区分任务类型的资源负载平衡算法——TDSA

刘克剑，刘心松，左朝树

(电子科技大学计算机科学与工程学院 四川成都 610054)

【摘要】介绍了一种区分任务类型的负载平衡算法。在该算法中，按照占用的系统资源状况，任务被划分成若干类型。调度时，各节点的负载依照待调度任务的任务类型和当前系统资源的负载动态计算，并依照计算结果在系统的一个子集范围内寻找适合解。该算法可以避免因为信息延迟造成的群聚效应，并可以满足少量对响应时间有特殊规定的任务的需求,已经用于自主设计的分布式操作系统DPOS，效果良好。

关 键 词 负载平衡； 区分任务类型； 分布式系统； 调度； 群聚效应波

中图分类号 TP316.4 文献标识码 A

A Task Differenced Scheduling Algorithm(TDSA) on Resource' s Load-Balancing

Liu Kejian，Liu Xinsong，Zuo Chaoshu

(School of Computer Science and Engineering, UEST of China Chengdu 610054)

Abstract We present a novel load-balancing algorithm to the distributed system. In the algorithm, tasks are classified by the resource which they would take up later. While scheduling, every node' s load is calculated dynamically through the task type and the resource' s overhead collected periodically. Then a suitable node is found to the task in a subset of the whole nodes. The algorithm can remove the herd effect caused by delayed information, and can decrease the mean response time of a few tasks that have special requirement in this regard. It has been adopted in a distributed operation system developed by our group.

Key words load-balancing; task differenced; distributed system; scheduling; herd effect

合理的负载平衡调度算法可以提高系统吞吐率，缩短任务响应时间[1~4]。很多负载平衡算法只考虑了系统的CPU资源，然而任务是多样性的，如计算型、IO型等。若调度时不区分任务类型，可能发生资源消耗不均衡的现象。相关的算法研究有多种，区分任务调度类型算法(Task Differenced Scheduling Algorithm, TDSA) 对k 子集算法进行改进，引入了任务类型和动态阈值，对大部分任务采用k 子集算法，仅对少量比例的任务采用最小负载算法。实验结果表明，如果比例恰当的话，在缩短此类任务响应时间的同时，还可提高整个系统的性能。

1 负载平衡算法介绍

定义 1 N为系统节点的集合，n为节点数，$n = |N|$。

收稿日期：2004 – 03 – 23

基金项目：四川省科技攻关基金资助项目(02GG006-018)

作者简介：刘克剑(1974 –)，男，在职博士生，现在西华大学计算机与数理学院工作，主要从事并行计算方面的研究；刘心松(1940 –)，男，教授，博士生导师，主要从事宽带网络、分布并行处理、操作系统和数据库等方面的研究；左朝树(1972 –)，男，博士生，主要从事分布式并行数据库方面的研究.

定义 2　M为系统资源集合，m为资源数，$m = |M|$。

定义 3　T_{upd}为信息更新周期。每隔T_{upd}时间，节点上的负载信息表就被更新一次。

定义 4　$\hat{\boldsymbol{a}}(i)$为节点i的资源负载向量，$\hat{\boldsymbol{a}}(i)=(\hat{\boldsymbol{a}}(i,1), \hat{\boldsymbol{a}}(i,2),\cdots, \hat{\boldsymbol{a}}(i, m))$。

定义 5　$\tilde{\boldsymbol{a}}(j)$为任务$j$的资源需求向量，$\tilde{\boldsymbol{a}}(j)=(\tilde{\boldsymbol{a}}(j,1), \tilde{\boldsymbol{a}}(j,2),\cdots, \tilde{\boldsymbol{a}}(j, m))$。

定义 6　$n\lambda$ (l<1)为系统总的任务到达率。

1.1　负载信息收集

负载信息收集方法有公告牌方式和分布式两种[1, 2]，后者的通信量较前者大。信息需要不断更新以保证数据的准确性，更新方式有周期性更新、连续性更新、独立更新和接入时更新几种。无论是哪种更新方式，负载信息的记录值与实际值总是不能完全同步，两者之间存在一个延迟。除了网络传输造成的延迟外，对周期性更新，延迟主要是由更新时间间隔引起的；对连续性更新，延迟主要是指从任务被调度到其真正到达指定节点进入就绪队列的时差。由于负载信息的延迟，调度时可能发生群聚效应(如最小负载策略，将在更新周期内到达的任务全部调度到同一节点上)，降低系统性能。文中采用了分布式负载收集方法，因为这样可以避免“单一节点失效”问题，更新方式采用周期性更新。

1.2　任务分类

根据任务对系统资源的需求状况进行分类，常见的任务类型有：1) 计算类，在 $\hat{\boldsymbol{a}}(j)$ 向量中，CPU资源的对应项数值最大。此类任务以消耗CPU资源为主；2) IO类，此类任务主要消耗磁盘IO带宽；3) 复合型任务，此类任务同时消耗多种资源，很难区分哪种资源是主要资源。

当然也可以按照虚拟的系统资源对任务进行分类，比如，可以将流媒体服务器的服务能力作为一种资源，将并发流数目作为负载指标。

1.3　节点负载计算方式

定义 7　节点i上任务j的任务负载 $L(i, j)$为

$$L(i, j)=\hat{\boldsymbol{a}}(i)\times\tilde{\boldsymbol{a}}(j)^T \tag{1}$$

定义 8　$y(j)$为任务j的负载阈值。$xload(j)=\max_{i\in N}\{L(i, j)\}$, $nload(j)=\min_{i\in N}\{L(i, j)\}$，并令$C_t$为一个常数，$C_t\in(0,1)$。轻载时$C_t$将为阈值门限。$y(j)$定义为

$$y(j) = \max\{(xload(j)+nload(j))/2, C_t\} \tag{2}$$

每类任务都有自己的资源需求向量，所以不同的任务看到的节点负载是不同的，定义7给出了描述。定义8 给出了一种动态的负载阈值表示方法，目的是保证被命中的节点有良好的服务能力，引入适当的C_t可以防止系统在轻载时选中高负载节点。

1.4　TDSA算法

即便缩短更新周期也不能完全消除延迟[1]，过小的刷新周期反而会加重网络负担。通常解决方法有k子集算法和基于时间预测的调度算法[1, 2]。k子集算法有较好的综合性能[1~4]。但由于概率原因，少数任务的调度结果也许远远差于平均值。在TDSA算法中，大部分任务将使用带有阈值的k子集算法进行调度，极小部分采用最小负载算法并设其概率为p_{short}。这样对于极少数有特殊要求的任务(比如紧急任务或者实时任务)，可以减少其平均响应时间。p_{short}应该随更新周期T_{upd}和任务到达率λ增大而减小。

$$p_{short}=\min\{1/(e^{T_{upd}/2}e^{\sqrt{T}}), C_s\}$$

其中C_s为一常数，$C_s\in(0,1)$，用于限定p_{short}的上限，防止群聚效应。TDSA算法过程如下：

TDSA algorithm:

Input: 任务j, 更新周期T_{upd}，系统总预计任务到达率l；

　　相关常数C_t ，C_s；

Output: 任务j分配的目标节点k；

1) 提取任务j的资源需求向量$\tilde{\boldsymbol{a}}(j)$；

2) 取得系统的节点负载向量$\hat{\boldsymbol{a}}(i)$；

3) $S(j)$为任务j待调度的节点集合，$S(j)\leftarrow\varnothing$；

4) 负载load记为L，

for i =1, 2, …, n , do

$L(i, j) \leftarrow \hat{\boldsymbol{a}}(i) \times \tilde{\boldsymbol{a}}(j)^T$;

计算阈值$y(j)$;

5) $p_{\text{short}} = \min\{1/(e^{T_{\text{upd}}/2} e^{\sqrt{T}}), C_s\}$ ；

6) 产生一个伪随机数r，有$r \in (0,1)$;

7) 计($r \leq p_{\text{short}}$) ，then

选择节点k，其中k满足条件$L(k, j) = \min_{l \in N}\{L(\text{i}, j)\}$；

8) else

for i =1, 2, …, n do

if ($L(i, j) \leq y(j)$) then $S(j) \leftarrow S(j) \cup \{i\}$;

if $|S(j)| = 0$ then

从1, 2,…, n中按照均匀概率随机选择一个节点k；

else if $|S(j)| = 1$ then

k取该集合中的唯一元素；

else

从集合$S(j)$中按照均匀概率随机选择两个节点a, d；

选择节点k，$k \in \{a, d\}$且满足$L(k, j) = \min\{L(a, j), L(d, j)\}$;

9) 返回节点k；结束。

2 算法性能分析

设节点按照负载值进行升序排列并依次编号。在k子集算法中，节点i成为k子集元素的概率为k/n，而它被命中的前提是k子集中不包含有任何排列在i之前的节点。故节点i被命中的概率px_i为

$$px_i = \begin{cases} \dfrac{\binom{n-i-1}{k-1}}{\binom{n-1}{k-1}} \dfrac{k}{n} & if \quad i \leq (n-k) \\ 0 \end{cases} \tag{3}$$

在不设定阈值的TDSA算法中，节点i被调度的概率为：

$$p_i = d_i p_{\text{short}} + (1 - d_i p_{\text{short}}) px_i \qquad d_i = \begin{cases} 1 & i = 0 \\ 0 \end{cases} \tag{4}$$

在设定阈值的情况下，设y为负载不高于阈值的节点数目。节点i命中的概率与式(4)类似，只需将式(4)中px_i项中的n全部替换为y 即可。由于$y < n$，故负载相对较小的节点的命中率进一步提高。

3 实 验

实验是在由8台PC机组成的同构系统上进行的，它们通过100 M的以太网交换机相连。节点配置如下：Pentium Ⅲ 1G、512 M内存、80 G硬盘。系统资源被划分为两类，一类是CPU资源，另外一类是磁盘IO资源。任务被划分为两类，第一种是计算型的任务，其主要消耗CPU资源，$\tilde{\boldsymbol{a}}$(cpu) = (1,0)；第二种是IO型任务，其主要消耗系统的磁盘IO和网络IO，$\tilde{\boldsymbol{a}}$(IO) = (0.2, 0.8)。

测试任务包括以视频流输出为代表的IO型任务，以矩阵乘法为代表的计算型任务。这些任务的运行时间服从均值为1的负指数分布，任务的到达强度服从泊松分布。TDSA算法中参数取值如下：k=2，C_t=0.5，C_s =0.5。测试每次持续300 s，平均响应时间为200次测试值的平均值。

图1表示了在l =0.5情况下，矩阵运算任务的平均响应时间。可以看出即便在不区分任务类型的情况下

(因为只考虑了CPU资源)，TDSA也有较好的性能。对比算法有随机算法和最小负载算法，k = 2 (k子集算法)。图2表示在混合任务情况下(50%的矩阵运算，50%的视频流任务)，l＝0.9 时任务的平均响应时间。图中p表示算法TDSA中的p_{short}的取值。其中p＝0时还附加了C_t＝1的条件。k = 2 表示不区分任务类型的纯粹的k子集算法。

实验结果表明，在TDSA算法中，轻载节点被命中的概率要高于k子集算法。另外，由于p_{short}的引入，能够在不产生群聚效应的情况下提升最小负载节点的命中概率。最后由于区分了任务类型，被调度节点上任务所需要的资源相对较为宽裕，会更适合任务的运行。总得而言，TDSA算法能够有效降低系统中任务的平均响应时间。

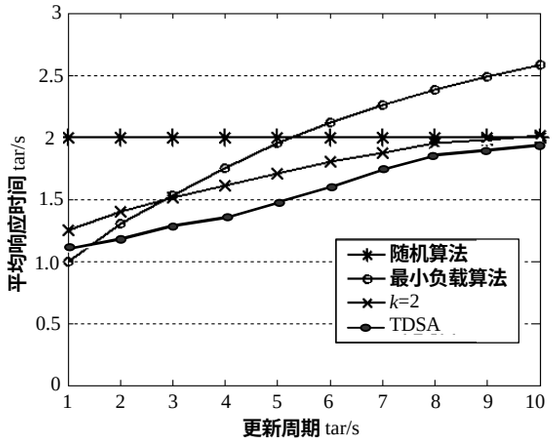

图1　l=0.5时任务的响应时间-更新周期曲线

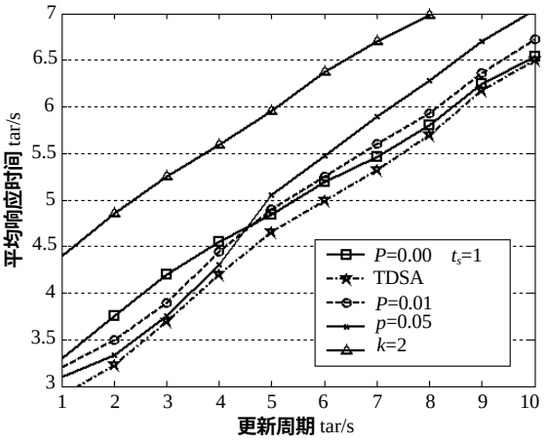

图2　l=0.9时任务的响应时间-更新周期曲线

4　结 语

TDSA算法将待调度的任务按其对系统资源的利用状况进行了分类，并从任务的角度去计算各节点的负载。在选择服务节点时，结合了k 子集算法、最小负载策略和动态阈值方法，提高了轻载节点的命中率。此外，TDSA算法中对部分任务采用了小概率的最小负载算法，可以缩短系统中一些有特殊要求的任务的平均响应时间。本算法已经在分布式操作系统(Distributed & Parallel Operation System, DPOS)中实现。

算法需要预先知道任务对资源的需求情况，并对任务类型进行了比较理想化的归类。这点可以在以后的工作中改进，比如加入自学习或反馈功能，比照任务的特点，按照其访问系统资源的频度进行自动归类。此外，还可以进一步考虑C_t、P_{short}对算法的影响。

参 考 文 献

[1] Michael M. The power of two choices in randomized load balancing[J]. IEEE Transactions on Parallel and Distributed Systems, 2001, 12(10): 1 094-1 104

[2] Michael D. Interpreting stale load information[J]. IEEE Transactions on Parallel and Distributed Systems, 2000, 11(10): 1 033-1 047

[3] Wang Yibing, Robert H. An improved algorithm of two choices in randomized dynamic load-balancing[A]. Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing[C]. Beijing 2002. 23-25

[4] 蒋　江, 张选民, 廖湘科等. 基于多种资源的负载平衡算法研究[J]. 电子学报, 2002, 30(8): 1 148-1 152

编　辑　熊思亮