面向多用户的集群资源部署策略

王明阳 洪 爵 冯圣中

（中国科学院深圳先进技术研究院 深圳 518055）

摘 要 集群系统中面向多用户的资源部署策略优先考虑的是资源能够在不同用户之间进行公平分配，同时实现用户之间的资源共享。为了充分利用集群资源，用户以一定的原则通常是采用权重比的方式共用集群资源。该策略并不关心每个用户所获取的资源总量，只关心用户所获取的资源比例是否总是符合原则，这样可以保证空闲资源总是能够被充分利用。该策略包括资源的定额分配和自主申请两部分，以满足用户的需求。同时提出了采用用户占用资源量与占用时间共同作为公平分配依据的策略，以适应更加复杂的情况。

关键词 集群；多用户；公平；资源共享

Resource Deployment Strategy for Multi-User Cluster

WANG Ming-yang HONG Jue FENG Sheng-zhong

(*Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen* 518055, *China*)

Abstract This resource deployment strategy for multi-user cluster gives priority to the fair allocation of resources between different users, while achieving resource sharing. In order to make full use of cluster resources, certain principles are used as weights for sharing cluster resources. It cares about whether the resources ratio which each user gets is in line with the principle, but not the total amount of resources, thus can ensure that the idle resources are utilized fully. The strategy consists of fixed allocation and independent application to meet users' need. Then we propose the allocating pattern based on resources and taken time together to adapt to the more complex situation.

Keywords cluster; multi-user; fair; resource sharing

1 引 言

随着计算技术的发展，对于集群系统的需求和应用日益重要。人们对于计算资源的需求总是无止境的，而科学技术的发展于此也显得捉襟见肘。在计算资源还不能很好地满足需求的时候，对于计算资源的合理调度即成为缓解这一压力的重要手段。

有鉴于Apache Mesos[1]的调度系统采用双层调度架构：上层调度器与计算框架。它允许不同的计算框架（Framework）接入，比如Hadoop、MPI、Hypertable、Shark等，这些计算框架即为其调度对象，由上层调度器为每个计算框架部署其所需的计算资源，再由计算框架自行为其内部的任务部署资源。此种方法具有极好的自由度，既能够让每个计算框架对其所提交的任务采用合适的调度策略，很好地满足每个计算的各自的不同需求，也能够对计算资源进行全局的管理。很多集群资源调度系统都延续了Mesos的这种双层调度架构，比如Hadoop[2]的YARN[3](原MapReduce)即是采用了这种结构。随着计算技术特别是云计算技术的发展，集群资源调度系统为了应对纷繁叠出的计算应用的多样化需求，双层调度架构乃至多层调度架构将成为未来集群资源调度系统的首要选择。特别是对于云计算平台，对计算资源的合理调度也是一项重要的研究课题。尤其是面向多样的用户和计算应用，如何很好地满足需求，同时能够进行合理调配，成为调度领域的重点。

本文设计并实现了一种在面向多个用户的集群系

作者简介：王明阳，硕士研究生，研究方向为高性能计算，E-mail: my.wang@siat.ac.cn；洪爵，助理研究员，研究方向为云计算、高性能计算；冯圣中，研究员，研究方向为高性能计算、网格计算、生物信息学。

统中进行资源部署的策略。其中的用户仅是一个概念，可以理解为计算框架、用户组等其它调度对象。考虑到能够充分利用有限的计算资源，在我们的集群调度系统中，并没有为每个用户划定固定使用的机器（即子集群），而是采用占用集群资源比例的方式，从而保证整个集群能够具备较高的资源利用率。而同时为了保证用户的实际占用资源符合其申请的比例，并且在用户组间进行资源调配，选择哪几个用户部署任务运行变得十分重要。基于这种思路我们提出了面向多用户集群系统的资源部署策略，可以称其为序列部署策略。

2　集群资源部署策略

根据现实需求，我们的资源部署策略分为两个部分：资源定额分配方式和任务自主申请的方式。资源可以是单一的资源类型如CPU、内存等，也可以是资源向量如<CPU，内存>。定额分配方式是指整个集群对所有任务都分配相同的计算资源，而在任务自主申请方式中任务可以自主申请其所需的计算资源。

2.1　资源定额分配方式

（1）用户部署顺序的确定

我们采用最大缺额的调度策略[4]为每个用户部署资源。考虑到集群中的用户数量较少，并且用户申请的资源量一般是固定的，因此事先将部署顺序记录下来，则可以在心跳信息到来时直接选取部署任务的用户。我们采用如下的策略确定用户的部署顺序：

①最大缺额方式：

有n个用户U_i，$0<i\le n$，每个用户所申请的资源量为R_i，其中R_i为正整数，假定总有$R_i\ge R_{(i+1)}$。按照以下方法进行缺额计算，令每个用户当前的可用资源量为$r_i=R_i$，其中$0\le r_i\le R_i$，分别对每个用户计算缺额$d_i=\dfrac{r_i-1}{R_i}$，然后选取$d_j=max_{0<j\le n}d_i$，如果存在多个d_j，则选取最大的j，令$r_j=r_j-1$，并将其用户标识U_j放到数组*SeqArray*中存储，数组大小即用户申请资源总量$R_{total}=\sum_{0<i<n}R_i$。重复以上过程，直至所有的$r_i$均为0。

②最小比例方式：

按照以下方法进行计算，令每个用户当前的已分配资源量为$r_i=0$，其中$0\le r_i\le R_i$，分别对每个用户计算当前已分配比例$b_i=\dfrac{r_i}{R_i}$，然后选取$b_j=\min_{(0<j\le n)}b_i$，如果存在多个b_j，则选取最小的j，令$r_j=r_j+1$，并将其用户标识U_j放到数组*SeqArray*中存储，数组大小即用户申请资源总量$R_{total}=\sum_{(0<i<n)}R_i$。重复以上过程，直至所有的$r_i=R_i$。

以上两种方式是等价的。

为了在心跳信息到来时选择可以部署的用户，并确定可用资源量的数目，我们记录用户资源总量中已用部分的变量R_{used}，显然有$0\le R_{used}\le R_{total}$。当有心跳到来显示本次可以运行的任务数为$c$，则应当在数组*SeqArray*中从$R_{used}+1$这个位置，依次取出用户标识$U_i$，并统计其相应的个数，直到取出$c$个任务。每取出一个任务令$R_{used}=R_{used}+1$，若$R_{used}==R_{total}$，则重新开始计数令$R_{used}=0$。

每当有新的用户添加时，需要对用户的部署顺序进行重新分配，为了不破坏资源分配的公平性，我们将新加入的用户的此次轮询的可用资源量按照当前资源分配比例添加，并重新制定部署顺序。采用如下的策略：现在要加入一个新的用户为U_{new}，其申请的资源量为R_{new}，为了满足部署顺序的公平性，则应当使这个新用户的已用资源量为$R'_{new}=\left\lfloor\dfrac{R_{used}}{R_{total}}*R_{new}+0.5\right\rfloor$，并将新的用户加入队列中，得到$R_{total}=R_{total}+R_{new}$及$R_{used}=R_{used}+R'_{new}$。根据最大缺额的部署策略可知，当重新计算用户的部署顺序后，新加入的用户能够很好地适应该序列，且此前用户的资源部署比例不会发生变化。

同样当有用户需要注销时，也要对用户的部署顺序进行重新分配，为了不破坏资源分配的公平性，我们将注销的用户的可用资源量按照当前资源分配比例删除，并重新制定部署顺序。采用如下的策略：现在要注销一个用户为U_{del}，其申请的资源量为R_{del}，为了满足部署顺序的公平性，则应当使这个用户的已用资源量为$R'_{del}=\left\lfloor\dfrac{R_{used}}{R_{total}}*R_{del}+0.5\right\rfloor$，并将该用户从队列中删除，得到$R_{total}=R_{total}-R_{del}$及$R_{used}=R_{used}-R'_{del}$。根据最大缺额的部署策略可知，当重新计算用户的部署顺序后，剩余的用户的资源部署比例不会发生变化。

如有用户A和B，其资源量分别为5和3，按照规则可以得到如下部署顺序：ABAABABA。假定每次可以部署5个任务，会得到表1所示的部署顺序。

经过分析，采用这种部署顺序即能保证计算资源按照用户的需求按照比例进行部署，而且总是能够得到准确值，并且在总体上可以保证计算资源分配的

表1 用户部署顺序示例

次数	部署顺序	实际得到任务总数		按照比例应得（四舍五入）	
		A	B	A(总数*5/8)	B(总数*3/8)
1	ABAAB	3	2	3	2
2	ABA|AB	6	4	6	4
3	AABAB	9	6	9	6
4	A|ABAA	13	7	13	7(五入为 8)
5	BABA|A	16	9	16	9
6	BAABA	19	11	19	11
7	BA|ABA	22	13	22	13
8	ABABA	25	15	25	15

公平性。另外能够避免在多个用户存在的情况下，不利于确定部署顺序的缺陷。比如有三个用户U_k、U_m、U_n，其占用资源量分别为3、1、1，假定每次部署2个任务，则三者所得资源量分别为1.2、0.4、0.4，U_k可得到其中1个，而对于另外一个则需要额外的处理来决定，并且每个用户要记录他的已获取任务数量。总的来说，最大缺额的调度策略能够有很好的调度效果。

（2）用户间的资源共享

有n个用户U_i,$0<i\leq n$，每个用户所申请的资源量为R_i，为了提高资源利用的效率，用户中的计算资源可以提供给其它用户使用，即当某个用户中没有可部署的任务，且其计算资源有所剩余，则可以提供给需要的用户，并打下欠条。例如用户U_t借到U_s的一部分资源，资源量为r，形如$U_t \xleftarrow{r} U_s$。当然此种借贷并非对于计算资源的实际交换，只是一张凭证，同样也没有利息。对于借入资源的用户，遵循诚信原则，当债务用户需要资源时即归还所借贷的资源，同时当其存在空闲资源时归还所借贷的资源。

为了避免环形债务的出现，采用债务转移的方式，使得用户仅有债务的借入或借出，抑或是没有与其他用户的借贷关系。如三个用户U_k、U_m、U_n，存在$U_k \xleftarrow{r_1} U_m$和$U_m \xleftarrow{r_2} U_n$，而当出现时$U_k \xleftarrow{r_3} U_n$，即可将$U_k \xleftarrow{r_3} U_n$这部分进行转移。会出现三种情况：①当$r_3<r_2$ 时，可以将r_3全部转移，则有$U_k \xleftarrow{r_1+r_3} U_m$和$U_k \xrightarrow{r_2-r_3} U_m$，此时$U_k$与$U_n$之间不存在借贷关系；②当$r_3=r_2$时，可以将$r_3$全部转移，则有$U_k \xleftarrow{r_1+r_3} U_m$，此时$U_k$与$U_n$之间、$U_m$与$U_n$之间均不存在借贷关系；③当$r_3>r_2$时，则只转移$r_2$的资源量，则有$U_k \xleftarrow{r_1+r_2} U_m$和$U_k \xleftarrow{r_3-r_2} U_n$，而此时$U_m$与$U_n$之间的借贷关系解除。

某个借出资源的用户需要这些资源时，会出现三种情况：

A）此时借贷用户具有空闲资源，则可以偿还其借贷的资源。例如用户U_m借到U_n的一部分资源，资源量为r，形如$U_m \xleftarrow{r} U_n$，当U_n需要这些资源，而U_m此时有空闲资源，且资源量为f，此时存在两种情况：①当$r\leq f$时，U_n可以取回所有资源，并解除与U_m之间的借贷关系；②而当$r>f$时，U_n仅能取回一部分资源，即f，同时消除相应的亏欠额度，此时有$U_m \xleftarrow{r-f} U_n$。

B）如果此时资源池中还有剩余资源，即其他的用户内存在空闲的资源时，则可以使用这些资源，同时将所造成的债务转移到它此前的借贷用户身上，例如用户U_m借到U_n的一部分资源，资源量为r，形如$U_m \xleftarrow{r} U_n$，当U_n需要这些资源，而U_m此时没有空闲资源，恰巧此时U_k有空闲资源，资源量为f，则U_n将使用U_k的资源，同时将U_m对U_n的欠条转移给U_k，也即此时U_m与U_k之间存在借贷关系。此时存在两种情况：①当$r\leq f$时，U_n可以取回所有资源，并将U_m的全部亏欠额度转移给U_k，则此时有$U_m \xleftarrow{r} U_k$；②而当$r>f$时，U_n仅能取回一部分资源，即f，同时将相应的亏欠额度交给U_k，此时有$U_m \xleftarrow{r-f} U_n$和$U_m \xleftarrow{f} U_k$。

C）而当不存在空闲资源时，同时适用于A）中情况②在完成资源交换后，此时可以向借贷用户强行索要资源，然而这么做必定会造成资源损耗。一般来说，我们并不提倡强制执行，对于不是十分紧要的任务（根据任务类型判定），允许借贷资源延期归还。在系统中，通常是将归还时间延长到下一次具有可用资源的机器心跳到来，这个时间一般在几十毫秒到1s之间，如此并不会为借贷用户造成太大损耗。

考虑到某个用户长时间内没有运行任务，那么他必定会有较大数量的贷出资源。而当其需要运行任务

时，会大量回收贷出资源，如此会导致向他借贷的用户长时间内还贷，从而无法运行新的任务，此种情形是应当避免的，考虑采用定额还贷的方式，即借贷用户每次仅返还其可用资源一定比例的资源。同样可以将该还贷方式推广到A）和B）中出现的情况，对于①，归还额度成为$r\times\mu$，而对于②，归还额度成为$f\times\mu$，其中$0<\mu\leq 1$。当然若是借贷用户并没有需要运行的任务，则可以将资源全部用于还贷。

每当有新的用户添加时，由于此前存在的用户资源申请量没有变化，所以不需要对先前用户之间的借贷关系进行处理。

如果用户注销，需要对该用户现有的借贷关系进行处理。借贷关系分为借出和借入两种，需要分别处理：①对于借出关系，此时注销用户被认为需要大量的资源，从而回收此前借出的资源，直至解除所有借贷关系，而回收的资源按照比例分配给当前所有的用户，可以理解为与该注销用户有关系借贷用户将其借贷关系分别转移到所有用户身上。对于要删除的用户U_d，存在$U_c \xleftarrow{r} U_d$，而此时要删除U_d，此时U_d的借出资源按照比例分配给其它所有用户。总的资源量$R_{total}=\sum_{(0<i<n)}R_i-R_d$，除去用户$U_d$之外的每个用户可以得到的资源量为 $R_i^{'}=\left\lfloor\frac{R_i}{R_{total}}*r\right\rfloor$，依次建立用户$U_i$与$U_c$之间的借贷关系，则有$U_c \xleftarrow{R_i^{'}} U_i$，并令$r=r-R'_i$；最后如果$r>0$则默认将剩余的那部分交给用户$U_c$，并以此删除$U_c$与$U_d$的借贷关系，然后删除用户$U_d$。②对于借入关系，与借出关系不同的是注销用户需要在还完贷款后才能从用户队列中删除，并清除其资源占用份额。与此前的还款方式不同的是此时将需要注销的用户所获取的资源全部用于偿还贷款。在注销用户完毕后需要重新制定剩余用户的部署顺序。

经过分析该调度策略能够很好地保证各个用户间资源分配量的公平性，并且能够很好地在各个用户间合理地共享计算资源。

2.2　任务自主申请资源量

（1）用户部署顺序的确定

对于自主申请资源量的任务部署问题是个NP问题，取得最优解是十分困难的，为了保证性能，获取近似最优解是常用的方法，此时往往伴随计算资源的浪费。既然浪费是不可避免的，我们采用对计算资源分块的方式，将任务所需计算资源整数化，从而能够更好地适应我们的模型。

对计算资源分块需要选取最小资源粒度。为了减少资源的浪费，最小资源粒度越精细效果越好，而此时对应的资源块数会比较大，反而会降低调度性能。比如申请的内存以M为单位，并且一般都在10 M以上，以0.1 M或者更小单位作为最小资源粒度虽然能够很好避免浪费，但是在此却不一定需要，因为浪费总是会发生的，而且因粒度划分所造成的份额只是很小的一部分。而采用1 M的资源粒度也会是不错的选择。

有n个用户U_i，$0<i\leq n$，每个用户所申请的资源量为R_i，总有$R_i\geq R_{i+1}$。假定最小资源粒度为R，则每个用户相应的资源块个数为$C_i=\frac{R_i}{R}$，一般我们需要选取适当的R，使得C_i为正整数，如此则可采用定额分配资源中的方式确认部署序列。

采用向量$<C_1,C_2,\cdots,C_n>$表示n个用户的部署序列，其中C_i表示用户U_i所获取的单位资源个数，且总为正整数。比如<2,1>可以表示A、B两个用户的部署序列，其中A、B申请的单位资源个数分别为2和1，则有<2,1>={ABA}。相应的可以证明存在

$$<mC_1,mC_2,\cdots,mC_n>$$
$$=\underbrace{<C_1,C_2,\cdots,C_n>+\cdots+<C_1,C_2,\cdots,C_n>}_{m}$$

，其中+表示连接，可以记作：

$<mC_1,mC_2,\cdots,mC_n>=m<C_1,C_2,\cdots,C_n>$，比如对于上例有<2×2,2×1>=<4,2>={ABAABA}={ABA}{ABA}=2{ABA}=2<2,1>。

因为在心跳到来时选取部署的用户并且确定相应资源块数量，采用的是循环计数方式。然而实际R_i通常为R的数十倍、数百倍或是更大的倍数，通过计算得到的C_i也往往是一个较大的整数，由于存放用户部署序列的数组SeqArray的大小为$C_{total}=\sum_{(0<i\leq n)}C_i$，会是一个很大的数组。同时机器中的可用资源也会是$R$的较大倍数，同样此时计算出的可用资源块个数也会是较大的整数，这样对于统计用户个数造成一定困难。

如果我们提取所有用户的资源块数量的最大公约数k，其中$C_i=kc_i$，则可以得到$<C_1,C_2,\cdots,C_n>=k<c_1,c_2,\cdots,c_n>$。而在$<C_1,C_2,\cdots,C_n>$中从位置$l$开始选取$c$个用户，等价于在$<c_1,c_2,\cdots,c_n>$中从位置$l'=l \bmod k$开始选取$c$个用户。又有在$<c_1,c_2,\cdots,c_n>$中从位置$l'=l \bmod k$开始选取c个用户，等价于先从$<c_1,c_2,\cdots,c_n>$中从位置l'=l mod k开始选取$c_{total}-l'$个用户，然后连续选取$\frac{c-(c_{total}-l')}{c_{total}}$个$<c_1,c_2,\cdots,c_n>$，接下来再选取

$$c-(c_{total}-l')-\left\lfloor\frac{c-(c_{total}-l')}{c_{total}}\right\rfloor*c_{total}$$

个用户。因为我们只是统计用户的个数，并不关心顺序，因此整个过程可以看作先选取 $\left\lfloor\frac{c-(c_{total}-l')}{c_{total}}\right\rfloor=\left\lfloor\frac{c+l'}{c_{total}}\right\rfloor-1$ 个 $<c_1,c_2,\cdots,c_n>$，接下来再从位置 $l'=l \bmod k$ 选取 $c-\left\lfloor\frac{c-(c_{total}-l')}{c_{total}}\right\rfloor\times c_{total}=c-\left\lfloor\frac{c+l'}{c_{total}}\right\rfloor\times c_{total}+c_{total}$ 个用户。

显然，若 $\left\lfloor\frac{c+l'}{c_{total}}\right\rfloor=\left\lfloor\frac{c}{c_{total}}\right\rfloor+1$，则问题等价于先选取 $\left\lfloor\frac{c}{c_{total}}\right\rfloor$ 个 $<c_1,c_2,\cdots,c_n>$，接下来再从位置 $l'=l \bmod k$ 选取 $c-\left\lfloor\frac{c}{c_{total}}\right\rfloor\times c_{total}$ 个用户。而当 $\left\lfloor\frac{c+l'}{c_{total}}\right\rfloor=\left\lfloor\frac{c}{c_{total}}\right\rfloor$ 时，则问题等价于先选取 $\left\lfloor\frac{c}{c_{total}}\right\rfloor-1$ 个 $<c_1,c_2,\cdots,c_n>$，接下来再从位置 $l'=l \bmod k$ 选取 $c-\left\lfloor\frac{c}{c_{total}}\right\rfloor\times c_{total}+c_{total}$ 个用户。则最终问题转化为先选取 $\left\lfloor\frac{c}{c_{total}}\right\rfloor$ 个 $<c_1,c_2,\cdots,c_n>$，接下来再从位置 $l'=l \bmod k$ 选取 $c-\left\lfloor\frac{c}{c_{total}}\right\rfloor\times c_{total}=c \bmod c_{total}$ 个用户。由于 $<c_1,c_2,\cdots,c_n>$ 中每个用户的个数是固定的，通过这种方式可以很快地得到最终结果，从而很好地提高整个选取过程的效率。

对于用户的添加与删除的操作，过程与定额分配过程相似，只要处理好相应的数据即可。

（2）对于用户间的资源共享

由于任务所需资源量的不确定性，很难保证机器的计算资源被全部利用，而且这种情况经常存在。用户层的调度不关心任务的选取、资源的匹配等问题，只是合理地为用户部署资源，保证资源部署的公平性，同时尽量提高资源的利用率。

每当有机器心跳到来时，用户层可以得到它的可用资源量 R，并按照上述方法统计每个用户可以获得的资源量 R_i。并将每个用户可以获取的资源量分发下去，并且获得用户返回的可以部署的任务信息，我们认为由用户返回的任务信息是该用户的最佳策略所产生的结果，用户层没有权力对其进行改动，只能按照其当前的顺序进行部署。

由于任务所需资源量的不同，我们不能保证每个用户得到的资源量在本次恰好用尽，同样也不能保证当前机器节点上的计算资源被完全利用。通常的做法是要求每个用户在返回任务信息时，附带额外的任务信息作为备用，以使得当前机器节点上剩余的计算资源可以充分利用。

表2 资源部署示例

30/8=3…6; ABAABA; l=0+6=6								
A	3*5+4=19	-	12,15	30-12-8=10	19-3=16	16-12=4	12	0.461538
B	3*3+2=11	-	8,6	10-6=4 l=6-4=2	11-1=10	10-8-6=-4	8+6=14	0.538462
35/8=4…3; AAB; l=2+3=5								
A	4*5+2=22	22+4=26	15,8,6	35-15-8=12	26-1=25	25-15-8=2	12+15+8=35	0.593220
B	4*3+1=13	13-4=9	10	12-10=2 l=5-2=3	9-1=8	8-10=-2	14+10=24	0.406780
32/8=4…0; l=3								
A	4*5=20	20+2=22	6,11,10	32-6-11=15	22-3=19	19-6-11=2	35+6+11=52	0.611765
B	4*3=12	12-2=10	11	15-11=4 l=3-4=-1=7	10-1=9	9-11=-2	24+11=33	0.388235
31/8=3…7; A|ABAABA; l=7+7=6								
A	3*5+5=20	20+2=22	10,14	31-10-5=16	22-1=21	21-10-14=-3	52+10+14=76	0.666667
B	3*3+2=11	11-2=9	5,7	16-14=2 l=6-2=4	9-1=8	8-5=3	33+5=38	0.333333
20/8=2…4; BABA; l=4+4=0								
A	2*5+2=12	12-3=9	9	20-9-7=4	9-2=7	7-9=-2	76+9=85	0.653846
B	2*3+2=8	8+3=11	7,6	l=0-4=4	11-2=9	9-7=2	38+7=45	0.346154

由于用户计算资源部署的公平性是体现在用户使用的计算资源在单位时间及长时间内的统计均符合其所申请的比例，与集群中的计算资源并没有关系。这样对于每次机器节点上剩余的计算资源可以分配给某些用户，一般我们是将附带的任务进行排序，优先选取符合资源量限制并且所需资源量最大的那些。而此时为了保证用户间的公平性，使用这些剩余资源的用户需要向其它用户提供借据。如果仍有剩余资源R_i，则这些资源将不参与调度，并且将用户序列的统计位置l回滚，并将这个范围内存在的用户减去相应的个数，此时$l=l-R_i$。

每当有机器节点心跳到来时，用户的部署资源量统计完毕后，需要偿还此前借贷其它用户的资源量，并以新的资源量作为依据进行任务部署。这样就可以保证用户资源部署的公平性，及计算资源的合理利用。

假定两个用户A、B，其资源量分别为50 M和30 M，最小资源粒度设为1 M。A与B所占比例分别为0.625和0.375。

A中的任务序列：12、15、8、6、11、10、14、9。

B中的任务序列：8、6、10、11、5、7、6、5。

机器节点资源量：30、35、32、31、20、33。

其中<A,B>=<50/1,30/1>=<50,30>=10<5,3><5,3>=ABAABABA；初始l=0。

资源的部署过程如表2所示。

可以考虑将用户占用资源的时间作为调度的参数，这样更加符合公平原则，在实际应用中一般采用用户占用资源量与其占用该资源的时长的乘积作为衡量标准。

3　实验结果与分析

3.1　资源定额分配方式

定额分配方式以CPU作为资源量，默认1个CPU作为一个单位量。实验采用运行时长从20 s到30 s之间的任务进行衡量，因为任务的时长不定，所以没有采用运行任务个数作为衡量标准，而是以任务的实际运行时长作为标准，即任务实际占用计算资源的时长。四个用户所申请资源量的比例为10、8、6、5。实验环境包括10台机器，每台使用10个CPU，对5000 s时长内的任务运行情况进行统计。

图1为我们所提供的部署策略的运行情况，图2为采用常用的比例方式部署策略的运行情况。前一段时间会有一定的启动时间。

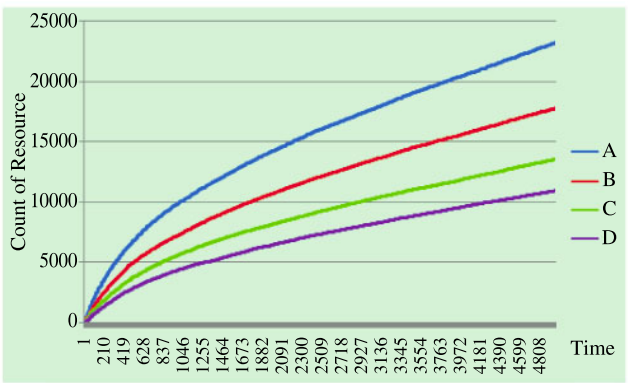

图1　定额分配方式下的序列部署策略资源量统计图

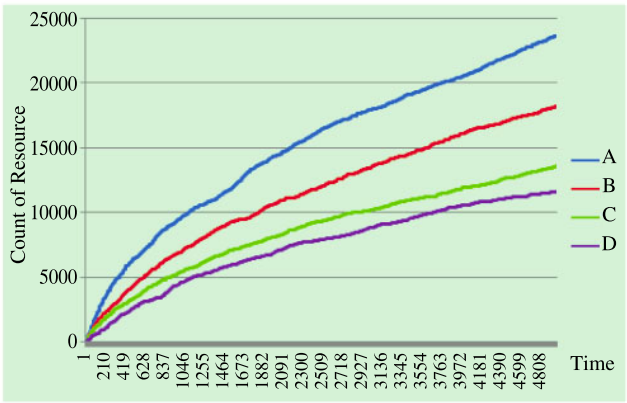

图2　定额分配方式下的比例部署策略资源量统计图

3.2　任务自主申请资源量

自主申请方式以内存作为资源量。实验采用运行所需内存从5 M到40 M之间不等的任务进行衡量，同时每个任务的运行时长与其申请的内存成正比，之所以如此设计，是因为我们需要采用占用资源大小与占用资源时长的乘积作为衡量标准，为了简化实验，最后只对使用的计算资源总量进行统计即可。四个用户所申请资源量的比例为20、14、8、5。实验环境包括10台机器，每台使用100 M内存，对1000 s时长内的任务运行情况进行统计。

图3为我们所提供的部署策略的运行情况，图4为采用常用的比例方式部署策略的运行情况。

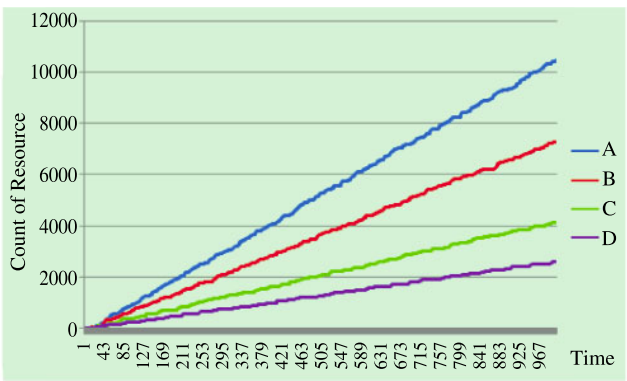

图3　自主申请方式下的序列部署策略资源量统计图

常用的比例方式部署策略在两种方式下的巨大差

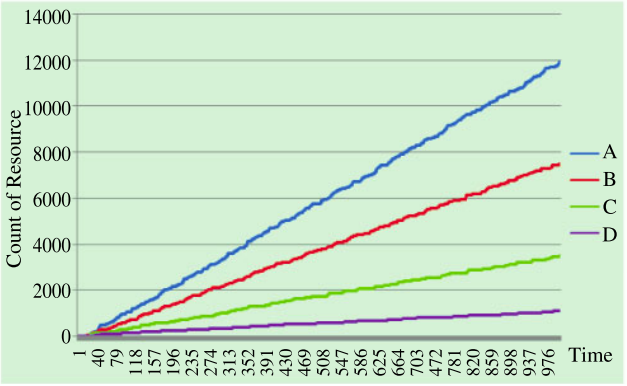

图4 自主申请方式下的比例部署策略资源量统计图

异是因为随着资源粒度的精细，该方式获取的值更能够接近理想值。

4 结束语

本文提出了一种面向多用户集群资源部署策略，可以提高集群资源部署的灵活性，同时保证集群系统具有较高的资源利用率。并且对于集群系统的用户添加删除、集群系统计算资源的扩充与收缩都能够很好地适应。当然这种策略存在一定的缺陷，由于并不关心系统中的实际计算资源量，则会出现资源过剩或是不足的情况，这时需要一个额外的控制器对集群系统的计算资源进行适时的扩充与收容。

参 考 文 献

[1] Apache Mesos [EB/OL]. http://incubator.apache.org/mesos/index.html.

[2] Apache Hadoop [EB/OL]. http://hadoop.apache.org/.

[3] YARN [EB/OL]. http://hadoop.apache.org/docs/r2.0.3-alpha/hadoop-yarn/hadoop-yarn-site/YARN.html.

[4] Shreedhar M, Varghese G. Efficient fair queueing using deficit round robin [J]. IEEE/ACM Transactions on Networking, 1996,4(3): 375-385.