A Novel Algorithm for Optimizing Multiple Services Resource Allocation

Amjad Gawanmeh
Department of Electrical and Computer Engineering, Khalifa University, UAE and
Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada

Alain April
Department of Software Engineering, Université du Qu’ebec, École de Technologie Supérieure, Montreal, Canada

Abstract—Resource provisioning becomes more and more challenging problem in cloud computing environment since cloud-based services become more numerous and dynamic. The problem of scheduling multiple tasks for multiple users on a given number of resources is considered NP-Complete problem, and therefore, several heuristic based research methods were proposed, yet, there are still many improvements can be done, since the problem has several optimization parameters. In addition, most proposed solutions are built on top of several assumptions and simplifications by applying computational methods such as game theory, fuzzy logic, or evolutionary computing. This paper presents an algorithm to address the problem of resource allocation across a cloud-based network, where several resources are available, and the cost of computational service depends on the amount of computation. The algorithm is applicable without restrictions on cost vector or compaction time matrix as opposed to methods in the literature. In addition, the execution of the algorithm shows better utility compared to methods applied on similar problems.

Keywords—Cloud computing; Cloud Services; Scheduling; Parallel and Distributed systems

I. INTRODUCTION

Cloud computing [1] has emerged recently as a new paradigm that moved enterprise computing from the classical host-based architecture pattern into the elastic computing pattern. This new service-oriented vision delivers resources and applications on-demand based on the pay-per-use concept. The cloud system should provide the application users with robustness, fault tolerance, execution automation, and powerful computing facilities. This implies various cloud service requirements and QoS to be maintained. Therefore, several challenges arise throughout the design development and deployment of these systems. In addition, unlike conventional software and hardware systems, a wide range of different issues should be considered in the design and operation of cloud based systems. On the other hand, the size of data centers have been continuously increased in order to accommodate the increasing demand while at the same time reducing the management costs. Finally, virtualization has been heavily used in order to increase the utilization of server resources by consolidating many virtual machines into a single physical server. Therefore, as cloud-based services become more numerous and dynamic, resource provisioning becomes more and more challenging.

While QoS problems in service and cloud based systems have been addressed in the literature, the problem of constrained resource allocation problem attracts a lot of attention, as it is considered difficult problem since it is affected by several aspects, such as assumptions about the services, tasks, subtasks, and communication between servers. The problem of scheduling multiple tasks for multiple users on given number of resources is considered difficult, and therefore, several research methods were proposed, yet, there are still many improvements can be done, since the problem has several optimization parameters and in addition, most proposed solutions are built on top of several assumptions and simplifications.

In this paper, we address the problem of resource allocation in which service demanders intend to solve sophisticated parallel computing problem by requesting the usage of resources across a cloud-based network, where several resources are available, and the cost of computational service depends on the amount of computation. The contributions in this paper include providing a simple algorithm for resource allocation for single user with multiple subtasks. Then, propose another algorithm for the multiple users problem using a selection function. The proposed algorithm is illustrated on example that was proposed in the literature, and the proposed method in this paper outperforms previous ones by two aspects: first, it can be applied on the general case without any restriction on the type of the computation time matrix for resource, nor on the cost function, as opposed to previous work. Second, the algorithm provides better schedule in terms of utility.

The rest of this paper is organized as follows: Section II provides a brief review on the state of the art on the subject. Section III describes the formalization of the single and multiple users problem. Section IV presents the proposed algorithm to solve the single user scheduling problem. Section V presents a selection function based algorithm for scheduling the multiple users problem. The proposed method is illustrated on an example in Section VI. Finally Section VII concludes the paper with future work hints.

II. RELATED WORK

In this section we present a brief state of the art review on related work that addressed the problem of resource allocation in cloud computing. In fact, more detailed reviews can be found in several surveys, for instance, [2], [3], [4], [5], [6], and [7].
Walker et al. [8] presented a model in which the resultant profit is calculated by obtaining the storage space from cloud. Buyya et al. [9] have proposed an infrastructure for resource allocation in multiple clouds based on commodity pricing method. The work in [10] used virtualization in order to allocate data center resources dynamically based on application demands and support. The authors used skewness to measure the unevenness in the multidimensional resource utilization of a server and combined different types of workloads hoping to improve the overall utilization of server resources by reducing skewness. The work in [11] proposed a resource allocation technique by ranking the tasks of users based on error criteria to maintain the consistency in the priorities of the tasks. The authors in [12] presented a web semantic based resource allocation method using multi-agent technologies to model interoperability between the users and different resource.

Ergu et al. [13] proposed a model for task-oriented resource allocation in a cloud computing environment based on available resources and user preferences. The method considers response time as the measure in their allocation procedure. Genetic algorithms based tasks scheduling in cloud systems were addressed in [14], [15], and [16]. The work in [17] addressed distributed convergence to fair allocations of CPU resources for time-sensitive applications for applications with service-level adaptation, that are executed by each application independently. The work in [18] addressed resource allocation in computational mobile grid.

Karthik et al. [19] suggested to allocate the resources based on the speed and cost of different VMs in IaaS. It differs from other related works, by allowing the user to select VMs and reduces cost for the user. Zhen Kong et al. [20] proposed a mechanism to allocate virtualized resources among selfish VMs in a non-cooperative cloud environment. Hence, VMs care essentially about their own benefits without any consideration for others. Paul et al. [21] allocated cloud resources by obtaining resources from remote nodes when there is a change in user demand. In this method over provisioning and under-provisioning of resources can cause several issues. The work in [22] dynamically allocated CPU resources to meet QoS objectives by first allocating requests to high priority applications. This method depends on prioritizing tasks which has several limitations in cloud environment. The work in [23] proposed to use live migration as a resource provisioning mechanism but all of them use policy based heuristic algorithm to live migrate VM which is difficult in the presence of conflicting goals.

Several other computing techniques were adopted for resource allocation in cloud computing. For instance, game theory has also been used in several locations for resource allocation in cloud computing, for instance in [24], and [25]. Several other methods used fuzzy pattern recognition to propose dynamic resource-allocation algorithms for multiple cloud nodes such as [26], [27], and [28]. On the other hand, cloud resource allocation using auction mechanism is addressed in [29], [30], [31], [32], and [33]. These mechanisms does not ensure profit maximization due to its truth telling property under constraints.

Despite the large amount of work on the subject, there is still lack of practicable solution for cloud computing systems because most cloud-based computational services are multiple QoS-constrained, in order to improve the utility of scheduling algorithms [25]. The problem addressed in this paper considers multi-users with dependent tasks, each is composed of several subtasks to be scheduled on multiple resources. This paper presents a generic solution that can be applied the problem without the constraints on the execution time and cost imposed in previous work. The proposed mechanism is based on utility function trying to find a local optimum solution for the NP-hard problem.

III. PROBLEM DESCRIPTION AND FORMALIZATION

We first present problem description for resource allocation for a single user, then we describe the general case, for multiple users as presented in previous work. A user $u$ has a service $S$ that is composed of $k$ sub tasks that are parallel and dependent subtasks with equal amount of computation. In fact, this assumption is practical, since in case of unequal amounts of computing, the cost values of resources can be adjusted to normalize the amount of computations for tasks. There are $m$ computational resources that are available to the user, $\{R_1, R_2, ..., R_m\}$. Each resource, $R_i$, has a fixed price $p_i$ according to its capacity forming the price vector $p = \{p_1, p_2, ..., p_m\}$. In addition, each resource $R_i$ require specific time, $t_i$, to execute any subtask forming the execution time vector $t = \{t_1, t_2, ..., t_m\}$. The objective is to assign subtasks for each user to a number of resources such that the total cost is minimized, where cost represents the expense and execution time for completing tasks for all users. The scheduling problem solution is a non-negative vector $v$ of $m$ elements, each represent the number of subtasks assigned for each resource. The entry $v_i$ is the number of subtasks of the task $S$ allocated to resource $R_i$. The allocation vector $v$ must satisfy $\sum_{i=1}^{m} v_i = k$.

We derive two vectors from vectors $v$, $t$, and $p$ as follows: the first is the completion time vector, $\hat{t}$ and the second is the expense vector $\hat{e}$. The entry $\hat{t}_i$ of $\hat{t}$ is the turnaround time it takes for resource $R_i$ to complete $v_j$ subtasks of the task $S$. The entry $\hat{e}_i$ of vector $\hat{e}$ is the expense $S$ pays for resource $R_i$ to complete $v_j$ subtasks. These two vectors are defined as follows: $\hat{t} = v \cdot t$, $\hat{e} = v \cdot t \cdot p$. Based on these we calculate two values for schedule $v$, the total execution time $t_{max}$, and the total expense $e_{max}$. The execution time for task $S$ is the maximum execution time of tasks assigned to resources, $t_{max} = \text{max}(\{t_i|i \in \hat{t}\})$, where $t_i$ denotes the $i^{th}$ element of the vector $\hat{t}$. The total expense $e_{v}$ is the summation of all expenses paid to all resources, $e_{v} = \sum_{i=1}^{m} e_i$. We assign weights for schedule costs as follows, $w_x$ for execution time weight, and $w_e$ for expense weight. Then we can define a merit value of the expense using the following utility function:

$$u(v) = \frac{1}{w_x \times t_v + w_e \times e_v} = \frac{1}{w_x \times \text{max}(\{t_i|i \in \hat{t}\}) + w_e \times \sum_{i=1}^{m} e_i}$$

The objective is to find $v$ that maximizes $u$, while we will present our proposed methodology for a generalizers case, as opposed to the presented solution in there. Next we give an example to illustrate the problem described above. Given five computational resources $(R_1 - R_5)$, $m = 5$, and given the price vector $p = (1, 1.2, 1.5, 1.8, 2)$, there is a task $(S)$, that has three subtasks, $k = 3$. The execution time vector for each subtask using above resources is given as $t = (5, 4.2, 3.6, 3, 2.8)$. Assume that a
schedule \(v = (1, 1, 1, 0, 0)\) is used, then we can calculate \(t\) and \(\hat{e}\) as follows: \(t = (5, 4, 2, 3, 6, 0, 0)\) and \(\hat{e} = (5, 5, 0.4, 5, 4, 5, 0, 0)\), then we can calculate, \(t_v = 5\) and \(e_v = 15.44\). Assuming \(w_e = w_t = 1\), then \(u = \frac{1}{w_t+w_e} = 0.0489\). On the other hand, the schedule \(v = (0, 0, 1, 1, 1)\) will yield to \(u = 0.05\).

Next, we present the model for the problem with multiple users sharing the same number of resources as described above. Given \(n\) users \(\{u_1, u_2, \ldots, u_n\}\), who are interested in executing \(n\) services, \(\{S_1, S_2, \ldots, S_n\}\), where every \(S_i\) is one single task that is composed of \(k_i\) subtasks, that are parallel and dependent subtasks with equal amount of computation. \(k = \{k_1, k_2, \ldots, k_n\}\) represents the number of subtasks for all users. There are \(m\) computational resources that are shared by all users, \(\{R_1, R_2, \ldots, R_m\}\) with price vector \(p\). All users who allocate their services to resource \(R_j\) proportionally share the capacity and expense of resource \(R_j\). In addition, each resource \(R_t\) require specific time, \(t_{ij}\), to execute subtask \(\tau\) for user \(u_j\) using resource \(R_t\), forming the execution time vector \(T\) for every resource. The collection of all vectors \(T_1, T_2, \ldots, T_m\) forms the execution time matrix \(T\). Since all subtasks of \(S_i\) are parallel and dependent, the completion time of task \(S_i\) is defined similar to above as \(max\{T_{ij}\} < T_i\}\), where \(T_i\) denotes the element of matrix \(T\) in row \(i\) and column \(j\).

The objective is to assign subtasks for each user to a number of resources such that the the total cost is minimized, where cost represents the expense and execution time for completing tasks for all users. The scheduling problem solution is a non-negative matrix \(A\) of \(m\) rows and \(n\) columns, each element represents the number of subtasks assigned for each resource. The entry \(A_{ij}\) is the number of subtasks of the task \(S_i\) allocated to resource \(R_j\). The allocation matrix \(A\) must satisfy the following constraint \(\sum_{j=1}^{n} A_{ij} = k_j\), where \(j = \{1, 2, \ldots, m\}\).

We derive two matrices from matrix \(T\), schedule \(A\), and vector \(p\) as follows: the first, is the completion time matrix, \(\hat{T}\) and the second is the expense matrix \(\hat{E}\). The entry \(\hat{T}_{ij}\) of \(\hat{T}\) is the turnaround time it takes for resource \(R_t\) to complete \(A_{ij}\) subtasks of the task \(S_i\). The entry \(\hat{E}_{ij}\) of matrix \(\hat{E}\) is the expense user \(S_i\) pays for resource \(R_j\) to complete \(A_{ij}\) subtasks. These two matrices are defined as follows:

\[
\hat{T}_{ij} = \sum_{l=1}^{n} A_{ij} \cdot t_{ij},
\]

\[
\hat{E}_{ij} = \frac{\hat{T}_{ij} \cdot p_j}{\sum_{l=1}^{n} A_{lj}},
\]

Based on these two matrices, we calculate two vectors for schedule \(A\), the first vector, \(T_{max}\), is composed of \(m\) elements, each element \(T_{max}\), represents the execution time for task \(S_i\) using schedule \(A\), and is defined as the maximum execution time of tasks assigned by user \(i\) to all resources, \(T_{max} = max\{T_{ij}\}\) \(j = \{1, 2, \ldots, n\}\). The second vector, \(E_{sum}\), is composed of \(m\) elements, each element, \(E_{sum}\), represents the total expense for user \(S_i\), and is defined as the summation of all expenses paid to all resources by user \(i\), \(E^A = \sum_{j=1}^{n} \hat{E}_{ij}\). We assign weights for schedule costs as follows, \(w_t\) for execution time weight, and \(w_e\) for expense weight. Then we can use definition for a merit value of the expense use the utility function defined above:

\[
u_t^1 = \frac{1}{w_t \cdot T_{max} + w_e \cdot E^A}, \quad \hat{u} = \frac{1}{w_t \cdot max\{T_{ij}\} + w_e \cdot \sum_{j=1}^{m} E_{ij}}.
\]

Then the utility of schedule \(A\) can be defined as the summation of utilities of all users: \(u^A = \sum_{i=1}^{n} u^A_i\).

Next we give an example to illustrate the problem described above for multiple users. We will use a similar problem to the one described in [25], however, problem description in [25] is based on the following two assumptions: (1) the price vector of all resources \(p = (p_1, p_2, \ldots, p_m)\) satisfies \(p_1 < p_2 < \cdots < p_m\), and (2) the corresponding execution time of any subtask of an arbitrary task \(S_i\) satisfies \(T_{i1} > T_{i2} > \cdots > T_{im}\). We believe that the above assumption is valid for only limited applications, and the method used based on game theory can have a Nash equilibrium only under the given assumption. The algorithm proposed in this paper can be applied on any price vector, \(p\), and execution time matrix, \(T\), regardless of the order. We will explain the problem statement using the same example, since it is going to be used as illustrative example on the proposed algorithm later. Given the price vector \(p\) described above, \(k = \{2, 3, 4\}\), the execution time matrix \(T\) and schedule \(A\) given as below, then we calculate the two matrices \(T\), \(\hat{E}\) and two vectors \(T_{max}\) and \(E_{sum}\) as follows:

\[
T = \begin{pmatrix} 6 & 5 & 4 & 3.5 & 3 \\ 4 & 3.5 & 3 & 2.8 & 2.4 \end{pmatrix}, \quad A = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}
\]

\[
\hat{T} = \begin{pmatrix} 0 & 0 & 0 & 7 & 6 \\ 0 & 8.4 & 7.2 & 6 & 0 \\ 4 & 4.2 & 4.8 & 0 & 4.8 \end{pmatrix}, \quad \hat{E} = \begin{pmatrix} 6.3 & 6 \\ 5.04 & 5.4 & 5.4 & 0 \\ 4 & 4.2 & 4.8 & 0 & 4.8 \end{pmatrix}
\]

\[
A^A = (7, 8.4, 7), \quad E^A = (12.3, 15.84, 17.8)
\]

\[
u^A_i = \left( \begin{array}{c} \frac{1}{7+4.123} = 0.051835 \\ \frac{1}{8.4+15.84} = 0.0412541 \\ \frac{1}{17.8} = 0.0403226 \end{array} \right), \quad \text{and finally}
\]

\[
u^A = \sum_{i=1}^{n} u^A_i = 0.13339.
\]

This example shows the optimization needed in order to enhance utility of the scheduling problem, while there is tradeoff between the execution time and the price for every assignment. First, a proposed solution for the single user case is provided, where the optimum solution can be obtain. In fact, while similar problems have been addressed in the literature, we believe that the solution we provide is the most efficient one. The algorithm for the multiuser problem, which is considered NP compete problem, will be based on a selection function that is designed using the solution for the single user problem. We define a function that calculates the effect of scheduling tasks into resources, and use it for the choice of proper allocation of every service.
Algorithm 1: Single User Resource Allocation

1: procedure SINGLE USER ALLOCATION
2: Input: k, m, t, p, n.
3: Output: v.
4: Initialize:
5: ● Initialize vectors and variables
6: v = 0, t_{max} = 0, j = 0; initialize all elements to 0
7: ● Initialize the allocation function vector \( \delta \)
8: REPEAT
9: \( \delta_j = p_j \times t_j \)
10: \( j = j + 1 \)
11: UNTIL \( j = k \); simulation time ends
12: ● Process all elements elements using allocation function
13: v = 0; Reset counter
14: REPEAT
15: find \( l \) such that
16: (1) \( 1 \leq l \leq n \),
17: (2) \( \forall i \cdot 1 \leq i \leq n \Rightarrow \delta_i \leq \delta_l \).
18: \( v_l = v_l + 1 \), allocate current task to \( R_l \)
19: if \( v_l > t_{max} \) then
20: \( t_{max} = v_l \times t_l \)
21: \( \delta_l = \max(t_{max}, t_l \times (1 + v_l)) \)
22: \( j = j + 1 \)
23: UNTIL \( j = k \); all tasks are allocated
24: END
25: END procedure

IV. Optimizing Single User Scheduling

In this section we present the solution for the single user optimization problem based on the description in the previous section. The objective is to find the allocation vector \( v \) that maximizes the utility \( u(v) \). For \( k \) subtasks and \( m \) resources, it is obvious that the search space for the problem is non-polynomial, hence exploring all possible scenarios is nonlinear.

In order to show how the proposed algorithm finds the optimum solution, we first create a variable \( t_{max} \) and initialize it to 0. Then, we define a selection function, \( \delta \), as a vector of \( m \) elements, \( \delta = (\delta_1, \delta_2, \ldots, \delta_m) \), which is defined initially as follows:

\[
\delta_l = \max(t_{max}, t_l \times (1 + v_l)) + p_l \times t_l \times (1 + v_l).
\]

Then, we start the allocation of \( k \) subtasks by following an iterative process, where in every iteration, we assign one subtask into one resource. We choose \( l \), such that satisfies the following conditions: (1) \( 1 \leq l \leq n \), and (1) \( \forall i \cdot 1 \leq i \leq n \Rightarrow \delta_i \leq \delta_l \). This means that \( \delta_l \) is minimum in \( \delta \). Then, we assign the current subtask into resource \( R_m \). After the assignment, we modify the schedule \( v \) by incrementing \( v_l \), calculate the new \( t_l \), \( \delta_l \), and \( t_{max} \). Then we repeat the process again, until all items are processed.

This algorithm has a complexity of worst case \( O(nm) \), i.e. it should perform in linear time vs \( n \times m \). Applying the above algorithm on the single user example explained above will result in \( v = (0, 0, 1, 1, 1) \), with utility \( u = 0.05 \). In addition considering the scheduling problem for multiusers presented in the previous section, if the problem is solved individually, i.e. every user schedule his subtasks using the algorithm 1 above, then, the resulting schedule will be \( v_1 = (0, 0, 0, 1, 1), v_2 = (0, 0, 1, 1, 1), v_3 = (1, 1, 1, 0, 1) \), all combined together will give schedule \( A \) provided below.

Hence, under this assumption, the schedule for multiuser will have the utility of \( u^A = 0.1273 \). Obviously, this is not the optimum solution for the multiuser case. For instance, the schedule given in the example in the previous section yields better utility. Therefore, in the next section, we parent a selection function based algorithm for utility optimization for the multiuser scheduling problem described above.

\[
A = \begin{pmatrix}
0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 1
\end{pmatrix}
\]

V. Optimizing Multi-user Scheduling

The algorithm is developed based on the fact that the single user algorithm alone does not lead into the optimum solution when generalized into the multiple users case. Obviously, the reason is that any subtask scheduled at a given resource, affects the execution time of that resource for other users. Therefore, we will develop the scheduling algorithm based on two issues: scheduling subtasks as pairs, and using a selection function that takes into account the current optimum choice for the single users scheduling problem and combine it with other facts, such as the number of unscheduled subtasks. In fact, the objective is to find the allocation matrix \( A \) that maximizes the utility \( u^A \) for a given execution time matrix \( T \) and price vector \( p \). The problem is described in the literature as NP-Complete, and hence heuristic based solutions can be used to provide the best possible solution. For \( n \) users, each has \( k_i \) subtasks, where \( 1 \leq i \leq n \), and \( m \) resources, we first create two empty matrices \( T \) and \( E \), each of \( n \times m \) elements and initialize them to zeros. We also create the two vectors \( T_{max} \), and \( E_{sum} \), each of \( n \times m \) elements, and a variable \( T_E \), all initialized to zeros as well. Let \( \Delta_i \) be a matrix of \( n \times m \) elements, initialized as follows: \( \Delta i = p_j \times T_{ij} + T_{ij} \).

Next, we define a vectors, \( A_i \) of \( m \) elements initialized to zero, represents the total number of subtasks assigned to every resource. At any time during the scheduling process, \( A_i \), \( T_{A_j} \) and \( T_E \) are updated. Finally, we define the selection function, \( \Theta_i = 0.01 \times k_i + \frac{1}{n} \), that takes into account the number of unscheduled subtasks, where \( \Delta_i \) is minimum in row \( i \) in the matrix \( \Delta \).

Subtasks are then allocated for users using the selection function, \( \Theta \), to choose two users, \( \alpha \) and \( \beta \), such that \( 1 \geq \alpha, \beta \leq n \), and the following two conditions are satisfied, (1) \( \forall i \cdot 0 > 0, i \leq n \Rightarrow \Theta_i \geq \Theta_i \), and (2) \( \forall i \cdot 0 > 0, 0 \leq n, i \neq \alpha \Rightarrow \Theta \beta \geq \Theta_i \). Then, for each user, \( \alpha \) and \( \beta \), we choose the entry that leads to best single user utility using the single user optimization algorithm, \( \gamma_i \) for task \( \alpha \) and \( \gamma_i \) for task \( \beta \), such that \( 1 \geq \gamma_i \leq k_i \), and \( 1 \leq \gamma_i \leq k_i \), and the following two conditions are satisfied: (1) \( \forall \beta \cdot \gamma_i > 0, j \leq m \Rightarrow \Delta \gamma_j \leq \Delta \alpha_j \) and (1) \( \forall \beta \cdot m > 0, j \leq n, j \neq \gamma_i \Rightarrow \Delta \beta_j \leq \Delta \beta_j \). Hence, two subtasks with minimum values in \( \Delta \) are to be scheduled for two user.

In fact, the assignment of the two subtasks are chosen simultaneously for a valid reason, when a single subtask is assigned the allocation matrix will be modified, and hence the local utility matrix \( \Delta \) will be modified. After several considerations, it was found that sleeting a single subtask and assigning it to the best available entry \( \Delta_ij \) leads to bad
**Algorithm 2** Multiple User Resource Allocation Algorithm

1: procedure ALLOCATION
2: Input: \( k, m, n, T, p \).
3: Output: \( A \).
4: Initialize:
5: Initialize vectors and variables
6: \( A = \{ \} \), \( T_{\text{max}} = \{ \} \), \( E_{\text{sum}} = \{ \} \), \( T_E = 0 \), \( T = \{ \} \); initialize all elements to 0
7: \( i = 0 \);
8: Repeat
9: \( i = 0 \);
10: Repeat
11: \( j = 0 \), \( i = i + 1 \);
12: \( \Delta_{ij} = p_j \times T_{ij} + T_{ij} \);
13: \( j = j + 1 \);
14: Until: \( j = m \);
15: Until: \( i = n \);
16: Choose two elements using selection function \( \Theta \).
17: Update all variables for assigning subtask for user \( \alpha \) assigned to service \( \gamma_\alpha \).
18: Update all variables for assigning subtask for user \( \beta \) to resource \( \gamma_\beta \).
19: Repeat above steps using \( \beta \) and \( \gamma_\beta \) instead of \( \alpha \) and \( \gamma_\alpha \).
20: Until: at most one task has unscheduled subtasks
21: Choose \( \alpha \) such that \( k_\alpha > 0 \).
22: Repeat
23: Choose \( \gamma_\alpha \) such that \( \Delta_{\gamma_\alpha} \) is minimum.
24: Update all variables for \( \alpha \) and \( \gamma_\alpha \).
25: Until: \( k_\alpha = 0 \).
26: End procedure

### VI. Illustrative Example

Let us consider the multi-user example given in Section 3, and demonstrate the algorithm above in order to find an appropriate allocation matrix. All initial values for inputs are as given in the example above. We start calculating initial values using the algorithm above, then we choose \( \alpha = 3, \beta = 2 \) using the selection function. Then we choose two entries in \( \Delta \) with minimum values: \( \gamma_\alpha = 5 \) and \( \gamma_\beta = 4 \). The combination \((\gamma_\alpha, \gamma_\beta) = (5, 4)\) is selected, hence, a subtask for user \( u_3 \) will be scheduled to service \( R_5 \), followed by a subtask of user \( u_2 \) to service \( R_4 \). Based on this, the allocation matrix, \( A \), will be updated with these schedules in the next step. This is illustrated in the initial step indicated as \( \lambda_0 \) in Table I below, where the numbers in bold represent the active ones.

In the next step, \( \lambda_1 \), \( k_3 \) and \( k_2 \) are decremented, leading to \( k = \{2, 2, 3\} \). All variables are updated and new selection function values are calculated, which will lead to \( \alpha = 3, \beta = 2, \gamma_\alpha = 2, \gamma_\beta = 3 \). Hence, subtask for user \( u_3 \) is assigned to \( R_2 \), and for user \( u_2 \) is assigned to \( R_3 \), which will appear in \( A \) in the next step \( \lambda_2 \), where \( k \) becomes \( \{2, 1, 2\} \), and the selection functions gives \( \alpha = 3, \beta = 1, \gamma_\alpha = 1, \) and \( \gamma_\beta = 5 \). Hence, subtask of \( u_3 \) is assigned to \( R_1 \), and subtask of \( u_1 \) is assigned to \( R_5 \). The step \( \lambda_3 \) leads to \( k = \{1, 1, 1\} \), \( \alpha = 3, \beta = 2, \gamma_\alpha = 2, \) and \( \gamma_\beta = 4 \). Hence, subtask of \( u_3 \) is assigned to \( R_2 \), and subtask of \( u_2 \) is assigned to \( R_4 \). In step \( \lambda_4 \), \( k = \{1, 0, 0\} \), therefore only user \( u_1 \) has subtasks, selection function for \( \alpha = 1 \) will give \( \gamma_\alpha = 3 \), hence, the last subtask for user \( u_1 \) is assigned to \( R_3 \). The final allocation matrix obtained as below, which give \( u^A = 0.134 \). Note that this utility value is better than the result obtained using game theory with Nash equilibrium in [25].

\[
A = \begin{pmatrix}
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 2 & 0 \\
1 & 2 & 0 & 0 & 1
\end{pmatrix}
\]

We observed that the selection function has direct effect on the outcome for the algorithm, therefore, we did execute the algorithm by choosing the selection function as follows \( \Theta_i = \frac{1}{1 + \Delta_{i\gamma}} + \frac{1}{1 + \Delta_{i\gamma}} \); then applying the above steps will lead into the schedule \( A \) given below, which has a utility of \( u^A = 0.1416 \), obviously, this shows an enhancement of around 6.2% over the utility in achieved in [25].

\[
A = \begin{pmatrix}
0 & 0 & 0 & 0 & 2 \\
0 & 0 & 1 & 2 & 0 \\
2 & 0 & 0 & 0 & 0
\end{pmatrix}
\]
TABLE I: Execution of the algorithm on the above example

<table>
<thead>
<tr>
<th>Step</th>
<th>$A$</th>
<th>$\Delta$</th>
<th>$k$</th>
<th>$\Theta$</th>
<th>$\alpha, \gamma_{\beta}$</th>
<th>$\beta, \gamma_{\phi}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_0$</td>
<td>0 0 0 0 0</td>
<td>12.0 11.0 10.0 9.8 9.0</td>
<td>2</td>
<td>0.131</td>
<td>3, 5</td>
<td>2, 4</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0 0</td>
<td>10.0 9.2 9.0 8.4 8.4</td>
<td>3</td>
<td>0.149</td>
<td>3, 5</td>
<td>2, 4</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0 0</td>
<td>8.0 7.7 8.0 7.8 7.2</td>
<td>4</td>
<td>0.179</td>
<td>3, 5</td>
<td>2, 4</td>
</tr>
<tr>
<td>$\lambda_1$</td>
<td>0 0 0 0 0</td>
<td>27.6 26.6 25.6 31.9 30.0</td>
<td>2</td>
<td>0.059</td>
<td>3, 2</td>
<td>2, 3</td>
</tr>
<tr>
<td></td>
<td>0 0 0 1 0</td>
<td>22.6 21.8 21.6 24.0 26.2</td>
<td>2</td>
<td>0.066</td>
<td>3, 2</td>
<td>2, 3</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0 0</td>
<td>21.2 20.9 21.2 26.8 22.8</td>
<td>3</td>
<td>0.078</td>
<td>3, 2</td>
<td>2, 3</td>
</tr>
<tr>
<td>$\lambda_2$</td>
<td>0 0 0 0 0</td>
<td>38.9 46.4 41.4 42.6 40.2</td>
<td>2</td>
<td>0.046</td>
<td>3, 1</td>
<td>1, 5</td>
</tr>
<tr>
<td></td>
<td>0 0 1 1 0</td>
<td>33.3 40.2 35.9 34.7 35.8</td>
<td>1</td>
<td>0.040</td>
<td>3, 1</td>
<td>1, 5</td>
</tr>
<tr>
<td></td>
<td>0 1 0 0 0</td>
<td>31.4 34.6 38.2 36.4 33.0</td>
<td>2</td>
<td>0.052</td>
<td>3, 1</td>
<td>1, 5</td>
</tr>
<tr>
<td>$\lambda_3$</td>
<td>0 0 0 0 0</td>
<td>59.4 56.4 55.8 53.9 55.6</td>
<td>1</td>
<td>0.028</td>
<td>3, 2</td>
<td>2, 4</td>
</tr>
<tr>
<td></td>
<td>0 0 1 1 0</td>
<td>58.8 56.2 53.2 52.0 60.0</td>
<td>1</td>
<td>0.029</td>
<td>3, 2</td>
<td>2, 4</td>
</tr>
<tr>
<td></td>
<td>1 1 0 0 0</td>
<td>51.4 50.6 54.2 52.4 54.4</td>
<td>1</td>
<td>0.030</td>
<td>3, 2</td>
<td>2, 4</td>
</tr>
<tr>
<td>$\lambda_5$</td>
<td>0 0 0 0 0</td>
<td>71.4 76.9 67.6 72.2 67.6</td>
<td>1</td>
<td>0.025</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0 0 1 2 0</td>
<td>68.4 73.5 65.0 66.8 69.6</td>
<td>0</td>
<td>0.015</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1 2 0 0 0</td>
<td>63.4 66.1 64.4 67.8 66.4</td>
<td>0</td>
<td>0.016</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$\lambda_6$</td>
<td>0 0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

VII. Conclusion and Future Work

While cloud computing technology is increasingly being used, effective resource allocation methods are required in order to maximize profit for cloud service providers, provide energy efficient methods, and at the same time achieving user satisfaction. This paper proposes a model for task-oriented resource allocation in cloud computing, where the problem of scheduling single user with multiple dependant subtask on multiple available resources with different execution time and cost is addressed first. Then, the general problem of scheduling multiple users, each with multiple subtask on multiple available resources is addressed. The proposed solution is provided by introducing a selection pairwise function based on subtasks completion time and task costs. The problem addressed in this paper is described as NP-Complete, hence, the presented solution present a local optimum schedule. The proposed method, compared to existing techniques, is applicable on scheduling problems without any restrictions on the execution time and price, in particular as compared to game theoretic approaches [25], where the execution time must be given in ascending order, and the price in descending in order to find Nash equilibrium. In addition, the proposed method outputs schedule with better utility than game theoretic one.

We intend to provide implementation for the method, and test for problems with large number of users and resources. In addition, we intend to try to modify the selection function to further improve the utility of the schedule by considering the weight of the subtasks to be scheduled in the future including price and completion time. In addition, since formal methods have been proposed thoroughly for the analysis of different cloud applications [34], we intend to use formal analysis to model and validate the proposed algorithm.

References


www_ijacsa_thesai_433_page_2016


