维普资讯 http://www.cqvip.com

基于优先级的抢占式并行调度算法设计与分析

张国斌　潘金贵

（南京大学计算机软件新技术国家重点实验室　南京 210093）

摘　要　并行作业调度系统负责对高性能计算系统中作业队列的管理。其核心功能是在每次调度发生时，选择下一个被执行的作业。最简单的调度算法是先来先服务(FCFS)。但这种方法的缺点是资源利用率很低。解决这个问题，目前常用的算法有 EASY Backfilling。但 EASY 算法也存在两个缺陷：要求用户估计作业运行时间和偏爱小作业。针对这两个问题，本文设计了一种新的调度方法：基于优先级的抢占式并行调度(Priority-based Preemptive Scheduling)，并实现了两种算法的模拟系统，从性能和公平性两个角度对 PPS 算法和 EASY 算法进行了比较分析，表明了 PPS 算法的有效性。

关键词　并行调度，基于优先级的抢占式调度，回填调度，公平性

Design and Analysis of Prority-based Preemtiv Parallel Scheduling Algorithm

ZHANG Guo-Bin　PAN Jin-Gui

(State Key Lab. for Novel Software, Nanjing University, Nanjing 210093)

Abstract　Parallel job scheduler is important to High Performance Computing system. The kernel function is to choose a job to execute whenever a scheduling event happens. The simplest policy is FCFS. This approach suffers from low system utilization. EASY Backfilling[4] was proposed to improve system utilization and has been implemented in several production schedulers. The main problem with EASY is that it requires estimates of job runtimes to be available. Another problem is that it moves forwards as many smaller jobs as possible. It is unfair to the larger ones. To attack these problems, we propose a new method, Priority-based Preemptive Scheduling.

Keywords　Parallel job scheduling, Priority-based preemptive scheduling, EASY backfilling, Fairness

1　引言

并行作业调度系统负责对高性能计算系统中作业队列的管理。其核心功能是在每次调度发生时，选择下一个被执行的作业。最简单的调度算法是先来先服务(FCFS)。但这种方法的缺点是资源利用率很低。为了解决这个问题，引入了 EASY Backfilling[4] 算法，是目前使用最广泛的算法。但 EASY 算法也存在两个缺陷：要求用户估计作业运行时间和偏爱小作业。为了解决这两个问题，本文设计了一种新的调度方法：基于优先级的抢占式并行调度(Priority-based Preemptive Scheduling)。该方法摒弃了要求用户估计作业执行时间的限制，大幅度提高了服务的公平性水平，并进一步优化了系统性能。本文实现了两种算法的模拟系统，从性能和公平性两个角度对 PPS 算法和 EASY 算法进行了比较分析。

2　EASY Backfilling 算法

FCFS 算法存在的问题：如第一个作业[1] 不能运行，则等待队列中其它作业都必须等待，造成大量资源闲置浪费。为了解决这个问题，引入了 EASY Backfilling 算法。

EASY 算法放松了这个限制：如果第一个作业现在不能运行，则系统为其预约资源和开始运行时间(Shadow Time)。只要等待队列中其他作业不影响第一个作业的按期运行，就可以打破 FCFS 的运行顺序，后来的作业可以先于第一个作业运行，减少了资源闲置浪费。

EASY 算法描述可参考文[1,4,5]。实现 EASY 算法用到两个基本数据结构：一个是等待作业队列和第一个作业的预约开始运行时间。一个是处理器资源的使用记录，它存储了每个处理器的当前状态(空闲或已分配)。EASY 算法要求用户提供作业需要的处理器数目和用户估计时间[2]。

EASY 算法存在的缺陷：

1. 用户必须提供作业的运行时间估计，用户的估计很不可靠[1]。

2. 偏爱小作业，小作业获得更多的回填机会。

3. 性能不稳定，容易受用户估计时间影响。

为了解决 EASY 算法存在的问题，本文引入基于优先级的抢占式并行调度方法(Priority-based Preemptive Scheduling)。

3　PPS 算法

PPS 算法使用的数据结构简单，只需要存储等待队列和运行队列信息。PPS 算法使用的概念：

1. 优先级：本文根据提交先后顺序确定优先级，提交越早优先级越高。

2. 优先级调度：系统在从等待队列选择作业调度时，优先选择优先级高的作业。

3. 可抢占资源：作业的动态属性，是指系统运行队列中所有比该作业优先级低的作业占用的资源总和。

4. 可抢占式调度条件：当一个作业等待执行时，系统当前的空闲资源不能满足其资源请求，但空闲资源加上作业的可抢占资源可以满足其资源请求。

5. 基于优先级的抢占式调度：当一个作业满足可抢占式

张国斌　硕士研究生，主要研究方向为高性能计算；**潘金贵**　教授，博士生导师，主要研究方向为中间件，Agent 技术，多媒体信息处理技术及应用。

[1]第一个作业是指等待队列中最早到达的作业。

[2]用户估计时间时指用户估计的作业执行时间，下文通用。

维普资讯 http://www.cqvip.com

调度条件时，系统当前的空闲资源不能满足作业的资源请求，而空闲资源加上作业的可抢占资源能满足其资源请求，那么系统就会冻结部分低优先级作业，释放资源，满足该作业资源请求，保证高优先级作业优先执行。本算法采用的抢占策略是优先抢占优先级低的作业，释放所占用资源，直到系统空闲资源可以满足作业的资源请求。

PPS算法描述如下：

1. 依据系统的优先级定义，评价系统中所有作业的优先级。

2. 将等待队列中的作业按优先级从高到低排序。

3. 循环遍历等待队列中的每一个作业。

a）如果系统当前的空闲资源可以满足作业的资源请求，执行该作业，转至d。

b）计算该作业的可抢占资源数量。

c）如果该作业满足可抢占式调度条件，调用抢占式调度子程序。

d）进入下一个作业处理循环。

抢占式调度子程序：

1. 选择当前运行队列中优先级最低的作业。

2. 冻结该作业，并将其按优先级插入等待队列，释放其占用资源。

3. 如果当前空闲资源可以满足高优先级作业资源请求，执行该作业，结束。

4. 转至1。

当作业提交和结束两个事件发生时，上面的调度过程都会被循环执行。

下面通过实验对PPS算法和EASY算法进行比较。

4 评价方法

评价主要考虑性能和服务公平性两类指标。性能指标主要有：平均响应时间和平均延迟系数。

1. 作业响应时间(Response Time)：从作业提交到完成的时间间隔，又称周转时间。偏向大作业。

2. 作业延迟系数(Slowdown)：定义为作业响应时间与执行时间的比，又称带权周转时间。作业执行时间是指在批调度系统中从作业开始运行到完成的时间间隔。

小作业的延迟系数一般很大，如执行时间为1s的作业等待[3]10min，延迟系数为600；而执行时间为1周的作业等待1年，延迟系数仅为50，为避免小作业的影响[2]，引入了有界延迟系数[4](Bounded Slowdown)，定义如下：

$$b\_sld=\begin{cases}\dfrac{t_r}{t_e} & \text{若 } t_e>10\\ \max(\dfrac{t_r}{10},1) & \text{否则}\end{cases}$$

其中 b_sld 是作业延迟系数，t_r 指作业响应时间，t_e 指作业执行时间。

平均响应时间和延迟系数均是以作业为单位的指标。从系统角度考虑的性能指标有资源利用率，它是用来衡量系统资源被有效利用的程度。假设系统有 N 个处理器，执行了 m 个作业，第一个作业的到达时间为0，最后一个作业的结束时间为 T，s 表示作业大小[5]，t 表示作业执行时间，则资源利用率定义为：

$$\mu=\frac{\sum_{i=1}^{m}s_it_i}{N\times T}$$

公式中，p、t 和 N 都是由负载确定，给定系统资源和负载后，T 的大小可由负载和调度算法确定，可见资源利用率主要是由负载和调度算法决定的。因为PPS和EASY均采用回填技术[4]来提高资源利用率，而EASY的主要设计目标就是最大化资源利用率[4]，进一步提高资源利用率的余地不大[9,10]。当负载中存在大量用户估计错误[6]时，EASY的部分计算可能被丢弃，而PPS不会出现这种情况。这在模拟实验中也得到验证。

以上指标均是为提高系统效率而设计的指标，为衡量用户对服务公平性的满意度，本文引入了公平性水平概念。

公平性的基本原则[6]：FCFS。即作业按照提交先后顺序依次被调度执行最公平。由于资源条件的限制，而FCFS的资源利用率很低，为了提高系统效率，降低对公平性水平的要求来换取更高的系统效率，现已出现了多种实现方案，EASY是其中最流行的一个。为了量度不同方案牺牲公平性的程度，引入不公平程度来表示服务顺序偏离FCFS顺序的程度。设Si和Ei分别表示作业i的提交次序和执行次序[7]。

Si<Ei：表示作业i被优先执行。

Si=Ei：表示作业i按照FCFS顺序执行，

Si>Ei：表示作业i被推迟执行，受到不公平服务。

| Si− Ei |表示第i作业偏离FCFS顺序的程度。| Si− Ei |的分散程度表示了服务偏离FCFS顺序的程度。具体衡量指标采用| Si−Ei |标准方差 ρ，ρ 的值越大表示偏离FCFS顺序的程度越大，偏离公平性越远。ρ 的值越小表示越接近FCFS顺序，公平程度越高。

5 实验方法与结果

本文分别实现了EASY和PPS的模拟系统，均采用事件驱动模型，主要处理作业提交和完成两种基本事件。

模拟系统处理SWF格式[12]的负载(Workloads)，生成相应算法的调度信息。实验用到的负载包括实际高性能计算机的历史负载日志和负载模型。

历史日志：

CM5：Los Alamos National Lab 1024-node Thinking Machine CM-5 (122,055 jobs, 10/1994～9/1996)，

KTH：the Swedish Royal Institute of Technology 100-node IBM SP2 (28,490 jobs, 9/1996～8/1997)

O2K：Los Alamos National Lab 2048-node Origin 2000 (121,989 jobs, 11/1999～4/2000)。

负载模型：

Lublin：该模型基于3个实际负载[11]，包括CM5和KTH (100,000 jobs)。

以上所有负载的详细信息可参见文[3]。O2K和Lublin中缺少用户估计时间。本文使用文[13～15]中的工具增加用

[3] 作业等待时间＝响应时间－执行时间

[4] 下文提到的延迟系数均指有界延迟系数。

[5] 作业大小指其所请求的处理器资源数量。

[6] 用户估计时间小于作业执行时间。

[7] 为表示抢占式调度系统中用户感觉到的起始执行顺序，本文引入虚拟起始时间，定义为：作业起始时间＝结束时间－运行时间。

维普资讯 http://www.cqvip.com

户估计时间。实验运用批次平均法(Batch Means)[7],将单个大的负载分成多个小的负载分批运行。批大小取文[8]中的推荐值5000。在分批之前对日志进行了预处理,剔除了日志中状态为Failed或Cancelled的作业,负载依次被分成20,4,19和20批。

表1 平均响应时间和平均延迟系数

Workload	负荷因子	平均响应时间			平均延迟系数		
		EASY	PPS	difference	EASY	PPS	difference
CM5	0.50	3006	2353	−21.7%	8.0	3.4	−57.5%
	0.75	5011	2689	−46.3%	24.0	8.1	−66.3%
	1.00	14207	3631	−74.4%	117.8	19.7	−83.3%
	1.25	36145	6105	−83.1%	379.1	50.8	−86.6%
	1.50	92914	14968	−83.9%	1095.4	183.1	−83.3%
KTH	0.50	6543	6500	−0.7%	6.6	5.6	−15.2%
	0.75	6883	6818	−0.9%	11.5	9.9	−13.9%
	1.00	7456	7298	−2.1%	20.5	18.3	−10.7%
	1.25	8411	8066	−4.1%	37.0	30.3	−18.1%
	1.50	9722	9516	−2.1%	58.6	58.1	−0.9%
O2K	0.50	3217	3079	−4.3%	1.4	1.1	−21.4%
	0.75	3333	3084	−7.5%	1.6	1.1	−31.3%
	1.00	3485	3093	−11.2%	2.0	1.1	−45.0%
	1.25	3976	3114	−21.7%	3.9	2.0	−48.7%
	1.50	4589	3236	−29.5%	14.5	4.5	−69.0%
Lublin	0.50	4664	2978	−36.1%	123.3	53.1	−56.9%
	0.75	7457	4007	−46.3%	256.2	98.5	−61.6%
	1.00	11140	6056	−45.6%	423.7	198.7	−53.1%
	1.25	15440	9243	−40.1%	622.2	364.3	−41.4%
	1.50	21120	13833	−34.5%	872.5	608.6	−30.2%

* difference表示调度方法从EASY变为PPS所引起的变化

表2 不公平程度和资源利用率

Workload	负荷因子	不公平程度			资源利用率		
		EASY	PPS	difference	EASY	PPS	difference
CM5	0.50	6	1	−5	0.239	0.239	0.000
	0.75	12	2	−10	0.358	0.358	0.000
	1.00	29	3	−26	0.472	0.477	0.005
	1.25	60	5	−55	0.572	0.594	0.022
	1.50	115	7	−108	0.634	0.705	0.071
KTH	0.50	2	2	0	0.195	0.195	0.000
	0.75	4	3	−1	0.293	0.293	0.000
	1.00	6	5	−1	0.390	0.390	0.000
	1.25	8	6	−2	0.486	0.486	0.000
	1.50	14	9	−5	0.580	0.581	0.001
O2K	0.50	3	1	−2	0.183	0.183	0.000
	0.75	5	2	−3	0.266	0.266	0.000
	1.00	7	3	−4	0.343	0.343	0.000
	1.25	15	4	−11	0.416	0.416	0.000
	1.50	34	8	−26	0.483	0.483	0.000
Lublin	0.50	3	2	−1	0.197	0.197	0.000
	0.75	5	3	−2	0.294	0.295	0.001
	1.00	7	4	−3	0.386	0.390	0.004
	1.25	9	5	−4	0.474	0.484	0.010
	1.50	11	6	−5	0.550	0.574	0.024

* difference表示调度方法从EASY变为PPS所引起的变化

为考察不同负荷水平对算法的影响,本文通过改变作业的到达速率来改变负荷水平。假设CM5的负荷水平为L,若将日志中每个作业的提交时间都乘以负荷因子1/1.5,则获得负荷水平为1.5L的负载,记作CM5-1.5。

实验结果如下:

• 平均响应时间和平均延迟系数

和EASY相比,PPS在这两项指标上均取得明显改善,平均改善程度分别为29.8%和44.7%(见表1)。而且在负荷加重的情况下,改善程度更明显。

• 资源利用率

PPS和EASY在这项指标上没有差别(见表2),两者的资源利用率都较高[4]。

• 公平性水平

和EASY相比,PPS的公平性水平得到大幅度提高(见表2)。PPS平均偏离公平的程度为4,EASY平均偏离公平的程度为18,显然PPS的公平性水平要远远高于EASY。而且在负荷加重的情况下,PPS偏离公平性的恶化速度明显慢于EASY。

结论 本文提出的PPS方法的两大优点:一是不需要用户估计作业执行时间;二是在最大化资源利用率的同时,最大程度的保证了服务的公平性。模拟实验结果表明,在性能指标和公平性指标两方面,PPS均优于EASY算法。可见,PPS方法是有效的。

参考文献

1 Mu'alem A W, Feitelson D G. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. IEEE Trans. Parallel and Distributed Systems, 2001, 12(6): 529～543

2 Feitelson D G. Metrics for parallel job scheduling and their convergence. In: Feitelson D G Rudolph L, eds. Proceedings of the 7th Workshop on Job Scheduling Strategies for Parallel Processing, volume 2221 of Lecture Notes in Computer Science, Springer-Verlag, 2001. 188～206

3 Parallel Workload Archive. http://www.cs.huji.ac.il/labs/parallel/workload/

4 Lifka D. The ANL/IBM SP Scheduling System. Job Scheduling Strategies for Parallel Processing. In: Feitelson D G, Rudolph L, eds. Springer-Verlag, 1995. 295～303

5 Srinivasan S, Kettimuthu R, Subramani V, Sadayappan P. Characterization of backfilling strategies for job scheduling. In: 2002 Intl. Workshops on Parallel Processing, held in conjunction with the 2002 Intl. Conf. on Parallel Processing, ICPP, 2002

6 Sabin G, Kochhar G, Sadayappan P. Job fairness in non-preemptive job scheduling. In: Inremnrionnl Conference on Parallel Processesing, 2004

7 MacDougall M H. Simulating Computer Systems: Techniques and Tools. MIT Press, 1987

8 Franke H, Jann J, Moreira J E, Pattnaik P, Jette M A. An Evaluation of Parallel Job Scheduling for ASCI Blue-Pacific. In: Proceedings of the ACM/IEEE SC99 Conference (SC'99)

9 Jones J P, Nitzberg B. Scheduling for parallel supercomputing: a historical perspective of achievable utilization. In: Job Scheduling Strategies for Parallel Processing, Springer-Verlag, Lect. Notes Comput Sci. 1999. 1659

10 Feitelson D G. On the Scalability of Centralized Control. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)

11 Lublin U, Feitelson D G. The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs. J. Parallel & Distributed Comput, 2003, 63(11): 1105～1122

12 Chapin S J, Cirne W, Feitelson D G, Patton Jones J, et al. Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. In: Job Scheduling Strategies for Parallel Processing, D. G. Feitelson, L. Rudolph, eds. Springer-Verlag, Lect. Notes Comput Sci, 1999, 1659: 66～89. URL http://www.cs.huji.ac.il/labs/parallel/workload/swf.html

13 Tsafrir D, Etsion Y, Feitelson D G. Modeling User Runtime Estimates. In: 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP). Lecture Notes in Computer Science Jun, 2005, 3834: 1～35

14 Tsafrir D, Etsion Y, Feitelson D G. A Model/Utility for Generating User Runtime Estimates and Appending Them to a Standard Workload Format File. http://www.cs.huji.ac.il/labs/parallel/workload/m_tsafrir0