEE International Parallel and Distributed Processing Symposium Shanghai, China: IEEE, 2012: 715-726.

[7] Murat A, Novruz A. Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms[J]. Expert System with Applications, 2011, 38(3): 1313-1320.

[8] LIFa-chao, XU Li-da, JIN Chen-xia, et al. Intelligent bionic genetic algorithm (IB-GA) and its convergence[J]. Expert System with Applications, 2011, 38(7): 8804-8811.

[9] 梁昔明, 朱灿, 颜东煌. 基于物种选择的遗传算法求解约束非线性规划问题[J]. 中南大学学报: 自然科学版, 2009, 40(1): 185-189.

作者简介:

吕燕彬 男, (1990-), 硕士研究生. 研究方向为海量视频数据的存储和检索. E-mail: lyb_lizard@163.com.

秦品乐 男, (1978-), 副教授, 硕士生导师. 研究方向为机器视觉、三维重建等.