doi:10.3772/j.issn.1002-0470.2018.08.001

企业私有云环境下面向高性能计算的资源弹性分配算法①

刘晓东②*** 赵晓芳③*** 金 岩** 罗 刚*** 陈雅静*** 赵曙光***

(*中国科学院大学 北京 100049)

(**中国科学院计算技术研究所 北京 100190)

(***中国石油集团东方地球物理公司物探技术研究中心 涿州 072751)

摘 要 为应对企业私有云环境下高性能计算面临的提高集群资源利用率和保障用户计算能力的挑战,本文提出了一种基于用户资源配额的资源弹性分配算法(QREA)。该算法在计算资源充足时,通过将空闲资源追加给弹性作业以提高资源利用率;当计算资源紧张时,通过资源回收以保障用户的资源配额。为提高多用户共享集群资源的公平性,提出了基于用户历史信用值的资源追加和回收算法。大量的实验结果表明,在低作业负载下,QREA 算法与基本资源分配算法 BRA 和多用户公平调度算法 Fair Scheduler 相比,作业平均完成时间缩短 50%;在高作业负载下,QREA 算法比 BRA 算法在资源利用率上提高 10%,在多用户公平性上,比基于当前资源使用情况的资源弹性分配算法 CREA 提高34.5%。

关键词 资源弹性分配,企业私有云,高性能计算,资源利用率,多用户环境

0 引 言

高性能计算(high performance computing,HPC)利用大量计算单元的聚合计算能力来解决复杂的科学计算或数值计算问题,被广泛应用在天气预报、石油勘探、分子模拟、基因测序、卫星图像处理、情报分析等领域[1]。由于科学研究对计算能力的需求是永无止境的,随着集群规模和用户数量的不断增加,传统的高性能计算面临集群资源利用率低、用户服务质量下降、系统管理难度增加等问题[2]。

云计算(cloud computing)是一种新型的计算模式,通过资源整合与隔离,提供了一种按需访问共享资源池的方法,具有资源共享、弹性配置和按需服务等特点。云计算的灵活弹性计算模式一定程度上解决了传统高性能计算所遇到的问题,已经广泛地运用在高性能计算领域。但是,由于云计算模式下数据所有权和管理权的分离,带来诸多新的安全问题和挑战[3,4]。

目前,基于云计算理念构建 HPC 企业私有云,充分利用传统高性能计算资源可控、高可用、安全的优势和云计算资源共享、易于管理的优势,为企业内部业务实施提供计算服务平台,是企业信息化的发展方向[5]。

资源分配算法是影响集群资源利用率和用户服务质量[6-8]的主要因素之一,已成为学术界和工业界研究的热点。传统的高性能计算以提高集群资源利用率和作业吞吐量为目的,通过多用户共享整个集群计算资源的模式来提高以上两个指标,但是其很少关注多用户的计算能力保障。而云计算下的用户按需付费,具有严格的资源限定,虽然为用户提供了明确的计算能力,但是降低了资源共享的程度,因而存在计算资源使用不均衡的问题,即某些用户的计算资源紧张而某些用户的计算资源空闲。

① 国家重点研发计划(2016YFB0201500)资助项目。

② 男,1989 年生,博士生;研究方向:云计算,高性能计算;E-mail:liuxiaodong@ict.ac.cn

③ 通信作者,E-mail:zhaoxf@ict.ac.cn

(收稿日期:2018-01-06)

在固定计算资源情况下,多用户使用的高性能计算系统达到资源利用率和用户计算能力的平衡,是企业私有云环境下高性能计算追求的目标和面临的挑战。其中,资源配额是衡量计算能力最直接、最有效的标准。在高性能计算环境下,资源配额根据高性能作业的编程框架,具有不同的数值含义。例如,如果作业的资源请求是节点(或者 CPU 核、GPU 等),那么资源配额指的是节点数(或者 CPU 核数、GPU 数等)。此外,一个高性能计算系统中可能支持多种编程框架的作业,与之对应多种数值含义的资源配额,但是数值含义对资源分配算法是透明的,仅需横向扩展资源分配算法即可。

本文针对企业私有云环境下的高性能计算,提出了一种基于用户资源配额的资源弹性分配算法(quota-based resource elastic allocation, QREA)。其基本思想是在集群计算资源充足时,通过将空闲资源追加给弹性作业以提高资源利用率;在资源紧张时,通过资源回收以保障用户的资源配额。弹性作业是资源分配的基本单元,其资源请求数处于一个区间,作业可以在获得此区间的任一资源数时运行。由于弹性作业具有高效、灵活、可扩展的优点,是高性能计算中并行计算框架的常规计算模式。作为并行编程的事实标准 MPI[3],其接口库早已支持动态地增加和删除并行处理的进程和结点。在进行资源追加和回收时,为提高多用户共享集群资源的公平性,提出了基于用户历史信用值的资源追加和回收算法,考虑用户历史资源的使用情况,而不仅仅是当前资源的使用情况。

1 相关工作

对于传统的高性能计算,资源分配的目标是提高集群资源的利用率,通过多用户共享整个集群资源来实现。经典的算法有先来先服务(FCFS)、短作业优先(SJF)、首次适应(First-Fit)、回填算法(Back-fill)等,由于以上算法均没有考虑多用户的场景,因此无法为用户提供明确的计算配额保障。考虑到多用户的服务质量,常用的解决方法是为每个用户分配私有的计算集群,这些资源能够满足用户峰值计算的需求,但是用户的作业负载是波动的,不会一直处于峰值状态,因此会造成资源的浪费,同时增加了管理多个集群的负担。

云计算通过资源虚拟化、容器等技术,对资源进行统一管理,解决了管理多个集群的负担。但是,云计算的运营模式是按需付费,要求同时保证资源提供者和消费者的双方利益[9],具有严格的资源配额限定,即使有空闲资源也不会将其分配给消费者,面临资源利用率低的问题。本文将这种资源分配算法称为基本资源分配算法(basic resource allocation, BRA)。

为保障用户公平共享集群资源,Facebook 的公平调度算法 Fair Scheduler[10] 和 Yahoo 的容量调度算法 Capacity Scheduler[11] 被广泛用于多用户环境下的资源分配算法中。Fair Scheduler 将作业按照用户(或其他属性)放到对应的作业池,每个作业池具有最小资源配额。集群剩余资源以尽量公平的方式分配给用户。当最小资源没有满足且超过等待时间时,调度系统会通过中止其他作业池的作业任务来抢占资源。与 Fair Scheduler 相比,当最小资源没有满足时,Capacity Scheduler 会把接下来作业运行结束释放的资源优先分配给该用户。对于高性能计算作业,作业运行时间长且运行时间不定,Fair Scheduler 的中止任务的抢占方式会导致作业运行失败,造成计算的浪费。而 Capacity Scheduler 的等待作业运行完再优先分配的方式,不能保障用户的服务质量。文献[8]中,针对多用户的环境,提出了一种动态资源分配算法,通过分析用户的历史执行信息得到各个用户资源需求随时间变化的规律,要求用户提交的作业具有一定的规律,而大部分实际场景下,用户请求是自相似的[12]。所谓自相似是指即使在很长的时间跨度范围内也不会存在前后相似的用户请求负载。

综上所述,已有的资源分配算法虽然考虑了集群资源利用率和多用户环境,但是均没充分考虑企业私有云环境下高性能计算的特性以及带来的挑战。因此,亟需一个更高效的资源分配算法,在保障用户拥有明确计算能力的同时,尽可能地提高集群资源利用率。表 1 给出了本文需要使用到的符号的

基本定义。

表 1　基本定义

符号	定义
$\boldsymbol{E}$	计算资源,$\boldsymbol{E}=(\boldsymbol{e}_1,\boldsymbol{e}_2,\cdots,\boldsymbol{e}_N)$,$\boldsymbol{e}_i$ 表示计算单元 i。
$\boldsymbol{U}$	用户,$\boldsymbol{U}=(\boldsymbol{u}_1,\boldsymbol{u}_2,\cdots,\boldsymbol{u}_N)$,$\boldsymbol{u}_i$ 表示用户 i。
$\boldsymbol{J}$	作业,$\boldsymbol{J}=(\boldsymbol{J}_1,\boldsymbol{J}_2,\cdots,\boldsymbol{J}_N)$,$\boldsymbol{J}_i$ 表示用户 i 的所有排队作业,$\boldsymbol{J}_i=(\boldsymbol{j}_1,\boldsymbol{j}_2,\cdots,\boldsymbol{j}_m)$。
$\boldsymbol{R}$	所有用户作业的资源请求,$\boldsymbol{R}=(\boldsymbol{R}_1,\boldsymbol{R}_2,\cdots,\boldsymbol{R}_N)$,$\boldsymbol{R}_i$ 表示用户 i 的所有作业的资源请求 $\boldsymbol{R}_i=(\boldsymbol{r}_1,\boldsymbol{r}_2,\cdots,\boldsymbol{r}_m)$。
r	作业的资源请求,$r=\{\alpha,\beta,\lambda\}$,分别表示作业的资源请求的基本值,最大值和实际值。
Q	所有用户的资源配额集合,$Q=(q_1,q_2,\cdots,q_N)$,q_i 表示用户 i 的资源配额。
q	资源配额,$q=\{\tau,\mu\}$,分别表示配额和已用配额。
C	用户的信用值集合,$C=(c_1,c_2,\cdots,c_N)$,c_i 表示用户 i 的信用值。

2　QREA 算法

2.1　QREA 算法原理

QREA 算法原理如图 1 所示,其基本原理如下:

(1) 系统为用户设定资源配额,资源配额与物理计算资源没有固定的对应关系。

(2) 用户提交的弹性作业(资源请求数[α,β])被分发到对应的用户作业队列。

(3) ERAM 算法为用户作业队列中的作业分配所需的基本资源 α,并计入用户已使用配额 μ。如果用户已用资源配额超过其配额($\mu>\tau$),则停止为该用户的排队作业分配资源。

(4)如果有空闲计算资源,QREA 算法将对未达到资源请求最大值的作业进行资源追加,追加的资源不计入用户已用配额,这是因为追加资源可以在任意时刻被回收。

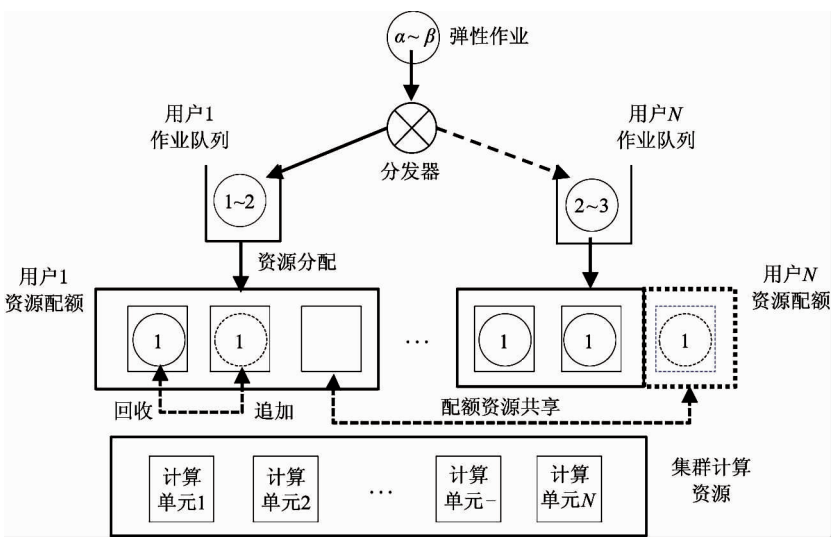


图 1　QREA 算法原理图

(5)如果用户资源配额没有满足,ERAM 算法将回收已追加的资源,回收并不会引起作业的运行失败,这是因为弹性作业支持资源的动态伸缩。QREA 的算法描述如算法 1。

算法 1 资源弹性分配算法 QREA

输入:计算资源 E,用户 U,作业 J,资源请求 R,用户配额 Q

输出:资源分配决策 *AllocDecs*,资源追加决策 *FurtherDecs*,资源回收决策 *LimitDecs*

```
1. userPri = sortUserByRatio(U)
2. for u in userPri
3.    jobPri = sortJob (J_u)
4.    for j in jobPri
5.       if q_u.τ >= q_u.μ + R_u[j].α
6.          idleE = countIdleE(E)
```

```
7.        if idleE > = R_u[j]. α
8.          dec = < j, R_u[j]. α >
9.          AllocDecs. insert( dec)
10.       else
11.       decs = LimitR( R_u[j]. α-idleE, E, U, J, R)
12.       LimitDecs. insert( decs)
13. idleE = countIdleE( E)
14. if idleE >0
15.   decs = FurtherR( E, U, J, R)
16.   FurtherDecs. insert( decs)
```

其中,sortUserByRatio($\boldsymbol{U}$)是对用户按照其配额利用率(μ/τ)排序,利用率越低优先级越高。默认情况下 sortJob (J_u)是对用户的排队作业按照提交的先后进行排序,即先来先服务(FCFS)。在初次为作业分配计算资源时,分配作业所需的基本资源数 $\boldsymbol{R}_u[j].\alpha$,如果空闲资源不能满足作业的资源请求,则通过资源回收算法 $LimitR(\boldsymbol{R}_u[j].\alpha\text{-idleE}, \boldsymbol{U}, \boldsymbol{J}, \boldsymbol{R})$回收 $\boldsymbol{R}_u[j].\alpha$-idleE 个计算资源。当有空闲计算资源时,通过资源追加算法 $FurtherR(\boldsymbol{E}, \boldsymbol{U}, \boldsymbol{J}, \boldsymbol{R})$进行资源追加。QREA 算法的时间复杂度为 $O(N)$,N 为全部排队的作业数。

2.2 基于信用值的资源追加和回收算法

在进行资源追加和资源回收时,选择哪个用户的作业进行追加和回收将直接影响多用户资源共享的公平性。常用的方法是根据用户当前已追加资源的数量,优先追加给已追加资源最少的用户,优先回收已追加资源最多的用户。如在 Fair Scheduler 算法中资源的共享和抢占策略都是根据用户当前资源的使用情况。

但是,以上方法未考虑用户对资源的历史使用情况,造成多用户共享资源的不公平,以及用户对资源的“欺骗”使用。例如,如果某用户申请的资源配额小于其实际的作业负载,但是在系统运行期间的任意时刻该用户与其他用户平等地获得共享资源。从整个运行期间考虑,该用户获得了更多的共享资源,因此存在用户“欺骗”申请资源配额的情况。

为提高多用户共享资源的公平性,本文提出了基于信用值的资源追加和回收算法,信用值是指用户共享自身配额资源而获得的收益与用户占用空闲资源的支出的差值。用户的信用值越大,被追加资源的优先级越高,被回收资源的优先级越低。此外,通过维护用户的信用值,可以为用户提供合理的资源配额参考。例如,如果用户的信用值不断地增加,则说明该用户申请的配额超过实际的作业负载,因此可减少申请资源配额,反之增加。t 时刻用户 i 的信用值计算公式如下:

$$c_i(0) = 0 \tag{1}$$

$$c_i(t) = c_i(t - T) + \Delta c_i \tag{2}$$

$$\Delta c_i = profit_i + expend_i \tag{3}$$

$$profit_i = \theta_i \times \sum_{u=0}^{u=N} \sum_{j=0}^{j=M} (R_u.r_j.\lambda - R_u.r_j.\alpha) \tag{4}$$

$$expend_i = \sum_{j=0}^{j=M} (R_i.r_j.\lambda - R_i.r_j.\alpha) \tag{5}$$

$$\theta_i = (q_i.\tau - q_i.\mu) / \sum_{u=0}^{u=N} (q_u.\tau - q_u.\mu) \tag{6}$$

其中,T 为计算周期,Δc_i 是一个周期中用户 i 的信用值的变化量。式(4)是计算用户 i 的信用收益 $profit_i$,其中 θ_i 是用户 i 共享配额资源的比例,第二项表示所有用户的信用支出之和;在式(6)中,分子表示用户 i 共享的资源配额数,分母表示所有用户共享的资源配额数。$expend_i$ 是用户 i 的信用支出,是用户所有正在运行的作业被追加的资源之和。

基于信用值的资源追加算法具体见算法 2。

算法 2 资源追加算法 $FurtherR(\boldsymbol{E}, \boldsymbol{U}, \boldsymbol{J}, \boldsymbol{R})$

输入:计算资源 $\boldsymbol{E}$,用户 $\boldsymbol{U}$,作业 $\boldsymbol{J}$,资源请求 $\boldsymbol{R}$,用户信用值 $\boldsymbol{C}$

输出:资源追加决策 *Decs*

```
1. userPri = sortUserByCredit( C)
2. for u in userPri
3.   jobPri = sortJob( J_u)
4.   for j in jobPri
5.     Δ = R_u[j]. β-R_u[j]. λ
6.     if Δ >0
7.       idleE = countIdleE( E)
8.       if idleE > = Δ
9.         dec = < j, Δ >
10.      else
11.        dec = <j, idleE >
12.      Decs. insert( dec)
```

其中,sortUserByCredit($\boldsymbol{C}$)是对用户按照信用值

进行排序,信用值越大追加的优先级越高。默认情况下 sortJob (J_u)是对用户 u 正在运行的作业按照作业提交的先后顺序进行排序,优先追加先提交的作业。资源追加算法的时间复杂度为 $O(N)$,N 为全部运行作业数。

基于信用值的资源回收算法见算法 3。

算法 3 资源回收算法 *LimitR*(n, U, J, R)

输入:回收的资源数 n,计算资源 E,用户 U,作业 J,资源请求 R,用户信用值 C

输出:资源回收决策 *Decs*

```
1.  δ = n
2.  userPri = sortUserByCredit (C)
3.  for u in userPri
4.    jobPri = sortJob (J_u)
5.    for j in jobPri
6.      Δ = R_u[j].λ - R_u[j].α
7.      if Δ > =δ
8.        dec = < j, δ >
9.        δ = 0
10.         Decs.insert(dec)
11.         break
12.       else if Δ >0
13.         dec = < j, Δ >
14.         δ - = Δ
15.         Decs.insert(dec)
16.   if δ = =0
17.     break
```

其中,sortUserByCredit(C)是对用户按照信用值进行排序,信用值越小优先级越高。sortJob(J_u)是对用户 u 的正在运行的作业按照作业提交的先后顺序进行排序,优先回收后提交的作业。对于一个回收作业,首先计算其资源请求的实际值 λ 与基本值 α 的差值 Δ(第 6 行)。如果 Δ 大于预回收的资源数 δ,则回收 δ 个资源(第 8 行);如果 Δ 大于 0,则回收 Δ 个资源(第 13 行)。资源回收算法的时间复杂度为 $O(N)$,N 为全部运行作业数。

3 实验结果与分析

为评价 QREA 算法的有效性,本文实现了多用户公平调度算法 Fair Scheduler、基本资源分配算法 BRA 和基于当前资源使用情况的资源弹性分配算法(current-based resource elastic allocation, CREA),并在集群资源利用率、作业平均完成时间、作业失败率、用户公平性 4 个性能指标上进行了对比。本文中集群资源利用率是指所有计算单元的平均作业占用率,对于一个计算节点来说如果在系统运行过程中一直有作业运行,则该节点的利用率为 100%。其中,BRA 算法是指各用户具有严格的资源配额限制,在用户的配额范围内进行资源分配。与 QREA 算法相比,在 CREA 算法中,资源追加和回收根据用户当前已追加的资源数。

对于一个弹性作业,被分配的计算资源数是可变的,作业的完成时间与计算资源数成反比。理论上,弹性作业的资源请求的最大值 β 是没有上限的,一个作业可以在所有可用的计算资源中并行计算,这种资源分配方式可以最大程度地利用集群计算资源。但实际中,受到数据依赖与同步、作业扩展性与稳定性的矛盾等影响,作业的资源请求范围是经过大量的测试实验得到的,如何设置合理的资源请求范围不是本文的研究点。

集群系统能够并行处理作业的数量是衡量其作业处理能力的重要标准,也是用户服务质量的核心指标,因此初次对弹性作业进行资源分配时,默认分配其资源请求的基本值。例如,一个用户的资源配额是 50,作业的资源请求数是[1,2],那么用户在配额范围内可以同时运行的最大作业数为 50。对于 BRA 和 Fair Scheduler 算法,在作业的整个运行期间,实际被分配的资源数是固定的 α,对于 QREA 算法和 CREA 算法,是弹性伸缩的(α ~ β)。

实验平台中包含 3 种角色:客户端、资源分配器和计算单元。客户端用于产生用户的作业请求;资源分配器为作业分配计算单元并将作业下发到对应的计算单元;计算单元运行作业并向分配器汇报运行情况。定义作业的计算量为一个计算单元完成作业计算任务的用时。计算单元周期性(默认值为 1s)地向资源分配器发送作业心跳,当分配器收到的作业心跳数之和等于作业的计算量时表示作业运行完成。用户作业请求由合成负载产生器 FGN[13] 产生,FGN 是 Berkeley 大学开发的,被广泛用于产生

符合自相似特性的用户请求[14,15]。

为对比 QREA、BRA、CREA 和 Fair Scheduler 算法在集群资源利用率、作业平均完成时间和作业失败率三个性能指标上的表现，设计了实验 1，实验参数如表 2 所示。

表 2　实验 1 的实验参数

	数量	运行节点	测试集
用户数	4		
用户的资源配额	50		
集群计算单元	200	1Core 2.4GHz，1GB 内存，Centos6.5	
客户端	1	1Core 2.4GHz，1GB 内存，Centos6.5	FGN
资源分配器	1	1Core 2.4GHz，2GB 内存，Centos6.5	

为避免不同用户配额对以上三个性能指标的影响，设定 4 个用户具有相同的资源配额。弹性作业的资源请求数是[1，2]，计算量为 10。实验中对各算法在不同作业请求率下的性能表现进行了测试。其中，对于某一作业请求率，负载产生器 FGN 为每个用户产生自相似的作业请求负载测试集，Hurst 参数为0.89，测试集包含 100 个样本点，每个样本点包含某一用户的多个作业请求，样本点的平均值为作业请求率。图 2 显示了平均请求率为 5 请求/s，4 个用户的作业请求负载情况。

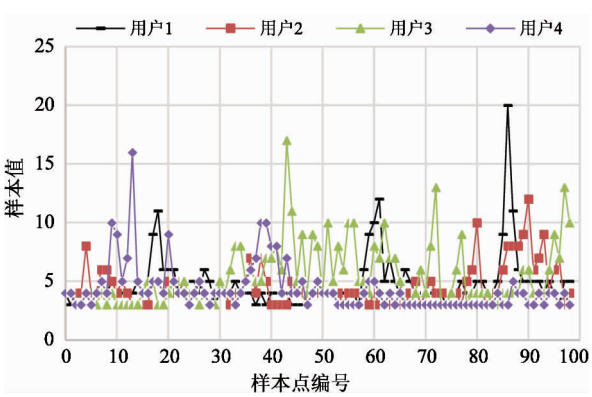
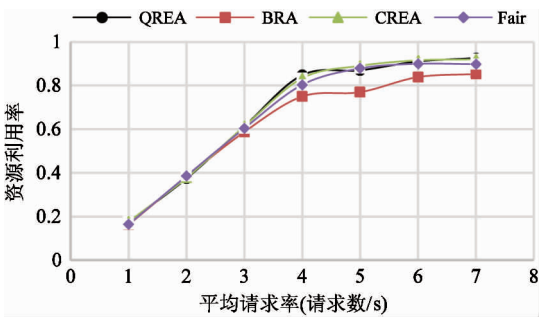


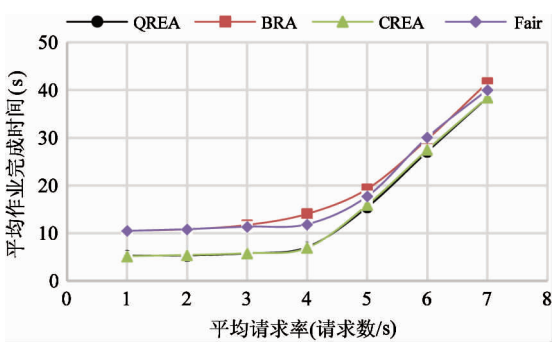
图 2　作业请求负载

从图 2 中可以看出，样本点的值是无规律、自相似的，样本值围绕作业的平均请求率上下波动。三个性能指标的实验统计结果如图 3 所示。

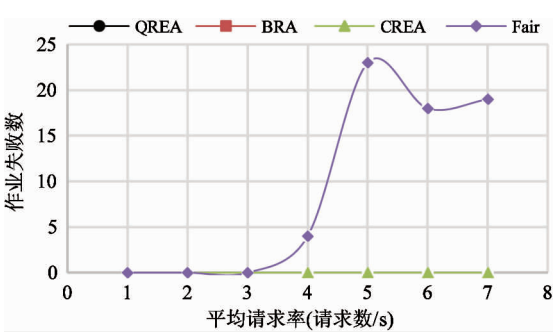
如图 3(a)所示，在平均请求率小于 3 请求/s 时，4 个算法具有基本相同的资源利用率，这是由于作业请求负载较小，用户的资源配额足以满足作业的资源需求。随着请求率的增加，BRA 算法的利用



(a) 资源利用率



(b) 作业平均完成时间



(c) 失败作业数

图 3　不同请求率下的性能表现

率低于其他算法,这是由于 BRA 算法不支持资源共享造成的。在系统运行期间,有的用户资源受限,而有的用户资源空闲。

在图 3(b)中,随着作业请求率的增加作业的平均完成时间不断增加。QREA 算法和 CREA 算法具有基本相同的作业平均完成时间,在请求率小于 4 请求/s 时,空闲资源被追加给弹性作业,因此其作业平均完成时间缩短 50%。当请求率超过 4 请求/s,随着请求率的增加 4 个算法的作业平均完成时间差距越来越小,这是由于随着作业负载的增加,空闲资源越来越少,导致弹性伸缩的程度随之下降。

如图 3(c)所示,Fair 算法会导致作业运行失败,这是由于该算法通过杀死作业来抢占计算资源导致的。当作业请求率较小时,集群计算资源可以满足所有用户作业对资源的需求,因此失败作业数很小,几乎为零。

为评价 QREA 算法在多用户环境下资源共享的公平性,设计了实验 2,将 QREA 算法和 CREA 算法进行了对比。在实验 2 中,用户 1 的作业请求率从小到大变化,而其它 3 个用户的作业请求率为 4 请求/s。其他实验参数与实验 1 相同。定义非公平性指数 φ 作为公平性的评价指标,其反映了各用户信用值的偏离程度,计算公式如下:

$$\varphi = \sum_{i=0}^{i=N} \left(c_i - \frac{\sum_{u=0}^{u=N} | c_u |}{N} \right)^2 \tag{7}$$

式(7)中,用户数为 N,c_i 表示用户 i 的信用值,$|c_u|$是对 c_u 取绝对值。非公平指数的实验结果如图 4 所示。

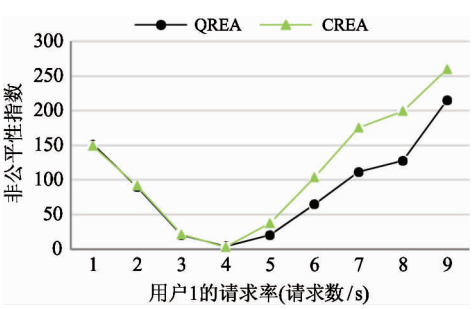


图 4　非公平性指数

如图 4 所示,当用户 1 的请求率为 4 请求/s 时,非公平性指数最小,这是因为此时 4 个用户的资源配额和作业的请求率都是相同的,所有用户公平地共享集群资源。当用户 1 的请求率偏离 4 请求/s 时,偏离度越大非公平性指数越大。当用户 1 的请求率小于 5 请求/s 时,集群资源可以满足所有弹性作业资源请求的最大值 β,两种算法的非公平指数基本相同。当作业负载增大,集群资源紧张时,QREA 算法的非公平性指数小于 CREA 算法,当平均作业请求率处于 5 ~9 请求/s 时,非公平性指数平均下降 34.5%。

4　结　论

在多用户共享集群资源的企业私有云环境下,高性能计算面临提高资源利用率和保障用户计算能力的挑战,为此本文提出了一种基于用户资源配额的资源弹性分配算法 QREA。该算法能够自适应作业负载的变化,通过动态地追加和回收计算资源,在保障用户资源配额的基础上,提高了集群资源利用率,缩短了作业的平均完成时间。而且基于用户历史信用值的资源追加和回收算法,有效地提高了多用户共享集群资源的公平性。大量的实验结果表明,在低作业负载下,QREA 算法与 BRA 算法和 Fair Scheduler 算法相比,作业平均完成时间缩短 50%;在高作业负载下,其比 BRA 算法在资源利用率上提高 10%,在多用户公平性上,比 CREA 算法提高 34.5%。

参考文献

[1] 洪文杰, 李肯立, 全哲,等. 面向神威·太湖之光的 PETSc 可扩展异构并行算法及其性能优化[J]. 计算机学报, 2017, 40(9):2057-2069

[2] 罗萱, 林新华, 金耀辉. HPC 与云融合之道[J]. 中国教育网络, 2011, (9):19-21

[3] 王国峰, 刘川意, 潘鹤中,等. 云计算模式内部威胁综述[J]. 计算机学报, 2017, 40(2):296-316

[4] 张玉清, 王晓菲, 刘雪峰,等. 云计算环境安全综述[J]. 软件学报, 2016, (6):1328-1348

[5] 焦毅, 李琳, 王颖慧,等. 一种面向企业私有云的数据分布策略[J]. 计算机研究与发展, 2011, 48(1):239-244

[6] 曹宏嘉, 卢宇彤, 谢旻,等. 并行作业启动及其可扩展性分析[J]. 计算机研究与发展, 2013, 50(8):1755-1761

[7] 魏豪, 周抒睿, 张锐,等. 基于应用特征的 PaaS 弹性资源管理机制[J]. 计算机学报, 2016, 39(2):223-236

[8] 陈重韬. 面向多用户环境的 MapReduce 集群调度算法研究[J]. 高技术通讯, 2017, 27(4):295-302

[9] 苑迎, 王翠荣, 王聪,等. 基于非完全信息博弈的云资源分配模型[J]. 计算机研究与发展, 2016, 53(6):1342-1351

[10] Fair Scheduler [EB/OL]. http://hadoop. apache. org/docs/ r2. 7. 2/hadoop-yarn/hadoop-yarn-site/FairScheduler. html:2016

[11] Capacity Scheduler [EB/OL]. http://hadoop. apache. org/docs /r2. 7. 1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler. html:2016

[12] Crovella M, Bestavros A. Explaining World Wide Web Traffic Self-Similarity[M]. Boston: Boston University, 1995

[13] Paxson V. Fast approximation of self-Similar network traffic[J]. *ACM SIGCOMM Computer Communication Review*, 1997,27(5): 5-18

[14] Xiao L J. Performance analysis of priority scheduling mechanisms under heterogeneous network traffic [J]. *Journal of Computer and System Sciences*, 2007,73(8): 1207-1220

[15] Silva R A C D, Fonseca N L S D. Topology-aware virtual machine placement in data centers[J]. *Journal of Grid Computing*, 2016,14(1):75-90

Quota-based resource elastic allocation algorithm for high performance computing under private cloud

Liu Xiaodong[*,**], Zhao Xiaofang[**], Jin Yan[**], Luo Gang[***], Chen Yajing[***], Zhao Shuguang[*,**]

([*] University of Chinese Academy of Sciences, Beijing 100049)

([**] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***] Geophysical Technique Research Center, BGP, CNPC, Zhuozhou 072751)

Abstract

In order to address the challenge of improving resource utilization and guaranteeing the computing power of users for high performance computing under private cloud, a quota-based resource elastic allocation algorithm, called QREA, is proposed in this paper. When the computing resources are sufficient, the algorithm furthers idle resources to elastic jobs so as to improve resource utilization. When resource is constrained, the algorithm limits furthered resources to satisfy the quota of user. In addition, for the purpose of improving user fairness of sharing cluster resource, the policies of furthering and reducing resource are based on historical credit of users. The experimental results indicate that under low job workload, the average completion time of jobs in QREA decreases 50% compared with the basic resource allocation algorithm (BRA) and Hadoop Fair Scheduler. In high job workload situation, QREA improves the resource utilization by 10% than BRA, and increases user fairness by 34.5% than the current-based resource elastic allocation algorithm (CREA).

Key words: resource elastic allocation, private cloud, high performance computing, resource utilization, multi-user environment