Future Generation Computer Systems 55 (2016) 62–73

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Service level agreement based adaptive Grid superscheduling

Mohammad Hasanzadeh Mofrad [a,b], Omid Jalilian [c], Alireza Rezvanian [b,d,*], Mohammad Reza Meybodi [b]

[a] *Department of Computer Science, University of Pittsburgh, Pittsburgh, USA*

[b] *Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran*

[c] *Department of Engineering, Payame Noor University (PNU), Tehran, Iran*

[d] *Department of Computer Engineering, Hamedan Branch, Islamic Azad University, Hamedan, Iran*

HIGHLIGHTS

- The legacy architecture of superscheduling with monitoring and discovery system.
- A service level agreement based on adaptive superscheduling (SAS) algorithm.
- Improving the chance of successful execution of users' job on Grid resources.
- Balancing revenue statistics and user quality of service constraints.
- High hit ratio and superb resource utilization of proposed algorithm in simulation.

ARTICLE INFO

Article history:
Received 2 May 2015
Received in revised form
3 August 2015
Accepted 21 August 2015
Available online 9 September 2015

Keywords:
Grid job scheduling
Request forwarding
Service level agreement
Success rate strategy
Service pricing strategy
Quality of service

ABSTRACT

Grid computing brings heterogeneity and decentralization to the world of science and technology. It leverages every bit of idle computing resources and provides a straightforward middleware for integrating cross-domain scientific devices and legacy systems. In a super big Grid, job scheduling is challenging specifically when it needs to have access to vast amount of resources. The process of mapping jobs onto Grid resources requires significant consideration in terms of Grid architecture design, consumer demands and provider revenues. In this paper, we simultaneously utilize the legacy architecture of superscheduling, forwarding strategy, service level, success rate, and service pricing strategies and finally propose a service level agreement based on adaptive superscheduling (SAS) algorithm. SAS algorithm presents unified connectivity via efficient diffusion of jobs through the Grid infrastructure that is fueled from the previous scheduling events across the Grid. Moreover, by enforcing the service level agreement terms from a rich set of ask and bid prices, system performance, and load statistics, SAS successfully boosts revenue and utilization statistics. We perform an extensive experimental analysis for different Grid scales. Based on our experimental result, the SAS algorithm maximizes revenue while guarantees quality of service. More specifically, the quality of service is achieved through a high ratio of completed jobs and remarkable utilization of resources.

1. Introduction

A Grid system encompasses a set of computing, storage and network resources. The loosely coupled interconnections of these resources construct the underpinning fabric layer of Grid. This fabric layer needs a mediator to assist Grid systems for solving complex real-world problems by using parallel processing techniques on scalable, interoperable and dynamically configurable fabric resources. Moreover, as computing technology advances, the number and types of these fabric resources are grown exponentially within the Grid. Thus, providing a reliable middleware for scheduling, managing and monitoring fabric resources is a critical challenge for Grid systems to overcome.

Grid computing is the abstract framework of a set of diverse resources and services under different administrative domains.

* Corresponding author at: Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Tehran, Iran. Tel.: +98 2164545120; fax: +98 2166495521.
E-mail address: a.rezvanian@aut.ac.ir (A. Rezvanian).

http://dx.doi.org/10.1016/j.future.2015.08.012
0167-739X/© 2015 Elsevier B.V. All rights reserved.

These resources and services are connected through the Internet and they seek to provide an agile production environment for researchers to harness the outburst of Grid resources across different systems of governance. The variety of resources available to Grid, leverages its resource sharing role. A Grid middleware provides an infrastructure for managing and loading the required resources, data and devices for users. Moreover, it could provide the user with a hosted superstructure that is accessible via web browsers for improving the user's experience. There are several Information Technology issues in planning a grid architecture such as: (1) Solving the Adaptability, extensibility and scalability of computer networks, (2) Employing different management policies, (3) Allocating geographically distributed shared resources, (4) Enforcing Service Level Agreements (SLAs), e.g., cost and profit criteria and (5) Providing the Quality of Service (QoS).

The Grid scheduling is a composite problem that outlines the essence of leveraging elastic resources of Grid's Virtual Organizations (VOs) [1]. A set of geographically distributed resources that follows predefined policies of different administrative domains is nominated as VO. The ultimate goal of Grid scheduling heuristics is to enable efficient and cost effective execution of scientific problems on a plethora of wide-ranging computational resources across different VOs.

The Grid superscheduling [2] is defined as scheduling jobs on different Grid resources such as computational clusters, parallel supercomputers and desktop computers that belong to different administrative domains. In superscheduling, the Grid Federation Agent (GFA) [2] and Local Resource Management System (LRMS) [3] are respectively used to enable resource sharing and resource management between these different administrative domains. A superscheduler may enable the following additional functionalities: (1) probing the Grid Resource Information System (GRIS) [4] for requested resources, (2) negotiation and regulation of SLA terms and (3) scheduling jobs across Grid.

The SLA based Adaptive Superscheduling (SAS) utilizes an economic framework for scheduling and allocating resources. This framework ensures Grid to reach an optimal equilibrium in demand and supply of resources. This equilibrium dynamically regulates resource prices and acts as the cornerstone of Grid. The emulated Grid follows a decentralized and distributed model that utilizes both GFA and LRMS components for resource brokering and resource handling. The GFA component negotiates the SLA with LRMS component and end user while the LRMS component manages the local resources of the Grid node.

The GFA component of SAS algorithm embodies a *request forwarding strategy* that is designed for dispersing users' jobs over Grid network. The request forwarding strategy utilizes global feedbacks of success rate strategy so that it rapidly creates optimal paths for various job requests and exposes different Grid resources to Grid users.

In addition to request forwarding strategy, the GFA component of SAS algorithm incorporates a SLA into the delivered services to the customers. A SLA is a contract between resource providers and consumers that defines both quality and quantity of a specific service. The proposed *SLA management strategy* sets out the levels of QoS such as service delivery time and expected budget between Grid node and end user. It guarantees that the task executes within the approximated service delivery time and presents the following advantages: (1) controlling the resource allocation process, (2) preventing to submit jobs that the Grid is not capable of running them, (3) assuring to reach optimal QoS levels and (4) reducing the resource queue waiting time.

The LRMS component of SAS algorithm employs new time constraints for harnessing the capacity of Grid. The *job scheduling strategy* tries to minimize the negotiation overhead by applying simple scheduling operations on the integrated sets of Grid resources. Thus, this strategy can assign tasks to proper resources rapidly.

Also, LRMS component utilizes the *success rate strategy* [5] that for different Grid resources calculates the probabilities of successful execution of jobs. The SAS algorithm frequently imports these probabilities to request forwarding strategy for unlocking the Grid assets and delivering agility over these loosely coupled assets.

Furthermore, generating new streams of revenue and promoting the dealing process of resources is not possible unless fair pricing policies are applied to the Grid ecosystem. The *service pricing strategy* of LRMS component of SAS algorithm allows providers to adapt their bid prices with respect to the load of their on-premises resources. This strategy enhances the purchasing power of consumers by adopting their ask prices.

This paper is organized as follows: the related work of Grid scheduling is presented in Section 2. Section 3 presents the idea of SLA based adaptive Grid superscheduling. The simulation results are reported in Section 4. Finally, some conclusions are drawn in Section 5.

2. Related work

In general, a job scheduling algorithm tries to solve a large and distributed optimization problem. Thus, job scheduling is more a NP-complete problem than a mere search and assign one. There are several researches that have investigated Grid job scheduling. This section provides a review of these studies and positions the proposed algorithm in comparison with them.

2.1. Data Parallel Applications scheduling

Parallel applications need data partitioning for resource management and selection. In [6], a Scheduler for Data Parallel Applications (SDPA) is introduced that enables efficient task partitioning and resource load balancing. This algorithm maintains job execution priorities while cutting the idle time of Grid nodes. Since this algorithm divides the jobs into smaller fragments, jobs will not wait even if the total number of computational resources is not sufficient for the complete execution of them.

2.2. Bacterial Foraging scheduling

Bacterial Foraging Optimization (BFO) is a global optimization algorithm for distributed optimization and control. In [7], a novel Bacterial Foraging Optimization based Hyper Heuristic (BFOHH) resource scheduling algorithm has been designed to effectively schedule jobs on available resources in a Grid environment. This algorithm utilizes a Resource Provisioning (RP) unit for verification of QoS parameters and offering a candidate set of resources. Also, the BFOHH algorithm makes the final decision about executing jobs on resources with regard to cost and makespan of application's execution.

2.3. Cost-Greedy Price-Adjusting scheduling

In [8], an incentive-based scheduling algorithm is proposed that promotes users satisfaction and providers profit. The Cost-Greedy Price-Adjusting (CGPA) algorithm sets the jobs with few candidate resources as high priority and maps them to the cheapest available resources. Also, a price-adjusting algorithm is used for adjusting the price of candidate resources to maximize provider's profit fairness.

2.4. Group-based Parallel Multischeduling

A Group-based Parallel Multi-Scheduler (GPMS) is presented in [9]. By means of different jobs and machines grouping methods, this algorithm splits jobs and machines into paired groups for independently scheduling of jobs on different machines. These grouping methods suit varying characteristics of incoming jobs. There are two methods in splitting jobs into groups: (1) Execution Time Balanced (ETB) that uses an estimation of the processing time for each job and (2) Execution Time Sorted and Balanced (ETSB) that is similar to ETB but the jobs are sorted by their size. Also, there are two methods in splitting machines: (1) Evenly Distributed (EvenDist) that evenly divides the machines into groups and (2) Similar Together (SimTog) that assembles machines based on their performance characteristics.

2.5. Preemption-aware metascheduling

External and internal users' requests with various QoS requirements bring the challenge of resource provisioning to Grid environments. In high demand federated Grids, the resource provisioning is performed via Virtual Machines (VMs) preemption. Therefore, in [10] a set of algorithms are proposed to decrease the number of VM preemptions in the emulated Grid environment. The Prediction-aware workload Allocation Policy (PAP) determines the heavily loaded resource clusters and eliminates them from the set of available resources by decreasing their routing probabilities. Furthermore, the Request Type Dispatch Policy (RTDP) algorithm reduces the number of VM preemptions for important requests while building a deterministic sequence for requests dispatching. The proposed algorithm are employed in InterGrid which is a federated Grid environment for connecting islands of virtualized Grids.

2.6. Double Auction Metascheduling

A metascheduler maps the incoming jobs to the computational resources of Grid where each of them has its own local scheduler. Unlike system-centric scheduling metrics, in [11], a user-centric metascheduling approach is proposed. This algorithm takes advantages of both auctions and system-based schedulers to satisfy the user' requests while maintaining the resource utilization across the Grid. The Double Auction-inspired Metascheduling (DAM) composes of three steps: (1) in *collection* step the resources and jobs information such as queue time and waiting time are collected, (2) in *valuation* step the corresponding values of collected information are calculated by metascheduler and (3) in *matching* step the assignment of jobs to resources are done based on the retrieved information of previous step. Also, in DAM sellers and buyers submit their offers through Call Auction (CA) procedure. Finally, the emulated Grid inspired from the European Data Grid (EDG).

2.7. Community-aware scheduling

The term metascheduling refers to the orchestration of local resources of independent network nodes through the Grid or Cloud. The Community Aware Scheduling Algorithm (CASA) [12] comprises of 5 heuristics including: (1) *job distribution* for submitting a job, (2) *job delegation request acceptance* for checking the job requirements, (3) *job assignments* for sending the job to a remote node, (4) *job rescheduling* for selecting the current optimal choice for the job and (5) *job rescheduling request acceptance* for ensuring the best offer for the current job. The experimental results of this algorithm show that CASA improves the average job slowdown and average job waiting time and message overhead.

2.8. QoS-aware metascheduling

An integrated multi QoS support job scheduling algorithm is proposed in [13]. This algorithm orchestrates various scheduling and reservation in advance methods for job scheduling. It uses rescheduling techniques to improve the Grid's throughput and maximize SLA acceptance. The SLA template of this algorithm follows the Web Services-Agreement (WS-Agreement) specification which defines the structure and negotiation protocol. Moreover, this paper presents a Reservations in Advance (RA) algorithm that is divided into two steps: (1) selection of resources and (2) reservation of resources. Also in the first step of RA algorithm, a Meta-Scheduling in Advance (MSA) algorithm is used for controlling the future execution of jobs on resources. Finally, a rescheduling algorithm is used to reorganize the schedule by performing multiple scheduling and rescheduling processes and migrating jobs between time and resources.

2.9. Resource discovery based on Distributed Learning Automata

Grid resource discovery is the problem of locating and retrieving the computational resources within the Grid. Finding the feasible resources for job queries needs an adaptive and intelligent algorithm. In [14], a resource discovery algorithm is proposed that uses Distributed Learning Automata (DLA) [15–18] which is a network of Learning Automata [19]. In this algorithm, multiple DLA are used to forward domain specific queries through Grid nodes and each node utilizes the First Come First Served (FCFS) strategy as the local scheduling algorithm. Experimental results have shown that RDDLA is the best choice for small scale Grids.

2.10. Distributed optimization grid resource discovery

To find suitable resources for user's jobs, resource discovery is a vital prerequisite of job scheduling. In [20], a DLA based multi swarm discrete Particle Swarm Optimization (PSO) [21] approach is developed. In Distributed Optimization Grid (DOG) algorithm, swarms of particles are assigned to different resource metrics while groups of DLA are the control unit of each swarm. This algorithm presents the following set of concepts: (1) Following the design principles of Monitoring and Discovery System (MDS) architecture [22] that is the information service component of Globus Toolkit, (2) combining centralized local resource management with decentralized global request forwarding, (3) employing an adaptive resource discovery approach, (4) utilizing an scalable Grid overlay network and (5) adopting the FCFS scheduling policy as the local resource scheduler.

2.11. SLA based coordinated superscheduling

In [2], a market based SLA coordination mechanism based on the Contract Net Protocol (CNP) is proposed. The SLA based CNP Superscheduling (SCS) is facilitated by GFA (Grid Scheduler) and LRMS entities. The aforementioned model is focused on the decentralized commodity market model for setting resource prices based on demand, supply and value. Also, this SLA model regulates the balance of supply and demand of resources in terms of offering motivation to the resource owners. Moreover, in this model a Greedy Backfilling (GB) scheduling algorithm is utilized for maximizing the resource owner's payoff function. This approach presents the following advantages: (1) controlling the amounts of Grid workload, (2) delivering a specific degree of QoS to the end user, (3) reducing the queue waiting time, (4) optimizing resource provider's payoff and (5) autonomous allocation of Grid resources.

Table 1
Main characteristics of Gird scheduling algorithms.

Algorithm name	Grid core	Grid scheduler	SLA coordination	Resource discovery	Resource reservation	Queue scheduler
SDPA [6]			SLA components			SDPA
BFOHH [7]			Job offer deadline	RP		BFOHH
CGPA [8]			Job offer deadline			CGPA
GPMS [9]			Job offer deadline			GPMS
PAP/RTDP [10]	InterGrid	Metascheduler	Job offer deadline			FCFS
DAM [11]	EDG	Metascheduler	Job offer deadline	CA		GB
CASA [12]	MDS	Metascheduler	Community-aware nodes			FCFS
MSA/RA [13]	MDS	Metascheduler	WS-Agreement		RA	MSA
RDDLA [14]	MDS	Metascheduler	Job offer deadline	DLA		FCFS
DOG [20]	MDS	Metascheduler	Job offer deadline	DLA	PSO	FCFS
SCS [2]	MDS	Superscheduler	Contract Net Protocol			GB
SAS	MDS	Superscheduler	Bid/Ask SLA management	RF		AGB

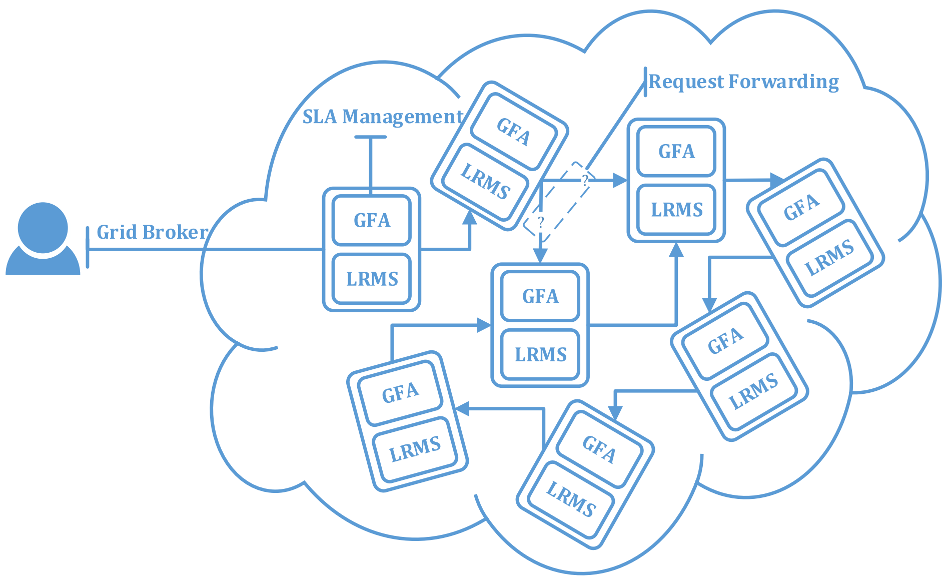

Fig. 1. Grid architecture.

2.12. *Comparison*

Table 1 shows the comparison summary of this section. The columns of this table present the following information:

(1) The Grid core column shows that like SAS most of the Grid scheduling algorithms follows the MDS principles.
(2) The Grid scheduler column shows that metascheduling is the most common scheduling framework for Grids.
(3) The SLA coordination column shows that most of the scheduling algorithms ignore the SLA management strategy.
(4) The resource discovery and resource reservation columns show that these features provides auxiliary support for Grids.
(5) The queue scheduler column shows that FCFS and GB are frequently implemented as the queue scheduler for Grid nodes.

3. SLA based adaptive Grid superscheduling

3.1. *Grid architecture*

Fig. 1 shows the emulated Grid topology. Having a look at this figure, each Grid node contains two key components: (1) GFA component that receives/forwards the job and establishes an SLA that meets both criteria of service provider and consumer, and (2) LRMS component that manages the local resources and monitors the job queue. The coalition between these two components brings the autonomy to Grid nodes and enables them to form different management domains.

As shown in Fig. 1, when a user enters the Grid system via a Grid broker, it will connect to the first nearest geographically Grid node. Then, the corresponding GFA component probes the user's job and negotiates the SLA terms based on service provider and consumer criteria. If the SLA terms are met for the job, the job will be enqueued to the resource queue of LRMS component and wait for future execution, otherwise it will be forwarded to another adjacent Grid node by using a decentralized request forwarding strategy.

3.2. *SLA based Adaptive Superscheduling algorithm*

The SAS algorithm comprises of two customized components of GFA and LRMS. The GFA component includes two sub-algorithms:

(1) The request forwarding strategy that forwards resource requests to other Grid nodes based on power and success criteria.
(2) The SLA management strategy that negotiates the SLA terms.

Also, the LRMS component includes three sub-algorithms:

(1) The job scheduling strategy that aggregates the LRMS information and offers qualified resources to the end user.
(2) The success rate strategy that calculates the probability of successful execution of incoming jobs and conveys this information to request forwarding strategy for future request routing.

(3) The service pricing strategy that aims to maximize the resource provider's payoff while preserving the QoS metrics for resource consumer.

The pseudo-code of SAS algorithm is shown in Algorithm 1.

Algorithm 1: SLA based Adaptive Superscheduling (SAS)

```
Define
    Consider start node $v_s$
    Consider function Call $(A, B)$ for invoking the $B$ sub-algorithm of
A component of Grid node
Begin
  For each user Do in Parallel
      The user connects to the Grid node $v_s$ via a Grid broker
      For each job Do
          Call (GFA$_s$, request forwarding strategy)
      End For
  End For
End
```

3.3. Grid federation agent

3.3.1. Request Forwarding strategy

The Grid Request Forwarding (RF) [4] is defined as: "finding the proper node for job scheduling by transmitting the jobs through the Grid". When a user connects to the Grid, the Grid broker will deliver the user's job to the nearest geographically Grid node. After that, the GFA of the corresponding node will start to negotiate the SLA terms with the user. If the SLA terms are satisfied, the job will be delivered to the LRMS resource queue for potential execution, otherwise the current GFA will forward the job to the next adjacent node by using the request forwarding strategy.

$$\text{Grid Node}\,\Big|_{\text{Max}(power,\ probability)} \tag{1}$$

where in (1), *power* is the total computing power of a Grid node and *probability* is the probability of the successful execution of jobs on a Grid node.

In request forwarding strategy (Algorithm 2), the process of selecting a node starts with finding the adjacent nodes and stops with selecting the best adjacent node in terms of computing power or success rate. At first, a graph analogous to the Grid overlay network is constructed and then the adjacent Grid nodes that reside in the local neighborhood are determined. Next, the Grid node with the highest computational power or success rate will be selected based on pieces of information that are provided by adjacent Grid nodes. At last, the request forwarding strategy calls the SLA management strategy. If the SLA conditions are met, the job will be scheduled for plausible execution, otherwise it will be forwarded to another Grid node. The pseudocode of the request forwarding strategy of GFA component is shown in Algorithm 2.

Algorithm 2: Request Forwarding Strategy

```
Define
    Construct a graph $G = (V, E)$ where $V$ is a set of Grid nodes and $E$
is a set of network links
    Construct Grid nodes where each Grid node constitutes a Grid
Federation Agent (GFA) and Local Resource Management Service
(LRMS)
    Consider start node $v_s$, current node $v_k$ and next node $v_n$
    Consider hop number $h = 0$ and maximum hop number $H = 99$
    Consider $J_{i,j,k}$ as $i_{\text{th}}$ job of $j_{\text{th}}$ user at $k_{\text{th}}$ GFA
    Consider $R_{k,l}$ as $l_{\text{th}}$ resource of $k_{\text{th}}$ LRMS
    Consider $R_{power\,k}$ as computing power of $k_{\text{th}}$ LRMS
    Consider $R_{probability\,k}$ as success rate of $k_{\text{th}}$ LRMS
    Consider function Call $(A, B)$ for invoking the $B$ sub-algorithm of $A$
component of Grid node
    Consider function Set $(A, B)$ for copying variable $B$ to variable $A$
    Consider function Add $(v)$ for adding a Grid node $v$ to traverse
path $\pi$
Begin
    Set start node $v_s$ to current node $v_k$
  Set $(v_s, v_k)$
  While$(h < H)$ Do
      If $(v_k$ is not visited$)$ Then
          Let $v_n$ be the adjacent node to $v_k$ with the highest $R_{power\,k}$
          Start SLA negotiation
        Call (GFA$_k$, SLA management Strategy)
            If (Return (0)) Then
                $J_{i,j,k}$ is scheduled on LRMS$_k$
                Start processing next job
                Exit (0)
            End If
            Else If (Return (1)) Then
                $J_{i,j,k}$ is not scheduled on LRMS$_k$
                Update the traverse path $\pi_{i,j}$
                Add $(v_k)$
                Set next node $v_n$ to current node $v_k$
                Set $(v_k, v_n)$
            End Else If
        End Call
      End If
      Else If $(v_k$ is visited$)$ Then
          Let $v_n$ be the adjacent node to $v_k$ with the highest $R_{probability\,k}$
          Update the traverse path $\pi_{i,j}$
        Add $(v_k)$
          Set next node $v_n$ to current node $v_k$
        Set $(v_k, v_n)$
      End Else If
       $h++$
  End While
End
```

3.3.2. SLA management strategy

After selecting a proper Grid node by using the request forwarding strategy, the user's job will be delivered to the GFA component of the selected Grid node for SLA negotiation. The SLA management strategy consists of two major steps:

(1) Check availability: Since the Grid nodes may disconnect sporadically, the first step is to check whether the selected GFA is enabled or not.
(2) Take snapshot: Next, the GFA takes snapshots from LRMS's resource pool.

In second step, the SLA management strategy uses a double-sided auction for the process of receiving and providing the services when plausible service consumers submit ask prices and plausible service providers submit bid prices. Furthermore, the SLA management strategy guarantees a fair auction for both sides.

3.3.2.1. Processing ask prices. In SAS, $J_{i,j,k}$ denotes the job from the jth$|j \in [1, \dots, m]$ user that is resident in the kth$|k \in [1, \dots, o]$ Grid node. After checking the availability of GFA$_k$, the total length of $J_{i,j,k}$ is calculated using (2).

$$J_{length\ j,k} = \sum_{i=1}^{n} J_{length\ i\in[1,\dots,n],j,k} \tag{2}$$

where in (2), $J_{length\,j,k}$ is the length of $J_{j,k}$ in FLOPS. Also, jth user has a total ask price $J_{ask\,j,k}$ that is divided by (3) between its jobs.

$$J_{ask\,i\in[1,\dots,n],j,k} = \frac{J_{length\,i\in[1,\dots,n],j,k}}{J_{length\,j,k}} \times J_{ask\,j,k} \tag{3}$$

where in (3), $J_{ask\,i,j,k}$ is the ask price of $J_{i,j,k}$ in dollar Also, (3) guarantees a fair distribution of total ask price by considering the job's length as its primary criterion.

3.3.2.2. Processing bid prices. Before processing the bid prices, the GFA$_k$ takes a snapshot from resource pool of LRMS$_k$ and calculates the processing power of member resources. The processing power of lth$|l \in [1, \dots, p]$ resource is calculated through (4):

$$R_{power\,k,l} = R_{cores\,k,l} \times R_{clock\,k,l} \times \frac{R_{FLOPs\,k,l}}{R_{cycle\,k,l}} \tag{4}$$

where the following parameters are used in (4):

- $R_{power\,k,l}$ is the computational power of $R_{k,l}$ in FLOPS.
- $R_{cores\,k,l}$ is the number of processor cores of $R_{k,l}$.
- $R_{clock\,k,l}$ is the clock rate of $R_{cores\,k,l}$.
- $\frac{FLOPs_{k,l}}{Cycle_{k,l}}$ is the FLOPs per clock rate ($R_{clock\,k,l}$) cycle of $R_{k,l}$ processor cores ($R_{cores\,k,l}$).

Also, the total computing power of LRMS$_k$ is calculated by (5):

$$R_{power\,k} = \sum_{l=1}^{p} R_{power\,k,l}. \tag{5}$$

After calculating the $R_{power\,k,l}$, the $R_{k,l}$ will be sorted by the objective function of (6).

$$J_{i,j,k} \left| \mathrm{Max} \left(R_{power\,k,l \in [1,\dots,p]} \right) \right. . \tag{6}$$

Since the initial resource prices are set with regard to their computational power, $J_{i,j,k}$ may not afford to accept the bid. So, in (7) the $J_{ask\,i,j,k}$ is checked against the bid price $R_{bid\,i,j,k}$ to verify whether it can afford leasing the $R_{k,l}$ or not.

$$J_{negotiation\,i,j,k} = \begin{cases} 1 & \exists l \in [1, \dots, p] | J_{ask\,i,j,k} \geq R_{bid\,k,l} \\ 0 & \forall l \in [1, \dots, p] | J_{ask\,i,j,k} < R_{bid\,k,l} \end{cases} \tag{7}$$

where in (7), $J_{negotiation\,i,j,k}$ is the negotiation status of $J_{i,j,k}$ on $R_{k,l}$. After sorting the LRMS$_k$ member resources using (6), there are two conditions that may occur in (7):

(1) If there exists at least one $R_{bid\,k,l \in [1,\dots,p]}$ that is less than the $J_{ask\,i,j,k}$, the $J_{i,j,k}$ will be scheduled on LRMS$_k$.
(2) If there is no $R_{bid\;k,l \in [1,\dots,p]}$ that is less than the $J_{ask\,i,j,k}$, the $J_{i,j,k}$ will be routed to the next available GFA by request forwarding strategy.

Finally, the negotiation timeout of $J_{i,j,k}$ is updated by (8):

$$T_{negotiation\,i,j} = T_{negotiation\,i,j} + T_{negotiation\,i,j,k} \tag{8}$$

where in (8), $T_{negotiation\,i,j}$ is the total negotiation timeout of $J_{i,j}$ and $T_{negotiation\,i,j,k}$ is the negotiation timeout at the GFA$_k$.

Algorithm 3 is the pseudocode of SLA management strategy of GFA component.

Algorithm 3: SLA Management Strategy

```
Define
    Consider current node $v_k$ where $k$ is the Grid node index
    Consider $J_{i,j,k}$ as $i_{th}$ job of $j_{th}$ user at $k_{th}$ GFA
    Consider $R_{k,l}$ as $l_{th}$ resource of $k_{th}$ LRMS
    Consider function Call ($A$, $B$) for invoking the $B$ sub-algorithm of $A$
component of Grid node
    Consider function Set ($A$, $B$) for copying variable $B$ to variable $A$.
Begin
  If (GFA$_k$ is enabled) Then
      GFA$_k$ takes a snapshot from LRMS$_k$ resource pool
      Calculate ask price $J_{ask\,i,j,k}$ of $J_{i,j,k}$ by (2) and (3)
      Calculate resource power $R_{power\,k,l}$ of $R_{k,l}$ by (4)
        If ($\forall l \in [1, \dots, p] | R_{bid\,k,l} = null$) Then
            For each resource Do
                Initialize bid price $R_{bid\,k,l}$
            End for
        End If
        If ( $\exists l \in [1, \dots, p] | J_{ask\,i,j,k} \geq R_{bid\,k,l}$) Then
            SLA conditions are met
            Set negotiation status $J_{negotiation\,i,j,k}$ of $J_{i,j,k}$ to 1
            Set ($J_{negotiation\,i,j,k}$, 1)
            Update negotiation timeout $T_{negotiation\,i,j}$ by (8)
            Start scheduling $J_{i,j,k}$ on LRMS$_k$
            Call (LRMS$_k$ , job scheduling strategy)
                If (Return (0)) Then
                    $J_{i,j,k}$ is scheduled on LRMS$_k$
                    Exit (0)
                End If
                Else If (Return (1)) Then
                    $J_{i,j,k}$ is not scheduled on LRMS$_k$
                    Exit (1)
                End Else If
            End Call
        End If
        Else If ($\forall l \in [1, \dots, p] | J_{ask\,i,j,k} < R_{bid\,k,l}$) Then
            SLA conditions are not met
            Set negotiation status $J_{negotiation\,i,j,k}$ of $J_{i,j,k}$ to 0
            Set ($J_{negotiation\,i,j,k}$, 0)
            Exit (1)
        End Else If
  End If
End
```

3.4. Local resource management system

3.4.1. Job scheduling strategy

After accepting the SLA terms in above step, $J_{i,j,k}$ is submitted to LRMS$_k$ for job scheduling. The job scheduling strategy follows the basic paradigm of greedy backfilling scheduling algorithm [2] along with a number of customized time constraints. Since the service pricing strategy that will be discussed in Section 3.4.3 provides support for plausible job submission on LRMSs, the proposed job scheduling strategy can be seen as an Adaptive Greedy Backfilling (AGB) scheduling algorithm. At first, the execution time of $J_{i,j,k}$ on $R_{k,l}$ is calculated through (9):

$$T_{execution\,i,j,k} = \frac{J_{length\,i,j,k}}{R_{power\,k,l}} \tag{9}$$

where in (9), the $T_{execution\,i,j,k}$ is the period of time that $J_{i,j,k}$ will need for execution on $R_{k,l}$. Then, the submission timeout of $J_{i,j,k}$ is updated by (10):

$$T_{submission\,i,j} = T_{submission\,i,j} + T_{submission\,i,j,k} \tag{10}$$

where in (10), $T_{submission\,i,j}$ is the total submission timeout of $J_{i,j}$ and $T_{submission\,i,j,k}$ is the submission timeout at the GFA$_k$. Next, by adding the negotiation and submission timeouts $T_{negotiation\,i,j}$ and $T_{submission\,i,j}$, the submit time of $J_{i,j,k}$ is calculated by (11):

$$T_{submit\,i,j,k} = T_{submission\,i,j} + T_{negotiation\,i,j} \tag{11}$$

where in (11), $T_{submit\,i,j,k}$ is the submit time of $J_{i,j,k}$ to the $R_{k,l}$. After calculating the submit time, the remaining time of $J_{i,j,k}$ is calculated through (12):

$$T_{remaining\,i,j,k} = T_{deadline\,i,j} - (T_{enter\,i,j} + T_{submit\,i,j,k} + T_{execution\,i,j,k}) \tag{12}$$

where in (12), $T_{remaining\,i,j,k}$ is the period of time that $J_{i,j,k}$ could be in LRMS$_k$, $T_{deadline\,i,j}$ is the expiry time of $J_{i,j}$, $T_{enter\,i,j}$ is the time when $J_{i,j}$ is entered to the Grid. Next, in (13) the remaining time is checked against execution time. If the remaining time is more than the execution time, the $J_{i,j,k}$ will be moved to next step, otherwise the request forwarding strategy will route the $J_{i,j}$ to another GFA.

$$J_{submission\ i,j,k} = \begin{cases} 1 & T_{remaining\ i,j,k} \geq T_{execution\ i,j,k} \\ 0 & T_{remaining\ i,j,k} < T_{execution\ i,j,k} \end{cases} \tag{13}$$

where in (13), $J_{submission\ i,j,k}$ is the submission status of $J_{i,j,k}$ on $R_{k,l}$. After passing the conditions of (13), the plausible finish time of $J_{i,j,k}$ is calculated through (14):

$$T_{finish\ i,j,k} = T_{submit\ i,j,k} + T_{execution\ i,j,k} \tag{14}$$

where in (14), $T_{finish\ i,j,k}$ is the finish time of $J_{i,j,k}$ on $R_{k,l}$. Next, in (15) the finish time is checked against queue time of $R_{k,l}$. If the finish time is less than the queue time, the $J_{i,j,k}$ will be scheduled for execution, otherwise the request forwarding strategy will route the $J_{i,j}$ to another GFA.

$$J_{submission\ i,j,k} = \begin{cases} 1 & T_{finish\ i,j,k} \leq T_{queue\ k,l} \\ 0 & T_{finish\ i,j,k} > T_{queue\ k,l} \end{cases} \tag{15}$$

where in (15), $T_{queue\ k,l}$ is the time that $R_{queue\ k,l}$ becomes free. Moreover, the $T_{waiting\ i,j,k}$ is calculated by (16):

$$T_{waiting\ i,j,k} = T_{queue\ k,l} - T_{submit\ i,j,k} \tag{16}$$

where in (16), $T_{waiting\ i,j,k}$ is the time that $J_{i,j,k}$ should wait in the $R_{queue\ k,l}$ for plausible execution. Finally, the queue time of $R_{k,l}$ is updated through (17):

$$T_{queue\ k,l} = T_{queue\ k,l} + T_{execution\ i,j,k}. \tag{17}$$

Fig. 2 sketches the timeline of SLA negotiation and job scheduling strategies of SAS.

Algorithm 4 is the pseudocode of job scheduling strategy of LRMS component.

Algorithm 4: Job Scheduling Strategy

```
Define
    Consider current node $v_k$ where $k$ is the Grid node index
    Consider $J_{i,j,k}$ as $i_{th}$ job of $j_{th}$ user at $k_{th}$ GFA
    Consider $R_{k,l}$ as $l_{th}$ resource of $k_{th}$ LRMS
    Consider function Call $(A, B)$ for invoking the $B$ sub-algorithm of $A$ component of Grid node
    Consider function Set $(A, B)$ for copying variable $B$ to variable $A$.
Begin
    Start scheduling of $J_{i,j,k}$
    Calculate execution time $T_{execution\ i,j,k}$ by (9)
    Update submission timeout $T_{submission\ i,j}$ by (10)
    Calculate submit time $T_{submit\ i,j,k}$ by (11)
    Calculate remaining time $T_{remaining\ i,j,k}$ by (12)
    If $(T_{remaining\ i,j,k} \geq T_{execution\ i,j,k})$ Then
        Calculate finish time $T_{finish\ i,j,k}$ by (14)
        If $(T_{finish\ i,j,k} \leq T_{queue\ k,l})$ Then
            Set submission status $J_{submission\ i,j,k}$ of $J_{i,j,k}$ on $R_{k,l}$ to 1
            Set $(J_{scheduling\ i,j,k}, 1)$
            Submit $J_{i,j,k}$ to the job queue $R_{queue\ k,l}$ of $R_{k,l}$
            Submit$(J_{i,j,k}, R_{k,l})$
            Update queue time $T_{queu\ k,l}$ of $R_{queue\ k,l}$ by (17)
            Update total ask price $J_{ask\ j,k}$
            Call(LRMS$_k$, success rate strategy)
            Call(LRMS$_k$, service pricing strategy)
            Exit (0)
        End If
        Else If $(T_{finish\ i,j,k} > T_{queue\ k,l})$ Then
            Set submission status $J_{submission\ i,j,k}$ of $J_{i,j,k}$ on $R_{k,l}$ to 0
            Set $(J_{scheduling\ i,j,k}, 0)$
            Call(LRMS$_k$, success rate strategy)
            Call(LRMS$_k$, service pricing strategy)
            Exit (1)
        End Else If
    End If
    Else If $(T_{remaining\ i,j,k} < T_{execution\ i,j,k})$ Then
        Set submission status $J_{submission\ i,j,k}$ of $J_{i,j,k}$ on $R_{k,l}$ to 0
        Set $(J_{scheduling\ i,j,k}, 0)$
        Call(LRMS$_k$, success rate strategy)
        Call(LRMS$_k$, service pricing strategy)
        Exit (1)
    End Else If
End
```

3.4.2. *Success rate strategy*

To have an adaptive and dynamic feedback from Grid environment, the scheduling status of jobs is tracked during the request routing. This mechanism is inspired from the adaptive approach of success rate strategy [23]. While employing this strategy, a high success rate demonstrates a high service volume and consequently a high resource efficiency.

$$R_{probability\ k} = \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\frac{AND\left(J_{negotiation\ i,j,k}, J_{submission\ i,j,k}\right)}{OR\left(J_{negotiation\ i,j,k}, J_{submission\ i,j,k}\right)} \right) \tag{18}$$

where in (18), $J_{negotiation\ i,j,k}$ and $J_{submission\ i,j,k}$ are the negotiation and submission status of $J_{i,j,k}$ on $R_{k,l}$ that are derived from (7), (13) and (15). $R_{probability\ k} \in [0, 1]$ is the corresponding probability of the portion of jobs that are successfully scheduled by LRMS$_k$. (18) tries to note the following hints:

(1) If $R_{probability\ k}$ is near zero, most of incoming jobs are rejected by SLA management strategy or Grid job scheduler.
(2) If $R_{probability\ k}$ is near one, the SLA management mechanism will guarantee the plausible execution of incoming jobs on LRMS$_k$.

Algorithm 5 is the pseudocode of success rate strategy of LRMS component of SAS.

Algorithm 5: Success Rate Strategy

```
Define
    Consider current node $v_k$ where $k$ is the Grid node index
    Consider $J_{i,j,k}$ as $i_{th}$ job of $j_{th}$ user at $k_{th}$ GFA
    Consider $R_{k,l}$ as $l_{th}$ resource of $k_{th}$ LRMS
Begin
    Updating success rate of LRMS$_k$
    Calculating LRMS$_k$ success rate $R_{probability\ k}$ by (18)
End
```

3.4.3. *Service pricing strategy*

After submitting the $J_{i,j,k}$ to $R_{queue\ k,l}$, the service pricing strategy is employed to promote the resource providers' profit while preserving the purchasing power for resource consumers. At first, the $R_{k,l}$ load $R_{load\ k,l}$ and LRMS$_k$ average load $R_{load\ k}$ are calculated by (19) and (20) respectively.

$$R_{load\ k,l} = \frac{T_{execution\ i,j,k}}{T_{remaining\ i,j,k}} \tag{19}$$

$$R_{load\ k} = \frac{\sum_{l=1}^{p} R_{load\ k,l}}{p}. \tag{20}$$

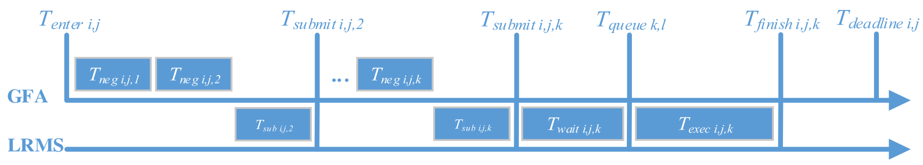

Fig. 2. SAS algorithm timeline while scheduling $J_{i,j,k}$ on $R_{k,l}$.

Table 2
Service pricing criteria.

Row	Load	Price	Bid price
1	$R_{load\ k,l} < R_{load\ k}$	$R_{bid\ k,l} > J_{ask\ i,j,k}$	Decrease (↓)
2	$R_{load\ k,l} < R_{load\ k}$	$R_{bid\ k,l} < J_{ask\ i,j,k}$	Increase (↑)
3	$R_{load\ k,l} \geq R_{load\ k}$	$R_{bid\ k,l} > J_{ask\ i,j,k}$	No change (×)
4	$R_{load\ k,l} \geq R_{load\ k}$	$R_{bid\ k,l} < J_{ask\ i,j,k}$	Increase (↑)

After calculating the load of $R_{k,l}$ and LRMS$_k$, the four conditions of Table 2 are applied to the submitted jobs of $R_{queue\ k,l}$ to demonstrate the bargaining power of service pricing strategy.

The following hints can be drawn from Table 2:

(1) If $R_{load\ k,l} < R_{load\ k}$ and $R_{bid\ k,l} > J_{ask\ i,j,k}$, the $R_{bid\ k,l}$ will be decreased by (21).

(2) If $R_{load\ k,l} < R_{load\ k}$ and $R_{bid\ k,l} < J_{ask\ i,j,k}$, the $R_{bid\ k,l}$ will be increased by (22).

(3) If $R_{load\ k,l} \geq R_{load\ k}$ and $R_{bid\ k,l} > J_{ask\ i,j,k}$, the $R_{bid\ k,l}$ will have no change.

(4) If $R_{load\ k,l} \geq R_{load\ k}$ and $R_{bid\ k,l} > J_{ask\ i,j,k}$, the $R_{bid\ k,l}$ will be increased by (22).

$$R_{bid\ k,l} = R_{bid\ k,l} - (R_{weight\ k} \times R_{bid\ k,l}) \quad (21)$$

$$R_{bid\ k,l} = R_{bid\ k,l} + (R_{weight\ k} \times R_{bid\ k,l}) \quad (22)$$

where in (21) and (22), $R_{weight\ k}$ is the weighted factor of LRMS$_k$. Algorithm 6 is the pseudocode service pricing strategy of LRMS component.

Algorithm 6: Service Pricing Strategy

```
Define
    Consider current node $v_k$ where $k$ is the Grid node index
    Consider $J_{i,j,k}$ as $i_{th}$ job of $j_{th}$ user at $k_{th}$ GFA
    Consider $R_{k,l}$ as $l_{th}$ resource of $k_{th}$ LRMS
    Consider function Call (A, B) for invoking the B sub-algorithm of A
component of Grid node
    Consider function Set (A, B) for copying variable B to variable A.
Begin
    Enforcing pricing strategy to $R_{bid\ k,l}$
    Calculate resource load $R_{load\ k,l}$ by (19)
    Calculate LRMS$_k$ average load $R_{load\ k}$ by (20)
    If ($R_{load\ k,l} < R_{load\ k}$) Then
        If ($R_{bid\ k,l} > J_{ask\ i,j,k}$) Then
            Decrease $R_{bid\ k,l}$ by (21)
        End If
        Else If ($R_{bid\ k,l} < J_{ask\ i,j,k}$) Then
            Increase $R_{bid\ k,l}$ by (22)
        End Else If
    End If
    If($R_{load\ k,l} \geq R_{load\ k}$) Then
        If ($R_{bid\ k,l} > J_{ask\ i,j,k}$) Then
            Do nothing
        End If
        Else If ($R_{bid\ k,l} < J_{ask\ i,j,k}$) Then
            Increase $R_{bid\ k,l}$ by (22)
        End Else If
    End If
End
```

The service pricing strategy of SAS presents the following features:

(1) The price of resources that reside in a busy LRMS will be increased and eventually the LRSM total load will decrease gradually.
(2) Since a busy LRMS increases its resource prices, its total payoff will be increased because jobs with higher prices can just meet the new SLA terms.
(3) The price of resources that reside in an idle LRMS will be decreased and eventually the LRSM total load will increase gradually.
(4) Since an idle LRMS decreases its resource prices, its total payoff will be increased because jobs with lower prices can just meet the SLA terms.

3.5. *SLA based Adaptive Superscheduling model*

Fig. 3 depicts the overall model of SAS algorithm. The GFA and LRMS are two components of SAS algorithm that each of which composes of subordinate sub-components. At first, the request forwarding strategy forwards the incoming job to the GFA component of an adjacent node and the GFA's SLA management strategy processes the consumer ask prices and provider bid prices. After calculating the negotiation time and accepting the SLA terms, the job will be sent to LRMS component for plausible scheduling. If the job scheduling strategy of LRMS component could schedule the incoming job, it will be enqueued, otherwise it will be forwarded to another Grid node. Then, whether the job is scheduled or not in the current LRMS, the success rate and service pricing strategies will be executed. The success rate strategy estimates the LRMS probability that conveys the probability of successful execution of jobs on its local resources. This probability will be used for further distribution of jobs that may pass this Grid node. Finally, the service pricing strategy fuses the information about LRMS load, consumer ask price and provider bid price to update the provider bid price. The new bid price will be utilized for further SLA negotiations.

4. Simulation results

4.1. *Network settings*

Most of real-world networks such as Internet, Social networks and Cloud and Grid infrastructures are scale-free networks. If a scale-free network is modeled by a directed graph of $G = (V, E)$ where V is a set of nodes and E is a set of directed edges, the number of head and tail endpoints adjacent to a node is called the in-degree and out-degree, respectively. The out-degree of scale-free network follows a power law distribution. The power law is a functional relationship between two parameters, where one parameter changes as a power of another. A scale-free network has a large number of nodes with low out-degrees and a small number of nodes with extremely high out-degrees. Since the power law distribution is scale invariance, the structure and dynamicity of a scale-free network is independent of the network size.

In order to generate a scale-free network, the Waxman topology model [24] is used to generate a random topology. In Waxman model, the nodes of the overlay network are uniformly distributed according to the Waxman's probability model. Also,

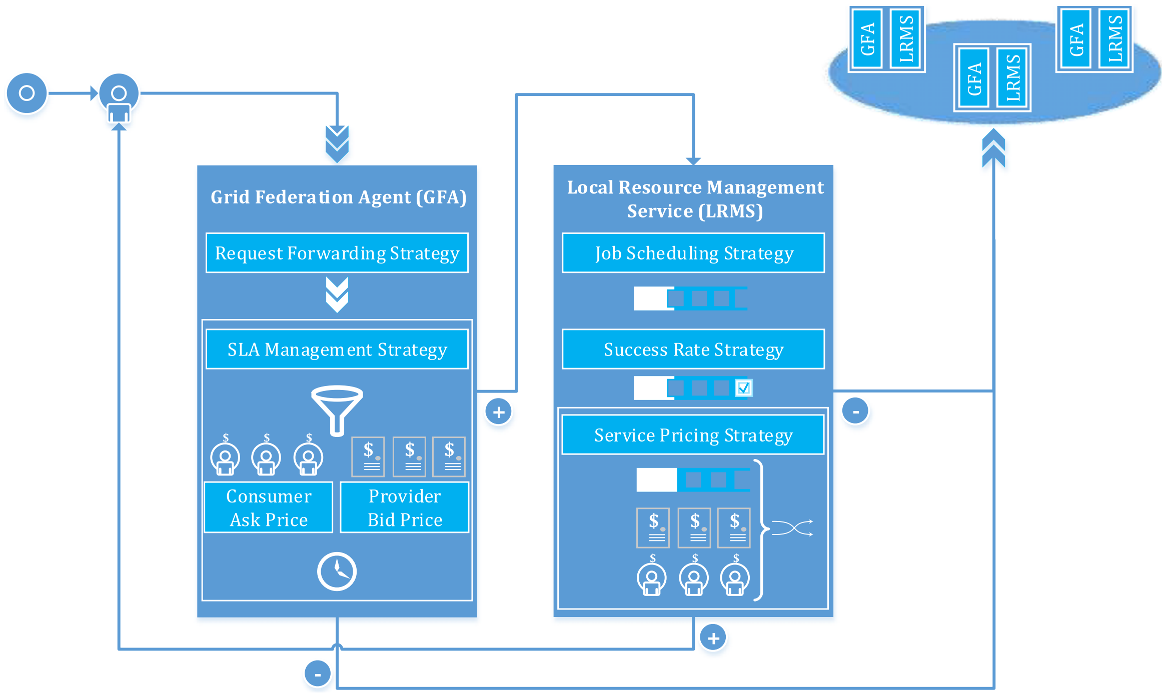

Fig. 3. SAS Model.

the Barabási–Albert algorithm [25] is utilized to employ a power law distribution in the frequency of out-degrees while generating a scale-free network.

4.2. Simulation settings

The SAS algorithm is compared with the SLA based CNP Super-scheduling (SCS) algorithm [2]. These algorithms are implemented in GridSim toolkit [26] which is a Java based simulation library for modeling and simulation of distributed resource management and scheduling. This toolkit allows simulation of entities such as users, applications and brokers for evaluation of scheduling algorithms.

In SAS algorithm, each Grid node composes of GFA and LRMS components. Emulating a realistic Grid environment, the Grid nodes (sets of GFA and LRMS) are linked in a network topology. All links are full duplex with similar characteristics as follows:

(1) Baud rates of 1 GB per second.
(2) Propagation delay of 10 ms.
(3) Maximum Transmission Unit (MTU) of 10 MB in unit.

4.2.1. Grid scales

The performance of SAS and SCS algorithms is investigated numerically in three Grid scales:

(1) Small scale Grid with 100 nodes.
(2) Medium scale Grid with 500 nodes.
(3) Large scale Grid with 1000 nodes.

4.2.2. Grid workloads

In the emulated Grid, there are 30 users with a set of 100 jobs. These jobs are from real world datasets of Grid Workloads Archive (GWA) [27]. The jobs' length, total ask price and ask price of user are three key job characteristics that have the following details:

(1) The job length has an Exponential distribution with parameter $\mu = 3500$.
(2) The total ask price is \$700.
(3) The job ask price is calculated as for the length of job and total ask price from (3).

4.2.3. Grid resources

The numbers of resources are equal to the number of Grid nodes in different scales. Like Grid workloads, the Grid resources should have similar features. The number of cores per processor, clock per core and processor price is three key resource characteristics. The details of these resource characteristics are as follows:

(1) The number of cores per processor has a Poisson distribution with parameter $\lambda = 7$.
(2) The clock rate per core has an Exponential distribution with parameter $\mu = 3500$.
(3) The processor price has a Poisson distribution with parameter $\lambda = 7$.

4.2.4. Grid pricing context

Experiments are conducted for two pricing contexts for Grid resources:

1. Static pricing context: In this context, the scheduling algorithms do not have any control on resource prices and resource prices are fixed during simulation. Thus, the service pricing strategy of SAS algorithm is disabled while using this context. This pricing context is used in the simulation setups of SCS [2].
2. Dynamic pricing context: In this context, the service pricing strategy that was introduced in Section 3.4.3 is employed in both SAS and SCS algorithms. This strategy adjusts the resource price based on resources and LRMS's loads.

4.2.5. Grid quality of service criteria

The hit ratio, resource utilization, provider earnings and consumer savings are used as Quality of Service (QoS) criteria to investigate the SAS performance. The details of these criteria are as follows:

(1) Hit ratio: The hit ratio is the portion of jobs that are successfully scheduled on different Grid resources. Hit ratio shows the overall performance of scheduling algorithm. This criterion is

shown in (23):

$$\text{Hit ratio} = \left(\frac{\text{Number of hits}}{(\text{Number of hits} + \text{Number of misses})} \times 100\right)\%. \quad (23)$$

(2) Resource utilization: The resource utilization is the percentage of Grid resources that are engaged during the process of job scheduling. Resource utilization demonstrates the load balancing feature of scheduling algorithm. This criterion is shown in (24):

$$\text{Resource utilization} = \left(\frac{\text{Number of matched resources}}{\text{Number of resources}} \times 100\right)\%. \quad (24)$$

(3) Provider earnings: The resource provider earnings is the total payoff of resource providers from bid prices. This amount shows the received profit of providers. This criterion is shown in (25), where k is the number of Grid nodes (GFAs):

$$\text{Provider earnings} = \sum_{k=1}^{o} \text{payoff}_k. \quad (25)$$

(4) Consumer savings: The resource consumer savings is the total saving of resource consumers from ask prices. This amount shows the received cutoff of consumers. This criterion is shown in (26):

$$\text{Consumer savings} = \text{Total ask price} - \sum_{k=1}^{o} \text{payoff}_k. \quad (26)$$

4.3. Experiment 1—hit ratio

The results of SAS and SCS algorithms for both static and dynamic prices are depicted in Fig. 4. The SAS's hit ratio starts from 80% for 100 Grid nodes and increases to 90% for 1000 Grid nodes while SCS's hit ratio starts from 40% for 100 Grid nodes and diminishes to 30% for 1000 Grid nodes. The hit ratio is strictly increasing in SAS algorithm while it is strictly decreasing in SCS algorithm. This pattern is mutual between both static and dynamic price contexts with a significant difference between them that demonstrates the superiority of dynamic pricing context.

From red columns of Fig. 4, the hit ratio of SAS algorithm in static price context is almost twice as much as the hit ratio of SCS algorithm. This result is due to the advanced interworking of GFA and LRMS components in each Grid node. The SLA management strategy of GFA component tries to prune underqualified jobs before submitting them to the job scheduling strategy of LRMS component. Thus, this algorithm outperforms the SCS algorithms in term of hit ratio.

From black columns of Fig. 4, the hit ratio of SAS algorithm in dynamic price context is almost three times as many as the hit ratio of SCS algorithm. Since the request forwarding strategy of SAS algorithm is designed for highly dynamic environments, SAS algorithm could easily maintain its performance and keep a large performance difference with SCS algorithm in all Grid scales.

4.4. Experiment 2—resource utilization

The average resource utilization of SAS and SCS algorithm are depicted in Fig. 5. This figure shows that in contrast to SCS algorithm which cannot utilize large number of resources, the SAS algorithm can utilize a large portion of resources in both small and medium scale Grids. Although the resource utilization of both algorithms is sharply decreased in large scale Grids, the SAS algorithm still outperforms the SCS algorithm in terms of number of visited resources. Finally, the SAS algorithm utilizes around 60%, 50% and 15% of Grid resources in small, medium and large Grid scales.

The request forwarding strategy of GFA component tries to effectively disseminate jobs within the Grid overlay. This strategy actively exchanges information with success rate strategy of LRMS component. The success rate strategy computes the probability of the successful execution of forthcoming jobs for the visited Grid nodes. So, if the current Grid node is not visited yet then the request will be forwarded to the most powerful Grid node, otherwise it will be forwarded to the node with the highest probability. These conditions guaranty the fair distribution of Grid jobs and make use of the previous experiments of other Grid jobs. On the other

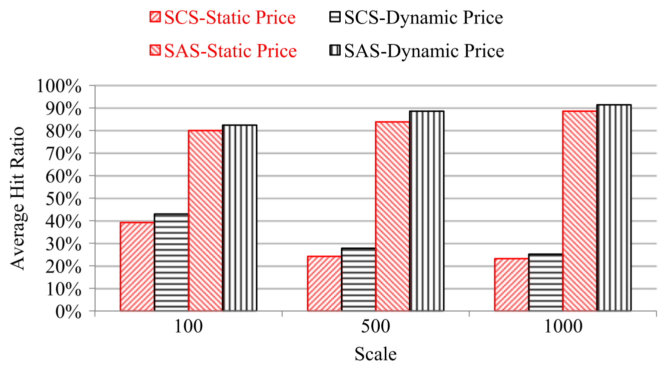

Fig. 4. The hit ratio of SAS and SCS algorithms for static (red columns) and dynamic (black columns) price contexts. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

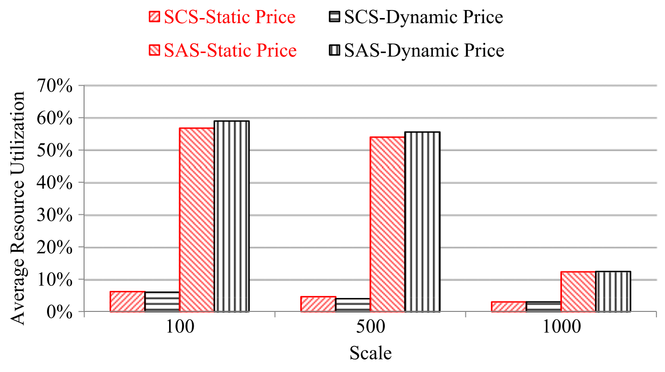

Fig. 5. The resource utilization of SAS and SCS algorithms for static (red columns) and dynamic (black columns) price contexts. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

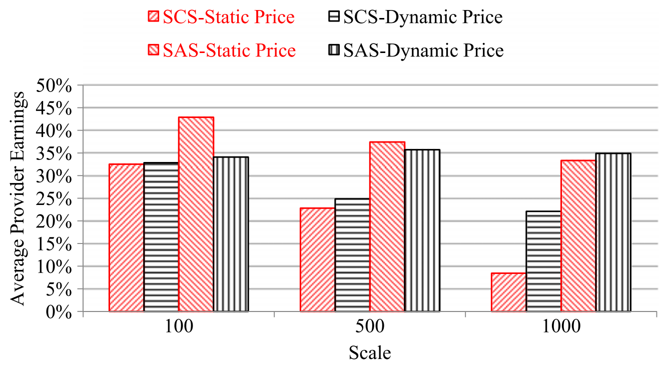

Fig. 6. The provider earnings of SAS and SCS algorithms for static (red columns) and dynamic (black columns) price contexts. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

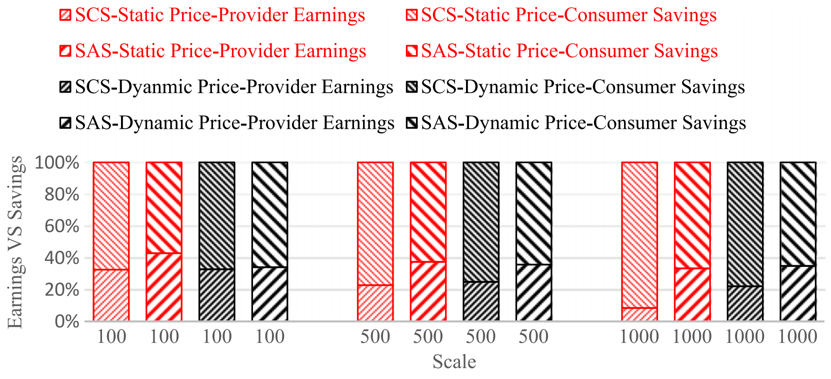

Fig. 7. The consumer savings versus provider earnings of SAS and SCS algorithms for static (red columns) and dynamic (black columns) price contexts. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

hand, since the SCS algorithm uses a greedy backfilling strategy for suggesting Grid resources, it cannot effectively diffuse Grid jobs and puts a small portion of Grid nodes (that are resided near the initial Grid node) under a huge execution load. This will lead to an unbalanced load distribution and consequently low resource utilization of SCS algorithm.

Having a look at Fig. 5, one can see a subtle difference between static and dynamic price contexts. This observation shows the superior results of dynamic pricing context. Interworking of the request forwarding and dynamic pricing strategies will lead to enhance the resource utilization of SAS algorithm. Dynamic pricing strategy manipulates the resource prices based on jobs' price and resources' load. On the contrary, although the SCS algorithm is equipped with this strategy, it cannot utilize it effectively due to the lack of request forwarding strategy.

4.5. *Experiment 3—revenue statistics*

In this section the economics of Grid is evaluated from the provider earnings and consumer savings views. The ability to schedule Grid works based on different time constraints promotes the revenue statistics of SAS algorithm. Also, the optimized structure of SAS algorithm improves the provider profits and consumer savings.

4.5.1. *Provider earnings*

Fig. 6 shows the provider earnings of two scheduling algorithms for two price contexts. The SAS algorithm suppresses SCS algorithm in all three different Grid scales and lets providers to earn more money. The profits of SAS algorithm are exceptionally higher than the SCS algorithm especially for large scale Grids. This observation completely justifies the dynamic, distributed and heterogeneous structure of SAS algorithm.

In summary, the provider earnings for static price is almost higher than the provider earnings for dynamic price in small and medium Grid scales because of the high density of available resources in these two scales that eases the process of job scheduling. On the other hand in large scale Grid that contains 1000 nodes, the results of dynamic price context are significantly higher than static price context especially for SAS algorithm. These results are due to the effective integration of dynamic pricing strategy and severe terms of SLA management strategy. The advanced interworking of these strategies eliminates the non-profitable jobs and prevents high computational overhead of resources.

4.5.2. *Provider earnings versus consumer savings*

Fig. 7 shows a comparative view of average consumer savings and average provider earnings. From this figure, by increasing the Grid scale the consumer savings and provider earnings of SAS algorithm slightly increases and decreases, respectively. This repeated pattern almost verifies the scalability feature of SAS algorithm.

Combining the SLA management and dynamic pricing strategies in SAS algorithm not only do providers earn more money by bidding fair prices, but also consumers save more money by asking feasible prices. Moreover, the average hit ratio of SAS algorithm is preserved in all three scales (Fig. 4). From the synthetic point of view, the balanced selection of bid and ask prices leads to a superior revenue statistics for SAS algorithm. For example, in large scale Grids, although the consumer savings is lower than provider earnings, the SAS algorithm could fulfill more than 90% of jobs (Fig. 7). So, this combined viewpoint acknowledges the superior power of service pricing strategy for processing bid prices while preserving a higher hit ratio compare to SCS algorithm.

5. Conclusion

Grid is a unified collection of distributed and heterogeneous computational resources which allows utilities sharing between geographically distributed clients. Furthermore, superscheduling enables job scheduling across different administrative domains with different scheduling policies.

In this paper, we propose a decentralized Grid architecture that uses the GFA and LRMS components in Grid nodes. The GFA component contains a native request forwarding strategy for query dispatching and a brokering based SLA management strategy for preserving optimal QoS constraints. The LRMS component contains a job scheduling strategy for aggregating Grid resources information, a success rate strategy for improving the chance of successful execution of users' job on Grid resources, and a service pricing strategy for maximizing the resource providers' payoff and resource consumer's satisfaction.

We conduct extensive experiments for different Grid scales including small, medium and large scale Grids. SAS algorithm shows superior performance in terms of scheduling efficiency and resource loading balance and utilization. Moreover, a dynamic pricing model is incorporated in all experiments that balance the QoS parameters such as provider earnings and consumer savings.

References

[1] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: Enabling scalable virtual organizations, Int. J. High Perform. Comput. Appl. 15 (3) (2001) 200–222.
[2] R. Ranjan, A. Harwood, R. Buyya, SLA-based coordinated superscheduling scheme for computational grids, in: 2006 IEEE International Conference on Cluster Computing, 2006, pp. 1–8.

[3] A. Luther, R. Buyya, R. Ranjan, S. Venugopal, Peer-to-peer grid computing and a .NET-based alchemi framework, in: L.T. Yang, M. Guo (Eds.), High-Performance Computing, John Wiley & Sons, Inc., 2005, pp. 403–429.
[4] A. Iamnitchi, I. Foster, A peer-to-peer approach to resource location in grid environments, in: J. Nabrzyski, J.M. Schopf, J. Węglarz (Eds.), Grid Resource Management, Springer, US, 2004, pp. 413–429.
[5] H.-G. Beyer, H.-P. Schwefel, Evolution strategies—a comprehensive introduction, Nat. Comput. 1 (1) (2002) 3–52.
[6] K.H. Khan, K. Qureshi, M. Abd-El-Barr, An efficient grid scheduling strategy for data parallel applications, J. Supercomput. 68 (3) (2014) 1487–1502.
[7] Rajni, I. Chana, Bacterial foraging based hyper-heuristic for resource scheduling in grid computing, Future Gener. Comput. Syst. 29 (3) (2013) 751–762.
[8] H. Xu, B. Yang, An incentive-based heuristic job scheduling algorithm for utility grids, Future Gener. Comput. Syst. 49 (2015) 1–7.
[9] G.T. Abraham, A. James, N. Yaacob, Group-based parallel multi-scheduler for grid computing, Future Gener. Comput. Syst. 50 (2015) 140–153.
[10] M. Amini Salehi, B. Javadi, R. Buyya, QoS and preemption aware scheduling in federated and virtualized Grid computing environments, J. Parallel Distrib. Comput. 72 (2) (2012) 231–245.
[11] S.K. Garg, S. Venugopal, J. Broberg, R. Buyya, Double auction-inspired meta-scheduling of parallel applications on global grids, J. Parallel Distrib. Comput. 73 (4) (2013) 450–464.
[12] Y. Huang, N. Bessis, P. Norrington, P. Kuonen, B. Hirsbrunner, Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Gener. Comput. Syst. 29 (1) (2013) 402–415.
[13] J. Conejero, B. Caminero, C. Carrión, L. Tomás, From volunteer to trustable computing: Providing QoS-aware scheduling mechanisms for multi-grid computing environments, Future Gener. Comput. Syst. 34 (2014) 76–93.
[14] M. Hasanzadeh, M.R. Meybodi, Grid resource discovery based on distributed learning automata, Computing 96 (9) (2014) 909–922.
[15] H. Beigy, M.R. Meybodi, Utilizing distributed learning automata to solve stochastic shortest path problems, Internat. J. Uncertain. Fuzziness Knowledge-Based Systems 14 (5) (2006) 591–615.
[16] A. Rezvanian, M. Rahmati, M.R. Meybodi, Sampling from complex networks using distributed learning automata, Physica A 396 (2014) 224–234.
[17] A. Rezvanian, M.R. Meybodi, Finding maximum clique in stochastic graphs using distributed learning automata, Internat. J. Uncertain. Fuzziness Knowledge-Based Systems 23 (1) (2015) 1–31.
[18] M. Hasanzadeh Mofrad, S. Sadeghi, A. Rezvanian, M.R. Meybodi, Cellular edge detection: Combining cellular automata and cellular learning automata, AEU-Int. J. Electron. Commun. 69 (9) (2015) 1282–1290.
[19] K.S. Narendra, M.A.L. Thathachar, Learning Automata: An Introduction, Prentice-Hall, Inc., 1989.
[20] M. Hasanzadeh, M.R. Meybodi, Distributed optimization grid resource discovery, J. Supercomput. 71 (1) (2015) 87–120.
[21] M. Hasanzadeh, M.R. Meybodi, M.M. Ebadzadeh, Adaptive cooperative particle swarm optimizer, Appl. Intell. 39 (2) (2013) 397–420.
[22] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing, in 10th IEEE International Symposium on High Performance Distributed Computing, 2001. Proceedings, 2001, pp. 181–194.
[23] M. Hasanzadeh, S. Sadeghi, A. Rezvanian, M.R. Meybodi, Success rate group search optimiser, J. Exp. Theoret. Artif. Intell. 0 (0) (2014) 1–17.
[24] B.M. Waxman, Routing of multipoint connections, IEEE J. Sel. Areas Commun. 6 (9) (1988) 1617–1622.
[25] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.
[26] R. Buyya, M. Murshed, Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurr. Comput.: Pract. Exper. 14 (13–15) (2002) 1175–1220.
[27] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D.H.J. Epema, The grid workloads archive, Future Gener. Comput. Syst. 24 (7) (2008) 672–686.

Mohammad Hasanzadeh Mofrad received his B.Sc. degree in Software Engineering from Payame Noor University (PNU), Birjand, Iran, in 2009 and M.Sc. in Artificial Intelligence from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran. He is currently pursuing Ph.D. degree in Computer Science at the Computer Science Department of the University of Pittsburgh, Pittsburgh, USA. His current research interests include grid computing, computational intelligence and machine learning.

Omid Jalilian received his B.Sc. degree in Software Engineering from Islamic Azad University of Qazvin, Iran in 2007, and the M.Sc. degree in Computer Engineering from Islamic Azad University of Qazvin, Qazvin, Iran, in 2011. He has been a Faculty Member of Payame Noor University (PNU) since 2013. His current research interests include information retrieval, web mining and grid computing.

Alireza Rezvanian received the B.Sc. degree in Computer Engineering from Bu-Ali Sina University of Hamedan, Hamedan, Iran, in 2007, and the M.Sc. degree in Computer Engineering with honors from Islamic Azad University of Qazvin, Qazvin, Iran, in 2010. He is currently pursuing Ph.D. degree in Computer Engineering at the Computer Engineering & Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran. His research activities include soft computing, evolutionary algorithms, complex social networks, and learning systems.

Mohammad Reza Meybodi received B.Sc. and M.Sc. degrees in Economics from the Shahid Beheshti University, Tehran, Iran, in 1973 and 1977, respectively. He also received M.Sc. and Ph.D. degrees from the Oklahoma University, Oklahoma, USA, in 1980 and 1983, respectively, in Computer Science. Currently he is a Full Professor in Computer Engineering Department, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran. Prior to his current position, he worked from 1983 to 1985 as an Assistant Professor at the Western Michigan University, and from 1985 to 1991 as an Associate Professor at the Ohio University, USA. His current research interests include, learning systems, cloud computing, soft computing and social networks.