一种公平的动态轮转算法

胡 赛[1],赵碧海[1,2],熊慧军[1*]

(1. 长沙学院信息与计算科学系,中国 长沙 410003;2. 中南大学信息科学与工程学院,中国 长沙 410083)

摘 要 时间片轮转算法作为一种经典的调度算法得到了广泛的应用. 针对时间片轮转算法的调度策略和时间片长度的选取等问题开展深入的研究,提出了一种改进的动态轮转算法,算法是短作业优先算法、多级队列算法和时间片轮转算法的综合和发展. 利用生灭过程理论建立了时间片轮转算法和动态轮转算法的性能模型,分析了两种算法的平均等待时间和平均周转时间,引入性能提高百分比的概念对比两种算法的差异. 实验结果和理论分析均表明改进算法的性能优于传统的时间片轮转算法.

关键词 时间片轮转;短作业优先;动态时间片;性能

中图分类号 TP311 **文献标识码** A **文章编号** 1000-2537(2012)05-0030-07

A Fair Dynamic Quantum Algorithm

HU Sai[1], *ZHAO Bi-hai* [1,2], *XIONG Hui-jun*[1*]

(1. Department of Information and Computing Science, Changsha University, Changsha 410003, China;
2. School of Information Science and Engineering, Central South University, Changsha 410083, China)

Abstract As a classic scheduling algorithm, round robin algorithm has been widely applied. According to the scheduling policy and the selection of the length of time slice for the round robin algorithm, a large number of in-depth studies are carried out. An improved dynamic quantum algorithm is proposed. The algorithm is the development and integration of shortest job first algorithm, multi-level queue scheduling algorithm and round robin algorithm. The performance models of the round robin algorithm and the dynamic quantum algorithm are built by using the theory of birth-death process, including average waiting time and average turnaround time of the two algorithms. The concept of performance improvement percentage is introduced in order to show the differences between the two algorithms. Both theoretical analysis and experimental results indicate that the performance of the improved algorithm is better than the traditional algorithm. The advantages of the improved algorithm become more obviously when the number of tasks increase and the rate of service improves.

Key words round robin; shortest job first; dynamic quantum; performance

---

现代操作系统已经从原来的单任务模式演化成多任务并发模式. CPU 作为一种主要的计算资源,被多个任务同时竞争使用. 为任务分配 CPU 时,调度程序应尽量保证对所有任务公平,同时获得最小的周转时间、响应时间和上下文切换次数. 时间片轮转算法作为一种最经典的调度算法得到了广泛的应用,尤其在分时操作系统中. 国内外许多学者对此展开了大量的研究,并提出了一些改进的算法. 为了获得更小的等待时

---

* 收稿日期:2010-12-23
基金项目:国家自然科学基金资助项目(60473117);国家科技支撑计划资助项目(2007BAH14B01);湖南省教育厅基金资助项目(11C0125);湖南省十二五规划课题基金资助项目(XJK011CXJ002);长沙市科技基金资助项目(K110718-11)

* 通讯作者,E-mail:bihaizhao@163.com

间,文献[1~4]修改了时间片的设置方式,时间片长度由任务的服务时间决定,文献[5]修改了轮转算法在同一周期内的调度策略;为了追求更好的公平性,文献[6,7]提出以任务的到达时间和服务时间作为优先调度的依据. 文献[8~9]研究了实时系统和嵌入式系统环境的轮转算法. Shukla 使用马尔科夫链模型分析了 FCFS 服务算法、时间片轮转算法和多级队列算法发生状态转移时的概率关系[10-11],Pandey[12]教授分析了任务的服务时间与每次轮转完成的任务数量之间的关系,提出了一种两级队列的时间片轮转算法. 上述这些算法只注重提高性能的某一方面而忽略了其他方面,如为了追求更好的等待时间而牺牲算法的公平性;缺乏严格的数学分析,模拟仿真不具有普遍性. 本文通过综合时间片轮转算法、短作业优先算法和多级队列算法,提出一种公平的动态轮转算法.

1　算法设计

1.1　定义及相关符号说明

为便于描述,首先给出论文中涉及的一些定义和符号说明. λ: 进程进入等待队列的到达率,μ: CPU 为进程提供服务的服务率,ρ: 服务强度,即 $\rho=\frac{\lambda}{\mu}$,L: 平均队长,L_q: 平均等待的进程数量.

定义 1　性能提高比 $R_{\text{performance\_improve}}=\frac{W_{\text{RR}}-W_{\text{DRR}}}{W_{\text{DRR}}}\times 100\%$,$W_{\text{RR}}$指传统的时间片轮转算法的平均周转时间,$W_{\text{DRR}}$指改进的动态时间片轮转算法的平均周转时间.

定义 2　若进程调度过程具有下列性质:

(1) 进程到达等待队列的数量服从参数为 λ 的泊松分布,时间间隔服从参数为 λ 的指数分布;

(2) 进程服务时间服从参数为 μ 的指数分布;

(3) 在一个无限短的时间间隔内只有一个进程到达或离开.

则称之为一个生灭过程.

1.2　动态轮转算法

本文提出的动态轮转算法(DRR)综合了短作业优先算法、时间片轮转算法(RR)和多级队列调度算法. 传统时间片轮转算法中每个进程获得的时间片相同. 从公平的角度来说,运行时间长的进程希望得到更多的时间片,因为它们等待的时间更长. 多级队列调度算法解决了这一问题,比时间片轮转算法具有更好的公平性. 但是这种算法又带来了另一种不公平. 由于算法采用多个队列,调度程序从当前队列中调度进程前会检查优先级高于当前队列的等待队列中是否有进程,若有进程在等待,则会优先调度优先级较高的队列中的进程,直到所有优先级较高的队列中都为空时才调度当前队列中的进程. 由于有新进程不断进入,那些沦入优先级低的队列中的进程虽然得到了较长的时间片,却一直得不到被调度的机会,甚至会出现某些进程长时间在队列中休眠.

针对时间片长度的选取,动态轮转算法综合了时间片轮转算法和多级队列算法的优势. 处于就绪状态的进程在等待队列中等候调度,每个进程被调度时所获得的时间片长度各不相同: 进程首轮调度时,时间片长度为固定值 a,从第二轮开始,进程所得到的时间片长度与该进程已经获得的 CPU 服务时间成正比. 用 T_i 表示进程第 i 轮调度时得到的时间片长度,则

$$T_i=\begin{cases}a, & i=1,\\ ab(1+b)^{i-2}, & i>1.\end{cases}$$

进程控制块(PCB)包含有进程的各种计时信息,进程的调度次数和已经获得的 CPU 的时间,进程被调度时,从 PCB 中获取调度次数和执行时间,并以此计算出本次调度可获得的时间片长度,调度结束后对 PCB 中的该部分信息进行更新.

华金先生曾指出,在允许优先占用时,对需要服务时间短的顾客给与高优先权的方式是减少平均等待时间的好办法. 基于这一思想,在调度策略上,本文采取短作业优先算法和时间片轮转算法相结合,即在同一个轮转调度周期内将原有的先来先服务的调度方式改为短进程优先的方式,并在算法中引入了服务队列的概念. 服务队列是指等待队列中的进程按照所需服务时间从短到长的顺序排列而形成的进程序列,进程调度程

序从服务队列中调度进程并获取 CPU 的服务.

以下是算法的具体步骤:

算法 DRR(P, a, b)

输入: 随机产生的进程集 $P\{p_1, p_2, \cdots, p_n\}$,初始时间片长度 a,时间片增长系数 b;

输出: 平均周转时间 W,平均响应时间 W_r;

1. $W = 0$; $W_r = 0$; RemainProcess = Length(P);
2. While RemainProcess > 0 do
   2.1 for $i = 0$ to Length(P) − 1 do
   2.2 if $P[i]$ 是刚抵达等待队列的新进程
   2.3 根据 $P[i]$ 的类型及资源需求预测 $P[i]$ 的服务时间
   2.4 $P[i]$ 进入服务队列;
   2.5 NewCount = NewCount + 1;
   2.6 if NewCount > 0 then
   2.7 j = 新进程在队列中的位置
   2.8 if (NewCount > 0) and (j < Top) then
   2.9 CurrentProc = $P[j]$
   2.10 NewCount = NewCount − 1;
   2.11 Else CurrentProc = P[Top];
   2.12 从 CurrentProc 的 PCB 中读取调度次数 SchedulTimes 和运行时间 RunTime;
   2.13 if SchedulTimes = 0 then
   2.14 $T = a$;
   2.15 Else
   2.16 $T = b$* RunTime;
   2.17 进程 CurrentProc 运行时间片 T;
   2.18 更新 CurrentProcPCB 中的计时信息
   2.19 RemainProcess = RemainProcess − 1;
3. for $i = 0$ to Length(P) − 1 do
   3.1 $W = W + P[i]$ 的周转时间;
   3.2 $W_r = W_r + P[i]$ 的响应时间;
4. $W = W$/ Length(P); $W_r = W_r$/ Length(P);
5. Output(W, W_r);

2 算法分析

假定进程来源是无限的,进程到达时间与系统当前状态和以前的进程到达时间无关,服务时间与系统当前的状态及以前进程获得的时间无关.

当上述假设成立,进程到达等待队列的数量服从参数为 λ 的泊松分布,λ 为到达率,CPU 为进程提供服务的进程个数服从参数为 μ 的泊松分布,μ 为服务率.

上述定义中,“生”指的是进程的到达,“灭”指的是进程的离开. 生灭过程的瞬时状态一般很难求得,但可以求得稳定状态分布. 对于平稳的生灭状态而言,从平均意义上讲,到达等于离开. 图 1 是稳定的生灭过程的状态转移图.

2.1 时间片轮转算法分析

根据分析可知,采用时间片轮转算法时,进程到达等待队列的数量服从参数为 λ 的泊松分布,时间间隔服从参数为 λ 的指数分布,服务时间服从参数为 μ 的指数分布. 由于提供服务的 CPU 的数量为一个,在无限短的时间内,只有一个进程到达或离开. 由定义 7 可知,满足生灭过程的性质. 下面将利用生灭过程的状态转

移图分析时间片轮转算法的平均等待时间和平均周转时间.

以 $p_n(t)$ 表示时刻 t 时系统状态 $X(t) = n$ 的瞬时概率,p_n 表示对应的稳定概率. $p_n = \lim_{t\to\infty} p_n(t)$.

生灭过程的基本原理是,在任意状态下 n 达到稳态平衡的条件: 产生该状态的平均速率 = 该状态转变成其他状态的平均速率,即到达等于离开.

根据上述基本原理,可得下列平衡方程.

状态	离去	进入
0	$\lambda_0 p_0$	$= \mu_1 p_1$
1	$(\lambda_1 + \mu_1) p_1$	$= \lambda_0 p_0 + \mu_2 p_2$
⋮		⋮
$n-1$	$(\lambda_{n-1} + \mu_{n-1}) p_{n-1}$	$= \lambda_{n-2} p_{n-2} + \mu_n p_n$
n	$(\lambda_n + \mu_n) p_n$	$= \lambda_{n-1} p_{n-1} + \mu_{n+1} p_{n+1}$

整理得: $p_1 = \frac{\lambda_0}{\mu_1} p_0, p_2 = \frac{\lambda_1\lambda_0}{\mu_2\mu_1} p_0, \cdots, p_n = \frac{\lambda_{n-1}\lambda_{n-2}\cdots\lambda_1\lambda_0}{\mu_n\mu_{n-1}\cdots\mu_2\mu_1} p_0$. 从定理 2 可知,不同的时间间隔内随机变量独立且同分布. 所以 $\begin{cases}\lambda_n = \lambda, n \geqslant 0\\ \mu_n = \mu, n \geqslant 1\end{cases}$,则有 $p_n = \frac{\lambda_{n-1}\lambda_{n-2}\cdots\lambda_1\lambda_0}{\mu_n\mu_{n-1}\cdots\mu_2\mu_1} p_0 = \left(\frac{\lambda}{\mu}\right)^n p_0 = \rho^n p_0$. 当达到稳定状态时,$\rho = \frac{\lambda}{\mu} < 1$,由于 $\sum_{n=0}^{\infty} P_n = 1$,则有 $\sum_{n=0}^{\infty} \rho^n p_0 = p_0 \frac{1+\rho^n}{1-\rho} = 1$. 由于 $\sum_{n=0}^{\infty} \rho^n = \frac{1+\rho^n}{1-\rho}, \rho < 1, n \to \infty, \rho^n \to 0$,所以 $p_0 = 1-\rho$, $p_n = \rho^n p_0 = (1-\rho)\rho^n$.

随机变量 X 表示系统处于稳态时的进程数,采用时间片轮转算法时的平均队长 L_{RR} 是 X 的数学期望. 所以,

$$L_{RR} = E(X) = \sum_{n=0}^{\infty} n p_n = \sum_{n=0}^{\infty} n(1-\rho)\rho^n = (1-\rho)\rho \sum_{n=1}^{\infty} n\rho^{n-1} = (1-\rho)\rho \sum_{n=1}^{\infty} (\rho^n)' =$$

$$(1-\rho)\rho\left(\sum_{n=0}^{\infty} \rho^n\right)' = (1-\rho)\rho\left(\frac{1}{1-\rho}\right)' = \frac{\rho}{1-\rho}.$$

随机变量 X_q 表示系统处于稳态时排队等待的进程数,采用 RR 算法时平均等待的进程数 L_{q-RR} 可表示为

$$L_{q-RR} = E(X_q) = \sum_{n=1}^{\infty} (n-1) p_n = \sum_{n=1}^{\infty} n p_n - \sum_{n=1}^{\infty} p_n = L_{RR} - (1-p_0) = \frac{\rho}{1-\rho} - \rho = \frac{\rho^2}{1-\rho}.$$

由 Little 公式可知,RR 算法的平均周转时间 $W_{RR} = \frac{L_{RR}}{\lambda} = \frac{\rho}{(1-\rho)\lambda} = \frac{\lambda/\mu}{(1-\lambda/\mu)\lambda} = \frac{1}{\mu-\lambda}$. 平均等待时间

$$W_{q-RR} = \frac{L_{q-RR}}{\lambda} = \frac{\rho^2}{(1-\rho)\lambda} = \frac{\lambda}{\mu(\mu-\lambda)}.$$

2.2　改进的动态轮转算法分析

定理 3　若有 $n(n>0)$ 个进程 $p_1, p_2, \cdots, p_n$,服务时间相互独立且满足参数为 μ 的指数分布,则从中选取服务时间最短的进程优先执行,其服务时间满足参数为 $n\mu$ 的指数分布.

证　由于进程 $p_1, p_2, \cdots, p_n$ 的服务时间相互独立,且均满足参数为 μ 的指数分布,因此不妨用随机变量 $X_1, X_2, \cdots, X_n$ 分别表示它们的服务时间分布. 则根据指数分布的定义,$X_1, X_2, \cdots, X_n$ 的概率密度可分别表示为 $f_1(x) = \mu e^{-\mu x}, f_2(x) = \mu e^{-\mu x}, \cdots, f_n(x) = \mu e^{-\mu x}, x > 0$,分布函数可分别表示为 $F_1(x) = 1 - e^{-\mu x}, F_2(x) = 1 - e^{-\mu x}, \cdots, F_n(x) = 1 - e^{-\mu x}, x > 0$.

若用随机变量 Z 表示从 n 个进程中取服务时间最短的进程的服务时间的分布,则有 $Z = \min(X_1, X_2, \cdots, X_n)$,$Z$ 的分布函数可表示为 $F_{\min} = 1 - (1 - F_1(x))(1 - F_2(x)) \cdots (1 - F_n(x))$. 由于 $X_1, X_2, \cdots, X_n$ 相互独立且具有相同分布,所以 $F_{\min}(z) = 1 - (1 - F_1(z))^n = 1 - (1 - (1 - e^{-\mu z}))^n = 1 - e^{-n\mu z} (z > 0)$. 于是 $Z = \min(X_1, X_2, \cdots, X_n)$ 的概率密度为 $f_{\min}(z) = n\mu e^{-n\mu z} (z > 0)$,满足参数为 $n\mu$ 的指数分布.

本文提出的基于短作业优先的动态轮转算法(DRR) 是对 RR 算法的服务方式的改进,进程的到达方式并未改变. 因此采用 DRR 算法时,进程到达等待队列的数量仍然服从参数为 λ 的泊松分布,到达的时间间隔服从参数为 λ 的指数分布. 关于服务方式,DRR 算法不再采用原有的先来先服务方式,而是每次从服务队列中选择所需服务时间最短的一个进程调度执行. 若队列中有 n 个进程,根据定理 3,从 n 个进程中选择服务时

间最短的进程执行,服务时间满足参数为 $n\mu$ 的指数分布,所以采用 DRR 算法时,仍满足生灭过程的定义,故本文将借助生灭过程中处于稳定状态时的状态转移图分析 DRR 算法的平均周转时间和平均等待时间.

当系统处于稳态时,DRR 算法将从 λ 个进程中选取服务时间最短的进程执行. 若用 μ' 表示 DRR 算法的服务率,则根据定理 3,$\mu' = \lambda\mu$.

用 $\mu'_1,\mu'_2,\cdots,\mu'_n$ 分别表示状态为 $1,2,\cdots,n$ 时的服务率. 从图 1 状态转移图中不难得出

$$p_1 = \frac{\lambda_0}{\mu'_1}p_0, p_2 = \frac{\lambda_1\lambda_0}{\mu'_2\mu'_1}p_0, \cdots, p_n = \frac{\lambda_{n-1}\lambda_{n-2}\cdots\lambda_1\lambda_0}{\mu'_n\mu'_{n-1}\cdots\mu'_2\mu'_1}p_0.$$

由于进程到达数量服从参数为 λ 的泊松分布,因此单位时间到达的平均数量为其分布的数学期望,$E(X) = \sum_{x=0}^{\infty} xP(x) = \sum_{x=0}^{\infty} x\frac{\lambda^x}{x!}e^{-\lambda} = \lambda e^{-\lambda}\sum_{x=1}^{\infty}\frac{\lambda^x}{x!} = \lambda$. 同理,单位时间内被服务的进程平均数量为 μ. 因此,当系统处于状态 1 时,DRR 算法从 $(\lambda_0 + \mu_2)$ 个进程中选取服务时间最短的进程执行. 根据定理 3 有 $\mu'_1 = (\lambda_0 + \mu_2)\mu_1$. 同理可得 $\mu'_2 = (\lambda_1 + \mu_3)\mu_2, \cdots, \mu'_n = (\lambda_{n-1} + \mu_{n+1})\mu_n$. 因为 $\lambda_n = \lambda, n \geqslant 0, \mu_n = \mu, n \geqslant 1$,所以 $p_n = \left(\frac{\lambda}{\mu(\mu+\lambda)}\right)^n p_0$. 则有

$$p_0 = 1 - \frac{\lambda}{\mu(\mu+\lambda)} = \frac{\mu^2 + \mu\lambda - \lambda}{\mu(\mu+\lambda)}, p_n = \frac{\mu^2 + \mu\lambda - \lambda}{\mu(\mu+\lambda)}\left(\frac{\lambda}{\mu(\mu+\lambda)}\right)^n = \frac{(\mu^2 + \mu\lambda - \lambda)\lambda^n}{\mu^{n+1}(\mu+\lambda)^{n+1}}.$$

当系统处于稳态时,采用 DRR 算法时的平均队长 L_{DRR} 可表示为

$$L_{\mathrm{DRR}} = \sum_{n=0}^{\infty} np_n = \sum_{n=0}^{\infty} n\frac{(\mu^2 + \mu\lambda - \lambda)\lambda^n}{\mu^{n+1}(\mu+\lambda)^{n+1}} = \sum_{n=0}^{\infty} n\left(\frac{\lambda^n}{(\mu(\mu+\lambda))^n} - \frac{\lambda^{n+1}}{(\mu(\mu+\lambda))^{n+1}}\right) = \frac{\lambda}{(\mu+\lambda)\mu - \lambda}.$$

平均等待的进程数 $L_{q-\mathrm{DRR}}$ 可表示为

$$L_{q-\mathrm{DRR}} = \sum_{n=1}^{\infty}(n-1)p_n = \sum_{n=1}^{\infty} np_n - \sum_{n=1}^{\infty} p_n = L_{\mathrm{DRR}} - (1 - p_0) =$$

$$\frac{\lambda}{(\mu+\lambda)\mu - \lambda} - \left(1 - \frac{\mu^2 + \mu\lambda - \lambda}{\mu(\mu+\lambda)}\right) = \frac{\lambda^2}{(\mu(\mu+\lambda))(\mu(\mu+\lambda) - \lambda)}.$$

因此采用 DRR 算法时,由 Little 公式可知,平均周转时间 $W_{\mathrm{DRR}} = \frac{L_{\mathrm{DRR}}}{\lambda} = \frac{1}{(\mu+\lambda)\mu - \lambda}$,平均等待时间

$$W_{q-\mathrm{DRR}} = \frac{L_{q-\mathrm{DRR}}}{\lambda} = \frac{\lambda}{(\mu(\mu+\lambda))^2 - \lambda\mu(\mu+\lambda)}.$$

根据分析可知,

$$W_{\mathrm{RR}} - W_{\mathrm{DRR}} = \frac{1}{\mu - \lambda} - \frac{1}{(\mu+\lambda)\mu - \lambda} = \frac{\mu(\lambda - 1) + \mu^2}{(\mu - \lambda)((\mu+\lambda)\mu - \lambda)},$$

$$W_{q-\mathrm{RR}} - W_{q-\mathrm{DRR}} = \frac{\lambda}{\mu(\mu - \lambda)} - \frac{\lambda}{(\mu(\mu+\lambda))(\mu(\mu+\lambda) - \lambda)} = \frac{\lambda}{\mu}\left(\frac{1}{\mu - \lambda} - \frac{1}{(\mu+\lambda)(\mu(\mu+\lambda) - \lambda)}\right).$$

当系统处于稳态时有,$\mu > \lambda > 1$,$W_{\mathrm{RR}} - W_{\mathrm{DRR}} > 0$,

$$(\mu+\lambda)(\mu(\mu+\lambda) - \lambda) - (\mu - \lambda) > (\mu+\lambda)(\mu(\mu+\lambda) - \lambda) - (\mu+\lambda) = (\mu+\lambda)(\mu^2 - 1 + \lambda(\mu - 1)) > 0.$$

即 $\frac{1}{\mu - \lambda} - \frac{1}{(\mu+\lambda)(\mu(\mu+\lambda) - \lambda)} > 0$,所以,$W_{q-\mathrm{RR}} - W_{q-\mathrm{DRR}} > 0$.

由此可见,相比 RR 算法,采用 DRR 算法之后,进程的平均等待时间和平均响应时间都有所降低,这表明 DRR 算法的性能要优于 RR 算法. 接下来本文将进一步分析 2 种算法之间的差异.

根据定义 3 可知,$R_{\mathrm{performance\_improve}} = \frac{W_{RR} - W_{\mathrm{DRR}}}{W_{\mathrm{DRR}}} = \frac{1}{\mu - \lambda} - \frac{1}{(\mu+\lambda)\mu - \lambda}((\mu+\lambda)\mu - \lambda) = \frac{\mu^2 + \lambda\mu - \mu}{\mu - \lambda}$.

当 λ 递增时,$R_{\mathrm{performance\_improve}}$ 递增,这说明随着 λ 的增大,DRR 算法的性能优势越明显. 令 $f(\mu) = \mu^2 + \lambda\mu - \mu$,$g(\mu) = \mu - \lambda$,则 $f'(\mu) = 2\mu + \lambda - 1$,$g'(\mu) = 1$. 由于 $f'(\mu) > g'(\mu) > 0$,所以当 μ 递增时,$R_{\mathrm{performance\_improve}}$ 递增,即服务率越高,DRR 算法的性能优势越大.

衡量调度算法的指标除了平均周转时间和平均响应时间外,系统开销也是非常重要的,DRR 算法时间片动态增长,进程每次调度都能获得比上次调度更长的时间片长度,能有效地减少进程调度的次数. 系统在

发生进程切换时,需要保留 CPU 的现场信息,同时引发进程上下文的切换,增大了系统开销,因此通过时间片增长缩减调度次数的方式,能有效地降低系统开销.

综上所述,改进的动态轮转算法的综合性能优于传统的时间片轮转算法.

3　模拟实验和结果分析

本节将利用模拟实验结果定量分析 2 种算法之间的性能差异. 为了更加客观地描述两者间的差异,引入性能提高比 $=\dfrac{\text{RR 性能值}-\text{DRR 性能值}}{\text{RR 性能值}}\times 100\%$.

以下分别用 improve_turn,improve_wait,improve_response,improve_switch 表示平均周转时间提高比,平均等待时间提高比,平均响应时间提高和平均切换次数提高比.

本文将传统的 RR 算法和自适应的 DRR 算法分别模拟调度实现. 本次实验基于 Windows 平台,采用 Matlab 程序模拟实现. 模拟系统包括任务生成模块和调度模块两部分. 任务生成模块负责随机生成任务,任务到达时间和估计服务时间,由系统在设定的区间内随机生成. 任务数量和服务时间变化区间大小是可调大小参数,以验证任务数量和服务时间对算法性能的影响; 调度模块分别采用 RR 算法和 DRR 算法调度任务生成模块产生的任务.

本次实验应达到如下 3 个目的: 1) 在模拟调度环境下,对比 RR 算法和 DRR 算法的综合性能差异. 2) 研究不同的任务数目和服务时间跨度对 2 种算法性能的影响. 3) 研究 RR 算法时间片长度对两种算法性能差异的影响.

为此,本文设置了 4 组实验.

实验 1　RR 算法与 DRR 算法综合性能比较

本次实验生成 50 个任务,预期服务时间在区间 [1,1 000] 内随机产生,RR 算法的时间片 $q=50$,DRR 算法的增长速率调节参数 $b=2$.

实验结果如图 2 所示, X 轴表示实验次数,Y 轴表示性能提高比. 从图 2 可以看出,平均周转时间和平均等待时间缩短了 20% 左右,平均响应时间缩短了 98%,平均切换次数减少了 17% ~35%. 从实验结果可知,DRR 算法的综合性能优于传统的 RR 算法.

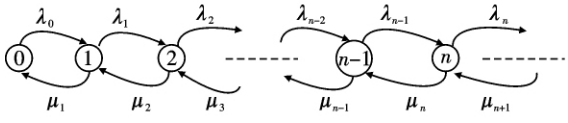

图 1　状态转移图

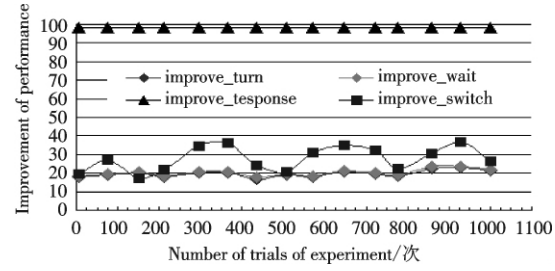

图 2　RR 算法与 DRR 算法的综合性能比较

实验 2　任务数目对算法性能的影响

本次实验任务数目设定在区间 [2,1 000] 内,任务到达时间随机产生,预期服务时间在区间 [1,2 000] 内随机产生,RR 算法的时间片 $q=50$,DRR 算法的增长速率调节参数 $b=2$.

实验结果如图 3 所示,其中 X 轴表示任务数目,Y 轴表示性能提高比. 由图 3 可以看出,平均周转时间缩短了 10% ~30%,平均等待时间缩短了 19% ~30%. 随着任务数目的增加,平均周转时间提高比和平均等待时间提高比缓慢提高,平均切换次数减少了 50% ~70%,随着任务数目的增多,平均切换次数提高比有所降低. 由此可见,任务数目越多,DRR 算法的综合性能优势逐步增大.

实验 3　服务时间跨度对算法性能的影响

本次实验生成 50 个任务,任务到达时间随机产生,任务预期服务时间跨度为 1 000 ~25 000,RR 算法的时间片 $q=50$,DRR 算法的增长速率调节参数 $b=2$.

实验结果如图 4 所示,其中 X 轴表示任务预期服务时间,Y 轴表示性能提高比. 由图 4 可以看出,随着预期服务时间跨度增大,平均切换次数提高比不断增长,而平均周转时间和平均等待时间提高比呈波浪式缓慢增长.

当服务时间跨度在 $b(1+b)^i(i>0)$ ms 左右时,性能提高比位于波峰;而当服务时间跨度在 $b(1+b)^i \sim b(1+b)^{i+1}$ ms 之间,性能提高比位于波谷. 由此可见,任务服务时间跨度增大,DRR 算法的综合性能优势缓慢增长.

综上所述,在模拟调度环境中,DRR 算法的综合性能优于 RR 算法. 当任务数目增多,服务时间跨度增大时,DRR 算法的性能更优, RR 算法的时间片较小时,DRR 算法的平均切换次数更优, RR 算法的时间片增大时,DRR 算法的平均响应时间优势增大.

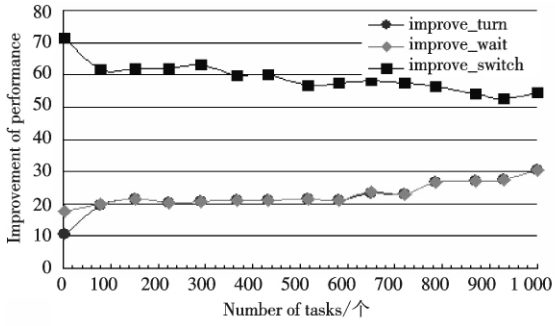

图3 任务数目对算法性能的影响

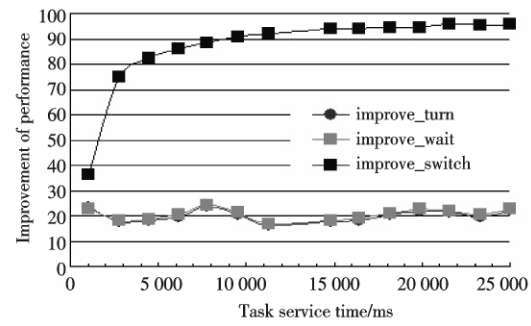

图4 任务服务时间跨度对算法性能的影响

4 结论及进一步研究工作

本文提出一种改进的动态轮转算法,算法将同一轮转周期内先来先服务方式改进为短作业优先方式,有效地缩短了进程的平均周转时间;将时间片由原来的定长方式修改为根据累计运行时间而自适应地动态增长方式,解决了原有的轮转方式中存在的时间片长度选取的矛盾,同时也提高了进程的响应时间,缩短了平均周转时间和等待时间.

本文的分析是以进程服务时间服从指数分布为前提. 下一步工作中,我们将分析服务时间服从一般分布时算法如何有效工作等问题.

参考文献:

[1] BEHEAR H S, MOHANTY R, DEBASHREE N. A new proposed dynamic quantum with re-adjusted round robin scheduling algorithm and its performance analysis [J]. Inter J Comput Appl, 2010,5(5):10-15.

[2] RAMI J M. Self-adjustment time quantum in round robin algorithm depending on burst time of the now running processes [J]. Am J Appl Sci, 2009,6(10):1831-1837.

[3] SAAD Z R, SAFWAT H H, SAMIH M. Improving waiting time of tasks scheduled under preemptive round robin using changeable time quantum [EB/OL]. 2010, http://arxiv.org/abs/1003.5342.

[4] RAKESH K Y, AHHISHEK K M, NAVIN P. An improved round robin scheduling algorithm for CPU scheduling [J]. Inter J Comput Sci Eng, 2010,2(4):1064-1066.

[5] MAMUNUR R, NASIM A. A new multilevel CPU scheduling algorithm [J]. J Appl Sci, 2006,6(9):2036-2039.

[6] HELMY T, DEKDOUK A. Burst round robin as a proportional-share scheduling algorithm [C]//IEEE-GCC, Urumchi, Xinjiang, China, 16-18 Aug, 2007. Wiley: IEEE Computer Society Press, 2007:424-428.

[7] MOHAMMED A. Best-job-first CPU scheduling algorithm [J]. Infor Tech J, 2007,6(2):288-293.

[8] YAASHUWANTH C. A new scheduling algorithms for real time tasks [J]. Inter J Comput Sci Infor Security, 2009,6(2):61-66.

[9] PANDURANGA R M, SHET K,ROOPA K. A simplified study of scheduler for real time and embedded system domain [J]. Inter J Comput Cog, 2009,5(22):60-73.

[10] SHUKLA D, OJHA S, JAIN S. Data model approach and Markov chain based analysis of multi-level queue scheduling [J]. J Appl Comput Sci Math, 2010,8(4):50-56.

[11] SHUKLA D, JAIN S, SINGHAI R, *et al*. A Markov chain model for the analysis of round-robin scheduling scheme [J]. J Adv Network Appl, 2009,1(1):1-7.

[12] PANDEY D, VANDANA. Improved round robin policy a mathematical approach [J]. Inter J Comput Sci Eng, 2010,2(4):948-954.

(编辑 沈小玲)