# A Novel Workflow Based Data Processing Platform as a Service

Ming Xu[1], Feng Xie[2]
[1]School of Computer Science, [2]Department of Automation
[1]Beijing University of Posts and Telecommunications
[2]Tsinghua University
Beijing, China
[1]xum@bupt.edu.cn,
[2]xief10@mails.tsinghua.edu.cn

Haohan Wang[3], Zhen Chen[4]
[3]School of Computer Science, [4]RIIT
[3]Beijing University of Posts and Telecommunications
[4]Tsinghua University
Beijing, China
[3]WangHaohan.of.BUPT@gmail.com
[4]zhenchen@tsinghua.edu.cn

***Abstract*—This paper presents a novel platform as a workflow engine service which helps users to utilize cloud computing for addressing problems of large scale data processing without knowledge of underlying configuration details for computing applications. Our platform provides elastic resources for workflow applications to ensure that users leverage resources efficiently with the consideration of their budgets. To achieve this goal, we propose a cost-oriented scheduling strategy to guarantee workflow applications to match appropriate resources and to be scheduled in a minimum makespan, These workflows consist of serial and parallel tasks. In addition, we discuss the architecture and the key modules of our platform.**

***Keywords- workflow; Paas; scheduling; cloud comptuing***

## I. INTRODUCTION

Large scale data processing applications have been used more and more widely in many areas such as monetary, Internet, and scientific research. Some challenges are generated in this trend: a great demand for high-performance computing resource and large scale storage resources are inevitable growing. To clear away the bottleneck generated by large scale data processing, cloud computing, a typical high performance computing technology is expected to provide shared resource, software, information on-demand. The user can obtain different resources whatever he wants. Cloud computing provides services to application users in various levels of abstraction and type of service. For example, infrastructure virtualization enables provider such as Amazon to offer virtual machine for user applications, platform as a Service (PaaS), such as Google application Engine expose a higher level software development and runtime environment for building and deploying applications on cloud infrastructures without worrying about the complex details of managing the underlying hardware, software, provisioning host capabilities and the complete life cycle of software. Such services may expose domain specific concepts for rapid applications development. Further up in the cloud stack is Software as a service (Saas) providers who offer end-users with standardized software solutions that could be integrated into existing applications.

As a modeling technology, workflow, which simplifies the complexity of execution and management of applications, is used widely in business area. Utilizing this technology requires developers to design an application consisted of a series of computation activities and data-transfer activities. With Directed Acyclic Graphs (DAGs) being a convenient model to represent workflows, the vast amount of literature on DAG scheduling is of relevance to the problem of workflow scheduling [1,2,3,4]. In recent years, there has been a revival of interest in the context of problems especially motivated by scientific workflow execution and heterogeneous environments [5,6,7,8,9]. The majority of these works is mainly to achieve an optimal scheduling such as minimizing the workflow execution time or minimizing resource utilization under a given deadline. In this paper, we present an easy-to-use workflow engine integrated into PaaS cloud computing. The platform users can develop a large scale data processing application by leverage cloud resources and gain the benefits from using parallel computing capability such as map-reduce more simplify. We propose SPCP, a parallel tasks' workflow scheduling algorithm based on CPOP [2] and integrate it into our cost-oriented scheduling strategy to ensure delivering high aggregate performance to users on the base of their budgets.

The remainder of this paper is organized as follows. Section 2 presents the model and architecture of our platform. Section 3 describes our workflow scheduling strategy. System experimental results are descripted in Section 4. Finally, we draw conclusions and give ideas for future work in Section 5.

## II. THE PLATFORM ARCHITECTUTE

### A. System model

A workflow models a process as a series of activities that simplifies the complexity by detaching the control flow and business logic modules from the applications. Before we discuss the details of the architecture, we give some important formalized definition in our platform as follows:

*a) Definition 1:* Acitivity is abstract description of a execution step in a whole workflow, represented as four tuples (*Id*, *Type*, *Pre*, *Se*)，where *Id* denotes the unique identifier of a activity, *Type* denotes the type of a activity, which can divide into two types in details, serial activity and parallel activity. Serial activity only can be run on a single

machine in its lifetime while parallel activity can be run on multiple machines. *Pre* denotes a set of predecessor of the current activity. *Se* denotes which workflow should be attached by current activity.

*b) Definition 2:* Workflow is an abstract of an operational process, we can formulate it as a three tuples (*id*, *S*, *D*), where *id* denotes unique identify of a workflow. *S* denotes a set of activities consist of a workflow. *D* denotes a set of data be processed in each activity of a workflow.

### *B. Platform architecture*

Fig. 1 presents the software architectural view of our workflow management system using cloud resources to support the execution of a workflow application. The system consists of three major software layers. The underlying layer provides infrastructure supported distributed storage capability and parallel computing mode such as map-reduce. The middle layer is workflow engine service which provides services for infrastructure management and application execution management. This layer is composed of modules as follows:

- Event Service is responsible for communicating and message delivering between components of each activity and host process.
- Queue Service stores all the workflow instances to wait for scheduling and executing by workflow engine.
- Rule Service is responsible for combining activities into a workflow specified by users
- Communication Service is responsible for communicating of multiple components deployed on same or different sites.
- Workflow Engine generates scheduling to map the activities executed on the associated virtualized node. It creates one thread to scheduling for each workflow and involves two key modules to implement two levels scheduling. Resources selection module determines which resources composition could be most suitable depending on the uses' payment and the workflow evaluated execution time. After that, the schedule execution module maps workflows to resources.
- The OPER-ENV module is responsible for billing management.
- The STI-ENV module provides a simulation environment for pre-executing a workflow to predict the execution time of every activity.
- Develop Kit of workflow designer provides a simple set of APIs and graphical tools for developing applications. These APIs expose different programming abstractions, such as task model and thread model and map-reduce.

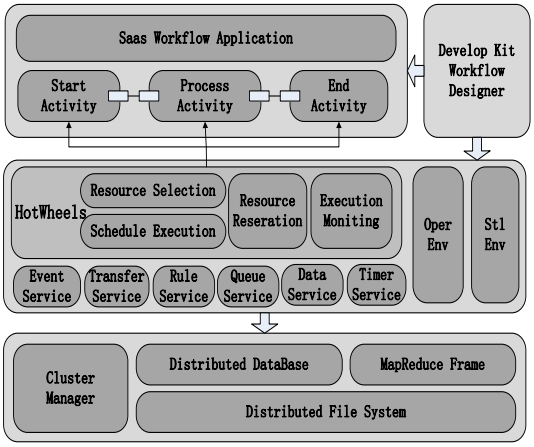


Figure 1. architecture view of the platform

### *C. Application Isolation and Fault Tolerance*

Our platform providing performance isolation is on the foundation of virtual machines. The benefit of using virtualized node for workflow execution, as opposed having direct access to the physical machine, is the reduced need for securing the physical resource from malicious code using techniques such as sandboxing. Since activities handling large scale data generally need to be run for a long time and once failing happens, the consequence could be immeasurable, it is critical to make the workflow fault-tolerant. To achieve this we construct a workflow persistence mechanism. Our platform can persist the state of a workflow into storage system and can retrieve the workflow after an activity crumbled. Certainly it increases the system workloads, for this reason, Our platform persists an activity in database when the workload of virtual machine is less than a threshold and it is determined by user whether this function is enable.

## III. SCHEDULING STRATEGY

A workflow application is described as a set of ordered activates, and each activity has one or more resources satisfying its need for execution. Therefore, resource selection is a scheduling process. The resource of our platform allocated to applications is specified as virtual machine. We assume that there are *m* classes of virtual machines available owned various CPU capacity and memory size. Let *P* denotes the set of price of these VM, as in (1):

$$P = (p_1, p_2, \ldots, p_m) \tag{1}$$

where $p_k$ denotes the price of $k^{th}$ classes of VM. To simplify the problem, we give two assumptions: 1) an activity should be executed on virtual machines in non-preemptive manner. That is to say, an activity is executed on a virtual machine without interruption till it finishes, a virtual machine can only be assigned to an activity, and 2) an activity can be executed on one VM or all of VMs if it is a parallel task. Our scheduling strategy with the goal of minimizing the makespan of workflow has two major phases as follows.

### A. Resources selection phase

This phase requires deciding which resources composition is most suitable for users depend on their payment. Let $R$ denotes the set of resources allocated to a user application as in (2):

$$R = (r_1, r_2, ..., r_m) \tag{2}$$

where $r_k$ denotes the number of $k^{th}$ classes of VM. Let $W$ denotes a workflow consisted of $n$ activities, as in (3):

$$W = (a_1, a_2, ..., a_n) \tag{3}$$

Next we give evaluated execution time matrix $T$ and parallel tasks vector $pt$ pre-run by STI-ENV as follows:

$$T = \begin{pmatrix} t_{11} & \cdots & t_{1n} \\ \vdots & \ddots & \vdots \\ t_{m1} & \cdots & t_{mn} \end{pmatrix} \tag{4}$$

$$pt = (pt_1, pt_2, ..., pt_n) \tag{5}$$

$$pt_k = \begin{cases} 1 & \text{if } k^{th} \text{ task is a parallel task} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The evaluated execution time of $i^{th}$ task is calculated by (7)

$$EET(a_i) = \frac{1}{m}\sum_{h=1}^{m} t_{hi} \tag{7}$$

The aim of resources selection problem is minimize the result of $\sum_{i=1}^{n} EET(a_i)$ .We use two different approaches respectively according to whether parallel activities occupy an important proportion in the workflow. If there are amounts of parallel activities, it is as possible as homogeneous high performance VMs should be selected. Otherwise, we should select the VMs composition which can minimize the workflow execution time, it can be analogy for a complete knapsack problem which addressed by dynamic programming algorithm we used. Fig. 2 presents the details of our algorithm.

### B. Resources mapping phase

In terms of the case that all of the activities are not parallelizable, it can be solved by HEFT [2]. In practice, the parallel tasks of data processing are quite common and this paper focuses this situation. First, we define two key terms used in this paper:

$$Degree_u(a_i) = \max_{\forall a_j \in Child(a_i)} \{Degree_u(a_j) + EET(a_i) + DTT(a_i, a_j)\} \tag{9}$$

$$Degree_d(a_i) = \max_{\forall a_j \in Parent(a_i)} \{Degree_d(a_j) + EET(a_i) + DTT(a_i, a_j)\} \tag{10}$$

$Degree_u$ and $Degree_d$ denote the length of the longest path to an activity from start activity and from end activity respectively. We can calculate the $Degree_u$ value of each activity starting from the end activity recursively. The $Degree_d$ can be calculated in an analogy process for $Degree_u$.

```
SelectResouce()
if  sum_{if pt_k=1} EET_k > (1/n) sum_{h=1}^{n} EET_h  then
    find the VM k which minimize sum_{j=1}^{n} t_kj .
    while  num_vmk * p_k < c_power  do
        selectVM(m) and num_vmk ← num_vmk + 1
    end while
else
    f[0..c_power]:=0;
    for i=1..m
        for v = p_i..c_power
            f[v]:= max(f[v],  f[v − p_i] + 1 / ((1/n) sum_{k=1}^{n} t_ik) );
            if f[v − p_i] + 1 / EET_i > f[v] then
                selectVM(i);
            endif
endif
```

Figure 2. The resources selection algorithm

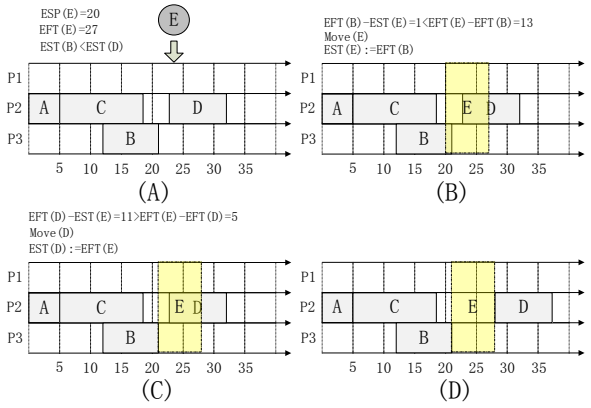


Figure 3. The process of a parallel activity scheduling

Next the data transfer time between activity $a_i$ and activity $a_i$ is defined by (11):

$$DTT(a_i, a_j) = \begin{cases} data_{i,j} / abw * count(R) & \text{if } pk_j = 1 \\ data_{i,j} / abw & \text{otherwise} \end{cases} \tag{11}$$

where $abw$ denotes the average bandwidth of VMs available, and $count(R)$ represents the number of VMs. Then we define that $EST$ and $EFT$ are the earliest start time and the earliest finish time of an activity, and they can be calculated by (12) and (13):

$$EST(a_i) = \max_{\substack{\forall a_j \in Parent(a_i) - SC, \\ \forall a_k \in Parent(a_i) \cap SC}} \left\{ \begin{matrix} EST(a_j) + EET(a_j) + DTT(a_j, a_i), \\ AST(a_k) + EET(a_k) + DTT(a_k, a_i) \end{matrix} \right\} \tag{12}$$

$$EFT(a_i) = EST(a_i) + EET(a_i) \tag{13}$$

where $SC$ denotes a set of already scheduled activities. The $EET$ of a parallel activity is equal to the value evaluated as a serial activity divided by the number of VMs available. In

```
MappingTaskResouce()
Calculate the order of task for each task by Degree_u(a_i) + Degree_d(a_i) in the workflow
Put start task into CP list head.
a_k ← a_start
while a_k is the exit task do
    if a_i ∈ Child(a_k) and Degree_u(a_i) + Degree_d(a_i) == Degree_u(a_start) + Degree_d(a_start)
    put a_i into CP list
    a_k ← a_i
endwhile
Put the remained tasks into Subordinate list and sort them by nonincreasing order of Degree_u values
Select a virtual machine p_cp as a critical-path processor by random.
while the CP list is not empty
    Get a_i from CP list and assign a_i on p_cp
endwhile
while the Subordinate list is not empty
    Get a_i from Subordinate list
    if a_i is a non-parallel task then
        Assign a_i to the processor p which minimizes the EFT(a_i).
    else
        Search the task a_v on all the processors during the time between EST(a_i) and EFT(a_i) and put into Collision list.
        while Collision list is not empty do
            Get task a_v from Collision list and calculate EFT(a_i) − EST(a_v) and EFT(a_v) − EST(a_i)
            if EFT(a_i) − EST(a_v) > EFT(a_v) − EST(a_i) then
                EST(a_i) ← EFT(a_v)
            else    EST(a_v) ← EFT(a_i) and adjust a_v scheduling.
        endwhile
        assign a_i.
endwhile
```

Figure 4. The resources mapping algorithm

the first step of this phase we set the priority of each activity with the result of $Degree_u$ added by $Degree_d$, which maximized value are on the critical-path and have the highest scheduling priority. Then we sorted the remaining activities by decreasing order of $Degree_u$ and put them into scheduling list. After the activities prioritizing phase, we select a critical-path processor by random and schedule the critical-path activities on it. Improvement on previous algorism is parallel actives mixed in, since they occupied all the processors in their lifetime. When a parallel activity $a_p$ arrives, all the processors should be checked whether scheduled activities conflict with the $a_p$. If such activities exist, our algorism finds $a_k$ whose *EST* is earliest conflicted with $a_p$, then compares *EFT* of $a_k$ minus *EST* of $a_p$ and *EFT* of $a_p$ minus *EST* of $a_k$. If the former is larger than the latter, *EST* of $a_k$ should be adjusted to equal to *EFT* of $a_p$, otherwise *EST* of $a_p$ should be adjusted to equal to *EFT* of $a_k$. Fig. 3 illustrates the scheduling process when a parallel activity *E* arrives. Fig. 4 explains the details of SPCP resources mapping algorism.

## IV. SYSTEM EVALUATION AND PERFORMANCE EXPERIMENT

We evaluate HEFT and CPOP against our SPCP algorithm. We randomly generate three workflows and each of them consists of 1000 activities and 400 edges. The only difference among them is number of parallel activities. They include respectively 100, 200 and 500 parallel activities. The execution time of each activity is randomly selected from 2 to 20 time units. The average data transfer time is selected from 2 to 5 time units randomly. The three workflows run on four computers by simulated. SPCP achieves a good perfor-

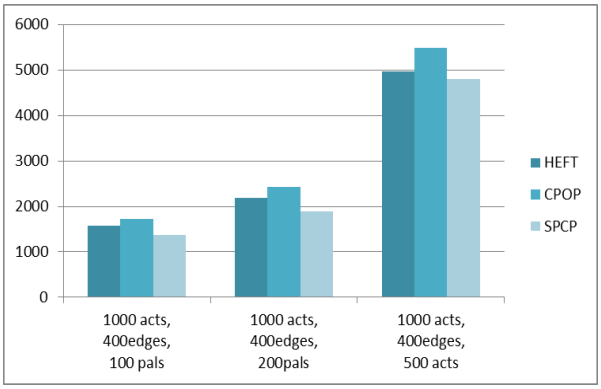


Figure 5. The resources selection algorithm

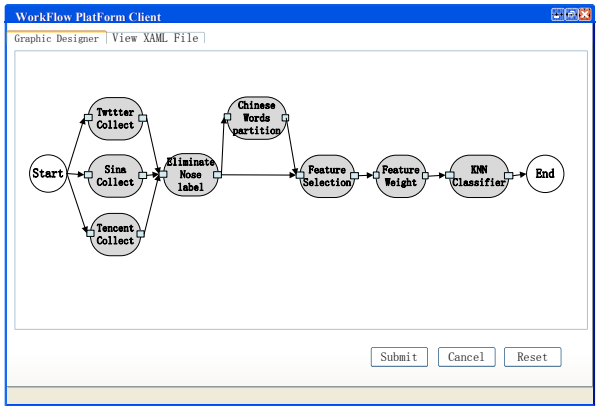


Figure 6. A text mining application submited to the platform

TABLE I. RUNNING TIME OF VARIOUS STAGES OF TEXT MINING

| | data collecting | preprocessing | classification |
|---|---|---|---|
| **10 instance** | 17954 | 23.343 | 5.697 |
| **20 instance** | 15468 | 16.502 | 4.145 |
| **50 instance** | 12522 | 10.302 | 2.987 |

mance for fully random workflows. The reason is that HEFT schedules activities as early as possible so that a parallel activity arrives, it cannot find the idle fragment for the activity which must be delayed in the worst way. Fig. 5 shows the estimated time of our experiments.

Then, we implement a prototype of the platform on Amazon EC2 and utilize it to develop a text mining application. The workflow of the application is illustrated by Fig. 6. The data collecting modules is limited at 2GB data. The workflow generally can be divided into three steps: data collecting, preprocessing, feature extraction and classification. After changing the payment for VMs by simulation, the application can obtain the various VMs composition. Tab. 1 shows the running time of each step.

## V. CONCLUSIONS AND FUTURE WORKS

This paper presents a workflow based data processing PaaS architecture and a cost-oriented scheduling strategy. The experiment results demonstrate that the cost-orient scheduling strategy and SPCP scheduling algorism can deliver the high performance for parallel tasks' workflow execution.

In the future work, we aim to do more experiments on our scheduling strategy, considering multiple QoS targets and multiple workflow scheduling.

## ACKNOWLDGMENT

This work is supported by Ministry of Science and Technology of China under National 973 Basic Research Program Grant No. 2011CD302600, Grant No. 2011CB302805, Grant No. 2011CB302601, and Grant No. 2012CB315800.This work is also supported by China NSFC A3 Program (No.61161140320).

## REFERENCES

[1] Fangpeng Dong, Selim G. Akl. PFAS: A Resource-Performance-Fluctuation-Aware Workflow Scheduling Algorithm for Grid Computing. In Proceedings of IPDPS 2007.

[2] H. Topcuoglu, S. Hariri. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Transaction on Parallel and Distributed Systems, 2002.

[3] Eun-Kyu, Byun. Efficient Resource Capacity Estimate of Workflow Applications for Provisioning Resources. University of Southern California, Technical Report(08-898).

[4] Eun-Kyu Byun. Cost optimized provisioning of elastic resources for application workflows. Future Generation Computer Systems, 2011.

[5] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Resource Allocation Strategies for Workflows in Grids. In IEEE International Symposium on Cluster Computing and the Grid (CCGrid), 2005.

[6] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu and L. Johnsson. Scheduling Strategies for Mapping Application Workflows onto the Grid. In IEEE International Symposium on High Performance Distributed Computing (HPDC 2005), 2005.

[7] R. Sakellariou and H. Zhao. A Low-Cost Rescheduling Policy for efficient mapping of Workflows on Grid Systems. Scientific Programming, vol. 12, 2004.

[8] M. Wieczorek, R. Prodan and T. Fahringer. Scheduling of Scientific Workflows in the ASKALON Grid Environment. SIGMOD Record, volume 34(3), 2005.

[9] J. Yu and R. Buyya. A Budget Constraint Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms Proceedings of the Workshop on Workflows in Support of Large-Scale Science (WORKS06), 2006.