Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.

This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

Simulation Modelling Practice and Theory 19 (2011) 5–16

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

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

Modelling of a self-led critical friend topology in inter-cooperative grid communities

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

[a] Department of Computer Science and Technology, University of Bedfordshire, UK
[b] Department of Informatics, University of Fribourg, Switzerland
[c] Department of Information and Communication Technologies, University of Applied Sciences of Western Switzerland, Fribourg, Switzerland

ARTICLE INFO

Article history:
Available online 7 July 2010

Keywords:
Grid
Grid scheduling
Critical friend model (CFM)
Self-led critical friend
Inter-cooperative grid

ABSTRACT

For decades, much work has been done to increase the effectiveness and efficiency of job sharing amongst available computational resources. Resources can be organized into a variety of topologies, and recent work has shown that a decentralized distributed resource topology is a crucial but complicated scenario. This is because decentralized resources are normally grouped into independent virtual organizations (VOs) and isolated from each other by VO boundaries.

To convey jobs across gaps between various virtual organizations, a novel resource topology called the self-led critical friend model (CFM) is proposed in this work. The CFM deals with trust credits between resources according to their historical collaboration records. This trust reveals a feasible, realistic, and transferable correlation to facilitate the resource selection process for job delegation between arbitrarily connected physical resources. Consequently, the CFM is able to overcome the constraints caused by virtual organization boundaries.

Besides the theoretical model, a simulation-based implementation is carried out together with a complementary high-level community-aware scheduling protocol. After evaluating a number of compositional scenarios and criteria objectives, the observed results show the benefit of job scheduling across multiple VOs, as well as the capability of the self-led critical friend model as a novel cross-VO resource topology to represent and interconnect resources.

Crown Copyright © 2010 Published by Elsevier B.V. All rights reserved.

1. Introduction

Numerous works investigate how to enable effective and efficient job processing within the scope of cluster and single node [1,2], with some [3–5] starting to exploit the benefits of automatic cooperation amongst multiple nodes (different meta-schedulers). However, there remains room for improvement to enable automatic and self-manageable job exchange across different VOs. Indeed, there are many complex and volatile technical, political and personal factors which work against easy solutions. Thus, adaptive and dynamic solutions are necessary.

The contribution of this paper is the design of a novel model entitled the critical friend model (CFM). The CFM considers interconnected nodes, whichever VO they may be in, known via historical collaboration records as self-led critical friends

* Corresponding author. Tel.: +44 1582743476.
E-mail addresses: nik.bessis@beds.ac.uk (N. Bessis), ye.huang@unifr.ch (Y. Huang), peter.norrington@beds.ac.uk (P. Norrington), antony.brown@beds.ac.uk (A. Brown), pierre.kuonen@hefr.ch (P. Kuonen), beat.hirsbrunner@unifr.ch (B. Hirsbrunner).

1569-190X/$ - see front matter Crown Copyright © 2010 Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2010.06.020

Author's personal copy

(CF). Specifically, a critical friend is a node which has passed a threshold test of competence for service delivery, as determined by the node seeking the service. Taking a collection of such nodes known to one node, these are that node's critical friend community (CFC). The historical records are maintained within a node's updated Metadata Snapshot. Further definitions and discussions are introduced in the following sections at relevant points. Further more, regarding the CFM is working on nodes with various local scheduling algorithms and policies, a commentary high-level scheduling algorithm is needed. In this paper, the CFM is integrated together with a novel algorithm named the community-aware scheduling protocol (CASP) [6,7]. The CASP relies on a set of messages, i.e., REQUEST, ACCEPT, INFORM, and ASSIGN, to dispatch jobs to available resources within a reachable grid, in order to increase average job success rate, job response time and resource efficiency.

Bessis and Huang presented a vision of resource discovery in a decentralized distributed heterogenous grid environment for job allocation and scheduling in and across virtual organizations (VO) in [8], introducing the concept of the critical friend model (CFM) for grid scheduling. In this paper, we provide a graph theoretic foundation for the CFM, and an evaluation of an implementation of the model under various scenarios.

The remaining of this paper is organized as following, in Section 2, the modeling of self-led critical friend model is introduced. In Section 3, considered criterions and evaluated scenarios are detailed, followed by the relevant discuss on obtained experimental results in Section 4. Finally, conclusions and future work are given in Section 5.

2. Modelling of self-led critical friend

In describing our model we use standard graph theoretic notation. A graph, $G = (V,E)$ consists of a set of *vertices* $V = \{v_1, v_2, \ldots, v_n\}$ and a set of edges, $E = \{(v_1, v_2), (v_i, v_j), \ldots, (v_{n-1}, v_n)\}$ representing connections between pairs of vertices v_i and v_j. In our case, we will be making reference to a weighted graph $G = (V,E,W)$ where V and E are as defined earlier, and W is a mapping of a *weight* w_i to each edge $e_i \in E$. This construct then allows us to define a value e_i in the range [0–1], whereby low trust or distrust leads towards 0, and high trust leads towards 1, that can be used to model the notion of *trust* within our graph theoretic framework. Each vertex in the graph represents a node α_i, and the edges represent knowledge one node has about another node. At this point we would like to point out that we supplement our graph theoretic model with a number of additional (non-graph theoretic) concepts. The use of these additional concepts adds to the richness of model.

2.1. Critical friend trust scores

In order for a critical friend community (CFC) to be formed, a definition is required for how trust values propagate from one node to another. We begin by defining a *trust* score between nodes, then refine it to include temporal elements before settling on a proposed method for determination of critical trusted friends. We then describe how the topology of the CFC changes under the different policies defined above.

2.1.1. Trust

We define the neighborhood of a node α_i as the set of all nodes which it has knowledge of:

$$N(\alpha_i) = \{\alpha_j : (\alpha_i, \alpha_j) \in E\}.$$

We also define the common neighborhood of two nodes as:

$$N(\alpha_i, \alpha_k) = N(\alpha_i) \cap N(\alpha_k).$$

When first discovering another node, we need an initial value of *trust* to apply to that node. To do this, we ask all our common neighbors with the new node for their opinion of the node

$$TS_{initial}(\alpha_i, \alpha_j) = \frac{TS_{default}(\alpha_i) + \sum_{\alpha_k \in N(\alpha_i, \alpha_j)}^{\alpha_j} (TS(\alpha_i, \alpha_k) TS(\alpha_k, \alpha_j))}{|N(\alpha_i, \alpha_j)| + 1}.$$

Here we ask each neighbor who knows them how much they trust α_j. Each of these trust scores must be mitigated by how much we trust the node providing the *trust* score, so we take the product of our *trust* in their *trust*. In terms of the graph, we are multiplying the weights of the edges along the path between α_i and α_j. We take the sum of these *trust* scores of all neighbors of a node α_i who currently have knowledge of α_j (including the default *trust* $TS_{default}(\alpha_i)$), and then average these over the number of relevant neighbors (plus 1) to arrive at an overall (average) *trust* score. This allows us to take account of all relevant judgements when arriving at a decision. This initial *weight* is then assigned to the new edge from $\alpha_i \to \alpha_j$. Similarly, α_j will determine its own *trust* of α_i in the same manner, setting the *weight* from $\alpha_j \to \alpha_i$. Note that in the event that α_i has no links to any nodes who know α_j then $TS_{initial}$ will equal $TS_{default}$. The idea of default *trust* could be extended to allow nodes to have different default *trust* values for different VOs, allowing the node to trust VOs (based on past experiences with nodes belonging to that VO) and use that *trust* value as an initial basis for trust with nodes from that VO.

When asking a friend how much to trust another node, we would prefer that they had recent experience of that node. Knowing the level of service six months ago is obviously of less importance than knowing the level of service yesterday and so, trust gained recently should be given more *weight*.

Author's personal copy

We can make a modified version of our trust model that allows trust to decay over the time between interactions. We define $S(\alpha_i, \alpha_j)$ as the time since the last interaction between two nodes. We can also define S_i as the half-life time of α_i, such that it is the time in which α_i's *trust* will halve, without any further interactions. Thus we can define the decayed Trust Score $dTS(\alpha_i, \alpha_j)$ as:

$$dTS(\alpha_i, \alpha_j) = \frac{TS(\alpha_i, \alpha_j)}{2^{S(\alpha_i, \alpha_j)/S_i}}.$$

However, each node will want to evaluate the trust of other nodes based upon their own threshold value, rather than that of the node supplying the *TS*. Therefore it is useful to define the decayed Trust Score $dTS(\alpha_i, \alpha_j, \alpha_k)$ as:

$$dTS(\alpha_i, \alpha_j, \alpha_k) = \frac{TS(\alpha_j, \alpha_k)}{2^{S(\alpha_j, \alpha_k)/S_i}}.$$

Another factor to take into consideration is confidence of Trust. Trust based upon 2–3 interactions would often be judged to be more questionable to trust based on 200–300 interactions. When forming Trust for the first time, a node should give more *weight* to trust judgements that are based on more interactions. Each node α_i can track the total number of interactions there it has had with another node α_j as $I(\alpha_i, \alpha_j)$ and we can define the total interactions for all nodes in their common neighborhood $NI(\alpha_i, \alpha_j)$ as:

$$NI(\alpha_i, \alpha_j) = \sum_{\alpha_k \in N(\alpha_i, \alpha_j)} I(\alpha_k, \alpha_j).$$

This allows the calculation of a weighted *trust* value, incorporating a weighted average of decayed Trust Scores:

$$wdTS_{initial}(\alpha_i, \alpha_j) = \frac{TS_{default}(\alpha_i) + \sum_{\alpha_k \in N(\alpha_i, \alpha_j)}^{\alpha_j} wdTS(\alpha_i, \alpha_k) I(\alpha_k, \alpha_j) dTS(\alpha_i, \alpha_k, \alpha_j)}{|NI(\alpha_i, \alpha_j)| + 1}.$$

2.1.2. Critical friend trust score

Critical friend trust score (CFTS) includes both the time decay and weighting, plus the ability to have separate *trust* scores for different services. We use *x* to represent the type or category of a particular service. This allows *trust* to be built up for a particular node providing a particular service, without having to trust that node for other non-related services

$$CFTS_{x,initial}(\alpha_i, \alpha_j) = \frac{TS_{default}(\alpha_i) + \sum_{\alpha_k \in N(\alpha_i, \alpha_j)}^{\alpha_j} CFTS_x(\alpha_i, \alpha_k) I(\alpha_k, \alpha_j) dTS_x(\alpha_i, \alpha_k, \alpha_j)}{|NI(\alpha_i, \alpha_j)| + 1}.$$

All of the information required to calculate the CTFS can be contained within the Metadata snapshots maintained by each node.

2.2. Critical friends

We can define a node's critical friends as a subset of their neighborhood that have weighted *trust* scores higher than a critical threshold $C_{x,i}$.

$C_x(\alpha_i)$ is defined as the critical *trust* level for service *x*, as determined by node *i*. Any nodes that exceed this *trust* level will be considered as critical friends for the purposes of this type of service

$$CFN_x(\alpha_i) = \{\alpha_j : (\alpha_j) \in N(\alpha_i), CFTS_x(\alpha_i, \alpha_j) > C_{x,i}\}.$$

The overall critical friend neighborhood (CFN) of a node $CFN(\alpha_i)$ is an union of all of the service specific CFNs:

$$CFN(\alpha_i) = CFN_0(\alpha_i) \cup CFN_1(\alpha_i) \cup CFN_2(\alpha_i) \cdots \cup CFN_x(\alpha_i).$$

2.3. Self-led critical friend topology growth rules

The self-led critical friend topology, also known as the critical friend community (CFC), can be established and grow through historical job sharing experience; the CFC can be constructed following these four rules:

i. *NONE:* The critical friend model is disabled, no critical friend community will be established.
ii. *Restrict-CFC:* Once a job has been successfully executed on a remote node n_j, such job will be sent back to node n_i from which it was delegated. Afterward, node n_i will consider remote node n_j as a critical friend (CF) node of itself and put node n_j into its critical friend (CF) list

$$Discover(\alpha_i, \alpha_j) \rightarrow N(\alpha_i) = N(\alpha_i) + \alpha_j, \quad N(\alpha_j) = N(\alpha_j) + \alpha_i.$$

Author's personal copy

iii. *Optimized-CFC:* Once a job has been remotely executed on node n_j and sent back to its owner node n_i, not only node n_j itself, but also the CF nodes of n_j will be considered as critical friends of node n_i. *Optimized-CFC* aims at connecting CF lists of different nodes together, enlarges the scope of CF list of individual node, and constructs an integrated CFC finally

$$Discover(\alpha_i, \alpha_j) \rightarrow N(\alpha_i) = N(\alpha_i) \cup CFN(\alpha_j), \quad N(\alpha_j) = N(\alpha_j) \cup CFN(\alpha_i).$$

iv. *Massive-CFC:* One more step is considered based on *Optimized-CFC* policy. Once a remotely executed job is sent back from node n_j to n_i, the network neighbor list of executing node n_j, which is collected via the adopted information system of n_j instead of the participation of CFC and having the same critical friend level as n_i, will also be considered as critical friends of node n_i. Massive-CFC tries to utilize any possibility to enlarge the scope of CF list on each node and establish an integrated CFC which covers as many nodes of the overall grid as possible

$$Discover(\alpha_i, \alpha_j) \rightarrow N(\alpha_i) = N(\alpha_i) \cup N(\alpha_j), \quad N(\alpha_j) = N(\alpha_j) \cup N(\alpha_i).$$

It is noteworthy that if all nodes of the CFC agree to share their knowledge obtained via self-adopted Information Systems, then the *Massive-CFC* policy turns out to be a flooding like approach and might produce significant network overhead to participating nodes. On the other hand, however, regarding the CFM focuses on critical friend correlation maintenance between individual nodes in a decentralized manner, independent node owners have their freedom to determine the adoption of the *Massive-CFC* policy according to its realtime network overhead.

2.4. Self-led critical friend job dispatching rules

When a node needs a job done, it broadcasts a REQUEST message out to nodes in its neighborhood. The node receiving the REQUEST can either accept the job itself or choose to pass the message to one of its CFs. Which CFs the job can be passed to depends on the Job Dispatch policy in place.

We define $REQUEST(\alpha_i, \alpha_j, j_x)$ as a REQUEST message being sent from node α_i to node α_j for job j of type x.

Where a job is not accepted but passed onto one or more other nodes, we define $RELAY(\alpha_i, \alpha_j, \alpha_k, j_x)$ as node α_j relaying (passing on) a REQUEST message, that originated from node α_i to α_k. We note, this is not the same as the concept of job delegation, in which node α_j would make on behalf of node α_i the decision to accept or not the availability of node α_k to take the job. In effect, a RELAY is a REQUEST with a policy restriction

$$RELAY(\alpha_i, \alpha_j, \alpha_k, j_x) \rightarrow REQUEST(\alpha_j, \alpha_k, j_x).$$

The policy restrictions can be formulated for the situations matching the topology growth rules allowing expansion beyond the first level of nodes contacted, as in the previous section.

i. *Optimized-CFC:* Only CFs of α_j relevant to the job type are contacted, as being more likely to be able to complete the job

$$RELAY(\alpha_i, \alpha_j, \alpha_k, j_x) \quad \text{If } (\alpha_k \in CFN_x(\alpha_j)).$$

ii. *Massive-CFC:* All CFs of α_j are contacted to massively expand the nodes who may do the job.

$$RELAY(\alpha_i, \alpha_j, \alpha_k, j_x) \quad If \ (\alpha_k \in CFN(\alpha_j)).$$

2.5. Self-led critical friend job acceptance rules

Once a node receives a REQUEST it wishes to fulfill, it sends an ACCEPT response back to the initiator node. The initiator node then evaluates all the ACCEPT responses, based on the suitability of the nodes sending them. Let $SUITABILITY(\alpha_i, \alpha_k, j_x)$ be the suitability of a node α_k, to perform a job j of type x for node α_i. This is based on whichever set of criteria the initiator is most interested in, e.g., the least time to completion or lowest execution cost, or the weighted average if more than one criteria is used, but this is out of the scope of this work. This suitability can be modified by the trust the relaying node has in the node offering to accept, allowing past experience to allow nodes to give preference to nodes that have proven themselves reliable, while avoiding those that fail to live up to their claims.

Let $TSUITABILITY(\alpha_i, \alpha_k, j_x)$ be the trusted suitability of a node to perform a job j of type x, defined as:

$$TSUITABILITY(\alpha_i, \alpha_k, j_x) = CFTS_x(\alpha_i, \alpha_k) SUITABILITY(\alpha_i, \alpha_k, j_x).$$

Here, the initiator can weight the suitability of the ACCEPT responses by the CFTS it has for the responding nodes. Once this is done, the trusted suitability scores for each ACCEPT message can be compared, and the job will be transferred to the chosen best one by means of an ASSIGN message. During the Scheduling and Rescheduling Phases, a similar process can take place with the ACCEPT messages sent back to the originating node.

Author's personal copy

3. Evaluation

Evaluation of the modelling of the self-led critical friend is discussed in this section. Especially, imposed scheduling effectiveness and efficiency benefits are examined both in an isolated-VO environment and a crossed-VO situation. A fair number of scenarios have been considered to evaluate a variety of behaviors. More specifically, the objectives of the experiment are introduced in Section 3.1, followed by the utilized models in Section 3.2 and adopted simulator and configuration in Section 3.3. Lastly, the scenarios evaluated are discussed in Section 3.4.

3.1. Objectives

The reference experiment is to evaluate the effectiveness and efficiency of the self-led critical friend model during cooperation with other complementary scheduling strategies and facilities, particularly the community-aware scheduling protocol (CASP). A fair number of criteria are considered during the evaluation as presented below:

(A) *REQUEST*: The community-aware scheduling protocol (CASP) replies in its *Job Submission Phase* to deliver local received jobs to reachable remote resources of the grid by means of *REQUEST* messages. In other words, the number of generated *REQUEST* messages refers to the scope of the reachable grid, which is a crucial target of the self-led critical friend model.
Assuming there are in total m nodes in the grid, each node i has generated $Msg_i(request)$ messages during its lifecycle, then the total number of generated *REQUEST* messages of the overall grid can be calculated by:

$$REQUEST_{total} = \sum_{i=1}^{m} Msg_i(request).$$

(B) *ACCEPT-R*: The CASP requests node which is capable of handling incoming job delegation requests to response by means of the *ACCEPT* messages. In this case, more *ACCEPT* messages stand for better effectiveness imposed by the self-led critical friend model.
Similarly as above, each node i has generated $Msg_i(accept)$ messages during its lifecycle to answer incoming job delegation requests, then the total number of generated *ACCEPT* messages of the overall grid can be obtained by:

$$ACCEPT(R)_{total} = \sum_{i=1}^{m} Msg_i(accept).$$

(C) *ASSIGN-R*: With regard to the CASP, when a candidate remote node is selected for a corresponding to-delegate job, such a job will be transferred to the remote node by means of a *ASSIGN* message.
If each node i has sent $Msg_i(assign)$ jobs to remote nodes, the total number of generated *ASSIGN* messages of the entire grid is:

$$ASSIGN(R)_{total} = \sum_{i=1}^{m} Msg_i(assign).$$

(D) *RJC*: The rate of successfully executed jobs of the overall community (RJC) is adopted to prove the functional effectiveness brought by the design of job sharing within an interoperable grid community. The RJC is supposed to be maximized because it presents the capability and effectiveness of delivering jobs to appropriate nodes of the grid community, instead of suspending them because no proper resources can be discovered in time.
Assuming there are in total m nodes in the grid community, wherein k nodes have local job submission input $job_i(submitted)$. Once the simulation is finished, successfully executed jobs on each node is $job_i(exed)$. Then the RJC can be obtained by:

$$RJC = \frac{\sum_{i=1}^{m} job_i(exed)}{\sum_{i=1}^{k} job_i(submitted)}.$$

(E) *CE*: The community efficiency (CE) refers to average usage of all resources of the entire grid community. Regarding jobs submitted to specific node can be easily scheduled and delegated to any remote node of the grid, the resource usage efficiency should be considered within the scope of grid, instead of the individual node.
Together there are m nodes existing in the grid. Each node n_i has CPU_i processing elements (PEs), and has been kept running for $Makespan_i$ seconds till the end of the simulation. Meanwhile, each node has successfully executed p jobs, each job job_j needs to run t_j seconds on cpu_j PEs. Consequently, the Community Efficiency is calculated by:

$$CE = \frac{\sum_{i=1}^{m} \sum_{j=1}^{p} cpu_j * t_j}{\sum_{i=1}^{m} CPUi * Makespan_i}.$$

Author's personal copy

(F) *Network-Coverage*: In our simulation, each node of the grid employs an information system (IS) for remote resource discovery and monitoring. Discovered remote node information is cached within each individual node and kept up-to-date. In other words, each node recognizes the existence of several remote nodes of the grid. The number of known remote nodes relies on the adopted Information System, and will affect the behavior of grid scheduling.
Assuming there are in total m nodes in the grid. Node n_i knows $P_{known}(i)$ remote nodes due to the adopted IS, thus $R_i(network)$ refers to the rate of known remote nodes for each individual node n_i according to the adopted information system. *Network-Coverage* can be calculated by:

$$Coverage_{avg-network} = \frac{\sum_{i=1}^{m} R_i(network)}{m}, \quad R_i(network) = \frac{P_{known}(i)}{m}.$$

(G) *CFC-Coverage*: Similarly to the *Network-Coverage*, each node also has some knowledge of remote nodes due to participation of self-led critical friend topology, which also affects grid scheduling behavior on each node.
Assuming there are in total m nodes in the grid. Node n_i knows $Q_{known}(i)$ remote nodes due to the self-led critical friend topology, thus $R_i(cfc)$ refers to the rate of known remote nodes on each individual node n_i, then:

$$Coverage_{avg-cfc} = \frac{\sum_{i=1}^{m} R_i(cfc)}{m}, \quad R_i(cfc) = \frac{Q_{known}(i)}{m}.$$

3.2. Simulation models

Though no doubt grid systems vary widely depending on the usage scenarios. However, one typical example of a computational grid is still the execution of computationally intensive batch jobs on collaborative computers. Some models retrieved from this scenario are utilized in our experiment, and represented thus:

3.2.1. Job model

The job concerns computationally intensive batch jobs submitted by users continuously through time. In our case, each submitted job is comprised of several parameters, e.g., requested run time, requested number of PEs, and requested type of operating system. Both sequential and parallel jobs are supported as long as the execution can be restricted within one single machine with enough processing elements. Job migration and preemption are not considered at present.

3.2.2. Machine model

Machines considered in this simulation refer to the massive parallel processor systems (MPPs), which are commonly comprised of several processing elements (PEs) connected by fast interconnections. Each processing element is a single processing system with local CPU and memory, and is able to process jobs exclusively. All processing elements of the same MPP are using the same operating system.

3.2.3. Node model

Nodes of this simulation contribute their computational resources and share received local submitted jobs with other nodes of the same grid. Each node consists of one or several machines and is managed by a single high-level grid scheduler. Each node has its own resource management system or middleware, as well as specific local policies. The resources between different nodes are heterogeneous.

3.3. Simulator and configuration

A grid simulator named MaGate simulator [9] is adopted to evaluate the aforementioned introduced objectives.

The MaGate simulator is implemented on GridSim [10] and Alea [11], it supports the modelling of a variety of essential grid components, such as grid jobs with various parameters, heterogeneous grid resources, and grid users. A grid scheduler named MaGate scheduler [12] is implemented by default within the MaGate simulator, which is capable of sharing received jobs with other remote resources to increase the effectiveness and efficiency of the overall grid. In addition, novel grid scheduling algorithms, such as community-aware scheduling protocol (CASP), are also supported to help the job dispatching within the scope of the reachable grid. Lastly, to support scheduling activities, the MaGate simulator relies on a grid overlay simulator named the Solenopsis [13] to simulate the adopted grid information system (IS).

Both job and machine parameters adopted in the simulation are either constants, or randomly generated according to an uniform distribution. The scale of the grid is setup based on known real grids [14,15] to the best of our knowledge. More details about the configuration can be found in Table 1.

To obtain stable values, each scenario result was averaged from two repeated iterations. The experiments are performed upon an Intel Core Duo 2.53 GHz machine, with 4 GB RAM. The scale of input jobs is limited in order to execute all scenarios and obtain first results within a reasonable time.

Author's personal copy

Table 1
Simulation configuration of the experiment.

Simulation configuration	
Number of nodes of the overall grid	75
Number of nodes having local job input	5
Number of existing virtual organizations	3
Number of nodes per virtual organization	[15,30]
Number of jobs submitted to each node (node having local job input)	1500
Job arrival time	[0–24 h]
Average job estimated execution time	1000 s
Average job estimated MIPS	1000
Number of PEs required by each individual job	[1–5]
Number of MPP machine per MaGate node	1
Number of PEs per machine	[64–128]
Average MIPS of each PE	1000
Types of operating system required by job	[Linux,Win,Mac]
Types of machine operating system	[Linux,Win,Mac]

3.4. Evaluation scenarios

Variety of parameters and policies, including topology establishing and job dispatching rules, can affect behaviors of the self-led critical friend model. In this simulation, four types of policy are considered, and as a result, evaluated scenarios are a composition of selected policies from the following types:

3.4.1. CASP policies

The community-aware scheduling protocol (CASP) dedicates to enable job sharing within the scope of the overall grid. Especially, during its *Job Submission Phase*, each node can apply its own policy to determine whether a remote job should be accepted for local execution. Various policies can be adopted here but here we only consider an extremely strict one, namely the *CASP Strict Dispatching Policy*.

In terms of CASP, nodes receiving jobs from local users are defined as *initiator nodes*. Once a remote node, also known as *responder node*, receives a job delegation request from an *initiator node*, the *responder node* firstly checks whether the requirement of such a remote job can be fulfilled by local resources. If so, the *responder node* needs also to ensure available resources, such as free processing elements, can be obtained instantly. If both requirement can be satisfied, the *responder node* will generate an ACCEPT message and send it back to the *initiator node*. In case no candidate remote nodes can be discovered during the *Job Submission Phase*, submitted jobs will be suspended and discarded by corresponding *initiator nodes*.

The philosophy of *CASP Strict Dispatching policy* is supposed to lead to well controlled job slowdown because only nodes with expected and instantly available PEs are selected for job sharing. On the other hand, many jobs might be suspended and discarded because no free nodes can be discovered at the moment of job arrival through the adopted information system (IS), the obtained RJC and community efficiency (CE) could be therefore low.

3.4.2. Self-led critical friend VO isolation policies

To evaluate the performance difference between grid being separated into several isolated-VOs and grid with crossed-VO boundaries, two VO isolation policies are counted.

i *isolated-VO*: nodes of the overall grid exist in separated virtual organizations (VOs). The boundaries between VOs cannot be crossed while an *isolated-VO* policy is applied; thus all generated job delegation requests and responses according to the community-aware scheduling protocol can only be transferred within each individual VO despite the same information system (IS) being adopted. In addition, in terms of the self-led critical friend model, only nodes of the same VO can be discovered and considered as critical friend nodes of another grid node.
More specifically, as depicted in Table 1, in total 75 grid nodes are grouped into three VOs. Both VO_1 and VO_2 have 30 nodes, while VO_3 has the remaining 15 nodes. Two nodes of VO_1 and VO_2 have local job submission input, and one node of VO_3 has local job input. With regards to the launched *isolated-VO* policy, all locally submitted jobs can only be shared by nodes of the same VO, therefore the overall grid is actually isolated into three small cliques, even though they are physically connected.

ii. *crossed-VO*: In contrast to the *isolated-VO* policy, when a *crossed-VO* policy is activated, nodes from diverse VOs can interact and share jobs, in which case they are considered as critical friends each other; consequently, a well constructed self-led critical friend model based topology, also defined as critical friend community (CFC), can behave as glue to connect isolated virtual organizations together.

Author's personal copy

Table 2
Evaluated self-led critical friend model scenarios.

Evaluated scenarios	
ID	Scenario composition
NONE	NONE
R-D	Restrict-CFC; restrict-dispatch
R-T	Restrict-CFC; tolerant-dispatch
O-R	Optimized-CFC; restrict-dispatch
O-T	Optimized-CFC; tolerant-dispatch
M-R	Massive-CFC; restrict-dispatch
M-T	Massive-CFC; tolerant-dispatch

3.4.3. Self-led critical friend topology growth policies

As discussed in subsection 2.3, the self-led critical friend topology, also known as the critical friend community (CFC), can be established and enlarged through historical job sharing experience. Such CFC construction rules will be evaluated in this section.

3.4.4. Self-led critical friend job dispatching/acceptance policies

Once the self-led critical friend model (CFM) is enabled and critical friend community (CFC) is established due to the aforementioned topology policies, jobs can be transferred within both network formed by adopted information system and the CFC. The community-aware scheduling protocol is launched firstly to assign a job to an appropriate node within the Job Submission Phase; in case no ACCEPT messages sent by appropriate nodes are obtained, the CFM will be invoked to try another job delivery possibility. More specifically, the following four policies are considered in this experiment:

i. *NONE*: The CFM job dispatching is disabled. No matter whether the CFC has been established or not, jobs dispatched failed from the CASP Job Submission Phase will be suspended till the end of the simulation.
ii. *Restrict-Dispatch*: During the CASP Job Submission Phase, if an incoming job delegation REQUEST message from node n_i cannot be satisfied by the hosting node n_j, instead of simply ignoring such a message, the CFM dispatching process on node n_j will be launched. Node n_j firstly checks whether node n_i is considered as a critical friend (CF) to itself; if so, node n_j then tries to find an appropriate remote node (e.g., node n_k) from its CF list and sends an ACCEPT message under the name of n_k as the replacement of itself to the initiator node n_i. In this case, two CF nodes (n_i, n_k) are connected due to their mutual critical friend node n_j.
iii. *Tolerant-Dispatch*: The *Tolerant-Dispatch policy* follows a similar workflow as the *Restrict-Dispatch policy*. The difference is that instead of checking whether initiator node n_i is a known critical friend of its own, node n_j launches the CFM job dispatching process for all incoming job delegation REQUEST messages which cannot be satisfied by local resources. Thus the CF nodes of node n_j are "recommended" to other remote initiator nodes. In the case that "recommended" CF nodes of n_j are selected by REQUEST initiator nodes and finally successfully execute delegated jobs, the CFC surrounding node n_j will be enlarged.

3.4.5. List of scenarios

The above policies are introduced in terms of CASP strategy, self-led critical friend topology policies, and job dispatching/acceptance rules. The experimental scenarios are then organized as combination of these policies. To be specific, all seven self-led critical friend related scenarios listed in Table 2 are evaluated under two self-led critical friend VO isolation policies (*isolated-VO* and *crossed-VO*), respectively, thus in total 14 scenarios are evaluated.

4. Results

Based on the above scenarios, objectives introduced in Section 3.1, including rate of successfully executed jobs of the overall community (RJC), community efficiency (CE), network coverage, critical friend community (CFC) coverage and CASP message processing, are evaluated within the adopted simulation platform [9]. Results discussion is provided as below to reveal the effectiveness and efficiency impacted by applying the self-led critical friend model (CFM) into the reality of decentralized distributed computing.

4.1. RJC/community efficiency and network/CFC coverage

As shown in Fig. 1, grids with the same node topology but different VO isolation policies lead to different job sharing and resource execution efficiency.

At first, when an *isolated-VO* grid is deployed, if the self-led critical friend model (CFM) is disabled (scenario:NONE), the achieved RJC is 33.02%, and the overall averaged resource usage efficiency (CE) is 13.09%. Afterwards, the CFM is activated with both *Restrict-CFC* and *Optimized-CFC* topology growth policies, as well as all available CFM job dispatching policies;

Author's personal copy

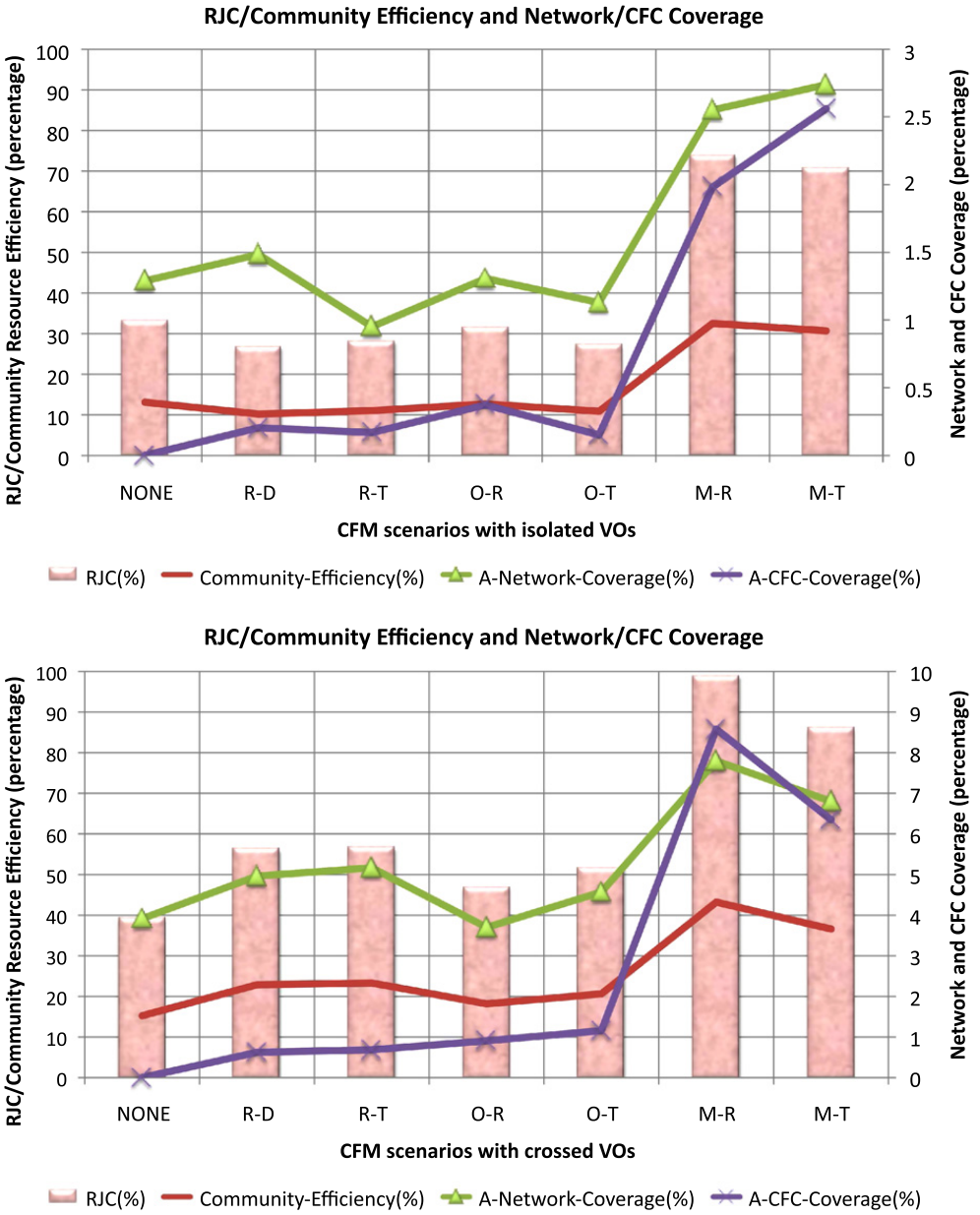

Fig. 1. RJC and community efficiency vs. network and CFC coverage on isolated-VOs and crossed-VOs.

however, the obtained RJC and CE are not impressively improved. It is mainly because the resulting network and CFC coverage are both kept in a quite low level which is no more than 1.5%.

Once the *Massive-CFC* topology growth policy is applied, each grid node can visit the neighborhood knowledge of its own critical friend nodes, wherein remote node information is collected both from information system (IS) and CFC; this enables the possibility of enlarging each node's knowledge of remote nodes. As a result, both the RJC and average community efficiency are almost doubled compared to previous adopted CFM topology growth policies. In addition, the coverage of network and CFC are also impressively increased.

It is also noteworthy that the curve of network coverage matches the increase of RJC exactly, which means that when the scale of reachable grid/VO is limited, the community job sharing efficiency is closely related to the knowledge of the neighborhood network of each participating node.

Secondly, when a *crossed-VO* grid is deployed, if the CFM is disabled, although nodes from diverse VOs still have the opportunity to discover and interact with each other due to the adopted information system, the observed RJC and community efficiency are only slightly improved by 5% and 1%, respectively.

Afterwards, when the *Restrict-CFC* topology growth policy is applied, both the CFM *Restrict-Dispatch policy* (scenario: R-D) and the *Tolerant-Dispatch policy* (scenario: R-T) have similar impact with regards to the improvement of resulting RJC and community efficiency by 15% and 8%, respectively. Because the CFC coverage of each node is still kept at a quite similar

Author's personal copy

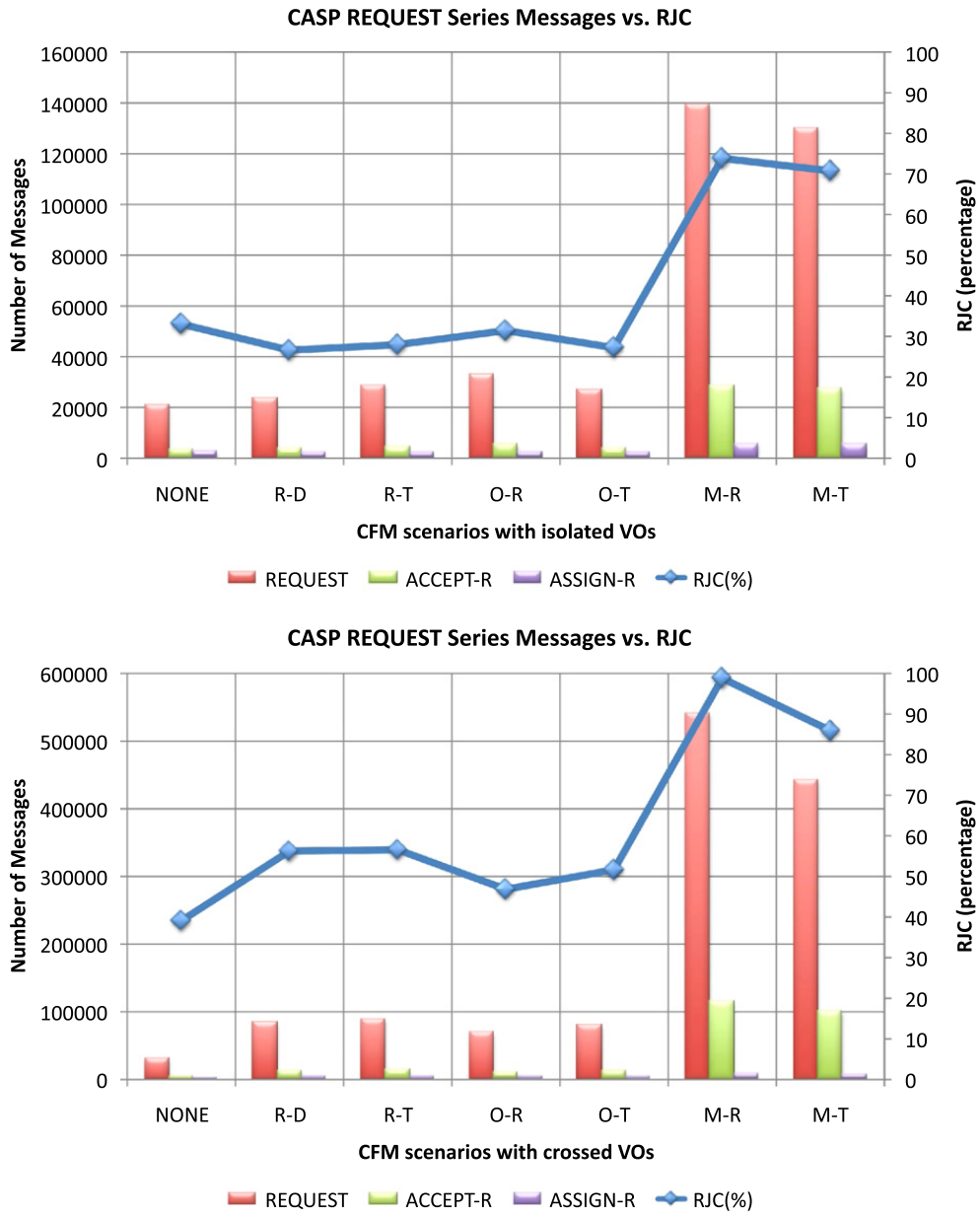

Fig. 2. CASP REQUEST series messages vs. RJC on isolated-VOs and crossed-VOs.

and low level scale (less than 1.06%), each participating node does not have much information in terms of remote critical friend nodes, therefore the improvement of RJC and community efficiency is still limited.

Once the CFM topology growth policy *Massive-CFC* is applied, adopted CFM job dispatching polices, such as scenario M-R and M-T, are able to lead to better RJC and community efficiency up to 99% and 42%, respectively. This is because size of the self-led critical friend topology, also known as critical friend community (CFC), is expanded, then more remote nodes are available for jobs needing to be executed remotely.

In short, Fig. 1 illustrates that no matter the adopted topology growth and job dispatching/acceptance polices, a *crossed-VO* grid can generally lead to better job sharing and resource usage efficiency compared to an *isolated-VO* grid. Further more, an aggressive topology growth policy, e.g., *Massive-CFC*, turns out to have higher priority compared to the impact of job dispatching policies. At last, the CFM *Restrict-Dispatch policy* upon a *Massive-CFC crossed-VO* grid (scenario:M-R) leads to the best RJC and community efficiency in our simulation.

4.2. CASP REQUEST series messages vs. RJC

As mentioned before, the self-led critical friend model is now working together with a high-level heuristic called the community-aware scheduling protocol (CASP) [6,7]. The CASP consists of two phases, namely the *Job Submission Phase* and *Dynamic Rescheduling Phase*. The *Job Submission Phase* plays an important role while trying to dispatch locally submitted jobs

Author's personal copy

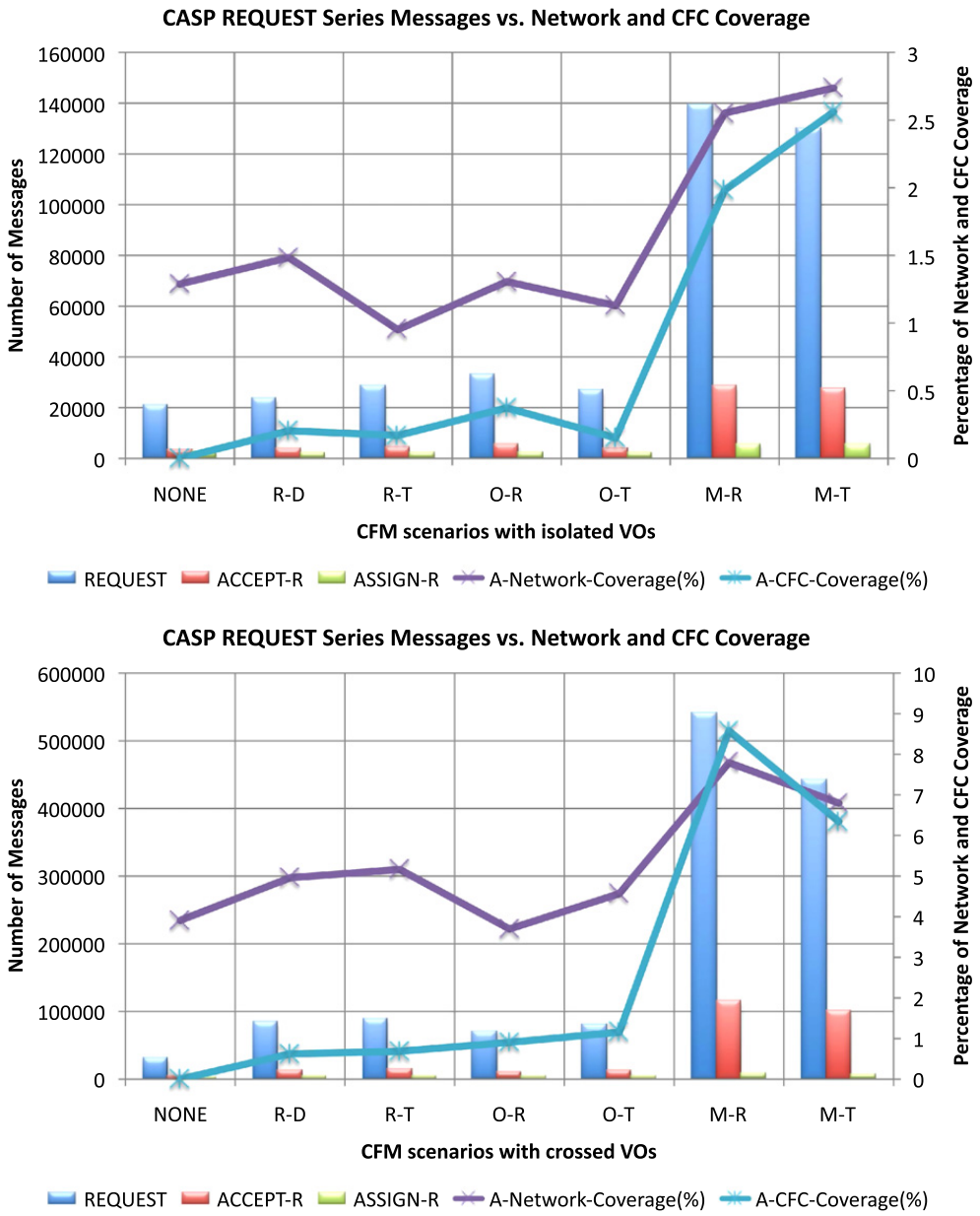

Fig. 3. CASP REQUEST series messages vs. network and CFC coverage on isolated-VOs and crossed-VOs.

to appropriate nodes of the reachable grid by means of generated REQUEST messages. Remote nodes receiving such REQUEST messages and competent in job executing are supposed to response by sending ACCEPT messages back. Finally, candidate nodes are selected and corresponding jobs will be transferred through the newly created ASSIGN messages.

As shown in Fig. 3, in both an *isolated-VO* grid and a *crossed-VO* grid, the number of generated REQUEST messages corresponds to the monitored job sharing effectiveness (RJC) very well, which means that a large number of REQUEST messages can help to find proper nodes for job delegations and consequently increase the success rate of executing such jobs on reachable resources. Additionally, the number of generated REQUEST messages in a *crossed-VO* grid is much more than the number generated in an *isolated-VO* grid, and the obtained RJC in a *crossed-VO* grid is therefore better. This result again confirms the action of the self-led critical friend model in maximizing the effectiveness and efficiency of job sharing amongst decentralized distributed nodes.

4.3. CASP REQUEST series messages vs. network and CFC coverage

Besides the discuss in Section 4.2, a further analysis is given here to check the impact of grid with crossed-VO boundaries.

As depicted in Fig. 2, when an *isolated-VO* grid is deployed, the curve of the individual node's network coverage matches the number of generated REQUEST messages; when a *crossed-VO* grid is deployed, the curve of generated REQUEST messages matches CFC coverage better compared to the network coverages. This means that when VO boundaries of a grid are crossed,

Author's personal copy

both the CASP protocol and the CFM model can be enhanced, but not the traditional grid information system (IS). This phenomenon demonstrates action differences between conventional grid information systems (ISs) and the proposed self-led critical friend topology.

5. Conclusion and future work

To increase job scheduling effectiveness and efficiency, a novel resource topology named the self-led critical friend model (CFM) is introduced in this paper. Unlike other conventional scheduling approaches which emphasize discovering free resource "slots" from known resources, the CFM focuses on bridging gaps between variety of grid virtual organizations (VOs) which are normally generated due to various technique and political issues, and interconnects nodes from such independent VOs together to construct a bigger and *trust* based resource pool.

The self-led critical friend model (CFM) is comprised of a set of systematic components, including *critical friend*, *critical friend trust scores*, *topology growth policies*, *job dispatching* and *acceptance rules*, in order to describe the correlation between nodes which already have historical job sharing records, as well as a *trust* based approach to evaluate such correlation. In addition, regarding the CFM is working on nodes despite adopted local scheduling algorithms, a high-level community-aware scheduling protocol is integrated to facilitate the cross-node job scheduling process without interfering participating nodes's control on physical machines and preferred local policies.

Following previous works [16], in order to reveal the practical impact brought by enabling job sharing between nodes from multiple independent VOs, a fair amount of scenarios have been evaluated in simulation. The observed results demonstrate that many important criteria, such as rate of averaged successfully executed jobs, average resource utilization efficiency, individual node's knowledge on neighboring nodes and critical friends, can be impressively improved as long as the boundaries amongst different VOs are crossed.

Regarding the future work, an in-depth research will be carried out to find an more CFM based scheduling polices to achieve an optimal balance between efficiency and security. In addition, realistic workload archives are scheduled in future evaluation to examine the behavior of CFM under load of real grid systems. Finally, more scenarios are to be constructed to adapt the self-led critical friend model to extensive fields, such as the cloud computing.

Acknowledgements

This research is financially supported by the EU Marie Curie Knowledge Transfer Programme and Swiss Hasler Foundation in the framework of the "ManCom Initiative", Project No. 2122.

References

[1] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Software: Practice and Experience 32 (2) (2002) 135–164.
[2] J. Schopf, Ten actions when super scheduling: a grid scheduling architecture, in: Workshop on Scheduling Architecture, Global Grid Forum, Tokyo, 2003, pp. 15–24.
[3] V. Yarmolenko, R. Sakellariou, Towards increased expressiveness in service level agreements, Concurrency and Computation: Practice and Experience 19 (14) (2007) 1975–1990.
[4] O. Waldrich, P. Wieder, W. Ziegler, A meta-scheduling service for co-allocating arbitrary types of resources, Lecture Notes in Computer Science 3911 (2006) 782.
[5] E. Huedo, R. Montero, I. Llorente, The GridWay framework for adaptive scheduling and execution on grids, Scalable Computing: Practice and Experience 6 (3) (2005) 1–8.
[6] 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.
[7] A. Brocco, A. Malatras, Y. Huang, B. Hirsbrunner, ARiA: a protocol for dynamic fully distributed grid meta-scheduling, in: 30th International Conference on Distributed Computing Systems, ICDCS 2010, IEEE, Genoa, Italy, in press.
[8] Y. Huang, N. Bessis, A. Brocco, P. Kuonen, M. Courant, B. Hirsbrunner, Using metadata snapshots for extending ant-based resource discovery service in inter-cooperative grid communities, in: International Conference on Evolving Internet, INTERNET 2009, IEEE Computer Society, Cannes, French Riviera, France, 2009, pp. 89–94.
[9] Y. Huang, A. Brocco, M. Courant, B. Hirsbrunner, P. Kuonen, MaGate simulator: a simulation environment for a decentralized grid scheduler, in: Proceedings of the Eighth International Symposium on Advanced Parallel Processing Technologies, Springer, 2009, p. 287.
[10] R. Buyya, M. Murshed, GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency and Computation: Practice and Experience 14 (13–15) (2002) 1175–1220.
[11] D. Klusacek, L. Matyska, H. Rudova, Alea-grid scheduling simulation environment, Lecture Notes in Computer Science 4967 (2008) 1029.
[12] Y. Huang, A. Brocco, M. Courant, B. Hirsbrunner, P. Kuonen, MaGate: an interoperable, decentralized and modular high-level grid scheduler, International Journal of Distributed Systems and Technologies 1 (3) (2010).
[13] A. Brocco, B. Hirsbrunner, M. Courant, Solenopsis: a framework for the development of ant algorithms, in: IEEE Swarm Intelligence Symposium, 2007, pp. 316–323.
[14] GridWorkloadsArchive. <http://gwa.ewi.tudelft.nl/pmwiki/>.
[15] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D. Epema, The grid workloads archive, Future Generation Computer Systems 24 (7) (2008) 672–686.
[16] Y. Huang, N. Bessis, A. Brocco, S. Sotiriadis, M. Courant, P. Kuonen, B. Hirsbrunner, Towards an integrated vision across inter-cooperative grid virtual organizations, in: Future Generation Information Technology, FGIT 2009, LNCS, Springer, Jeju Island, Korea, 2009, pp. 120–128.