山东农业大学学报(自然科学版),2015,46(5):779-782,784 VOL.46 NO.5 2015
Journal of Shandong Agricultural University ( Natural Science Edition ) doi:10.3969/j.issn.1000-2324.2015.05.026

# 基于计算机动态任务分配表的负载均衡新算法

方文英

杭州万向职业技术学院, 浙江 杭州 310023

**摘 要:** 随着计算速度的飞速发展，并行计算系统中，任务调度是解决多任务多资源情况下的最有效办法，但是目前常见的任务调度问题是一个 NP-Hard 问题，在任分配的负载均衡上还存在不足之处。本文通过改进并设计一个动态的负载均衡 Work-stealing 算法，来加强计算机集群动态任务分配过程中的效率，使得各个任务能够有条不紊的进行，从而提高整个计算机系统的资源利用率和整体性能。

**关键词:** 任务调度; 负载均衡; 动态任务分配表; work-stealing 算法

**中图法分类号:** TP391 **文献标识码:** A **文章编号:** 1000-2324(2015)05-0779-04

# An Improved Algorithm for Load Balance Based on the Dynamic Task Scheduling List of Computer System

FANG Wen-ying

*Hangzhou Wanxiang Polytechnic, Hangzhou* 310023, *China*

**Abstract:** With the development of computer science, the task scheduling method is the most efficient method for managing multitask and resources in a parallel computing system, but the problem of how to schedule tasks is a NP-Hard problem, the balance of scheduling has some shortcomings. In this article, we designed an improved dynamic task scheduling work-stealing method to strengthen the efficient of task scheduling in computers cluster and make a balance among all tasks so as to improve the resource use ratio and performance of whole computer system.

**Keywords:** Task scheduling; load balance; dynamic task scheduling list; work-stealing algorithm

## 1 计算机动态任务调度基本概念和算法

在云计算框架中，影响计算机性能的是任务的调度问题。经过大量研究者的研究表明，云计算或并行计算中的任务调度问题是公认的复杂度较大的 NP-Hard 问题。在众多任务调度算法中，效果最好的是基于负载均衡的调度算法，负载均衡是提高并行计算性能和计算机系统资源利用率的关键技术[1]。在计算机任务调度与分配过程中，主要分为调度模型和调度算法两个关键点。选择好的模型与对应算法，可以更好的进行任务调度的负载均衡处理。

### 1.1 调度模型

常见的动态任务分配表的调度模型分为分布式和集中式任务调度模型。基于分布式原型设计出的动态任务分配管理器，并投入实际任务调度使用的模型系统有 Yarn，Falkon，Gearman 等系统[2]。以上三个调度模型能够非常好的适应各种调度算法，以保证整个系统的负载均衡模式。其中，Gearman 模型是目前最常用的一种调度模型，在该模型上的各种负载均衡算法都有很好的应用。

### 1.2 调度算法

目前，基于负载均衡的调度算法研究主要分为两类[3]：(1) 静态渐变传播和交流算法，通过相连节点不断均衡达到整个系统负载均衡；(2) 动态负载均衡共享算法，通过随机选择多个节点进行均衡以达到最后的负载均衡，不强调节点相连。

本文主要针对目前较为常见的动态负载均衡算法进行研究，如今研究较多的动态负载均衡算法包括，随机算法、轮询算法、最小负载优先算法以及任务窃取算法等。本文着重分析 Work-stealing 任务窃取算法。

## 2 Work-stealing 任务窃取算法

### 2.1 Work-stealing 算法基本过程

**收稿日期:** 2014-08-20 **修回日期:** 2014-09-04

**作者简介:** 方文英(1976-),女,浙江杭州人,本科,讲师,主要研究方向为计算机应用. E-mail:136520276@qq.com

数字优先出版: 2015-10-06 http://www.cnki.net

Work-stealing[4]算法通过基于任务窃取的方式进行整体系统的负载均衡。该算法的窃取策略通过一种范式的方式均衡分配任务量，能够及时发现没有充分利用资源的计算节点，并从另一些处于繁忙计算的节点中解放一些任务给空闲状态中的计算节点使用，从整体架构上看，这样的策略可以使系统任务负载均衡。

Work-stealing 算法中每个处理器都拥有一个双端队列，且可以看成是一个调用栈，从底部插入就绪空闲状态的线程，等到其他处理器窃取任务的时候，从顶端删除该线程。可以将该队列命名为任务调度窃取栈，如图 1 所示。

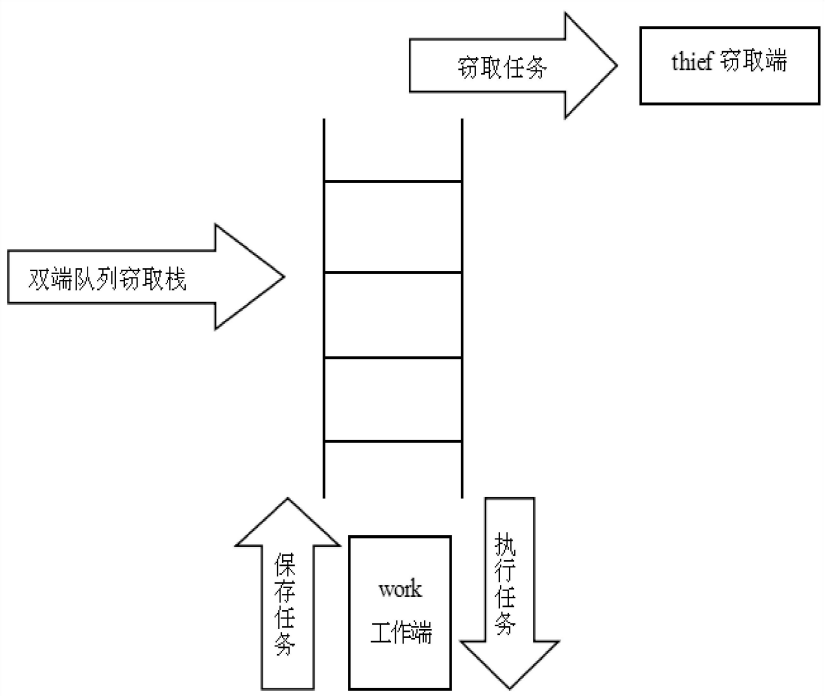


**图 1 任务调度窃取栈**
**Fig.1 Task scheduling steal stack**

当 Work 工作端发现负载不均衡的时候，将调用 Work-stealing 算法完成任务的窃取，将窃取到的任务从窃取栈中弹出，给窃取端使用。基本过程包括：

(1) 发现负载不均衡并想要窃取任务的服务器被设置成一个窃取端。

(2) 当待窃取端请求中心任务调度器需要窃取任务，中心调度器发来可以被窃取的处理器的相关信息，窃取端首先查询每个处理器管理的窃取栈，若栈非空，那么取出栈顶元素作为窃取的服务器。

(3) 若窃取栈为空，窃取端则会尝试随机选择另一个处理机进行窃取。通过不断的迭代寻找，直到寻找到可以窃取对应任务的处理机，然后访问该处理机并窃取处理不完的任务放进自己的任务队列中，根据队列规则依次执行任务。

### 2.2 Work-stealing 算法三种窃取任务数量策略

任务数量选择策略上，传统的 Work-stealing 算法提供了三种不错的计算策略，分别是加法级数策略、乘法级数策略和二分法级数策略。

(1) 加法级数策略：当需要确定窃取任务的数量的时候，采取加法级数的策略分析当前所需要的任务数，随着不断改变处理机，任务数随着加法级数增加。

(2) 乘法级数策略：当需要确定窃取任务的数量的时候，采取乘法级数的策略分析当前所需要的任务数，随着不断改变处理机，任务数随着乘法级数增加。

(3) 二分法级数策略：当需要确定窃取任务的数量的时候，首先统计获得的处理机的总队列任务，然后选择处理总队列中的一半任务。

三种算法策略适用于不同的场合，以下是各种策略的对比表格，如表 1 所示。

**表 1 三种窃取任务数量策略比较**

**Table 1 Comparison among three steeling tasks**

| | 加法级数<br>Additive series | 乘法级数<br>Multiplication series | 二分法<br>Dichotomy |
|---|---|---|---|
| 初始值 | S | S | 无 |
| 第 n 次窃取数量 | N*S | $S^n$ | 处理机总数量的一半 |
| 函数接口 | Int setNum(W vim) | Int setNum(W vim) | Int setNum(W vim) |

### 2.3 Work-stealing 算法两种窃取任务时机策略

Work-stealing 算法在窃取时机选择策略主要有两种，主要包括节点空闲时的任务窃取和节点快要空闲时的任务窃取策略。

2.3.1 节点空闲的时候进行任务窃取 当需要进行任务窃取的时候，处理机首先向中心任务调度服务器提出窃取任务请求。中心任务调度服务器查询各个处理机的当前状态，并返回处于满负荷状态下的处理机，这样需要执行窃取的处理机就可以进行窃取操作，改善那些处于满负荷状态下的处理机。

2.3.2 节点快要空闲的时候进行任务窃取 某节点在就绪队列中正在执行任务，但是该任务快要执行完成了，这时候进行任务窃取的请求。同时，请求过程也与节点空闲时候进行任务窃取的过程相同。

实际情况下，以上两种策略各有优缺点，视具体情况选择不同的算法策略。

## 3 改进的 Work-stealing 任务窃取算法

上文中介绍了传统 Work-stealing 任务窃取算法使用的窃取任务数量和窃取任务时机策略，这些策略能够很好的解决并行计算系统中的负载均衡问题。但是，到目前为止，该算法的所有策略研究都仅限于静态的某几个单一策略的组合，不能够很好的体现并行系统的时序性。

本文的改进策略是根据设定窃取时机的不同，每当需要进行任务窃取的时候，就开始启动 Work-stealing 算法，并选择相应的可以进行窃取的处理机，并在该处理机上执行窃取任务的操作。本文将负载最大的处理机作为备选的待窃取任务处理机，本文改进的算法是最大负载优先窃取算法。

### 3.1 改进算法流程

过窃取时机的确定，本文将需要窃取任务的处理机设置为“窃取机”，当被设置为“窃取机”之后，窃取机向任务调度中心服务器请求任务窃取，服务器首先会执行动态的负载均衡算法进行选择，挑选出目前负载最大的或者次大的等多个候选处理机作为“候选机”，候选机的选择如下：

(1) 任务调度中心服务器轮询现有队列中的各个处理机状态，通过计算每个处理机的最大任务队列的长度，比较选出结果最大的处理机，将该处理机作为候选机，然后将窃取机相关信息通过进程间通信返回给窃取机。

(2) 通过上步中的窃取机信息进行任务的窃取。由于进程间通信的延迟缘故，在窃取机接收到候选机的进程号的时候，该进程的具体信息需要延迟一段时间才能传播过来，这时候窃取机对其进行简单的预窃取工作。对于窃取任务的数量，本文则根据当前传播过来的候选机的任务数量做出最终决定，从候选机的就绪任务队列中弹出目前需要的任务数量，直到被窃取的任务数量达到为止。任务窃取成功，转(3)，否则，转(4)。

(3) 当前任务窃取已经成功，这时候需要动态更新目前保存在任务调度中心服务器的各个处理机的负载情况。

(4) 当前任务窃取失败，说明了所有处理机的任务都已经完成了，目前不存在可以进行窃取的任务了。

(5) 窃取机从候选机中窃取到了相应数量的任务，并执行这些任务。

(6) 算法结束，系统中目前的任务全部执行完成。

### 3.2 改进算法流程

本文改进的算法是最大负载优先的 Work-stealing 任务窃取算法。概算能够根据当前各个处理机的任务量的多少来决定如何分配窃取任务。根据 3.1 节中提到的算法步骤可以得到本文算法的流程

图，如图 2 所示。

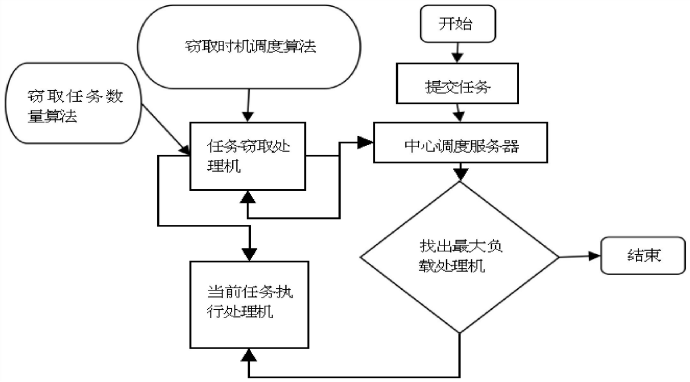


**图 2 本文改进算法流程图**

**Fig.2 Flow chart of the improved algorithm in this paper**

其中，窃取时机调度算法细节如图 3 所示。在处理机需要窃取多少任务的数量确定流程上，如图 4 所示。

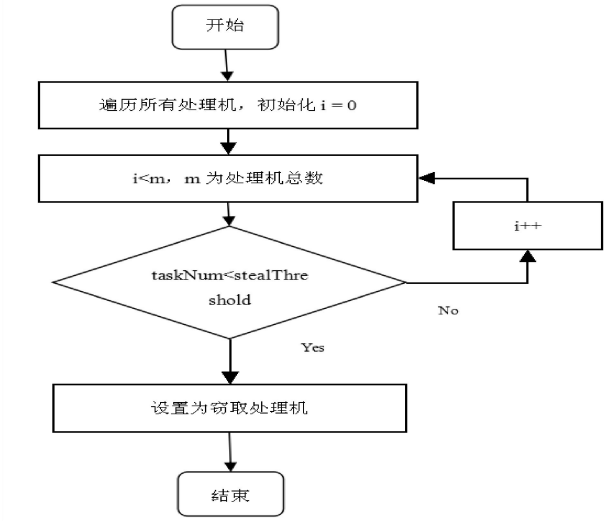


**图 3 窃取时机调度算法流程图**

**Fig.3 Flow chart of Scheduling algorithm for stealing time**

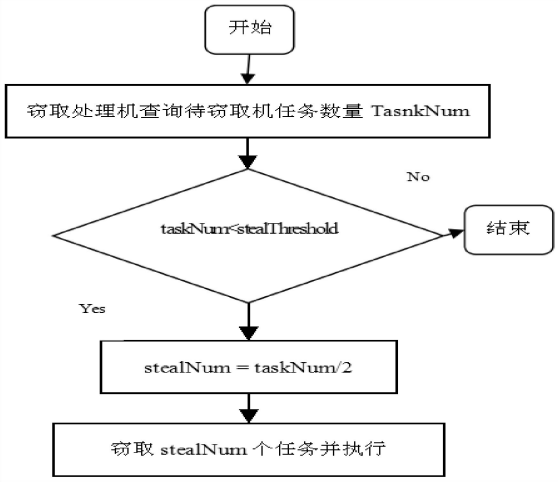


**图 4 窃取任务数量流程图**

**Fig.4 Flow chart of stealing tasks**

### 3.3 改进算法与传统 Work-stealing 算法实验对比

本文通过搭建一个原型系统测试本文改进算法与传统 Work-stealing 算法。在原型系统中创建了 10 台服务端，每个服务端最高负载量为 10 个任务。在传统算法中，分别对三种窃取任务数量策略和两种窃取任务时机策略进行实验验证。验证结果如下表 2 所示。

**表 2 传统算法与改进算法比较**

**Table 2 Comparison between the traditional and improved algorithms**

| | 加法级数 Additive series | 乘法级数 Multiplication series | 二分法 Dichotomy | 空闲时机 Free time | 快要空闲时机 Going to free | 本文算法 Algorithm in this paper |
|---|---|---|---|---|---|---|
| 初始平均任务量 | 10*10 | 10*10 | 10*10 | 10*10 | 10*10 | 10*10 |
| 负载均衡过程中平均任务量 | 10*8.5 | 10*7.2 | 10*7.9 | 10*8.2 | 10*7.4 | 10*6.2 |
| 负载均衡过程中最大处理机任务量 | 10 | 8 | 7 | 8 | 9 | 6 |
| 负载均衡过程中时间消耗(ms) | 1324 | 3512 | 2122 | 1899 | 1766 | 898 |

从上表中可以看出，本文改进算法的优势在于，由于是通过动态不断改变窃取任务量的，所以本文算法的任务量较为平均，负载较为均衡。另外，本文算法能够快速确定窃取任务数量和时机，时间复杂度低。

(下转第 784 页)