异构环境下 MapReduce 动态任务调度技术研究

范 宇[1,2]，郭会明[3]

(1. 北京航天长峰科技工业集团有限公司，北京 100039；2. 中国航天科工二院，北京 100039；3. 北京航天长峰股份有限公司，北京 100039)

摘 要：针对 MapReduce 在异构环境下各节点性能不均衡，导致整体计算效率低下的问题进行了研究，从节点与任务两方面入手，提出了一种将节点性能量化并排序与将任务按相似度划分相结合的算法。该方法根据历史日志以及实时回传的日志信息将节点按照性能高低排序；根据任务执行完成的信息，将其与新任务进行比对得到相似度，从而推测出新任务的执行时间，执行时间长的认为是复杂的任务；最后进行动态调度，使高性能节点处理更复杂的任务。在随机生成的数据集上的实验结果表明，所提出的动态调度算法与默认调度算法相比，数据集为 20 GB 大小时执行速度提高 27.4%，数据集为 100 GB 大小时执行速度提高了 74.1%。

关键词：异构；MapReduce；任务调度；节点排序；任务划分

中图分类号：TP399 **文献标志码**：A **文章编号**：1001-3695(2018)05-1408-04

doi:10.3969/j.issn.1001-3695.2018.05.027

Research on MapReduce dynamic task scheduling technology in heterogeneous environment

Fan Yu[1,2], Guo Huiming[3]

(1. *Changfeng Science Technology Industry Group Corp*, *Beijing* 100039, *China*; 2. *The* 2*nd Institute of China Aerospace Science & Industry Corp*, *Beijing* 100039, *China*; 3. *Beijing Aerospace Changfeng Co. Ltd.*, *Beijing* 100039, *China*)

Abstract: The performance of each node in heterogeneous environment is not balanced, so computational efficiency of MapReduce is inefficiency. To solve this problem, from both aspects of node and task, this paper proposed a new algorithm which combined quantificating node's performance and dividing task. Firstly it sorted the nodes by their performance according to both the history logs and the information fed back real-time. Then it compared with new task according to task execution information. Finally it dynamically scheduled those tasks, made sure high performance nodes dealing with complex tasks. The experimental results show that compared with the default scheduling algorithm, the dynamic scheduling algorithm can improve 27.4% of execution speed for 20 GB dataset and 74.1% for 100 GB dataset.

Key words: heterogeneous; MapReduce; task scheduling; sorting nodes; dividing tasks

0 引言

近年来，随着互联网技术的迅猛发展以及办公自动化的普及，尤其是随着 Facebook、微博、微信等为代表的自媒体信息发布方式的不断涌现，社会运行中很大一部分的活动都与数据的创造、采集、处理、分析等息息相关，大数据的应用范围不断扩大，影响力也日渐增长。有专家指出，近两年所产生的数据量相当于人类有史以来所产生的所有数据量的总和[1,2]。全世界最知名的咨询公司 McKinsey 曾在 2011 年发布了一份关于大数据未来发展的详细报告，并称大数据是下一代创新、竞争和生产力的前沿阵地[3]。

在所有大数据技术中，以 MapReduce 为代表的并行计算框架因其出色的表现一直受到人们的关注。由于可以将作业划分为多个任务并行执行，MapReduce 对基础硬件要求不高，可以将多台配置较低的计算机并联到一起组成计算集群，对其计算能力进行统一调配，达到对大数据处理较快的计算速度。但是在现实使用中，由于部分机器老化等原因需要更换这部分机器，会使得原有集群中计算机的性能千差万别，从而产生异构性。在异构环境下，集群中不同计算节点的计算能力则大不一样，所以按照原有的平均分配任务方式将会造成部分性能较高的节点处于饥饿状态，而性能较差的节点始终处于饱和状态[4]；又由于 MapReduce 具有显著的木桶效应，只有当最后一个任务完成整个任务才算完成，所以被分配了同样多任务的性能较差的节点会严重拖慢执行时间，从而影响整体性能。

1 相关工作

针对 MapReduce 在异构环境下的任务调度问题，最早的研究当属 Zaharia 等人[5]在加州大学伯克利分校 AMP 实验室中提出的 Straggler 问题，即 MapReduce 中最慢的任务被称之为 Straggler，它的执行时间直接影响着整个任务的时间。应对该问题 Zaharia 等人提出了 LATE 算法，其思想是通过系统观察找出是否存在某一任务的执行进度慢于某一设定的阈值，如果

收稿日期：2016-12-26；**修回日期**：2017-02-16

作者简介：范宇(1991-)，男，黑龙江齐齐哈尔人，硕士，主要研究方向为大数据与深度学习(alex3936@163.com)；郭会明(1966-)，男，湖北天门人，研究员，硕导，主要研究方向为计算机信息系统集成与应用.

存在,则选择某一节点执行该任务的备份任务,以确保该任务不会由于节点问题而拖慢整个任务。但该方法存在着明显的不足之处:a)对于 Straggler 任务的判断;b)该方法需要预留一部分节点做备份,这样不能够达到最大化利用现有计算资源,与大数据的思想相悖;c)对于备份节点的选择,如果执行备份的节点相较原节点性能没有明显的提高,那么执行备份的过程就纯粹是浪费资源,而如果备份节点全是高性能节点,又明显会拖慢原有任务执行速度。

除 Zaharia 之外,还有很多学者在这方面做了很多的工作,如 Snavely 等人[6]提出了在 HPC 中使用性能模型预测任务执行时间的方法;Yong 等人[7]提出让每个任务节点收集自身资源消耗的信息并汇总统计后分配之后的任务;Xie 等人[8]设计了一种新型数据分布策略使 MapReduce 获得了更高的数据局部性;Sun 等人[9]提出一种更全面、更精确的估计任务剩余时间的方法;Rasooli 等人[10]提出了将 FIFO、公平选择、分类及优化等多种策略相结合的方法。上述方法都是在原有调度策略基础上对性能进行了优化和提升。

本文在对上述几种算法分析研究的基础之上,从节点与任务两方面入手,提出一种基于性能监测的节点动态排序算法与基于相似度的任务划分算法相结合的办法,用于提高 MapReduce 的执行效率。核心思想是依照性能检测的排序算法将节点按照性能高低排序并不断更新排序顺序,同时依照任务相似度划分算法将相似度高的任务归并。在执行过程中遵循两个原则:将复杂任务交给高性能节点处理;将归并相似的任务交给相同节点处理。在最后将本方法与默认的任务调度算法相比较,得出本方法在作业完成时间、作业对于系统性能的利用率,包括 CPU 和内存的利用率上均有显著提升。

2　算法设计

2.1　基于性能检测的节点动态排序算法

首先在任务执行过程前,依据最近一段时间各节点执行任务的日志,通过本算法将节点按照性能高低排序;然后在任务执行过程中通过实时回传的执行日志不断将该排序结果进行更新,做到动态排序。

在异构环境下,某一节点的性能决定因素有很多,如主机的 CUP 性能、I/O 速率、内存大小等,针对不同的硬件设备,其决定因素的权重也不尽相同。假设系统中共有 v 个节点,可以用 $X^1, X^2, \cdots, X^v$ 表示这些节点,p 为影响节点性能因素的个数,则可以使用 X_p^v 表示第 v 个节点的第 p 个影响因子。具体算法过程如下:

a)获取最近一段时间内节点 i 在执行计算任务时的历史记录日志。若第一条用 X^{i1} 表示,则 n 条记录可以用如下所示的矩阵列表示[11]。

$$\begin{pmatrix} X_1^{i1} & X_2^{i1} & \cdots & X_m^{i1} \\ X_1^{i2} & X_2^{i2} & \cdots & X_m^{i2} \\ \vdots & \vdots & & \vdots \\ X_1^{in} & X_2^{in} & \cdots & X_m^{in} \end{pmatrix}$$

将得到记录数据整理完毕后,计算节点 i 中 n 条记录的影响因子的均值 U。

$$U_m^i = \frac{1}{n}\sum_{j=1}^{n} X_m^{ij} \tag{1}$$

b)分别计算节点 i 中 n 条记录的影响因子的方差 σ^2。

$$(\sigma_m^i)^2 = \frac{1}{n}\sum_{j=1}^{n}(X_m^{ij} - U_m^i) \tag{2}$$

c)利用上述计算得到的结果,推测计算出当前节点 i 的性能分数。

$$P_i(n) = \prod_{i=1}^{m} p[x_j^{in}; u_j^i, (\sigma_j^i)^2] = \prod_{j=1}^{m} \frac{1}{\sqrt{2\pi\sigma_j^i}} \exp\left(-\frac{(x_j^{in} - u_j^i)^2}{2(\sigma_j^i)^2}\right) \tag{3}$$

对于全部节点进行如上计算,将节点按照上述计算结果得到的性能得分从高到低进行排列,得到一组节点数组 $L = [X^a, X^b, \cdots, X^n]$。由于使用的是历史记录,所以上述计算工作可以在节点非工作状态下进行,即它们并不占用任务执行时间。接下来在任务计算过程中,根据任务大小不同,采用不同时间间隔对节点日志进行采集,并重新计算节点性能,采用实时数据对所得的节点数组 L 进行更新。这样可以得到一个最接近的能够表明节点性能高低的数组,然后根据本文后续算法内容,将复杂、推测会长时间执行的任务交给高性能节点,并将任务依次分配;将数据结构单一、数据量小等被推测出耗时较短的任务交给性能较低的节点执行。

2.2　基于相似度的任务划分算法

本算法的核心指导思想是:根据相似度影响因子等一系列计算手段,将待执行的任务按照相似的程度划分成若干份。尽量使用相同的节点处理相似任务,然后根据先执行的任务执行时间可以推测出相似任务所需要的大致执行时间,据此决定是否需要调整执行这一类相似任务的执行节点。

本算法的实现依据是相似的任务在同一环境下执行时间同样具有相似性。为此进行了如下实验:采用最基础的 WordCount 算法,在同一台机器上,采用伪分布式的方式,分别执行数据量大小相等(均为 3 GB 左右),但分别由阿拉伯数字、英文小写字母和简体中文组成的若干组任务,比较各组任务执行时间长短。其中 a ~ c 组的内容都是由阿拉伯数字组成、d ~ f 组的内容由英文小写字母组成、g ~ i 组的内容由简体中文组成。所有数据均没有明显规律且随机生成。九组数据完全执行所需时间如表 1 所示。

表 1　九组不同数据执行 WordCount 算法时间

阿拉伯数字			英文小写字母			简体中文		
a	b	c	d	e	f	g	h	i
38.6	40.7	36.4	42.1	39.8	45.3	52.0	57.7	54.6

显然相同种类文字组成的任务执行时间更为接近。由此可以推测出结构相似的任务在同一环境下执行时间同样相似的结论是正确的。

接下来要考虑到的关键问题是如何量化不同任务的影响因子。同样,对于异构环境下的任务,应该根据硬件环境的不同,从而采取不同的影响因子权重策略。如对于一个内存丰富但 CPU 运算速度较慢的节点,CPU 影响整个速率的权重显然应当比内存更大;而对于拥有强悍 CPU 但内存不足的节点,内存的重要性便尤为显著。为此本文采用已经成熟运用于权重设置方面的二次规划算法[12]。二次规划是非线性规划的一种特殊类型,它的一般形式为[13]

$$QP = \begin{cases} \min f(x) = \frac{1}{2}x^{\mathrm{T}}Qx + c^{\mathrm{T}}x \\ \text{s.t.} \quad x \in D \end{cases} \tag{4}$$

其中:Q 是 Hessian 矩阵;T 是有限指标集;D 是 R^n 中的一个多

面体，$c \in R^n$。

本文采用的具体量化的两个不同任务 X、Y 的相似度公式如下：

$$d(X,Y) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2} \tag{5}$$

其中：x_i 和 y_i 是它们的影响因子；w_i 为影响因子的权重；n 为影响因子的个数。

在这里假设当前的一个工作节点中，一共有 n 条任务执行的日志文件，其中每一个任务都有 p 个影响因子[14]，则所得到的计算方法为

$$R_{xy} = 1 - \frac{|t - t_{avg}|}{\max\{t, t_{avg}\}} \tag{6}$$

$$L_{xy} = R_{xy} - \sum_{k=1}^{p} S_{xyk} w_k \tag{7}$$

其中：R_{xy} 是任务执行后通过公式计算出的真实相似度；t 表示真实的执行时间；t_{avg} 表示预测执行时间的平均值；S_{xyk} 代表的是任务 x 和 y 在第 k 个属性上的相似度；$\sum_{k=1}^{p} S_{xyk} w_k$ 是两个任务通过式(5)计算得到的相似度结果；L_{xy} 表示真实相似度和计算相似度之间的差别。通过以上计算，可以得到通过二次规划得到的异构环境下，任意两个任务 X 和 Y 的相似度计算结果：

$$W = \sum_{i=1}^{n} \sum_{j=T+1}^{n} L_{ij}^2 \tag{8}$$

由此便可以通过设置不同的阈值，将全部任务划分为若干不同的小组任务。这样可以认为相同小组中的任务为相似任务，它们在同一环境同一节点上的执行时间也是相似的。因此每一小组中在某一节点上执行一个任务的时间可以作为该小组中后续任务执行时间的参考。

2.3 基于性能排序和任务划分的动态任务调度算法

经过以上两种算法，分别可以得到一组表示各节点性能由高到低降序排列的数组 $L = [X^a, X^b, \cdots, X^n]$，以及根据相似度划分所得到的若干相似任务小组。

动态调度算法基于如下思想：根据每一个执行完毕的任务在某节点上执行所需时间，可以推测认为该组任务在该节点上执行时间基本相同，再根据具体执行时间的长短，判断是否需要将该组任务动态调度到其他节点执行，以及应该调度到哪个节点去执行。

算法 1 动态任务调度算法。

输入：

$L = [X^a, X^b, \cdots, X^n]$：节点性能排序数组；

TaskGroup $T_1, T_2, \cdots$：相似任务小组；

其中 $T_n = [t_{na}, t_{nb}, \cdots]$，$t_{nm}$ 为第 n 组第 m 个任务。

```
for each t_np proceeding in X^i
    compute time_n = proceeding time
    meanTime = (time_1 + time_2 + ⋯ + time_n + ⋯ + time_max)/max
for each time_n do
    if time_n > meanTime
        t_nq proceeding in X^j (q > p, j < i)
    if time_n < meanTime
        t_nq proceeding in X^j (q > p, j > i)
end
```

3 实验结果及分析

本章采用实验验证的方式，在一个由 21 台规格不尽相同的台式机组成的异构环境集群中，通过 WordCount 程序多次执行任务量不同的数据集，验证本文方法与默认的调度策略执行效果优劣。实验采用 Ganglia[15] 来检测集群状态，对实验结果进行分析比对。

具体节点的配置信息如表 2 所示。

表 2 硬件配置信息

节点	数量	硬件信息
master	1	四核，2.8 GHz，8 GB 内存，1 TB 机械硬盘
slave	2	四核，2.8 GHz，4 GB 内存，512 GB 机械硬盘
slave	3	双核，2.8 GHz，4 GB 内存，512 GB 机械硬盘
slave	2	双核，2.8 GHz，4 GB 内存，256 GB 固态硬盘
slave	4	双核，2.2 GHz，4 GB 内存，512 GB 机械硬盘
slave	8	单核，2.2 GHz，2 GB 内存，512 GB 机械硬盘

所有机器的操作系统均采用 Ubuntu 14.04 版本，Hadoop 版本均为 2.6.0，JDK 版本均为 1.7。实验数据为随机生成的由小写英文字母组成的 10 个不同大小的数据集，并将其分为两组，第一组的数据量较小，分别是 1 GB、2 GB、3 GB、4 GB、5 GB；第二组的数据量则较大，分别是 20 GB、40 GB、60 GB、80 GB、100 GB。

分别使用默认调度策略与上述动态任务调度策略进行不同实验，得到的结果如图 1、2 所示。

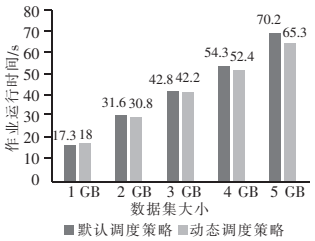


图 1 数据集较小的一组作业运行时间

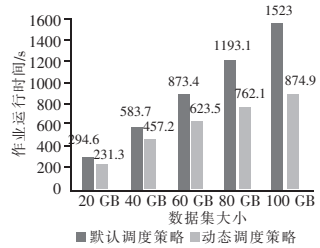


图 2 数据集较大的一组作业运行时间

通过实验结果可以明显看出，动态调度算法相比默认调度算法有很大优势。由于动态调度算法需要在节点计算过程中进行额外的排序和划分，所以在数据量较小时算法优化所带来的时间精简与这部分计算时间相抵消，导致优化幅度有限。但当数据量增大到 10 GB 为单位的程度时，动态调度策略所带来的时间收益极为明显。因此，本调度算法实现了在异构环境下减少任务执行时间的目的，尤其对于 10 GB 级别以上大数据处理，更具有现实意义。

同时在执行 20 GB 数据集的过程中，采集了 150 ~ 200 s 两种调度策略的 CPU 利用率和内存利用率。采集结果如图 3、4 所示。

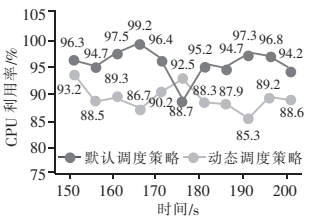


图 3 执行 20 GB 数据集任务过程中 CPU 利用率

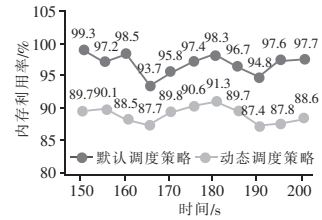


图 4 执行 20 GB 数据集任务过程中内存利用率

从图中可以看出，在使用默认调度策略执行时，CPU 利用率和内存利用率均居高不下，长时间维持在 95% 甚至更高。长时间过高的负载势必会导致性能的损耗，既会降低接下来任务的执行能力，而且还会加快硬件的损耗。而使用动态调度策略则可以使 CPU 利用率和内存利用率更多地维持在 90% 以下，这表明该方法可以更加合理地使用系统资源。由此可以看出，动态调度策略在资源利用率方面也体现了它的优势。

4　结束语

在本文中,为了应对异构环境下由于木桶效应导致的任务执行时间过长,提出了一种将执行节点排序与任务划分相结合的动态算法,使得最高性能节点执行最复杂任务,次高性能节点执行次复杂任务并依此类推。在评价节点性能高低与任务相似程度的过程中都充分考虑到了不同条件下影响因子的权重不同所导致的影响。最后经过实验验证,发现本算法确实在减少任务执行时间、合理化资源利用率方面有着明显的提升效果。下一步的工作可以将本算法在处理大量数据方面的优势更加深化,并针对云平台的特点进行优化与改进,将其部署到云平台上,从而最大化地发挥本算法的特点与优势。

参考文献:

[1] 程学旗,靳小龙,杨婧,等. 大数据技术进展与发展趋势[J]. 科技导报,2016,34(14):49-59.

[2] 李国杰. 大数据研究的科学价值[J]. 中国计算机学会通信,2012,8(9):8-15.

[3] Manyika J,Chui M,Brown B,*et al*. Big data:the next frontier for innovation,competition,and productivity[EB/OL]. (2012-10-02)[2016-12-25]. http://www.mckinsey.com/Insights/MGI/Research/Techonlogy_and_Innovation/Big_data_The_next_frontier_for_innovation.

[4] 刘朵,曾锋,陈志刚,等. Hadoop 平台中一种 Reduce 负载均衡贪心算法[J]. 计算机应用研究,2016,33(9):2656-2659.

[5] Zaharia M,Konwinski A,Joseph A,*et al*. Improving MapReduce performance in heterogeneous environments[C]//Proc of USENIX Symposium on Operating Systems Design & Implementation. 2008:29-42.

[6] Snavely A,Wolter N,Carrington L. Modeling application performance by convolving machine signatures with application profiles[C]//Proc of IEEE International Workshop on Workload Characterization. 2001:149-156.

[7] Yong M,Garegrat N,Mohan S. Towards a resource aware scheduler in Hadoop[C]//Proc of the 7th IEEE International Conference on Web Services. Washington DC:IEEE Computer Society,2009:102-109.

[8] Xie Jiong,Yin Shu,Ruan Xiaojun,*et al*. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters[C]//Proc of IEEE International Symposium on Parallel & Distributed Processing Workshops and PHD Forum. 2010:1-9.

[9] Sun Xiaoyu,He Chen,Lu Ying. ESAMR:an enhanced self-adaptive MapReduce scheduling algorithm[C]//Proc of the 18th International Conference on Parallel and Distributed Systems. Washington DC:IEEE Computer Society,2012:148-155.

[10] Rasooli A,Down D. A hybrid scheduling approach for scalable heterogeneous Hadoop systems[C]//High Performance Computing,Networking Storage and Analysis. Washington DC:IEEE Computer Society,2012:1284-1291.

[11] 侯佳林,王佳君,聂洪玉. 基于异常检测模型的异构环境下 MapReduce性能优化[J]. 计算机应用,2015,35(9):2476-2481.

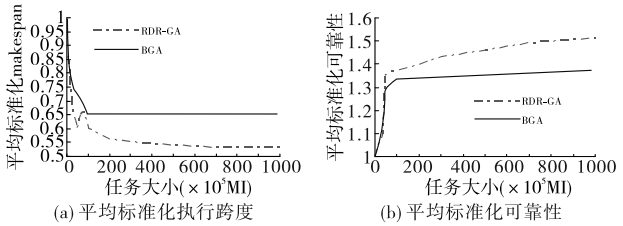
[12] Frank M,Wolfe P. An algorithm for quadratic programming[J]. Naval Research Logistics Quarterly,2006,3(1-2):95-110.

[13] 唐冲. 基于 MATLAB 的非线性规划问题的求解[J]. 计算机与数字工程,2013,41(7):1100-1102.

[14] 郑思. 大规模数据处理系统中 MapReduce 任务划分与调度关键技术研究[D]. 长沙:国防科学技术大学,2014.

[15] Massie M,Li B,Nickoles B,*et al*. Monitoring with Ganglia[M]. Sebastopol:O'Reilly Media,2012:20-63.

(上接第 1394 页)工作流调度问题,提出一种基于时间依赖的资源信誉度评估模型及算法。可靠性驱动信誉度利用任务失效率对资源信誉度进行定义,并应用指数失效模型评估任务可靠性。算法以遗传模型作为基础,以工作流执行跨度和可靠性的同步最优化作为进化目标,同时设计新的进化和评估机制,实现了较好的寻优能力。实验结果表明,算法不仅能以更精确的信誉度改进工作流应用执行可靠性,而且以比传统遗传算法更快的收敛速度得到进化更优解。



(a) 平均标准化执行跨度　　(b) 平均标准化可靠性

图7　算法性能

参考文献:

[1] Buyya R,Yeo S,Venugopal S,*et al*. Cloud computing and emerging IT platforms:vision,hype,and reality for delivering computing as the 5th utility[J]. Future Generation Computer Systems,2011,25(6):599-616.

[2] Wu Fuhui,Wu Qingbo,Tan Yusong. Workflow scheduling in cloud:a survey[J]. Journal of Supercomputing,2015,71(9):3373-3418.

[3] Juve G,Chervenak A,Deelman E,*et al*. Characterizing and profiling scientific workflows[J]. Future Generation Computer Systems,2013,29(3):682-692.

[4] 景维鹏,吴智博,刘宏伟,等. 多 DAG 工作流在云计算环境下的可靠性调度方法[J]. 西安电子科技大学学报:自然科学版,2016,43(2):83-88.

[5] Topcuoglu H,Hariri S,Wu Minyou. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE Trans on Parallel & Distributed Systems,2012,13(3):260-274.

[6] Zheng Wei,Sakellariou R. Budget-deadline constrained workflow planning for admission control[J]. Journal of Grid Computing,2013,11(4):633-651.

[7] Arabnejad H,Barbosa J G. A budget constrained scheduling algorithm for workflow applications[J]. Journal of Grid Computing,2014,12(4):665-679.

[8] 闫歌,于炯,杨兴耀. 基于可靠性的云工作流调度策略[J]. 计算机应用,2014,34(3):673-677.

[9] 马俊波,殷建平. 云计算环境下带安全约束的工作流调度问题的研究[J]. 计算机工程与科学,2014,36(4):607-614.

[10] Doğan A,Özgüner F. Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems[J]. Journal of Computer,2013,48(3):300-314.

[11] Dongarra J J,Jeannot E,Saule E,*et al*. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems[C]//Proc of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures. New York:ACM Press,2007:280-288.

[12] 王岩,汪晋宽,王翠荣,等. QoS 约束的云工作流调度算法[J]. 东北大学学报:自然科学版,2014,35(7):939-943.

[13] Chen Weiwei,Deelman E. WorkflowSim:a toolkit for simulating scientific workflows in distributed environments[C]//Proc of the 8th International Conference on E-Science. New York:IEEE Press,2012:1-8.