https://doi.org/10.1051/jnwpu/20244220319

一种用于星载虚拟化平台的任务容器调度算法

刘明轩[1], 郭博渊[2], 刘曦[2], 梁欣欣[2], 赵强龙[1], 杨晓峰[1], 谷建华[1]

(1.西北工业大学 计算机学院, 陕西 西安 710072; 2.西安微电子技术研究所, 陕西 西安 710065)

摘 要:星载虚拟化平台借助容器等轻量级虚拟化技术,将计算任务封装到容器中形成任务容器,从而实现资源的高效利用。然而,该平台的任务容器调度问题是一个亟需解决的难题。针对这一问题建立了一个基于非阻塞通信模式的可分容器任务多趟调度模型。在该基础上,提出了一种新的调度算法,旨在确定最佳的处理机调度顺序和调度趟数。该算法结合可分任务容器和多趟调度的概念,通过将任务分解为可执行的子任务,并在多个调度阶段中进行任务分配和处理机调度,从而优化调度顺序,提高整体处理效率。该算法是一种改进的遗传算法,在传统遗传算法的基础上添加了子种群隔离的优化策略,其核心思想是将种群划分策略引入算法过程,从而改善遗传算法的性能和效果。通过实验验证了该算法的有效性和收敛性,结果表明,该算法缩短了任务完成时间。

关 键 词:星载计算;虚拟化;容器;可分任务调度

中图分类号:TP3 **文献标志码**:A **文章编号**:1000-2758(2024)02-0319-09

星载虚拟化平台在航天领域的应用日益广泛。该平台搭载有多种高性能计算资源提供强大的算力,同时借助容器等轻量级虚拟化技术高效部署各类计算任务,从而实现资源的高效利用。

虽然星载虚拟化平台可以实现星载计算资源的高效利用,但是,星载虚拟化平台中的任务容器调度问题是一个亟需解决的难题。与通用虚拟化技术中的容器调度相比,一方面,星载计算环境通常具有严格的低延迟要求和资源限制,包括计算能力、内存、存储和能源等,需要考虑尽量减少任务的调度延迟,以满足实时性要求。另一方面,星载计算环境通常具有有限带宽的通信网络。在虚拟化调度中,需要考虑任务之间的通信需求,并合理规划任务的部署位置,以减少通信开销和延迟。

为了解决星载虚拟化平台中的容器任务调度问题,重点关注可分任务容器多趟调度[1-3]。可分任务容器多趟调度是指一组任务容器需要在一定时间执行完成,而每个任务容器可以被分成多个子任务容器,不同的子任务容器可以在不同的计算节点上并行执行。每个计算节点完成相应的子任务后,不需要将计算的结果回传到主节点进行整合。由于负载繁重,一次计算任务往往需要进行多趟调度才能完成。在航天领域中,此类调度被广泛应用于图像处理、数据分析和科学计算等任务。

可分任务容器多趟调度的基础是可分负载理论[4-5](divisible load theory, DLT),基本思想是计算负载可以被连续地划分为多个部分,这些部分可以在分布式系统的各个节点独立处理。可分任务容器多趟调度的核心问题是如何根据处理的数量和性能以及任务特性,确定最优的子任务划分方案,以实现最大的性能和效率。为了解决这一问题,需要对任务特性、处理性能、通信延迟等进行建模和分析,并采用优化算法寻找最优的子任务划分方案。现有的解决方案包括贪心算法、动态规划、遗传算法、模拟退火等方法。这些方法都可以用来求解任务容器的最优调度顺序、处理机数目、调度趟数等。研究高效的可分任务调度策略,可以使任务的完成时间最小化,对提高星载虚拟化平台的性能至关重要。

本文提出了一种可分任务容器多趟调度算法,旨在确定最佳的处理机调度顺序和调度趟数,以优化任务容器的执行,提高任务容器的处理效率。首先,针对星载分布式环境下可分多趟任务容器调度

收稿日期:2023-02-23

基金项目:陕西省重点研发计划(2023-ZDLGY-08)资助

作者简介:刘明轩(1997—),博士研究生

通信作者:谷建华(1965—),教授 e-mail:gujh@nwpu.edu.cn

问题,建立基于非阻塞通信模式的可分容器任务多趟调度模型。接着,假设处理机顺序给定,提出 2 种算法分别解决固定调度趟数确定处理机数目与固定处理机数目确定调度趟数这 2 个问题。综合上述 2 种算法提出了处理机顺序给定可分任务多趟调度算法,以确定最佳的处理机数目与调度趟数。之后,提出了基于子种群隔离的优化遗传算法,解决了多趟任务调度中处理机调度顺序问题。最后,通过实验来验证 2 种遗传算法的有效性。实验结果表明,相比于现有的调度算法,所提算法在任务完成时间方面有更好的性能。

1 问题描述

假设卫星搭载了一颗高分辨率摄像头,实时拍摄太空图像。对于源源不断的图像输入,该卫星需要实时执行图像处理任务,由于其计算资源和存储资源有限,为了充分利用资源,采用星载虚拟化平台来分布式处理这些任务。图 1 是一个典型的星载虚拟化平台的工作流程。任务程序的开发人员在完成程序开发后,将任务程序封装至容器镜像中,之后将这些任务容器镜像上传至星载虚拟化平台的镜像仓库。当指挥人员使用平台发起若干项计算任务时,星载计算集群的中心控制节点做出调度决策并通知各计算节点,从镜像仓库中拉取相应的任务容器镜像到本地。随后,各个计算节点将会启动任务容器来执行计算任务。

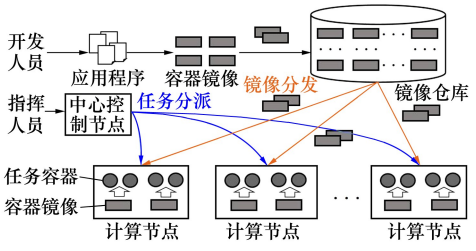

图 1 计算业务调度分派流程

在该星载虚拟化平台中,主处理节点(即处理机)是卫星上的主控制器,各个协处理节点是卫星上的其他处理单元(例如 GPU 和 NPU)或附加设备(例如 FPGA 等)。卫星主处理节点上存储着待处理的图像数据。主处理节点根据任务容器的大小,将其划分为独立的子任务容器,每个子任务容器包含一部分图像数据。根据预先设定的调度策略,主处理节点将这些子任务容器分派给各个协处理节点。每个协处理节点接收到一个子任务容器后,对其中的图像数据进行处理,比如图像去噪、图像增强或特征提取等操作。

主处理节点和协处理节点之间通过异构的通信链路进行数据传输,构成星形网络结构。这些链路可以是不同的传输介质,比如卫星内部的板载总线、网络连接或其他专用的通信通道。星形网络由一个主处理节点和多个协处理节点构成,其中各个协处理节点与主处理节点通过异构的通信链路相连。任务容器起初位于主处理节点上,主处理节点需将任务容器划分为独立的子任务容器,并按照调度策略将其分派给各个协处理节点执行计算任务。

对星载虚拟化平台的系统做如下假设:

1)任务容器可以被划分为任意大小的子任务容器,各个子容器任务之间相互独立,且子容器任务可以被调度到任意一个协处理节点上处理。

2)子任务容器在节点上的运行时间和在网络上的传输时间与容器大小呈正比。

3)使用非阻塞通信,即协处理节点收到子任务容器后立刻开始计算,无需等待全部子任务容器均传输完成。

由于星载虚拟化平台中处理机和网络的异构性,处理节点的计算速度、网络传输速度和启动开销各不相同。因此,在设计模型时需要考虑处理机的通信启动开销、任务传输时间、计算启动开销和任务计算时间。本研究采用了一种非阻塞通信模式,实现了通信和计算的充分重叠。图 2 展示了非阻塞可分任务多趟调度的过程,纵坐标为处理机的调度顺序,横坐标为任务的完成时间。各个子任务的通信开销为主处理节点 p_0 传输大小为 $\alpha_i V$ 的子任务给协处理节点 p_i 所需的时间,记为 $o_i + \alpha_i V z_i$。其中,o_i 表示链路 l_i 的启动时间,z_i 表示链路 l_i 传输单位任务所需要的时间。子任务的计算开销表示处理节点 p_i 处理大小为 $\alpha_i V$ 的子任务所需时间,记作 $s_i + \alpha_i V w_i$,其中,s_i 表示计算启动时间,w_i 表示节点处理单位任务的时间。

实验证明,当所有协处理节点同时完成对子任务处理时,可以实现任务执行时间最短[4]。因此,为了确保系统中的各个协处理节点能够在相同的时间内完成子任务,最后一次调度策略与之前的调度策略不同。在这项研究中,可分任务的多趟调度过程被划分为 2 个阶段:内部调度和最后一轮调度。

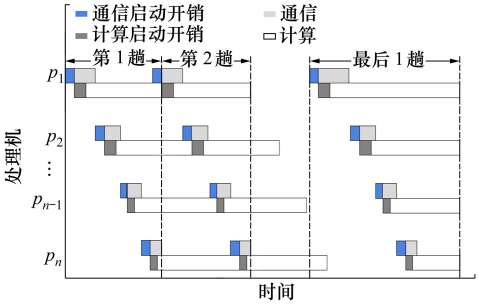

图 2　非阻塞可分任务多趟调度模型

内部调度指除最后一轮调度外的所有调度,每一轮内部调度中,主处理节点都使用相同的任务划分方式将计算任务划分并分配给协处理节点。最后一轮调度则采取新的调度策略来处理,在保证各个从处理节点同时完成计算的前提下,对最后一轮任务进行划分与调度。已有研究建议内部调度的任务规模相等[4],因此在本研究中假设多趟调度中每趟任务规模相等。

2　可分任务容器多趟调度模型

在航天领域中,可分任务容器多趟调度模型是指将数据处理、图像分析、模拟计算等航天计算任务划分为可分的子任务。这些子任务基于容器化技术,将子任务和相关的计算资源、数据打包成独立的容器镜像,并使用多趟调度策略将这些子任务容器镜像分派到不同的计算节点上进行实例化执行。可分任务容器多趟调度模型和相关的符号定义如表 1 所示。

表 1　符号定义

符号	含义
W	任务的规模
V	每趟调度中分配给处理机的任务规模
m	内部调度的趟数
n	处理机数量
σ	处理机调度顺序
$\alpha_i V$	每趟内部调度中分配给处理机 p_i 的任务规模
$\beta_i V$	最后一趟调度中分配给处理机 p_i 的任务规模
w_i	处理机 p_i 计算单位任务的时间
z_i	链路 l_i 传输单位任务的时间
s_i	处理机 p_i 的计算启动开销
o_i	处理机 p_i 的通信开销
$T(W)$	任务总完成时间

2.1　内部调度

在内部调度中,为了保证协处理机处理完分配的子任务时,主处理节点恰好要向该节点发送下一趟的子任务,本研究假设所有协处理机的通信时间与计算时间之和在每一趟内部调度中都是相同的。基于这一理论确定分配给每个协处理机的子任务的规模。根据该理论可以得出

$$o_{\sigma_1}+s_{\sigma_1}+\alpha_{\sigma_1}Vw_{\sigma_1}=o_{\sigma_2}+s_{\sigma_2}+\alpha_{\sigma_2}Vw_{\sigma_2}=\cdots=o_{\sigma_n}+s_{\sigma_n}+\alpha_{\sigma_n}Vw_{\sigma_n} \tag{1}$$

式中:$o_{\sigma_i}+s_{\sigma_i}+\alpha_{\sigma_i}Vw_{\sigma_i}$ 表示调度顺序为 σ 时第 i 台主机处理分配的子任务的时间,α_i 为主处理机 p_0 分配给协处理机 p_i 的任务规模,因此有

$$\alpha_{\sigma_1}+\alpha_{\sigma_2}+\cdots+\alpha_{\sigma_n}=1 \tag{2}$$

根据(1)~(2)式可得

$$\alpha_{\sigma_1}+\frac{\alpha_{\sigma_1}Vw_{\sigma_1}+o_{\sigma_1}-o_{\sigma_2}+s_{\sigma_1}-s_{\sigma_2}}{Vw_{\sigma_2}}+\frac{\alpha_{\sigma_1}Vw_{\sigma_1}+o_{\sigma_1}-o_{\sigma_3}+s_{\sigma_1}-s_{\sigma_3}}{Vw_{\sigma_3}}+\cdots+\frac{\alpha_{\sigma_1}Vw_{\sigma_1}+o_{\sigma_1}-o_{\sigma_n}+s_{\sigma_1}-s_{\sigma_n}}{Vw_{\sigma_n}}=1 \tag{3}$$

为方便表示,定义变量 Δ_{σ_i} 与 Φ_{σ_i} 为

$$\Delta_{\sigma_i}=\frac{w_{\sigma_1}}{w_{\sigma_i}},\quad \Phi_{\sigma_i}=\frac{o_{\sigma_1}+s_{\sigma_1}-o_{\sigma_i}-s_{\sigma_i}}{w_{\sigma_i}} \tag{4}$$

将(4)式代入(3)式得

$$\alpha_{\sigma_1}\left[1+\sum_{i=2}^{n}\Delta_{\sigma_i}\right]+\frac{1}{V}\sum_{i=2}^{n}\Phi_{\sigma_i}=1 \tag{5}$$

综合(1)~(5)式可得到节点在每一趟内部调度中分配的子任务大小

$$\begin{cases}\alpha_{\sigma_1}=\dfrac{1-\dfrac{1}{V}\displaystyle\sum_{i=2}^{n}\Phi_{\sigma_i}}{1+\displaystyle\sum_{i=2}^{n}\Delta_{\sigma_i}} & i=1,2,\cdots,n\\ \alpha_{\sigma_i}=\alpha_{\sigma_1}\Delta_{\sigma_i}+\dfrac{\Phi_{\sigma_i}}{V}\end{cases} \tag{6}$$

2.2　最后一趟调度

在最后一趟调度过程中,要求所有协处理机同时完成子任务的处理,这样可以确保任务的总计算时间最小化。根据图 2 的调度过程结合非阻塞通信模式特点可以得到调度顺序中第一个处理机 p_1 完成最后一趟调度所用时间为

$$T(\beta_{\sigma_1})=o_{\sigma_1}+s_{\sigma_1}+w_{\sigma_1}\beta_{\sigma_1}V \tag{7}$$

式中:β_i 表示最后一趟调度中主处理机 p_0 分配给协

处理机 p_i 的任务规模。同时也可以得到处理机 p_i 最后一趟调度的任务完成时间。处理机最后一趟调度的任务完成时间等于前 $i-1$ 个处理机最后一趟调度的任务传输时间加上最后一趟调度的任务处理时间。

$$T(\beta_{\sigma_i}) = \sum_{j=1}^{i-1} (o_{\sigma_j} + z_{\sigma_j}\beta_{\sigma_j}V) + o_{\sigma_i} + s_{\sigma_i} + w_{\sigma_i}\beta_{\sigma_i}V \tag{8}$$

由于最后一趟调度中所有协处理机的任务执行时间相等,即 $T(\beta_{\sigma_i}) = T(\beta_{\sigma_{i+1}})$,于是可以得到

$$s_{\sigma_i} + w_{\sigma_i}\beta_{\sigma_i}V = z_{\sigma_i}\beta_{\sigma_i}V + o_{\sigma_{i+1}} + s_{\sigma_{i+1}} + w_{\sigma_{i+1}}\beta_{\sigma_{i+1}}V, \quad i = 1,2,\cdots,n-1 \tag{9}$$

为了简化模型表示,定义变量 $\delta_{\sigma_{i+1}}$ 和 $\varepsilon_{\sigma_{i+1}}$ 为

$$\delta_{\sigma_{i+1}} = \frac{s_{\sigma_i} - (s_{\sigma_{i+1}} + o_{\sigma_{i+1}})}{w_{\sigma_{i+1}}} \quad \varepsilon_{\sigma_{i+1}} = \frac{w_{\sigma_i} - z_{\sigma_i}}{w_{\sigma_{i+1}}} \tag{10}$$

将变量 $\delta_{\sigma_{i+1}}$ 和 $\varepsilon_{\sigma_{i+1}}$ 代入(9)式并化简可得

$$\beta_{\sigma_{i+1}} = \frac{1}{V}\delta_{\sigma_{i+1}} + \varepsilon_{\sigma_{i+1}}\beta_{\sigma_i}, \quad i = 1,2,\cdots,n-1 \tag{11}$$

定义变量 E_{σ_i} 和 Γ_{σ_i} 为

$$E_{\sigma_i} = \prod_{j=2}^{i}\varepsilon_{\sigma_j}, \quad \Gamma_{\sigma_i} = \sum_{j=2}^{i}\left(\delta_{\sigma_i}\prod_{k=j+1}^{i}\varepsilon_{\sigma_k}\right) \tag{12}$$

由于 $\sum_{i=1}^{n}\beta_{\sigma_i} = 1$,递归求解(11)式可得处理机 p_i 在最后一趟调度中的分配的任务规模

$$\begin{cases} \beta_{\sigma_1} = \dfrac{1 - \dfrac{1}{V}\sum_{i=2}^{n}\Gamma_{\sigma_i}}{1 + \sum_{i=2}^{n}E_{\sigma_i}} \\ \beta_{\sigma_i} = E_{\sigma_i}\beta_{\sigma_i} + \dfrac{1}{V}\Gamma_{\sigma_i} \end{cases} \quad i = 2,\cdots,n \tag{13}$$

至此,可以将任务的总完成时间表示成 $T(W)$。

$$T(W) = m(o_{\sigma_1} + s_{\sigma_1} + \alpha_{\sigma_1}Vw_{\sigma_1}) + o_{\sigma_1} + s_{\sigma_1} + \beta_{\sigma_1}Vw_{\sigma_1} \tag{14}$$

2.3 问题模型

可分任务调度的核心目标是使任务完成时间最短。从(14)式中可以看出,可分任务的完成时间取决于参数 α,β,m,n 和 $\boldsymbol{\sigma}$。完成时间以及各节点每趟调度分配的子任务规模最终依赖于调度趟数 m、参与计算的处理机数目 n 以及处理机调度顺序 $\boldsymbol{\sigma} = \{p_1,p_2,\cdots,p_n\}$。于是可分任务调度问题转换为求解最佳的 m,n 以及 $\boldsymbol{\sigma}$,使得可分任务的完成时间最短。因此,本研究为多趟可分任务调度建立优化模型如(15)式所示。

$$\begin{aligned} &\min_{m,n,\alpha,\beta,\boldsymbol{\sigma}} T(W) = [m(o_{\sigma_1} + s_{\sigma_1} + \alpha_{\sigma_1}Vw_{\sigma_1}) + o_{\sigma_1} + s_{\sigma_1} + \beta_{\sigma_1}Vw_{\sigma_1}] \\ &\text{s.t.} \quad (1)\ \alpha_{\sigma_i} > 0, \quad i = 1,2,\cdots,n; \\ &\qquad (2)\ \beta_{\sigma_i} > 0, \quad i = 1,2,\cdots,n; \\ &V = \frac{W}{m+1} \\ &\begin{cases} \alpha_{\sigma_1} = \dfrac{1 - \dfrac{1}{V}\sum_{i=2}^{n}\Phi_{\sigma_i}}{1 + \sum_{i=2}^{n}\Delta_{\sigma_i}} \\ \alpha_{\sigma_i} = \alpha_{\sigma_1}\Delta_{\sigma_i} + \dfrac{\Phi_{\sigma_i}}{V} \\ \Delta_{\sigma_i} = \dfrac{w_{\sigma_1}}{w_{\sigma_i}} \\ \Phi_{\sigma_i} = \dfrac{o_{\sigma_1} + s_{\sigma_1} - o_{\sigma_i} - s_{\sigma_i}}{w_{\sigma_i}} \end{cases} \\ &\begin{cases} \beta_{\sigma_1} = \dfrac{1 - \dfrac{1}{V}\sum_{i=2}^{n}\Gamma_{\sigma_i}}{1 + \sum_{i=2}^{n}E_{\sigma_i}} \\ \beta_{\sigma_i} = E_{\sigma_i}\beta_{\sigma_1} + \dfrac{1}{V}\Gamma_{\sigma_i} \\ \Gamma_{\sigma_i} = \sum_{j=2}^{i}\left(\dfrac{s_{\sigma_{j-1}} - (s_{\sigma_j} + o_{\sigma_j})}{w_{\sigma_j}}\prod_{k=j+1}^{i}\dfrac{w_{\sigma_{k-1}} - z_{\sigma_{k-1}}}{w_{\sigma_k}}\right) \\ E_{\sigma_i} = \prod_{j=2}^{i}\dfrac{w_{\sigma_{j-1}} - z_{\sigma_{j-1}}}{w_{\sigma_j}}, \end{cases} \end{aligned} \tag{15}$$

3 顺序给定的可分任务容器多趟调度

本节假设处理机调度顺序给定,求得使任务的完成时间最少的处理机数目 n 与内部调度趟数 m。根据已有研究[6],可分任务多趟调度模型存在如下 2 条性质:

性质 1 在问题模型有可行解的前提下,随着处理机数目 n 的增加,任务完成时间 T 单调递减。

性质 2 在问题模型存在可行解的前提下,任务的完成时间 T 随着内部调度趟数 m 的增加呈现先递减后递增的变化趋势,并且当 $(m+1)(m+2) \leqslant w_1W(b-d)/(bd\lambda)$ 时,一定有 $T(m,n) \geqslant T(m+1,$

$n)$,否则 $T(m,n) < T(m+1,n)$。其中,$a = \sum_{i=2}^{n} \Phi_i$,$b = 1 + \sum_{i=2}^{n} \Delta_i$,$c = \sum_{i=2}^{n} \Gamma_i$,$d = 1 + \sum_{i=2}^{n} E_i$,$\lambda = s_1 - \frac{w_1 a}{b}$。

3.1　调度趟数给定时的最优处理机数目求解算法

需要指出的是,由于问题模型中每个节点都有通信启动开销与计算启动开销,当选择参与调度的处理机数目较大而任务规模较小时,会存在任务完成时间小于零的情况,即建立的问题模型没有可行解。因此在求解最佳处理机数目时,必须保证模型有可行解。

由性质 1 可得,在模型有可行解的基础之上,当固定内部调度趟数 m 时,随着参与计算的处理机数目增多,任务完成时间逐渐减小。可以利用这条性质求解最佳的参与计算的处理机数目。借鉴二分查找算法的思想,每次判断区间中点的元素是否满足使模型有可行解,若满足,将问题区间一分为二,在中点右侧的区间内继续进行二分搜索;否则,在中点左侧的区间内继续进行二分搜索,直到找到使问题有可行解的最大处理机数目 n^*。以参考二分查找思想设计的调度趟数给定时的最优处理机数目求解算法的伪代码如下所示。

算法 1:调度趟数给定时最优处理机数目求解

输入:任务规模 W,处理机数目上限 n_{upper},调度趟数 m

输出:最优处理机数目 n^*

```
step1: n_lower = 2
step2: if T(m, n_upper) > 0
           goto step4
step3: while n_lower ≤ n_upper do
           n_mid = (n_lower + n_upper)/2
           if T(m, n_mid) > 0
               n_lower = n_mid + 1
           else
               n_upper = n_mid - 1
           goto step3
       end while
step4: n* = n_upper
step5: return n*
```

3.2　处理机数目给定时的最优调度趟数求解算法

根据性质 2 的描述,为了找到任务完成时间最短的最优调度趟数,提出了算法 2 进行求解。

算法 2:处理机数目给定时的最优调度趟数求解

输入:任务规模 W,处理机数目 n,调度趟数下限 m_{lower}

输出:最优调度趟数 m^*

```
step1: m' = (√(1 + 4w_1 W(b - d)bdλ) - 3)/2
       m_1 = ⌈m'⌉, m_2 = ⌊m'⌋
       if 0 < T(m_1, n) < T(m_2, n)
          M = m_1
          goto step4
       else if 0 < T(m_2, n) < T(m_1, n) or
               T(m_1, n) < 0 < T(m_2, n)
          M = m_2
          goto step4
       else if T(m_1, n) < T(m_2, n) < 0
          m_upper = m_2
          goto step2
step2: while m_lower ≤ m_upper do
           m_mid = (m_lower + m_upper)/2
           if T(m_mid, n) > 0
              m_lower = m_mid
           else
              m_upper = m_mid - 1
       end while
step3: m* = m_upper
step4: return m*
```

3.3　调度顺序给定的可分任务多趟调度算法

根据算法 1 可确定在调度趟数固定时的最佳处理机数目,而算法 2 则可以在处理数目给定时求得最佳调度趟数。结合算法 1 与算法 2,给出处理机顺序固定时的可分任务多趟调度算法 3。

算法 3:处理机顺序固定的可分任务多趟调度

输入:任务规模 W,处理机总数目 N,处理机调度顺序 $\boldsymbol{\sigma}$

输出:最优调度趟数 m^*,最优处理机数目 n^*,最短任务完成时间 $T(W)$

```
step1: m_lower = 1, n_lower = 1, n_upper = N
step2: n_upper = 算法 1(m_lower)
       m_upper = 算法 2(n_upper)
       if T(m_upper, n_upper) > 0
          m* = m_upper
          n* = n_upper
          T(W) = T(m_upper, n_upper)
```

```
        goto step4
    else
        goto step3
step3: while n_lower ≤ n_upper do
        n_upper = n_upper − 1
        m_upper = 算法 2(n_upper)
        if T(m_upper, n_upper) > 0
            m* = m_upper
            n* = n_upper
            T(W) = T(m_upper, n_upper)
            goto step4
        else
            goto step3
    end while
step4: return n*, m*, T(W)
```

4 最优顺序的可分任务容器多趟调度

第 3 节研究了一种处理机顺序固定的可分任务多趟调度算法。在固定处理机调度顺序的前提下，该算法能够找到最佳的调度趟数和最佳处理机数目，使任务的总完成时间最小化。但是在实际任务调度过程中各个参与调度的节点的调度顺序也是影响任务总完成时间的重要因素。因此，以任务总完成时间最短为目标，以遗传算法为基础，提出一种分布式异构环境下最优调度顺序的可分任务周期性多趟调度算法。

4.1 遗传算法的不足

传统的遗传算法往往存在着早熟的问题。根据模式定理[7]和积木块假说[8]，遗传算法的进化原理是基因序列中不同位置的小积木不断组合，最终可能组合成最优解。然而，由于种群规模的有限性，经过多次迭代后，整个种群会被具有指数级增长的高平均适应度的模式所占据，而低适应度的模式会被迅速淘汰，导致种群中的模式数量逐渐减少。这种情况下，遗传算法会逐渐收敛并趋向于局部最优解。

4.2 最优顺序的可分任务多趟调度算法

为了应对遗传算法的早熟问题，本研究提出了一种基于子种群隔离的优化遗传算法。具体而言，包括如下步骤：

1）编码与初始化

在遗传算法中，问题的解被编码成一个个体，也称为染色体或基因组。编码的目的是将问题的解转换为计算机可以处理的形式。该算法的目标是寻找最优的处理机数量 n^* 和内部调度趟数 m^* 以及计算节点的调度顺序 $\boldsymbol{\sigma}$，使得任务的完成时间最短。因此设计的编码需要包含这 3 项内容。

遗传算法在开始进化前必须有一个初始种群，即必须完成第一代种群的初始化。初始化算法应尽量做到简单且结果应均匀分布。本研究采用洗牌算法来初始化第一代种群，将 1 到 N 的递增序列按照洗牌算法随机排序，生成一串随机的编码作为种群中的一个个体。洗牌算法拥有较低的时间与空间复杂度，且能够保证初始种群的个体比较均匀地分布在解空间中。

2）适应度评估

适应度用于评价个体的优劣程度，适应度越高个体越好，适应度越低则个体越差。按适应度大小对个体进行选择，确保适应度高的个体更有机会繁殖后代，从而传递优良特性。可分任务调度的主要目标是尽量减少任务完成时间，因此需要将任务完成时间最小化的目标映射到适应度函数中。

3）隔离策略

基于子种群隔离的优化遗传算法的关键步骤是隔离操作。隔离操作借鉴了生物学中地理隔离的特点，即地理环境限制了同一物种的自由迁移和交配，从而阻止了基因的交流，导致不同地理区域内的物种进化出独特的特征。

为了解决遗传算法的不足，在地理隔离原理启发下，本研究为了降低重复性，将种群分成多个子种群，并独立地在每个子种群中进行复制、选择和交叉操作，确保每个子种群的进化过程互不干扰。每隔一段时间，重新组合并分配所有个体，形成新的、相互隔离的子种群，并循环执行这一过程。这个步骤的目的是在进化的不同阶段中，打破原有子种群之间的关联性，引入更多的随机性和多样性，以促进种群的探索能力和避免过早陷入局部最优解。

4）交叉算子

交叉算子是遗传算法中生成新个体的主要操作之一，它通过以一定概率交换 2 个个体之间部分染色体的信息来产生新的个体。在本研究中采用了单点交叉的方式，具体来说，即随机选择一个交叉点，然后将 2 条配对染色体在交叉点前后的信息进行互换。

需要注意的是，交叉后的染色体存在重复的元

素,这与其代表的处理机调度顺序这一物理意义存在冲突。因此需要按照匹配原则将重复的元素用缺失的元素替换掉,保证交叉后的染色体序列可以表示一组处理机调度顺序。

5) 变异算子

在遗传算法的进化过程中,有时会出现优势个体过度复制的情况。为了避免过早收敛并增强算法的局部搜索能力,变异操作被引入以保持群体的多样性。常见的变异方式包括基本位变异、均匀变异和非均匀变异等。在本研究中,采用了基本位变异。具体地,随机选择染色体中的一个位置,然后再随机选择另一个位置,并交换这两个位置上的基因,从而生成一个新的个体。这种变异操作有助于引入新的基因组合,增加群体的多样性。

6) 选择算子

在遗传算法中,选择算子是指根据染色体适应度来选择优秀的个体,使其有更高的概率被选中作为下一代的父代。选择算子的作用是增加群体中优秀个体的比例,从而逐步优化群体。本研究采用的是轮盘赌选择算子。轮盘赌选择通常被称为适应度比例选择,是一种根据染色体适应度来选择优秀个体的算子。轮盘赌选择的原理是将每个个体的选择概率与一个轮盘上的扇区相对应,然后根据扇区大小进行随机选择。

具体来说,轮盘赌选择算子的实现步骤为:①对每个个体进行适应度值计算,并将适应度值转换为选择概率。这种转换可以采用比例缩放或指数缩放等方法来实现。②将所有个体的选择概率分别分配到轮盘上,通常是按照适应度值大小为权重分配,适应度值大的个体分配扇区更宽。③随机生成一个 0 到 1 之间的随机数,然后将其与轮盘上的扇区相比较,确定被选择的个体。重复上述步骤直到选出足够数量的个体作为下一代的父代。

7) 基于子种群隔离的优化遗传算法

求解最优调度顺序的可分任务多趟调度算法的核心即基于子种群隔离的优化遗传算法 SIAG(sub-population isolation genetic algorithm)。算法详细内容如下所示。

算法 4:基于子种群隔离的优化遗传算法

输入:种群大小 S_{pop},处理机数目 N,交叉概率 P_c,变异概率 P_m,进化代数 T,异构处理机参数 o_i,s_i,z_i,w_i,任务规模 W,划分子种群个数 S_{sub}。

输出:最优个体,包括最佳处理机调度顺序 $\boldsymbol{\sigma}$、处理机数目 n、内部调度趟数 m 与任务的最短完成时间 $T(W)$。

step1: 初始化:使用 $N+2$ 维的整数向量$(m,n,c_1,c_2,\cdots,c_N)$ 来编码个体,生成初代种群 $P(t)$,设置进化代数 $t=0$。

step2: while $t < T$

for $i = 1,2,\cdots,S_{sub}$

(1) 选择:计算子种群中所有子体的适应度,根据最优保护策略从子代个体集合中选出适应度最大的个体直接保留到下一代子种群中,然后使用轮盘赌选择策略从子代个体集合选择$(S_{pop}/S_{sub}-1)$ 个个体加入新的子种群中。

(2) 交叉:对子种群中除最佳个体外的每个个体,每次以交叉概率 P_c 选择 2 个个体,进行交叉操作。

(3) 变异:对子种群中除最佳个体外的每个个体,以变异概率 P_m 对其按变异算子进行变异操作,使用变异产生的新个体替换原来的个体。

end for

if t mod 20 = 0

采取洗牌算法将种群 $P(t+1)$ 打乱,使各个子种群的个体混合,形成 S_{sub} 个新的子种群。

$t = t+1$

end while

5　实验评估

5.1　实验环境

本次实验的实验环境如下:设备的处理为 Intel(R) Core(TM) i7 8700 CPU,共有 12 个 CPU 核心,内存大小为 16 GB,主频大小为 3.20 GHz。本实验基于模拟仿真系统生成 20 个计算节点的性能参数。实验中遗传算法的详细参数为:种群大小 $S_{pop}=90$,交叉概率 $P_c=0.8$,变异概率 $P_m=0.02$,子种群个数 $S_{sub}=3$,终止条件为进化代数 $T=1\ 000$。迭代次数设置为 1 000 主要目的是保证 2 种遗传算法都稳定地收敛到最优解,方便对比 2 种算法的进化趋势。

5.2　实验结果

1) 性质验证

本研究设计实现一组实验来验证在处理机数目确定的情况下,随着调度趟数 $m+1$ 的增加,总完成时间先递减再递增。为了避免随机性误差对结果的影响,本实验随机生成一组调度序列作为处理机调

度顺序,且通过多组实验,分别展示当处理机数量固定为 12,14,16,18,20 的情况下,内部的调度趟数与可分任务的完成时间的关系,具体结果如图 3 所示。

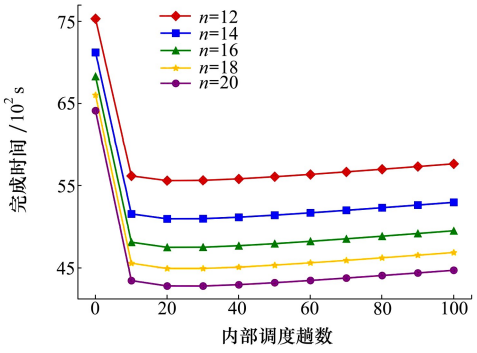

图 3 调度趟数与完成时间的关系

实验结果显示,在固定处理机数目 n 的前提下,内部调度趟数在 0~30 单调递减,30~100 单调递增。通过实验的验证,充分证实了第 3 节中性质 2 的正确性。

2)调度算法对比

实验中调度的任务为目标检测程序,以图像数量与图像分辨率的乘积代表任务规模,输入的图像数据集为航天器真实拍摄所得。图 4 展示了任务规模从 50 000~500 000 时 4 种算法所获得的任务完成时间,IZ 表示 Amins Alg[5] 算法采取传输时间递增的调度顺序,即以网络传输单位任务所消耗的时间递增的顺序作为处理机的调度顺序,IW 表示 Amins Alg 算法采取计算时间递增的调度顺序,即按照节点处理单位任务所需的时间递增的顺序作为处理机调度顺序。由图可得,在不同的任务规模下,标准遗传算法 GA 算法与 SIAG 算法求得的任务完成时间均优于 Amins Alg 算法 2 种调度顺序所得调度结果。GA 与 SIAG 求得的完成时间基本相同。

本节设计实验对比了标准遗传算法与基于子种群隔离的遗传算法的进化趋势。为了排除随机误差的影响,采用了相同的输入和初始种群。将算法的终止条件设置为 1 000 次迭代后停止,以确保算法有足够的进化时间来达到最终的收敛状态。2 种算法进化趋势如图 5 所示。

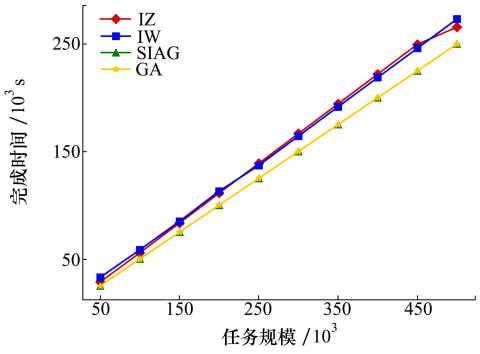

图 4 不同算法任务完成时间对比图

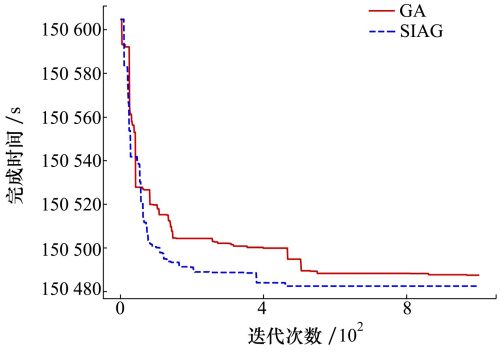

图 5 2 种算法迭代进化趋势图

从图 5 可以清晰地看到在 0 代至 210 代期间,基于子种群隔离的遗传算法呈现连续的进化趋势。在第 470 代时,该算法达到了一个稳定的解,最终收敛到 150 483 s。相比之下,标准的遗传算法进化速度较慢。从 150 代开始,它经历了多次进化的平台期,直到 870 代才稳定收敛到一个较低的值,最终解收敛到 150 488 s,与基于子种群隔离的遗传算法接近。

6 结 论

星载虚拟化平台的任务容器调度问题是一个亟需解决的难题。本文针对星载分布式环境下的可分多趟任务容器调度问题,建立了一个基于非阻塞通信模式的可分容器任务多趟调度模型,提出了基于子种群隔离的优化遗传算法问题解决了多趟任务调度问题中处理机调度顺序问题。实验验证了 2 种遗传算法的有效性。

参考文献:

[1] KAZEMI M, GHANBARI S, KAZEMI M. Divisible load framework and close form for scheduling in fog computing systems[C]//Proceedings of the Fourth International Conference on Soft Computing and Data Mining, Melaka, Malaysia, 2020: 22-23

[2] NIKBAKHT AALI S, BAGHERZADEH N. Divisible load scheduling of image processing applications on the heterogeneous star and tree networks using a new genetic algorithm[J]. Concurrency and Computation: Practice and Experience, 2020, 32(10): e5498

[3] ROBERTAZZI T G, SHI L, ROBERTAZZI T G, et al. Divisible loads and parallel processing//Networking and Computation: Technology, Modeling and Performance[M]. Springer, 2020: 101-137

[4] SHOKRIPOUR A, OTHMAN M, IBRAHIM H, et al. New method for scheduling heterogeneous multi-installment systems[J]. Future Generation Computer Systems, 2012, 28(8): 1205-1216

[5] BHARADWAJ V, GHOSE D, MANI V. Multi-installment load distribution in tree networks with delays[J]. IEEE Trans on Aerospace and Electronic Systems, 1995, 31(2): 555-567

[6] WANG X, VEERAVALLI B. Performance characterization on handling large-scale partitionable workloads on heterogeneous networked compute platforms[J]. IEEE Trans on Parallel and Distributed Systems, 2017, 28(10): 2925-2938

[7] HAMMERMAN N, GOLDBERG R. Algorithms to improve the convergence of a genetic algorithm with a finite state machine genome//Practical Handbook of Genetic Algorithms[M]. Boca Raton: CRC press, 2019: 119-238

[8] BAI W, REN J, LI T. Modified genetic optimization-based locally weighted learning identification modeling of ship maneuvering with full scale trial[J]. Future generation computer systems, 2019, 93: 1036-1045

A task container scheduling algorithm for spaceborne virtualization platform

LIU Mingxuan[1], GUO Boyuan[2], LIU Xi[2], LIANG Xinxin[2], ZHAO Qianglong[1], YANG Xiaofeng[1], GU Jianhua[1]

(1.School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China;
2.Xi'an Institute of Microelectronics Technology, Xi'an 710065, China)

Abstract: With the help of lightweight virtualization technology such as containers, the spaceborne virtualization platform encapsulates computing tasks into containers to form task containers, so as to achieve efficient utilization of resources. However, the platform's task container scheduling problem is a difficult problem that needs to be solved urgently. In this paper, we aim at this problem by establishing a multi-pass scheduling model for separable container tasks based on non-blocking communication mode. On the basis of this model, we propose a new scheduling algorithm, aiming at determining the optimal processor scheduling sequence and scheduling times. The algorithm combines the concept of divisible task container and multi-pass scheduling. By decomposing the task into executable subtasks, and performing task allocation and processor scheduling in multiple scheduling stages, it can optimize the scheduling order to improve the overall processing efficiency. This algorithm is an improved genetic algorithm that adds the optimization strategy of subpopulation isolation to the traditional genetic algorithm. Its core idea is to improve the performance and effect of the genetic algorithm by introducing the population division strategy into the algorithm process. We verify the effectiveness and convergence of the algorithm through experiments, and the experimental results show that the algorithm makes the task have less completion time.

Keywords: spaceborne computing; virtualization; container; separable task scheduling

引用格式: *刘明轩, 郭博渊, 刘曦, 等. 一种用于星载虚拟化平台的任务容器调度算法[J]. 西北工业大学学报, 2024, 42(2): 319-327*

LIU Mingxuan, GUO Boyuan, LIU Xi, et al. A task container scheduling algorithm for spaceborne virtualization platform [J]. Journal of Northwestern Polytechnical University, 2024, 42(2): 319-327 (in Chinese)