云环境下面向跨域作业的调度方法

李 焱[1],郑亚松[1],李 婧[2],朱春鸽[1],刘欣然[1]
(1. 国家计算机网络应急技术处理协调中心,北京 100029;2. 中国盲文出版社,北京 100050)

摘 要: 云环境下,因数据局部性或是任务对资源的特殊偏好,一个作业所包含的任务往往需要在不同的数据中心局点上运行,此类作业称为跨域作业. 跨域作业的完成时间取决于最慢任务的执行效率,即存在木桶效应. 针对各域资源能力异构条件下不合理的调度策略导致跨域作业执行时间跨度过长的问题,本文提出一种面向跨域作业的启发式调度方法 MIN-Max-Min,优先选择期望完成时间最短的作业执行. 通过实验表明,与先来先服务的策略相比,该方法能将跨域作业平均执行时间跨度减少 40% 以上.

关键词: 云计算; 跨域数据中心; 跨域作业
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2017)10-2416-09
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.10.015

A Scheduling Strategy for Jobs Across Geo-Distributed Datacenters in Cloud Computing

LI Yan[1], ZHENG Ya-song[1], LI Jing[2], ZHU Chun-ge[1], LIU Xin-ran[1]
(1. *National Computer Network Emergency Response Technical Coordination Center, Beijing* 100029, *China*;
2. *China Braille Press, Beijing* 100050, *China*)

Abstract: In cloud computing, tasks in a job often need to run on different datacenters due to the input data locality or special preference for resources, that is, the job runs across geo-distributed sites. The different tasks in a job have to be scheduled in different domain (data center) to execute for their personalization requirements, so the job completion time depends on the slowest task, which is called "barrel effect". As geo-distributed scheduling strategy without regard to heterogeneous resources leads too long execution time span, this dissertation proposes an optimization strategy for geo-distributed scheduling named MIN-Max-Min. The strategy gives priority to select the expectation shortest completion job to execute by heuristic rule. Experiments show that compared with first come first service strategy, the strategy can reduce cross domain average execution time span to less than 40% under the simulation load.

Key words: cloud computing; geo-distributed data centers; jobs across geo-distributed data centers

1 引言

云服务模式能够缩短上层应用开发周期、节省系统部署成本,越来越多的企业基于该模式构建信息系统. 例如,在网络搜索和用户广告定制、产品推荐、用户行为分析等商业智能化应用领域,处理数据量大、时间复杂度高的计算通常被抽象为一个个数据密集型作业(Data Intensive Jobs),运行在云计算框架(如 Hadoop①、Spark[1]、Dryad[2]等)之上,并成为云数据中心重要负载之一[3]. 这些作业往往由大量的任务(Task)组成,每个任务负责处理一部分数据,并在云资源节点上高度并发执行,以满足业务分析的时间需求.

然而,在某些应用场景下,需分析的数据高度分散甚至跨越多个数据中心局点②. 传统的作业集中执行模式下,需先将该作业所需的分散数据集中在某数据中心局点,并由部署在该局点的大量任务并发执行以完成数据处理. 然而,随着数据规模的日益增大,作业单局点集中执行模式面临着越来越大的跨域数据传输压力,并使得作业完成时间过长[4~7]. 而且,某些数据通常会因为信息安全或隐私保护的原因不允许离开数据产生地或是特定数据中心局点[4,5,8]. 因此,作业的跨域分布执行模式逐渐受到青睐,作业中任务执行位置取决

收稿日期:2016-04-28;修回日期:2016-10-17;责任编辑:孙瑶
基金项目:国家自然科学基金(No. 61402464, No. 61602467)

① http://hadoop.apache.org.
② 数据中心局点是指数据中心里分布在同一地域上资源的集合

于输入数据所在的位置,将任务下发至待处理数据所在的数据中心局点执行,并回传执行结果. 文献[4~6]表明,作业跨域分布执行避免了大规模数据迁移,能将跨域网络带宽消耗降低至原消耗的 1/200 以下,能将 90% 以上的作业完成时间降低至原完成时间的 1/3 以下.

数据放置约束并不是造成作业跨域执行的唯一原因,文献[9]描述了一类运行于跨域数据中心(Geo-distributed Data Centers,GDC)之上的面向互联网运行状态分析的测试类应用. 为实时监控从全国各地访问某些特定网站响应速率,应用需在各地域数据中心的各网络接入类型的资源节点执行对应计算任务,以测量网站响应速率,并将结果汇总至上层业务模块进行深入分析与诊断. 此类任务通常对资源物理位置、接入网络资源类型有着硬性约束,导致同一作业中的任务需要在多个数据中心局点上执行.

作业的跨域分布执行给作业调度带来了新的挑战[3],作业中执行最慢的任务决定了作业的完成时间,只提高部分数据中心上任务的执行效率并不会降低作业完成时间. 同时,各局点服务能力的差异性、作业中任务(负载)在各局点分布的不均衡性,进一步加剧了跨域作业调度的难度. 特定任务集合在某局点数据中心上的执行时间跨度取决于任务量的多少和该数据中心可用的资源量,相当于在单一数据中心下,寻找批量任务的最优调度策略以最小化执行时间跨度,这是一个 NP 问题[10]. 跨域作业执行时间跨度最小化问题可以转化为多个数据中心环境下批量任务执行时间跨度最小化问题. 目前,针对云数据中心环境下跨域作业调度问题的研究很少. 文献[3]为降低批量作业的平均执行时间跨度提出了一系列的优化技术,如子作业重排序、各数据中心负载感知的作业调度方法等,但这些技术假设作业中各任务执行时间复杂度一致、各数据中心资源服务能力一致且各数据中心以单一队列提供服务.

本文研究了任务大小不一和数据中心资源异构环境下批量跨域作业调度问题,提出一种名为 MIN-Max-Min 调度方法,以启发式的方式寻求最优解,并通过实验分析说明该方法的有效性.

2　跨域作业模型

域通常是指特定对象集合. 如数学领域中的值域指定义域中所有元素在某映射法则下对应的所有对象组成的集合. 而计算机网络中的域是指网络中一组计算机的逻辑集合,是活动目录逻辑结构中的核心单元,它定义了安全边界,每个域均有各自的安全策略以及与其他域的信任关系. 在没有授权的情况下,不允许其他域中的用户访问本域中的资源. GDC 中的域一般是指地域,即此类数据中心的资源分布在多个地域空间内.

跨域作业中的域是指数据中心内特定资源的集合. 云数据中心往往通过虚拟化技术屏蔽底层硬件的差异,对上层呈现的是虚拟资源. 资源域可表示为 $RD = \{vm_1, vm_2, \cdots, vm_m\}$. 一个作业是跨域的意味着该作业中的任务至少需要运行在两种资源域上. 最简单的,一个局点(地域)的数据中心中所有的资源可以看成是一个域,即按照地域来划分资源. 但是,仅仅按地域划分资源有时是不够的,有些任务对资源节点除了有地域要求外,还对其接入网络的服务商类型、接入方式有着个性化的需求[9]. 图 1 描述了按地域和服务商类型的域划分情况,同一域的资源分布在同一局点且接入的网络服务商一致.

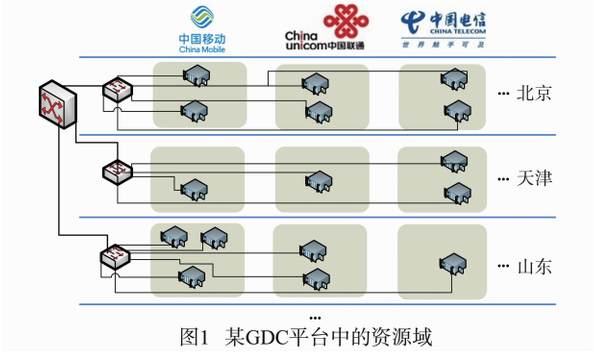

图1　某GDC平台中的资源域

一个跨域作业是一系列任务的集合. 进一步地,可根据满足特定任务执行的资源域不同将作业划分成一个个子作业(SubJob),同一子作业中的任务可在同一域中的任何资源上执行. 为便于描述跨域作业及所包含的任务,本文用元组来表示任务 T_i,即 $T_i = \{id_i, pl_i, np_i, st_i, len_i, et_i\}$,其中:

id_i:任务的标识;

pl_i:任务执行所需要资源的物理位置,取值为枚举类型;

np_i:任务执行所需资源接入网络的服务商,取值为枚举类型,可取 CM(中国移动)、CU(中国联通)、CT(中国电信);

st_i:任务的起始执行时间,初始值为-1 表示还未执行;

len_i:预计任务执行所包含的机器指令数,单位为百万;

et_i:任务结束执行的时间,初始值为 0 表示还未执行或未执行结束.

任务执行时长可预知是众多任务调度算法的前提条件[3,11~16]. 在不考虑资源异构情况下,通常会直接给出任务执行时间长度[3];在考虑资源异构时一般会用任务所包含机器指令数表示任务长度,并根据资源节

点 MIPS(每秒可处理的百万条指令数,Million Instructions Per Second)性能[14-15],预估任务在该节点上执行所需时间.

同样地,作业和子作业也可以用元组来表示,即子作业 $SJ_i = \{id_i, pl_i, np_i, st_i, len_i, et_i, T_i\}$,前三个元素分别表示子作业的标识、对应域的物理位置及所接入的网络服务商,pl_i 和 np 共同决定了能够满足子作业执行的资源域. st_i 表示子作业的起始执行时间,等于 T_i 中最早执行任务的起始执行时间,len_i 表示所包含的任务数量,et_i 表示子作业的完成时间,等于 T_i 中最后执行完毕任务的完成时间,T_i 表示所包含的任务集合,其中的每个任务都可以在同一资源域上执行.

一个跨域作业是特定子作业的集合,表示如下:$J = \{id_i, st_i, len_i, et_i, SJ_i\}$,$id_i$ 为作业的标识,st_i 为作业的起始执行时间,等于 SJ_i 中最早执行子作业的起始执行时间,len_i 为包含的子作业个数,et_i 为作业的完成时间,等于 SJ_i 中最后执行完毕的子作业完成时间,SJ_i 表示所包含的子作业集合.

图 2 描述了一个跨域作业实例 J_1,该作业一共包含 10 个任务,需在上海和北京局点数据中心里互联网接入服务商为中国移动的资源节点上执行. 因此,可按资源域的不同将该作业分成 2 个子作业 SJ_1 和 SJ_2,并各包含 5 个任务. 因为这 10 个任务刚刚到达对应的资源域并未开始执行,它们的起始执行时间及完成时间都为初始值.

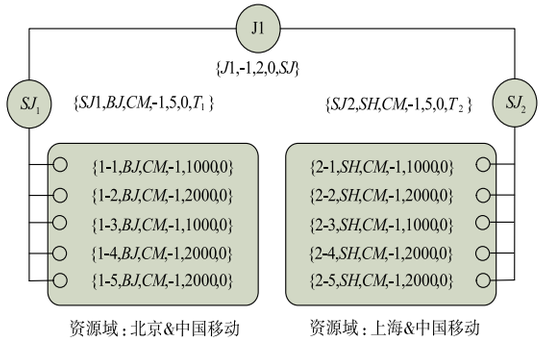

图2 一个跨域作业实例

3 问题描述

降低批量作业(任务)执行时间跨度(MakeSpan)是作业调度的最常见目标,各资源域服务能力水平差异性及各作业中任务在各资源域分布不均衡性加大了跨域作业调度的难度. 本章以一个实例引出研究的问题.

表 1 列出三个依次到达的跨域作业 A、B、C 及所包含任务在 RD_1、RD_2 和 RD_3 三个域中分布情况,作业 A 共四个任务,作业 B 与作业 C 各包含两个任务. 作业 A、B、C 所包含的任务需要运行在三个资源域上,各资源域表示如下:$RD_1 = \{vm_{10}\}$,$RD_2 = \{vm_{20}, vm_{21}\}$,$RD_3 = \{vm_{30}\}$. 图 3 描绘了各资源的 MIPS 指标(资源下部刻度用以度量该资源 MIPS 值),该指标能够反映出资源的服务能力. 其中,RD_1 包含一个资源 vm_{10},服务能力为 2000MIPS;RD_2 包含两个资源,vm_{21} 服务能力最强,为 3000MIPS,vm_{20} 服务能力为 1000MIPS; RD_3 包含一个资源 vm_{30},服务能力为 1000MIPS.

表 1 跨域作业到达序列及任务分布情况

ID	到达顺序	RD_1 中任务	RD_2 中任务	RD_3 中任务	任务总数
A	1	$t_{a10}\{len=3000\}$	$t_{a20}\{len=1000\}$ $t_{a21}\{len=3000\}$	$t_{a30}\{len=1000\}$	4
B	2	$t_{b10}\{len=2000\}$	$t_{b20}\{len=3000\}$		2
C	3	$t_{c10}\{len=2000\}$		$t_{c30}\{len=2000\}$	2

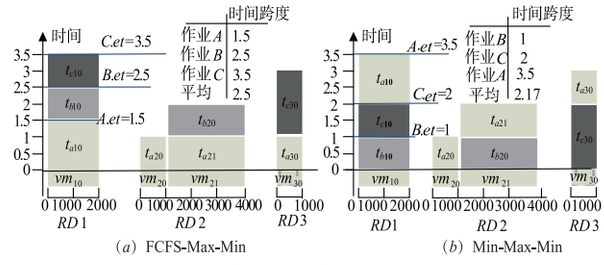

图3 跨域资源及跨域作业调度实例

根据任务执行所需资源位于资源域的不同可将 A、B、C 分成多个子作业. 其中,A 包含三个子作业,即 $A.SJ = \{SJ_{a1}, SJ_{a2}, SJ_{a3}\}$,$SJ_{a1}$ 包含一个任务,即 $SJ_{a1}.T = \{t_{a10}\}$,同样地,$SJ_{a2}.T = \{t_{a20}, t_{a21}\}$,$SJ_{a3}.T = \{t_{a30}\}$;$B$ 包含两个子作业,$B.SJ = \{SJ_{b1}, SJ_{b2}\}$,$SJ_{b1}.T = \{t_{b10}\}$,$SJ_{b2}.T = \{t_{b20}\}$;$C$ 同样包含两个子作业,$C.SJ = \{SJ_{c1}, SJ_{c3}\}$,$SJ_{c1}.T = \{t_{c10}\}$,$SJ_{c3}.T = \{t_{c30}\}$. 各任务的执行长度见表 1.

完成一个跨域作业需各个子作业的任务都执行完毕. 因此,为降低作业执行时间跨度,需降低作业中每个子作业的执行时间跨度. 子作业中的任务都执行在同一个资源域下,最小化子作业的完成时间等同于求解批量任务的调度策略以最小化任务执行跨度,该问题是一个 NP 问题,已有 Min-Min、Max-Min[17] 等众多求解方法. 其中 Max-Min 较为常见,其基本思想是优先调度大任务到执行效率最高的资源上执行以降低批量任务的整体完成时间,本文选用 Max-Min 算法作为子作业内任务的调度策略.

图 3 展示了针对 A、B、C 三个跨域作业,采用不同作业调度策略(子作业内任务调度都采用 Max-Min 算法)的调度结果. 横轴表示共有三个资源域(RD_1、RD_2、RD_3),域中含有斜线或者方格线的方块表示域中资源,

方块长度(见下方刻度,单位为 MIPS)表示该资源的服务能力;纵轴表示作业(任务)执行时间;两轴交叉的右上方区域展示了任务的执行情况,方块面积表示任务的长度,方块的高度表示任务的执行时长,浅色方块表示属于作业 A 的任务,灰色方块表示属于作业 B 的任务,深灰色方块表示属于作业 C 的任务.

在针对批量作业的调度上,先来先服务(First Come First Service,FCFS)是一种最直观的方法,即将按照作业到达的先后次序来进行调度,优先为最先到达的作业及作业所包含的任务提供服务. 图 3(a)展示了 FCFS 调度策略下各作业的任务执行情况,作业执行顺序与作业到达顺序保持一致,作业 A、B、C 的完成时间分别为:$A.\ et=1.5$、$B.\ et=2.5$、$C.\ et=3.5$,三个作业的平均执行时间跨度为 2.5. 然而,FCFS 方法并未达到最优调度效果. 由于 B、C 相对 A 来讲都是较小的作业,作业 A 优先执行容易造成小作业的盲等,延长了批量作业的平均执行时间跨度,可通过适当调整作业执行顺序来降低作业平均执行时间跨度. 图 3(b)展示了以 BCA 的顺序进行作业调度时的任务执行情况,作业 A、B、C 的完成时间分别为:$A.\ et=3.5$、$B.\ et=1$、$C.\ et=2$,三个作业的平均执行时间跨度为 2.167,与 FCFS 策略相比,降低了 13.3% 的执行时间跨度. 目前,针对跨域云数据中心资源异构条件下跨域作业调度问题的研究非常少,本文将 FCFS 策略作为基准线以比较说明所提作业调度策略的有效性.

本文研究的跨域作业调度问题描述如下:给定一个跨域作业集合 $\boldsymbol{J}=\{J_0,J_1,\cdots,J_{n-1}\}$($n$ 为集合中作业的个数,各作业按下标数字升序的顺序到达),可满足各作业执行需求的资源域集合 $\boldsymbol{R}=\{RD_0,RD_1,\cdots,RD_{m-1}\}$(即按任务执行对资源需求的不同可将云数据中心的资源分成 m 个资源域),求解最优的调度策略,使得作业集合中作业平均完成时间最小,即式(1)值最小.

$$\min\left(\left(\sum_{i=0}^{n}J_i.\ et\right)/n\right) \tag{1}$$

4　MIN-Max-Min 调度方法

本文提出一种名为 MIN-Max-Min 的两级调度方法以最小化批量作业执行时间. 资源域内部,基于 Max-Min 算法求解使子作业完成时间最小的调度方法以完成任务级调度;资源域全局,基于最小完成时间优先的方法实现作业级调度.

4.1　两级调度架构

本文提出的两级调度架构如下图 4 所示. 图中共有两个作业(J_1,J_0)正在执行,三个作业(J_2,J_3,J_4)待调度,两个作业已提交(J_5,J_6),作业下标值越小表明作业提交越早. 资源分布在 n 个数据中心,每个数据中心根据局点内资源的特点并基于任务常见的执行约束(如资源所接入的互联网服务商)将资源划分成更小的域. 每个域都对应一个子作业任务调度器,负责不同作业中该域对应子作业的调度.

作业调度器部署在跨域数据中心主控节点,负责全局作业级调度,调度结果为各作业执行顺序. 作业调度器维持一个作业窗口,并定时启动调度程序,完成窗口内批量作业的调度,窗口大小根据调度模块负载进行调节. 主控节点同时会将各个作业的子作业下发至相应资源域的子作业调度器. 完成作业调度后,作业调度器通过主控节点将作业执行顺序下发至相应子作业调度器.

子作业调度器根据作业执行顺序调度各子作业任务在本资源域运行,并通过心跳机制定期向主控节点反馈自身状态信息. 图 4 中,作业 J_0 和 J_1 经作业调度器调度后执行顺序为 $J_1\rightarrow J_0$,则 RD_{10} 和 RD_{n0} 子作业调度器优先为作业 J_1 的子作业分配资源. 确定子作业执行顺序后,各子作业中任务调度方式可采用 Max-Min 算法实现. 作业各个子作业全部执行完毕后则该作业完成.

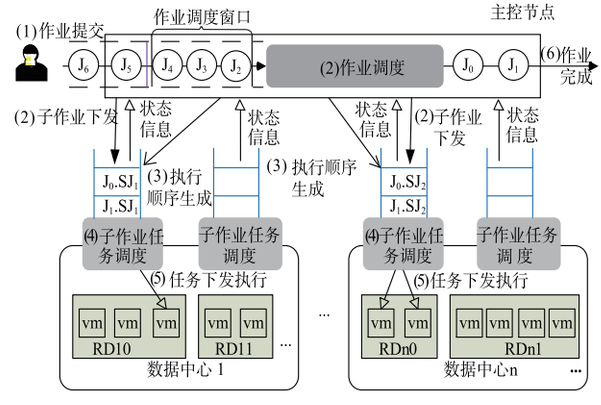

图4　跨域作业两级调度架构

图 4 还描述了作业所经历的主要处理步骤:

(1)作业提交. 即由用户将作业提交给云数据中心平台.

(2)作业调度. 作业调度器根据一定策略决定批量作业的执行顺序. 为提高任务下发的效率,在作业调度的同时可将子作业下发至对应的资源域.

(3)执行顺序生成. 子作业任务调度模块按照作业调度结果生成本地子作业的执行顺序.

(4)子作业任务调度. 子作业任务调度模块按序完成每个子作业中任务的调度.

(5)任务下发执行. 任务下发至匹配的资源上执行.

(6)作业完成. 在所有子作业中所有的任务执行完

毕后,该作业执行完成.

4.2 MIN-Max-Min 算法

特定资源域下,确定各子作业调度顺序后,子作业内任务的调度问题等同于云数据中心资源异构条件下批量任务执行时间跨度最小化问题,后者可用经典的 Max-Min 算法或者更为复杂的遗传算法来解决. 本文主要对作业级的调度策略进行研究,以最小化批量作业的平均完成时间.

作业完成时间为作业等待时间和执行时间之和,为降低批量作业平均执行时间跨度,最直观的资源分配原则是:能尽快完成的作业应尽快执行. 基于上述思想,本文提出一种名为 MIN-Max-Min 的启发式算法,优先选择期望完成时间最短的作业执行. 资源在作业级分配原则为:优先执行预期完成时间短的作业. 具体地,可用 Max-Min 算法评估各子作业内批量任务在对应资源域上最快完成时间,进而计算得出作业的最快完成时间,该值越小的作业越优先执行,作业完成后更新资源域相应资源的就绪时间,直至所有作业执行完成. 算法如算法 1 所示.

算法 1 MIN-Max-Min 批量跨域作业调度算法

```
算法 MIN-Max-Min(J,R)
输入:跨域作业集合 J(n 个作业 J_0 ~ J_{n-1}),资源域集合 R(RD_0 ~ RD_{m-1})
输出:资源序列 L(越靠前的作业越优先执行)
1.  List L;//初始化为空,用以按序存放调度完的作业
2.  while(L. size() < n) {//若还有未调度的作业
3.      for(Job J_i in J) m[i] = cpTimeandAlloRes (J_i,R,false);//计算每个作业的预期完成时间
        minCT = min(m[ ])//数组 m 中最小的值
4.      Find targetJ:targetJ. et = = minCT;//预期完成时间最小的作业为 targetJ
5.      L. add(targetJ);//放入资源序列
6.      cpTimeandAlloRes (targetJ,R,true); //为 targetJ 分配资源
7.      J. remove(targetJ);//从作业集合中移除 targetJ}
8.  Return L; //返回作业调度结果
```

```
算法 cpTimeandAlloRes(J,R,allocate)
输入:跨域作业 J,资源域集合 R(RD_0 ~ RD_{m-1}),布尔型值 allocate
输出:作业 J 的期望完成时间 et
1.  float et = 0;//初始化期望完成时间
2.  List SJ = J. SJ;//J 的子作业集合 SJ
3.  for (SubJob SJ_i in SJ){
4.      RD = SJ_i. domain();//RD 为子作业对应的资源域
5.      ct[i] = maxmin(SJ_i,RD,allocate);//计算每个子作业按 Max-Min 算法调度时的预计完成时间}
6.  et = max(ct[ ]);//et 为最慢子作业的完成时间
7.  Return et;//返回作业的期望完成时间,若 allocate 为 true 则同步完成资源的分配
```

```
算法 maxmin(SJ,RD,allocate)
输入:子作业 SJ,资源域 RD,布尔型值 allocate
输出:子作业 J 的期望完成时间 et,若 allocate 为 true 则需更新任务拟执行资源的就绪时间
1.  float et = 0;//初始化期望完成时间
2.  VMList = RD. res;// VMList 为 RD 中资源集合
3.  List tList = SJ. T; //子作业 SJ 的任务集合 tList
4.  while(tList. size() > 0){//找出 tList 中执行时长最长的任务,直至为空
5.      for(task t_i in tList){//
6.          for(vm_j in VMList){
7.              e_ij = t_i. len/ vm_j. MIPS;//预期执行时长为任务长除以资源 MIPS 值
8.              c_ij = e_ij + r_j; //计算每个任务在每个资源上的预期完成时间,r_j 为资源就绪时间,无资源执行时为 0}}
9.      for(task t_i in tList) minct[i] = min(c[i][ ]) ;//找出任务 t_i 最快完成时间
10.     t. et = max(minct[i]);// 求出最快完成时间最大的任务 t 及对应资源 vm
11.     allocate(t,vm);//将 t 分配至资源 vm 执行
12.     vm. r + = t. len/vm. MIPS;//更新 vm 的就绪时间
13.     t. setET(vm. r);//设置 t 的完成时间
14.     tList. remove(t);//t 已完成调度,从 tList 中删除}
15.     et = Max(t. et);//子作业完成时间为最慢的任务执行结束时间
16.     if(! allocate) VMList. rollback();//若只是为计算预期完成时间则还需将 VMList 中的每个虚拟资源回滚至任务分配前的状态,即将就绪时间设为原值
17.     Return et;//返回子作业期望完成时间
```

算法 MIN-Max-Min 需调用 cpTimeandAlloRes 方法. 后者计算特定作业预期完成时间,若 allocate 值为 true 则计算过程中会将资源分配给该作业,它调用 maxmin 方法计算每个子作业在对应域上最快完成时间. maxmin 方法是 Max-Min[17] 的实现. MIN-Max-Min 本质上是一种贪婪算法,作业选取原则是期望完成时间最早的先执行.

maxmin 算法中,设子作业的任务数量为 t,资源域中资源的数量为 d,则该算法的时间复杂度为 $O(t^2m)$. cpTimeandAlloRes 算法中,设子作业的个数为 s,则该算法的时间复杂度为 $O(st^2m)$,由于每个子作业中任务的数量及相应资源域资源数量不同,其中的 t 和 m 可取平均值. MIN-Max-Min 算法中,作业的数量为 n,则该算法的复杂度为 $O(n^2st^2m)$,s 为各作业所包含子作业数量的平均值,t 为子作业所包含任务数量的平均值,m 为各资源域所包含资源数量的平均值,具体复杂度和各作业所包含子作业数量、各子作业包含任务的数量及资源在各域中的分布情况有关. 下一节通过实验说明 MIN-Max-Min 算法有效性时,将着重分析其时间及空间

消耗.

5　实验分析

目前,针对云数据中心资源异构条件下跨域作业调度问题的研究非常少,本节使用 CloudSim3.0 模拟器[18]构建实验环境,并将 FCFS 策略作为基准以对比说明方法的有效性. FCFS 策略下,作业级调度时依据先来先服务的原则决定调度次序,而在子作业任务级调度同样采用 Max-Min 算法. 因此,FCFS 策略可相应的表示为 FCFS-Max-Min. 实验通过模拟器仿真构造出一系列作业负载,并分别从效果及调度消耗(内存和时间)两方面进行对比.

5.1　MIN-Max-Min 策略效果

模拟出的数据中心及负载概况如表 2 所示. 为了比较验证在不同量级的资源域下两种调度策略针对相同作业的调度效果,本文设计了四种场景. A 场景下,数据中心可分为 3 个域,共包含 4 个资源;并分别模拟出了 2 个、10 个、100 个作业,作业中一共含有的任务总数分别为 5、48、545,各作业中的任务在各资源域随机分布形成相应的子作业. A 场景主要用以模拟资源数量较少且负载相对较重的情况. B 场景下,数据中心包含了 10 个资源域,一共拥有 100 个资源;同样模拟出 2 个、10 个、100 个作业,作业中一共含有的任务总数分别为 19、63、541. 与 A 场景相比,B 场景下任务负载基本不变,但资源域和资源规模大幅增长. B 场景主要用以模拟资源相对充裕的情况. C 场景下,资源进一步分散,100 个资源分布在 30 个资源域下,主要用以模拟资源相对分散但资源量不大的情况;场景 D 扩大了资源的总数及分布范围,共模拟出 10000 个资源并分布在 100 个资源域下,主要用以模拟资源量较大且较为分散的情况. 每种场景下模拟出不同数量的作业及任务,用以比较不同负载下两种策略的调度效果,结果如图 5 所示. 为便于观察,左侧纵轴采用对数刻度(单位为 s),并绘制了完成时间对比情况.

表 2　模拟的数据中心及负载概况

场景	数据中心概况		负载概况(任务个数)		
	资源域个数	资源数	2 个作业	10 个作业	100 个作业
A	3	4	5	48	545
B	10	100	19	63	541
C	30	100	122	610	5710
D	100	10000	2169	15063	143708

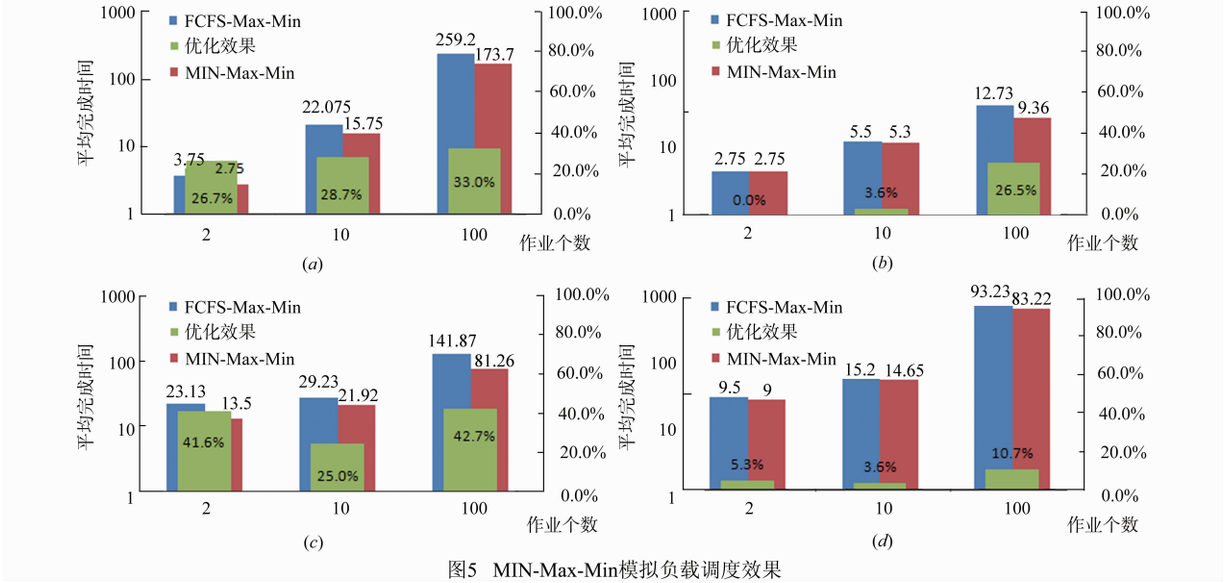

图5　MIN-Max-Min模拟负载调度效果

图 5(*a*)为场景 A 下作业个数取不同值时,两种调度策略下作业的平均完成时间. 当有 2 个作业共 5 个任务时,尽管负载较轻(平台共有 4 个资源),但 MIN-Max-Min 策略下作业平均完成时间仍旧比 FCFS-Max-Min 策略下平均完成时间减少 26.7%. 当作业数增至 10 个、任务总数增至 48 个时,MIN-Max-Min 策略能减少 28.7%的作业平均完成时间. 作业数增至 100 时,MIN-Max-Min 策略能减少 33%的作业平均完成时间.

B 场景下数据中心负载相对较轻,2 个作业时总任务只有 19 个,远小于资源总数,两种调度策略下作业平均完成时间一致. 原因在于各域资源总量充足情况下,各任务无需等待,作业完成时间取决于任务执行时间. 但资源总量充足并不表明任务无需等待,如作业个数为 10、任务总量为 63 时,MIN-Max-Min 策略仍能提升效果,因为资源在各域分布及不同作业在不同资源域中的任务分布都是非均匀的,造成某些资源域负载过重. 当作业数增至 100、任务总量达 541 时,MIN-Max-Min 策略能减少 26.5%的平均作业完成时间.

与场景 B 相比,场景 C 将 100 个资源分布至 30 个资源域,各策略作业完成时间如图 5(*c*)所示. 当有 2 个作业共计 122 个任务时,MIN-Max-Min 策略能减少 41.6%的平均作业完成时间,但此时整体负载并不重. 相反,当有 10 个作业共计 610 个任务时,MIN-Max-Min 策略只能减少 25%的平均作业完成时间. 而当作业数增至 100、任务总量增至 5710 时,MIN-Max-Min 策略能减少 42.7%的平均作业完成时间. 场景 C 实验表明,MIN-Max-Min 策略效果与作业中各任务在各域分布及资源域负载都有关系.

场景 D 将资源域数量增至 100,资源总数增至 10000 个. 与场景 C 相比,其负载并不重,最大负载为 100 个作业共计 143708 个任务,平均每个资源需执行 14.37 个任务. 而场景 C 负载最大时,平均每个资源需执行 57.1 个任务. 因此,场景 D 下任务负载最大时,MIN-Max-Min 策略只能减少 10.7% 平均作业完成时间,远低于场景 C 下 42.7%的效率提升.

上述实验表明,MIN-Max-Min 策略能有效降低批量作业平均完成时间. 效果与平台负载及各作业在各域任务分布相关,负载越重、各域任务分布越不均衡 MIN-Max-Min 策略效果越明显.

5.2 MIN-Max-Min 策略复杂度分析

本节对比分析 FCFS-Max-Min 策略和 MIN-Max-Min 策略时间复杂度、空间复杂度. 上节实验时,记录了两种调度策略下批量作业完成调度所需时间及占用内存大小. 调度节点为一台曙光 A620r-F 服务器(双核 2.6GHz AMD2218CPU、4GB 内存),操作系统为 Turbo Linux 3.4.3 版本,调度算法用 Java 实现,运行环境为 jre1.7,各场景下 FCFS-Max-Min 策略和 MIN-Max-Min 策略时间及内存消耗如图 6 所示,四个场景分别对应(*a*)、(*b*)、(*c*)、(*d*)四图.

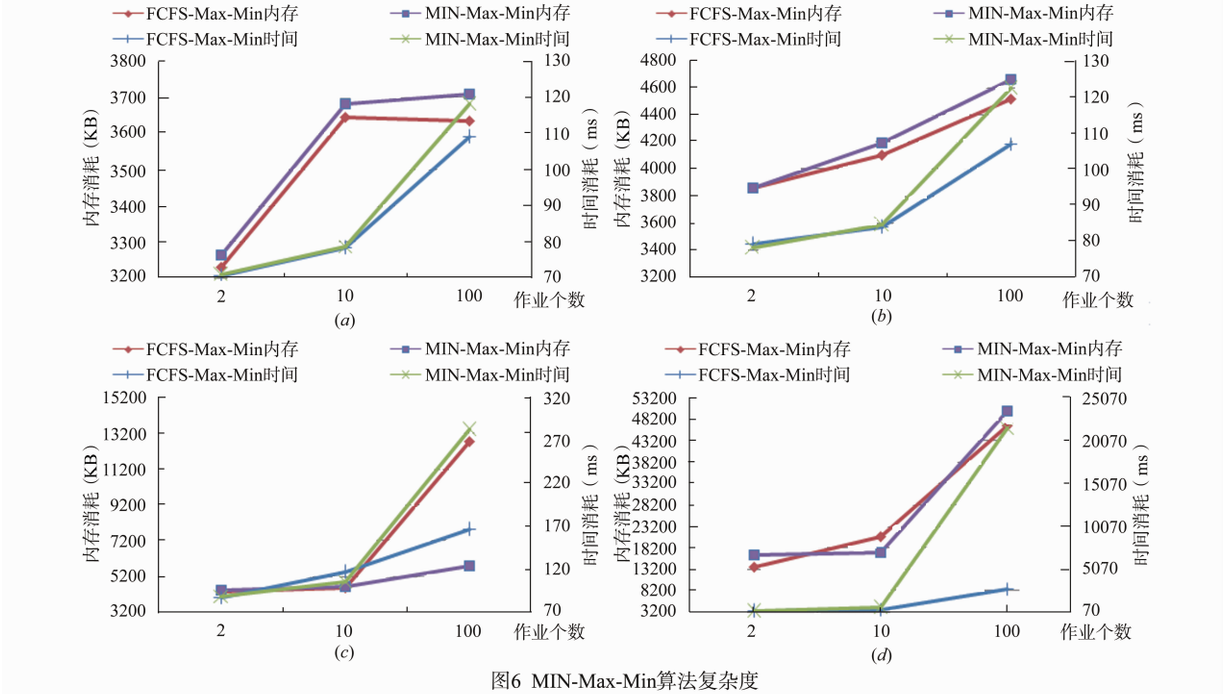

图6 MIN-Max-Min算法复杂度

从图 6 可以看出,MIN-Max-Min 策略和 FCFS-Max-Min 策略空间复杂度基本一致. 随着作业数量、任务数量、资源域个数、资源规模的增加,两种策略占用空间都会增长. 场景 D 中,资源域达 100 个、资源总数达 10000 个,而作业数量为 100、任务数量为 143708 个时,FCFS-Max-Min 策略内存消耗为 49MB,在可以接受范围内.

FCFS-Max-Min 策略和 MIN-Max-Min 策略时间复杂度分别为: $O(nst^2m)$、$O(n^2st^2m)$,n 为作业数量,s 为各作业所包含子作业数量的平均值,t 为子作业所包含任务数量的平均值,m 为各资源域所包含资源数量的平均值. 由图 6 看出,作业量由 10 增至 100,FCFS-Max-Min 策略与 MIN-Max-Min 策略时间消耗差距会迅速扩大,但并未成 n 倍增长. 因为,具体复杂度和各作业所包含子作业数量、各子作业包含任务的数量及资源在各域中分布有关. 场景 D 中,作业数量为 100 时,FCFS-Max-Min 时间消耗为 2.84s,MIN-Max-Min 时间消耗为 21.6s. 两者时间消耗差距拉大原因在于,此时 n 值变为 100,且资源域个数增至 100 时意味各作业包含子作业数量 s 也会急剧增长,而 t 值由之前场景中的个位数

(如 C 场景下,100 个作业时,t 值为 1.9)增加为十位数(D 场景下 100 个作业时 t 值为 14.4),多重因素导致场景 D 中 MIN-Max-Min 时间消耗增至 21.6s. 但应该看到,尽管调度的作业数量为 100,但共调度的任务量为 14.4 万,平均每个任务调度所需时间为 0.15ms,在可接受范围之内.

6　结束语

本文提出一种以降低批量跨域作业平均执行时间跨度为目标的调度方法 MIN-Max-Min. 该方法基于启发式原则,优先选取预期完成时间最早的作业执行,在资源域内部利用 Max-Min 算法以最小化批量任务执行时间跨度. 通过实验对比了 MIN-Max-Min 方法与基于先来先服务方法调度效果,模拟任务负载下 MIN-Max-Min 方法能将跨域作业平均执行时间跨度减少 40% 以上.

参考文献

[1] ZAHARIA M,CHOWDHURY M,FRANKLIN M J,et al. Spark:Cluster computing with working sets[J]. HotCloud,2010,10:10 - 10.

[2] ISARD M,BUDIU M,YU Y,et al. Dryad:Distributed data-parallel programs from sequential building blocks[J]. ACM SIGOPS Operating Systems Review,2007,41(3):59 - 72.

[3] HUNG C C,GOLUBCHIK L,YU M. Scheduling jobs across geo-distributed datacenters[A]. Proceedings of the Sixth ACM Symposium on Cloud Computing[C]. USA:ACM,2015. 111 - 124.

[4] VULIMIRI A,CURINO C,GODFREY P B,et al. Global analytics in the face of bandwidth and regulatory constraints[A]. The 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)[C]. USA:USENIX,2015. 323 - 336.

[5] VULIMIRI A,CURINO C,GODFREY B,et al. WANalytics:Analytics for a geo-distributed data-intensive world[A]. Proceedings of CIDR[C]. USA:ACM,2015. 27353635.

[6] HAJJAT M,MALTZ D,RAO S,et al. Dealer:application-aware request splitting for interactive cloud applications[A]. Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies[C]. USA:ACM,2012. 157 - 168.

[7] Zhou A,Wang S,Zheng Z,Hsu C,Lyu M,Yang F. On cloud service reliability enhancement with optimal resource usage[J]. IEEE Transactions on Cloud Computing,DOI:10.1109/tcc.2014.2369421.

[8] 俞能海,郝卓,徐甲甲,张卫明,张驰. 云安全研究进展综述[J]. 电子学报,2013,41(2):371 - 381.
YU Neng-hai,HAO Zhuo,XU Jia-jia,ZHANG Wei-ming,ZHANG Chi. Review of cloud computing security[J]. Acta Electronica Sinica,2013,41(2):371 - 381. (in Chinese)

[9] LI Y,ZHANG H,WANG Y,et al. A Scheduling framework for periodic tasks in geo-distributed data centers[A]. IEEE Symposium on Service-Oriented System Engineering (SOSE)[C]. USA:IEEE,2015. 247 - 252.

[10] 童钊,肖正,李肯立. 分布式系统中多用户网络应用的概率型调度算法研究[J]. 电子学报,2016,44(7):1679 - 1688.
TONG Zhao,XIAO Zheng,LI Ken-li. A queueing model and probabilistic scheduling for multi-user network applications[J]. Acta Electronica Sinica,2016,44(7):1679 - 1688. (in Chinese)

[11] SCHWARZKOPF M,KONWINSKI A,ABD-EL-MALEK M,et al. Omega:flexible,scalable schedulers for large compute clusters[A]. Proceedings of the 8th ACM European Conference on Computer Systems[C]. USA:ACM,2013. 351 - 364.

[12] OUSTERHOUT K,WENDELL P,ZAHARIA M,et al. Sparrow:distributed,low latency scheduling[A]. Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles[C]. USA:ACM,2013. 69 - 84.

[13] 谭一鸣,曾国荪,王伟. 随机任务在云计算平台中能耗的优化管理方法[J]. 软件学报,2012,23(2):266 - 278.
TAN YM,ZENG GS,WANG W. Policy of energy optimal management for cloud computing platform with stochastic tasks[J]. Journal of Software,2012,23(2):266 - 278. (in Chinese)

[14] JOE-WONG C,SEN S,LAN T,et al. Multiresource allocation:Fairness-efficiency tradeoffs in a unifying framework[J]. IEEE/ACM Transactions on Networking (TON),2013,21(6):1785 - 1798.

[15] KIM K H,BELOGLAZOV A,BUYYA R. Power-aware provisioning of cloud resources for real-time services[A]. Proceedings of the 7th International Workshop on Middleware for Grids,Clouds and e-Science[C]. USA:ACM,2009. 1.

[16] 陶晓玲,韦毅,王勇. 一种基于分层多代理的云计算负载均衡方法[J]. 电子学报,2016,44(9):2106 - 2113.
TAO Xiao-ling,WEI Yi,WANG Yong. A load balancing method based on hierarchy and multi-agent for cloud computing platform[J]. Acta Electronica Sinica,2016,44(9):2106 - 2113. (in Chinese)

[17] MAHESWARAN M,ALI S,SIEGAL H J,et al. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems[A]. Proceedings

of Heterogeneous Computing Workshop (HCW ´99) (Eighth)[C]. USA:IEEE,1999. 30 – 44.

[18] CALHEIROS R N,RANJAN R,BELOGLAZOV A,et al. CloudSim:A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms[J]. Software:Practice and Experience,2011,41(1):23 – 50.

作者简介

李 焱 男,1984 年 2 月生,湖北随州人. 2016 年在中国科学院计算技术研究所获工学博士学位. 主要研究方向为分布式计算、云计算等.

E-mail:liyan@ncic.ac.cn

郑亚松 男,1987 年 9 月生,河南濮阳人. 2014 年在中国科学院计算技术研究所获工学博士学位. 现在国家互联网应急中心工作,主要研究方向为大数据分析.

E-mail:zys_2009@163.com