A Methodology for Task placement and Scheduling Based on Virtual Machines

XiaoJun Chen[1], Jing Zhang[1,2], and JunHuai Li [1]
[1] School of computer science and engineering, Xi'an University of Technology
Xi'an, 710048, P.R.China
[e-mail: army.net@163.com]
[2] State Key Laboratory for Manufacturing Systems Engineering, Xi'an Jiaotong University
Xi'an, 710048, P.R.China
[e-mail: ZhangJing@xaut.edu.cn]
*Corresponding author: Jing Zhang

Received January 18, 2011; revised July 18, 2011; accepted September 7, 2011; published September 29, 2011

Abstract

Task placement and scheduling are traditionally studied in following aspects: resource utilization, application throughput, application execution latency and starvation, and recently, the studies are more on application scalability and application performance. A methodology for task placement and scheduling centered on tasks based on virtual machines is studied in this paper to improve the performances of systems and dynamic adaptability in applications development and deployment oriented parallel computing. For parallel applications with no real-time constraints, we describe a thought of feature model and make a formal description for four layers of task placement and scheduling. To place the tasks to different layers of virtual computing systems, we take the performances of four layers as the goal function in the model of task placement and scheduling. Furthermore, we take the personal preference, the application scalability for a designer in his (her) development and deployment, as the constraint of this model. The workflow of task placement and scheduling based on virtual machines has been discussed. Then, an algorithm *TPVM* is designed to work out the optimal scheme of the model, and an algorithm *TEVM* completes the execution of tasks in four layers. The experiments have been performed to validate the effectiveness of time estimated method and the feasibility and rationality of algorithms. It is seen from the experiments that our algorithms are better than other four algorithms in performance. The results show that the methodology presented in this paper has guiding significance to improve the efficiency of virtual computing systems.

Keywords: Virtual machine, virtual computing systems, performance, feature model, task placement, task scheduling

DOI: 10.3837/tiis.2011.09.004

1. Introduction

In high performance computing, the large and complex applications are submitted to the clusters constituted of several computing nodes to execute parallel computing. The set of tasks from applications working in clusters depends on the underlying physical structures of hardware very much [1], so the designers are required to fully understand the characteristics of underlying physical structures. On one hand, the clusters can raise the speed of applications. But on the other hand, they greatly restrict the development progress and increase the running time of applications. When we need to extend the functions of applications or change the structures of clusters, the source codes of applications should be modified or rewritten with a great of work. As a result, the dynamic adaptability of applications are too weak to adapt to the change of requirements. Meanwhile, it is very difficult for traditional clusters to improve the resource utilization, maintain loads balance and realize the peak performances of systems.

To solve above problems, the virtualization are taken to organize the resources in clusters. Researchers add a virtualization layer to the hardware to construct a virtual computing system. If we need to change the functions of applications or the structures of clusters, virtualization layer would greatly reduce the adjustment of source codes in applications development. Some researches focused on this area are as follows: (1) the performances of computing systems, management methods of resources [2][3] and fault-tolerant systems [4]; (2) parallel computing oriented multi-core processor [5][6]; (3) the applications of virtualization to high performance computing with more attention to message-driven, tasks mapping and dynamic load balance [7][8][9][10]; and (4) the deployment of virtualization to clusters with more stress on the design of *VMM*(virtual machines monitors) for high performance computing. These results show that we should make full use of advances of virtualization to implement resources integration and achieve a single system image with better transparency, compatibility and applicability [11] via the page-copy and migration. When we take present parallel programming model to develop the programs and then package them into virtual machines [12], VMM can adjust the deployment of virtual machines with dynamics to implement the optimization of resources and create an environment with scalable management based on the changing trend of loads in virtual machines [13]. The isolation and controllability of the virtual machines can improve IT security with vulnerability management [14]. The distributed memory virtualization [15], multiprocessor virtualization [16], I/O virtualization [17][18] and virtualization management [19][20] are studied to improve the efficiency of multiple virtual machines. It is concluded from theories and practice that the application of virtualization to high performance computing is necessary to lay out a solid foundation to the efficient operation of virtual computing systems in clusters.

Task placement and scheduling are traditionally studied in following aspects: resource utilization, application throughput, task execution latency, and starvation [21][22][23]. The partition of a grid service task into subtasks and the distribution of them on available resources have great influence on the extent of the service reliability and profits [24]. Resource allocation in heterogeneous computing (*HC*) environments should match tasks with machines and schedule the tasks to assigned machines [25]. The mapping of tasks into the machines of a distributed *HC* environment has been an NP-complete problem [26]. To maximize the performance of the system, dynamic mapping is performed when the arrival of tasks is not known a priori. The goal of a dynamic mapping heuristic is to maximize the value accrued of completed tasks in a given interval of time [27]. In addition, the minimization of the execution

time for an iterative task requires an appropriate mapping scheme to match and schedule the subtasks to the processors. Some researchers implement and evaluate a semi-static methodology involving the on-line use of off-line-derived mappings. The off-line phase is based on a genetic algorithm (*GA*) to generate high-quality mappings for a range of values for the dynamic parameters [28]. However, recent breakthroughs in the mathematical estimation of parallel genetic algorithm parameters are applied to the NP-complete problem of scheduling multiple tasks to a cluster of computers connected by a shared bus [29]. Some researchers proposed a load distribution (*LD*) algorithm to achieve better system performance by smoothing out any workload imbalance that may exist in a distributed system [30].

After the virtualization is applied to clusters, the task placement and scheduling are very different from those in traditional clusters, and the focus is more on application scalability and application performance. Many studies have shown that the methodologies of task placement and scheduling not only can affect the applications development and deployment, but also have a great effect on the performances of systems [31][32]. There existed the problem of low peak performances in some high performance computing systems, because the features of applications do not match the architectures of computing systems. The excellent scheme of task placement and scheduling not only can reduce the running time of systems and maintain their loads balance [33][34][35], but also raise the peak performances of systems by improving their resource utilization. Task placement and scheduling, a vital technology to assist the efficient operation of systems, refer to decomposing the tasks into subtasks and then placing them to different layers of virtual computing systems based on the requirements of users [36]. The problems in system-level and application-level are solved respectively in traditional clusters, in which, the task placement and scheduling are generally completed by a management node based on their time-relationships in parallel and serial process, just considering the factors such as load balance in computing nodes. The shortage of systematic planning in synchronization, communication and switching enables the task placement and scheduling generally lack of accuracy and efficiency [37]. Few studies on task placement and scheduling based on virtual machines make us present a corresponding methodology centered on tasks, so we focus mainly on task placement and scheduling with virtualization techniques to improve the performance of systems. On the basis of fully considering the performances of virtual computing systems and the convenience in applications development and deployment, the top-down tasks decompositions and subtasks aggregations are used to complete the placement of subtasks to threads, the mapping from threads to virtual machines and the scheduling of virtual machines in physical machines. We define the problems in section 2 and make an overview of task scheduling algorithms in section 3, and then, construct an model of task placement and scheduling in section 4. Algorithms to solve the goal function in model and perform tasks execution are designed in section 5 and the experiments are made in section 6.The last section concludes this paper.

2. Problems definition

The parallel applications with real-time constraints are out of the scope in this paper. Our study concentrates on the parallel applications without real-time constraints in virtualized high performance computing environments. We assume that virtual machines are scheduled by using the algorithms of the time slices rotation, such as *Credit*. Based on our assumption, the task placement and scheduling based on virtual machines require us to divide the tasks into subtasks.We construct some parallel layers in systems from the perspective of tasks to improve the efficiency of parallel computing and the convenience in developing and

deploying applications [38]. We should solve two problems as follows: What kind of methodologies for task placement and scheduling could maintain the loads balance and produce the minimum running time in clusters after virtualization? And what kind of methodologies for tasks decomposition could enable the development and deployment of applications more satisfy the requirements of designers?

(1) The relationship between tasks decomposition and performances is our first problem. For several subtasks implementing parallel computing, the running time are from the computation, synchronization, communication and switching of tasks [39].The switching time is decided by the quantity of subtasks and their scheduling algorithm. Generally, the fewer the granularities of subtasks are, the shorter is the time in synchronization, and the more the traffic information exists. In parallel layers of subtasks, the systems should consider four layers as follows: 1) If several subtasks are gathered into a thread, they can only execute in a serial way with no communication time, and their synchronization time is equivalent to the running time of frontier tasks. 2) If several subtasks run in a middleware with parallelism, the subtasks take a thread as a unit, whose performance depends on the communication and synchronization time among threads. As the middleware is running as a process in a virtual machine, we only need to install a middleware in a virtual machine. 3) If several subtasks run in a host with parallelism, the subtasks take a virtual machine as a unit, whose performance depends on the communication and synchronization time among virtual machines. 4) If several subtasks run in a cluster with parallelism, the subtasks take a host as a unit, whose performance depends on the communication and synchronization time among hosts. Let i be a layer number of the architecture in virtual computing systems, *ComputationTime* be the time from computation, *SwitchTime* be the time from switching of subtasks, *ComTime* be the time from communication, and *SysnTime* be the time from synchronization, thus, the overall time of four layers in virtual computing systems are:

$$Time = ComputationTime + \sum_{i=1}^{4}(SwitchTime_i + ComTime_i + SysnTime_i) \tag{1}$$

(2) The relationship between tasks decomposition and the development and deployment of applications is our second problem. For several subtasks implementing parallel computing, the fewer the granularities of subtasks are, the easier the implementation of the parallelism is. When there existed more subtasks in applications [40][41], it would be difficult to develop and deploy the application because of the intensive control flows and data flows in parallel subtasks. In parallel layers of subtasks, the systems should consider four layers as follows: 1) Several subtasks are gathered into a thread. They can be developed in the same way as the traditional single-threaded applications development, and they can be deployed in the same way as the traditional single-threaded applications deployment. 2) If several subtasks run in a middleware with parallelism, they can be developed in the way of multithreaded applications.Meanwhile, we should solve the problem of the deployment of multi-threads application to the middleware and the problem of the deployment of middleware to the virtual machines. 3)If several subtasks run in a host with parallelism, they can be developed and deployed in the way of distributed applications with parallel computing. 4) If several subtasks run in a cluster with parallelism, we should maintain the loads balance. Let I_1, I_2, I_3, I_4 be the parallelisms in such four layers, $f_1(.)$ be the membership functions of applications development, $f_2(.)$ be the membership functions of applications deployment, thus, the utility value of four layers in virtual computing systems are:

$$f = \sum_{i=1}^{4}[f_1(I_i) + f_2(I_i)] \quad 0 < I_i \le 1, \forall i \in (1,2,3,4) \tag{2}$$

We think that the tasks decomposition should base on the personal preference of a designer. On the basis of the designer's personal preference being satisfied, the running time limites to the minimum level of utility value, so the problems of task placement and scheduling based on virtual machines are transferred as the problems of tasks decomposition and the mapping of subtasks to different layers of virtual computing systems.

3. Task scheduling algorithms overview

There are many task scheduling algorithms that schedule tasks to processors.In general, task scheduling is presented in two forms: static and dynamic [42]. In static scheduling algorithms, all information needed for scheduling, such as the structure of the parallel application, the execution times of individual tasks and the communication costs among tasks must be known in advance. Static task scheduling takes place during compilation time before running the parallel application. In dynamic scheduling, however, tasks are allocated to processors upon their arrival, and scheduling decisions must be made at run time [43][44][45][46]. Based on the challenges caused by the dynamicity of virtualization and the vagueness of availability requirements in the scheduling strategy of virtual data centers, some researchers have research on the efficient dynamic task scheduling in virtualized data centers with fuzzy prediction [47]. It is a dynamic algorithm to schedule tasks without dependence, and different from our problem.

We design static task scheduling algorithm for the task placement and scheduling based on virtual machines. We select static task scheduling algorithm because many parallel applications have long execution times, and hence they require high quality task scheduler to minimize their running times. Additionally, the static scheduling time of several scientific and engineering applications is much lower than their run time on systems. For example, the execution times of more than 50% of the parallel applications that were run on four real parallel computing systems are between tens to thousands of minutes [48], while the static scheduling times of parallel applications with diverse characteristics, which were scheduled using several static scheduling algorithms, are lower than one second as shown in [49].

Static scheduling algorithms can be broadly classified into three main groups: heuristic algorithms, guided random algorithms and hybrid algorithms [49].

Ⅰ.Heuristic scheduling algorithms move from one point in the search space to another, following a particular rule. Such algorithms, though efficient, search some paths in the search space and ignore others [42][43]. Heuristic scheduling algorithms can be divided into three groups: list-based heuristics, clustering heuristics and duplication heuristics [49]. In list-based scheduling heuristics, each task is assigned a given priority. The tasks are inserted in a list of waiting tasks, such that tasks with higher priority are placed before those with lower priorities. Three steps are then repeated until all the tasks in the list are scheduled: task selection, processor selection and status update. Clustering heuristics trade off inter-processor communication overhead with parallelization by allocating heavily communicating tasks to the same processor. In such heuristics, the tasks are grouped into an unlimited number of clusters [49][50]. Duplication algorithms start by running a clustering or list based algorithm to create an initial schedule. This improvement in performance comes at the cost of increasing the complexity of scheduling process [41].

Ⅱ.Guided random scheduling algorithms mimic the principles of evolution and natural genetics to evolve near-optimal task schedules. Among the various guided random algorithms, *Genetic Algorithms (GA)* are the most widely used for the scheduling problem [19][53][55]. In attempts to obtain schedules of better quality, many well-known metaheuristics, including *Simulated Annealing (SA)* [53], *Tabu Search(TS)* [49][54], *Artificial Immune System (AIS)* [55], *Ant Colony Ooptimization (ACO)* [50], *Particle Swarm Optimization (PSO)* [56],

Simulated Annealing (SA) [34], *Tabu search (TS)* [57], and *Variable Neighborhood Search (VNS)* [35], have been adopted. However, *GA* usually takes more computing efforts to locate the optimal in the region of convergence [49], owing to the lack of local search ability. On the other hand, the trajectory method, such as *VNS* [36], has shown its potential in exploiting the promising regions in the search space with high quality solutions. Nevertheless, it is still prone to premature convergence traps due to the limited exploration ability. Thus, it's a natural choice to consider the hybridization of metaheuristics, also named memetic algorithm (*MA*) in some literatures [53][58], which has been applied to solve scheduling problems [44].

Ⅲ. Hybrid scheduling algorithms. A hybrid scheduling algorithm combines both heuristic algorithms and *GAs*. The *Genetic List Scheduling (GLS)* algorithm [53] is an example of this class of algorithms, but it has greater complexity than other algorithms.

Besides to those, there are also high efficient algorithms for the problem of task scheduling in heterogeneous distributed systems. Examples of these algorithms are: *Dynamic Level Scheduling (DLS)* [59], *Heterogeneous Earliest Finish Time* (*HEFT*) [49], and *Critical Path on a Processor (CPOP)* [49], *Mapping Heuristic (MH)* [60] and *Levelized Min Time (LMT)* [65].Where, *DLS* and *HEFT* algorithms are the improvement of heuristic scheduling algorithms. They are two of the best existing scheduling algorithms for heterogeneous distributed systems [49], and are employed as benchmark scheduling algorithms in many studies [52][54]. The *DLS* algorithm does not schedule tasks between two previously scheduled tasks. The *HEFT* starts by setting the computation costs of tasks and communication costs of edges to their mean values. Each task is assigned a value called upward rank. In this algorithm, the upward rank of a task is the largest sum of mean computation costs and mean communication costs along any directed path from this task to an exit task.

Our task placement is a flow from top to down, so it is a clustering problem in essentially. For such a problem solving, heuristic algorithm should be selected and used in priority, because virtualization can shield the heterogeneity of processors. The complex algorithms in heterogeneous distributed systems, such as *DLS*, *HEFT* and so on, should be excluded. In three static task scheduling algorithms, heuristic algorithms require direct information about the application and processors to carry out scheduling. Despite the greedy nature, heuristic-based approaches are not likely to produce consistent results on a wide range of problems. The heuristic-based scheduling algorithms are always efficient since they narrow the search down to a very small portion of the solution space by means of greedy heuristics. The heuristic-based scheduling algorithms are always efficient since they narrow the search down to a very small portion of the solution space by means of greedy heuristics.

The goals of task scheduling based on virtual machines are the shortest completion time based on users' preferences. Our study is different from present task scheduling algorithms in following aspects:

Ⅰ. Our algorithm includes two parts. One of them is to make a scheme to plan the task placement and scheduling from the view of whole system. The other is to perform the scheduling based on the scheme, in which, current task is selected to decide the next tasks.

Ⅱ. The tasks are combined by an optimal rule to form some suitable quantity of tasks in different layers from view of systematic clustering, so it is different from present static scheduling algorithms.

Ⅲ. For collaborative subtasks with complicate interactive time-relationship, more other factors affecting task placement and scheduling should be considered, such as communication time, switching time, and synchronization time, and so on.

Ⅳ. Some task scheduling algorithms just consider two layers including tasks and resources. The task scheduling based on virtual machines is divided into four layers including tasks, threads, virtual machines, physical machines in virtualized platform. Hierarchical aggregation

would transfer the tasks from top to underlying layer and let the tasks combine closely with each other.

Ⅴ. Because of the data dependence among tasks, the algorithm requires to obtain the output data from frontier tasks and input them into latter tasks.

4. The construction of model

We describe the tasks in virtualized environment via a new descriptive methodology named feature model *(FM)*, which lays out a solid foundation to develop the parallel applications. Combined with *FM*, the task placement and scheduling are performed with a high effective way. The model of task placement and scheduling takes *FM* to forecast the resource utilization, application throughput, application execution latency, and starvation. Thus,we need to speculate the performance and estimate the running time of applications considering the personnel preference in applications development and deployment. This model can help designers to make a good plan for task placement and scheduling in practice.

4.1 The description of tasks

Task placement and scheduling refer to the tasks decomposition and functions aggregation [61]. In order to map the subtasks to different layers of virtual computing systems, we analyze the features of subtasks in tasks decomposition from the view of parallel computing, and then, take the hierarchical method to get their modest subtasks in their granularities from coarse to fine. Each of the subtasks can complete a certain functions (called function of feature in this paper).The relationships among units are further identified to create a feature model.

Description 1 (*unit*): Unit, a subtask implementing the function of feature, is a tuple with 6 elements *unit=(Feature, CFs, IDFs ,ODFs,SCs,time)*.Where, *Feature* is the function of feature, *CFs* is the set of control flows, *IDFs* is the set of input data flows, *ODFs* is the set of output data flows, *SCs* is the set of outer interfaces, and *time* is the time of computation. A control flow is a tuple with 2 elements *CF=(dataitems,type),*of which, *dataitems* is the set of data items,and *type* is the type of this control flow with the values *{sequence, ifoption,switch,cycle}*. A data flow is a tuple with 3 elements *DF=(dataitems, fromFeature, tofeature),* of which, *dataitems* is the set of data items, *fromFeature* is the function of feature providing this data flow, and *tofeature* is the function of feature requesting this data flow.

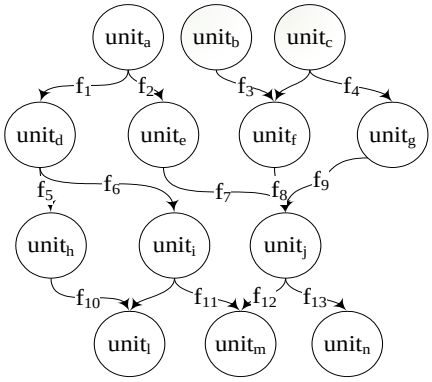

Fig. 1. Feature model

Description 2 *(FM)*: Feature model, a conceptual model of tasks, composes of units, control

flows and data flows. **Fig. 1**, as an instance of a feature model, is a tuple with 3 elements *FM=(units, CFs, DFs),* of which, *units* is the set of units,*CFs* is the set of control flows, and *DFs* is the set of data flows. In order to make the feature of tasks match the architectures, *FM* should be deployed into the layers of computing systems. *FM* is transferred as *{Threads,VMs,PMs}* though effective aggregation among layers. Where, *Threads* is the set of threads that units assign,*VMs* is the set of virtual machines that threads map to, and *PMs* is the set of physical machines that *VMs* are scheduled to.

Description 3 (*TPSL*): Task placement and scheduling layers are a tuple with 3 elements: *TPSL=(FM,layers,Mappings)*, of which, *FM* is the feature model of a task, and *layers* is the set of layers in virtual computing systems. We let *layers= (thread,VM,PM)*， where*, thread* is the layer of thread, *VM* is the layer of virtual machine, and *PM* is the layer of physical machine. We let *Mappings=(MT, MV, MP)*.Where, *MT* is the placement from units to threads: *unit→thread*, *MV* is the mapping from threads to virtual machines: *thread→VM*, and *MP* is the scheduling from virtual machines to physical machines:*VM→PM*.

4.2 The model of task placement and scheduling

For first problem presented in section 2, we determine the method for computing running time combined with the description of tasks based on formula (1).

Definition 1 (*SysnTime*): synchronization time, a deviation between the maximum computation time and the minimum computation time in parallel subtasks, can be divided into the time inside threads: $SysnTime_1$, the time among threads:$SysnTime_2$, the time among virtual machines: $SysnTime_3$, and the time among physical machines: $SysnTime_4$.

$$SysnTime_i = Psysn_i.\sum_{k=0}^{l}[\max(task_k.time) - \min(task_k.time)] \tag{3}$$

Where, $task_k$ is the one of *unit,thread,VM,PM* description, $max(task_k.time)$ is the maximum computation time in parallel subtasks of systems, $min(task_k.time)$ is the minimum computation time in parallel subtasks of systems, l is the quantity of parallel subtasks, and $Psysn_i$ is the coefficient of synchronization time.

Definition 2 (*ComTime*): communication time, the sum of time in tasks communication, can be divided into the time inside threads: $ComTime_1$, the time among threads:$ComTime_2$,the time among virtual machines:$ComTime_3$, and the time among physical machines:$ComTime_4$.

$$ComTime_i = Pcom_i.\sum_{k=0}^{n}(task_k.ESs.length) \tag{4}$$

Where, $Pcom_i$ is the coefficient of communication time.

Definition 3 (*SwitchTime*): switching time, the sum of switching time that underlying layer schedules the top layers, can be divided into the time inside threads:$SwitchTime_1$, the time among threads: $SwitchTime_2$,the time among virtual machines:$SwitchTime_3$, and the time among physical machines:$SwitchTime_4$.

$$SwitchTime_i = \frac{Pswitch_i.(tasks.length)}{Q_1Q_2Q_3} \tag{5}$$

Where, $Pswitch_i$ is the coefficient of switching time related to the architecture of computers,Q_1 is the quantity of physical machines,Q_2 is the quantity of *CPUs* in a physical machines, and Q_3 is the quantity of cores in a *CPU*.

Definition 4 (*I*): parallelism, the ratio of computation time between parallel computing and non-parallel computing in tasks, can be divided into the parallelism inside threads: I_1(I_1 is 1), the parallelism among threads: I_2, the parallelism among virtual machines: I_3, and the

parallelism among physical machine: I_4. These parallelisms not only can affect the running time from synchronization, communication and switching, but also decide the matching degree between tasks and their architectures. The parallelism I_i is expressed as:

$$I_i = \frac{\sum_{j=0}^{m} task'_j.time}{\sum_{j=0}^{n} task_j.time} \tag{6}$$

Where, $task'_j$ is the parallel part of tasks in $task_j$, n is the quantity of tasks, and m is the quantity of parallel tasks.

Definition 5 (*MD*): the matching degree between tasks and their architectures reflects the overall performance in layers aggregation. In our study, the ratio between the computation time of pure tasks and running time of computation is taken as the matching degree:

$$MD = \frac{(1-V).\sum_{j=0}^{n} unit_j}{(1-V).\sum_{j=0}^{n} unit_j + \sum_{i=1}^{4}(SwitchTime_i + ComTime_i + SysnTime_i)} \tag{7}$$

Where, $ComputationTime = (1-V).\sum unit_j$, and V is the satisfaction that Q_1,Q_2 and Q_3 take to the quantity of threads caused by feature model aggregation, $V \in [0,I_2]$.In our study, let V be

$$V = \begin{cases} I_2 & Q_1Q_2Q_3 > threads.length \\ I_2.\dfrac{Q_1Q_2Q_3}{threads.length} & Q_1Q_2Q_3 \le threads.length \,\&\, Q_1Q_2Q_3 \ne 1 \\ 0 & other s \end{cases}$$

For second problem presented in section 2, we determine the utility value of membership function based on formula (2).

Definition 6 (f_1): the development membership for applications designers refers to the personal preference and the acceptance for parallelism. We use the parallelism to measure the membership. Generally speaking, the higher the parallelism is, the more the subtasks are, thus, the smaller f_1 is. The exponential function is taken to describe this membership, so $f_1 = e^{-K_1 I_i}$, of which, K_1 is the personal preference coefficient in development.

Definition 7 (f_2): the deployment membership for applications designers refers to the personal preference and the acceptance for parallelism. We also use the parallelism again to measure the membership. The characteristics of f_2 are the same as f_1, so the exponential function is taken to describe this membership $f_2 = e^{-k_2 I_i}$, of which, K_2 is the personal preference coefficient in deployment.

On the basis of above definition, we create a model for task placement and scheduling based on virtual machines, which can be expressed as follows:

$$\begin{aligned} &\min Time = ComputationTime + \sum_{i=1}^{4}(SwitchTime_i + ComTime_i + SysnTime_i) \\ &\sum_{i=1}^{4}[f_1(I_i) + f_2(I_i)] \ge f_0 \quad 0 < I_i \le 1, \forall i \in (1,2,3,4) \end{aligned} \tag{8}$$

We assure that this model should first satisfy the personal preference f_0, and then reduce the overall time from all layers as much as possible to obtain the maximum matching degree

between the feature of tasks and the architectures of computing systems. Our methodology for task placement and scheduling should decrease overall time, improve resource utilization and maintain loads balance with accuracy and effectiveness.

5. The algorithms design

5.1 The workflow of task placement and scheduling

Task placement and scheduling in virtual computing systems are a tow-down workflow and the steps of them are as follows:

(1) The feature model is perfected and the data chains in serial computing are mined.

(2) The placement of units to threads, the mapping from threads to virtual machines and the scheduling of virtual machines in physical machines are finished. Thus, we obtain the hierarchical of task placement and scheduling in **Fig. 2**. Firstly, the relationships among units in feature model are determined and these units are gathered into some *Threads* (*Layer*$_1$ in **Fig. 2**) to finish task placement. Tasks among *Threads* communicate with each other by using shared memory. Secondly, *Threads* are gathered into several *VMs* and we run a *middleware* in each *VM* to coordinate these *Threads*. Tasks among *VMs* communicate with each other by using authorized page (*Layer*$_2$ in **Fig. 2**).Thirdly, *VMs* are gathered into several *PM*s and *VMM* runs in each host to coordinate their *VMs*. Tasks among *PMs* communicate with each other by using network protocols (*Layer*$_3$ in **Fig. 2**). And finally, *PMs* are connected into a *VirtualCluster*(*Layer*$_4$ in **Fig. 2**).

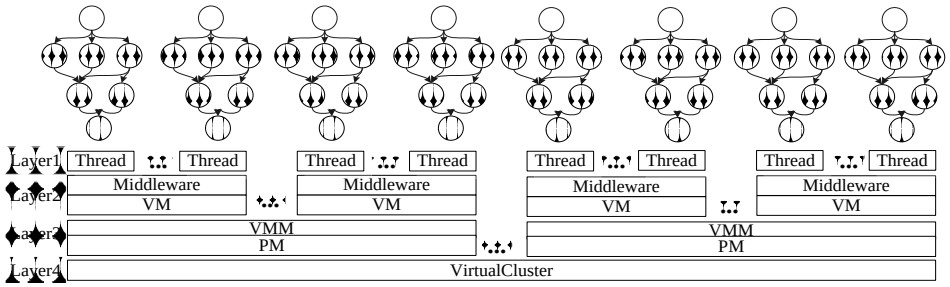

Fig. 2. Four layers in task placement and scheduling based on virtual machines

(3) The layers are connected. *Middleware*, connecting threads and *VM*, does not only complete the scheduling of threads, but also complete the communication and synchronization of tasks in external *VMs* [62]. *VMM,* connecting *VM* and *PM*, does not only complete the scheduling of *VMs*, but also complete the communication and synchronization of the tasks in external *PMs*. Network protocols, connecting all *PMs*, is responsible for the communication and synchronization with the tasks in this cluster.

5.2 The algorithms for task placement and scheduling

We design a task scheduling alogrithm (*TSA)* in this section, and *TSA* composes of two subalgorithms: *VPVM* and *TEVM*. According to the steps in the workflow of task placement and scheduling, the tasks are gathered into four layers via *TPVM* firstly. We require to fully consider the *Gathered Degree* of these layers in *TPVM*.

Definition 8 (*Gathered Degree*): *Gathered Degree*, a clustering criterion, gathers the tasks in top layers of virtual computing systems into the tasks in low layers. *Gathered Degree* can be measured by quantitative data, and its field belongs to *(0,1)*. Based on the layers of task

placement and scheduling, *Gathered Degree* can be divided into *Threads Gathered Degree: π*, *VM Gathered Degree ξ*, and *PM Gathered Degree ς*.

According to the model shown in formula (8), we set the *Gathered Degree π,ξ,ς* to aggregate the tasks into different layers, named as the scheme of task placement. We call the scheme with the maximum matching degree between the feature of tasks and the architectures as the optimal scheme. We first assign the subtasks in *FM.Units* to threads, in which, the initial *Threads Gathered Degree* π_0 should enable the system produce moderate number of threads. When we map the threads into virtual machines, the initial *VM Gathered Degree* ξ_0 assures that the quantity of virtual machines should be greater than that of physical machines. When we schedule the virtual machines into physical machines, the initial *PM Gathered Degree* ς_0 assures that the quantity of physical machines should be equal to or less than that of hosts in real cluster, and we determine the deployment of virtual machines in physical machines in final. A heuristic algorithm to solve the optimal scheme for task placement based on virtual machines *(TPVM)* is presented in following pseudo-code:

Algorithm1:***TPVM***
Input: FM; Q_1,Q_2,Q_3; f_0 ,K_1 ,K_2; $Psysn_{1\sim4}$,$Pcom_{1\sim4}$,$Pswitch_{1\sim4}$ **//**feature model, configuration of hosts, coefficient of designer's preference, coefficient of switching time
Output: threads, VMs, PMs; // the description of tasks in all layers

1 Determine the initial Gathered Degree π_0, ξ_0, ς_0 and $step_0$ using FM, Q_1, Q_2, Q_3;
2 minTime←0; step←$step_0$;
3 π←π_0; **While**(π>0){
4 Thread threads[];**foreach** (unit in FM.units) {compute_gather_unit (threads, unit, π);}
5 ξ←ξ_0; **While**(ξ>0){
6 VM VMs[];**foreach** (thread in threads) {compute_gather_thread (VMs, thread, ξ);}
7 ς←ς_0; **while**(ς>0){
8 PM PMs[];**foreach** (VM in VMs) {compute_gather_VM (PMs, VM, ς);}
9 Compute I_1, I_2, I_3, I_4 using threads, VMs, PMs by formula (6);
10 Compute f_1,f_2 by K_1 ,K_2
11 **if** ($\sum_{i=1}^{4}[f_1(I_i)+f_2(I_i)] \geq f_0$) {
12 Compute ComputeTime using FM,threads, Q_1, Q_2, Q_3;
13 Compute $SysnTime_{1,2,3,4}$ using threads, VMs, PMs , $Psysn_{1\sim4}$,by formula (3);
14 Compute $ComTime_{1,2,3,4}$ using threads, VMs, PMs, ,$Pcom_{1\sim4}$ by formula (4);
15 Compute $SwitchTime_{1,2,3,4}$ using threads, VMs, PMs, ,$Pswitch_{1\sim4}$ by formula (5) ;
16 Compute $Time = ComputationTime + \sum_{i=1}^{4}(SwitchTime_i + ComTime_i + SysnTime_i)$
17 **if**(CompareLower (minTime,Time)>0) ; { minTime←Time;
18 $MD = (1-V).\sum_{j=0}^{n} unit_j \Big/ (1-V).\sum_{j=0}^{n} unit_j + \sum_{i=1}^{4}(SwitchTime_i + ComTime_i + SysnTime_i)$;
19 SaveMinTimeData(threads,VMs,PMs, π,ξ, ς, minTime,MD); }
20 } π-←step ;} ξ-←step; } ς -←step;}
21 getMinTimeData (threads,VMs,PMs, π,ξ, ς, minTime, MD);
22 check whether minTime, MD can satisfy requirements;
23 **if**(minTime, MD satisfy requirement){
24 connect FM,threads,VMs,PMs; return threads,VMs,PMs ;}
25 **Else** { Determine initial Gathered Degree π_0, ξ_0, ς_0 and $step_0$ using π,ξ, ς,step; **goto** ;}
26 **End**.

Fig. 3. The description of *TPVM*

In above *TPVM* algorithm, the function of *compute_gather_unit* is to gather a *unit* in *FM* to a dynamic array *threads*, and then compute the *Gathered Degree* π between *unit* and *threads*. If π is greater than its criterion π_0, then, this *unit* is merged into current element in *threads*. Otherwise, a new element will be created and added into *threads,* and *unit* is merged into this new element, then, this new element is set as current element in next round. The functions of *compute_gather_thread* and *compute_gather_VM* are the same as *compute_gather_unit. TPVM* changes the initial *Gathered Degree* π_0, ξ_0, ς_0 according to $step_0$, and in different π, ξ, ς, personal preference f is computed. If the constraint can be satisfied via the judgement of f, the running time and matching degree are determined to compare with the value in the last round, and then, obtain the optimal solution of them in $step_0$. If the optimal solution in $step_0$ can not satisfy the requirement of designer, the initial *Gathered Degree*π_0, ξ_0, ς_0 and $step_0$ should be changed until the designer finds out the most satisfactory solution. The function of *CompareLower* is to compare the values of two time, the function of *SaveMinTimeData* is to save the computing results, and the function of *getMinTimeData* is to get the minimum time from computing results. We can obtain the relative optimal solution via π, ξ, ς by changing their *step* with progressive iterations,so the algorithm *TPVM* can be convergence to find out the optimal scheme of task placement in accordance with the model in a sense.

Let n be the quantity of units, m be the quantity of threads, l be the quantity of virtual machines, and k be quantity of physical machines. We suppose *TPVM* can complete in a group of determined π_0,ξ_0,ς_0 and $step_0$. In order to calculate the complexity of *TPVM* in this situation,we firstly determine the complexity of rows 3-20. Because the steps'number of *compute_gather_unit (threads, unit,* π*)* is m,the steps'number of *compute_gather_thread (VMs, thread,* ξ*)* is l,the steps'number of *compute_gather_VM (PMs, VM,* ς*)* is k, and the steps'number in rows 17-19 is $(\pi_0/step_0)$ $(\xi_0/step_0)(\varsigma_0/step_0)$,then,the steps'number in rows 7-19 is $((\pi_0/step_0)(\xi_0/step_0)(\varsigma_0/step_0)+k).l.\varsigma_0/step_0$,the steps'number in rows 5-19 is $[((\pi_0/step_0)(\xi_0/step_0)(\varsigma_0/step_0)+k).l.\varsigma_0/step_0+l].m.\xi_0/step_0$, and the steps'number in rows 3-20 is $\{[((\pi_0/step)(\xi_0/step)(\varsigma_0/step)+k).l.\varsigma_0/step+l].m.\xi_0/step+m\}.n.\pi_0/step$. Furthermore, we determine the steps of *getMinTimeData (threads,VMs,PMs,* π*,*ξ*,* ς*, minTime, MD)* in row 21 as $(\pi_0/step)$ $(\xi_0/step)(\varsigma_0/step)$,and the steps'number in other rows can be thought as $O(1)$. Then, the steps'number in rows 1-26 is:

$$\frac{\pi_0^{\,2}\xi_0^{\,2}\varsigma_0^{\,2}l.m.n}{step^6}+\frac{(klmn+1)\pi_0\xi_0\varsigma_0}{step^3}+\frac{lmn\xi_0\pi_0}{step^2}+\frac{mn\pi_0}{step}\le C.\frac{\pi_0^{\,2}\xi_0^{\,2}\varsigma_0^{\,2}lmn}{step^6}$$

As a result, the complexity of *TPVM* is $O(\frac{\pi_0^{\,2}\xi_0^{\,2}\varsigma_0^{\,2}lmn}{step^6})$.

Property 1 (The iterative property of *TPVM*): For any value of *Time*，if both sides of its minimum value exist the iterations, then *TPVM* would be monotonic in both sides of the minimum *Time*, and while its corresponding maximum *MD* would also be monotonic in its both sides. We explain this property as follows: Based on the expression of *Time*, after we change π, ξ, ς with progressive *step* and complete the aggregation of layers by three functions: *compute_gather_unit,compute_gather_thread, compute_gather_VM* ,the sum of time from $SysnTime_{1,2,3,4}$, $ComTime_{1,2,3,4}$, $SwitchTime_{1,2,3,4}$ would also change with the progressive iterations.This property enables the *Time* present monotonically decreasing in its left of the minimum value and monotonically increasing in the right of its minimum value. Based on the expression of *MD,* we see that *MD* has the opposite monotonic property as *Time*.

TPVM, as a problem solving algorithm for task placement scheme, lays out a solid basis for tasks execution based on virtual machine. When *TPVM* is performed in a virtualized platform, the output of *TPVM* are taken as the input of tasks execution. The return values from *TPVM*

are three arrays: *threads[],VMs[], PMs[]*.The elements of them are*: thread=(units[],VM),VM=(threads[],PM),PMs=(VMs[]),*which represent the clustering results of tasks in four layers. Based on *FM, threads[],VMs[], PMs[]*, the tasks execution algorithm based on virtual machines*(TEVM)* is shown in algorithms 2.

Algorithm 2：*TEVM*
Input: *FM*, *threads[],VMs[], PMs[]*;
Output: *running result*

```
1   Determine the begin_unit and end_unit from FM
2   Let task= begin_unit;
3   Let return_List be an array to store the return data of tasks.
4   Execute(task);
5   Function Execute(task){
6    If (task is not end_unit++ ){
7      input_data=obtainInputData(FM,task,return_List);
8      loadTask(task, input_data); // loadTask is defined as asynchronous function
9      Determine next_tasks[] in FM via task;
10     Foreach(next_task in next_tasks[]){ Execute(next_task); }
11   Else{Find the return_data of task from return_List as running result; }}
12  Function obtainInputData (FM,task, return_List){
13      Let isAllReceived=false;
14      Extract data_items that task require to take as input data in FM;
15      While(isAllReceived ==false){
16          isAllReceived =CheckDataValue(data_items);
17          If(isAllReceived==true){
18           Find input_data of task from return_List via data_items;return input_data }
19          } Else{ Sleep to wait;}}}
20  Function CheckDataValue(data_items){
21      Let count=0;
22      Foreach(data_item in data_items){
23        Find data_value of data_item from return_List;
24        If(data_value!=null){ count++;}}
25      If(count== data_items.Length) return true; else return false;}
26  Function loadTask(task, input_data){
27      Determine the thread,VM,PM that begin_unit from threads[],VMs[],PMs[];
28      send message to VM to query the Status of VM ;
29      if(VM has not been created){ Create VM in PM;}
30      if(thread has not been created){ Create thread in VM;}
31      Load task in VM thread via input_data ;
32      receive return_data from task;
33      Add return_data into return_List;}
```

Fig. 4. The description of *TEVM*

TEVM algorithm requires finishing the data transmitting, task loading and task switching. *TEVM* composes of four functions, which find out the initial task named as *begin_unit* and ending task named as *end_unit* from *FM* first, and then define the middle variables task and *return_List*. Then, the algorithm begins with *begin_unit* and performs the scheduling with the function *Execute()* until *end_unit* finishes its computation. The algorithm tidies the data in *return_List* as *running result*. In row 5-11, we define *Execute()* as a recursive calling. The *task,* as the current unit, obtains the input data of tasks via the function *obtainInputData*() and loads the task via the function *loadTask()*, and then finds out the set of tasks *next_tasks* extended by

the time-relationship of them. In row 12-19, *obtainInputData*(), as the function to obtain input data of task , judges whether all input data *data_items* exist in *return_List* via the sub function *CheckDataValue()*. If the values in *data_items* are existed, the *input_data* is constructed and return back. Otherwise, the system sleeps for a short time until the data arrive. In rows 20-25, the function *CheckDataValue*() is defined. It traverses the values in *data_items* to judge whether the values in *return_List* are null for corresponding items. The values for items in *return_List* are constructed by the return values from frontier tasks determined by time-relationship. In rows 26-33,*loadTask*() has implemented the loading of *task.* It first determines the *thread,VM* and *PM* that the *task* exists.In this function, the message is sent out to *VM* to query the status of *VM.* And if *VM* has not been created in *PM,* the *VM* would be created. In the same way, the *thread* is created in *VM,* and then the *task* is loaded into the *thread* to execute. The return values of *task* are obtained and added into *return_List*.

Let the average quantity of data flows for each unit be *p*. We require to compute the complexity of four functions to calculate the complexity of *TEVM* in this situation. The complexity of recursive function *Execute*(*task)* be $nlog_2n$. Row 23 do transverse the *return_List,* so this row's steps' number is *p.n*, and the the steps of function *CheckDataValue*(*data_items*) is *p. p.n*. The steps' number of *obtainInputData*() is equal to that of *CheckDataValue*().Because the complexity of function loadTask is *O(m.l.k)* and the steps' number of row 9 is *n*, we get the complexity of *TEVM* as : $(\mathrm{p}^2\mathrm{n}+\mathrm{m.l.k}+\mathrm{n})\mathrm{n}log_2n=(\mathrm{p}^2+2)\mathrm{n}^2log_2n\le C.\mathrm{p}^2\mathrm{n}^2log_2n$.As a result, in the best case, the complexity of *TPVM* is $\mathrm{O}(\mathrm{p}^2\mathrm{n}^2log_2n)$.

6. Experiments analysis

CloudSim is used as the experimental platform to test *TSA*. *CloudSim*, developed by *The Gridbus Project at the University of Melbourne,* provides a generalized and extensible simulation framework that enables modeling, simulation, and experimentation of emerging virtualized infrastructures and application services, allowing users to focus on specific system design issues that they want to investigate, without getting concerned about the low level details related to cloud-based infrastructures and services.The *CloudSim* toolkit supports both system and behavior modeling of cloud system components, such as data centers, task scheduling, virtual machines and resource provisioning policies.

We install *CloudSim* toolkit in a *PC* host of one *Intel Xeon 3.40GHz* processor with two cores, *2GB RAM* and *160GB SCSI* hard disk. The processor brings a *16KB* cache and *1024KB* secondary cache. The operation system installed in this machine is *Windows Server 2003. CloudSim* toolkit requires a Java development Kit *(JDK).* We install *jdk1.6.0* in *Windows Server 2003* and set the environment path for *CloudSim2.1.1*, which is the edition we select for the experiments. Our experiments under *CloudSim* platform are divided into three experiments to achieve three goals: 1) to evaluate the effectiveness of time estimated method for tasks in feature model and lay out a basis to verify the correctness of task placement and scheduling algorithms. 2) to verify the characteristics of *TPVM* algorithm and solve the problem in the task placement and scheduling model. 3) to compare the performance of *VSA* algorithm with other similar algorithms.

6.1 The experiment 1

In order to verify the effectiveness of time estimated method, we extend *CloudSim*, and then make experiment 1 to evaluate the theoretical values with the values in practice.

(1) *CloudSim* toolkit extension

The best task scheduling algorithms implemented in present distributed systems are *DLS* and *HEFT*, which schedule the tasks via *DAG*. We create these two algorithms in the open-source of *CloudSim*. In the implementation of *DLS* and *HEFT* algorithms, a new class *Cloudet_task* extended from *Cloudlet* of *CloudSim* is desgined, and a variable *taskAmount* is added as the member of itself to describe the task's amount. Meanwhile, the setter and getter functions are created for *Cloudet_task*.

Most of the present static task scheduling algorithms implement the scheduling from tasks to processor, and not related to virtual machines. In order to solve this problem to adapt to the requirement of virtual machines scheduling, we make some special transfers in virtualized platform to implement these algorithms. We create a new class *VirtualMachine_CPU* extended from class *VirtualMachine*, and add the calling *setCpus(1)* into its constructor. In this approach, we create the same quantity of virtual machines as that of *CPUs* in host. When the tasks are scheduled to virtual machines, the occupation to a *CPU* by a virtual machine refers to a task scheduling to a *CPU*,

The class *DatacenterBroker* in *CloudSim* does the task scheduling to virtual machines. For a task, the function *bindCloudletToVM(int cloudletId,int vmId)* in *DatacenterBroker* would bind the task labeled as *cloudletId* into the virtual machine labeled as *vmId*. In order to implement the task scheduling to *CPUs*, a new function *bindCloudletToCPU(Cloudet_task[] cloudlet_tasks,VirtualMachine_CPU[] vm_cpus)* is designed to bind multiple tasks to corresponding multiple *CPUs*.

Besides to them, a new class *Statistics* with the function *execute ()* is designed to record the time expenditure of *DLS* and *HEFT* in *CloudSim*. Based on the task amount, *execute()* function aims to sum the switching time, communication time, synchronization time, computation time and other time, and finally the total time of them in scheduling algorithms.

(2)The input data of experiment

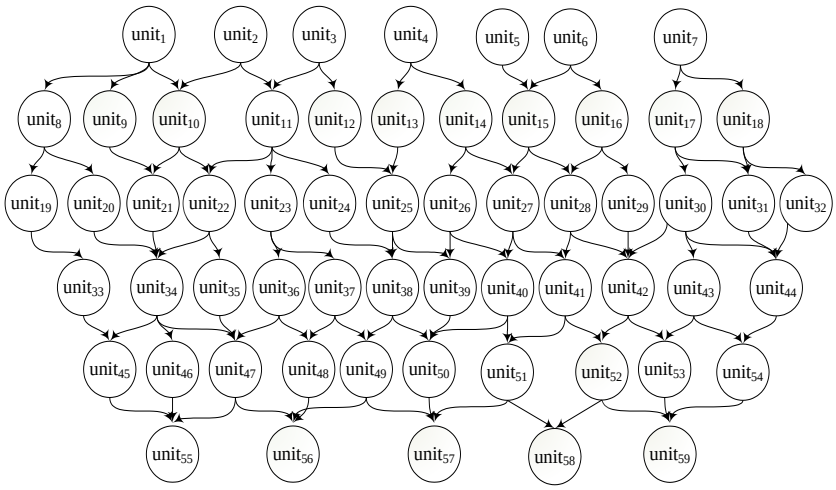

Fig. 5. Feature model of CFIE

Combined with description of tasks, we take a simulation of cold flow impulsive experiment for a car engine *(CFIE)*, the typical coupling process considering the affecting relations between flow field and structure, as an instance to analyze the *TPVM* algorithm. *CFIE*, a collaborative computing task, constitutes of *59* units after we determine its requirement combined with the descriptions of feature model, threads, virtual machines and physical machines. We set the numbers to them and determine their timing-relationships, and then the feature model of *CFIE* is shown in **Fig. 5** (Because of inconvenience in drawing the

figure, we just display the data flows and omit the control flows and the description of units).We determine the elements in feature model: *unit=(Feature, CFs, IDFs ,ODFs, SCs,time)* and create a *FM=(Units, controlflows, dataflows).* The time in each of unit in *FM.Units* is estimated as 15000~20000 millisecond .

The configuration data of this experiment: We use *2*, *4*, *6* and *8* hosts in this computing platform, and each machine owes a *CPU* with two cores. Because the quantity of virtual machines is equal to the quantity of *CPU*, the quantity of virtual machines is set as 2, *4*, *6* and *8*.

(3)The experimental results

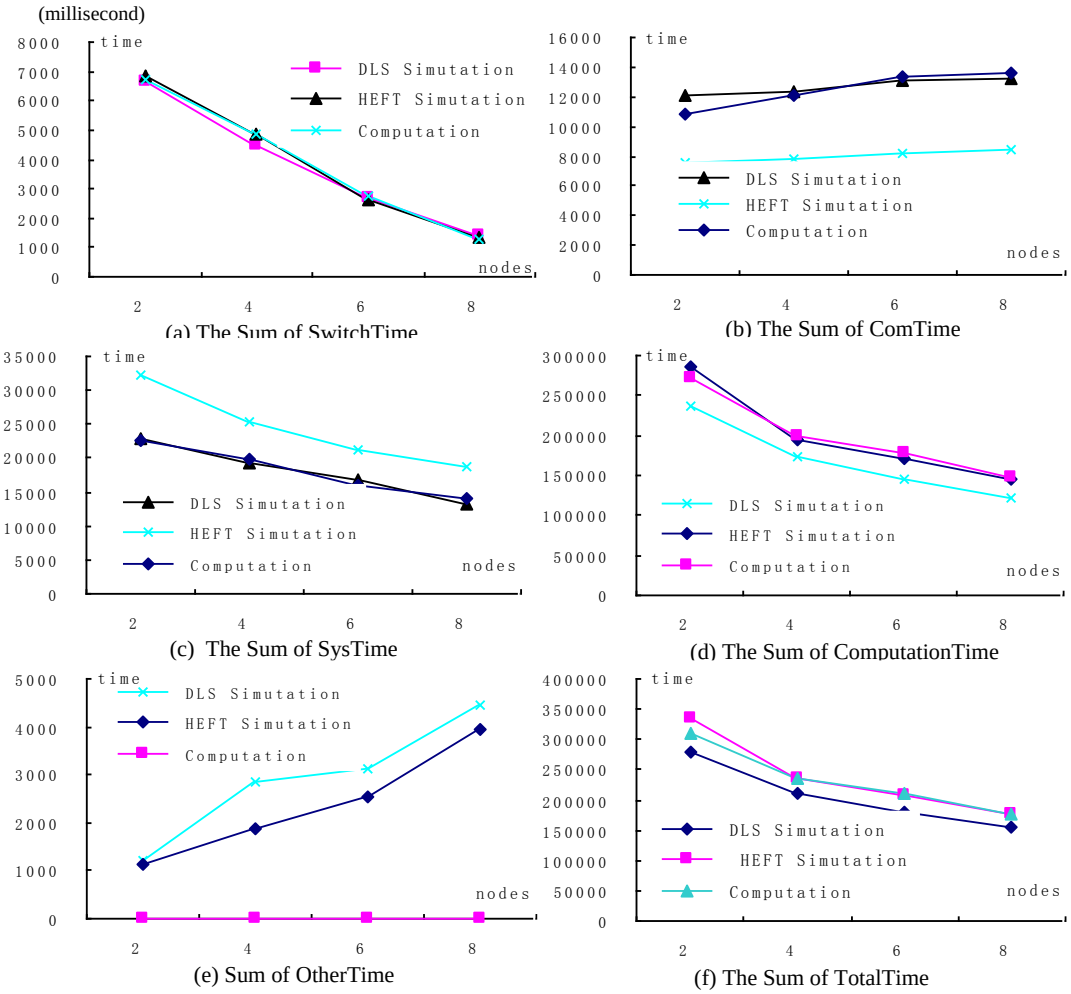

Fig. 6. The compasion of DLS,HEFT simulation time with computation time

We develop the simulation code in *CloudSim*, including the initiation of *Gridsim* library, the creation of data center, the construction of a *Broker*, the creation of virtual machines, the creation of tasks, the submission of tasks, task scheduling, starting the simulation, the data statistics, ending the simulation and outputting the results.

Based on the speed of *CPU* and the tasks' amount inputted into above *DLS* and *HEFT* algorithms, we estimate the computation time via related time estimated method in current research literature [66] and assign it into the *Unit.time*. Based on the computation time of tasks,

the formulas (3-5) in this paper are taken to compute the switching time, communication time and synchronization time.

Based on the steps described in above text, the experiment 1 is running for five times. The statistics results are taken to average their values. By comparing the experimental results of algorithms in *CloudSim* simulation and the results of our time estimated methods presented in this paper, the results are shown in **Fig. 6**, in which, the trend of the time in *DLS* algorithm, *HEFT* algorithm and our theoretical computation can be seen clearly. We show the sum of *SwitchTime*, the sum of *ComTime*, the sum of *SysTime*, the sum of *ComputationTime*, the Sum of *OtherTime* and the Sum of *TotalTime* in four layers in **Fig. 6** *(a)-(f)*.

It is shown from **Fig. 6** (a) that the switching time estimated by our method is almost overlapping with the switching time in *DLS* and *HEFT* algorithms. **Fig. 6** (b)and (c) show that the communication time and synchronization time estimated by our method close to the communication time and synchronization time in *DLS* algorithm. **Fig. 6** (d) shows that the tasks computation time estimated by our method closes to the tasks computation time in *HEFT* algorithm. Since our method does not compute other time, but besides such four kinds of time, there may exist other time in practice. **Fig. 6** (e) shows that the other time in *DLS* and *HEFT* algorithm experiments is not zero, but the values of them are small and could be neglected when comparing with the sum of total time in **Fig. 6** (f). It is seen from **Fig. 6** (f) that the estimated time in our study is in-between the sum of five kinds of statistics time in *DLS* and *HEFT* algorithms, which are not greater than 5% of them.

It is concluded from **Fig. 6** that the time sourcing from our estimated method and the time of *DLS* and *HEFT* algorithms have greater than 95% similarity. It has proved the effectiveness and the meaningfulness of our time estimated method. Therefore, the feature model presented in this study can use this estimated time as the input data to perform a static task placement and scheduling in *TPVM*. We further discuss the execution of *TPVM* and perform the experiments in *CloudSim* in next section.

6.2 The experiment 2

TPVM algorithm is developed in the source of *CloudSim* to extend the function of platform, thus, to verify the properties of *TPVM*.

(1) The *TPVM* algorithm implementation

In the implementations of *TPVM*, three new classes *Cloudet_unit*, *Cloudet_thread* and *Cloudet_FM* extended from *Cloudet* is designed, and the tuple of *Unit*, *Cloudet_unit[]*,and the tuple of *FM* are added into these classes as the members of themselves. At the same time, the setter and getter methods are designed to set and get the values of them. Then, a new class *Parameter* is designed with the members Q_1,Q_2,Q_3 ,f_0,K_1 ,K_2, $Psysn_{1\sim4}$, $Pcom_{1\sim4}$, $Pswitch_{1\sim4}$ to load the input values of *TPVM* algorithm, and the setter and getter methods are designed to set and get the values of them again.

The single task scheduling method *bindCloudletToVM (int cloudletId, int vmId)* requires changing as the multiple tasks scheduling method to implement the task scheduling to virtual machines, which is the same as the design of class *DatacenterBroker_CPU*. We design a new class *DatacenterBroker_FourLayers* extended from *DatacenterBroker*, whose function *bindCloudletToFourLayers(Cloudet_FM fm, Parameter para)* implements *TPVM* algorithm to solve the problem in the model of task placement and scheduling based on virtual machines.

(2) The input data of *TPVM*

In order to verify the convergence property and the rationality of *VPTA*, we continue to take the data of 59 subtasks in project *CEIE* as the testing data. Two steps in below section were made to verify the rationality of *TPVM*. Our experiments aim to determine the effect *that* physical configuration and personal preference have on the convergence of *TPVM*.

Ⅰ) The coefficient of personal preference f_0 is set as *0.6*, and the quantity of computing nodes in a cluster is set as *2, 4, 6, 8* respectively, of which, each node has a CPU with two cores.

Ⅱ) We have 4 computing nodes in a cluster and each node has a CPU with two cores. The coefficient of personal preference f_0 in applications development and deployment is set as *0.6, 0.5, 0.4, and 0.3* respectively.

Besides to those, K_1 and K_2 in such two steps of experiment are both set as *0.2*,$Psysn_{1\sim4}$, $Pcom_{1\sim4}$ and $Pswitch_{1\sim4}$ are all set as 1.

(3) The experimental results

When the input parameters of *TPVM* algorithm are fixed, (eg. $FM,K_1,K_2,Psysn_{1\sim4}$, $Pcom_{1\sim4}$,and $Pswitch_{1\sim4}$) , the relative optimal solution is necessity to depend on Q_1,Q_2,Q_3 and f_0. As a heuristic algorithm,the nature of *TPVM* aims to find a relative optimal scheme for task placement and scheduling through enough progressive iterations. For any group of the values in step Ⅰ and step Ⅱ , these iterations will begin with π_0, ξ_0, ς_0, and end to solve *Time* and *MD* satisfying constraint in accordance with $step_0$. In those iterations, based on the *property* of *TPVM*, we compare their computing results with pairwise comparison, thus the minimum *Time* and the maximum *MD* are inevitable to check out.

Step Ⅰ analyzes the trend of *Time* and *MD* with the iterations under the condition of different physical configurations, and the laws of them are shown in **Fig. 7** and **Fig. 8**. We can see from the result that *TPVM* could find out the optimal solution for any group of physical configuration and the optimal solution is in the lowest point of any of such four *Time* curves or the highest point of any of such four *MD* curves, which are consistent with the iterative nature of *TPVM*. The curves in **Fig. 7** have verified that *Time* present monotonic in both sides of the minimum *Time*, and **Fig. 8** has verified that *MD* present monotonic in both sides of the maximum *MD*. Different physical configurations would lead to different iterations required to find out their optimal solutions. We set the quantities of computing nodes are *2, 4, 6* and *8* respectively, thus, when the iterations of them reach to *2101, 1609, 1410* and *1202*, we get their relative solutions. The more the quantity of computing nodes are, the less are the iterations for *TPVM* to come to their lowest running time and highest matching degree, because strong hardware configuration would make the aggregation of four layers easier. It is seen from **Fig. 7** that more quantity of computing nodes in a cluster would produce a lower value in the minimum *Time*, because strong physical configuration would lead to a less time in synchronization, communication and switching in four layers, so the running time of 8 computing nodes in a cluster are less than that of *2, 4* and *6* computing nodes in a cluster. It is shown from **Fig. 8** that physical configuration almost has no effect on the maximum *MD:* different physical configurations would produce the same maximum *MD* inbetween *0.7~08.* The expression of *MD* demands *TPVM* obtain the optimal scheme for task placement and scheduling in any physical configuration environment, so the same maximum *MDs* in different parameters shows that we have gotten the optimal scheme. It is also seen from this experiment that the deviation of different matching degrees is less than that of running time in different physical configurations. The *MD* curves in **Fig. 8** compact with each other inbetween *0.28~0.77*, but the *Time* curves in **Fig. 7** are dispersed very much. It is concluded that the matching degree does not depend on the physical configurations, but the physical configuration has a great effect on the running time of virtual computing systems.

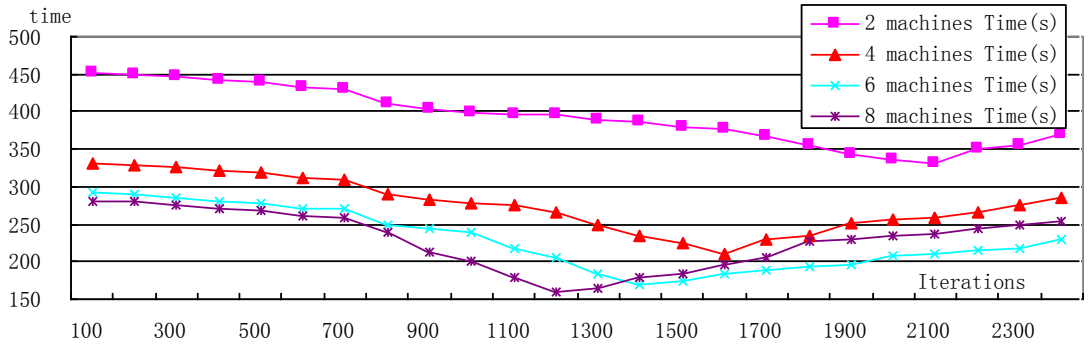

Fig. 7. The trend of *Time* in different physical configurations (millisecond)

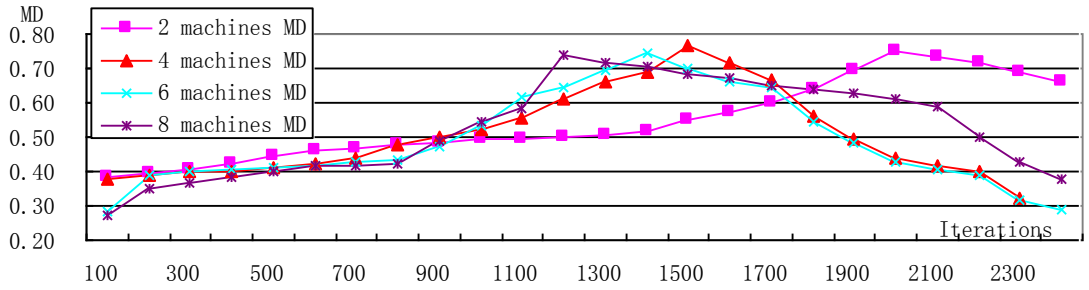

Fig. 8. The trend of *MD* in different physical configurations

Step Ⅱ analyzes the trend of *Time* and *MD* with the iterations under the condition of different personal preferences, and the laws of them are shown in **Fig. 9** and **Fig. 10**. We can see from the result that *TPVM* could find out the optimal solution for any group of personal preference and the optimal solution is in the lowest point of any of such four *Time* curves or in the highest point of any of such four *MD* curves, which are consistent with the iterative nature of *TPVM*. The curves in **Fig. 9** have also verified that *Time* present monotonic in both sides of the minimum *Time*, and **Fig. 10** has also verified that *MD* present monotonic in both sides of the maximum *MD*. Different personal preferences would lead to different iterations required to find out their optimal solutions. We set f_0 as *0.6, 0.5, 0.4,* and *0.3* respectively, thus, when the iterations of them reach to *1202, 1398, 1610* and *2108*, we get their relative solutions. The bigger f_0 is, the less the iterations are for *TPVM* to come to their lowest running time and highest matching degree, because bigger personal preference coefficient would make the aggregation of our layers easier. It is seen from **Fig. 9** that a smaller f_0 for designers would enable a higher value of the minimum *Time*, because the higher parallelism of tasks would produce a smaller *ComputationTime* in *Time*. It is also shown from **Fig. 10** that personal preference almost has no effect on the maximum *MD:* different personal preferences would produce the same maximum *MD* inbetween *0.7~0.8*. This conclusion is similar to the result of experiment 1. It is also seen from this experiment that the deviation of matching degrees is less than that of running time in different personal preferences. The *MD* curves in **Fig. 10** compact with each other inbetween *0.25~0.74*, but the *Time* curves in **Fig. 10** are dispersed very much. It is concluded that the matching degree does not depend on the personal preferences, but the personal preference has a great effect on the running time of virtual computing systems.

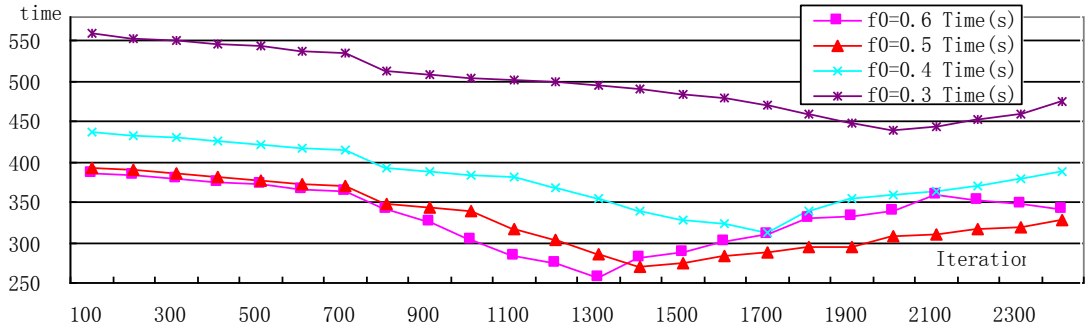

Fig. 9. The trend of *Time* in different personal preferences (millisecond)

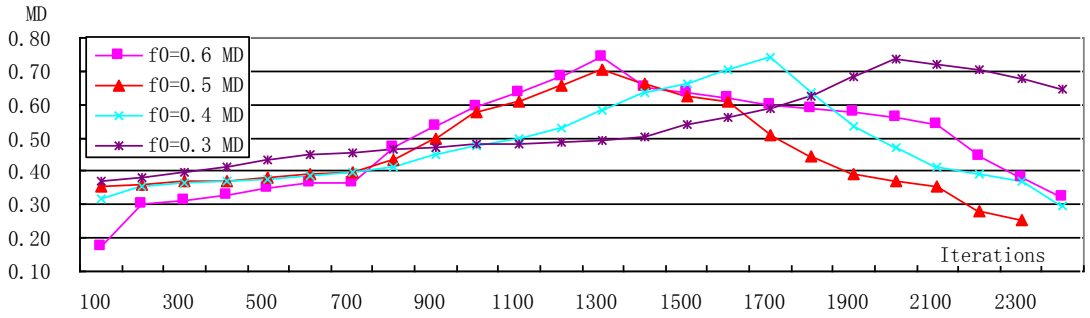

Fig. 10. The trend of *MD* in different personal preferences

It is concluded from above experiment that *TPVM* algorithm can find out a group of π,ξ,ς to promote the value of *Time* come to their minimum value and *MD* come to their maximum value for any personal preference and physical configuration, so the *TPVM* algorithm is feasible to implement task placement and scheduling. Based on the results of such two experiments, we set the personal preference coefficient f_0 as 0.4 in *CFIE*, then, four hosts are used to develop and deploy this application(each node has a CPU with two cores). We run our program of algorithm *TPVM* to get the optimal scheme of task placement and scheduling shown in **Fig. 11**. It is seen from **Fig. 11** that 59 units are assigned to 14 threads, of which, *thread1* and *thread2* are mapped into *VM1*, *thread3* and *thread4* are mapped into *VM2*, *thread5* and *thread6* are mapped into *VM3*, *thread7* and *thread8* are mapped into *VM4*, *thread9* and *thread10* are mapped into *VM5*, *thread11* and *thread12* are mapped into *VM6*, and *thread13* and *thread14* are mapped into *VM7*.Then, *VM1* and *VM2* are scheduled into *PM1*, *VM3* and *VM4* are scheduled into *PM2*, *VM5* and *VM6* are scheduled into *PM3*,and *VM7* is scheduled into *PM4*.

Based on the layers of task placement and scheduling shown in Fig.11, we have developed the application of *CFIE* by implementing parallel algorithm. The application would be taken to deploy according to the workflow in Fig.11. It is shown that *TPVM* can improve the matching degree between tasks and architectures to raise the peak performances of clusters and satisfy the requirement of designers in applications development and deployment. In this section , experiment 2 has verified the effectiveness of our heuristic task placement and scheduling alogorithm.

6.3 The experiment 3

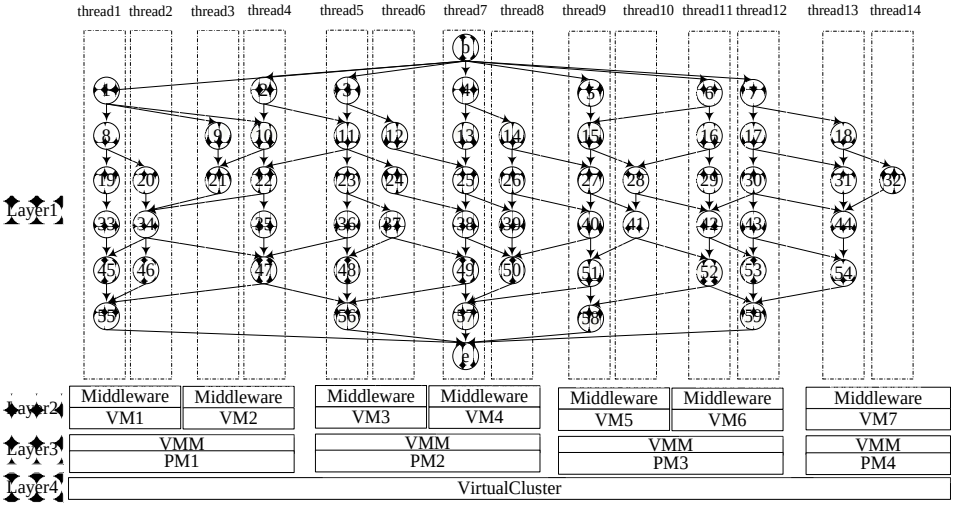

Fig. 11. CFIE task placement and scheduling based on virtual machines in four layers

In order to compare the *TSA* algorithm with the most popular and best task scheduling algorithms: *DLS*, *HEFT*, *CPOP* and *LMT*, we implement them in this section.

(1) The algorithms implementation

The implementation of *CPOP* and *LMT* is the same as *DLS* and *HEFT*. The implementation of *TSA* is as follows: On the basis of designing the new classes *Cloudet_unit*,*Cloudet_thread*,*Cloudet_FM* and *DatacenterBroker_FourLayers*, *Stastics* in above section, according to results of *TPVM*, an new function *executeCloudletInFourLayers (Cloudet_FM fm, Threads, VirtualMachineList VMs, host[] PMs)* is added into the class *DatacenterBroker_FourLayers* to implement the *TEVM*. Thus, we construct *TSA* in *CloudSim.*

(2) The input data of algorithms

We extend the *CFIE* project to make the experiment, and the quantity of subtasks is set as *70*, *82*, *90*, and *105*. We suppose that the system composes of enough hosts that the experiment requires, and each host has a *CPU* with two cores. In *TPVM* algorithm, the time estimated values in *FM* has been inputted into *TPVM*. As *TPVM* could optimize the task placement, we could find out the optimal *threads'* quantity, optimal *VMs'* quantity, and optimal *PMs'* quantity. In order to ensure the comparability between *TSA* and *DLS*, *HEFT*, *CPOP* and *LMT* in virtualized data center, the four algorithms should use the same hosts as that *TSA* algorithm produces, and ensure the same quantity of *CPUs*. *DLS*, *HEFT*, *CPOP* and *LMT* algorithms take *DAG* as the input data.

(3) The experiment result

We make the simulation code and run the program. After inputting the tasks with *59*, *70*, *82*, *90*, and *105* subtasks respectively into *TSA* algorithm, the best quantity of physical machines determined by *TPVM* for task scheduling are *4*, *6*, *7*, *8* and *10*. *DLS*, *HEFT*, *CPOP* and *LMT* algorithms also adopt 4, 6, 7, 8 and 10 physical machines respectively. We run the algorithms for five times and average their values. The comparison of five algorithms in performance is shown in **Fig. 12**. We list the sum of *SwitchTime*, the Sum of *ComTime*, the sum of *SysTime*, the Sum of *ComputationTime*, the sum of *OtherTime* and the sum of *TotalTime* in four layers in **Fig. 12** *(a)-(f)*.

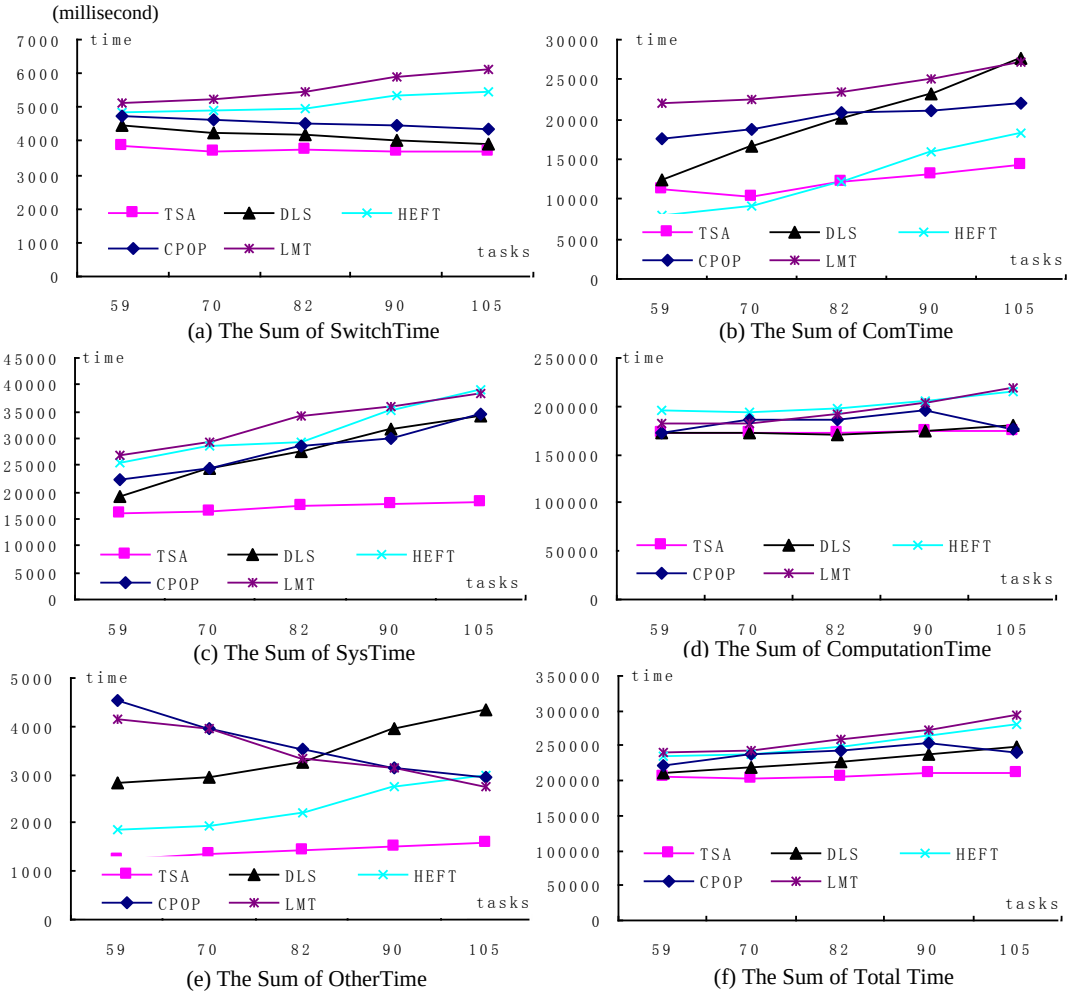

Fig. 12. The comparison of the performance for five algorithms

It is shown from **Fig. 12** (a) and (c) that, for parallel tasks with different subtasks, the switching time and communication time in *TSA* algorithm are less than those in present four algorithms, especially in switching time. With the increase of subtasks, the synchronization time in *DLS*, *HEFT*, *CPOP* and *LMT* increases rapidly, but such time in *TSA* algorithm presents the trend of slow increase. **Fig. 12** (b) show that the communication time in *TSA* algorithm is less than that of *DLS*,*CPOP* and *LMT*, and it is also less than that of *FEFT*, especically when there are more subtasks. And meanwhile, we see that the communication time in *TSA* algorithm presents the trend of slow increase. **Fig. 12** (d) shows that, the computation time in *TSA* algorithm is almost the same as that of *DLS*, *HEFT*, *CPOP* and *LMT*. With the increase of the quantity for tasks, the computation time in *DLS*, *HEFT* and *LMT* presents the trend of slow increase. And only when the quantity of physical machines increases, such time in *CPOP* algorithm decreases slowly. **Fig. 12** (e) shows that the other time in *TSA* algorithm is shorter than that in other algorithms and it can almost be neglected. The other time in *CPOP* and *LMT* decreases ,but such time in *DLS* and HEFT increases with the increase of quantity of subtasks. **Fig. 12** (f) shows that, the total time of *TSA* algorithm is the shortest in five algorithms, the next one is *DLS*, and the last one is *LMT*. The total time of *TSA* algorithm is 5%~8% shorter than that in *DLS*, and is 15~20% than *LMT*.

We analyze the data in Fig.12 further to exact the key data items in its subfigures. After tiding the data, we calculate the weights of the computation time in total time for five algorithms in the quantities of subtasks being 59,70,82,90 and 105 respectively. The result is shown in **Fig. 13**.

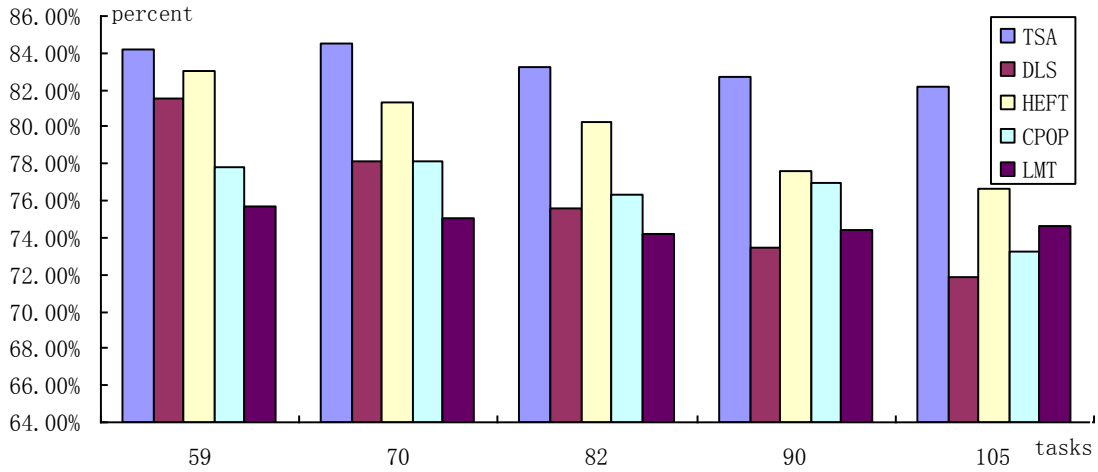

Fig. 13. The weight of computing time in total time for five algorithms

It is shown from **Fig. 13** that, for parallel computing tasks with different subtasks, the weight of computation time in total time in *TSA* algorithm is greater than that in *DLS*, *HEFT*, *CPOP* and *LMT* algorithms. The weight in *TSA* algorithm is about 83~85%. We also see that different quantities of subtasks woud lead to slight change of the weight. The weight in *HEFT* algorithm is above 75%~83%. But different quantities of subtasks lead to great change of weight in *DLS*, *HEFT*, *CPOP* and *LMT* algorithms. For example, for *DLS* algorithm, the weights are 81.56% and 71.83% for 59 subtasks and 105 subtasks, and the distance of them closes to 10%. The data show that *TSA* can utilize and allocate the resources in systems to tasks more efficiently than other algorithms, so it has the lowest time expenditure of all.

The experimental results show that the performance of *TSA* algorithm is better than other algorithms to solve the problem of task placement and scheduling based on virtual machines. Our algorithm reduces the time expenditure in four layers greatly and implements the balance among layers. The computation time of *TSA* algorithm is not much shorter than other algorithms, and the decrease of the total time is completed mainly by reducing the time in switching, communication and synchronization. The *TSA* algorithm has reduced the other time expenditure in addition to such four kinds of time via task scheduling in four layers. *TSA* algorithm has a better performance for most of the indexes, especially the running time, so we think that it is better than other four algorithms in virtualized platform.

7. Conclusions and suggestions

7.1 Conclusions

This paper presents a model for task placement and scheduling in the virtualized high performance computing environment. We introduce the virtualization technology into the clusters and study a methodology for task placement and scheduling based on virtual machines. The shortcomings of related work are summarized to present two problems: the performance of systems and the convenience in applications development and deployment. We describe the

tasks via the thought of feature model to formalize the work of task placement and scheduling. We can find out the highest matching degree between the feature of tasks and the architectures of computing systems via the model of task placement and scheduling. The steps of the workflow and the algorithms of task placement and task execution are designed to obtain the optimal scheme based on the constraint. It is concluded from the experiments that : (1) The estimated time is very close to the actual running time in our study, and the experimental work show the effectiveness of the time estimation approach. (2) *TPVM* algorithm can be convergence within prescribed limits and obtain the optimal solution for any group of physical configuration or personal preference, which shows that it is feasible and reasonable for us. (3) The weight of computation time in total time for *TSA* algorithm is less than those in other four algorithms, so it is better than other four algorithms in performance.Therefore, the optimal task placement and scheduling based on virtual machines can satisfy the requirements of task scheduling in different layers of virtual computing systems to improve their performances.

7.2 Suggestions

Our methodology provides a thought to solve the problems of low performance in clusters and the inconvenience in applications development and deployment. The proposed solution allows users to specify customized constraints for task placement and scheduling. However, there still existed some questions that requires us to do in future:

(1) It is a relatively complex problem for task placement and scheduling based on virtual machines. The unresolved problems are algorithms of task placement and scheduling oriented changing requirements and the construction of feature model for undetermined tasks [63]. We would further discuss these problems in future.

(2) The experimentation of this paper is based on a simulation enviroment *CloudSim*, so we can not make a conclusion that the proposed solution is also actually converge fast enough in a large-scale real clusters. Our methodology might require to make some adjustments to adapt to large-scale *HPC* platforms which compose of thousands of cores. Therefore, the task placement and scheduling problem is even more challenging in those platform. We will verify the feasibility of our thought in large-scale cluster systems in future work.

(3) Our study assumes that the parallel applications have no real-time constraints, so the proposed solution could not be applied to the prediction application with time constraints. But there are many parallel applications having real-time constraints in current virtualized environments, so the model of task placement and scheduling meeting real-time requirements should be considered in next stage.

Acknowledgement

We thank the State 863 projects of China (No. 2007AA010305) and the excellent doctor degree dissertation fund of Xi'an University of Technology (No.102-211007) for supporting this research. Moreover, we thank for the editors of TIIS and the anonymous reviewers for their helpful suggestions on the quality improvement of our present paper.

References

[1] E. Strohmaier, J.J. Dongarra, H.W. Meuer, H.D. Simon, "Recent Trends in the Marketplace of High Performance Computing," *Parallel Computing*, vol. 31, no.3-4, pp. 261-273, Mar. 2005, Article (CrossRef Link)

[2] S.L. Scott, G. Vallée, T. Naughton, H. Ong, "System-level Virtualization Research at Oak Ridge

National Laboratory," *Future Generation Computer Systems*, vol. 26, no. 3, pp. 304-307, Mar. 2009.
[3] J.P. Walters, V. Chaudhary, M. Cha, et al, "A Comparison of Virtualization Technologies for HPC," in *Proc. of 22nd International Conference on Advanced Information Networking and Applications*, pp. 2-5, Mar. 2008,. Article (CrossRef Link)
[4] T.S. Somasundaram, B.R. Amarnath, el al, "CARE Resource Broker: A Framework for Scheduling and Supporting Virtual Resource Management," *Future Generation Computer System*s, vol. 26, no. 3, pp. 337-347, Mar. 2010 . Article (CrossRef Link)
[5] M. Šušteršič , D. Mramor, J. Zupan, "Consumer Credit Scoring Models with Limited Data," *Expert Systems with Applications*, vol. 36,no. 3, pp. 4736-4744, Apr. 2009 . Article (CrossRef Link)
[6] Q. Li, J. Huai, J. Li,et al, " HyperMIP: Hypervisor Controlled Mobile IP for Virtual Machine Live Migration across Networks," in *Proc. of High Assurance Systems Engineering Symposium*, pp. 3-5, Aug. 2008. Article (CrossRef Link)
[7] D.A. Grove, P.D. Coddington, "Modeling Message-passing Programs with a Performance Evaluating Virtual Parallel Machine," *Performance Evaluation*, vol. 60, no. 1-4, pp. 165-187, May 2005. Article (CrossRef Link)
[8] F. Ortin, J.M. Redondo, J.B.G. Perez-Schofield, "Efficient Virtual Machine Support of Runtime Structural Reflection," *Science of Computer Programming*, vol. 74, no. 10, pp. 836-860, Aug. 2009. Article (CrossRef Link)
[9] M. Bellato, R. Isocrate, G. Meng,e tc al, "Remoting Field Bus Control by Means of a PCI Express-based Optical Serial Link," *Nuclear Instruments & Methods in Physics Research*, vol. 570, no. 3, pp. 518-524, Aug. 2007. Article (CrossRef Link)
[10] S. Fu, "Failure-Aware Resource Management for High-Availability Computing Clusters with Distributed Virtual Machines," *Journal of Parallel and Distributed Computing*, vol. 70, no. 4, pp. 384-393, Apr. 2010. Article (CrossRef Link)
[11] T. Giorgino, M.J. Harvey, G. de Fabritiis, "Distributed Computing as a Virtual Supercomputer: Tools to Run and Manage Large-scale BOINC Simulations," *Computer Physics Communications*, vol. 181, no. 8, pp. 1402-1409, Mar. 2010. Article (CrossRef Link)
[12] S.-M. Lee,D.-G. Kim, D.-R. Shin, "General Purpose Hardware Abstraction Layer for Multiple Virtual Machines in Mobile Devices," in *Proc. of International Conference on Advanced Communication Technology*, pp. 15-18, Feb. 2009. Article (CrossRef Link)
[13] M. Arnold, S.J Fink, D. Grove, M. Hind, P.F. Sweeney, "A Survey of Adaptive Optimization in Virtual Machines," in *Proc. of the IEEE Special Issue on Program Generation,Optimization,and Adaptation*, vol. 93, no. 2, pp. 449-466, June 2005. Article (CrossRef Link)
[14] K. Wang, A. Chang, L.V. Kale, J.A. Dantzig, "Parallelization of a Level Set Method for Simulating Dendritic Growth," *Journal of Parallel and Distributed Computing*, vol. 66, no. 11, pp. 1379-1386, Nov. 2006. Article (CrossRef Link)
[15] A. Acharya, N. Banerjee, D. Chakraborty, et al, "Programmable Presence Virtualization for Next-Generation Context-based Applications," in *Proc. of 2009 IEEE International Conference on Pervasive Computing and Communications*, pp. 9-13, Mar. 2009. Article (CrossRef Link)
[16] S.K. Nair, P.N.D. Simpson, B. Crispo, A.S. Tanenbaum, "A Virtual Machine Based Information Flow Control System for Policy Enforcement," *Electronic Notes in Theoretical Computer Science*, vol. 197, no. 1, pp. 3-16, Feb. 2008. Article (CrossRef Link)
[17] J. Wiegert, G. Regnier, J. Jackson, "Challenges for Scalable Networking in a Virtualized Server," in *Proc. of 16th International Conference on Computer Communications and Networks*, pp. 13-16, Aug. 2007. Article (CrossRef Link)
[18] B. Li, J. Shu, W. Zheng, "Design and Implementation of a Storage Virtualization System Based on SCSI Target Simulator in SAN," *Tsinghua Science & Technology*, vol. 10, no. 1, pp. 122-127, Jan. 2005. Article (CrossRef Link)
[19] X. Wanga, Z. Dua,Y. Chenb, S. Lia, "Virtualization-based Autonomic Resource Management for Multi-tier Web Applications in Shared Data Center," *Journal of Systems and Software*, vol. 81, no. 9, pp. 1591-1608, Sep. 2008. Article (CrossRef Link)
[20] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, "Resource Pool Management- Reactive versus Proactive or Let's be Friends," *Computer Networks*, vol. 53, no. 17, pp. 2905-2922, Dec. 2009.

Article (CrossRef Link)
[21] A. Mtibaa, B. Ouni, M. Abid, "An Efficient List Scheduling Algorithm for Time Placement Problem," *Computers & Electrical Engineering*, vol. 33, no. 4, pp. 285-298, July 2007. Article (CrossRef Link)
[22] K.G. Kakoulis, I.G. Tollis, "Algorithms for the Multiple Label Placement Problem," *Computational Geometry*, vol. 35, no. 3, pp. 143-161, Oct. 2006. Article (CrossRef Link)
[23] M. Fazlali, M. Sabeghi, A. Zakerolhosseini, K. Bertels, "Efficient Task Scheduling for Runtime Reconfigurable Systems," *Journal of Systems Architecture*, vol. 56, no. 11, pp. 623-632, Nov. 2010. Article (CrossRef Link)
[24] Y.-S. Dai , G. Levitin, X. Wang, "Optimal Task Partition and Distribution in Grid Service System with Common Cause Failures," *Future Generation Computer Systems*, vol. 23, no. 2, pp.2 09-218, Feb. 2007. Article (CrossRef Link)
[25] H.J. Siegel, S. Ali , "Techniques for Mapping Tasks to Machines in Heterogeneous Computing Systems," *Journal of Systems Architecture* , vol. 46, no. 8, pp. 627-639, June 2000. Article (CrossRef Link)
[26] T.D. Braun, H.J. Siegel, A.A. Maciejewski, Y. Hong, " Static Resource Allocation for Heterogeneous Computing Environments with Tasks Having Dependencies, Priorities, Deadlines, and Multiple Versions," *Journal of Parallel and Distributed computing*, vol. 68, pp. 11, pp. 1504-1516, Nov. 2008. Article (CrossRef Link)
[27] J.-K. Kim, S. Shivle, H.J. Siegel, et al, "Dynamically Mapping Tasks with Priorities and Multiple Deadlines in a Heterogeneous Environment," *Journal of Parallel and Distributed Computing*, vol. 67, no. 2, pp. 154-169, Feb. 2007. Article (CrossRef Link)
[28] Y.-K. Kwok, A.A. Maciejewski, H.J. Siegel, I. Ahmad, A. Gharfoor, "A Semi-static Approach to Mapping Dynamic Iterative Tasks onto Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 66, no. 1, pp. 77-98, Jan. 2006. Article (CrossRef Link)
[29] M. Moore, "An Accurate Parallel Genetic Algorithm to Schedule Tasks on a Cluster," *Parallel Computing*, vol. 30, no. 5-6, pp. 567-583, May 2004. Article (CrossRef Link)
[30] Sau-Ming Lau,Qin Lu,Kwong-Sak Leung, "Adaptive load distribution algorithms for heterogeneous distributed systems with multiple task classes," *Journal of Parallel and Distributed Computing*, vol. 66, no. 2, pp. 163-180, Feb. 2006. Article (CrossRef Link)
[31] ZhuHai,Wang Yu-ping, "Constrained Multi-objective Grid Task Security Scheduling Model and Algorithm," *Journal of Electronics & Information Technology*, vol. 32, no. 4, pp. 988-992, Mar. 2010. Article (CrossRef Link)
[32] James Broberg, Zahir Tari, Panlop Zeephongsekul, "Task assignment with work-conserving migration," *Parallel Computing*, vol. 32, no. 11-12, pp. 808-830, Dec. 2006. Article (CrossRef Link)
[33] Laura Gilbert, Jeff Tseng, Rhys Newman, etc, "Implications of virtualization on Grids for high energy physics applications," *Journal of Parallel and Distributed Computing*, vol. 66, no. 7, pp. 922-930, July 2006. Article (CrossRef Link)
[34] Yong Liao, Dong Yin, Lixin Gao, "Network virtualization substrate with parallelized data plane," *Computer Communications*, vol. 34, no. 13, Aug. 2011. Article (CrossRef Link)
[35] Flavio Lombardi, Roberto Di Pietro, "Secure virtualization for cloud computing," *Journal of Network and Computer Applications*, vol. 34, no. 4, July 2011. Article (CrossRef Link)
[36] Michail D. Flouris, Renaud Lachaize, Konstantinos Chasapis, Angelos Bilas, "Extensible block-level storage virtualization in cluster-based systems," *Journal of Parallel and Distributed Computing*, vol. 70, no. 8, pp. 800-824, Aug.2010. Article (CrossRef Link)
[37] Mohammad I. Daoud, Nawwaf Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 4, pp. 399-409, April 2008. Article (CrossRef Link)
[38] J.H. Abawajy, "Adaptive hierarchical scheduling policy for enterprise grid computing systems," *Journal of Network and Computer Applications*, vol. 32, no. 3, pp. 770-779, May 2009. Article (CrossRef Link)
[39] Matthew Witten, "The role of high performance computing in medicine and public health," *Future*

Generation Computer Systems, vol. 10, no. 2-3, pp. 223-232, June 1994. Article (CrossRef Link)
[40] Hung-Ming Chen, Yu-Chin Lin, "Web-FEM: An internet-based finite-element analysis framework with 3D graphics and parallel computing environment," *Advances in Engineering Software*, vol. 39, no. 1, pp. 55-68, Jan. 2008. Article (CrossRef Link)
[41] Tevfik Kosar, Miron Livny, "A framework for reliable and efficient data placement in distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 65, no. 10, pp. 1146-1157, Oct. 2005. Article (CrossRef Link)
[42] B. Hamidzadeh, L.Y. Kit, D.J. Lilja, "Dynamic task scheduling using online optimization," *IEEE Trans. Parallel Distrib. Syst*., no.11, pp. 1151–1163, Nov. 2000. Article (CrossRef Link)
[43] S. Bansal, P. Kumar, K. Singh, "Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs," *J. Parallel Distrib. Comput*, no. 65, pp. 479-491, Apr. 2005. Article (CrossRef Link)
[44] [W.F. Boyer, G.S. Hura, "Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments," *J. Parallel Distrib. Comput*., no. 65, pp. 1035-1046, Sep. 2005. Article (CrossRef Link)
[45] E. Ilavarasan, P. Thambidurai, R. Mahilmannan, "Performance effective task scheduling algorithm for heterogeneous computing system," in *Proc. 4th International Symposium on Parallel and Distributed Computing*, France, pp. 28-38, July 2005. Article (CrossRef Link)
[46] J. Kim, J. Rho, J.-O. Lee, M.-C. Ko, , "CPOC: Effective static task scheduling for grid computing," in *Proc. 2005 International Conference on High Performance Computing and Communications*, Italy , pp. 477-486, Apr. 2005. Article (CrossRef Link)
[47] Xiangzhen Kong, ChuangLin, YixinJiang,et al, "Efficient dynamic task scheduling in virtualized data centers with fuzzy prediction," *Journal of Network and Computer Applications*, pp. 1068-1077 July 2010. Article (CrossRef Link)
[48] A. Iosup, C. Dumitrescu, D. Epema, H. Li, L. Wolters, "How are real grids used? The analysis of four grid traces and its implications," in *Proc. 7th IEEE/ACM International Conference on Grid Computing*, Spain, pp. 262-269, Sep. 2006. Article (CrossRef Link)
[49] H. Topcuoglu, S. Hariri, M.Y. Wu , "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst*. no. 13, pp. 260-274, Mar. 2002. Article (CrossRef Link)
[50] S. Bansal, P. Kumar, K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst*. no. 14, pp. 533-544, june 2003. Article (CrossRef Link)
[51] A.Y. Zomaya, Y.H. Teh , "Observations on using genetic algorithms for dynamic load balancing," *IEEE Trans. Parallel Distrib. Syst*., no. 12, pp. 899-911, Sep. 2001. Article (CrossRef Link)
[52] S. Baskiyar, C. Dickinson, "Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication," *J. Parallel Distrib. Comput*., no. 65, pp. 911–921, Aug. 2005. Article (CrossRef Link)
[53] J. Grefenstette, Rank-based selection, in: T. Back, D.B. Fogel, Z. Michalewicz, "Handbook of Evolutionary Computation, first ed," Oxford Univ. Press, pp. 241-246,1997
[54] A. Radulescu, A.J.C. van Gemund , "Low-cost task scheduling for distributedmemory machines," *IEEE Trans. Parallel Distrib. Syst*., no. 13, pp. 648-658, Sep. 2002. Article (CrossRef Link)
[55] M. Wu, D. Dajski, "Hypertool: A programming aid for message passing systems," *IEEE Trans. Parallel Distrib. Syst.*, no. 1, pp. 330-343, July 1990. Article (CrossRef Link)
[56] S. Nesmachnow, H. Cancela, E. Alba, "Heterogeneous computing scheduling with evolutionary algorithms," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 15, no. 4, pp. 685-701, Apr. 2010. Article (CrossRef Link)
[57] P. Phinjaroenphan, S. Bevinakoppa, P. Zeephongsekul, "A method for estimating the execution time of a parallel task on a grid node," *Lecture Notes in Computer Science,* vol. 3470, pp. 226-236, Jan. 2005. Article (CrossRef Link)
[58] M.A. Iverson, F. Ozguner, L. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *IEEE Trans. Comput*., pp. 1374-1379, Apr. 1999. Article (CrossRef Link)

[59] G.C. Sih, E.A. Lee, "A compile-time scheduling heuristic for interconnectionconstrained heterogeneous processor architectures," *IEEE Trans. Parallel Distrib. Syst*., no. 4, pp. 175-187, Feb. 1993. Article (CrossRef Link)
[60] H. El-Rewini, T.G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *J. Parallel Distrib. Comput*., no. 9, pp. 138-153, June 1990. Article (CrossRef Link)
[61] S.V. Kumar, C.D. Peters-Lidard, Y. Tian, etc, "Land information system: An interoperable framework for high resolution land surface modeling," *Environmental Modelling & Softwar*, vol. 21, no. 10, pp. 1402-1415, Feb. 2006. Article (CrossRef Link)
[62] Deshi Ye, Guochuan Zhang, "On-line scheduling of multi-core processor tasks with virtualization," *Operations Research Letters*, vol. 38, no. 4, pp. 307-311, July 2010. Article (CrossRef Link)
[63] Katia Leal, Eduardo Huedo, Ignacio M. Llorente, "A decentralized model for scheduling independent tasks in Federated Grids," *Future Generation Computer Systems*, vol. 25, no. 8, pp. 840-852, Sep. 2009. Article (CrossRef Link)

XiaoJun Chen, male, doctoral student. He was born in GuiLin city, GuangXi province, P.R. China in September, 1980. He received bachelor degree in Department of Industrial Engineering of Xi'an University of Technology in 2004 and the master degree in School of Economics and Management of Xi'an University of Technology in 2009.He entered into School of Computer Science and Engineering of Xi'an University of Technology to learn about distributed computing technology in 2008. He engaged in software development job from September 2004 to September 2006 in Delta. Software Company. He has published 8 papers since 2006, and the representatives are as follows: Resource Allocation and Adjustment with Genetic Algorithm in Virtual Computing Systems, ICIC Express Letters,2010;Empirical research on quality supervision elements of cooperative manufacturing process in virtual enterprise, ICPOM, 2008 ;Research on quality supervision model for cooperative manufacturing preparing process in virtual enterprise, ICRMEM,2008;et al . Now he is engaged in the researches of virtual technology and cloud computing. XiaoJun Chen is now a member of IEEE Computer Society. E-mail: army.net@163.com

Jing Zhang, male, doctor, professor, doctoral supervisor. He was born in BaoJi city, ShaanXi, province, P.R.China in November, 1952. He received bachelor degree in Department of Automatic Control of Xi'an University of Technology in 1981, the master degree in Department of Software and Theory of Xi'an JiaoTong University in 1986, and the doctor degree in Department of Systems Engineering of Xi'an JiaoTong University in 1994. He has worked in Department of Computer of Xi'an University of Technology since 1977, and now is a professor and the PhD supervisor of School of Computer Science and Engineering, Xi'an University of Technology. Of which, he worked in Computer Training Center of Ministry of Education, PR,China in 1982 for a short time. He has published 60 papers and hosted 20 research projects in recent 10 years, of which, completed 4 the State 863 projects. He has published 4 books, of which, representatives are Artificial intelligence basis(Electronic Industry Press ,BeiJing,2000), Practical Course on Computer Network(Electronic Industry Press ,BeiJing,2007) and Computer Network(Xidian University Press,Xi'an,Shaanxi, 2007). Recently he concentrates on the researches of distributed system, virtualization , grid computing and cloud computing. Dr. Jing Zhang is now one of the national key new product projects consultants and a Xi'an Information technology consultant. He is also a ShaanXi manufacturing Informatization expert, the member of Services Computing Professional Committee in China Computer Federation, member of E-Government and Office Automation Committee in China Computer Federation, the evaluation expert of National Natural Science Foundation and the member of IBM Software Innovation Center Expert Committee.E-mail: zhangjing@xaut.edu.cn. (Corresponding author)

JunHuai Li, male, doctor, professor, master supervisor. He was born in BaoJi city, ShaanXi, province, P.R.China in November, 1969. He received bachelor degree in Department of Computer Science and technology of Xi'an University of Technology in 1994 ,the master degree in Department of Computer Application of Xi'an University of Technology in 1997, and the doctoral degree in Department of Software and Theory of NPU in 2002. He made a cooperation research in university of Tsukuba, Japan in 2004. He has worked in Department of Computer Science and Engineering of Xi'an University of Technology since 1997, and now is an professor and a master supervisor of School of computer science and engineering, Xi'an University of Technology. He is also the dean of Department of Computer Science and Technology and the dean of Network and Information Management Center. He has published 40 papers and hosted 12 research projects in recent 10 years, of which, completed 4 the State 863 projects. He has published 3 books, of which, representatives are Practical Course on Computer Network(Electronic Industry Press ,BeiJing,2007) and Computer Network(Xidian University Press,Xi'an,Shaanxi, 2007) and Network Security Technology (Xidian University Press,Xi'an,Shaanxi, 2010).He is engaged in the researches of network computing, distributed computing internet technology, RFID technology and web data mining.Dr JunHuai Li is now a member of Chinese Computer Society and also a member of IEEE. E-mail: lijunhuai@xaut.edu.cn.