Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm

Ye Huang [a,d,*], Nik Bessis [b,c], Peter Norrington [b], Pierre Kuonen [d], Beat Hirsbrunner [a]

[a] Department of Informatics, University of Fribourg, Switzerland
[b] Department of Computer Science and Technology, University of Bedfordshire, UK
[c] School of Computing and Mathematics, University of Derby, UK
[d] Department of Information and Communication Technologies, University of Applied Sciences of Western Switzerland (Fribourg), Switzerland

ARTICLE INFO

Article history:
Received 12 November 2010
Received in revised form
14 March 2011
Accepted 7 May 2011
Available online xxxx

Keywords:
Grid
Cloud
Scheduling
Meta-scheduling
Community-aware scheduling algorithm (CASA)
SmartGRID

ABSTRACT

Job scheduling strategies have been studied for decades in a variety of scenarios. Due to the new characteristics of the emerging computational systems, such as the grid and cloud, metascheduling turns out to be an important scheduling pattern because it is responsible for orchestrating resources managed by independent local schedulers and bridges the gap between participating nodes. Equally, to overcome issues such as bottleneck, single point failure, and impractical unique administrative management, which are normally led by conventional centralized or hierarchical schemes, the decentralized scheduling scheme is emerging as a promising approach because of its capability with regards to scalability and flexibility.

In this work, we introduce a decentralized dynamic scheduling approach entitled the community-aware scheduling algorithm (CASA). The CASA is a two-phase scheduling solution comprised of a set of heuristic sub-algorithms to achieve optimized scheduling performance over the scope of overall grid or cloud, instead of individual participating nodes. The extensive experimental evaluation with a real grid workload trace dataset shows that, when compared to the centralized scheduling scheme with BestFit as the metascheduling policy, the use of CASA can lead to a 30%–61% better average job slowdown, and a 68%–86% shorter average job waiting time in a decentralized scheduling manner without requiring detailed real-time processing information from participating nodes.

1. Introduction

Job scheduling strategies [1] have been extensively studied in the last few decades within a variety of scenarios, such as manufacturing systems and distributed computation environments. The increasing demand of computation resources has led to new types of cooperative distributed systems, such as the grid [2] and cloud computing [3]. Due to the new characteristics of emerging computational systems, conventional scheduling techniques need to evolve into more complex and sophisticated solutions in order to cover new scheduling constraints, such as heterogeneous resources, variety of job requirement, and dynamic and volatile networks. Furthermore, scheduling techniques allowing jobs to be shared between decentralized sites, virtual organizations (VOs), or even different grids/cloud providers, have appeared to be a promising approach because of their capability with regards to scalability and flexibility.

Metascheduling, also known as grid scheduling within the context of grid computing, turns out to be an important scheduling scheme because it is responsible for orchestrating resources managed by independent local schedulers and bridges the gap between isolated local computation resource pools. However, current research and implementation work have several crucial constraints and limitations, including: (a) scheduling for serving the hosting node, instead of the entire grid; and (b) assuming the detailed processing information of each participating node, such as the status of local job queue and real-time resource utilization, is known. In order to conquer such issues, a novel scheduling approach is desired to serve the overall grid, instead of each individual node, with a variety of preferred optimization objectives. Furthermore, such an approach is also supposed to work in a decentralized manner and be able to dynamically adapt to the changes in the grid through time.

Since the mid-1990s, the vision of a grid as a computation infrastructure has been widely accepted; numerous grid based

* Corresponding author at: Department of Informatics, University of Fribourg, Switzerland.
E-mail addresses: ye.huang@unifr.ch, huangye177@gmail.com (Y. Huang), nik.bessis@beds.ac.uk (N. Bessis), peter.norrington@beds.ac.uk (P. Norrington), pierre.kuonen@hefr.ch (P. Kuonen), beat.hirsbrunner@unifr.ch (B. Hirsbrunner).

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2011.05.006

resource sharing infrastructures such as Grid5000 [4], TeraGrid [5], D-Grid [6], EGEE [7], PlanetLab [8], and NorduGrid [9] have been established in different countries and continents serving both for production work and scientific research. Meanwhile, an obstacle has emerged since these grids are established for different purposes and work in isolation from each other. Some pilot work [10,11] has already observed that the next natural step is to enable interoperation between multiple grids, in order to serve much larger scientific communities and enhance the overall performance of the joint grids. Regarding nontrivial issues such as bottleneck, single point of failure, and impractical administrative management, which are normally led by conventional centralized or hierarchical design, these could get worse in an inter-operational grid-based infrastructure; the desire for a complete decentralized design has increased dramatically.

Meanwhile, research on the characteristics and performance of groups of jobs in grids [12] has shown that 70% of jobs, which consumed 80% of resource processing time, are submitted in a batch pattern. In other words, most jobs are submitted by specific batch engines with a very short interval time between each other. In this case, users of grids normally submit many jobs as a single batch with a single runtime estimation, instead of specifying estimate processing time for each individual job. Consequently, scheduling algorithms which rely on job estimate processing time, such as shortest job first (SJF), longest job first (LJF), and backfilling variants, are severely affected. Even concerning a centralized grid wherein each node maintains complete global information and is interconnected upon a static and stable network, the widely adopted backfilling series algorithms [13,14] can still lead to complex dynamic problems. Related research [15] has shown jobs requesting short processing time and few processors are likely to find "holes" easier in heavily loaded systems, which makes the prediction of future system performance as static even harder. In this case, a novel scheduling heuristic supporting dynamic rescheduling through time has shown its great importance, which is also covered in this paper.

As the extension to some pilot work [16], we propose a novel decentralized dynamic scheduling approach named the community-aware scheduling algorithm (CASA). In this work, based on the previously proposed two-phase scheduling protocol, a set of heuristic algorithms are designed to efficiently distribute jobs amongst participating nodes without asking for detailed node real-time processing information nor control authorities of remote nodes. The remainder of the paper is organized as follows: the related work in terms of decentralized scheduling is given in Section 2. The problem statement and algorithm principle is presented in Section 3, followed by the detailed discussion of each heuristic algorithm in Section 4. Section 5 introduces the experiment configuration for the algorithm evaluation, whilst Section 6 discusses the results observed in this experiment. Finally, Section 7 presents the conclusion and some insights to future work.

2. Related work

Metacomputing, the term firstly introduced by Smarr and Catlett [17], is widely accepted in the field of grid computing [2] to describe the computational pool formed from resources of different participating nodes. In general, metascheduling solutions are classified into three categories, namely the centralized, hierarchy, and decentralized schemes [18,10,15]. Specifically, the decentralized metascheduling scheme allows each node to own a metascheduler to receive job submissions originated by local users, and to assign such jobs to the local resource management system, i.e., local scheduler, of the node. Meanwhile, metaschedulers of different nodes are capable of exchanging information and sharing jobs between each other in order to balance the resource load amongst participating nodes. The nature of the decentralized scheme brings better scalability compared to other scheduling schemes, but leads to the issue of efficiency and overhead on the other hand.

NWIRE (Net-Wide-Resources) [19] is a brokerage and trading based metacomputing scheduling architecture. On the top logical level, NWIRE consists of a set of MetaDomains, wherein each MetaDomain is controlled by one MetaManager. By taking into consideration several scheduling properties, the NWIRE relies on a market mechanism [20] to trade resources between domains via the MetaManager, wherein a description of the requests and objectives appears to be a key element for the job allocation. In addition, NWIRE provides high flexibility in terms of fault tolerance because the failure of a single trader will not affect the whole metascheduling system.

The K-Distributed and K-Dual Queue Models [15,21] propose a distributed scheduling algorithm which redundantly distributes jobs to different sites simultaneously, instead of only sending jobs to the most lightly loaded sites. The K-Distributed model enables each metascheduler of each site to distribute their jobs to the K least loaded sites, whereby such jobs will be scheduled by all K sites respectively. The K-Dual model works on the K-Distributed model and gives priority to jobs originated by the local sites. Jobs transferred from remote nodes will be executed only if they do not adversely affect the start time of queued jobs which are originated from the local sites.

InterGrid [22] is a cross-grid cooperation architecture composed of a set of InterGrid Gateways (IGGs) responsible for managing peering arrangements between grids. InterGrid promotes interlinking of islands grids through peering arrangements to enable inter-grid resource sharing, and provides a scalable structure allowing grids to interconnect with each other and to grow in a sustainable way. Although the structure of the overall InterGrid ecosystem is hierarchical, the InterGrid Gateways employed upon the top of each participating grid are distributed in a decentralized manner. Each IGG is aware of the agreements with other IGGs, and is capable of enabling resource allocation across multiple grids with pluggable policies. The InterGrid/IGG relies on external decentralized approaches, such as a self-organizing economic model [23], to meet its design objectives, including incentive-oriented peering arrangements, decentralized resource management, and reservation and brokering across grids.

Delegated Matchmaking (DMM) [10] is a decentralized approach for grid inter-operation by temporarily binding resources from remote sites to the local environment. First, the DMM leverages a hierarchical architecture in which nodes represent computing sites, and nodes of the same hierarchical level are allowed to inter-operate to form a completely decentralized network. Second, the DMM employs two independent policies named the *delegation algorithm* and *local requests dispatching policy*, to disseminate resource requests within the established network. By delegating resources instead of the traditional job delegation way, the DMM aims to lower the administrative overhead of managing user/group accounts on each site during the inter-grid operations.

Grid-Federation [24,25] is a metascheduling framework which highlights a bid-based SLA contract negotiation model. Benefitting from the market-based SLA coordination mechanism, the Grid-Federation framework allows resource owners to have finer control over the resource allocation. An SLA is the agreement negotiated between a metascheduler, entitled the Grid Federation Agent (GFA), and the LRMSs of the local sites in terms of acceptable job QoS constraints, such as job response time and budget spent. Furthermore, the contract net protocol [26] based SLA bids are restricted with a certain expiration time, and a variety of economic parameters such as setting price, user budget and deadline. A greedy backfilling heuristic is also proposed for application

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

on the participating LRMSs during their cooperation with the metaschedulers.

Leal et al. [27] present a decentralized model consisting of a set of metaschedulers for scheduling independent tasks in federated grids. Four simple, decoupled and coarse-grained solution algorithms are considered as each metascheduler's mapping strategy. The advantage of this approach is no requirement for information of remote nodes' processing speed nor length of jobs; instead, it only needs information about the past performance of the resources for predicting a new objective. The results obtained from a two-grid based infrastructure have shown that the algorithm DO-AS outperforms other versions in reducing the makespan in all evaluated scenarios.

Wang et al. [28] proposed a dynamic resource selection heuristic for a non-reserved bidding-based grid environment, wherein a set of deterministic and probabilistic resource selection heuristics are evaluated to minimize the job turnaround time in online systems. Five basic heuristics and various levels of information released by resource providers are considered to yield the corresponding variants under different scenarios. The results have shown that heuristic Dissolve-P is superior to the others when information about competitors is not provided; on the other hand, the MCT-D heuristic outperforms when the information of the execution time of competitors is available.

Das et al. [29] introduced a combinatorial auction-based resource allocation protocol in which a user bids a price value for each of the possible combinations of resources required for its tasks' execution. In total, three main participants are involved, namely (1) the User Broker (UB) responsible for resource discovery, generation of combinations of resources, bid and job management; (2) the Grid Service Providers (GSPs) responsible for contributing their resources to the grid and charging the users for services; and (3) the Local Market for Auctions (LMA) supporting GSPs to post their characteristics as well as enabling the users to find the proper resources to fulfill their requirements. In addition, an approximation algorithm developed by Zurel et al. [30] is also involved for solving the combinatorial auction problem.

Unlike all aforementioned solutions, the community-aware scheduling algorithm (CASA) is a two-phase solution which makes job allocation decisions based on contacted nodes' real-time responses. In this case, each participating node does not need to expose its resource processing information, such as status of local job queue and resource utilization, during cooperation with other nodes. Furthermore, the CASA is able to reschedule jobs through time, in order to adapt to the unpredictable performance changes of independent underlying resources.

3. Problem statement and algorithm principle

To achieve the aforementioned objectives in a decentralized distributed grid environment, we propose a novel "phase-by-phase" algorithm named the community-aware scheduling algorithm (CASA). The CASA indicates a collection of implemented interfaces and heuristics used to facilitate job scheduling across decentralized distributed nodes. A variety of variables, such as resource heterogeneity and unpredictable job characteristics, are considered. The design of CASA yields a dynamic, adaptive scheduling algorithm to promote job sharing/execution efficiency and improves the job owners' experience.

Suppose there are in total $N = \{n_1, n_2, \ldots, n_\eta\}$ nodes distributed in a decentralized manner in a grid system with the same uptime T, wherein some of the nodes have local job submission. Each time a node attempts to (re)assign a job to another node (or the same node) for execution, the assignment initiator is called the requester node, and the node receiving such a request is called the responder node. Each job $j_{\alpha,\theta}$ sent from node n_α requests a certain number of processing elements (PEs) $j^{pe}_{\alpha,\theta}$, i.e., CPUs, as well as the estimated execution time $j^{length}_{\alpha,\theta}$, which together represent the weight of the job by: $w_\theta = j^{pe}_{\alpha,\theta} \cdot j^{length}_{\alpha,\theta}$. Furthermore, job $j_{\alpha,\theta}$'s characteristic profile also contains the speed estimation of the processing element $j^{mips}_{\alpha,\theta}$, which facilitates calculation of the job load by: $load_\theta = j^{pe}_{\alpha,\theta} \cdot j^{length}_{\alpha,\theta} \cdot j^{mips}_{\alpha,\theta}$ Considering job local submission distribution and job characteristics are impractical, if not impossible, to predict in a real system. In this case, all the information will be retrieved from the real grid workload trace in our work and is unknown a priori.

Regarding jobs which are imbalanced when submitted to the grid, and stochastic events through time, such as change of job status and resource usages, can make the situation even worse. CASA consists of two phases, namely the `job submission phase` and the `dynamic scheduling phase`, which work together to ensure both a rapid job distribution and an optimized rescheduling process.

Job submission phase. This is the first phase of the community-aware scheduling algorithm. Each time a node $n_\alpha \in N$ receives a job $j_{\alpha,\theta}$ submitted by its local user, node n_α behaves as a requester node and uses algorithm $h_{request}$ to generate a `request` message $req_{\alpha,\theta}$ for job $j_{\alpha,\theta}$. Job characteristic information including estimated execution time $j^{length}_{\alpha,\theta}$ and requested amount of PEs $j^{pe}_{\alpha,\theta}$ will be appended to the generated `request` message. Furthermore, extra job information in terms of heterogeneity, such as the required type of operating system, can also be included within the `request` message $req_{\alpha,\theta}$. Then, an interface $it_{neighbors}(n_\alpha)$ will be invoked by $h_{request}$ on node n_α in order to find a list of contactable remote nodes for the following job delegation attempts. A variety of resource discovery approaches [31] can be used as the implementation of interface $it_{neighbors}(n_\alpha)$, which is out of the scope of CASA itself. Afterwards, `request` message $req_{\alpha,\theta}$ is replicated and disseminated to each of the discovered remote nodes asking for the job delegation possibilities.

For all nodes $\forall n_\beta \in N$ receiving the job delegation `request` message $req_{\alpha,\theta}$, including the requester node n_α itself, are considered as responder nodes. Each responder node n_β needs to launch an algorithm named h_{accept} to decide whether node n_β is willing and able to execute the received job $j_{\alpha,\theta}$. Algorithm h_{accept} takes various factors, such as the responder node's capabilities and administrative preferences, into consideration to decide whether job $j_{\alpha,\theta}$ can be executed upon the responder node. If yes, an `accept[R]` message $acc^R_{\beta,\alpha,\theta}$ will be generated and sent back to the requester node n_α for the job delegation bidding. In addition, the estimated response time if job $j_{\alpha,\theta}$ is executed by node n_β is also appended to the generated `accept[R]` message $acc^R_{\beta,\alpha,\theta}$, which can be utilized by the requester node for responder node evaluation and selection.

Each time a `request` message $req_{\alpha,\theta}$ is generated by the requester node and disseminated to contactable remote nodes, the requester node waits and collects all received `accept[R]` messages and invokes an algorithm h_{assign} to select a proper remote node to which to delegate the job. Since no node has detailed global information in a decentralized distributed grid environment, then if all nodes greedily select a responder node offering the "best job execution condition" (e.g., shortest execution time) but with no reservation service, imbalanced job distribution can result and some nodes will become overheated. In this case, a probabilistic approach is adopted by the algorithm h_{assign}, whereby powerful nodes, e.g., a large number of processing elements or faster processing speed, are prone to receiving and executing more jobs; on the other hand, nodes with less powerful resources can still receive job delegations, therefore working as part of the integrated grid.

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

Table 1
List of notations and terminology.

Symbol	Description
N	Number of nodes of the grid.
T	The uptime of each node of the same grid, also the uptime of the grid as well.
n_α (n_β, n_γ, n_δ)	A node of the grid.
$j_{\alpha,\theta}$	A job sent from node $n_\alpha \in N$ for job delegation.
$j_{\gamma,\psi}$	A job sent from node $n_\gamma \in N$ for job rescheduling.
$j^{pe}_{\alpha,\theta}$	Number of processing elements (PEs) requested by job $j_{\alpha,\theta}$.
$j^{length}_{\alpha,\theta}$	Estimated processing time of job $j_{\alpha,\theta}$.
w_θ	Weight of job $j_{\alpha,\theta}$.
$load_\theta$	Load of job $j_{\alpha,\theta}$.
$t^{interval}$	The interval between two neighboring calls of algorithm $h_{resched}$.
$req_{\alpha,\theta}$	A `request` message generated by node n_α for job $j_{\alpha,\theta}$.
$acc^R_{\beta,\alpha,\theta}$	An `accept[R]` message sent from node n_β to node n_α for accepting a received delegation request for job $j_{\alpha,\theta}$.
$ass_{\alpha,\beta,\theta}$	An `assign` message used to convey job $j_{\alpha,\theta}$ from its requester node n_α to the selected responder node n_β.
$rec_{\beta,\alpha,\theta}$	A `receipt` message used to confirm the requester node n_α with regards to the reception of assigned job $j_{\alpha,\theta}$ on the responder node n_β.
$inf_{\gamma,\psi}$	An `inform` message used to disseminate the information of to-reschedule job $j_{\gamma,\psi}$ to remote nodes.
$acc^I_{\delta,\gamma,\psi}$	An `accept[I]` message sent from node n_δ to node n_γ for accepting a received delegation request for job $j_{\gamma,\psi}$.
$it_{neighbors}(n_\alpha)$	The interface of neighboring nodes discovery service of node n_α.
$h_{request}$	The algorithm used to allocate a local arrival job to a node of the grid by means of a generated `request` message.
h_{accept}	The algorithm used to determine whether a remote job delegation request should be accepted.
h_{assign}	The algorithm used to select a targeting remote node and assign the corresponding job.
$h_{resched}$	The algorithm used to determine whether to reschedule jobs, and how to reschedule.
h'_{accept}	The algorithm used to determine whether a remote job delegation request should be accepted with regards to existing schedule made for the to-schedule job.

Once a proper remote node, e.g., node n_β, is selected by the algorithm h_{assign}, node n_β is considered as an assignee node, an `assign` message $ass_{\alpha,\beta,\theta}$ will be generated and sent to the assignee node, wherein the job $j_{\alpha,\theta}$ and its related data are enclosed.

Dynamic scheduling phase. This is a complementary phase which dedicates facilitation of the community-aware scheduling algorithm to adapt to the uninterrupted changing grid infrastructure. For an arbitrary node $n_\gamma \in N$, due to the effect of the `job submission phase`, it has a set of jobs to execute which are obtained either from local submissions or from remote nodes' delegations; thus, node n_γ uses a local queue to store jobs that cannot be executed instantly. Despite different adopted local scheduling algorithms which can affect the sequence of queued jobs, e.g., the backfilling [13] variants, node n_γ can still identify which jobs are supposed to wait for a long time at a specific instant.

As a federated resource infrastructure, a grid is a naturally dynamic environment wherein resources contributed by different sites can join and leave through time; furthermore, issues like unexpected network delay, resource overhead, and job status modification make the status of a grid even more unpredictable. In this case, an optimal job distribution decision may not remain to be optimized through time. Therefore, the motivation of the `dynamic scheduling phase` is to keep the previous made scheduling decision optimized by allowing queued jobs to be kept rescheduled in accordance with the changes in the grid.

Suppose the arbitrary node $\forall n_\gamma \in N$ checks its local job queue to find several jobs which have the longest waiting time by means of an algorithm entitled $h_{resched}$ periodically with interval $t^{interval}_\gamma$. Such selected jobs will be tried in the following rescheduling process to improve their own job makespans and the resource utilization of the overall grid. The number of to-reschedule jobs is decided by $h_{resched}$ by considering the status of node n_γ itself, such as length of local job queue and current node overhead. Afterwards, interface $it_{neighbors}(n_\alpha)$ will be launched again to find a set of contactable remote nodes for job rescheduling and re-allocation. Algorithm $h_{resched}$ then generates an `inform` message $inf_{\gamma,\psi}$ for each to-schedule job $j_{\gamma,\psi}$ and disseminates those `inform` messages to all remote nodes discovered for negotiating job rescheduling possibilities. Each generated `inform` message $inf_{\gamma,\psi}$ contains similar information to the aforementioned `request` message, including estimated execution time $j^{length}_{\gamma,\psi}$, requested amount of PEs $j^{pe}_{\gamma,\psi}$, and job characteristic profile. Further, each `inform` message also includes the already made schedule for job $j_{\gamma,\psi}$ on current node n_γ, i.e., the estimated job finish time, which will be used for offer comparison by the contacted remote nodes later. It is noteworthy that the algorithm $h_{resched}$ can be triggered by other events as well depending on each node's local setting.

Each time a responder node $n_\delta \in N$ receives an `inform` message $inf_{\gamma,\psi}$, the responder node needs to launch the algorithm h'_{accept} to determine whether to accept this job delegation request or not. Algorithm h'_{accept} works in a similar way with the aforementioned h_{accept} except one crucial difference: regarding the `inform` message $inf_{\gamma,\psi}$ contains the current schedule made for job $j_{\gamma,\psi}$ on the requester node n_γ, the responder node n_δ needs to make sure a worthwhile benefit can be obtained from the job rescheduling action based on the predefined system optimizing objective. Finally, an approved job acceptance decision will lead to the generation of a corresponding `accept[I]` message $acc^I_{\delta,\gamma,\psi}$, wherein the offer provided by the responder node n_δ is also included and sent back the requester node. The requester node n_γ then collects and compares all received `accept[I]` messages for job $j_{\gamma,\psi}$ to launch the probabilistic based h_{assign} algorithm to select an appropriate remote node, and delegates job $j_{\gamma,\psi}$ to the selected remote node by means of a generated `assign` message $ass_{\gamma,\delta,\psi}$.

The notations used and algorithms introduced are listed in Table 1.

4. Community-aware scheduling algorithm

As introduced above and then listed in Table 1, in total 5 heuristic algorithms form the community-aware scheduling algorithm. Each scheduling algorithm is detailed in this section.

4.1. Job distribution

The algorithm for job distribution is represented as $h_{request}$ in Section 3. Each time a node n_α receives a job $j_{\alpha,\theta}$ submitted from its local users, instead of allocating this job to the local resource management system (LRMS) immediately, the hosting node n_α

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

needs to launch the `job submission phase` of CASA to find a proper node from the scope of the overall grid.

Unlike other approaches [32,33] which make instant job delegation decisions depending on information periodically exchanged from all nodes of the grid; the $h_{request}$ first invokes interface $it_{neighbors}$ to fetch an address list of known remote nodes, which is periodically updated by an external resource discovery service employed for this. Later on, the $h_{request}$ sends a job delegation request, which is comprised of information such as job characteristic and number of message replicas, to each known remote node. The decision of job delegation then depends on both the remote nodes' responses and the hosting node's adopted algorithm, which will be discussed in Sections 4.2 and 4.3 respectively.

Algorithm 1 Algorithm for Job Distribution $h_{request}$

Require: n_α: the requester node.
$j_{\alpha,\theta}$: the new arriving job on node n_α, which needs to be allocated to either the local node or a remote node;
$req_{\alpha,\theta}$: the `request` message generated for job $j_{\alpha,\theta}$;
$it_{neighbors}(n_\alpha)$: address list of known remote nodes of hosting node n_α;
x: length of the known remote node list returned by $it_{neighbors}(n_\alpha)$;
n_β: an arbitrary node from list $it_{neighbors}(n_\alpha)$;
Require: *send*: implementation methods.

1: $n_\alpha \rightarrow req_{\alpha,\theta}$
2: $(j^{pe}_{\alpha,\theta}, j^{length}_{\alpha,\theta}, j^{mips}_{\alpha,\theta}) \leftarrow j_{\alpha,\theta}$
3: $req_{\alpha,\theta} \leftarrow (j^{pe}_{\alpha,\theta}, j^{length}_{\alpha,\theta}, j^{mips}_{\alpha,\theta})$
4: $req_{\alpha,\theta} \leftarrow \frac{1}{x}$
5: **for all** $n_\beta \in it_{neighbors}(n_\alpha)$ **do**
6: *send* $req_{\alpha,\theta}$ to node n_β
7: **end for**

4.2. Job delegation request acceptance

The algorithm for job delegation request acceptance is represented as h_{accept} in Section 3. Each node of the grid $n_\beta \in N$ checks the received job delegation requests periodically and filters out jobs whose obligatory requirements, such as the type of operating system and requested number of processing elements, cannot be fulfilled by the local resources. After that, the responder node n_β needs to generate an `accept[R]` message and send the message back to the delegation requester node n_α, together with the expected response time in case job $j_{\alpha,\theta}$ is assigned to node n_β itself.

Algorithm h_{accept} calculates the estimated job response time by adding up the time used to execute the job itself, and the time used to execute already arrived/queued jobs before the job $j_{\alpha,\theta}$. Specifically, each node applying the CASA contains two queues used to host already received jobs and "promises" made. The `local job queue` of a node is the place where jobs have been accepted for local execution but the execution has not started yet. Despite the job sequence of this queue varying through time depending on the adopted local scheduling policy, the total amount of job execution workload of a `local job queue` $Load^{JQ}$ can still be calculated as depicted in Definition 1.

Definition 1 (*Single Node's Workload of Already Queued Jobs ($Load^{JQ}$)*)**.**

$$Load^{JQ}_\beta = \sum_{\forall j_\theta \in J_\beta} (j^{pe}_\theta \cdot j^{length}_\theta \cdot j^{mips}_\theta).$$

Assuming there are in total J_β jobs queued on node n_β, then the load of all queued jobs is the sum of each job's load, namely the product of each job's number of requested processing elements j^{pe}_θ, estimated processing element speed j^{mips}_θ, and estimated job execution time j^{length}_θ.

Besides, each time a node sends a generated `accept[R]` message back to the requester node for job bidding, such a "promise" made will also be queued upon the responder node since an `accept[R]` message represents a possible incoming job delegation in the near future. In this case, each node applying the CASA contains a `shadow job queue` to record the already sent `accept[R]` messages, wherein each record of the `shadow job queue` contains both the job characteristic profile and the estimated `accept[R]` message approval probability. For instance, if a node n_α propagates x `request` messages $req_{\alpha,\theta}$ for job $j_{\alpha,\theta}$ and sends one of those generated `request` messages to a responder node n_β, then when node n_β decides to accept the received job delegation request by means of a response `accept[R]` message $acc^R_{\beta,\alpha,\theta}$, the estimated message approval probability is: $\frac{1}{x}$, thus the total amount of job execution work of a `shadow job queue` $Load^{SQ}$ is:

Definition 2 (*Single Node's Estimated Workload of Accepted Jobs ($Load^{SQ}$)*)**.**

$$Load^{SQ}_\beta = \sum_{\forall j_\theta \in J'_\beta} (j^{pe'}_\theta \cdot j^{length'}_\theta \cdot j^{mips'}_\theta) \cdot \frac{1}{x_\theta}.$$

Assuming there are in total J'_β sent `accept[R]` messages queued on node n_β, then the estimated load of all corresponding jobs is the sum of each job's load with its approval probability, namely the product of each job's number of request processing elements $j^{pe'}_\theta$, estimated processing element speed $j^{mips'}_\theta$, estimated job execution time $j^{length'}_\theta$, and the `accept[R]` message approval probability $\frac{1}{x_\theta}$.

Afterwards, as depicted in Algorithm 2, the estimated job response time of a received job delegation request can be obtained by adding up the time to execute the job itself, the time to execute load of already queued jobs, and the time to execute estimated load of already accepted jobs by the algorithm h_{accept}. The responder node then generates an `accept[R]` message with calculated response time and sends it back to the job delegation requester node, where the job delegation decision will be made later.

It is noteworthy that the estimated time to execute queued/accepted jobs is a theoretical optimized value. In reality, regarding the amount and scale of "scheduling holes" significantly relying on the characteristic of received jobs as well as adopted local scheduling policies, such estimated time is not equal to the required real time. However, on the other hand, the estimated time for executing queued/accepted jobs of a responder node balances the node's current load as well as its processing capability, which gives the requester node the leverage to distribute jobs with regard to remote node load balance. Furthermore, the `accept[R]` message generation procedure upon a responder node is decoupled from the node's local scheduling policies, which makes the implementation of CASA in cooperation with variety of locally adopted scheduling algorithm viable.

4.3. Job assignment

The algorithm for job assignment is represented as h_{assign} in Section 3. Each time a `request` message or an `inform` message is replicated and disseminated to a set of remote nodes, the requester node n_α needs to check the received `accept` messages from diverse remote nodes in the next CASA execution cycle and selects an appropriate one to receive the job assignment. In case no

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

Algorithm 2 Algorithm for Job Delegation Request Acceptance h_{accept}

Require: n_α: the requester node.
n_β: the responder node.
$req_{\alpha,\theta}$: a `request` message sent from requester node n_α to responder node n_β.
$j_{\alpha,\theta}$: the job associated with `request` message $req_{\alpha,\theta}$.
$load_\beta^{jq}$: load of the `local job queue` of responder node n_β.
$load_\beta^{sq}$: estimated load of the `shadow job queue` of responder node n_β.
$acc^R_{\beta,\alpha,\theta}$: an `accept[R]` message generated by the responder node n_β for answering the incoming `request` message $req_{\alpha,\theta}$.
$mips_\beta$: average processing element speed of node n_β.
pe_β: number of processing elements of node n_β.
$t^{job}_{\beta,\theta}$: estimated time to execute job $j_{\alpha,\theta}$ on node n_β.
t^{jq}_β: estimated time to execute load of `local job queue` of node n_β.
t^{sq}_β: estimated time to execute load of `shadow job queue` of node n_β.
SQ_β: the `shadow job queue` of responder node n_β.
Require: *send*: implementation methods.

1: $j_{\alpha,\theta} \leftarrow req_{\alpha,\theta}$
2: $(t^{job}_{\beta,\theta}, t^{jq}_\beta, t^{sq}_\beta) \leftarrow j_{\alpha,\theta}$
3: $t^{job}_{\beta,\theta} = \frac{j^{pe}_\theta \cdot j^{length}_\theta \cdot j^{mips}_\theta}{mips_\beta \cdot pe\beta}$
4: $t^{jq}_\beta = \frac{load^{jq}_\beta}{mips_\beta \cdot pe\beta}$
5: $t^{sq}_\beta = \frac{load^{sq}_\beta}{mips_\beta \cdot pe\beta}$
6: $n_\beta \rightarrow acc^R_{\beta,\alpha,\theta}$
7: $acc^R_{\beta,\alpha,\theta} \leftarrow (t^{job}_{\beta,\theta}, t^{jq}_\beta, t^{sq}_\beta)$
8: $SQ_\beta \leftarrow acc^R_{\beta,\alpha,\theta}$
9: *send* $acc^R_{\beta,\alpha,\theta}$ to node n_α

`accept` message is sent back from any remote node nor the local node, then the job $j_{\alpha,\theta}$ appended to a `request/inform` message is considered as a task not suited to known resources of the grid and thus suspended.

Once a set of `accept` messages have been received, algorithm h_{assign} of the requester node n_α needs to select a responder node offering a shorter response time as the assignee node, and delegates job $j_{\alpha,\theta}$ to such a selected assignee node for execution. On the other hand, if all nodes of the grid greedily select responder nodes offering the shortest job response time for job assignments, then nodes with either sufficient processing elements or superior processing capability could be simultaneously selected as the assignee nodes and receive an imbalanced amount of jobs within one CASA execution cycle. In order to avoid the "punishment" effect on nodes with better processing power, algorithm h_{assign} adopts a probabilistic approach, wherein nodes with better power are prone to have a chance of receiving more jobs, but nodes with less power still have the probability of being selected as the assignee nodes for job delegations.

As shown in Definition 3, during CASA's `job submission phase` or `dynamic scheduling phase`, a responder node n_β's probability of being selected as the assignee node $\rho^{req}_{ass}(\beta)$ is determined by its estimated job response time when compared with `accept` messages issued by other responder nodes. Furthermore, since the estimated time for job $j_{\alpha,\theta}$'s execution upon node n_β is a deterministic value, whereas the estimated time for already queued/accepted jobs' execution is an empirical and underlying algorithm dependent value, a coefficient $\omega \in [0, 1]$ could be employed to weigh those two parts respectively depending on the setting of participating nodes.

Definition 3 (*Responder Node's Probability of Being the Assignee Node* (ρ^{req}_{ass}))**.**

$$\rho^{req}_{ass}(\beta) = \frac{\frac{1}{t^{job}_{\beta,\theta} + t^{jq}_\beta + \omega \cdot t^{sq}_\beta}}{\sum_{\forall \psi \in M} \left(\frac{1}{t^{job}_{\psi,\theta} + t^{jq}_\psi + \omega \cdot t^{sq}_\psi}\right)}.$$

Assuming after sending job $j_{\alpha,\theta}$ based `request/inform` messages to several remote nodes, the requester node n_α receives in total M `accept` messages from M diverse responder nodes. For each responder node n_β, the estimated response time of job $j_{\alpha,\theta}$ upon its local resources is the sum of time $t^{job}_{\beta,\theta}$, t^{jq}_β, and t^{sq}_β as discussed in Section 4.2. Then node n_β's probability of being selected as the assignee node for job $j_{\alpha,\theta}$ (by the requester node n_α) is then $\rho^{req}_{ass}(\beta)$.

When all responder nodes' selection probabilities have been calculated, the requester node n_α launches a random number generator in order to select an assignee node from all responder nodes with regard to their selection probabilities. Afterwards, job $j_{\alpha,\theta}$ is conveyed to the selected assignee node by means of a newly created `assign` message. In addition, the requester node n_α also needs to contact other responder nodes, which are not selected as the assignee node, to revoke the corresponding job `accept` messages from their own `shadow job queues` respectively.

It is noteworthy that the assignee node is selected depending on responder nodes' estimated job response time by means of the returned `accept` messages. The responder nodes do not need to send confidential information such as current resource utilization or current length of local job queue back to the requester node to bid for job delegation; this is a crucial criterion for decentralized metascheduling in reality because all participating nodes are independent and may even have competition relationships with each other.

Furthermore, as discussed in Section 4.2, a responder node's job response time is an averaged theoretical value calculated based on its estimated workload regardless of the adopted local scheduling algorithm. However, nodes with advanced local scheduling algorithms are able to execute more queued and newly assigned jobs during the interval between two CASA execution cycles, thus representing themselves as nodes with higher probabilities of getting job assignments due to less instant load and shorter job response time in the next CASA execution cycle. In this case, without knowing participating nodes' local settings, the CASA can still highlight nodes with advanced local scheduling algorithms and promote their job assignment probabilities.

4.4. Job rescheduling

The algorithm for job rescheduling is represented as $h_{resched}$ in Section 3. Once a node is selected as the assignee node for a delegated job's execution, numerous events such as newly arrived jobs and resources are able to keep changing the status of the overall grid dynamically through time. Therefore, if the assigned job is still not executed yet after some time, the current assignee node may not remain an optimal choice for the assigned job's execution because some other nodes could offer a shorter job response time because of their volatile resource usages. In this case, each node of the grid needs to check periodically whether some of the already queued jobs could be re-assigned to other nodes for obtaining better performance in terms of job execution and grid resource utilization.

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

Algorithm 3 Algorithm for Job Assignment h_{assign}

Require: ω: weight of estimated time to execute jobs of the shadow job queue, default value: 1.
Require: n_α: the requester node.
JQ_α: the `local job queue` of node n_α.
$j_{\alpha,\theta}$: the job requested for delegation by requester node n_α.
$msg_{\alpha,\theta}$: the `request/inform` message used to send delegation request for job $j_{\alpha,\theta}$.
M: all responder node answering the delegation request for job $j_{\alpha,\theta}$.
n_β: one arbitrary responder node, $n_\beta \in M$.
$acc_{\beta,\alpha,\theta}$: the `accept` message (either `accept[R]` or `accept[I]`) sent by responder node n_β.
$t^{job}_{\beta,\theta}$: estimated time to execute job $j_{\alpha,\theta}$ on node n_β.
t^{jq}_β: estimated time to execute load of `local job queue` of node n_β.
t^{sq}_β: estimated time to execute load of `shadow job queue` of node n_β.
$\rho^{req}_{ass}(\beta)$: node n_β's probability of being selected as the assignee node by the requester node.
n_φ: the selected assignee node for job $j_{\alpha,\theta}$'s delegation.
$ass_{\alpha,\varphi,\theta}$: the `assign` message used to deliver job $j_{\alpha,\theta}$ from the requester node n_α to the assignee node n_φ.
$f_{selector}$: the random number generator to select the assignee node with regard to each known responder node's selection probability.
Require: *send*, *revoke*: implementation methods.

```
1:  j_{α,θ} ← msg_{α,θ}
2:  if j_{α,θ} ∈ JQ_α then
3:    for all n_β ∈ M do
4:      (j_{α,θ}, t^{job}_{β,θ}, t^{jq}_β, t^{sq}_β) ← acc_{β,α,θ}
5:      ρ^{req}_{ass}(β) ← (t^{job}_{β,θ}, t^{jq}_β, t^{sq}_β, ω)
6:      f_{selector} ← ρ^{req}_{ass}(β)
7:    end for
8:    n_φ ← f_{selector}
9:    for all n_β ∈ M do
10:     if n_φ = n_β then
11:       n_α → ass_{α,φ,θ}
12:       ass_{α,φ,θ} ← j_{α,θ}
13:       send ass_{α,φ,θ} to n_φ
14:       JQ_φ ← j_{α,θ}
15:     end if
16:     revoke acc_{β,α,θ} in n_β
17:   end for
18: else
19:   for all n_β ∈ M do
20:     revoke acc_{β,α,θ} in n_β
21:   end for
22: end if
```

To make scheduling decisions adapt to changes of grid resource usages, some already queued jobs on each participating node need to be selected appropriately and re-assigned somewhere else. First, each node of the grid $n_{\gamma\in N}$ needs to constantly update its average queuing time $T^{queuing}_{avg}(\gamma)$ (shown in Definition 4) based on jobs already executed by a local resource. Second, for each unexecuted job $j_{\gamma,\psi}$ waiting within node n_γ's `local job queue`, its queuing time will be used to divide node n_γ's up-to-date average queuing time to generate such a job's relative queuing delay $\nu_{\gamma,\psi}$ (shown in Definition 5). If $\nu_{\gamma,\psi}$ is greater than a predefined system coefficient μ, job $j_{\gamma,\psi}$ is then considered as having waited for a long enough time and needs to be rescheduled if possible. All to-reschedule jobs of node n_γ are put into a temporary to-schedule array ϱ_γ ordered according to the descending order of their relative queuing delays.

Definition 4 (*Single Node's Average Queuing Time* ($T^{queuing}_{avg}$))**.**

$$T^{queuing}_{avg}(\gamma) = \frac{\sum_{\forall j_\psi}^{\iota^{JQ}_\gamma} t^{queue}_{\gamma,\psi}}{\iota^{JQ}_\gamma}, \quad j_\psi \in JQ_\gamma \text{ when } \iota^{JQ}_\gamma \neq 0$$

$$T^{queuing}_{avg}(\gamma) = \frac{\sum_{\forall j_\psi}^{\iota^{JC}_\gamma} t^{queue}_{\gamma,\psi}}{\iota^{JC}_\gamma}, \quad j_\psi \in JC_\gamma \text{ when } \iota^{JQ}_\gamma = 0 \& \iota^{JC}_\gamma \neq 0$$

$$T^{queuing}_{avg}(\gamma) = 0, \quad \text{when } \iota^{JQ}_\gamma = 0 \& \iota^{JC}_\gamma = 0.$$

Assuming there are in total ι^{JC}_γ executed jobs stored in node n_γ's job completion queue JC_γ; Meanwhile, there are in total ι^{JQ}_γ jobs waiting in node n_γ's local job queue JQ_γ. Each job j_ψ's queuing time upon node n_γ is $t^{queue}_{\gamma,\psi}$. Then node n_γ's average queuing time $T^{queuing}_{avg}(\gamma)$ can be calculated according to different queue conditions.

Definition 5 (*Single Node's Relative Queuing Delay* (ν))**.**

$$\nu_{\gamma,\psi} = \frac{t^{queue}_{\gamma,\psi}}{T^{queuing}_{avg}(\gamma)}, \quad j_\psi \in JQ_\gamma \text{ when } T^{queuing}_{avg}(\gamma) \neq 0$$

$$\nu_{\gamma,\psi} = 0, \quad \text{when } T^{queuing}_{avg}(\gamma) = 0.$$

For each unexecuted job j_ψ waiting to be executed in node n_γ's local job queue JQ_γ, its instant relative queuing delay $\nu_{\gamma,\psi}$ can be calculated by dividing its current queuing time $t^{queue}_{\gamma,\psi}$ by the node's up-to-date average queuing time $T^{queuing}_{avg}(\gamma)$.

Next, a set of `inform` messages $inf_{\gamma,\psi}$ will be generated for each to-reschedule job $j_{\gamma,\psi}$ and disseminated to known remote nodes discovered by interface $it_{neighbors}$ for job re-assignment negotiation. To facilitate the responder nodes' decision whether a better job execution offer can be provided, two more associated values are appended to the `inform` message, namely the average queueing time of the requester node $T^{queuing}_{avg}(\gamma)$, and the estimated response time of job $j_{\gamma,\psi}$ on node n_γ.

With regard to jobs continually rescheduled from one node to another, this could make the load of the grid unstably imbalanced, and sometimes lead to a "ping-pong" or even "deadlock" phenomena. In this case, the design of the algorithm for job rescheduling $h_{resched}$ only picks jobs which have been waiting for a long enough time on the hosting node for rescheduling consideration. Such a mechanism gives each just arrived job a period of "cooling down" rather than being rescheduled immediately. The threshold of rescheduling depends on the system coefficient μ.

The detail of job rescheduling is illustrated in Algorithm 4.

4.5. Job rescheduling request acceptance

The algorithm for job rescheduling request acceptance is represented as h'_{accept} in Section 3. It works in a similar way to the algorithm for job delegation request acceptance h_{accept} as discussed in Section 4.2. Once a node $n_\delta \in N$ receives a set of job rescheduling requests from their own requester nodes within a CASA execution cycle, responder node n_δ first filters out jobs which cannot be served by local resources, and then estimates the response time for each remaining job $j_{\gamma,\psi}$ by adding up the estimated time to execute job $j_{\gamma,\psi}$, the time to execute load of already queued jobs on node n_δ, and the time to executed estimated load of already accepted jobs by node n_δ. Furthermore, responder node n_δ needs to update its job

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

Algorithm 4 Algorithm for Job Rescheduling $h_{resched}$

Require: μ: system coefficient of rescheduling threshold.
Require: n_γ: the requester node of job rescheduling.
JQ_γ: the `local job queue` of requester node n_γ.
$j_{\gamma,\psi}$: an arbitrary job obtained from the corresponding queue of node n_γ.
ϱ_γ: the temporary to-schedule job array of node n_γ.
$T_{avg}^{queuing}(\gamma)$: node n_γ's job average queuing time.
$t_{\gamma,\psi}^{queue}$: job $j_{\gamma,\psi}$'s queuing time on node n_γ.
$v_{\gamma,\psi}$: job $j_{\gamma,\psi}$'s relative queuing delay on node n_γ.
$inf_{\gamma,\psi}$: the `inform` message used to send re-assignment delegation request for job $j_{\gamma,\psi}$.
$it_{neighbors}(n_\gamma)$: address list of remote nodes discovered for node n_γ.
n_δ: an arbitrary node from list $it_{neighbors}(n_\gamma)$.
$t_\gamma^{jq,\psi}$: estimated time to execute load of n_γ's `local job queue` before job $j_{\gamma,\psi}$.
$t_{\gamma,\psi}^{job}$: estimated time to execute job $j_{\gamma,\psi}$ on node n_γ.

Require: *send*: implementation methods.

1: $n_\gamma \rightarrow T_{avg}^{queuing}(\gamma)$
2: $index = 0$
3: **for all** $j_{\gamma,\psi} \in JQ_\gamma$ **do**
4: $t_{\gamma,\psi}^{queue} \leftarrow j_{\gamma,\psi}$
5: $v_{\gamma,\psi} \leftarrow (t_{\gamma,\psi}^{queue}, T_{avg}^{queuing}(\gamma))$
6: **if** $v_{\gamma,\psi} \geq \mu$ **then**
7: $\varrho_\gamma \leftarrow j_{\gamma,\psi} j$
8: **end if**
9: **end for**
10: **for all** $j_{\gamma,\psi} \in \varrho_\gamma$ **do**
11: $n_\gamma \rightarrow inf_{\gamma,\psi}$
12: $inf_{\gamma,\psi} \leftarrow T_{avg}^{queuing}(\gamma)$
13: $inf_{\gamma,\psi} \leftarrow (t_\gamma^{jq,\psi}, t_{\gamma,\psi}^{job}) \leftarrow j_{\gamma,\psi}$
14: **for all** $n_\delta \in it_{neighbors}(n_\gamma)$ **do**
15: *send* $inf_{\gamma,\psi}$ to node n_δ
16: **end for**
17: **end for**

average queuing time $T_{avg}^{queuing}(\delta)$, which is later used to determine the answer of job $j_{\gamma,\psi}$'s rescheduling request.

To ensure a better offer can be provided, when compared with job $j_{\gamma,\psi}$'s current schedule on its requester node n_γ, the responder node n_δ needs to make sure: (i) the estimated response time of executing job $j_{\gamma,\psi}$ on its own resource is shorter than job $j_{\gamma,\psi}$'s estimated response time on the requester node; (ii) the weighted average job queuing time of the responder node is shorter than such time of the requester node, i.e., the current job assignee node. The weight ϖ here is a non-negative number, whereby its default value 1 represents that a job will not be rescheduled until the rescheduling bidding node's average job queuing time is not less than the current assignee node's. Once both prerequisites can be fulfilled, an `accept[I]` message $acc_{\delta,\gamma,\psi}^{I}$ will generated and sent back to the requester node for job bidding. The same algorithm for job assignment h_{assign} as discussed in Section 4.3 is then used by the requester nodes for returned `accept[I]` messages evaluation and job assignment.

The detail of job rescheduling request acceptance is illustrated in Algorithm 5.

Algorithm 5 Algorithm for Job Rescheduling Request Acceptance h_{accept}'

Require: ϖ: the weight of average queuing time comparison during the process of job rescheduling, default value: 1.
Require: n_γ: the requester node of job rescheduling.
n_δ: the responder node of job rescheduling.
$j_{\gamma,\psi}$: the job expected to be rescheduled from node n_γ.
$inf_{\gamma,\psi}$: the `inform` message sent by requester node for job $j_{\gamma,\psi}$'s rescheduling delegation request.
$T_{avg}^{queuing}(\gamma)$: node n_γ's job average queuing time.
$t_\gamma^{jq,\psi}$: estimated time to execute load of n_γ's `local job queue` before job $j_{\gamma,\psi}$.
$t_{\gamma,\psi}^{job}$: estimated time to execute job $j_{\gamma,\psi}$ on node n_γ.
$T_{avg}^{queuing}(\delta)$: node n_δ's job average queuing time.
$t_{\delta,\psi}^{job}$: estimated time to execute job $j_{\gamma,\psi}$ on node n_δ.
t_δ^{jq}: estimated time to execute load of `local job queue` of node n_δ.
t_δ^{sq}: estimated time to execute load of `shadow job queue` of node n_δ.
$acc_{\delta,\gamma,\psi}^{I}$: an `accept[I]` message generated by the responder node n_δ for answering the incoming `inform` message $inf_{\gamma,\psi}$.
SQ_δ: the `shadow job queue` of responder node n_δ.

Require: *send*: implementation methods.

1: $(T_{avg}^{queuing}(\gamma), t_\gamma^{jq,\psi}, t_{\gamma,\psi}^{job}) \leftarrow inf_{\gamma,\psi}$
2: $(T_{avg}^{queuing}(\delta), t_{\delta,\psi}^{job}, t_\delta^{jq}, t_\delta^{sq}) \leftarrow n_\delta$
3: **if** $(t_{\delta,\psi}^{job} + t_\delta^{jq} + t_\delta^{sq}) < (t_\gamma^{jq,\psi} + t_{\gamma,\psi}^{job})$ **then**
4: **if** $T_{avg}^{queuing}(\delta) \cdot \varpi < T_{avg}^{queuing}(\gamma)$ **then**
5: $n_\delta \rightarrow acc_{\delta,\gamma,\psi}^{I}$
6: $acc_{\delta,\gamma,\psi}^{I} \leftarrow (t_{\delta,\psi}^{job} + t_\delta^{jq} + t_\delta^{sq})$
7: $SQ_\delta \leftarrow acc_{\delta,\gamma,\psi}^{I}$
8: send $acc_{\delta,\gamma,\psi}^{I}$ to node n_γ
9: **end if**
10: **end if**

5. Experiment configuration

To demonstrate the benefit of applying the community-aware scheduling algorithm (CASA) to schedule jobs upon a fully distributed heterogeneous resource pool, the scenarios with default CASA coefficients and different underlying information systems are compared with two reference conventional scheduling scenarios; thus in total four scenarios are evaluated.

5.1. Scenarios

As mentioned above, in total four scenarios are selected in this experimental work. On the one hand, the *Ind* and *Cen* scenarios are the reference scenarios which represent the conventional local scheduling and centralized metascheduling contexts respectively. In this regard, the observations of such reference scenarios reveal the optimal results in the present computational ecosystem. On the other hand, the *Dec-G* and *Dec-P* scenarios indicate the output of CASA upon decentralized distributed nodes, so that the benefits of the proposed algorithms can be observed and compared with the conventional approaches. Specifically, the *Dec-G* scenario discusses the behavior of CASA with an information system with global knowledge, which is a common choice in the reality; by contrast, the *Dec-P* details the behavior of CASA upon a information system with partial knowledge, which is more scalable and robust in a variety of resource infrastructures.

Ind. The independent scheduling scheme is the first concerned reference scenario wherein each participating node of the grid receives the same amount of jobs submitted by the local users. Because all nodes are isolated from each other, received jobs can only be handled and processed by the local node's resource

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

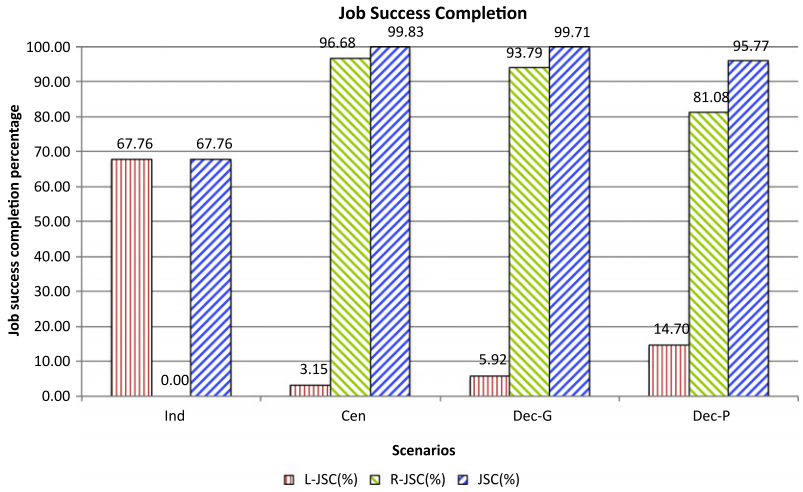

Fig. 1. Job success completion rate comparison between different scheduling schemes.

management system (LRMS) with regard to the adopted underlying local scheduling algorithms.

Cen. The centralized scheduling scheme is the second reference scenario, wherein the BestFit [34], a widely adopted scheduling policy which has been proven to be efficient in both research [35,18] and industry [36] scenarios, is utilized as the built-in metascheduling algorithm. In this scenario, all jobs are submitted to a unique centralized metascheduler which is aware of the detailed information of all nodes of the grid, including each node's number of processing elements (PEs), as well as each node's the number of instantly available PEs at any given time. To respond to the arrival of a newly submitted job, the centralized metascheduler checks the overall grid to build a collection comprised of all nodes that are able to execute such a job instantly. Afterwards, the node leaving the least free PEs, if such a job is executed there, is selected and will receive the job delegation from the metascheduler later. The BestFit algorithm is known for its capability to promote job response time; the centralized scheduling scheme is considered as the optimized scenario with regard to job related criteria, such as job response time, job waiting time and job slowdown.

Dec-G. Within the decentralized scenario, the metascheduler of each individual node is running the CASA with its default coefficients, wherein both the weight of estimated time to execute jobs of the `shadow job queue` (discussed in Algorithm 3) and the weight of node's average queuing time for job rescheduling (discussed in Algorithm 5) is 1.0. Furthermore, the relative queuing delay threshold for rescheduling (discussed in Algorithm 4) is set to 1.0 as well, which means jobs waiting in each node's `local job queue` longer than the average queueing time will be rescheduled if possible.

In addition, each participating node adopts a "global" resource discovery service which is able to return the addresses of all remote nodes of the grid for job delegation activities. It is noteworthy that the "global" resource discovery service is fundamentally different from the foregoing centralized scheme because the "global" service only enables a node to be aware of the existence of as many remote nodes as possible. Neither detailed information such as number of PEs of a remote node, nor any control authority upon the remote node will be conveyed, therefore the hosting node needs to rely on the CASA for the following job delegation behaviors.

Dec-P. This is similar to the aforementioned *Dec-G* scenario, while each node of the grid adopts a different "partial" resource discovery service. Each time a node launches a resource discovery action, the "partial" service is able to return the addresses of six randomly selected remote nodes, and sends them back to the node for job delegation related activities. In this case, each node employing the "partial" service has an incomplete "local perspective" with regard to nodes distributed in the grid.

5.2. Simulation setup

The workload trace archive and resource deployment topology of the Grid5000 [37] is selected to organize the experiment of this work. To obtain stable results as a mean of ten iterations, we took 1% of the overall input jobs (i.e., 10,201 jobs) averagely distributed within the original workload running time, and submit them to the simulated infrastructure (i.e., 26 nodes) with the same frequency within 1% of original job arrival duration. Each simulated job specifies the amount of required processing element as well as the estimated processing time, which can both be obtained from the workload trace.

In line with the underlying assumptions, metascheduling takes places over a pool of heterogeneous resources, therefore each grid node is characterized by a different profile. In this implementation, each node's profile is set by the installed operating system, which is chosen from a simplified distribution generated according to the list published on the TOP500 supercomputing site [38]. Specifically, we assume 80% of the grid nodes have Linux as the installed operating system, and the other 20% of nodes have Windows as the installed operation system. Meanwhile, it is also assumed that each job is also characterized by the profile of requested operating system which is chosen from the same distribution.

Besides, all jobs are fairly submitted to each node of the grid, thus each node receives the same number of local submitted jobs. Finally, the local resource management system (LRMS) of each participating node adopts the First-Come-First-Service (FCFS) as the local scheduling algorithm.

6. Results

The MaGateSim [39] is selected as the simulator for experiment evaluation, and the observed results are then discussed in this section.

6.1. Job success completion

As illustrated in Fig. 1, when the independent scheduling scheme is concerned (scenario:Ind), if a submitted job requires an operating system different from the one installed on the arrival node, such a job cannot be delegated to any remote nodes and

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

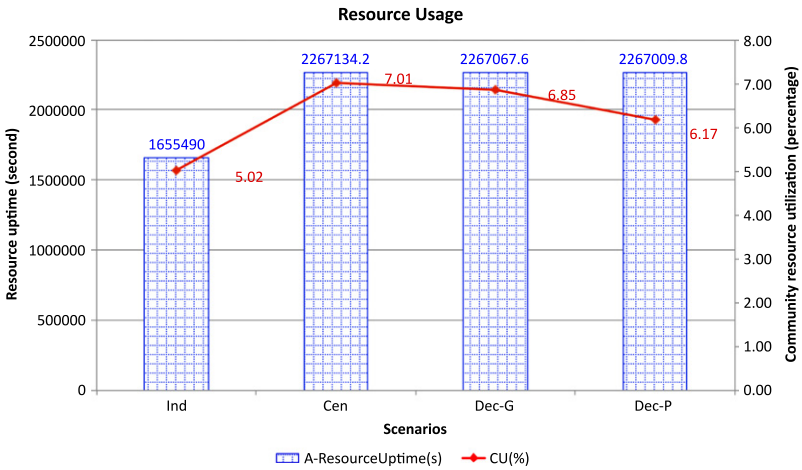

Fig. 2. Resource usage comparison between different scheduling schemes.

has to be suspended and later marked as failed. In this case, all successfully executed jobs are accomplished by their arrival nodes, and the overall job success completion is 67%. When the centralized scheduling scheme is evaluated (scenario:Cen), the centralized scheduler has full control of every participating node and is capable of assigning a received job to any competent node, if it exists. Therefore, over 99% of submitted jobs are successfully executed, wherein about 3% jobs are executed by the node hosting the centralized scheduler.

With regard to the decentralized scheme on an information system with "global perspective" (scenario:Den-G), each node is dedicated to finding an appropriate node for each arrival job during the `job submission phase`. Because the adopted "global" resource discovery service is able to find the addresses of all nodes of the grid, as far as there exists a competent node for a specific job, such a job can be executed sooner or later. In this case, the rate of job success completion of the decentralized scheme is also over 99%, wherein about 6% jobs are executed by their arrival nodes.

By contrast, the decentralized scheme upon an information system with "local perspective" (scenario:Den-P) leads to a job success completion rate around 95%. This is because the employed "partial" resource discovery service can only provide at most six remote candidate nodes each time for job delegation related requests; therefore there exist more unprocessed jobs because neither proper local resource nor remote nodes can be found for job execution. Meanwhile, the original job arrival node has a higher probability of being selected as the assignee node out of a total of 7 candidate nodes, from the scope of the overall grid, around 14.7% jobs are accomplished by the resource of arrival nodes.

6.2. Resource usage

The manner of job assignment to each node's local resource management system (LRMS) in the independent scheduling scenario is rather straightforward. As discussed above, more than 32% submitted jobs are not executed by any resource in the independent scenario, as shown in Fig. 2; the average resource uptime of the independent scheduling scenario (scenario:Ind) is lower than both the centralized (scenario:Cen) and decentralized (scenario:Dec-G, Dec-P) scenarios, wherein over 99% of jobs are executed. Besides, the community resource utilization of the of the independent scenario is about 5%, which is also lower than the resource utilization of the centralized and decentralized scenarios. This is because both the centralized and decentralized scenarios are capable of allocating a job to a resource which could provide a shorter job response time, which in turn promotes the utilization of the local resources.

On the other hand, the average resource uptime and the utilization of the decentralized scenario with "global perspective" information system is almost the same as the centralized scenario, which demonstrates that dynamic scheduling in a decentralized manner with CASA can promote resource usage as good as the Best-Fit based centralized scheduling approach. The metascheduler of the centralized scenario requires all instant detailed information and control authority of each participating node in order to ensure the schedule made for each arrival job is optimized and will not be denied by the responding node. On the contrary, the community-aware scheduling algorithm (CASA) provides the same level of resource utilization in a decentralized manner without asking for detailed resource information nor control authorities from the responding nodes, thus leaving great flexibility and autonomy to the participating nodes.

Furthermore, when the "partial" service is adopted (scenario: Dec-P), the obtained resource utilization is slighted decreased to about 6%. This phenomenon is caused by fewer executed jobs, while "partial" resource discovery service is employed. In addition, it can also be observed that the resource uptime of different scenarios is even as well, which means that using the CASA with a less qualified resource discovery service with local perspective does not lose much effectiveness when compared to an information system with global perspective.

It is noteworthy that as far as there are still scheduled but unfinished jobs remaining in the grid, all nodes of the grid are considered to be in run status. Therefore, if there exist jobs with a long execution time at the end of queues of some nodes, other nodes of the grid need to wait for a long period until the last job is executed, which is the case in this experiment. For instance, with regard to the independent scheduling scenario, as shown in Fig. 3, since about 17th May, the instant community resource utilization (CE) has dropped to less than 2.5%, which means most jobs submitted to the grid are either successfully executed or suspended and considered as failed, thus most nodes are idle. However, the remaining jobs on the grid take about 7 days to be executed, which keeps the status of all nodes of the grid active for another 7 days and dramatically brings down the performance in terms of resource uptime and resource utilization. In this case, the performance benefit brought by the CASA cannot be obviously illustrated by resource related criteria, but will be better presented by job related criteria, which will be discussed in the next subsection.

6.3. Job response time, job waiting time and job slowdown

The job response time represents the time used between a job's arrival time until its returned time. As shown in Fig. 4, the

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

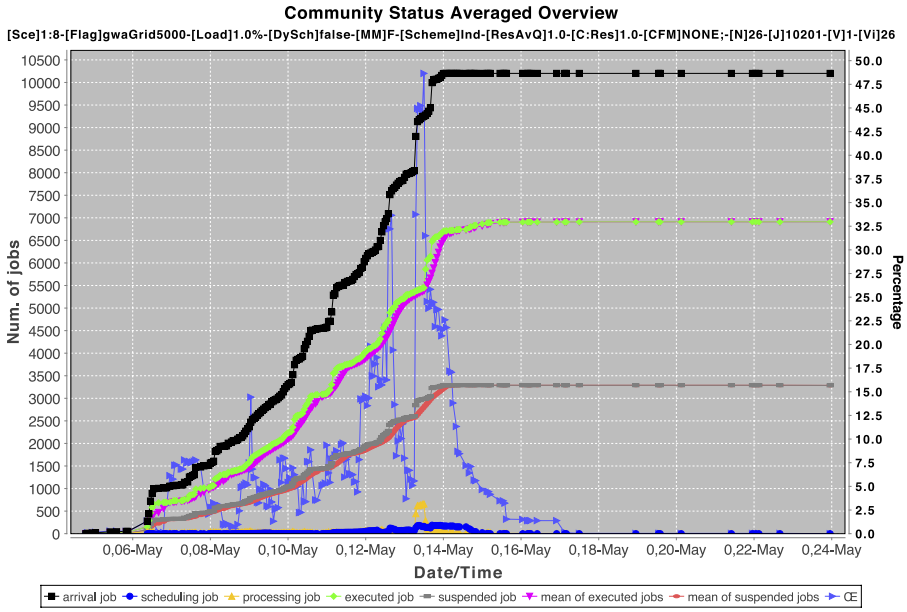

Fig. 3. Average job running status archive of the grid using the independent scheduling scheme. This is the simulator graphic archive after executing the independent scheduling scenario ten times, wherein the counts of some job states through time from all experimental iterations are recorded. Furthermore, the mean of some aforementioned counts are plotted as well, such as the "mean of executed jobs" and the "mean of suspended jobs". It can be observed that execution of the experiment is rather stable because the counts of job states from all iteration are compactly recorded, and the plotted means show little deviation from the original count records.

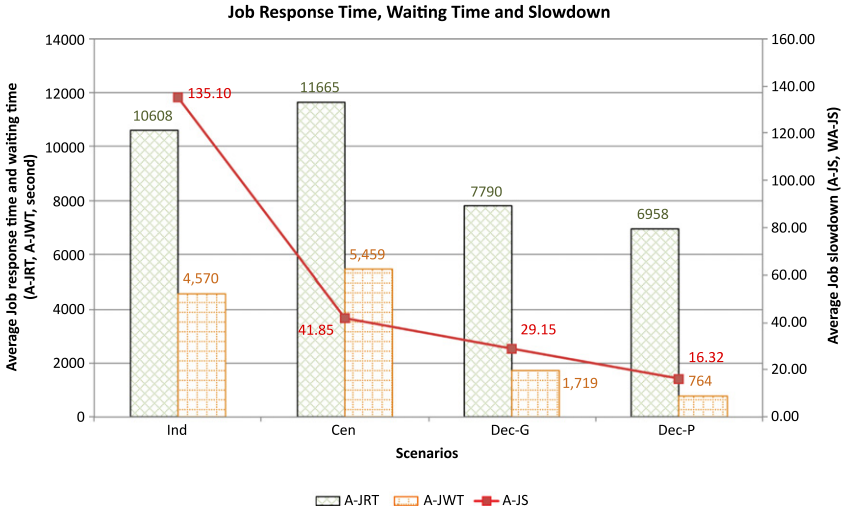

Fig. 4. Job response time and slowdown comparison between different scheduling schemes.

average job response time of the independent scheduling scenario (scenario:Ind) is 10,608 s, which is a little less than the average job response time of the centralized scenario (scenario:Cen). This is because over 32% of jobs of the independent scenario are suspended and then considered as failed without execution attempts due to various reasons, such as the expected number of PEs cannot be provided by the arrival node's underlying resource. In other words, many "heavy" jobs which normally require a large number of PEs and a long period of processing time are not executed, thus the average job response time is brought down because the average job processing time is reduced, instead of jobs being efficiently scheduled. This is proven by the value of average job slowdown and average job waiting time.

The job slowdown illustrates the ratio when comparing a job's response time with its actual processing time. Since the average job slowdown of the centralized scenario is 135, it means that averagely each job submitted to the independent scenario is delayed for 135 times before execution and return. By contrast, the average job slowdown of the centralized scenario is around 41, which is dramatically improved because the Best-Fit based centralized metascheduler is aware of the existence as well as the detailed information of all nodes of the grid, which facilitates making optimal scheduling decisions for each arrival job. The job waiting time is the duration between a job's arrival time and its execution start time. On average, each job in the centralized scenario needs to wait for 5459 s to start the execution started, which is around 1000 s more than the independent scenario. This is because the centralized metascheduler takes time to decide where to submit the jobs; furthermore, due to the lack of information on adopted local scheduling algorithms, improper job distribution decisions lead to more job waiting time on the delegated nodes.

The average job response time of the decentralized scenario on a global perspective information system (scenario:Dec-G) is 7790 s, which is less than both the independent and centralized

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

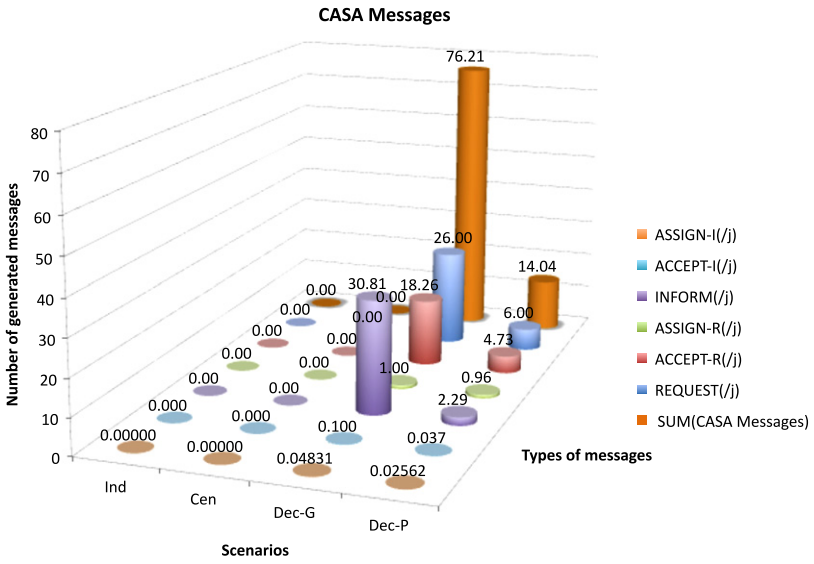

Fig. 5. Number of generated CASA messages compared between different scheduling schemes.

scenarios. The decentralized scenario executes as many jobs as the centralized scenario as shown in Fig. 1, the improvement of job response time is then yielded due to better scheduling efficiency, which is also proven by the obtained average job slowdown and job waiting time. The average job slowdown of this decentralized scenario is 29.15, which is about only 2/3 of the centralized scenario. In addition, the observed average job waiting time of the decentralized scenario upon the "global" resource discovery service is 1719 s, which is significantly improved compared to both the independent and centralized scenarios. This is because the probability based resource selection policy of the job assignment algorithm in CASA is able to find appropriate nodes for job delegations and executions without causing imbalanced job distribution. Furthermore, CASA's `dynamic scheduling phase` also helps to reschedule jobs to nodes with better performance through time.

The behavior of the CASA with "partial" resource discovery service is even more interesting. As shown in Fig. 4 (scenario:Dec-P), when a less qualified information system, such as the "partial" service, is employed, although each time only maximum six remote nodes can be found for the purpose of job delegation negotiation, the observed results are surprisingly improved when compared to the "global" service. Specifically, the job response time is 6958 s, wherein the job waiting time is 764 s. Such results are not only better than the independent and centralized scenarios, but also improved from the global perspective information system based decentralized scenario. In addition, the obtained job slowdown upon the local perspective information system is 16.32, which is again the most optimized value out of all scenarios evaluated. This phenomenon is because an information system with global perspective generally means longer resource discovery response time and heavier node workload; in this case, jobs which should have been rescheduled to gain better processing performance might have already been sent to a local resource before receiving `accept` message responses from nodes with better execution capability. In other words, some jobs might miss rescheduling opportunities.

6.4. CASA messages

Fig. 5 represents the overhead of various generated CASA messages. First of all, because the CASA is utilized by neither the independent nor the centralized scenario (scenario:Ind, Cen), the number of generated messages caused by the use of CASA is zero.

With regard to the decentralized scenario, when the "global" resource discovery service is adopted (scenario:Dec-G), the addresses of all nodes of the grid can be found for each job delegation attempt. In this case, for each submitted job, 26 `request` messages are generated so that each node of grid is notified about the newly arrived job. Afterwards, on average around 18 nodes of the grid respond to the job delegation requester node concerning its willingness and capability to process such a job by means of the `accept[R]` messages. Finally, one node is selected as the assignee node and receives the job retrieved from the delivered `assign[R]` message. During the `dynamic scheduling phase`, a total average of around 30 `inform` messages are generated for each job and disseminated to nodes of the grid for searching nodes with better job execution performance. Later on, each job has altogether around 10% probability of finding a remote node with better execution estimation (`accept[I]` message), and 4.8% probability to be rescheduled once (`assign[I]` message).

By contrast, while the "partial" resource discovery service is adopted (scenario:Dec-P), the message overhead is impressively reduced. At first, during the `job submission phase`, on average six `request` messages are generated for each arrival job, whereby each job will receive 4.7 `accept[R]` messages for job bidding, finally with a 96% probability of being delegated once to an assignee node. Secondly, during the `dynamic scheduling phase`, on average 2.3 `inform` messages will be generated for each job looking for rescheduling opportunities, whereby each job will have a 3.7% probability of finding a proper node for rescheduling, and 2.6% probability to be rescheduled once. It is then obvious that the use of CASA with the "partial" resource discovery service, an information system with "local perspective", leads to only 18.4% message overhead compared with the use of CASA with an information system with "global perspective".

7. Conclusion and future work

In this work, we propose a decentralized dynamic scheduling approach named the community-aware scheduling algorithm (CASA). The CASA is a two-phase solution comprised of an integrated collection of sub-algorithms used to facilitate job scheduling across decentralized distributed nodes.

The observed experimental results presented show that CASA is able to accomplish the same number of jobs as a centralized

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

approach in a decentralized manner. Furthermore, the CASA is able to improve both the average job slowdown and average job waiting time dramatically without asking for detailed information of participating nodes nor centralized control of the grid. In addition, while applying the CASA with a local perspective information system, both the performance benefit (e.g., job slowdown, job waiting time) and the overhead (e.g., generated/transferred messages) are significantly improved compared to the use of CASA upon a global perspective information system. This phenomenon illustrates that the proposed community-aware scheduling algorithm (CASA) has better performance while working with an information system with faster response time, instead of an information system with integrated global knowledge but slow reaction speed. The `dynamic scheduling phase` of the CASA can compensate for the fault of a local perspective information system by continuous job rescheduling with up-to-date "local knowledge".

Future work will include experiments using different grid workload archive traces [40] in order to gain a better understanding of CASA's correlation with improved metascheduling performance. Furthermore, additional widely adopted local scheduling algorithms, such as EASY backfilling and Shortest Job First, would need to be considered. Finally, the encouraging results observed from this work serve as the motivation to apply a future implementation of CASA within the context of current developments in cloud computing.

Acknowledgment

This research is financially supported by the Swiss Hasler Foundation in the framework of the "ManCom Initiative", project Nr. 2122.

References

[1] J. Ullman, NP-complete scheduling problems, Journal of Computer and System Sciences 10 (3) (1975) 384–393.
[2] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, International Journal of High Performance Computing Applications 15 (3) (2001) 200.
[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., A view of cloud computing, Communications of the ACM 53 (4) (2010) 50–58.
[4] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, et al., Grid'5000: a large scale and highly reconfigurable experimental Grid testbed, International Journal of High Performance Computing Applications 20 (4) (2006) 481.
[5] C. Catlett, P. Beckman, D. Skow, I. Foster, Creating and operating national-scale cyberinfrastructure services, CTWatch Quarterly 2 (2) (2006) 2–10.
[6] W. Gentzsch, D-grid, an e-science framework for German scientists, in: ISPDC'06: Proceedings of the Proceedings of The Fifth International Symposium on Parallel and Distributed Computing, IEEE Computer Society, Washington, DC, USA, 2006, pp. 12–13. doi:10.1109/ISPDC.2006.16.
[7] F. Gagliardi, The EGEE European grid infrastructure project, High Performance Computing for Computational Science-VECPAR 2004, 2005, pp. 194–203.
[8] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, M. Bowman, Planetlab: an overlay testbed for broad-coverage services, ACM SIGCOMM Computer Communication Review 33 (3) (2003) 12.
[9] P. Eerola, B. Konya, O. Smirnova, T. Ekelof, M. Ellert, J. Hansen, J. Nielsen, A. Waananen, A. Konstantinov, J. Herrala, et al. The Nordugrid production grid infrastructure, status and plans, in: Fourth International Workshop on Grid Computing, 2003, Proceedings, 2003, pp. 158–165.
[10] A. Iosup, T. Tannenbaum, M. Farrellee, D. Epema, M. Livny, Inter-operating grids through delegated matchmaking, Scientific Programming 16 (2) (2008) 233–253.
[11] M. de Assunção, R. Buyya, Performance analysis of allocation policies for interGrid resource provisioning, Information and Software Technology 51 (1) (2009) 42–55.
[12] A. Iosup, M. Jan, O. Sonmez, D. Epema, The characteristics and performance of groups of jobs in grids, Euro-Par 2007 Parallel Processing, 2007, pp. 382–393.
[13] D. Feitelson, A. Weil, Utilization and predictability in scheduling the IBM SP2 with backfilling, in: 12th International Parallel Processing Symposium, Citeseer, 1998, pp. 542–546.
[14] Y. Etsion, D. Tsafrir, A short survey of commercial cluster batch schedulers, Tech. Rep., Citeseer, 2005.
[15] V. Subramani, R. Kettimuthu, S. Srinivasan, S. Sadayappan, Distributed job scheduling on computational grids using multiple simultaneous requests, in: 11th IEEE International Symposium on High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings, 2002, pp. 359–366.
[16] Y. Huang, A. Brocco, N. Bessis, P. Kuonen, B. Hirsbrunner, Community-Aware Scheduling Protocol for Grids, in: 2010 24th IEEE International Conference on Advanced Information Networking and Applications, IEEE, 2010, pp. 334–341.
[17] L. Smarr, C. Catlett, Metacomputing, Communications of the ACM 35 (6) (1992) 44–52.
[18] V. Hamscher, U. Schwiegelshohn, A. Streit, R. Yahyapour, Evaluation of job-scheduling strategies for grid computing, Grid Computing GRID 2000, 2000, pp. 191–202.
[19] U. Schwiegelshohn, R. Yahyapour, Resource allocation and scheduling in metasystems, in: High-Performance Computing and Networking, Springer, 1999, pp. 851–860.
[20] C. Ernemann, V. Hamscher, R. Yahyapour, Economic scheduling in grid computing, in: Job Scheduling Strategies for Parallel Processing, Springer, 2002, pp. 128–152.
[21] G. Sabin, R. Kettimuthu, A. Rajan, P. Sadayappan, Scheduling of parallel jobs in a heterogeneous multi-site environment, in: Job Scheduling Strategies for Parallel Processing, Springer, 2003, pp. 87–104.
[22] M. De Assunção, R. Buyya, S. Venugopal, InterGrid: a case for internetworking islands of Grids, Concurrency and Computation: Practice and Experience 20 (8) (2008) 997–1024.
[23] D. De Roure, On self-organization and the semantic grid, IEEE Intelligent Systems 18 (4) (2003) 77–79.
[24] 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.
[25] R. Ranjan, A. Harwood, R. Buyya, SLA-based coordinated superscheduling scheme for computational Grids, in: Cluster Computing, 2006 IEEE International Conference on, IEEE, 2007, pp. 1–8.
[26] R. Smith, The contract net protocol: High-level communication and control in a distributed problem solver, IEEE Transactions on Computers 100 (29) (1980) 1104–1113.
[27] K. Leal, E. Huedo, I. Llorente, A decentralized model for scheduling independent tasks in federated grids, Future Generation Computer Systems 25 (8) (2009) 840–852.
[28] C. Wang, H. Chen, C. Hsu, J. Lee, Dynamic resource selection heuristics for a non-reserved bidding-based Grid environment, Future Generation Computer Systems 26 (2) (2010) 183–197.
[29] A. Das, D. Grosu, Combinatorial auction-based protocols for resource allocation in grids, in: 19th IEEE International Parallel and Distributed Processing Symposium, 2005. Proceedings, 2005, p. 8.
[30] E. Zurel, N. Nisan, An efficient approximate allocation algorithm for combinatorial auctions, in: Proceedings of the 3rd ACM Conference on Electronic Commerce, ACM, 2001, p. 136.
[31] S. Zanikolas, R. Sakellariou, A taxonomy of grid monitoring systems, Future Generation Computer Systems 21 (1) (2005) 163–188.
[32] L. Anand, D. Ghose, V. Mani, ELISA: an estimated load information scheduling algorithm for distributed computing systems, Computers & Mathematics with Applications 37 (8) (1999) 57–85.
[33] R. Shah, B. Veeravalli, M. Misra, On the design of adaptive and decentralized load balancing algorithms with load estimation for computational grid environments, IEEE Transactions on Parallel and Distributed Systems 18 (12) (2007) 1675–1686.
[34] D. Feitelson, Packing schemes for gang scheduling, in: Job Scheduling Strategies for Parallel Processing, Springer, 1996, pp. 89–110.
[35] C. Ernemann, V. Hamscher, A. Streit, R. Yahyapour, Enhanced algorithms for multi-site scheduling, Grid Computing GRID 2002, 2002, pp. 219–231.
[36] D. Jackson, Q. Snell, M. Clement, Core algorithms of the Maui scheduler, in: Job Scheduling Strategies for Parallel Processing, Springer, 2001, pp. 87–102.
[37] GWA, Grid5000 workload trace archive, http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Workloads.Gwa-t-2, 2010.
[38] TOP500, List statistics: operating system share for 06/2010, http://www.top500.org/, 2010.
[39] Y. Huang, A. Brocco, M. Courant, B. Hirsbrunner, P. Kuonen, MaGate Simulator: a simulation environment for a decentralized grid scheduler, Advanced Parallel Processing Technologies (2009) 273–287.
[40] GWA, Grid workload trace archive, http://gwa.ewi.tudelft.nl/, 2010.

Ye Huang obtained his Master degree from Chongqing University (China) in 2007. Since the end of 2007, he has worked on his Ph.D. (100% funded by the Swiss Hasler Foundation) at the University of Fribourg, Switzerland, as well as on the SmartGRID project. Meanwhile, Ye Huang also works for the Grid and Ubiquitous Computing Group, University of Applied Sciences of Western Switzerland. His research interests concern grid and distributed systems, service oriented interoperation and negotiation. Ye Huang is an IEEE member and an OGF member, and is involved in program committees of multiple conferences and journals. Ye Huang loves snowboarding, basketball, jogging, traveling and music.

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006

Nik Bessis is currently a Principal Lecturer (Associate Professor) in the Department of Computer Science and Technology at the University of Bedfordshire (UK). He obtained a BA from the TEI of Athens and completed his MA and Ph.D. at De Montfort University (Leicester, UK). His research interest is the analysis, research, and delivery of user-led developments with regard to trust, data integration, annotation, and data push methods and services in distributed environments. These have a particular focus on the study and use of next generation and grid technology methods for the benefit of various virtual organizational settings. He is involved in and leads a number of funded research and commercial projects in these areas. Dr. Bessis has published numerous papers and articles in international conferences and journals, and he is the editor of three books and the Editor-in-Chief of the International Journal of Distributed Systems and Technologies (IJDST). In addition, Dr. Bessis is a regular reviewer of several journals and conferences and has served as a keynote speaker, an associate editor, a conference chair, a scientific program committee member, and a session chair in numerous international conferences. More information is available from: http://www.beds.ac.uk/departments/computing/staff/nik-bessis.

Peter Norrington obtained an M.Sc. in Internet Technologies from the University of Luton in 2004. He received his Ph.D. in Web Security and Human Computer Interfaces from the University of Bedfordshire (UoB) in 2009. Since 2008 he has been an Educational Developer in the Centre for Excellence in Teaching and Learning at UoB, where he leads the deployment of its Web2.0 Personal Development Planning platform.

Pierre Kuonen obtained a Master degree in electrical engineering from the Swiss Federal Institute of Technology (EPFL) in 1982. After six year of experience in industry he joined the Computer Science Theory Laboratory at EPFL in 1988 and started working in the field of parallel and distributed computing. He received his Ph.D. degree from EPFL in 1993. Since 1994 he has steadily worked in the field of parallel and distributed computing. First at EPFL where he founded and managed the GRIP (Parallel Computing Research Group) then at the University of Applied Science of Valais. Since 2003 he has been a Full Professor at the University of Applied Science of Fribourg in the Information and Communication technologies department (TIC) where he is leading the GRID & Ubiquitous Computing Group. Besides his teaching activities he continues to actively participate in national and international research projects. Pierre Kuonen is the author or co-author of more than 50 scientific publications.

Beat Hirsbrunner is Full Professor and leader of the Pervasive and Artificial Intelligence research group at the University of Fribourg. He holds a diploma and Ph.D. in theoretical physics from ETHZ Zurich and University of Lausanne. He has been a postdoc, visiting researcher and Professor at the universities of Rennes and Berkeley, EPFL Lausanne, and IBM Research Labs San José. He has conducted research works on topics of parallel, distributed and pervasive computing, coordination languages, and human- computer interaction. He has been involved in several Swiss, European and US research projects. He served as dean of the Faculty of science of the University of Fribourg, and has since 2001 been a member of the Swiss NSF Council and vice-president of SARIT.

Please cite this article in press as: Y. Huang, et al., Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm, Future Generation Computer Systems (2011), doi:10.1016/j.future.2011.05.006