论文题目

A Cost-Effective Deadline-Constrained Dynamic Scheduling Algorithm for Scientific Workflows in a Cloud Environment

---

论文作者

Jyoti Sahni and Deo Prakash Vidyarthi

发表期刊信息

IEEE Transactions on Cloud Computing, vol. 6, no. 1, pp. 2-18, 1 Jan.-March 2018

技术问题

云环境与其他分布式环境存在两个主要方面的差异：按需资源供应和现收现付定价模型。因此要在云端实现工作流协调的真正好处是资源可以利用优势并且解决云环境所需的特定挑战的新颖方法。文章提出了一种用于公共云中科学工作流调度的动态、经济有效的期限约束启发式算法。该技术旨在充分利用云计算的优势，同时考虑到虚拟机（VM）的性能可变性和实例获取延迟，以确定最后期限的准时时间表，以较低的成本限制科学工作流程。

现实背景

云环境中的工作流执行涉及两个主要阶段。第一个阶段是资源供应阶段，用于识别和提供计算运行任务所需的资源。在第二阶段，生成计划，并将每个任务映射到适当的计算资源。以前的大部分工作都集中在规划上。分布式系统（如网格和集群）上的工作流，仅限于调度阶段。这是因为，网格和集群环境提供了一个静态的资源池，可以随时执行任务，其配置是提前知道的。然

而，云环境需要上述两个阶段都得到了处理和合并，以产生有效的执行计划。

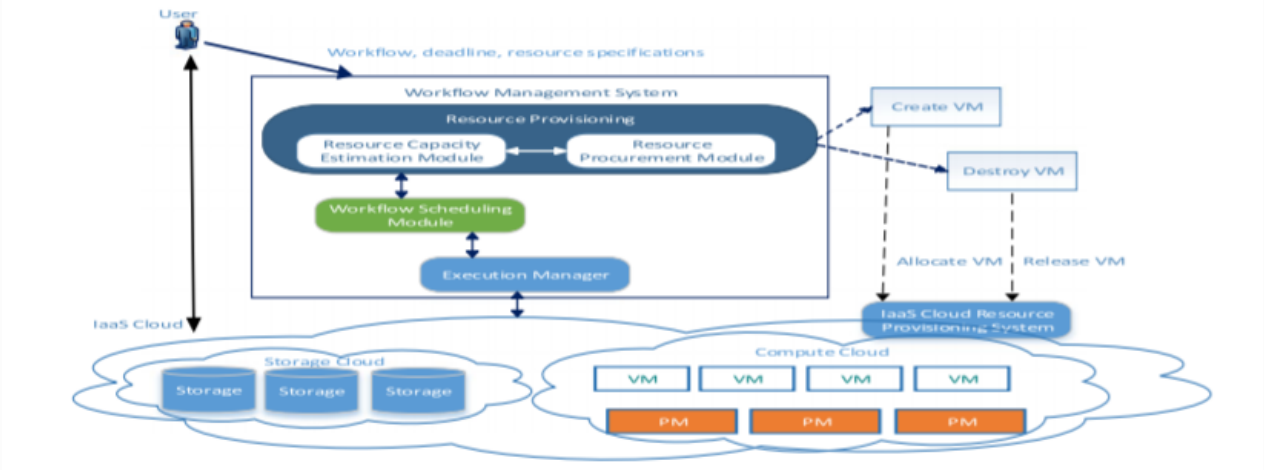
作者思路

作者首先讨论了为云环境量身定制的创新资源调度算法的发展，提出用云环境进行过去工作流的两个阶段，在但是在云环境中，除了执行时间之外，一个重要的参数是经济成本。更快的资源通常更昂贵，因此是在选择适当的服务时的时间成本权衡。所以作者认为云开发的调度策略应该评估各种时间/成本备选方案，以便在避免不必要的成本的同时提供时间有效的解决方案。之后作者介绍了科学的工作流应用模型和弹性资源工作流执行体系结构，之后提出了本文核心的调度算法以及它的解释，最后对该算法进行实验并进行性能评估。

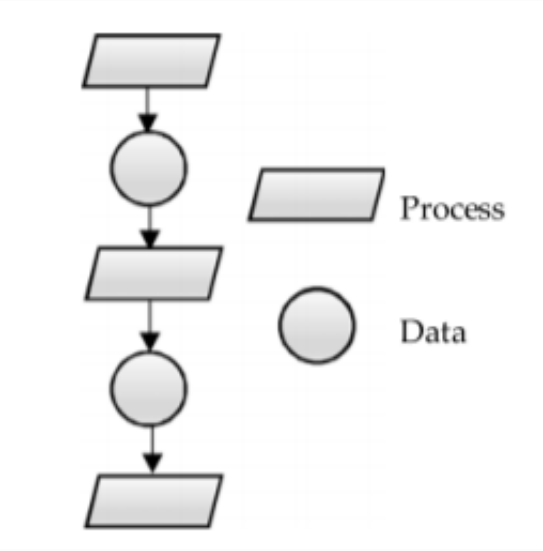
解决方案

构建模型

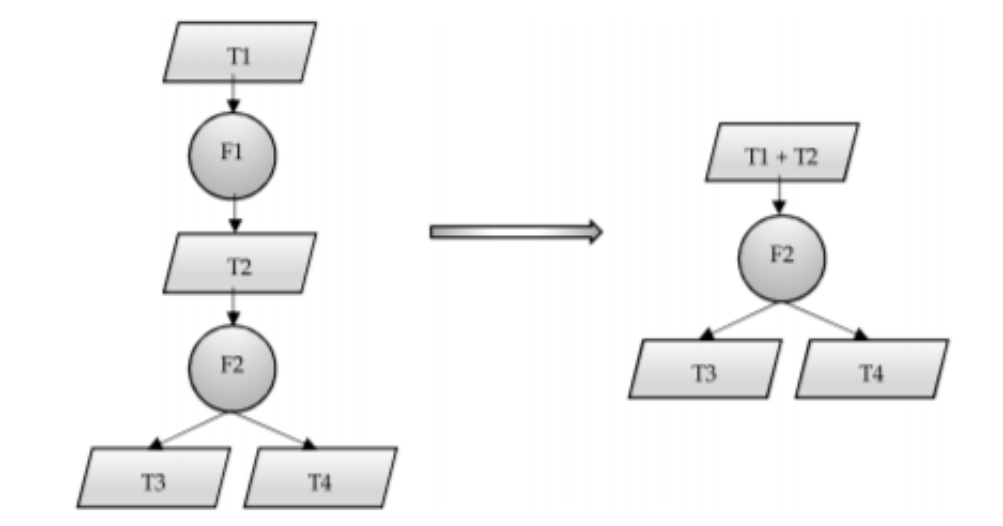
用于在云资源上执行科学工作流的计算平台模型



Pipeline workflow 模型



Pre-processing 模型



算法摘要

Just-in-time Workflow Scheduling

Algorithm 1. Just-in-time Workflow Scheduling

Input:

DAG $W(T, E)$ of a job consisting on n tasks
$1 \times m$ cost matrix C of the m VM types offered by the cloud provider
$n \times m$ ET matrix of execution time of tasks $t_i (i = 1 \, to \, n)$ on each VM type $VM_v (v = 1 \, to \, m)$
$n \times n$ TT matrix of transfer time between tasks
User specified deadline D of the job
VM acquisition time $acquistiondelay$
Lease Time interval *interval*

1. Begin
2. Compute MET_W using Equations 2, 3, 4 and 11
3. If D $\geq$ MET_W
4. Call *Pre-processing(W)*
5. Compute *MET*, *LFT* and *XET* matrices using Equations (2), (8) and (10) respectively
6. $\{t_{entry}\} \leftarrow$ Root nodes of the workflow graph W
7. For each $t_e \in \{t_{entry}\}$
8. $To\_Provision \leftarrow CheapesttaskVMMap(t_e)$
9. Procure a VM instance v_e of type *To_Provision* from the cloud
10. Schedule t_e on v_e at $XST(t_e)$
11. Update VM_Pool_Status
12. End for
13. While all tasks in T are not completed do
14. Send the scheduled tasks for execution to the execution manager
15. Update *AST*, *XFT* of scheduled tasks
16. $to\_be\_scheduled \leftarrow \{t_i \in T \,|\, \forall t_p \in t_i' \; sparent, t_p$ is scheduled and running$\}$
17. $Planandschedule$ ($to\_be\_scheduled$)
18. End while
19. Else
20. Prompt user to specify a deadline above MET_W
21. End If
22. End

Pre-processing (W)

Algorithm 2. Pre-processing (W)

Input: DAG $W(T, E)$ of a job consisting of n tasks.

1. Begin
2. $tksqueue \leftarrow \{t_{entry}\}$
3. While $tksqueue$ is not empty
4. $t_p \leftarrow tksqueue(front)$
5. $S_c \leftarrow \{t_c \mid t_c \text{ is the child of } t_p\}$
6. If $Cardinality\ (S_c) = 1$ and t_c has only one parent t_p
7. Replace t_p and t_c with t_{p+c}
8. Set t_{p+c} as the parent of $t'_c s$ children tasks
9. Update $ET\ (t_{p+c})$
10. Add t_{p+c} to the *front* of $tksqueue$
11. Else
12. Add $t'_p s$ children to the *rear* of $tksqueue$
13. End if
14. End While
15. End

Planandschedule (task_list)

Algorithm 3. Planandschedule (task_list)

1. $Active\_VMs \leftarrow$ List of active VMs in the VM pool
2. For each $t_i \in$ task_list do
3. $vmmap \leftarrow CheapesttaskVMMap(t_i)$
4. $Find\{v_k\} \in Active\_VMs$ s.t. $type(v_k) = vmmap$ and $XST(t_i) \leq \text{CLI}(v_k)$ end time and $XFT\ (t_i) \leq LFT(t_i)$ and for no child t_c of t_i $XST(t_c) > LST(t_c)$
5. If $\{v_k\}$ exists
6. Find the VM v_k, such that the difference between $XIST(v_k)$ and $XST(t_i)$ is minimum
7. Schedule t_i *on* v_k and update $XST(t_i)$
8. Update VM_Pool_Status
9. *Else*
10. Find$\{v_j\} \in Active_VMs$ s.t. $type(v_j) > vmmap$ and $XFT(t_i) \leq \text{CLI}(v_j)$ end time and $\text{XFT}(t_i) \leq LFT(t_i)$ and for no child t_c of t_i $XST(t_c) > LST(t_c)$
11. If $\{v_j\}$ exists
12. Find the VM v_j such that the difference between $XIST(v_j)$ and $XST(t_i)$ is minimum
13. Schedule t_i *on* v_j, update $XST(t_i)$
14. Update VM_Pool_Status
15. Else
16. Procure a new VMv of type $vmmap$ from the cloud at $(XST\ (t_i) - acquistiondelay)$
17. Schedule t_i *on* v at $XST\ (t_i)$
18. Update VM_Pool_Status
19. End if
20. End if
21. End for
22. Deprovision the idle VMs
23. Return

CheapesttaskVMMap(t)

Algorithm 4. CheapesttaskVMMap(t)

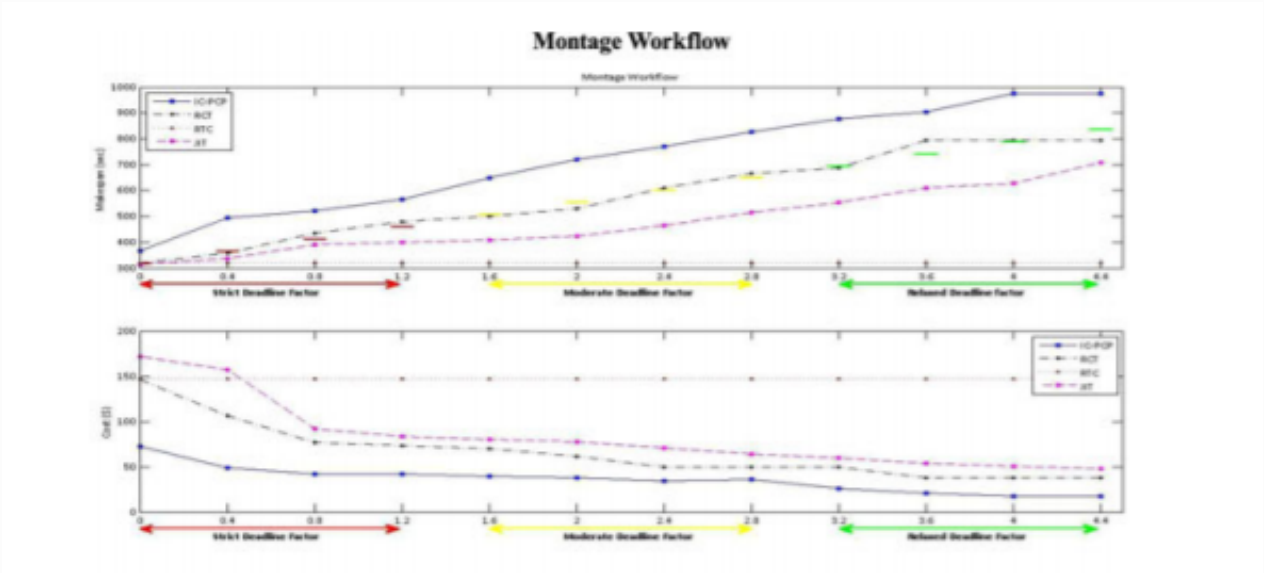
1. Begin
2. $taskvmmap = \phi$
3. If t is not an entry task then
4. $lastParent \leftarrow arg\,(\max_{t_p \in t's\ parent}\{XFT(t_p)\})$
5. $v_p \leftarrow VM\ on\ which\ lastParent\ is\ running$
6. $temp \leftarrow \max(XFT(lastParent), \max_{t_p \in t's\ parent\ and\ t_{p \neq lastParent}} \{XFT(t_p) + TT(t_p, t)\})$
7. If $((temp \geq XIST(v_p))\text{and}(temp + XET(t, type(v_p))) \leq D)$ then
8. $XST(t) \leftarrow temp$
9. $taskvmmap \leftarrow type(v_p)$
10. Return $taskvmmap$
11. Else
12. $XST(t) \leftarrow \max_{t_p \in t's\ parent}\{XFT(t_p) + TT(t_p, t)\}$
13. End if
14. Else
15. $XST(t) \leftarrow acquisitiondelay$
16. End if
17. Find $\{VM_k\} \in VM_{set}$ for which $(XST(t) + XET(t, VM_k)) \leq D$
18. $VM_j = arg\left(\min_{VM_k}(XET(t, VM_k)/\text{interval}) \times Cost(VM_k)\right)$
19. $taskvmmap \leftarrow VM_j$
20. Return $taskvmmap$

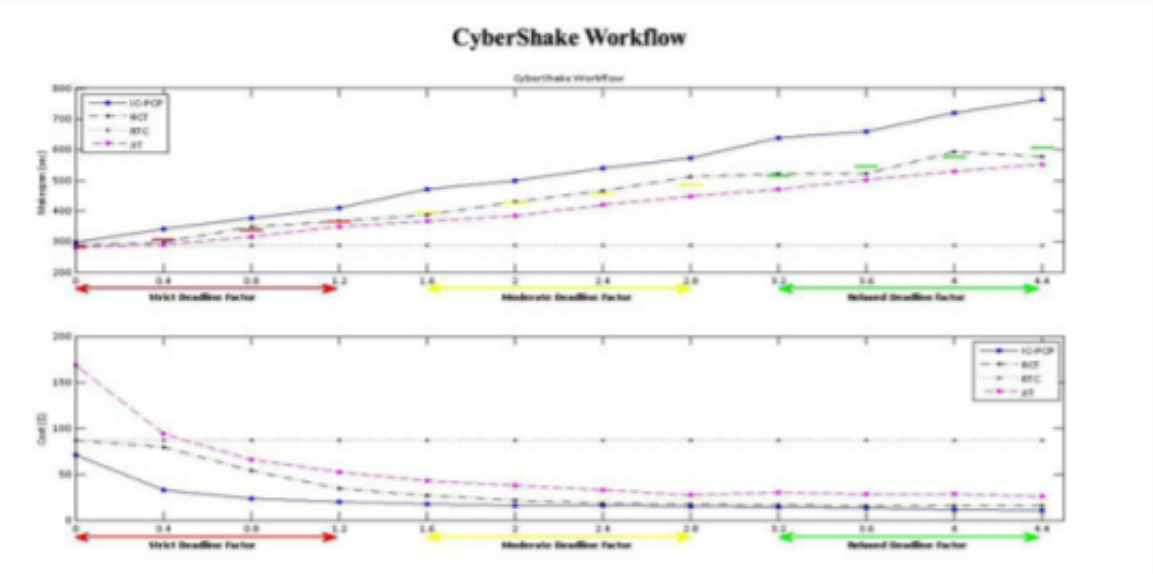
创新贡献

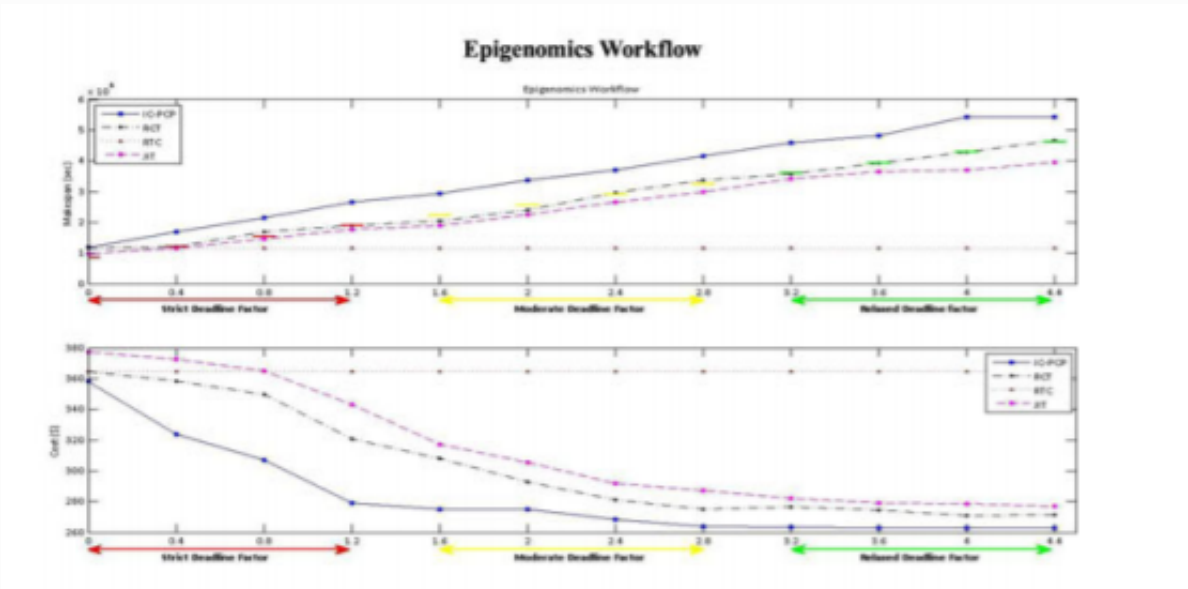
- 为了保持较低的执行成本，在需要资源之前就提供了资源。通过持续监控正在运行的任务，并动态地为后续任务制定经济有效的调度决策，以确保不违反最后期限限制，从而实现满足最后期限的目标。
- 利用可利用的空闲时间和宽松的最后期限，以产生更便宜的时间表和更低的执行成本。
- 在四个著名的工作流上进行的模拟实验表明，与其他最先进的启发式算法（IC-PCP、RCT和RTC）相比，该算法在满足最后期限方面显示出最高的命中率。
- 提出的调度算法解决了云平台的三个主要问题：虚拟机性能变化、资源获取延迟和云资源的异构性。它有潜力成为云资源管理的一个很好的候选者。有人提出，在未来，这项工作有潜力包括对任务和虚拟机故障的稳健性，这些故障可能会对整个工作流执行时间产生不利影响。

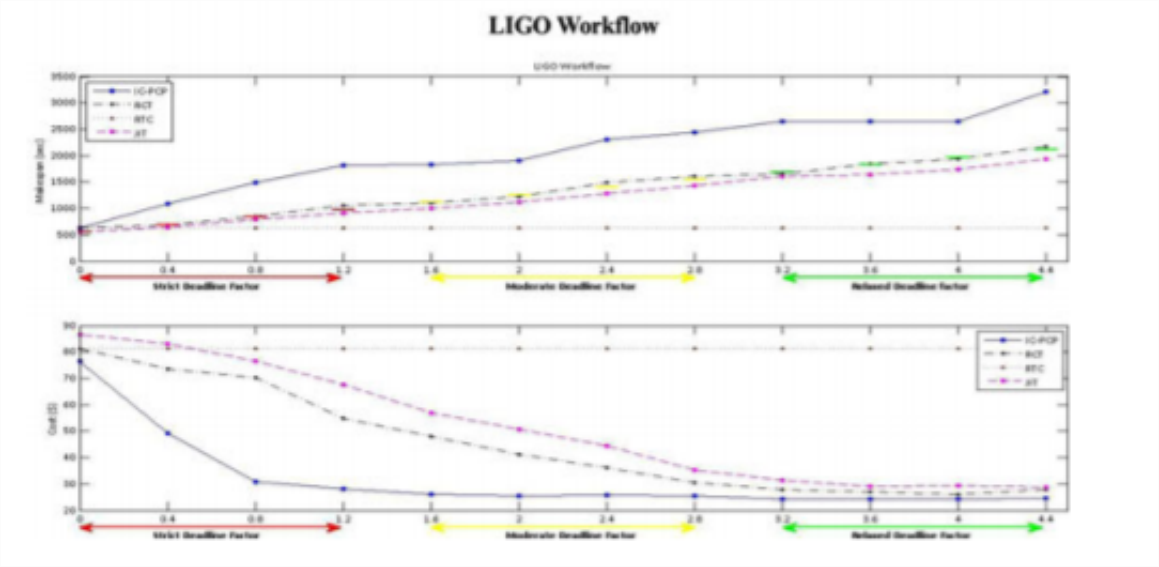
效果评价

在四个著名的工作流上进行的模拟实验表明，与其他最先进的启发式算法（IC-PCP、RCT和RTC）相比，该算法在满足最后期限方面显示出最高的命中率。与用于类似目的的最佳基线算法rtc相比，所提出的算法jit-c生成的调度平均成本降低34%。









个人感想

本文的优势

此文让在云中执行科学应用程序做出适当的资源调配和调度决策，满足了用户定义的期限的同时将总体执行成本降至最低。考虑到云计算环境提供的巨大潜力，这种方法是很有突破性的。

本文的劣势

该工作流系统尚未包括查询能力，例如更改截止日期并相应地修改对成本的影响。