计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201200126

基于机器学习的分布式星载 RTs 系统负载调度算法

谭双杰[1,2]　林宝军[1,2,3,4,5]　刘迎春[2,3,4]　赵　帅[2,4]
1 上海科技大学信息科学与技术学院　上海 201210
2 中国科学院微小卫星创新研究院　上海 201203
3 中国科学院大学计算机科学与技术学院　北京 100094
4 上海微小卫星工程中心　上海 201210
5 中国科学院空天信息创新研究院　北京 100094
(tanshj@shanghaitech.edu.cn)

摘　要　分布式星载多 RTs(Remote Terminal)系统的任务主要基于功能进行分配,而数据处理任务的突发性往往会使不同计算机之间负载不均衡。运用灵活的负载调度机制,可以有效调节不同计算机间的负载差异,从而在一定程度上提升计算机系统的整体性能。文中提出了一种基于机器学习的分布式星载 RTs 系统负载调度算法,包含样本采集、任务吞吐率预测模型构建、吞吐率预测和负载调度等 4 个步骤。在构建任务吞吐率预测模型环节,通过机器学习的线性回归正规方程获取模型权重,缩短了构建模型消耗的时间。在负载调度环节,若 RTs 的吞吐率之和大于系统总的负载数据量,则按吞吐率比例给各 RTs 分配数据,否则只给负载数据量小于自身吞吐率的 RTs 分配一定量的数据。在多台星载计算机电性能产品构建的地面模拟系统上的实验结果表明,该算法可以使系统所有节点的平均 CPU 利用率提高 23.78%,节点间的 CPU 利用率方差降低至 34.59%,同时目标任务的系统总吞吐量显著提升 225.97%。也就是说,该方法在确保系统负载均衡性的同时,可有效提高系统的资源利用率,提升星载计算机系统的数据实时处理性能。

关键词:分布式系统;星载计算机;机器学习;任务调度;动态负载均衡
中图法分类号　TP393

Load Scheduling Algorithm for Distributed On-board RTs System Based on Machine Learning

TAN Shuang-jie[1,2],LIN Bao-jun[1,2,3,4,5],LIU Ying-chun[2,3,4] and ZHAO Shuai[2,4]
1 School of Information Science and Technology,ShanghaiTech University,Shanghai 201210,China
2 Innovation Academy for Microsatellites,Chinese Academy of Sciences,Shanghai 201203,China
3 School of Computer Science and Technology,University of Chinese Academy of Sciences,Beijing 100094,China
4 Shanghai Engineering Center for Microsatellites,Shanghai 201210,China
5 Aerospace Information Research Institute,Chinese Academy of Sciences,Beijing 100094,China

Abstract　The tasks of distributed on-board multi-RTs (remote terminals) system are mainly distributed based on functions, while the burstiness of data processing tasks often leads to unbalanced load among different computers. Using a flexible load scheduling mechanism can effectively adjust the load difference between different computers, thereby improving the overall performance of the computer system to a certain extent. A load scheduling algorithm for distributed on-board RTs system based on machine learning is proposed in this paper, which includes four steps: sample collection, task throughput prediction model construction, throughput prediction and load scheduling. In the process of constructing the task throughput prediction model, the weight of the model is obtained through the linear regression normal equation of machine learning, which reduces the time spent in constructing the model. In the load scheduling link, if the total throughput rate of RTs is greater than the total load data volume of the system, data will be allocated to each RT in proportion to the throughput rate; otherwise, only a certain amount of data will be allocated to RTs whose load data volume is less than their own throughput rate. The test results on the ground simulation system constructed by multiple on-board computers electrical performance products show that the algorithm can increase the average CPU utilization rate of all nodes of the system by 23.78%, and reduce the variance of CPU utilization rate between nodes to 34.59%. The total system throughput of the task is significantly increased by 225.97%. In other words, this method can effectively improve system resource utilization while ensuring system load balance, and improve the real-time data processing performance of the on-board computer system.

到稿日期:2020-12-14　返修日期:2021-04-19　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
通信作者:林宝军(linbaojun@aoe.ac.cn)

Keywords Distributed system，On-board computer，Machine learning，Task scheduling，Dynamic load balancing

1 引言

由于人们对空间网络技术的要求越来越高，卫星通信系统逐渐向智能化发展[1]，其中比较具有代表性的有天地一体化网络系统。该系统覆盖太空、空中、陆地、海洋等自然空间，其通信链路中传输的数据不仅种类繁多，而且传输速率更是最高可达 100 G/s[2]，使星载计算机需要转发和处理的数据量都急剧增加。

为满足日益上升的信息处理需求，星载计算机需要具备更多的通信接口、更快的处理速度以及更大的存储空间。但是，星载计算机由于其工作环境的限制，性能提升相对滞后。例如，在国外，美国国家航空航天局主要使用的 RAD750 处理器的处理能力上限为 380 MIPS，内存只有 256 M[3]。在国内，国产“龙芯”1E/1F 处理器采用了 CPU＋FPGA＋FLASH 的综合结构，但其总体性能也只有 300 MIPS[4]，存储空间不超过 1GB。对海量数据进行复杂计算已经超过单台计算机的处理能力，这不仅会降低卫星信息传输的实时性，也给存储带来压力。在不改变硬件基础的条件下，采用分布式计算可以提高计算机网络中资源的有效利用率，加快计算机对大量数据的处理速度[5]。因此，在星载计算机系统上采用分布式计算是不可避免的。

中国科学院斗卫星已采用多处理器的分布式信息网络架构，但其微小卫星内部仅通过任务资源分散配置来实现系统的整体运行[6]，容易造成处理能力的冗余。例如，当某节点 RT 对速率为 5 Gb/s 的激光通信[7]传输的高清图像作去噪或加密处理时，往往会因为数据处理不及时而导致数据阻塞，使此 RT 成为通信瓶颈。同时，一些负责间歇运行任务的节点却处于空闲状态，系统资源利用不均衡。因此，如何为星载 RTs 系统选择一个合理的任务分配及调度方案，保证系统负载均衡将成为提升其性能的一个重要研究方向。

2 相关工作

分布式计算系统出现后，人们致力于解决如何将任务分配到合适的处理器上。异构系统任务的分配需要考虑的因素众多，如 CPU 的频率和利用率、ram 使用情况、磁盘 I/O 等，综合考虑所有因素得到最优的分配策略被证明是一个 NP 完全问题[8-10]。

随着机器学习（ML）技术的兴起[11-12]，分布式计算系统上任务的分配问题得到了有效解决。Nemirovsky 等[13]采用人工神经网络（ANN）技术建立对不同任务分配方案进行性能预测的回归模型，以达到最大化系统吞吐量的目的，但在建立模型时消耗的资源过多。文献[10]考虑到了这一点，使用 ANN 算法通过离线训练得到不同任务分配方案的性能预测模型，但离线训练的模式使得预测模型无法进行灵活的更新。同时，任务的分配是基于一对一映射模式的，当某些任务使用的资源远多于其他任务使用的资源时，即使是最优解也很难做到各处理器资源利用的均衡。

Micolet 等[14]使用 K-邻近算法和线性回归算法来构建预测模型，将任务以多线程模式和最优的方案分配到不同的内核上，在提升系统执行任务性能的同时做到相对负载均衡。但该方法在选择最优方案时消耗的计算资源过多，并且对已分配但未执行的任务没有调度功能。Patni 等[5]提出了一种以节点负载量和计算能力为依据，转移计算处理时间长的节点的负载到处理时间短的轻载节点上。但是，该算法获取节点的计算能力需要对内核进行操作，容易造成系统不确定性[14]。

考虑到星载计算机上任务种类较少的特点，本文算法的负载调度对象不再为任务整体，而是所有任务中某个任务需要处理的数据单元。以此为前提，本文提出了一种基于机器学习的分布式星载 RTs 系统负载调度算法（ML-OBLS），该算法主要在以下方面进行改进。

（1）采用机器学习回归问题中相对简单的线性回归（LR）[12,15-17]正规方程来构建性能预测模型，降低了模型的复杂程度。该算法使每个 RT 都建立各自的预测模型，将整体预测划分为多个分预测，减轻了主节点的负担。

（2）主节点以各个 RTs 预测的任务吞吐率和任务负载量为标准来调整负载，不需要循环迭代选择最优方案，缩短了分配过程造成的延时，避免了对内核进行操作。

3 分布式星载 RTs 系统描述

3.1 系统模型

某型中高轨卫星的分布式星载 RTs 系统如图 1 所示，其中 RT1 为主节点。各 RTs 上分别运行着不同的专项任务和相同的任务 taskA，并且所有专项任务运行所需资源都能得到单个 RT 的充分满足。而任务 taskA 运行消耗大量的 CPU 资源，RT1 无法独自承担，因此需要其他节点在自身的专项任务不使用计算资源时，帮助 RT1 处理数据。图 1 中，信息流①为需要 taskA 处理的数据，信息流②为子节点反馈给主节点的信息，包括 taskA 的吞吐率、吐率预测值和负载数据量等。

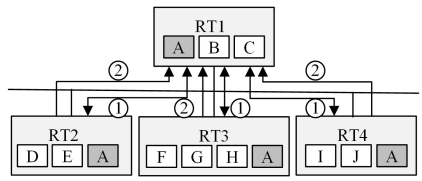

图 1 分布式星载 RTs 系统

Fig. 1 Distributed on-board RTs system diagram

进行负载调度时，使用反馈机制可以使负载平衡系统能够动态地调整参数和算法，从而获得更好的负载平衡效果[18]。ML-OBLS 算法采用了反馈机制，它主要包含两部分：1）节点 RTs 建立吞吐率预测模型，首先各 RTs 收集自身的负载信息，通过 LR 获取预测模型的权重 w，随后，RTs 获取其最新的负载信息，通过预测模型来预测各自 taskA 的吞吐率值 v'，并将 v' 及负载数据量等反馈给主节点 RT1；2）负载均衡调度，RT1 汇集所有节点信息，调用负载分配函数求得负载分配、重分配策略，在保证 RTs 的 CPU 利用率均衡的条件下

提升 taskA 的系统吞吐率。ML-OBLS 算法的负载调度流程如图 2 所示。

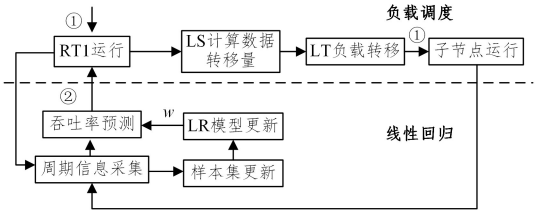

图 2 负载调度流程图

Fig. 2 Flow chart of load scheduling

由图 2 可以看出，调度算法运行是基于周期性反馈实现的主节点和子节点异步运行。这种调度模式使得它们之间的信息交流存在一定的时间差，主节点对子节点的控制是非实时的，因此本文算法的使用场合为对同步计算和实时等级要求不高的 s 级任务。

3.2 信息和信息集的定义

定义 1 给定周期时长 T，节点负载信息 $Load$ 为节点 RT 的最新一周期 T 内的负载详细信息：

$$Load=\langle c,c',r,io,a,v,rd\rangle$$

其中，c 为节点一周期内 CPU 利用率的百分比；c' 为 taskA 一周期的 CPU 占有率的百分比，因此有 $c'\leqslant c$；r 为节点内存利用率；io 为磁盘 I/O 等待队列长度；a 为节点中处于活跃状态的任务数；v 为 taskA 的周期吞吐率；rd 为 taskA 未处理的数据量。

定义 2 目标任务信息集 $Task$ 为主节点记录各 RTs 上 taskA 最新运行状况的集合：

$$Task=\{<v_i',rd_i,\max V_i,td_i>\mid \forall i\in\Omega\}$$

其中，$\Omega=\{1,2,\cdots,K\}$，K 为节点 RTs 的个数；v_i' 为节点 i 上 taskA 的预测吞吐率值；$\max V_i$ 为历史最大处理速度；rd_i 为未处理数据；td_i 为节点 i 的数据转移值，$td_i>0$ 时，表示需要接受 $|td_i|$ 的数据；$td_i<0$ 时，表示需要转出 $|td_i|$ 的数据。

定义 3 给定一正倍数 α 及所有节点中最大预测吞吐率的节点序号 j，有效节点集 $\Psi(Task)$ 为 taskA 的预测吞吐率不小于 v_j/α 的节点 id 集合：

$$\Psi(Task)=\{i\mid \alpha v_i\geqslant v_j,i\in\Omega\}$$

从定义 3 可以看出，非有效节点的任务 taskA 的处理速度明显慢于有效节点，说明此时非有效节点上其他任务占用了大量 CPU 资源，此时不宜向非有效节点转入数据。

4 ML-OBLS 算法设计和模型构建

本文算法制定负载调度策略，以各节点的预测模型预测的吞吐率为主要判断标准之一，因此构建完整的预测模型是算法运行的必要条件。本节将先详细阐述用线性回归技术构建吞吐率预测模型的过程，然后介绍主节点根据节点 rd 和 v' 的比值大小，即节点数据处理周期数的多少来计算各节点 td 的值，以及实现系统计算资源均衡利用的过程。

4.1 LR 预测模型构建

算法初始时，RT1 给各节点平均分配数据。随后，各 RTs 开始处理数据，并独自获取其节点负载信息 $Load$，将信息 $\langle 1,c',r,io,a\rangle$ 及 v 分别放入其负载信息样本集 $\boldsymbol{X}$ 和历史吞吐率集 $\boldsymbol{V}$ 中。算法运行一段时间后，当有 RTs 获取的新样本不少于样本容量 M 的一半时，通过 LR 正规方程得到吞吐率预测模型的权重 $\boldsymbol{w}$。

设 $x=<1,c',r,io,a>$，$\boldsymbol{x}\in\boldsymbol{X}$ 为模型的输入，$\boldsymbol{w}=\langle w_0,w_1,w_2,w_3,w_4\rangle^{\mathrm{T}}$，其中 w_j，$j\in\{0,1,\cdots,4\}$ 为 x 各维度对应的权重。由文献[16]可知，向量 $\boldsymbol{x}$ 和向量 $\boldsymbol{w}$ 相乘可得预测函数 $f(\boldsymbol{w};\boldsymbol{x})$，如式(1)所示：

$$f(\boldsymbol{w};\boldsymbol{x})=w_0+w_1c'+w_2r+w_3io+w_4a=\boldsymbol{xw} \tag{1}$$

建立预测模型的目的就是得到一个最优的 $\boldsymbol{w}$ 的值，使模型输出和 $\boldsymbol{x}$ 对应的实际吞吐率 v 的差距最小，为此需要选择一个损失函数来评估得到的 $\boldsymbol{w}$ 的准确性。一般情况下，使用均方误差(Mean Square Error，MSE)作损失函数，在回归问题中效果良好，将历史负载信息矩阵 $\boldsymbol{X}$ 代入式(1)后，结合历史吞吐率矩阵 $\boldsymbol{V}$，可求得所有样本的总损失代价函数，如式(2)所示：

$$L(\boldsymbol{w};\boldsymbol{X},\boldsymbol{V})=(f(\boldsymbol{w};\boldsymbol{X})-\boldsymbol{V})^2 \tag{2}$$

由文献[15]可知，当 $\boldsymbol{x}$ 维度小于 1 000 时，使用正规方程法[19]求拟合曲线的参数值比其他方法更为简单。由于篇幅有限，在此将不详细描述正规方程的求解过程，所求权重公式如式(3)所示：

$$\boldsymbol{w}=(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{V} \tag{3}$$

其中，$\boldsymbol{X}^{\mathrm{T}}$ 为负载信息矩阵 $\boldsymbol{X}$ 的转置矩阵。节点通过方程式获得 $\boldsymbol{w}$ 后开始进行吞吐率预测，由于预测的是节点未来一周期吞吐率的最大值，因此需求得 taskA 可能占有的最大 CPU 资源比例为 $100-(c-c')$。然后，将 $\boldsymbol{x}=<1,100-(c-c'),r,io,a>$ 及式(3)中的权重 $\boldsymbol{w}$ 代入式(1)，可得 taskA 的吞吐率预测值 $v'=f(\boldsymbol{w};\boldsymbol{x})$。各节点上吞吐率的预测算法如算法 1 所示。

算法 1 吞吐率预测算法

输入：(Load，**X**，**V**)

输出：v′

```
1. /* 各 RTs 周期性获取其负载信息 */
2. GetInfo(Load,X,V)
3. if |X|>M/2/* 若获取超过一半容量的新样本 */
4. w←lr(X,V)/* 线性回归方程得到模型权重 w */
5. end if
6. if ‖w‖≠0
7. x←{1−(c−c′),r,io,a}/* 获取预测模型输入 */
8. v′←xw/* 获得吞吐率预测值 */
9. else
10. v′←v/* 吞吐率预测值为上周期吞吐率 */
11. end if
12. return v′
```

设 $N=5$ 为 $\boldsymbol{x}$ 维数，由算法 1 可知，在没有建模的周期中只进行 $\boldsymbol{x}$ 和 $\boldsymbol{w}$ 的乘积操作，算法的时间复杂度为 $O(\boldsymbol{N})$；在进行建模的周期中，主要进行 $\boldsymbol{N}\times\boldsymbol{M}$，$\boldsymbol{M}\times\boldsymbol{N}$ 矩阵的乘运算和 $\boldsymbol{N}\times\boldsymbol{N}$ 矩阵的求逆矩阵运算，此时算法 1 的时间复杂度为 $O(\boldsymbol{MN}^2+\boldsymbol{N}^3)$，选择较少的负载信息减少了总算法的计算量[20]。因此，预测模型只在有限的周期里才会消耗较多的计算资源，不进行建模运算时，则可以快速地获得吞吐率的预测值。各 RTs 得到自己的吞吐率预测值 v' 后，将 v'、该周期的吞吐率 v、负载数据量 rd 反馈给主节点进行调度决策。

4.2 基于时间均衡的负载调度算法

主节点汇集 RTs 的负载信息到 Task 集合中，然后制定负载调度策略。调度的主要目的是保证 RTs 在下一周期内计算资源均衡使用，如使 RTs 的 CPU 占有率相同。除此之外，还要避免数据迁移占据过多的总线资源，当 taskA 的计算速度产生小幅度波动时，此变化并不影响系统节点在下周期内的计算资源使用情况，不进行数据迁移操作，从而节省通信资源。因此，本文根据所有节点总的剩余未处理数据量的不同，设计了两种数据负载调度策略。

方案 1　当$\sum rd_k$小于$\sum \max V_k$时，表明下周期系统可以处理完所有 taskA 的数据，并存在一定空暇资源。为保证各节点 CPU 利用率均衡，主节点根据预测速度比例分配数据，因此有数据转移量如式(4)所示：

$$td_i=\frac{v_i'*\sum rd_k}{\sum v_k'}-rd_i \tag{4}$$

若存在非有效节点的 td 值为正值，则将要转入它的数据均分到有效节点上，对所有节点数据的调度值作如式(5)所示的调整：

$$\begin{cases} td_j=td_j+td_i*\dfrac{v_j'}{\sum\limits_{k\in\Psi(Task)}v_k}, & j\in\Psi(Task) \\ td_i=0, & i\notin\Psi(Task)\wedge td_i>0 \end{cases} \tag{5}$$

方案 2　当$\sum rd_k$大于$\sum \max V_k$时，表明下一周期系统无法处理完所有 taskA 的数据，系统资源利用率均衡，即表明所有节点都拥有至少 $\max V$ 的数据，各节点都不存在资源空闲的情况。因此，若存在节点 $i\in\Psi(Task)$且td_i小于 $\max V_i$，则向该节点转入如式(4)所示的量的数据。数据转出方为所有节点中预估处理数据所需时间最长的节点，设其序号为 mId。为防止节点 mId 转出的数据过多，需要保证它转出数据后剩余的数据的处理时间不短于节点 i，因此节点 mId 的数据转移值如式(6)所示：

$$td_{mId}=\begin{cases} td_{mId}-td_i, \text{if } \dfrac{(rd_{mId}+td_{mId}-td_i)}{v'_{mId}}>\dfrac{rd_i+td_i}{v_i'} \\ td_{mId}-\left(rd_{mId}+td_{mId}-v'_{mId}*\dfrac{(rd_i+td_i)}{v_i'}\right), \\ \qquad \text{else} \end{cases} \tag{6}$$

若节点 mId 可以转出的数据小于 td_i，则选出新的负载最重节点，再通过式(6)计算调度值。循环此过程，直到转出值和转入值相等。主节点上负载转移量计算算法如算法 2 所示。

算法 2　负载转移量计算算法

输入：(Task，Ψ(Task))

输出：TD={td_i|i=1，…，K}

```
1. if Σ(k=1..K) rd_k < Σ(k=1..K) max V_k / * 选择计算方案 1 * /
2.    / * 用式(4)计算转移量 * /
3.    for all i∉Ψ(Task) and td_i>0
4.       / * 用式(5)调整转移量 * /
5.    end for
6. else/ * 选择计算方案 2 * /
7.    for all i∈Ψ(Task) and rd_i<max V_i
8.    / * 按式(4)计算有效节点 i 接收的数据量 * /
9. num←td_i
10.       while(num>0)/ * 没有足够数据转入节点 i * /
11. mId=arg max((rd_i+td_i)/v_i') and mId≠i
12.          / * 用式(6)计算节点 mId 转出数据量 * /
13.          / * 使 num 为 num 减去td_mId的减小值 * /
14.    end while
15.  end for
16. end if
17. return TD / * 返回节点数据转移量 * /
```

由算法 2 可知，方案(1)和方案(2)都是先通过式(4)计算 K 个节点的计算转移量，然后分别通过式(5)和式(6)调整所有节点的转移量，以保证数据的转入和转出平衡。因此，算法 2 的时间复杂度为 $O(K*K)$，即 $O(K^2)$，在节点数小于 100 的微小卫星内可以快速制定数据调度策略。

进行负载调度时需要考虑计算机的承受能力，避免处理器长时间处于高负荷运算状态，因此本文算法不对非有效节点调入数据。同时，当有效节点有不少于历史最大吞吐率的数据时，也不向其转入数据。此操作可以预防未来一周期内节点响应其他任务而导致 taskA 的 CPU 占有率下降时，数据又被转出，避免数据被频繁地在节点间转移，减轻了总线的通信压力。

4.3 实时预测模型构建

为了保证速度预测的准确性，需要不定期地更新线性回归模型中得到的权重 $\boldsymbol{w}$，为此必须对历史记录样本集中的数据进行实时更新。

(1)样本更新。给定样本邻域半径 ε，假设 $\boldsymbol{x}$ 和 v 分别为样本集 $\boldsymbol{X}$ 和 $\boldsymbol{V}$ 中最近加入的样本，现有最新获取的 taskA 负载信息 $\boldsymbol{x}'$和吞吐率 v'，若使$|x_j-x'_j|<\varepsilon$ 成立的x'的元素个数为 5，则表示节点及 taskA 的运行状况变化很小，更新$\boldsymbol{x}=(\boldsymbol{x},\boldsymbol{x}')/2$ 以及 $v=(v+v')/2$；否则，将 $\boldsymbol{x}'$和 v'加入 $\boldsymbol{X}$ 和 $\boldsymbol{V}$ 中，但当$|\boldsymbol{X}|=M$时，需要先删除最早的样本。

(2)模型更新。预测模型在两种情况下进行更新：1)任务没有数据需要处理的非工作时间；2)更新 $M/4$ 个样本时。设新的模型公式为$f_1(\boldsymbol{w};\boldsymbol{x})$，旧的模型公式为$f_0(\boldsymbol{w};\boldsymbol{x})$，$M'$为两次建模间的更新的样本数值，可得模型更新公式如式(7)所示：

$$f(\boldsymbol{x};\boldsymbol{w})=\frac{M'}{M}f_1(\boldsymbol{w};\boldsymbol{x})+\frac{M-M'}{M}f_0(\boldsymbol{w};\boldsymbol{x}) \tag{7}$$

5 测试与结果分析

5.1 测试环境

计算机网络配置：4 个基于 vxworks 系统的龙芯 1F03 开发板，CPU 频率 33 MHz，内存 128 M。如图 3 所示，主节点和子节点通过 1553B 数据总线连接，总线传输速率为 1 Mb/s。

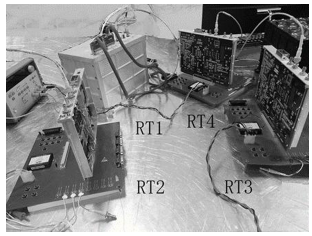

图 3　分布式星载 RTs 系统实物图

Fig. 3　Physical picture of distributed on-board RTs system

RT1 上主要布置有数据加密任务 tEncode，对 64 bytes

无符号数组进行 aes 加密，处理器一直持续运行，节点负载较重；RT2 上主要布置有自主定轨算法 tArith，根据卫星间双向测距值等采用卡尔曼滤波法定轨，处理器间歇式地工作，节点负载较轻；RT3 上主要布置有位置用户轨道多项式拟合任务 tOrbit，利用切比雪夫多项式拟合卫星轨道进行 GPS 定位，工作模式也为间歇式，节点负载较轻；RT4 上主要布置有星间链路建链辅助参数计算任务 tSatelliteLink，通过分析星历信息得到辅助建立卫星间链路的参数，只在有建链需求时才运行，因此节点负载整体适中。

5.2 测试过程

本文将 RT1 上的 tEncode 任务作为被调度任务，在 4 个节点上都布置该任务，并为节点分配需要加密的数据。实验主要从两方面进行：建立预测模型和负载数据调度，其中使用 LR 线性回归建立吞吐率预测模型的参数设置如表 1 所列。

表 1 线性回归参数

Table 1 Parameters of linear regression

模型参数	参数值
采样周期/s	6
样本容量	200
样本邻域半径 ε	0.05
$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 奇异值分解学习率	0.0001
奇异值分解迭代次数	100

调度数据时，各节点的内存消耗变化可忽略不计，资源利用变化以节点 CPU 利用率为主要标准。因此，为测试 ML-OBLS 算法的动态调整效果，每隔 $\Delta T=6s$ 个周期随机增加除 tEncode 任务外的其他任务的 CPU 占有率。经本算法调度后，系统负载均衡的直观表现为节点 CPU 利用率的均衡，设 $var(cpu)_i$ 为第 i 个周期 4 个节点的 CPU 利用率方差，平均方差如式(8)所示：

$$\overline{var(cpu)}=\frac{\sum_{i=1}^{400}var(cpu)_i}{400} \tag{8}$$

5.3 测试结果

系统各节点上 LR 模型的预测效果如图 4 所示，图中 x 轴表示算法运行的周期，y 轴表示模型预测的 tEncode 任务的吞吐率与其实际吞吐率的差的绝对值。

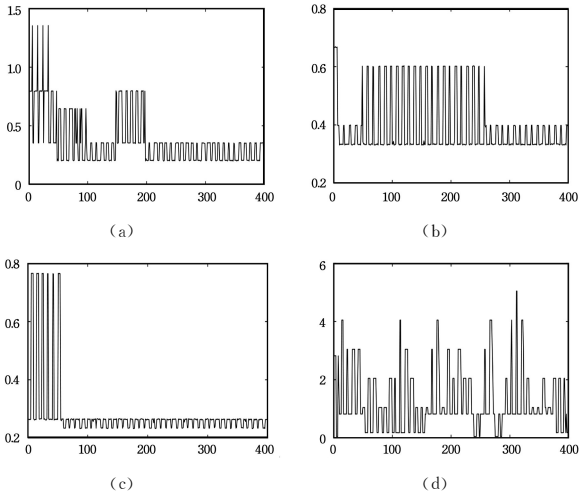

图 4 LR 吞吐率模型的预测误差

Fig. 4 Prediction error of LR throughput prediction model

由图 4 可知，4 个 RTs 上预测模型的预测误差平均值分别为 0.38，0.39，0.28，1.24，其中，模型在处理随机响应任务的 RT4 上预测误差最大，但在每次预测误差较大时，总能在约 3 个周期内进行调整，减小预测误差到 1 以内，因此模型的预测效果整体良好。同时，模型预测的吞吐率与其实际吞吐率间的 Pearson(皮尔逊)相关系数的值都大于 0.98，表示它们都有极高的线性相关关系，证明了算法选择线性回归方式求解 $\boldsymbol{w}$ 是正确的。

图 5 和表 2 给出了原系统和 ML-OBLS 算法调度后系统的 CPU 资源利用率情况。如图 5(a)所示，原系统中 RT1 的平均 CPU 利用率比系统中的其他节点普遍高出 40%以上，系统平均 CPU 利用率不到 50%。而使用分布式计算并采用本算法进行调度后的系统 CPU 利用率如图 5(b)所示，RT1 与其他节点的平均 CPU 利用率差值在 10%左右，系统平均 CPU 利用率比原系统提高了 23.78%。同时平均 CPU 方差降低了 65.41%。

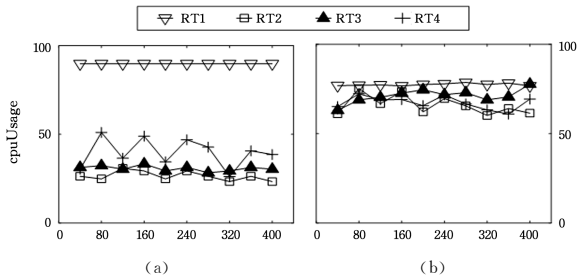

图 5 原系统和 ML-OBLS 调度后节点 CPU 利用率的曲线图

Fig. 5 CPU utilization graph of nodes on original system and system scheduled by ML-OBLS

表 2 原系统和 ML-OBLS 调度后节点的平均 CPU 利用率

Table 2 Average CPU utilization of nodes on original system and system scheduled by ML-OBLS

Algorithm	$\overline{cpu}_1$	$\overline{cpu}_2$	$\overline{cpu}_3$	$\overline{cpu}_4$	aver	$\overline{var(cpu)}$
原系统	89.79	26.47	30.72	39.64	46.66	1109.83
调度算法	77.42	65.95	71.13	67.28	70.44	383.92

tEncode 系统总吞吐率的对比结果如图 6 所示。除了 ML-OBLS 算法外，还加入了对比算法，该对比算法为 ML-OBLS 算法在负载调度时，只用调度方案 1 的版本。由图 6 可知，经本文算法调度后加密任务的系统平均总吞吐率从 5.98 组/ΔT 提升到 19.29 组/ΔT，比原系统提高了 225.97%，而对比算法只提高到 12.01 组/ΔT。由此可以看出，每次调度都根据 v' 比值来均衡节点负载的调度策略并不是最优的，其效果远不如用两种不同方案进行调度的 ML-OBLS 算法。

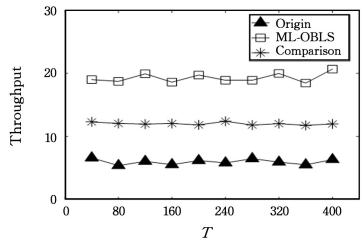

图 6 tEncode 系统总吞吐率的对比

Fig. 6 Comparison of total system throughput of tEncode

结束语 本文提出了基于机器学习的分布式星载 RTs 系统负载调度算法，通过线性回归正规方程建立任务吞吐率预测模型，根据源数据到达速率和任务未处理数据量的不同，采用按吞吐率比例均匀分配法或保证最少处理周期法来动态分配、转移节点间的数据。测试结果表明，使用本文算法后系统平均 CPU 利用率提高了 23.78%，节点平均 CPU 利用率方差降低了 65.41%，tEncode 任务系统总吞吐率提升了 225.97%，在保证负载均衡的前提下加快了任务的总处理速度。

但是，本文算法是在节点通信得到保证及被调度的任务只有一个的前提下实现的。而实际的分布式星载计算机系统的通信情况更为复杂，系统总线的通信率也是影响算法效果的重要因素。并且，在未来，一个 RT 上可能布置多个过载任务，调度算法需考虑更多的因素，如过载任务的优先级。因此，对系统总线通信情况的获取和评估，以及多任务数据处理调度将成为算法改进的重要方向。

参 考 文 献

[1] SAEED N, ELZANATY A, ALMORAD H, et al. CubeSat Communications: Recent Advances and Future Challenges[J]. IEEE Communication Survey & Tutorials, 2020, 22(3): 1839-1862.

[2] LI H W, WU Q, XU G, et al. Progress and Tendency of Space and Earth Integrated Network[J]. Science & Technology Review, 2016, 34(14): 95-106.

[3] HUANG C. Reliable Reconstruction Technology for On-board Computer Based on Loongson and FLASH[D]. Beijing: University of Chinese Academy of Sciences, 2017.

[4] LU S Q, LIANG H G, LIU D Y. Thoughts on the Status Quo and Development of Localized On-board Computer Technology [J]. Computer Knowledge and Technology, 2018, 6: 126-129.

[5] PATNI J C, ASWAL M S. Distributed Load Balancing Model for Grid Computing Environment[C]// 2015 1st International Conference on Next Generation Computing Technologies. 2015: 123-126.

[6] PENG T, HOFLINGER K, WEPS B, et al. A Component-Based Middleware for a Reliable Distributed and Reconfigurable Spacecraft Onboard Computer[C]// 2016 IEEE 35th Symposium on Reliable Distributed Systems. 2016: 337-342.

[7] REN J Y, SUN H Y, ZHANG L X, et al. Development status of space laser communication and new method of networking[J]. Laser & Infrared, 2019, 49(2): 143-150.

[8] LIU L D, QI D Y. An Independent Task Scheduling Algorithm in Heterogeneous Multi-core Processor Environment[C]// 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference. 2018: 142-146.

[9] ZHANG J, SUN S J, FAN H B, et al. Task Scheduling Algorithm in Heterogeneous Multi-core Processor with High Real-time Performance[J]. Computer Engineering, 2017, 43(5): 55-59.

[10] AN X, ZHANG Y, KANG A, et al. Machine learning based online mapping approach for heterogeneous multi-core processor system[J]. Journal of Computer Applications, 2019, 39(6): 1753-1759.

[11] IBM Cloud Education. Machine Learning Focuses on Applications that Learn from Experience and Improve their Decision-making or Predictive Accuracy over Time[EB/OL]. (2020-07-15)[2020-10-01]. https://www.ibm.com/cloud/learn/machine-learning.

[12] SAMIE F, BAUER L, HENKEL J. From Cloud Down to Things: An Overview of Machine Learning in Internet of Things [J]. IEEE Internet of Things Journal, 2019, 6(3): 4921-4934.

[13] NEMIROVSKY D, ARKOSE T, MARKOVIC N, et al. A machine learning approach for performance prediction and scheduling on heterogeneous CPUs[C]// Proceedings of the 2017 IEEE 29th International Symposium on Computer Architecture and High Performance Computing. Piscataway, NJ: IEEE, 2017: 121-128.

[14] MICOLET P J, SMITH A, DUBACH C. A machine learning approach to mapping streaming workloads to dynamic multicore processors[C]// LCTES 2016: Proceedings of the 2016 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems. New York: ACM, 2016: 113-122.

[15] ROTATION. Machine Learning—A Summary of Linear Regression[EB/OL]. (2010-01-19)[2020-10-01]. https://blog.csdn.net/fengxinlinux/article/details/86556584.

[16] XIE Y X, LI Y W, XIA Z J, et al. An Improved Forward Regression Variable Selection Algorithm for High-Dimensional Linear Regression Models[J]. IEEE Access, 2020, 8: 129032-129042.

[17] YUAN D M, PROUTIERE A, SHI G D. Distributed Online Linear Regressions [J]. Transactions on Information Theory, 2021, 67(1): 616-639.

[18] YU D F, LI H J, TANG H, et al. Dynamic Load Balancing Algorithm Design and Application based on Feedback[J]. Application Research of Computers, 2012, 29(2): 527-529.

[19] GAST N, IOANNIDIS S, LOISEAU P, et al. Linear Regression from Strategic Data Sources[J]. ACM Transactions on Economics and Computation, 2020, 8(2): 1-24.

[20] LIANG J B, ZHANG H H, JIANG C, et al. Research Progress of Task Offloading Based on Deep Reinforcement Learning in Mobile Edge Computing[J]. Computer Science, 2021, 48(7): 316-323.

TAN Shuang-jie, born in 1994, postgraduate. Her main research interests include distributed computing and embedded software.

LIN Bao-jun, born in 1963, Ph.D, professor, Ph.D supervisor. His main research interests include computer control technology and satellite overall.

（责任编辑：喻藜）