doi:10.3969/j.issn.1000-436x.2014.z2.001

满足本地化计算的集群资源调度策略

郭平[1,2]，宁立江[1]，陈海珠[3]
（1.重庆大学 计算机学院，重庆 400044；2.软件理论与技术重庆市重点实验室，重庆 400044；3.重庆电子工程职业学院，重庆 401331）

摘 要：将本地化计算分成节点本地化计算和机架本地化计算 2 个层次，分别称作节点本地化计算和机架本地化计算。结合主导资源公平调度策略 DRF 和 Delay 调度约束机制提出一种满足本地化计算的集群资源调度策略 DDRF，旨在既能达到一个较高的本地化计算水平，又能保证资源调度的“公平性”。通过数据实验分析了在 DDRF 调度策略下本地化计算延迟对作业执行效率的影响。

关键词：DDRF；资源调度；本地化计算；计算集群
中图分类号：TP301.6 **文献标识码**：A **文章编号**：1000-436X(2014)Z2-0001-08

Scheduling strategy for achieving locality in cluster

GUO Ping[1,2], NING Li-jiang[1], CHEN Hai-zhu[3]
(1.College of Computer Science, Chongqing University, Chongqing 400044, China;
2.Chongqing Key Laboratory of Software Theory & Technology, Chongqing 400044, China;
3.Department of Software Engineering, Chongqing College of Electronic Engineering, Chongqing 401331, China)

Abstract: The data locality is divided into two levels. One is called the node data locality, which placing tasks on nodes that contain their input data. The other one is called the rack data locality, which placing tasks on nodes whose rack contains their input data. A new scheduling strategy called DDRF is proposed which combines the DRF and the delay. The DDRF is not only able to meet high locality but also achieve fairness. In the DDRF, the simulation results show the influence on the efficiency of jobs' implement.

Key words: DDRF; scheduling; locality; computing cluster

1 引言

资源分配是任何一个集群计算机系统必不可少的关键模块。随着集群计算机系统的快速发展，单个集群中的用户量和应用程序种类不断的增加。不同的用户提交的应用程序通常具有不同的服务质量要求，典型的应用可分为 3 种：耗时较长，对完成时间没有严格要求的批处理作业；期望及时返回结果的交互式作业；要求有固定量资源保证的生产性作业。另外，不同的应用程序对资源的需求量是不同的，可分为计算密集型作业、内存密集型作业和 I/O 密集型作业等。因此，简单的资源调度策略不仅不能满足多样化需求，也不能充分地利用硬件资源，设计更加适用集群计算的资源调度策略不可或缺。

目前，常见的集群资源调度策略有以下几种。FIFO[1]即先来先服务。在 FIFO 调度策略下，所有的作业被提交到一个统一的队列中，集群系统按照提交的顺序依次运行作业，使用于批处理作业的场景。FIFO 调度策略实现简单，不能充分利用集群资源，也不能满足不同作业的服务要求。Max-Min Fairness[2]即最大最小公平策略。该策略的基本含义是使得资源分配向量的最小分量最大。Capacity Scheduler[3]即计算能力调度。该调度策略以队列为单位划分资源，支持多队列，每个队列可设置资源使用上限和最低保障，资源空闲时可共享给其他队

收稿日期：2014-10-23
基金项目：国家自然科学青年基金资助项目（61201347）；重庆市自然基金资助项目（2012jjA40022, 2011jjA40027, 2012jjA40011）
Foundation Items: The National Science Foundation for Young Scholars of China (61201347); The Natural Science Foundation Project of Chongqing CSTC(2012jjA40022, 2011jjA40027, 2012jjA40011)

列使用。Fair Scheduler[4]即公平调度等。该调度策略按资源池来组织作业，并把资源公平地分到这些资源池里，每个用户拥有独立的资源池，以保证每个用户获得等同的集群资源。与 Capacity Scheduler 相似，它以队列为单位划分资源并为每个队列设置资源使用上限和最低保障，同时，可为用户设置资源使用上限。该机制允许队列间空闲资源共享，允许每个队列单独配置调度策略（如 FIFO、Max-Min Fairness 等）。DRF（dominant resource fairness）[5]即主导资源公平调度，它将 Max-Min Fairness 计算用于主导资源上，从而将多类型资源调度问题转化为单一资源调度问题。LATE Scheduler[6]适用于异构集群，即集群中各节点性能不一。实时作业调度：Deadline Scheduler[7]适用于软实时作业，即有完成时间限制但允许一定超时的作业。Constraint-based Scheduler[8]适用于硬实时作业，即有完成时间限制且不容超时的作业。Dynamic Priority Scheduler[9]允许用户动态调整自己获取的资源量以保证服务质量要求。

随着数据规模的爆炸式增长，数据密集型计算是集群计算机系统需要解决的问题，集群中大量的数据移动成为集群系统发展的瓶颈。本地化计算概念的提出为解决这个瓶颈提供了一种方案：移动计算而不移动数据。然而，在集群计算机系统中，本地化计算思想和公平资源调度策略之间存在冲突，Delay Scheduling[10]调度机制为解决这个冲突设计的。但是，如何控制任务合理等待及如何设置任务等待时间仍是未能解决的关键问题。另一方面，Delay Scheduling 机制只是针对 MapReduce 计算模型下的 Map 任务。

本文基于 DRF 调度思想，提出满足本地化计算且适用于多类型作业任务、多资源类型的资源调度策略。

2 资源表示

集群将分散的物理机器通过网络聚集，共同为用户提供透明服务，其中包含有多种类型的资源，包括 CPU、内存、缓存、磁盘容量、网络、磁盘 I/O 等。能否让系统友好地理解应用程序调度资源的语义是由资源管理系统的优劣决定的，当前任何一个常见的资源管理系统能够表达的调度语义都是有限的，这种有限性决定了它对某些应用是友好的，但对于其他应用会带来性能瓶颈。

本文所说的资源管理不是以固定的资源组合（如 Hadoop 中的 Slot[11]）作为分配单位的，而是一种按照任务实际需求分配资源的细粒度资源分配方式[12]。讨论的资源包括 CPU 和内存 2 种资源。

1) CPU：在操作系统层面上，CPU 资源是按照时间片分配的，这样的特殊性决定了 CPU 划分的复杂性。本文中 CPU 最小单位 ECU（amazon elastic compute cloud compute unit），是以亚马逊弹性云平台 EC2 的 CPU 资源划分方式为标准[13,14]。因此，每个物理 CPU 可以虚拟成若干个 ECU，用户提交的应用程序可以指定需要的 ECU 个数。

2) 内存：内存的分配以物理内存的 1 MB 为最小单位。

由于集群通过网络聚集，常见的集群通常包括多个机架，每个机架又包含多个节点。根据资源所在位置的不同，资源可分成节点资源（资源在该节点上）和机架资源（资源在该机架内的某个节点上）。为方便讨论起见，本文中所指的资源管理只支持资源分配的调度语义，并且该语义下的资源分配向量可以是资源最小单位的任意组合形式，例如以如下方式分配资源。

分配某个特定节点上的特定资源量组合。例如：分配 node X 上的 4 个<1ECU, 4×1 024 MB 内存>。

分配某个特定机架上的特定资源量组合。例如：分配 rack X 上的 3 个<3ECU, 3 MB 内存>。

分配某个节点上的特定资源量组合。例如：分配 rack X 上的 3 个<5ECU, 0 MB 内存>。

3 DRF 调度策略和 Delay 调度约束

集群包括的资源是多类型的，并且不同的应用程序对资源的需求消耗是不同的，根据需求的不同，可分为计算密集型、内存密集型、I/O 密集型、网络密集型等。例如用户 A 的一个任务资源需求是<1ECU, 4 GB 内存>，用户 B 的一个任务资源需求是<3ECU, 1 GB 内存>。本文将应用程序对集群内不同资源的不同需求称为资源需求的异构性。

3.1 DRF 调度策略及分析

在 DRF 调度中，将资源需求占比最大的资源称为主导资源。针对资源需求的异构性，DRF 基本设计思想是将 Max-Min Fairness 计算用于主导资源上，从而将多类型资源调度问题转化为单一资源调度问题。DRF 算法描述如下。

算法：Dominant Resource Fairness

输入：资源 R，任务及其资源需求 D

输出：调度结果

过程：

R=<r_1,…, r_m>　　//m 种资源对应的总量

C=<c_1, …, c_m>　　//已消耗的资源，初始化为 0

s_i=0, i=1,…,n　　//用户 i 主导资源所占比例，初始化为 0

U_i=<$u_{i,1}$,…, $u_{i,m}$>, i=1,…,n　　//分配给用户 i 的资源，初始化为 0

选择 s_i 最小的用户 i

D_i←<用户 i 下一个任务的资源需求>

IF $C+D_i \leqslant R$ THEN

　$C=C+D_i$　　//更新已消耗资源向量 C

　$U_i=U_i+D_i$　　//更新分配给用户 i 的资源向量

　$s_i = \max\limits_{1\leqslant j\leqslant m}\{u_{i,j}/r_j\}$　　// 更新 s_i

ELSE

　RETRUN　　// 资源用完

END IF

END

例如，一个包含 9ECU 和 18 GB 内存的系统中运行 2 类任务。A 类任务的资源需求是：<1ECU，4 GB 内存>，B 类任务的资源需求是：<3ECU，1 GB 内存>。对于 A 类任务，每个任务消耗总 CPU 的 1/9，总内存的 4/18；对于 B 类任务，每个任务消耗总 CPU 的 3/9，总内存的 1/18。A 类任务内存资源需求占比最大，B 类任务 CPU 资源需求占比最大，因此，在该例中，A 的主导资源为内存，B 的主导资源为 CPU。用 DRF 对该例中 2 类任务进行分配，结果可以通过如下方式进行计算。

假设 x 和 y 分别表示 A 任务和 B 任务的启动数。那么 A 得到<xECU，4xGB 内存>，而用户 B 得到<3yECU, y GB 内存>。A 和 B 得到的总资源量为 (x+3y)ECU 和(4x+y) GB 内存。而 A 和 B 的主导资源所占的比例分别是 4x/18=2x/9，3y/9=y/3。DRF 分配问题可以表示为：

max(x, y)　　//目标是最大化 x 和 y 的值

约束条件为：

x+3y≤9　　//CPU 约束

4x+y≤18　　//内存约束

2x/9=y/3　　//等量化主导资源占比

上述优化问题的解为 x=3 和 y=2。因此，A 和 B 分别得到<3ECU，12 GB 内存>和<6ECU，2 GB 内存>。该结果的 DRF 调度过程如表 1 所示。

DRF 首先启动 B 的一个任务运行。结果 B 的资源比例变成了<3/9，1/18>，主导资源占比是<3/9,1/18>的最大值为 3/9。接着，DRF 选择 A，因为它的主导资源占比还是 0。上述过程持续，直到不能再运行新的任务终止。本例终止于 CPU 资源用尽。而通常情况下，即使有些资源已经用尽，仍然可以继续调度，因为有些任务是不需要已用尽的资源。此外，上述伪代码中未考虑任务运行结束释放资源的情况，当有资源释放时，DRF 重新选择具有主导资源占比的用户。

上述算法可以利用一个二进制堆存储每个用户的主导资源占比来实现。对于 N 类任务的系统，每一个调度决策的时间复杂度为 $O(\log N)$。

3.2　Delay 调度约束及分析

集群中的网络资源是有限的，大量的数据移动是集群系统性能提升的瓶颈之一。本地化计算是解决这个问题的方式，本地化计算指的是在给任务选择节点执行时，优先选择该任务数据所在的节点。虽然本地化计算能够减轻网络通信的压力，但是它和调度公平性之间存在冲突。Delay 调度约束机制是解决该冲突的有效方式之一。

3.2.1　Delay 调度约束

Delay 调度约束机制的基本思想是当不存在满

表 1　　**DRF 调度过程**

调度序列	A		B		CPU 使用量	内存使用量
	资源占比	主导资源占比	资源占比	主导资源占比		
B	<0,0>	0	<3/9,1/18>	1/3	3/9	1/18
A	<1/9,4/18>	2/9	<3/9,1/18>	1/3	4/9	5/18
A	<2/9,8/18>	4/9	<3/9,1/18>	1/3	5/9	9/18
B	<2/9,8/18>	4/9	<6/9,2/18>	2/3	8/9	10/18
A	<3/9,12/18>	2/3	<6/9,2/18>	2/3	1	14/18

足本地化计算的资源时，暂时将资源分配给其他任务，直至出现满足本地化计算的资源或者达到预先设置最大时间延迟。Delay 调度约束机制的关键是在很小一段延迟时间内，期望有满足本地化计算的资源被释放。

3.2.2 Delay 调度约束分析

本节分析在 Delay 调度约束机制下：最大延迟间隔 D 对本地化计算的比例和作业响应时间的影响以及如何设置 D 的大小以达到满意的本地化计算程度。

假设以 Hadoop 集群为例，集群中有 M 个节点，每个节点有 L 个 slot，则总共有 $S=ML$ 个 slot。在任何时间，P_j 表示具有作业 j 要处理的数据节点集合，把这些节点叫做作业 j 的“优先节点”。$p_j=|P_j|/M$ 表示作业 j 的“优先分数”。为了简化分析，假设所有的任务具有相同时间长度 T 且 P_j 内的节点是不相关的。

D 对本地化计算比例提高的影响。作业 j 启动一个满足本地化计算任务的概率是 p_j。则作业 j 等待 D 之后，启动一个不满足本地化计算任务的概率是 $(1-p_j)^D$，这个概率随 D 值增大指数性减小。例如，一个作业的数据存储在 10%的节点上（即 $p_j=0.1$）。则当 $D=10$ 时，该作业启动一个满足本地化计算任务的可能性约为 65%，当 $D=40$ 时，该作业启动一个满足本地化计算任务的可能性近 99%。

为满足本地化计算，作业需要多等待时间。因为集群中共有 S 个 slot，任务长度为 T，则平均每 T/S 时间后有一个 slot 空闲。因此，一旦作业 j 到达队列头部，它最多等待 DT/S 时间之后启动一个不满足本地化计算的任务。在 S 足够大情况下，这个等待时间远小于任务平均时间长度，而由于本地化计算任务不用考虑数据移动的时间开销，一个任务本地化计算的完成时间远远小于非本地化计算完成时间。此外，对于一个固定数量节点的集群，等待时间随着每个节点上 slot 数量的增加线性递减。

如何设置 D 的大小以达到一个期望的本地化计算程度。假设一个具有 M 个节点的集群，每个节点有 L 个 slot，数据副本数为 R，作业具有 N 个任务，期望达到本地化计算比例不少于 λ。计算作业 j 的本地化计算程度可通过如下方式：对当作业 j 还有 N，$N-1$，…，1 个任务时启动一个本地化计算任务的概率值进行平均。当作业 j 还有 K 个任务没有启动时，给定一个节点，其上没有作业 j 的数据副本的概率是 $(1-K/M)^R$，作业 j 的“优先分数”为 $p_j=1-(1-K/M)^R$，因此，在这种情况下，作业 j 启动一个满足本地化计算任务的概率为 A_k，则

$$A_k=1-(1-p_j)^D=1-(1-K/M)^{RD}\geqslant 1-\mathrm{e}^{-RDK/M}$$

作业 j 的本地化计算程度，用 $L(D)$ 表示

$$L(D)=\frac{1}{N}\sum_{K=1}^{N}A_k=\frac{1}{N}\sum_{K=1}^{N}1-\mathrm{e}^{-RDK/M}=1-\frac{1}{N}\sum_{K=1}^{N}\mathrm{e}^{-RDK/M}$$

$$\geqslant 1-\frac{1}{N}\sum_{K=1}^{\infty}\mathrm{e}^{-RDK/M}\geqslant 1-\frac{\mathrm{e}^{-RD/M}}{N(1-\mathrm{e}^{-RD/M})}$$

若期望作业的本地化程度不小于 λ，即 $L(D)\geqslant\lambda$ 得到

$$D\geqslant -\frac{M}{R}\ln\left(\frac{(1-\lambda)N}{1+(1-\lambda)N}\right)$$

例如，期望作业的本地化计算比例 $\lambda=95\%$，作业的任务数 $N=20$，数据副本数 $R=3$，则需要 $D\geqslant 0.23M$。在这样的延迟 D 情况下，为满足本地化计算的最大等待时间是：$DT/S=DT/LM\leqslant 0.23T/L$。如果每个节点的 slot 数 $L=8$，则这个等待时间不大于任务平均长度 T 的 2.8%。

综上，有如下结论：作业不满足本地化计算的程度随 D 的增大指数下降；为满足本地化计算的等待时间远小于任务平均时间长度且随着每个节点上 slot 数的增长线性减小。

4 DDRF 调度策略

DRF 调度策略保证主导资源公平，适用于具有异构性需求的资源调度，但是 DRF 不考虑本地化计算。而本地化计算是一种解决集群通信压力的有效机制。本文结合 DRF 调度策略和 Delay 调度约束机制提出一种满足本地化计算的调度策略（DDRF，delay-dominant resource fairness）。旨在“公平性”和本地化计算二者之间寻求一个合理的折衷，既能保证资源调度的“公平性”，又能达到一个较高本地化计算水平。

本文讨论的本地化计算分成 2 个层次：一是节点本地化计算，即任务和数据在同一个节点上；另一是机架本地化计算，即任务和数据不在同一节点上，但是在同一机架上。由于机架本地化计算仍需要局域内的网络通信，本文认为机架本地化计算的本地化程度比节点本地化计算的本地化程度低。因此，设置 2 种最大延迟 E_1 和 E_2，并且 $E_1<E_2$，当延迟达到 E_1 时，则不再等待节点本地化情况，选择机架本地化计算；当延迟达到 E_2 时，则认为任务等待

时间过长，不再等待，直接选择非本地化计算。

4.1 DDRF 调度算法

DDRF 调度策略的描述如下。

算法：DDRF

输入：资源 R、任务及其资源需求 D，任务数据所在的节点位置，2 个延迟 E_1、E_2

输出：调度结果

过程：

$R=<r_1,\cdots, r_m>$　　//集群中 m 种资源对应的总量

$C=<c_1,\cdots, c_m>$　　//已消耗的资源，初始化为 0

$s_i=0$,　$i=1,\cdots,n$　//作业 i 主导资源所占比例，初始化为 0

$U_i=<u_{i,1},\cdots, u_{i,m}>$,　$i=1,\cdots,n$　// 分配给作业 i 的资源，初始化为 0

i.skip←0;　　//将所有作业的延迟间隔初始化为 0

选择 s_i 最小的作业 i;

D_i←<用户 i 下一个任务 t 的资源需求>

IF $C+D_i \leqslant R$ THEN

IF 任务 t 所在的节点有空闲资源 Then

在该节点上启动任务；

C=$C+D_i$　　//更新已消耗资源向量 C

$U_i=U_i+D_i$　//更新分配给用户 i 的资源向量

$s_i = \max\limits_{1\leqslant j\leqslant m}\{u_{i,j}/r_j\}$　　// 更新 s_i

i.skip←0;　　//作业 i 的延迟间隔置为 0

ELSE　　//任务 t 所在的节点无空闲资源

i.skip← i.skip+1;　//作业 i 的延迟间隔+1

IF $E_2 \geqslant i$.skip$\geqslant E_1$ THEN　//作业 i 的延迟间隔达到机架最大值 E_1

IF 任务 t 的数据所在机架的节点有空闲资源 THEN

在该节点上启动任务

C=$C+D_i$　　//更新已消耗资源向量 C

$U_i=U_i+D_i$ //更新分配给用户 i 的资源向量

$s_i = \max\limits_{1\leqslant j\leqslant m}\{u_{i,j}/r_j\}$　　// 更新 s_i

i.skip←0;　// 重置为 0，后续任务重新计算延时

ELSE　//任务 t 的数据所在机架的节点无空闲资源

i.skip← i.skip+1;　//作业 i 的延迟间隔+1

IF i.skip$\geqslant E_2$ THEN　//作业 i 的延迟间隔达到集群最大值 E_2

在任意节点上启动任务 t(此时不满足本地化计算)

C=$C+D_i$　//更新已消耗资源向量 C

$U_i=U_i+D_i$　//更新给用户 i 的资源向量

$s_i = \max\limits_{1\leqslant j\leqslant m}\{u_{i,j}/r_j\}$　　// 更新 s_i

i.skip←0;　// 重置为 0，后续任务重新计算延时

END IF

END IF

END IF

END IF

ELSE

RETURN　　//资源用完

END IF

END

上述算法中，每当任务启动后（无论是否满足本地化计算）都将作业的延迟置为 0，后续任务启动时重新计算延迟。

4.2 DDRF 调度分析

由于机架本地化计算仍需要局域内的网络通信，因此它的本地化是打了折扣的。不防认为节点本地化计算为 1，机架本地化计算为θ (0<θ<1)，非本地化计算为 0，将θ定义为“贡献参数”。

为简化分析，做如下假设：集群有 2 种资源分别是 CPU 和内存，M 个同构的（每个节点拥有相同的资源类型和资源量）节点，每个机架具有 G 个节点，每个节点具有 L 个 CPU，W 个内存。则集群中共有 CPU：$S_{\text{CPU}}=ML$，内存：$S_{\text{mem}}=MW$。

在任何时间，让 P_i 表示具有作业 i 要处理的数据的节点集合且 P_i 是不相关的，把这些节点叫做作业 i 的“优先节点”，$p_i=|P_i|/M$ 表示作业 i 的“优先分数”，Q_i 表示与 P_i 节点在同一机架上的节点集合但不包括 P_i 中的节点，把这些节点叫做作业 i 的“次优先节点”，$q_i=|Q_i|/M$ 表示作业 i 的“次优先分数”所有的任务都具有相同的时间长度 T。

本节剩余部分从 3 个方面分析 DDRF 调度策略：最大延迟 E_1、E_2 对本地化计算程度如何影响；如何设置 E_1 和 E_2 的大小来达到一个期望的本地化计算程度；为满足期望的本地化计算程度，作业需要多等待多长时间。

计算最大延迟 E_1 和 E_2 对本地化计算程度的影响。不防假设机架本地化计算对本地化计算没有贡献。当作业 i 启动任务时，它启动一个满足节点本地化计算任务的概率是 p_i；则作业 i 等待了 E_1 之后，启动一个不满足本地化计算任务的概率是 $(1-p_i)^{E_1}$，则这个概率随 E_1 增大指数减少。假设一个作业的数据存储在 5%的节点上，即 p_i=0.05。表 2 反映了 E_1 和作业启动一个满足本地化计算任务的概率 p 之间的关系。

表 2　　E_1 值设置与本地化计算概率关系

E_1	p
10	40%
20	65%
30	79%
40	87%
50	93%
60	98%

而实际上，机架本地化计算对本地化计算具有贡献，因此，表 2 中概率值 p 随 E_1 的增大而更快地增大。

如何设置 E_1 和 E_2 的大小来达到一个期望的本地化计算比例。设置集群节点的副本数为 R（副本分别布局在不同的机架），每个作业的任务数为 N，期望的本地化计算比例为λ。计算作业 i 的本地化计算比例通过如下方式：对当作业 i 还有 N，N–1，…，1 个未启动的任务时，启动一个满足本地化计算任务的概率值进行平均。当作业 i 还有 K 个任务未启动时，给定一个节点，其上没有节点副本的概率为 $(1-K/M)^R$，此时作业 i 的“优先分数” p_i 为：$1-(1-K/M)^R$，作业启动一个满足节点本地化计算的概率为 B_k，则

$$B_k=1-(1-p_i)^{E_1}=1-\left(1-\frac{K}{M}\right)^{RE_1}\geqslant 1-\mathrm{e}^{-RE_1K/M}$$

作业 i 的节点本地化计算程度，用 $L(E_1)$表示。则

$$L(E_1)=\frac{1}{N}\sum_{K=1}^{N}B_k\geqslant\frac{1}{N}\sum_{K=1}^{N}1-\mathrm{e}^{-RE_1K/M}\geqslant 1-\frac{1}{N}\sum_{K=1}^{N}\mathrm{e}^{-RE_1K/M}$$

此外，给定一个机架，其上没有节点副本的概率为 $(1-K/(M-G))^R$，此时作业 i 的“次优先分数” q_i 为 $1-(1-K/(M-G))^R$，作业 i 启动一个满足机架本地化的概率为 C_k，则

$$C_k=1-((1-p_i)^{E_1}(1-q_i)^{E_2-E_1}$$

$$=1-\left(1-\frac{K}{M}\right)^{RE_1(E_2-E_1)}\left(1-\frac{K}{M-G}\right)^{R(E_2-E_1)}$$

$$\geqslant 1-\mathrm{e}^{\frac{-RE_1(E_2-E_1)K}{M}}\mathrm{e}^{\frac{-R(E_2-E_1)K}{M-G}}$$

$$>1-\mathrm{e}^{\frac{-R(E_2-E_1)K(E_1+1)}{M}}$$

根据 θ，将机架本地化折合成节点本地化，用 C'_k 表示，即

$$C'_k=1-\mathrm{e}^{-R(E_2-E_1)K(E_1+1)\theta/M}$$

作业 i 的机架本地化计算程度，用 $L(E_2)$表示

$$L(E_2)=\frac{1}{N}\sum_{K=1}^{N}C'_k=\frac{1}{N}\sum_{K=1}^{N}1-\mathrm{e}^{-R(E_2-E_1)K(E_1+1)\theta/M}$$

$$=1-\frac{1}{N}\sum_{K=1}^{N}\mathrm{e}^{-R(E_2-E_1)K(E_1+1)\theta/M}$$

用 $L(E)$表示作业的本地化程度，则

$$L(E)=L(E_1)+L(E_2)$$

若期望作业的本地化程度不小于λ，即 $L(E)\geqslant\lambda$ 得到 E_1、E_2 和λ的关系。

对上述分析，通过程序仿真得到如下实验数据，设置每个作业的任务数 N=20，数据副本数 R=3，贡献参数θ=0.05。计算节点数 M 为 100、200、300、400、500、600、700、800、900、1 000 时，E_1、E_2 对本地化程度的影响关系。

图 1 表明，对于固定的 E_1，本地化程度随着 E_2 增大而快速增大；且 E_1 越大，增速越大。

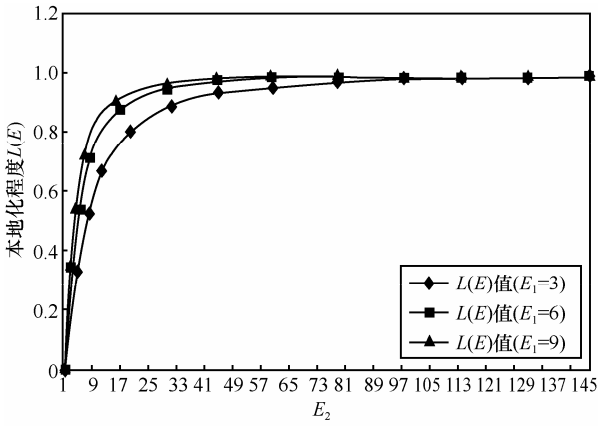


图 1　本地化程度 $L(E)$变化(M=1 000)

图 2 表明，不考虑机架本地化的贡献，本地化程度随着 $E1$ 的增大而增大。图 1 对比图 2 表明，机架本地化计算带来的较大的贡献，同时对 $E1$、$E2$ 值的设置有较大影响，影响程度与“贡献参数”θ有关。

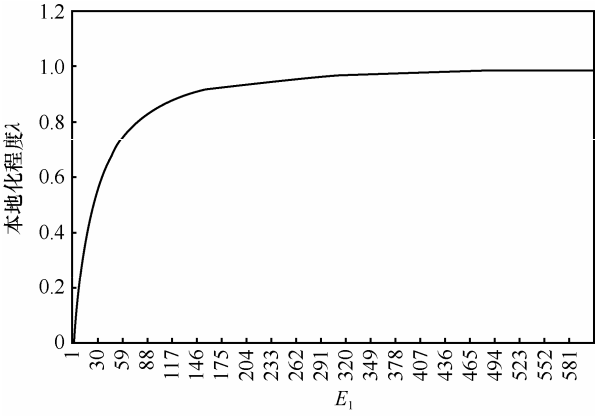


图 2　本地化程度λ随 E_1 的变化情况

图 3 表明，在固定的本地化程度（99%）要求下，E_2 随着 E_1 的增大迅速减小，并且，一旦 E_1 设置的足够大，只需在 E_1 的基础上增加一个很小值就足以达到期望的本地化程度。

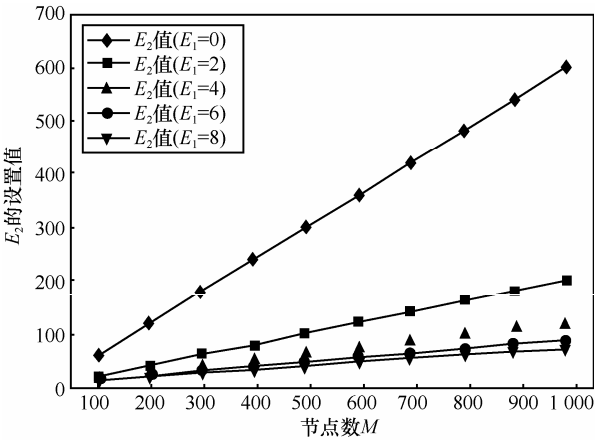


图 3　E_2-E_1 随 E_1 变化情况

为满足本地化计算作业需要多等待多长时间。因为集群中共有 S_{CPU} 个 CPU 和 S_{mem} 个内存，任务长度为 T，则平均每 T/S_{CPU} 时间释放一个空闲 CPU，平均每 T/S_{mem} 时间释放一个空闲内存，可以认为每 $X=(T/S_{\mathrm{CPU}}+T/S_{\mathrm{mem}})/2$ 时间释放一个主导资源。所以，一旦作业 i 到达队列头部，它最多等待

$$E_2X = E_2\,(T/S_{\mathrm{CPU}} + T/S_{\mathrm{mem}})/2 = TE_2(L+W)/2WLM$$

时间之后启动一个完全不满足本地化计算的任务。如果 CPU 总量 S_{CPU} 和内存总量 S_{mem} 足够大情况下，这个等待时间远小于任务平均时间长度 T。假设设置 L=8、W=8、E_1=6、E_2=0.09，这个等待时间只是任务平均长度 T 的 1.125%，而此时本地化程度达到 99%。另外，对于一个固定数量节点的集群，这个等待时间随着每个节点上资源数量增多线性递减。

图 4 是在达到 99%的本地化程度情况下，E_1、E_2 设置随节点数 M 的变化关系。

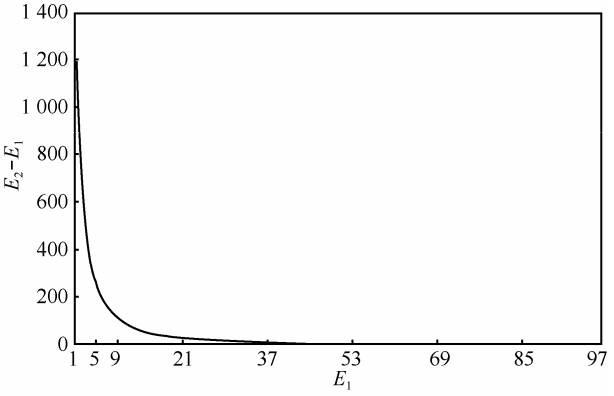


图 4　E_2 随节点数 M 的变化关系（λ=99%）

图 4 表明，为达到期望的本地化程度（99%），E_2 的设置随着节点数 M 的增长，线性增长。且随着 E_1 的增大，E_2 的增速减小，其中 E_1=0 表示不考虑机架本地化的贡献。

5　结束语

本文结合 DRF 调度策略和 Delay 调度约束机制提出一种满足本地化计算的调度策略 DDRF。DDRF 将本地化分成 2 个层次：节点本地化和机架本地化。通过理论和数据实验分析有如下结论：对于固定的延迟 E_1，本地化程度随着延迟 E_2 增大而快速增长，且 E_1 越大，增速越大；E_2 随着 E_1 的增大迅速减小，即一旦 E_1 设置的足够大，只需在 E_1 的基础上增加一个很小值就足以达到期望的本地化程度；为达到期望的本地化程度，E_2 大小的设置与节点数 M 呈线性增长关系；为满足本地化计算的等待时间远小于平均任务长度且随着每个节点上的资源数量的增加线性减小。

对于 DDRF，本文没有分析“贡献参数”θ对本地化程度的影响，这将是进一步的研究工作。

参考文献：

[1] FIFO[EB/OL].http://en.wikipedia.org/wiki/FIFO.

[2] LEBOUDEC J Y. Rate adaptation, congestion control and fairness: a tutorial[EB/OL]. http://moodle.epfl.ch/file.php/523/CC_Tutorial/cc.pdf.

[3] Capacity scheduler[EB/OL]. http://hadoop.apache.org/docs/r0.19.1/capacity_scheduler.html.

[4] ZAHARIA M, BORTHAKUR D, SARMA J S, *et al.* Job Scheduling for Multi-user Mapreduce Clusters: UCB/EECS-2009-55[R]. Berkeley: EECS Department, University of California, 2009.

[5] GHODSI A, ZAHARIA M, HINDMAN B, *et al*. Dominant resource fairness: fair allocation of multiple resource types[A]. Proceedings of the 8th USENIX Conference on NSDI[C]. Berbeley: USENIX Association, 2011.24-37.

[6] ZAHARIA M, KONWINSKI A, JOSEPH A D, *et al*. Improving mapreduce performance in heterogeneous environments[A]. 8th USENIX Symposium on Operating Systems Design and Implementation[C]. San Diego: USENIX Association, 2008.29-42.

[7] POLO J, CARRERA D, BECERRA Y, *et al*. Performance-driven task co-scheduling for mapreduce environments[A]. Network Operations and Management Symposium (NOMS)[C]. San Diego: IEEE, 2010. 373-380.

[8] KC K, ANYANWU K. Scheduling hadoop jobs to meet deadlines[A]. Cloud Computing Technology and Science (CloudCom)[C]. San Diego: IEEE, 2010: 388-392.

[9] Dynamic priority scheduling[EB/OL]. http://en.wikipedia.org/wiki/Category: Scheduling_algorithms

[10] ZAHARIA M, BORTHAKUR D, SEN SARMA J, *et al*. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling[A]. Proceedings of the 5th European conference on Computer systems. Paris: ACM, 2010: 265-278.

[11] SHVACHKO K, KUANG H, RADIA S, *et al*. The hadoop distributed file system[A]. Mass Storage Systems and Technologies (MSST)[C]. Lake Tahoe, Nevada, USA: IEEE, 2010: 1-10.

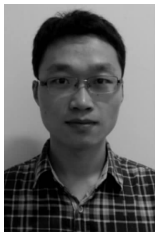
[12] GUO P, NING L, SU L, *et al*. A new strategy of resource management for cloud computing[J]. Information Technology Journal, 2013, 12(18).

[13] Amazon elastic compute cloud[EB/OL]. http://aws.amazon.com/cn/ec2/

[14] OSTERMANN S, IOSUP A, YIGITBASI N, *et al*. A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing Cloud Computing[M]. Springer Berlin Heidelberg, 2010.115-131.

作者简介：

郭平（1963-），男，四川眉山人，博士，重庆大学教授，主要研究方向为膜计算、云计算、智能信息系统。



宁立江（1988-），男，山东莘县人，重庆大学硕士生，主要研究方向为云计算、智能信息系统。

陈海珠（1980-），女，广西柳州人，博士，重庆电子工程职业学院副教授，主要研究方向为云计算、膜计算。

ISSN 1000-436X

发行代号：国内2-676 / 国外M395

（2014）京新出报刊增准字第（501）号

2014年11月30日出版 定价：78.00元