2010 24th IEEE International Conference on Advanced Information Networking and Applications

Community-Aware Scheduling Protocol for Grids

Ye Huang*, Amos Brocco*, Nik Bessis†, Pierre Kuonen‡, and Beat Hirsbrunner*

*Department of Informatics, University of Fribourg, Switzerland
Email: {ye.huang, amos.brocco, beat.hirsbrunner}@unifr.ch
†Department of Computer Science and Technology, University of Bedfordshire, UK
Email: nik.bessis@beds.ac.uk
‡Department of Information and Communication Technologies,
University of Applied Sciences Western Switzerland (Fribourg)
Email: pierre.kuonen@hefr.ch

***Abstract*—Much work has been done to exploit the effectiveness and efficiency of job scheduling upon distributed computational resources. With regard to existing resource topology and administrative constraints, scheduling approaches are designed for different hierarchic layers, for example, scheduling for job queues of local resource management systems (local scheduling), and scheduling for job queues of high level schedulers (also known as meta-schedulers or grid schedulers). Such scheduling approaches mainly focus on optimizing job queues of the hosting nodes, which are interconnected with computational resources directly or indirectly. In the real world (or in a community-based grid), a grid is comprised of nodes with different computing power and scheduling preferences, which in turn, raise a notable opportunity that is to exploit and optimize the process of job sharing between reachable grid nodes via improving the job allocation and efficiency ratio.**

In our work, we introduce a novel scheduling protocol which dedicates to disseminate scheduling events happened on each involving node to as many candidate nodes as possible. By means of the proposed protocol, scheduling process of each received job consists of several phases with awareness of grid volatility, and dynamic scheduling and rescheduling is allowed as long as the job execution has not started yet. To this end, a set of concerning algorithms and processing steps are described. A prototype of our scheduling approach is being implemented within the SmartGRID project.

***Index Terms*—Community-Aware Scheduling Protocol, CASP, Grid, Scheduling, SmartGRID, MaGate**

I. Motivation

The Grid concept, first introduced in [1] does not cease to gain in importance during the last years as a medium-term challenge of what is often called the "Computing Internet". The main goal of grid computing is to provide high performance computing facilities over large computers networks. In contradiction to conventional distributed computing systems that are confined in controlled environment with finite and stable resources, grid computing environments are inherently more open as computing elements can join or leave the system at any point in time. This characteristic makes hard to maintain consistent information about the overall state of the system. Moreover, its impractical to maintain information in a centralized way, and the outdated information may thereby lead to wrong scheduling decisions as well. In this context, scheduling policies for achieving good resource management decisions in the presence of such unpredictability become an essential part of a computational grid.

The scheduling system is mainly categorized into two types: the local scheduler and the grid scheduler. The local scheduler (e.g., [2] [3]) is working in local computational environment, which normally stands for reliable and fast connection, uniform environment, and full control of homogeneous resources. Regarding the research on local scheduler is long time before the proposal of grid computing, addtional non-trivial issues introduced by grid computing have to be addressed by complementary components, namely the grid schedulers.

The conventional grid scheduler, also known as meta-scheduler sometimes (e.g., [4] [5]), is responsible for orchestrating resources managed by diverse local schedulers together, in order to bridge the gap between grid applications and isolated local resource management systems. Both local scheduler and a grid scheduler are crucial components for grid computing because it determines the effectiveness and efficiency of a grid system by identifying, characterizing, discovering, selecting, and allocating the resources that are best suited for each individual job.

Although some research work [6] [7] [8] [9] have started to exploit the incentive benefited from cooperation amongst multiple grid nodes, aforementioned solutions still have several common constraints, including: (a) scheduling algorithms mainly aim at optimizing the job queue of hosting grid node; (b) the scope of grid system is assumed to be known a-priori.

With respect to existing solutions, a novel approach named Community-Aware Scheduling Protocol (CASP) is introduced in this work. The design principle of CASP is to provide scheduling solution for the scope of overall grid (or reachable grid community), instead of each single grid node. More precisely, as soon as an involving grid node receives a submitted job from its end user, the CASP broadcasts submitted job profile to the grid community for a lightweight and fast grid-wide pre-scheduling. Additionally, the winner node of the pre-scheduling can inform other remote nodes about local schedule made for the assigned job before its execution; therefore, if another node happens to be able to provide better solution compared to the existing job schedule due to various reasons such as node join or job cancellation, CASP dynamic (re-

)scheduling process can help to achieve load balance with up-to-date information under a decentralized and distributed grid infrastructure.

The remainder of the paper is organized as follows: principle of CASP is introduced in Section II, followed by a details discussion in Section III. Section IV illustrates the planned CASP implementation in a realistic project. Finally, Section V presents conclusion of current work and some insights to future development.

II. Community-Aware Scheduling Protocol Principle

The basic idea of Community-Aware Scheduling Protocol (CASP) is to make scheduling decisions upon grid community, instead of isolated job queue on each grid node. The protocol dedicates to spread the all involving scheduling events, e.g., job submission and job queue optimization, across the network in order to reach as many candidate nodes as possible. Purpose of the Community-Aware Scheduling Protocol is to serve the entire grid community as a whole. The CASP is comprised of two main phases:

1) Job Submission Phase: Grid users can submit jobs to any nodes of the grid community. Each job is identified by an UUID [10]. The initial job recipient, referred as the *initiator*, issues a resource discovery across the grid overlay by broadcasting a REQUEST message. A REQUEST contains the profile of the resources required to carry out the job, an estimated job running time referring to a grid-level accepted computing power baseline, as well as additional job execution constraints (such as Virtual Organization or geographic limitations). The *initiator* then waits for a predefined timelapse for incoming query replies.

A node receiving a REQUEST messages checks if the required profile matches its own capability. Accordingly, it computes an estimated completion time/cost based on actual resources and current scheduling, and forwards this information by means of an ACCEPT message.

It's noteworthy that in case of failed REQUEST messages dissemination by adopted information system due to various reasons, such as network delay, profile non-matching, the *initiator* node can still transfer the generated REQUEST messages to its neighboring nodes, which are known due to existing historical interaction [11] and weighted by means of Critical Friendship [12]. Moreover, message broadcasting via resource discovery systems and historical Critical Friendship can be concurrently.

The *initiator* evaluates incoming ACCEPT responses, and selects the best suited node (i.e. the node providing the least time to completion or lowest execution fee). The latter is assigned the job by means of an ASSIGN message, and is referred to as the *assignee*.

2) Dynamic Rescheduling Phase: The *assignee* is responsible to manage and execute the assigned job according to its own scheduling mechanism and policy. Based on the *initiator*'s offer selection mechanism, the *assignee* is the node that provides the shortest time-to-completion or lowest cost for that particular job. Nevertheless, availability of resource on the grid and scheduling of jobs may change. This can be either the result of new nodes connecting to the grid or existing job cancelation. Thus, the *assignee* may not remain the best solution.

Accordingly, while execution has not yet started, a number of INFORM messages are sent over the network using a low-overhead random walking protocol. INFORM messages' content is similar to REQUEST messages, but their purpose is to inform other nodes about the job's current schedule on the *assignee* node. More specifically, INFORM messages not only contain the profile of the job, but also the estimated solution on current *assignee*. A node will typically generate INFORM messages for several jobs that are at the end of the scheduling queue, and thus have larger waiting times that open up more possibilities for dynamic rescheduling.

A node receiving an INFORM message checks if it matches its up-to-date profile [13]. Furthermore, it evaluates an estimated value (e.g. completion time) according to its own schedule. If such estimation leads to a better result than the INFORM message, the node will send an ACCEPT message to the node that currently manages the job. Threshold are put in place to determine if the re-assign is worth (i.e. do not accept if the improvement is just few better).

The current *assignee* receiving the message may then choose to re-assign the job by means of an ASSIGN message. To enable tracking of jobs for the purpose of node crash tolerance and neighboring node weighting, each re-assignment is logged and notified to the *initiator* node.

III. Community-Aware Scheduling Protocol Implementation

A comprehensive description of the Community-Aware Scheduling Protocol is illustrated as below:

A. Job Dispatch and Evaluation

As mentioned above, in contrast to conventional grid, job submission information on each *initiator* node will be analyzed in order to generate a corresponding REQUEST message, which contains job profile, estimated processing time, resource requirement, and a set of additional constraints. The generated REQUEST messages are then sent to some remote nodes by means of the adopted underly systems, as shown in algorithm 1. Either an efficient resource discovery system or a cached neighboring node list with up-to-date information can fulfill the target. Regarding the volatility of grid, the employed message dispatching system is expected to be lightweight, fast, and being able to reach as many remote nodes as possible. A recommended solution will be introduced in section IV.

Afterward, the *initiator* node waits for a while, and evaluates received ACCEPT messages in order to select the best suited node.

Definition 1 (Release Time): The time when the *initiator* node stops receiving newly arrival ACCEPT messages and turns to job assignment is called release time r_j,

Algorithm 1 REQUEST message generation

```
Require: all remote nodes discovery solutions: $allns$;
    each single remote nodes discovery solution: $ns$;
    each returned remote node: $node$;
    profile of the submitted job: $job_j$;
    generated REQUEST message based on $job_j$: $request$;
Require: $f_{request}(job_j)$: local formula of generating REQUEST message based on $job_j$;
1: $request$ = $f_{request}(job_j)$
2: for all $ns \in allns$ do
3:     for all $node \in ns$ do
4:         send $request$ to $node$
5:     end for
6: end for
```

$$r_j = \begin{cases} s_j + \alpha(d_j - c_{ij}) & \text{if } p_{ij} > 0, \\ s_j + \delta(d_j - p_j) & \text{if } p_{ij} = 0. \end{cases} \quad (1)$$

where job_j is submitted at time s_j, estimated job running time referring to a grid-level accepted computing power baseline is p_j, and the predefined job due time, which means when the job is expected to be delivered back to job owner, is d_j. If computing power of the *initiator* node matches resource requirement of job_j, then job_j's estimated processing time on *initiator* node is p_{ij}, and estimated completion time is c_{ij}. $\alpha, \delta \in (0, 1)$ are coefficients used to adjust acceptable delay by the *initiator* node. Larger α and δ mean more time is allowed for each *Job Submission Phase*, but poor system flexibility and robustness might be incurred.

As illustrated by algorithm 2, once a node has received a REQUEST message generated for job_j, it retrieves the objective obj_j from the REQUEST message. If such obj_j could be recognized by local policy, recipient node then evaluates the estimated value according to its local resource characteristic and status. The objective of job_j defined within REQUEST message can be either a simple value (e.g., estimated completion time), or a multiple one (e.g., estimated completion time as the primary objective, and estimated execution cost as the secondary one). After that, if the estimated result for job objective can be determined, a ACCEPT message will be generated and sent back to the *initator* node of job job_j. Regarding the stage of *Job Submission Phase*, no existing schedules will be obtained from the REQUEST message, it will be discussed in subsection III-B.

As presented in algorithm 3, the *initiator* node of job_j checks the received ACCEPT messages at release time r_j, and selects the best suited remote based on a set of necessary local evaluation formulas, including job objective fulfillment, node power, weight of remote node based on previous interaction [11] [12], etc. ASSIGN message will be generated and sent to the selected best node, aka. the *assignee* node, together with job job_j. Only one remote node can receive the generated ASSIGN message, and future incoming ACCEPT messages after time r_j will be ignored by the *initiator* node.

Algorithm 2 ACCEPT message generation

```
Require: received requesting message: $request$
    ($request \in$ { REQUEST message, INFORM message});
    recognizable objective repository: $repository$;
    retrieved objective array from $request$: $objectives$;
    requesting node: $requester$
    ($requester \in$ { initiator node, assignee node});
    final estimation results: $results$;
    each retrieved objective: $obj$;
    each calculated estimation: $res$;
    existing schedule list: $schedules$;
    each existing schedule: $schedule$;
    existing threshold for each $schedule$: $threshold$;
    ACCEPT message for sending to requester node: $accept$;
Require: $f_{estimation}(obj)$: local formula of getting estimated value on a recognizable objective;
    $f_{accept}(results)$: local formula of generating ACCEPT message based on estimation results;
1: $requester \leftarrow request$
2: $objectives \leftarrow request$
3: $schedule \leftarrow request$
4: for all $obj \in objectives$ do
5:     if $obj \in repository$ then
6:         $res = f_{estimation}(obj)$
7:         $results$ += $res$
8:     end if
9: end for
10: if $results \neq \emptyset$ then
11:     if $schedules \neq \emptyset$ then
12:         for all $res \in results$ do
13:             if $res \in schedules$ then
14:                 $schedule \leftarrow schedules$
15:                 $threshold = f_{threshold}(schedule)$
16:                 if $res$ - $threshold \leq 0$ then
17:                     exit
18:                 end if
19:             end if
20:         end for
21:     end if
22:     $accept = f_{accept}(results)$
23:     send $accept$ to $requester$
24: end if
```

Once the *assignee* node receives the transfered ASSIGN message, it retrieves the attached job and puts it into its local job queue. It is noteworthy that decentralized nodes normally are using different scheduling algorithms and policies, e.g., FCFS, Easy Backfilling, Flexible Backfilling, etc. In this case, how to put the assigned job job_j into local queue relies on the preference of different nodes.

B. Dynamic Scheduling and Rescheduling

The work illustrated in subsection III-A describes how a job can be submitted to the entire grid, or at least part of the grid.

Algorithm 3 ASSIGN message generation

```
Require: job to assign: $job_j$;
    all objectives of $job_j$: $objectives$;
    each received ACCEPT message: $acceptMsg$;
    all received ACCEPT messages: $acceptMsgs$;
    answered objective of each received ACCEPT message: $answeredObj$;
    estimated value of each answered objective within each ACCEPT message: $estimatedRes$;
    calculated weight for each $\{answeredObj, estimatedRes\}$ pair: $weight$;
    all approved candidate remote nodes, paired with the calculated $weight$: $candidates$;
    selected best suited remote node: $assignee$;
    generated ASSIGN message: $assignMsg$;
Require: $f_{eva}(answeredObj, estimatedRes)$: local formula of evaluating specific {objective, estimated result} pair; if the estimated value for given objective is acceptable, return a weight value larger than 0; otherwise, return -1;
    $f_{order}(candidates)$: local formula for searching the best suited $acceptMsg$ according to paired largest $weight$ value, as well as historical interaction records;
    $f_{assignMsg}$: local formula of generating ASSIGN message based on job profile $job_j$;
1: $objectives \leftarrow job_j$
2: for all $acceptMsg \in acceptMsgs$ do
3:     for all $\{answeredObj, estimatedRes\} \in acceptMsg$ do
4:         $weight$ = 0
5:         if ($answeredObj \in objectives$)
           & ($f_{eva}(answeredObj, estimatedRes) > 0$) then
6:             $weight$ += $f_{eva}(answeredObj, estimatedRes)$
7:         else
8:             continue
               (abandon this $\{answeredObj, estimatedRes\}$)
9:         end if
10:        $candidates$ += $\{acceptMsg, weight\}$
11:    end for
12: end for
13: $assignee$ = $f_{order}(candidates)$
14: $assignMsg$ = $f_{assign}(job_j)$
15: send $assignMsg$ to $assignee$
```

It can be comprehended as a lightweight pre-scheduling phase with very limited information and scheduling time. Afterward, submitted job will be managed by the *assignee* node under specific local scheduling policies.

Regarding the infrastructure of grid is rather volatile due to various reasons, such as node join, node leave and resource overloaded, the selected *assignee node* in *Job Submission Phase* may no longer the best suited node through time for job job_j. Therefore, a dynamic scheduling and rescheduling procedure through time with up-to-date information is supposed to increase the effectiveness and efficiency of previous scheduling decisions.

Each node that follows the Community-Aware Scheduling Protocol is supposed to check its waiting job queue periodically (algorithm 4). Jobs which are considered can not be well served (for example, jobs have a long wait-for-execution time) by local node will be picked up, and a INFORM message will be generated for each selected job. Such generated INFORM messages will be sent over the network by means of employed information system for the purpose of seeking better candidate nodes. The INFORM message contains both job profile and current schedule on the *assignee* node, so that recipient nodes are able to determine whether estimated solution on their local resource can be better than the *assignee* node. Threshold on specific objectives will also be put into the INFORM message to prevent unnecessary job re-assignment and system shaking. Furthermore, to ensure a fair scheduling process, jobs been visited for generating INFORM message will be ignored next time. Once INFORM messages have been sent out, the *assignee* node waits for a while, which can be calculated using the similar formula in Definition 1 with self-defined coefficients α and δ, and checks the returned ACCEPT messages afterward.

The recipient nodes of INFORM messages determine their responses using the same process discussed in algorithm 2. It is noteworthy that since each INFORM message contains the estimated schedules based on job profile and *assignee* node capability, responding nodes of INFORM messages will be asked to compare their results with existing schedules. The ACCEPT messages won't be sent back to *assignee* node unless enough benefit can be obtained, i.e., the predefined threshold on estimated objective values have to be achieved.

After certain period of waiting, current *assignee* node checks the returned ACCEPT messages for the targeted jobs, selects the best candidates according to algorithm 3, and sends generated ASSIGN messages together with the referred jobs to the selected nodes, i.e., the new *assignee* nodes.

Finally, all successful assignment/re-assignment of job_j will trigger a notification message to job_j's *initiator* node, so that weight of remote nodes on the *initiator* node can be kept up-to-date.

C. Reference Scenario

Figure 1 illustrates a typical working scenario of Community-Aware Scheduling Protocol. Assuming that a grid is comprised of interconnected $Node_i$, $i \in \{A, B, C, ...\}$, *Job X* is submitted to *Node A*. The Community-Aware Scheduling Protocol then leads to the following phases and steps for job assignment and dynamic scheduling:

1) Job Submission Phase:

Step 1: *Job X* (with primary objective as completion time)is submitted to *Node A*.

Step 2: *Node A* receiving the job submission is referred as to the *initiator*. It generates a REQUEST message according to the retrieved requirement of *Job X*, and broadcasts such message to the grid by the employed lightweight decentralized information system.

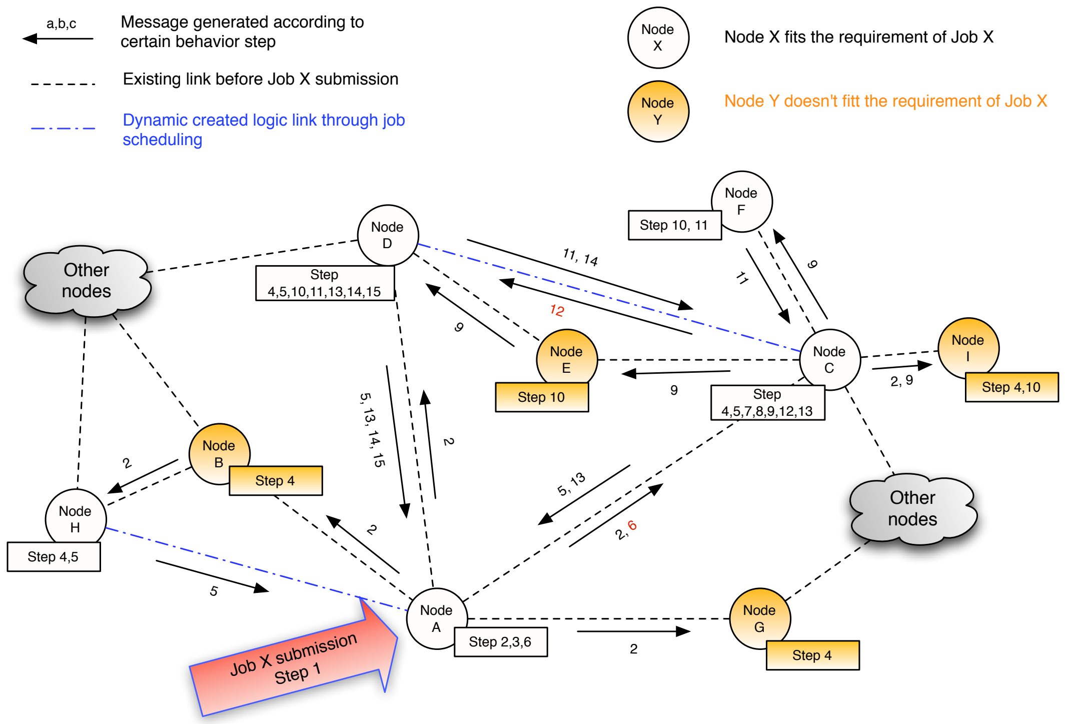

Fig. 1. Community-Aware Scheduling Protocol Reference Scenario

Step 3: The *initiator Node A* then waits for a predefined retardation, which is calculated according to the estimated processing time and due date of *Job X*.

Step 4: Nodes receiving the broadcasted REQUEST message (*i.e. Node B, C, D, G, H, I*) check if the the required job profile can be matched by the local resources.

Step 5: If yes, each candidate node (*Node C, D, H*) computes an estimated completion time according to its current scheduling and resource status, and delivers the information by means of an ACCEPT message.

Step 6: The *initiator Node A* evaluates received ACCEPT messages and selects the best candidate node based on several parameters, such as the promised time to complete, node weight due to historical interaction records, etc. The selected node, referred as the *assignee*, is assigned the job by means of an ASSIGN message. In our case, *Node C* is considered as the best candidate and *Job X* is assigned.

2) Dynamic Rescheduling Phase:

Step 7: The *assignee Node C* takes the assigned *Job X* and puts it into its local job queue.

Step 8: The *assignee Node C* periodically picks jobs from its local job queue, which have large enough waiting time and not been selected recently. Afterward, the *assignee Node C* generates INFORM messages, which contains both job profile and estimated completion time on the itself, for each selected jobs. In our case, regarding *Job X* is just appended to end of queue of *Node C*, a corresponding INFORM message is also generated for *Job X*.

Step 9: The INFORM message for *Job X* is sent over the network using the employed low-overhead walking protocol. As illustrated in Figure 1, *Node E, D, F, I* have received such message.

Step 10: A node receiving aforementioned INFORM message (*Node E, D, F, I*) checks if the local resource and scheduling status could satisfy the profiled job; furthermore, it also evaluates whether the estimated completion time on local resource is worthy enough (i.e. the threshold mentioned in the INFORM message can be fulfilled).

Step 11: If the evaluation result from above step is positive, an ACCEPT message will be generated and delivered to the *assignee Node C*.

Step 12: The *assignee Node C* evaluates the received ACCEPT messages according to the message content and node weight, which is calculated based on historical interaction records. In our case, we assume that *Node C* already has some historical collaboration records on *Node D* and *Node F*, which has revealed that *Node F* is rather unstable and used to halt during

Algorithm 4 INFORM message generation

```
Require: local job queue of assignee node: $jobQueue$;
    each job in assignee node's local queue: $job$;
    $job$'s existing schedule on assignee node: $schedules$;
    generated INFORM message: $inform$;
    all generated INFORM messages: $inform\_queue$;
    thresholds for each existing schedules: $thresholds$;
    all remote nodes discovery solutions: $allns$;
    each remote nodes discovery solution: $ns$;
    each discovered remote node: $node$;
Require: $f_{reassign}(job)$: local formula to determine whether
    a job needs to be reassigned;
    $f_{visitable}(job)$: local formula to check if a job has already
    been tried for reassign last time; if yes, then such a job
    will be skipped this time; otherwise, continue the dynamic
    scheduling process
    $f_{schedule}(job)$: fetch existing schedules of a job on the
    assignee node, i.e., according to local resource, the esti-
    mated values of job objectives;
    $f_{threshold}(schedules)$: local formula of calculating
    threshold for each existing schedule;
    $f_{inform}(job, schedules, thresholds)$: local formula of
    generating INFORM message based on job profile,
    fetched existing schedule, and calculated schedule thresh-
    old;
    $f_{disable\_nextvisit}(job)$: mark a job cannot be reassigned
    next time;
    $f_{enable\_nextvisit}(job)$: mark a job can be reassigned next
    time;
1: for all $job \in jobQueue$ do
2:   if $f_{reassign}(job)$ then
3:     if $f_{visitable}(job)$ then
4:       $schedules = f_{schedule}(job)$
5:       $thresholds = f_{threshold}(schedules)$
6:       $inform = f_{inform}(job, schedules, thresholds)$
7:       $inform\_queue$ += $inform$
8:       $f_{disable\_nextvisit}(job)$
9:     else
10:      $f_{enable\_nextvisit}(job)$
11:    end if
12:  end if
13: end for
14: for all $inform \in inform\_queue$ do
15:   for all $ns \in allns$ do
16:     for all $node \in ns$ do
17:       send $inform$ to node
18:     end for
19:   end for
20: end for
21:
```

nights; therefore, although both *Node D* and *Node F* can bring up the job completion time (no big difference on completion time is assumed), *Node D* is selected and *Job X* is re-assigned by means of an ASSIGN message.

Step 13: To enable tracking of jobs for the purpose of node crash tolerance, each re-assignment is logged and notified to the *initiator Node A*.

Step 14: To enable node weighting for future scheduling, job completion status is sent back to the original *assignee Node C* and *initiator Node A*.

Step 15: The final job execution result is sent back to the *initiator Node A*.

IV. Community-Aware Scheduling in SmartGRID

To evaluate the idea of Community-Aware Scheduling Protocol, a prototype is being implemented under an ongoing grid scheduling framework named the SmartGRID [14] [15].

A. SmartGRID Framework

SmartGRID is a generic and modular framework that supports intelligent and interoperable grid resource management using swarm intelligence algorithms. SmartGRID is structured as a loosely coupled architecture, which is comprised of two layers and one internal interface, as shown in Figure 2.

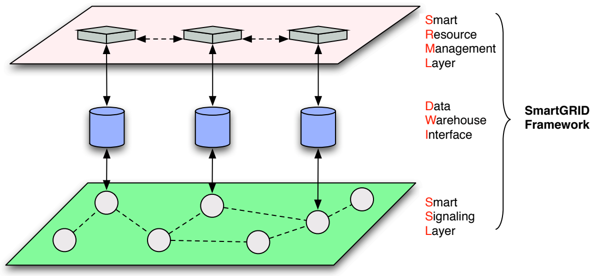

Fig. 2. SmartGRID architecture overview

1) Smart Resource Management Layer (SRML): SRML is responsible for grid level dynamic scheduling and interoperation serving grid applications with dynamically discovered computing resources. SRML is composed of all the engaged *MaGates* schedulers [16] [17]. Each participating MaGate shares the received jobs with other schedulers of the same grid community, which are normally managed under different administrative constraints and policies. Moreover, a set of other relevant issues are also targeted, such as utilizing dynamic resource discovery service and open structured for cooperating with external grid components. In order to address different purposes within an uniform and loosely coupled architecture, the MaGate scheduler is modular designed, as illustrated in Figure 3:

- the *Kernel Module* is responsible for MaGate self-management and addressing of internal events;
- the *Community Module* tackles the interoperation with external schedulers;

- the *LRM Module* performs job allocation and management on Local Resource Management systems (LRM);
- the *External Module* interacts with external grid services for additional functionalities;
- the *Interface Module* manages the interface for accepting job submission from various local invokers, and delivers the results back.

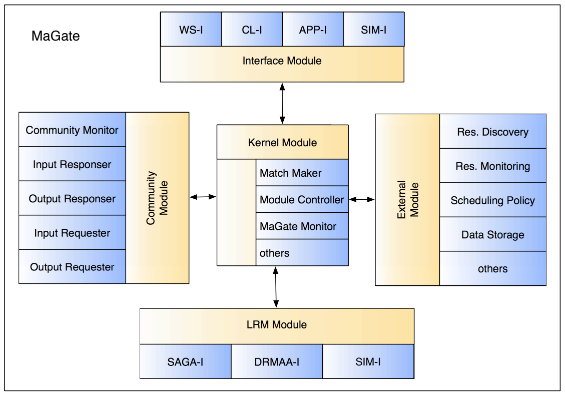

Fig. 3. MaGate modular architecture

2) The Smart Signaling Layer (SSL): SSL represents the interface from and to the network of the SmartGRID framework, and provides information about the availability of resources on other nodes, as well as their status. The SSL hides the complexity and instability of the underlying network by offering reliable services based on distributed ant algorithms. Ants are defined as lightweight mobile agents traveling across the network, collecting information on each visited node. A middleware named *Solenopsis* [18] is developed to run each ant node, providing an environment for the execution of ant colony algorithms, specially the BlåtAnt collaborative ant algorithm [19], [20].

B. Community-Aware Scheduling in SmartGRID

To implement Community-Aware Scheduling Protocol within the context of SmartGRID, the MaGate schedulers of SRML are responsible for making scheduling decisions, while Ant Nests of SSL are in charge of remote nodes discovery and messages broadcasting.

1) Implementation within SRML: Each MaGate scheduler of SRML needs to implement following services and interfaces, which are grouped into corresponding MaGate modules:

Interface Module

- receiving job submissions from end users.
- retrieving and validating job profile.

Kernel Module

- generating REQUEST messages based on job profiles.
- transferring generated REQUEST messages to the *Community Module* and *External Module*.
- calculating received REQUEST messages and generating ACCEPT messages if necessary.
- transferring generated ACCEPT messages to the *Community Module*.
- determining the best candidate node for specific jobs according to the received ACCEPT messages, and generating corresponding ASSIGN messages.
- transferring generated ASSIGN messages to the *Community Module*.
- picking up jobs that cannot be well served by local resource, and generating INFORM messages.
- transferring generated INFORM messages to the *Community Module* and *External Module*.
- evaluating received INFORM messages and generating ACCEPT messages if necessary.

Community Module

- sending locally generated REQUEST message to known neighboring nodes, e.g. critical friend nodes [12].
- sending locally generated INFORM messages to known neighboring nodes.
- collecting received REQUEST messages and transferring them to the *Kernel Module*.
- collecting received INFORM messages and transferring them to the *Kernel Module*.
- collecting received ACCEPT messages and transferring them to the *Kernel Module*.
- sending locally generated ASSIGN messages to remote nodes for job delegation.
- sending locally generated ACCEPT messages to remote nodes.

External Module

- broadcasting generated REQUEST messages over the network via employed resource discovery system, i.e., Ant-based Smart Signaling Layer (SSL).
- broadcasting generated INFORM messages over the network via employed resource discovery system.
- receiving REQUEST messages from SSL and transfer them to the *Kernel Module*.
- receiving INFORM messages from SSL and transfer them to the *Kernel Module*.

2) Implementation within SSL: Similarly, each Ant Nest of the SSL is also supposed to provide a set of supports, including:

- broadcasting transfered REQUEST messages from the Smart Resource Management Layer (SRML).
- broadcasting transfered INFORM messages from the SRML.
- collecting received REQUEST messages and transferring them to the *Community Module* of the interconnected SRML MaGate scheduler.
- collecting received INFORM messages and transferring them to the *Community Module* of the interconnected MaGate scheduler.

Moreover, regarding that each individual MaGate scheduler may use different scheduling policies, all local formulas mentioned in algorithm 1, 2, 3, and 4 need to be implemented according to adopted local polices.

V. Conclusion and Future work

The paper presents a novel scheduling approach named the Community-Aware Scheduling Protocol (CASP), which is inspired by the motivation of enable grid scheduling for the scope of the overall grid, instead of each single node. In contradiction to conventional grid scheduling solutions, the CASP is supposed to broadcast all relevant scheduling events to as many candidate remote nodes as possible, including job submission, job scheduled information, etc. Moreover, the CASP enables the possibility of dynamic (re-)scheduling in order to make full use of grid characteristics such as instantaneity and volatility.

The CASP is comprised of two phases: the *Job Submission Phase* and the *Dynamic Rescheduling Phase*. The *Job Submission Phase* is responsible for broadcasting job submission to the grid, and as soon as the best candidate has been selected, the submitted job will be assigned to the so called *assignee* node.

The *Dynamic Rescheduling Phase* is in charge of monitoring jobs that can not be well served on each grid node, informing local schedules made for those jobs to other remote nodes, and enabling job rescheduling if better offer can be discovered.

The Community-Aware Scheduling Protocol is being implemented into an ongoing grid project named SmartGRID. A prototype of SmartGRID scheduling layer has been implemented [17], as well as an Ant-based information systems [18] that can be used as message dissemination system employed by the CASP. Besides, Grid Workload Archive [21] based trace data will be utilized to simulate the expected behavior of CASP in realistic grid environment.

Further more, regarding the trend that computational resources are no longer hard to obtain due to technique evolution, especially the emerging resource virtualization and Cloud Computing, the basic purpose of scheduling has been shifted from finding expected resources for specific jobs (provision centered) into finding/generating appropriate resources for specific jobs under certain cost and constraints (demand centered). In this case, idea of job broadcasting and dynamically (re-)scheduling introduced by the CASP proposes a potential theoretical and practical direction in this field.

VI. Acknowledgments

This research is mainly financially supported by the Swiss Hasler Foundation in the framework of the “ManCom Initiative”, project Nr. 2122, and partly supported by EU Marie Curie Knowledge Transfer Programme.

References

[1] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200, 2001.

[2] openPBS. http://www.openpbs.org/, 2009.

[3] LSF. http://www.platform.com/Products/platform-isf, 2009.

[4] E. Huedo, R.S. Montero, and I.M. Llorente. The GridWay framework for adaptive scheduling and execution on grids. *Scalable Computing: Practice and Experience*, 6(3):1–8, 2005.

[5] O. Waldrich, P. Wieder, and W. Ziegler. A meta-scheduling service for co-allocating arbitrary types of resources. *Lecture Notes in Computer Science*, 3911:782, 2006.

[6] V. Yarmolenko and R. Sakellariou. Towards increased expressiveness in service level agreements. *Concurrency and Computation: Practice and Experience*, 19(14), 2007.

[7] R. Ranjan, A. Harwood, R. Buyya, and A. Victoria. SLA-based coordinated superscheduling scheme for computational Grids. In *IEEE International Conference on Cluster Computing (Cluster 2006), Barcelona, Spain*, pages 1–8. Citeseer, 2006.

[8] C. Grimme, J. Lepping, and A. Papaspyrou. Prospects of collaboration between compute providers by means of job interchange. *Lecture Notes in Computer Science*, 4942:132, 2008.

[9] V. Yarmolenko, R. Sakellariou, D. Ouelhadj, and J.M. Garibaldi. SLA based job scheduling: A case study on policies for negotiation with resources. In *Proceedings of e-Science All Hands Meeting (AHM2005)*, pages 20–22, 2005.

[10] P. Leach, M. Mealling, and R. Salz. A universally unique identifier (uuid) urn namespace. *RFC4122, July*, 2005.

[11] Y. Huang, N. Bessis, A. Brocco, P. Kuonen, M. Courant, and B. Hirsbrunner. Using Metadata Snapshots for Extending Ant-based Resource Discovery Service in Inter-cooperative Grid Communities. In *Proceedings of the 1st International Conference on Evolving Internet*, pages 89–94, Cannes, French Riviera, France, 2009. INTERNET 2009, IEEE Computer Society.

[12] Y. Huang, N. Bessis, A. Brocco, S. Sotiriadis, M. Courant, P. Kuonen, and B. Hirsbrunner. Towards an integrated vision across inter-cooperative grid virtual organizations. In *Future Generation Information Technology*, pages 120–128, Jeju Island, Korea, 2009. FGIT 2009, Springer.

[13] N. Bessis. *Grid Technology for Maximizing Collaborative Decision Management and Support: Advancing Effective Virtual Organizations*, chapter Model Architecture for a User Tailored Data Push Service in Data Grids. IGI, 2009.

[14] B. Hirsbrunner, M. Courant, A. Brocco, and P. Kuonen. SmartGRID: Swarm Agent-Based Dynamic Scheduling for Robust, Reliable, and Reactive Grid Computing. Technical report, Working Paper 06-13, Department of Informatics, University of Fribourg, Switzerland, 2006.

[15] Y. Huang, A. Brocco, P. Kuonen, M. Courant, and B. Hirsbrunner. SmartGRID: A Fully Decentralized Grid Scheduling Framework Supported by Swarm Intelligence. In *Seventh International Conference on Grid and Cooperative Computing, 2008. GCC ’08*, pages 160–168, China, 2008. IEEE Computer Society.

[16] Y. Huang, A. Brocco, M. Courant, B. Hirsbrunner, and P. Kuonen. MaGate: an interoperable, decentralized and modular high-level grid scheduler. *International Journal of Distributed Systems and Technologies (IJDST)*, 2010.

[17] Y. Huang, A. Brocco, M. Courant, B. Hirsbrunner, and P. Kuonen. MaGate Simulator: A Simulation Environment for a Decentralized Grid Scheduler. In *Proceedings of the 8th International Symposium on Advanced Parallel Processing Technologies*, page 287. Springer, 2009.

[18] A. Brocco, B. Hirsbrunner, and M. Courant. Solenopsis: A Framework for the Development of Ant Algorithms. In *Swarm Intelligence Symposium*, pages 316–323, Honolulu, Hawaii, April 2007. SIS, IEEE.

[19] A. Brocco, F. Frapolli, and B. Hirsbrunner. BlatAnt: Bounding Networks’ Diameter with a Collaborative Distributed Algorithm. In *Sixth International Conference on Ant Colony Optimization and Swarm Intelligence*, Bruxelles, September 2008. ANTS, Springer.

[20] A. Brocco, F. Frapolli, and B. Hirsbrunner. Bounded Diameter Overlay Construction: A Self Organized Approach. In *IEEE Swarm Intelligence Symposium*, Nashville, Tennessee, USA, April 2009. SIS 2009, IEEE.

[21] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D.H.J. Epema. The grid workloads archive. *Future Generation Computer Systems*, 2008.