Resource Assignment in Computational Grid Based on Grid Market Equilibrium

Xi XIE and Satoshi FUJITA

Department of Information Engineering, Hiroshima University
Higashi-Hiroshima, 739-8527, Japan

Abstract—In this paper, we propose a resource assignment scheme in the computational grid based on the notion of market equilibrium. Market equilibrium is a key concept commonly used in the field of game theory, and in this framework, we determine the “price” of resources owned by the service providers so that it fulfills the Nash equilibrium of the given market consisting of clients and service providers. The degree of satisfaction of clients is modeled as a linear utility function of acquired resources, and as a constraint concerned with the clients, we use the notion of budgets. Our proposed scheme a semi-algorithm which finds an assignment of resources to the clients so that it maximizes the utility of the clients provided that: 1) all resources are completely exhausted and 2) the surplus of the clients is at most $\epsilon$ using $O(n \log(nM/\epsilon))$ maximum flow computations, where $M$ is the total amount of budgets given to the clients at the initial state and $\epsilon$ is a positive real representing the accuracy of approximation.

1. Introduction
1.1 Background

Our daily life is being enriched with the aid of advanced distributed systems such as cloud computing (e.g., Google Apps and Amazon EC2), social networking services (e.g., Twitter and Facebook), and on-line streaming services (e.g., YouTube and Ustream) which have a high computational availability compared with classical distributed systems with respect to the performance, reliability, usability, and the fault-tolerance. As the availability of such advanced systems increases, however, it has raised another crucial issue so that how to utilize miscellaneous resources distributed over the network in an efficient and productive manner. In fact, the resource management has been recognized as a key issue to realize an efficient utilization of fully distributed systems such as volunteer computing and peer-to-peer systems, and in the past three decades, a huge number of resource management schemes have been proposed (see [11], [3] for survey).

Computational grid, which is often called Grid, is an emerging technology to improve the utilization of shared resources in large-scale distributed systems. Grid has been used to solve many important problems in the field of business, finance, medicine, and Grand Challenge applications in many fields of science and technology. In addition, there are several national projects concerned with the development and the utilization of Grid, e.g., Fusion Collaboratory Project in United States [14], EU DataGrid in EU [15], China National Grid in China [12], NAREGI in Japan [13], and others. The reader should note that the resource management in Grid should satisfy the requirement from both of “clients” and “service providers.” More specifically, from the client side, it is required that spending the least money and getting the highest satisfaction. On the other hand, from the service provider side, it is required to make a full utilization of given resources.

There are many resource management schemes proposed in the literature, and some of them could certainly improve the efficiency of resource utilization in Grid. Agent-based scheme [7] and reputation-based scheme [1] are two representatives in this direction. A resource management based on the notion of economics, which is referred to as Grid Economy, has also received considerable attention in the past few years [16], [8]. The Grid Economy provides mechanisms to trade-off QoS parameters, deadline and computational cost, and offers an incentive for relaxing their requirements. However, most of those conventional schemes are based on a classical model of distributed systems such that resources should be assigned to clients so that a certain cost function is optimized subject to a certain constraint. In other words, it defines the problem from the viewpoint of the system manager and ignores the “emotion” of individual clients (clients) such as incentive, satisfaction, and regret.

1.2 Our Contribution

To overcome such a critical problem existing in conventional schemes, in this paper, we adopt the notion of market equilibrium to satisfy requirements issued by clients and service providers. Market equilibrium is a key concept commonly used in the field of game theory. In this framework, we will determine the “price” of each resource owned by the service providers so that it fulfills the Nash equilibrium of the given market consisting of clients and service providers. More concretely, we assume the existence of a centralized manager, and instead of determining the price of resources using an autonomous (but one-sided) mechanism such as buyer initiated or seller initiated auction,
the price of resources will be determined by the centralized manager so that:

1) the satisfaction of each client is maximized, and
2) given resources are fully utilized.

The degree of satisfaction is modeled as a linear utility function which is independently defined for each client, and as a constraint concerned with the clients, we use the notion of budget in the price determination process.

As a concrete method to calculate such a market equilibrium, we focus on a polynomial time algorithm proposed by Devanur et al. in 2002 [9]. This algorithm calculates a market equilibrium using \( O(n^4 (\log n + \log U + \log M) \) max-flow (maximum flow) computations, where \( n \) is the number of service providers, \( U \) is the maximum coefficient in the utility functions and \( M \) is the total money possessed by the clients at the initial state (the details of the algorithm will be described in Section 4). Thus, although it is in fact polynomial time and always outputs an exact solution, it rapidly grows as the size of the given instance increases. In this paper, we reduce the time complexity of Devanur et al.’s algorithm using the notion of approximation. More precisely, our proposed scheme is a semi-algorithm which calculates the price of resources maximizing the utility of clients provided that: 1) all resources are completely exhausted and 2) the surplus of the clients is at most \( \epsilon \) for any \( \epsilon > 0 \), using \( O(n \log(nM/\epsilon) \) max-flow computations. The reader should note that our scheme solves (slightly) different problem from the original problem in the sense that we allow the remaining of very small surplus (of at most \( \epsilon \)) whereas such surplus is strictly prohibited in the original problem.

The performance of the proposed scheme is experimentally evaluated with respect to the following factors: 1) the percentage of instances successfully solved by the proposed scheme (recall that our scheme is not an algorithm but a semi-algorithm), 2) comparison of the execution time with the Devanur et al.’s algorithm, and 3) certification on the approximation, i.e., whether or not the surplus in the resultant solution is certainly smaller than given \( \epsilon \).

The reminder of this paper is organized as follows. Section 2 overviews related work. Section 3 defines a service model of Grid adopted in this paper. Section 4 outlines the primal-dual algorithm proposed by Devanur et al. Section 5 describes our proposed scheme. Section 6 shows the result of experiments. Finally, Section 7 concludes the paper with topics for future work.

2. Related Work

This section overviews several resource management schemes proposed in the literature which can be applied to the resource management in the Grid.

2.1 Economic Model

In order to encourage the incentive of the clients, Buyya et al. introduced the notion of economy to the grid environment.
since under the auction mechanism, each client should pay much money to get a good resource although there may exist resources that are less attractive but with a reasonable price.

2.2 Reputation Model

Alunkal et al. introduced the notion of reputation to the resource management in Grid [1]. More concretely, their resource management scheme adopts the concepts of dynamic trust and reputation adaptation, which are based on the community experiences such as the classification, selection, and the tuning of the allocation of entities such as resources and services. They proposed a sophisticated architecture to select trusted resources which best satisfies the application requirements with respect to a predefined trust metric.

The overview of an individual reputation service is shown in Figure 2. It consists of a calculation manager, collection manager, storage manager and reputation reporter. The calculation manager is responsible for calculating the reputation value of the entities according to a prespecified context and providing the calculated values to the storage manager which stores them to maintain a global and historical view. The collection manager evaluates the quality statement, describes the requested reputation, and collects relevant data from the entities. The reporter contacts the storage manager to make a report whenever it is requested by the other clients in the Grid.

With the aid of reputation, Alunkal et al.’s model can reduce the influence of malicious entities, while it still ignores the satisfaction of clients. In addition, it faces to the same problem with the economic model such that the given resources could not be fully utilized in many cases.

2.3 Broker Model

Elmroth proposed a resource management scheme based on the notion of decentralized brokers [10]. The role of a broker is to select a resource which minimizes the completion time of a given job, considering the time for file staging, batch queue waiting, and the job execution. A grid architecture based on decentralized brokers is shown in Figure 3. In this architecture, each resource registers itself to at least one index server, where each index server can also register itself to index servers at a higher level; i.e., it forms a hierarchy of index servers. All clients can access resources only through their own broker, where each broker contacts index servers to discover what resources are available in the system. After acquiring the name and the contact of the resources from index servers, a broker requests individual resources for their detailed information, and performs job submission and job control by directly communicating with the resources.

3. Service Model

3.1 Model

The objective of resource management considered in this paper is described as follows. Consider a distributed system consisting of a set of resources $A$, a set of clients $B$, and a central manager $C$. Each resource in $A$ models a particular service provided by the grid computer, e.g., disk space, CPU power, peripheral devices, and so on. Let $n = |A|$ denote the number of resources and $n' = |B|$ denote the number of clients. For each resource $j \in A$, the price $p_j$ of the resource (per unit) is determined by the manager. Each client $i \in B$ is given budget $e_i$, and client $i$ can request several kind of resources to maximize her profit, provided that the total expense does not exceed $e_i$.

The main role of central manager $C$ is to manage the assignment of resources to the clients so that all resources are completely assigned to the clients while maximizing the profit of each client. In this system, such an assignment is realized by using a market model. More concretely, each
client selfishly tries to maximize its profit, and the total cost of requested resources does not exceed her budget. In order to “clear” such a market (i.e., in order to realize a situation in which all resources are assigned to clients and all clients exhaust their budget), central manager $C$ can “control” the price of each resource. That is, by increasing the price of a resource, it becomes less attractive for all clients and it relaxes the congestion of requests to the resource, and by decreasing the price of a resource, we could attained a full utilization of the resource since it becomes attractive for all clients interested in the resource.

The objective of the central manager is to find a market clearing price $\tilde{p} = (p_1, \ldots, p_n)$, after each client is assigned an optimal set of resources relative to these prices, there is no surplus or deficiency of any of the resources. The structure of the central manager is illustrated in Figure 4. As shown in the figure, it consists of four components, i.e. task manager, utility manager, resource manager and the allocation manager, where the most important part is the allocation manager which is responsible for computing and allocating resources to the clients. Task manager is responsible for receiving tasks from the clients and utility manager is responsible for receiving the utility function from the clients. Finally, the role of resource manager is to manage the resources coming to or leaving from the system.

### 3.2 Utility Function

As was described previously, we model the satisfaction of clients by a collection of utility functions. In this paper, we assume that the utility functions are linear with respect to the amount of assigned resources. Let $x_{i,j}$ be the amount of resource $j$ assigned to client $i$ under price vector $\tilde{p}$. Then, the utility of client $i$ with respect to the assignment is represented by

$$\sum_{j=1}^{n} u_{i,j} \times x_{i,j}$$

using $n$ constants $u_{i,1}, \ldots, u_{i,n}$. The reader should note that such an assumption on the linearity of utility functions makes the analysis of algorithms much easier than the case of other utility functions such as concave ones, although it is less practical than those general ones. In fact, in actual situations in the real world, it is unlikely that the degree of satisfaction “linearly” increases as increasing the amount of acquired resources, e.g., although a kid could increase her satisfaction twice as increasing the number of candies from one to two, her satisfaction does not increase to the twice by increasing the number of candies from one hundred to two hundreds.

### 3.3 Match Making in Grid

A typical scenario for the match making in Grid proceeds as follows. At first, all clients register their utility function to $C$. Tasks issued by the clients are managed by $C$ in a synchronous manner i.e., $C$ repeats synchronous rounds in each of which:

1) tasks issued by the clients are received at the beginning of a round,
2) tasks are assigned resources, and
3) assigned resources are used by the tasks until the beginning of the next round.

As was mentioned previously, how to calculate an appropriate assignment is the main concern in this paper.

### 4. Original Algorithm

#### 4.1 Graph $N(\tilde{p})$

The following price determination algorithm is borrowed from [9]. The goal of this algorithm is to find a price vector $\tilde{p}^*$ such that both resources and budgets are exhausted under $\tilde{p}^*$. Since the budget is limited and we are assuming linear utility functions, for any price vector $\tilde{p}$, a reasonable client wishing to maximize her profit must spend her money to the resources which maximize her profit per price, i.e., $u_{i,j}/p_j$. In other words, $\alpha_i = \max_j \{u_{i,j}/p_j\}$ is considered to be the bang per buck for client $i$. Note that client $i$ is interested in any resource $j$ such that $u_{i,j}/p_j = \alpha_i$, and if there are several such resources, she is equally happy with any combination of such resources.

To represent such a preference of clients (under price vector $\tilde{p}$), we use a directed capacitated graph $N(\tilde{p}) = (V, E)$, which is defined as follows:

- $V = A \cup B \cup \{s, t\}$, where $A$ is the set of resources, $B$ is the set of clients, and $s$ and $t$ are vertices which are not contained in $A \cup B$.
- For any $j \in A$ and $i \in B$, $(i, j) \in E$, $(s, j) \in E$, and $(i, t) \in E$.
- Edge $(i, j)$ connecting vertices in $A \cup B$ is given infinite capacity, edge $(s, j)$ is given capacity $p_j$, and edge $(i, t)$ is given capacity $e_i$.

Note that $p_j$ is the price of resource $j$ and $e_i$ is the budget given to client $i$.

By definition, any combination of resources allocated along the edges of $N(\tilde{p})$ will make clients happiest under price vector $\tilde{p}$. Computing the largest amount of resources which can be allocated in this manner, without exceeding the budget of clients or the amount of resources available, can be accomplished by computing a max-flow in $N(\tilde{p})$.

With graph $N(\tilde{p})$, the algorithm tries to increase the price of each resource starting from an initial price vector, by keeping an invariant such that

“$(\{s\}, A \cup B \cup \{t\})$ is a min-cut of graph $N(p)$.”

See Figure 5 for illustration. This invariant ensures that all resources can be allocated to some clients (i.e., exhausted).
and an increase of the price vector monotonically decreases the surplus money of each client. In other words, it guarantees that when the surplus vanishes, a market clearing price vector which exhausts all resources and all budgets, is obtained.

4.2 Tight Sets

The second idea of the algorithm is to identify a set of resources whose price is to be “frozen” during the succeeding steps of the algorithm. Such a set of resources is called a tight set, which is formally defined as follows: For a set \( S \subseteq A \) of resources, let \( p(S) \) denote the total expense of resources in \( S \), and for a set \( T \subseteq B \) of clients, let \( m(T) \) denote the total money possessed by the clients in \( T \). Let \( \Gamma(S) \) denote the set of clients adjacent to \( S \); i.e., \( \Gamma(S) \) is the set of clients who are interested in a resource in \( S \) at the current price vector \( \vec{p} \). A resource set \( S \) is said to be tight if it holds \( p(S) = m(\Gamma(S)) \), i.e., if the expense of \( S \) exactly equals to the money possessed by the clients interested in \( S \). The reader should note that clients in \( \Gamma(S) \) are completely satisfied with the current prices of \( S \), and such a satisfaction does not change even if we increase the price of the other unfrozen resources.

In the algorithm, the price of some unfrozen resources monotonically increases, where instead of increasing the price of all resources at the same time, we focus on a subgraph of \( N(\vec{p}) \) called active graph, and try to increase the price of resources in the graph (detailed definition of the active graph will be given later). The price of resources in the active graph is conducted in such a way that it ensures that the edges in the active graph are retained during the increasing process (i.e., it does not become “too high”). To this end, we (uniformly) multiply the price of these resources by \( x \) and gradually increase \( x \). It is shown in [9] that we will have a tight set when \( x \) is set to the following value:

\[
x^* = \min_{S \subseteq A} \frac{m(\Gamma(S))}{m(S)}.
\]

Here value \( x^* \) and the corresponding tight set \( S^* \) can be found in polynomial time using \( n \) max-flow computations.

4.3 Balanced Flows

Given a flow \( f \) in network \( N(\vec{p}) \), the surplus of client \( i \) denoted by \( \gamma_i(\vec{p}, f) \), is the residual capacity of edge \((i, t)\) with respect to \( f \), which equals to \( e_i \) minus the flow sent through \((i, t)\). Let \( \gamma(\vec{p}, f) := (\gamma_1(\vec{p}, f), \gamma_2(\vec{p}, f), \ldots, \gamma_n(\vec{p}, f)) \) denote the surplus vector of clients with respect to \( \vec{p} \) and \( f \). In order to accelerate the increase of \( \vec{p} \), the algorithm focuses on resources adjacent to “high-surplus” clients, since it would increase the chance of significantly increasing the price while keeping the invariant. However, the surplus of clients may take different values for different maximum flows even in the same graph. Thus, in order to well-define the surplus of clients, the algorithm defines a kind of canonical flow defined as follows: Given a price vector \( p \), a maximum flow that minimizes \( ||\gamma(\vec{p}, f)|| \) over all choices of \( f \) is called a balanced flow, where \( ||\vec{v}|| \) denotes the \( l_2 \) norm of vector \( \vec{v} \). If \( ||\gamma(\vec{p}, f)|| < ||\gamma(\vec{p}, f')|| \), then we say \( f \) is more balanced than \( f' \), and for a given \( \vec{p} \) and a flow \( f \) in \( N(\vec{p}) \), we denote the residual network of \( N(\vec{p}) \) with respect to \( f \) by \( R(\vec{p}, f) \).

4.4 Main Algorithm

The main idea of the algorithm is to reduce \( ||\gamma(\vec{p}, f)|| \) in every phase. This goal is achieved by finding a set of high-surplus clients in the balanced flow and by increasing the price of resources interested by the clients, i.e., resources adjacent to the high-surplus clients in \( N(\vec{p}) \). If a subset of resources becomes tight after such an increase, then we have reduced \( ||\gamma(\vec{p}, f)|| \) since the surplus of high-surplus clients is now dropped to zero. Another possibility is that a new edge is added to \( N(\vec{p}) \), but such an edge will also help us to make the surplus vector more balanced.

In each phase, it identifies a subgraph of \( N(\vec{p}) \) induced by a set of clients \( H \subseteq B \) and a set of resources \( H' \subseteq A \) where initially, \( H \) is the set of clients whose surplus equals to the maximum surplus \( \delta \) in \( B \) and \( H' \) is the set of resources adjacent to \( H \). Such a subgraph is called active graph. As was previously described, the price of resources in the active graph increases in such a way that all edges in the graph retained, which can be ensured by multiplying the price of resources in the subgraph by \( x \), and by gradually increasing \( x \). Each phase is divided into several iterations. In each iteration, \( x \) increases until either a new edge is added to \( N(\vec{p}) \) or a subset of resources becomes tight. In the former case, we recompute the balanced flow \( f \), and add all vertices which can reach a member of \( H \) in \( R(\vec{p}, f) \setminus \{s, t\} \) to subset \( H \). If a subset becomes tight as a result of increase, the iteration terminates. If \( A \) becomes tight, the algorithm terminates.

Finally, the initial price vector satisfying the invariant is determined as follows [9]:

- Fix the price of each resource to \( 1/n \).
- If there exists \( j \in A \) such that \( \Gamma(\{j\}) = \emptyset \) (i.e., if no client is interested in resource \( j \) due to high price), then
reduce the price of $j$ to $\max_i \left( \frac{b_i + \varepsilon}{m_i} \right)$, and repeat such a modification until no such resource exists.

5. Proposed Scheme

In this section, we propose a scheme to find a market-clearing price vector in an approximated manner. The time complexity of the proposed (semi-)algorithm is much lower than the original (exact) algorithm described in Section 4, although it merely generates an approximated solution to the problem in the sense that the amount of remaining budget is at most $\varepsilon$ for any $\varepsilon > 0$. Similar to the previous section, in the following explanation, we will continue to use graph $N(\bar{p})$ to represent the relation between the set of resources $A$ and the set of clients $B$ under the given price vector $\bar{p}$, while there are two additional vertices $s$ and $t$, as before. The reader should remind that in graph $N(\bar{p})$, client $i$ is connected with resource $j$ by an edge if and only if $j$ is a resource which maximizes $u_{ij}/p_j$.

5.1 Algorithm

Similar to the original algorithm, the proposed algorithm starts with finding a price vector which does not violate the invariant, which is ensured by making the sum of prices is less than or equal to the surplus of clients. The algorithm is divided into phases. In each phase, it identifies an active graph consisting of a set of clients $H \subset B$ and a set of resources $H' \subset A$, and increases the price of resources contained in $H'$, i.e., the price of resources not in $H'$ is not increased. Such an increase of the price is conducted in such a way that the edges in the active graph are retained, which is ensured by multiplying the price of all resources in $H'$ by $x$ and by gradually increasing $x$ from 1, as before. Let $\delta$ be the maximum surplus in $B$ (under current price vector $\bar{p}$). Initially, active graph is constructed such that $H$ is a set of clients whose surplus equals to $\delta$ and $H'$ is the set of resources adjacent to $H$. Let $p(H')$ be the sum of prices of resources in $H'$ and $m(H)$ be the total money possessed by the clients in $H$. In order to accelerate the increase of parameter $x$, in the proposed algorithm, we repeat to “double” the value of $x$ to find a sufficiently large value, and then conduct a binary search to find a value satisfying the given approximation ratio.

More concretely, in the first step, we double the value of $x$ until one of the following conditions holds:

Case 1: $m(H) - \varepsilon/n < p(H') \leq m(H)$,
Case 2: $p(H') > m(H)$, or
Case 3: $(i, j)$ with $i \in H$ and $j \in A \setminus H'$ becomes an edge in the active graph.

In Case 1, it terminates the phase by moving all resources in $H'$ to the frozen set. It then recomputes the balanced flow to obtain a new active graph for the next phase. If we met Case 2 for the first time, on the other hand, it starts a binary search to find $x$ satisfying Case 1. For example, if $x = 128$, the value of $x$ is updated as $(128 + 64)/2 = 96$. If the updated value still meets Case 2, the value further reduces to $(64 + 96)/2 = 80$ while it increases to $(128 + 96)/2 = 112$ if it meets neither of Cases 1 and 2 at $x = 96$. Such an update-and-check is repeated until it meets Case 1.

Finally, in Case 3, it tries to find the largest $x$ using another binary search such that all resources existing in the last active graph are still being members of the new active graph, i.e., every resource in the active graph consisting of old and new resources is connected to the clients in the active graph so that the amount of budgets of those clients is at least the current price of the resource (such a condition is necessary to guarantee that all resources are exhausted). If it could not identify such $x$, it terminates the algorithm without outputting a solution. However, if there is such $x$, after adding new resources and all clients interested in the set of resources to the active graph, and starts the procedure from the beginning of the phase.

5.2 Analysis

The correctness and the time complexity of the algorithm are proved in the following claims.

Theorem 1: If the proposed algorithm outputs a price vector $\bar{p}^*$ as the solution, it guarantees that under $\bar{p}^*$ all resources are exhausted and the surplus of the users is at most $\varepsilon$.

Proof: By the description of the algorithm, when resources in set $H'$ is moved to the frozen set, the surplus in $H$, which is the set of clients interested in $H'$, is at most $\varepsilon/n$. Set $H'$ is not empty even in the worst case. In addition, each user is interested in at least one resource at the initial price and all resources in $A$ should be eventually moved to the frozen set if it outputs a solution. Thus, we can conclude that the surplus of users in $B$ is at most $\varepsilon$ after the termination of the algorithm.

Let $M$ denote the total budget initially given to the clients in $B$, i.e., $M = m(B)$. Let $k$ be the number of iterations spent to meet Case 1 in a phase. An upper bound on $k$ could be given as in the following lemma.

Lemma 1: $k = O(\log \frac{2M}{\varepsilon})$.

Proof: Since the doubling of $x$ is repeated at most $\lceil \log_2 M \rceil$ times, it takes $O(\log M)$ time to identify an upper bound on the target value. After identifying the upper bound, it takes $O(\log M)$ time to identify a range of size one including the target value since the upper bound is at most $M$, and it needs $O(\log(n/\varepsilon))$ additional time to identify the target within a range of size at most $\varepsilon/n$. Thus the lemma follows.

The time complexity of the proposed scheme is proved in the following theorem.

Theorem 2: The proposed scheme executes at most $O(n \log \frac{2M}{\varepsilon})$ max-flow computations.

Proof: Since each phase moves at least one resource to the frozen set, the algorithm repeats the phase at most $n$
times. In each phase, it executes the max-flow computation in one of the following three cases: 1) at the beginning of the phase to identify an initial active graph, 2) after increasing the value of $x$ to check which of Cases 1, 2 and 3 is met, and 3) after encountering Case 3 to identify an updated active graph. The third case occurs at most $n$ times during the execution of the algorithm, since once a resource is included in an active graph in a phase, it is moved to the frozen set at the end of the phase or the algorithm gives up to output a solution. By Lemma 1, the increase of $x$ is repeated $O(\log (nM/\epsilon))$ times in each phase. Thus the total number of executions of max-flow computation is given by $O(n \log (nM/\epsilon))$. Hence the theorem follows.

6. Experiments

6.1 Setup

We experimentally evaluated the performance of the proposed scheme using GridSim environment [5]. In the simulation, we consider a Grid consisting of 100 resources and 100 users (i.e., $n = n' = 100$). The budget of each user is 100 dollars, so that the total budget is 10000 dollars, and the initial price of each resource is randomly selected from 10 to 50 dollars. We use MIPS (Million Instructions Per Second) value to represent the characteristic of resources. The MIPS value of each resource is randomly selected from range $[300, 500]$ and the MIPS value requested by each user is randomly selected from range $[200, 400]$. The value of $\epsilon$ is varied as 50, 100, 150, 200 and 250, and for each value of $\epsilon$, we conducted 20 runs. Finally, as the concrete max-flow algorithm, we used the Edmonds-Karp algorithm [17].

6.2 Results

The result of simulations is summarized as follows. At first, we evaluated the ratio of failed instances for each $\epsilon$. The result is shown in Figure 6. The ratio gradually decreases as $\epsilon$ increases, and in total, it successfully outputs an approximated solution for 53% of the examined instances (even if $\epsilon$ is set to 0.5% of the given budget, it successfully outputs an approximated solution for 40% of the instances). As for the average execution time, we found that for any $\epsilon$, it takes less than 3 sec for succeeded instances whereas it takes less than 0.5 sec for failed instances. Since the original exact algorithm proposed by Devanur et al. takes more than 5 min for the same instances, we can conclude that the proposed scheme effectively gives an approximated solution provided that it is executed as a preprocessor of Devanur et al.’s algorithm.

Parameter $\epsilon$ represents the maximal surplus at the end of succeeded computation. Thus finally, we evaluated the average surplus for the succeeded instances. Figure 7 shows the results. From the figure, we can observe that the average surplus is much less than $\epsilon$, and specifically, we could attain an average surplus of 120 when we set $\epsilon$ to 250.

7. Concluding Remarks

In this paper, we propose a resource assignment scheme for computational grid based on the notion of market equilibrium. The proposed scheme is a semi-algorithm which outputs an approximated solution by using $O(n \log (nM/\epsilon))$ max-flow computations if it successfully terminates.

There are several issues to be addressed as the future work. The first issue is to increase the success ratio of the proposed scheme. To this end, we need to refine the scheme so that the change of the configuration of the active graph is appropriately managed. The second issue is to reduce the execution time. Although it could certainly reduce the number of max-flow computations in the original algorithm, it is still too large since the max-flow computation is an expensive task.

Acknowledgements

This work was supported in part by the Scientific Grant-in-Aid from Ministry of Education, Science, Sports and Culture of Japan and the Telecommunications Advancement Foundation.
References


