(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 106934537 A
(43)申请公布日 2017.07.07

(21)申请号 201710121606.4

(22)申请日 2017.03.02

(71)申请人 北京工业大学
地址 100124 北京市朝阳区平乐园100号

(72)发明人 徐秀杰 孙婷 肖创柏 田国忠

(74)专利代理机构 北京思海天达知识产权代理有限公司 11203
代理人 刘萍

(51)Int.Cl.
G06Q 10/06(2012.01)

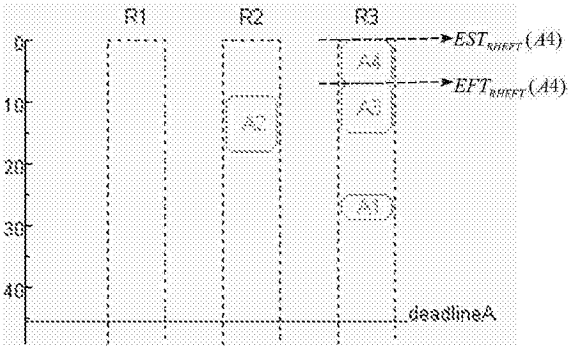
权利要求书1页 说明书6页 附图1页

(54)发明名称

基于反向工作流调度的子期限获取优化方法

(57)摘要

基于反向工作流调度的子期限获取优化方法属于网格或云计算领域，有期限约束的工作流在一组固定资源上调度时任务子期限是一个重要的参数，现有方法所获取的子期限有时并不能保障剩余任务的完成，本发明提供新方法来获取合理可用的子期限。针对工作流调度，提出的RHEFT方法通过工作流反向思想首先对任务求反向权值并排序，然后依次取出任务逐个遍历资源获取反向任务的最早开始时间和完成时间，最后对应求出子期限及最迟开始时间。这个过程不仅考虑剩余关键路径长度，还考虑到DAG并行度的影响，使得每个任务在子期限约束下到资源映射后仍能确保其余任务有足够调度时间。通过DAG并行度变化对子期限的影响，验证了本发明获取的子期限更为合理有效。



CN 106934537 A

1.基于反向工作流调度的子期限获取优化方法，其特征在于，步骤如下：

步骤(1)：计算每个任务n_i的到入口任务n_{entry}的最大平均距离参数，也就是任务n_i的反向权值$rank_r(n_i)$，如公式(1)，反向是相对于HEFT算法中的向上权值而定义的；

$$rank_r(n_i)=\overline{w_i}+\max_{n_p\in prec(n_i)}(c_{p,i}+rank_r(n_p)) \quad (1)$$

其中：$prec(n_i)$表示任务n_i的所有直接父任务，n_p为任务n_i的其中一个直接父任务，$\overline{w_i}$为任务n_i在R中各资源上执行时间平均值，$c_{p,i}$父任务n_p到任务n_i的数据通信时间花费，$rank_r(n_p)$为父任务n_p的向下权值；

因为公式是需要迭代计算的，首先需要计算入口任务n_{entry}，而n_{entry}没有直接父任务，其反向权值$rank_r(n_{entry})=\overline{w_{entry}}$，$\overline{w_{entry}}$为任务$n_{entry}$在R中各资源上执行时间平均值；

步骤(2)：任务排序；

按照DAG中每个任务n_i的$rank_r(n_i)$降序排列，得到任务到资源的调度映射列表，使得出口任务n_{exit}优先调度，入口任务n_{entry}最后调度；

步骤(3)：选择资源；

从调度列表中按顺序取出任务n_i，从R中寻找具有最早完成时间的任务的资源的时隙；若n_i为第一个n_{entry}任务，直接判定R中各资源的最小执行时间就能找到资源；否则，由于反向，任务n_i所有子任务均已经映射到资源，先要计算出任务n_i所有子任务到各个资源的最大传递时间；当前任务在满足最大传递依赖的基础上找到各个资源上的可用时隙，取具有最小结束时间的资源实现调度映射；任务n_i映射到资源后以求得其执行最早结束时间$EFT_{RHEFT}(n_i)$和开始时间$EST_{RHEFT}(n_i)$；

步骤(4)：计算任务子期限和最迟开始时间；

任务n_i的子期限或者最迟完成时间$subDL(n_i)$定义为：

$$subDL(n_i)=D-EST_{RHEFT}(n_i) \quad (2)$$

其中：D为DAG期限

任务n_i最晚开始时间$LST(n_i)$定义为：

$$LST(n_i)=D-EFT_{RHEFT}(n_i) \quad (3)$$

步骤(5)：调度列表中去除任务n_i，重复步骤(3)(4)(5)，直到DAG所有任务完成向资源的调度映射，输出每个任务计算结果$subDL(n_i)$、$LST(n_i)$。

基于反向工作流调度的子期限获取优化方法

技术领域

[0001] 本发明属于网格计算、云计算领域，具体地说，是一种涉及有期限约束的工作流在一组固定资源上调度时任务子期限获取优化方法。

背景技术

[0002] 在有期限约束的DAG工作流共享一组静态资源的调度中，子期限通常作为判定每个任务优先级的计算参数之一。另外，有关用户调度费用的优化，为了得到最合理的资源分配，考虑到每个任务允许费用优化的程度，子期限通常是一个重要指标[1,2]。因此，合理获取每个任务的子期限具有重要的现实意义。

[0003] 与著名的HEFT调度算法[3]假设相同，假定一个DAG多个任务节点需要映射到一组异构分布式计算资源R上并行调度。每个有向边两端的节点对应于父、子任务之间的，由于控制和数据依赖关系的存在，所有的父任务完成后子任务才能执行。根据著名的HEFT算法假设，每个任务n_i在各计算资源上的执行时间花费已知的，资源带宽假定为1，$c_{i,j}$表示任务n_i传递到n_j的数据通信时间花费，映射在同一资源上的任务间通信时间花费为0。在DAG中，没有任何父任务的任务被称为入口任务，表示为n_{entry}，而没有任何子任务的任务被称为出口任务，表示为n_{exit}。如果给定工作流在DAG中包含了多于一个的入口任务或出口任务，可以生成一个零花费的伪入口任务。这样的假设并不会影响到工作流调度时的执行花费[4]。

[0004] HEFT算法其关键思想分为两步，第一是根据任务的执行时间以及与父任务的数据传输时间计算得到这个任务到出口任务之间的最大距离，即向上权值$rank_u(n_i)$，如公式(1)：

[0005] $$rank_u(n_i)=\overline{w_i}+\max_{n_j\in succ(n_i)}(c_{i,j}+rank_u(n_j)) \quad (1)$$

[0006] 其中：$\overline{w_i}$为任务n_i在R中各资源执行时间花费的均值，$succ(n_i)$为n_i的直接子任务集合，n_j属于$succ(n_i)$，$c_{i,j}$任务n_i和其子任务n_j之间的数据通信时间花费，$rank_u(n_{exit})=\overline{w_{exit}}$出口任务$n_{exit}$向上权值，$\overline{w_{exit}}$为出口任务$n_{exit}$在R中各资源上执行时间平均值。。

[0007] 根据这个数值进行任务优先级排序。

[0008] 第二个阶段是根据第一个阶段得到的任务排序，选取未被调度的任务中优先级最高的任务，遍历每个处理机，查找到能够最早完成任务的处理机，插入到可用的空闲时间中。

[0009] 现有文献[1,2]关于获取DAG任务子期限的方法通常是基于计算当前任务到出口任务的近似关键路径进行定义的，如公式(2)。

[0010] $$\begin{aligned} &LFT(n_{exit})=Deadline \\ &LFT(n_i)=\max_{n_j\in succ(n_i)}(LFT(n_j)+\overline{c_{i,j}}+\min(w_i)) \end{aligned} \quad (2)$$

[0011] 其中：$succ(n_i)$为任务n_i的所有直接子任务，n_j属于$succ(n_i)$，$\overline{c_{i,j}}$为任务n_i和其所有子任务n_j之间的数据通信时间花费均值，$LFT(n_j)$为子任务n_j的子期限，$\min(w_i)$为n_i在不

同处理器上的最小执行时间。

[0012] 这种获取子期限的方法并没有考虑到并行任务的多少对子期限的影响，在衡量任务紧急程度方面不够准确，具有较大的改进的空间。

发明内容

[0013] 本发明针对提出了对有期限约束的DAG反向调度获取任务子期限的方法。子期限就是要找到所有任务在一组资源上调度时的最迟允许完成时间，也就是每个任务尽可能的推迟完成，直到再推迟会造成整个DAG超出期限。为了达到每个任务的最大推迟完成，用DAG反向逐个调度获取最早完成时间及最早开始时间，通过期限反转即可得到子期限和最迟运行开始时间。DAG的反向调度策略首先对DAG中所有的边做反向处理，即直接子任务和直接父任务的角色交换，然后再采用让每个任务具有最早完成时间的HEFT算法进行调度。实际调度时，为了更快捷地得到反向DAG采用HEFT算法的调度结果，并不需要对DAG做实际反向，只需要基于HEFT算法，实现反向HEFT调度的RHEFT方法。具体步骤如下：

[0014] 步骤(1)：计算每个任务n_i的到入口任务n_{entry}的最大平均距离参数，也就是任务n_i的反向权值$rank_r(n_i)$，如公式(3)，反向是相对于HEFT算法中的向上权值而定义的。

[0015] $$rank_r(n_i)=\overline{w_i}+\max_{n_p\in prec(n_i)}(c_{p,i}+rank_r(n_p)) \quad (3)$$

[0016] 其中：$prec(n_i)$表示任务n_i的所有直接父任务，n_p为任务n_i的其中一个直接父任务，$\overline{w_i}$为任务n_i在固定资源组R中各资源上执行时间平均值，$c_{p,i}$父任务n_p到任务n_i的数据通信时间花费，$rank_r(n_p)$为父任务n_p的向下权值。因为公式是需要迭代计算的，首先需要计算入口任务n_{entry}，而n_{entry}没有直接父任务，其反向权值$rank_r(n_{entry})=\overline{w_{entry}}$，$\overline{w_{entry}}$为任务$n_{entry}$在R中各资源上执行时间平均值。

[0017] 步骤(2)：任务排序。

[0018] 按照DAG中每个任务的$rank_r(n_i)$降序排列，得到任务到资源的调度映射列表，使得出口任务n_{exit}优先调度，入口任务n_{entry}最后调度。

[0019] 步骤(3)：选择资源。

[0020] 从调度列表中按顺序取出第一个任务，从所有资源中寻找具有最早可完成时间的任务的资源时隙。若第一个任务为n_{exit}，直接判定各资源的最小执行时间就可找到资源；否则，由于反向，所取出的第一个任务n_i所有子任务均已经映射到资源，先要计算出n_i的所有子任务到每个资源的最大传递时间。当前任务在满足最大传递依赖的基础上找到各个资源上的可用时隙，取具有最小结束时间的资源实现调度映射。例如一个简单的四任务DAG的反向调度结果如图1所示，任务n_i映射到资源后可以求得其执行最早开始时间$EST_{RHEFT}(n_i)$和最早结束时间$EFT_{RHEFT}(n_i)$。

[0021] 步骤(4)：计算任务子期限和最迟开始时间。

[0022] 任务n_i的子期限或者最迟完成时间$subDL(n_i)$可定义为：

[0023] $subDL(n_i)=D-EST_{RHEFT}(n_i)$ (4)

[0024] 其中：D为DAG期限

[0025] 任务n_i最晚开始时间$LST(n_i)$可定义为：

[0026] $LST(n_i)=D-EFT_{RHEFT}(n_i)$ (5)

[0027] 步骤(5)：调度列表中去除任务n_i，重复步骤(3)(4)(5)，

[0028] 直到DAG所有任务完成向资源的调度映射，输出每个任务计算结果subDL(n_i)、LST(n_i)。

[0029] 通过本发明所获取到有期限DAG在一组资源上调度时的每个任务子期限和最迟开始时间，由于是真正调度得到的结果，每个任务在子期限前完成均可保证所在DAG中剩余任务的如期完成，比起现有以关键路径思想获取的子期限更具有可信性。子期限的获取还可进一步应用到费用优化的算法中，作为费用优化时选择资源的判定基准。

附图说明：

[0030] 图1：反向DAG采用HEFT算法调度结果示例

[0031] 图2：低并行度简单DAG示例数据

[0032] 图3：具有高并行度的简单DAG示例数据

具体实施方式

[0033] 有期限约束工作流调度的子期限获取优化方法的算法描述如下：

[0034] 输入：一组计算资源R，一个DAG各任务在资源组上的运行时间矩阵W，任务间数据传递矩阵C，DAG期限D

[0035] 1：根据任务所在层倒序排列任务，同一层按原有顺序排列

[0036] 2：根据公式(2)计算每个任务n_i反向权值$rank_r(n_i)$

[0037] 3：对所有任务按反向权值排序得到未映射任务列表unMapList

[0038] 4：WHILE(unMapList≠Φ)DO

[0039] 5：取出反向权值最大的任务n_i

[0040] 6：根据DAG反向调度思想所有子任务(反向父任务)已被映射到资源，获取n_i的所有子任务

[0041] 7：对各处理器上计算n_i的最早可结束时间FT，(FT＝子任务执行结束时间+与子任务数据传递时间+该处理器上的执行时间)的，得到任务映射到各处理器的最早完成时间$EFT_{RHEFT}(n_i)$及对应最早开始时间$EST_{RHEFT}(n_i)$。

[0042] 8：从unMapList去除任务n_i

[0043] 9：对G_k每个任务n_i根据公式(4)计算subDL(n_i)和公式(5)计算最晚开始时间LST(n_i)

[0044] 10：END WHILE

[0045] 11：返回DAG中各任务子期限和最迟开始时间

[0046] 设置DAG并行度0.5(每层并行任务均值与处理器个数关系)，随机生成简单DAG1(A)如图2，图中的圆圈表示任务号，任务之间的边方向均为反向，边权重为数据传递时间花费，并给出了各任务在一组处理器上的任务执行时间花费。

[0047] 若DAG期限为38.8，根据RHEFT算法，按照本专利的实现方法进行逐步计算：

[0048] 步骤(1)：通过上述给定DAG示例数据，基于公式(3)计算每个任务的反向权值$rank_r(n_i)$：

[0049] $rank_r(A_1)$：7.925 $rank_r(A_2)$：19.175 $rank_r(A_3)$：31.175

[0050] $rank_r(A_4)$：20.95 $rank_r(A_5)$：43.6

[0051] 步骤(2)：按照DAG中每个任务的$rank_d(n_i)$降序对任务排序，得到任务调度顺序为：A5，A3，A4，A2，A1，依次放到调度映射列表。

[0052] 步骤(3)：从调度列表中按顺序取出一个任务，从所有资源中寻找具有最早可完成时间的任务的资源时隙。

[0053] 假定反向后子任务先调度，子任务传递给父任务，对于每个资源所有已调度子任务传递数据到达该资源的最大传递时间。当前任务在满足传递依赖的基础上找到各个资源上的可用时隙，取具有最早结束时间EFT的资源实现调度映射，同时得到近似最早开始时间EST。A5从0开始，在R_2具有最小执行时间9.1，因此选择资源R_2。第二轮取任务时会取出A3，由于反向传递，A5传递数据给A3时间花费为9.1，选择R_1、R_3、R_4上的允许开始时刻是9.1+3＝12.1，R_2上传递时间为0，允许开始时刻为9.1，因此A3在四个处理器最早结束时间分别为32.3、27.9、31.7、31.9，因此选择资源R_2。其他任务依次类推，每轮选择一个计算，最后得到A1的EST、EFT。具有DAG1(A)的反向调度结果示意如图2所示，调度过程中可以求得每个任务反向执行最早开始时间和最早完成时间，EST、EFT及对应选择的资源具体结果见表1：

[0054] 表1

[0055]

任务反向执行EST	任务反向执行EFT	选择资源R_i
EST(A5)：0.0	EFT(A5)：9.1	选R_2
EST(A3)：9.1	EFT(A3)：27.9	选R_2
EST(A4)：10.1	EFT(A4)：18.8	选R_3
EST(A2)：15.1	EFT(A2)：24.2	选R_1
EST(A1)：27.9	EFT(A1)：35.7	选R_2

[0056] 步骤(4)：按照公式(4)、(5)计算每个任务子期限subDL和最迟开始时间LST，并公式(2)计算任务subDL进行对比，对比结果见表2。

[0057] 表2

[0058]

RHEFT方法LST	RHEFT方法subDL	公式（2）方法subDL
LST(A5)：29.7	subDL(A5)：38.8	subDL(A5)：38.8
LST(A3)：10.9	subDL(A3)：29.7	subDL(A3)：26.7
LST(A4)：20.0	subDL(A4)：28.7	subDL(A4)：28.7
LST(A2)：14.6	subDL(A2)：23.7	subDL(A2)：23.7
LST(A1)：3.1	subDL(A1)：10.9	subDL(A1)：3.9

[0059] 由两类方法获取的子期限可知，在DAG每层任务并行度较小使得最多任务数小于处理器个数时，公式(2)方法计算出来各任务子期限通常是大于RHEFT方法获取的子期限。另外RHEFT方法同一层上的任务子期限有更大的差别，且大小顺序与反向权值一致，而公式

(2)方法计算出来各任务子期限同一层大小相对集中，未考虑跟权值的关系。当采用同样的并行度随机生成更多任务的DAG时(20/50/100个任务)，依然具有这样的规律。另外，若按公式(2)计算出的任务A1的子期限为3.9，对于任务A1在各个处理器上的执行时间最小为7.7，所以在子期限内根本没有办法执行该任务。事实上，当采用HEFT算法对DAG1(A)进行调度时，得到的最大完成时间是36.4<38.8，也就说明期限内是可以完成所有任务的，由此也可以看出公式(2)计算子期限方法的不合理。

[0060] 若设置随机产生DAG的并行度提高到2(约2倍于处理器个数随机生成各层任务)，仍采用四个处理器，即某些层中任务并行个数远大于处理器个数，对应具有高并行度的DAG2(B)结构及数据参数如图3，期限为27。对应依然采用两种方法所计算出来子期限进行对比，对比结果见表3。从这个对比结果我们可以看到，RHEFT方法获取的任务子期限仅有n6的子期限是大于现有方法的，而这个任务在各个处理器平均执行时间显然大于其他任务，属于关键路径中的一个任务。当处理器数目保持不变，并行度依然为2，继续随机产生更多任务的DAG时(20/50/100个任务)，同样，现有方法仅有关键路径中的任务子期限是小于RHEFT方法获取的任务子期限。因此，当DAG任务具有高并行度并大于处理器个数时，RHEFT方法得到的任务子期限通常大于现有方法得到的任务子期限。

[0061] 表3

[0062]

RHEFT方法LST	RHEFT方法subDL	公式（2）方法subDL
LST(B10)：24.5	subDL(B10)：27.0	subDL(B10)：27.0
LST(B6)：17.5	**subDL(B6)：24.5★**	**subDL(B6)：22.5★**
LST(B9)：16.9	subDL(B9)：22.5	subDL(B9)：22.5
LST(B4)：14.4	subDL(B4)：19.5	subDL(B4)：19.5
LST(B8)：17.4	subDL(B8)：23.5	subDL(B8)：23.5
LST(B5)：20.3	subDL(B5)：23.5	subDL(B5)：23.5
LST(B7)：13.4	subDL(B7)：17.5	subDL(B7)：19.5
LST(B2)：15.3	subDL(B2)：17.4	subDL(B2)：20.5
LST(B3)：11.1	subDL(B3)：16.9	subDL(B3)：20.5
LST(B1)：5.1	subDL(B1)：9.1	subDL(B1)：10.5

[0063] 从二者的对比可以看到，基于RHEFT算法获取的任务子期限更分散，大小顺序与反向权值一致，约束会更合理。当每个任务临近其子期限时刻执行完成后，比其优先级更低的任务仍然可以执行完成，而基于公式(2)获取的子期限由于没考虑任务的并行性，在高优先级任务临近其子期限剩余时间则有可能完成不了全部剩余任务。

[0064] 参考文献

[0065] [1]Abrishami S,Naghibzadeh M,Epema D H J.Deadline-constrained

workflow scheduling algorithms for Infrastructure as a Service Clouds[J].Future Generation Computer Systems,2013,29(1):158-169.

[0066] [2]Jia Y,Buyya R,Chen K T.Cost-Based Scheduling of Scientific Workflow Application on Utility Grids[C]//International Conference on E-Science and Grid Computing.DBLP,2006:8pp.-147.

[0067] [3]Topcuoglu H.,Hariri S.and Min-You W..Performance-effective and Low-complexity Task Scheduling for Heterogeneous Computing.IEEE Transactions on Parallel and Distributed Systems,2002(13):260-274.

[0068] [4]Sakellariou R.,Zhao H..A hybrid heuristic for DAG scheduling on heterogeneous systems[C]//Parallel and Distributed Processing Symposium,2004.Proceedings.International.IEEE,2004:111.

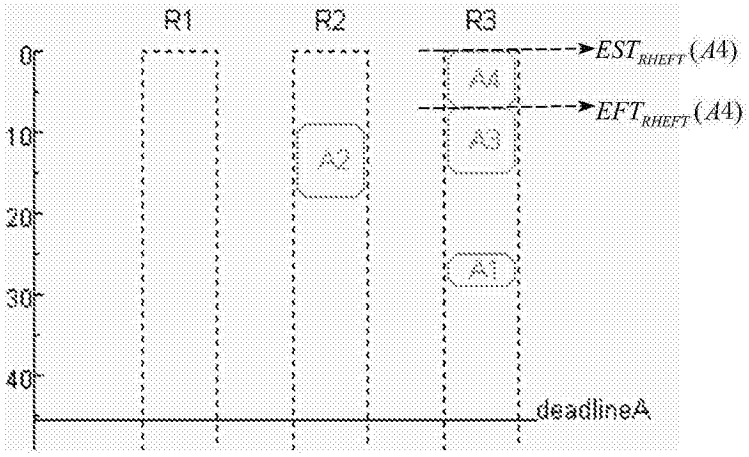
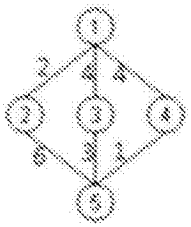
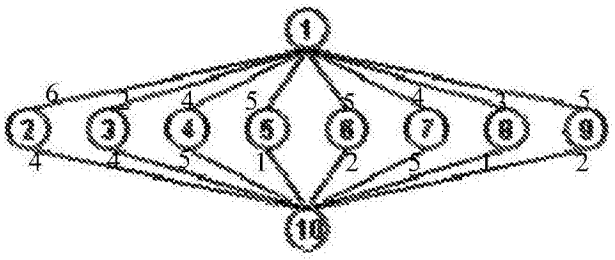


图1



n_i/R_j	R_1	R_2	R_3	R_4
1	8.4	7.8	7.7	7.8
2	9.1	9.5	9	9.4
3	19.7	18.8	19.2	19.3
4	9	9.2	8.7	9.2
5	9.5	9.1	9.5	9.6

图2



n_i/R_j	R_1	R_2	R_3	R_4
1	4	4.3	4.3	4.4
2	2.1	2.1	2.1	2.1
3	5.7	5.8	5.8	5.8
4	5	5	5.1	5.1
5	3.1	3.1	3.2	3.2
6	7	7	7.1	7.6
7	4.1	4.1	4.1	4.2
8	5.9	5.9	6	6.1
9	5.5	5.6	5.7	5.7
10	2.5	2.5	2.5	2.5

图3