Agent-Based Decentralized Grid Model

HUI KANG1,2, JIAHUI LI1, LIN ZHANG3, TIE FENG1, QIUWEN YIN4, AND YUJUN LIU1

1College of Computer Science and Technology, Jilin University, Changchun 130012, China
2Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China
3Financial Department, Jilin University, Changchun 130012, China
4Jilin Traffic Planning and Design Institute, Changchun 130012, China

Corresponding author: Tie Feng (fengtie@jlu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFC0831706, and in part by the Science and Technology Development Plan Project of Jilin Province under Grant 20200201166JC.

ABSTRACT
Decentralization of grid systems plays an important role in improving their efficiency and fault tolerance. To enhance the performance and stability of grid and mitigate the problems of centralized grid, an agent-based decentralized grid model (ADGM) with universality and functional integrity is proposed. In this paper, we build an agent-based grid structure and propose an agent-based grid consensus algorithm (AGCA). A group membership protocol, a consistency protocol and a view change protocol in AGCA are also designed. Furthermore, based on Pi calculus, the information registration, resource sharing and error recovery of the grid model in the decentralized environment are formally specified. Finally, we analyze the performance of the AGCA algorithm and compare it with other consensus algorithms, the simulation results demonstrate that the AGCA algorithm achieves better comprehensive performance and more symmetrical time performance, space performance and fault tolerance than other consensus algorithms.

INDEX TERMS
Agent-based modeling, grid computing, decentralization, Pi calculus.

I. INTRODUCTION
As the scale and complexity of networks increase [1], people must interconnect resources that are distributed around the world to realize comprehensive resource sharing to eliminate resource islands and information islands, so that they can collaborate to solve large-scale computing problems; hence, grid computing was developed.

A grid is a type of system that enables the synergistic use of resources in a distributed environment using standard, open, and common protocols and interfaces while also providing exceptional quality of service [2]. Due to their convenience and high performance, the grid systems have been used for most complex scientific, engineering and business problems that need a huge amount of resources for execution [3], [4], [5].

Most grid systems use a central server to centrally manage the structure and resource information of the grid. Due to the single point of failure and frequent communication with the central server, the centralized management method limits the scalability and fault tolerance of the system and restricts the system’s further improvement in terms of computing performance and scale [6]. Therefore, it is of substantial significance in improving the efficiency and fault tolerance of the grid system to study a decentralized model of the grid system.

However, the current research on the decentralization of the grid system is insufficient. Existing frameworks and models of decentralized grids are proposed in specific application contexts [7], [8], which means that they are difficult to be expanded to suit more general applications. Moreover, the functions of these frameworks and models are not generic [9], [10], which means that they cannot perform the all basic functions of the grid in a decentralization environment. In addition, there is also a lack of uniform quantitative standards for performance, efficiency, and fault tolerance in decentralized grid models.

To overcome the abovementioned issues, we propose an agent-based decentralized grid model (ADGM) with generality and functional integrity in this work. To realize the basic functions of the decentralized grid systems, such as the operations of joining, leaving, searching and acquiring resources of the grid nodes, an agent-based grid structure is constructed, and an agent-based grid consensus algorithm, namely, AGCA, is further proposed. Moreover, by analyzing the characteristics and requirements of the grid systems, we set up experimental items in the simulation, thereby verifying and evaluating the proposed algorithm.
The main contributions of our proposed scheme are summarized as follows:

1) Based on agent technology and mechanism of autonomously reaching consensus among grid nodes, we propose a decentralized grid model ADGM with characteristics of universality and function integrity to solve the shortcomings of centralized grid models.

2) By designing protocols for the stipulation of the grid nodes, we propose a grid consensus algorithm AGCA to implement the all basic function of grid systems and enhance the effectiveness and fault tolerance of grid models.

3) We formally describe the group membership protocol, consistency protocol, and view change protocol in AGCA based on Pi calculus, which can show and verify these protocols from a mathematical perspective. Moreover, it may have some inspirations for the modeling of other complex and dynamic concurrent systems.

4) By employing the nPict language, we implement ADGM in simulation. Moreover, we compare AGCA with other consensus algorithms in terms of time performance, space performance and fault tolerance, thereby verifying the effectiveness of the proposed algorithm.

The remainder of the paper is organized as follows: Related research on grid decentralization models is presented in Section II. In Section III, the structure of the agent-based grid model and the principles of AGCA are analyzed. The three protocols are formally described based on Pi calculus in Section IV. Furthermore, the results of a performance evaluation analysis of AGCA are presented in Section V. The final conclusions of this study are presented in Section VI.

Notations: For ease of exposition, in this paper, grid nodes are represented in bold font, variables and processes are represented in italics, and messages are highlighted by being enclosed in angle brackets, e.g., <message>.

II. RELATED WORK
Few decentralized models of grid systems are available. Scholars have conducted work on the decentralization of grid systems and have proposed several decentralization methods of grid systems. These methods differ in terms of the technologies that are used, the contexts that are targeted, and the decentralization phase. The representative methods are introduced and summarized in this section.

To realize market-based resource allocation in the grid systems, Kang and Parkes [7] propose a decentralized auction framework for dynamic resources in a computational grid. The framework leverages simplifying assumptions of "uniform failure" and "threshold reliability", which boosts the consensus between the end-user and the resource owner in a decentralized environment. However, this framework provides a decentralization method only for the dynamic resource auction context of grid systems and is not applicable to the other general contexts.

Wu and Xu [11] propose a gossip-based reinforcement learning (GRL) method for decentralized job scheduling in grid systems. In the GRL method, a decentralized scheduling architecture that is based on multi-agent reinforcement learning is presented for improving the scalability and adaptability of job scheduling. A gossip mechanism is designed for realizing autonomous coordination among the decentralized schedulers. However, this method only realizes the decentralization of the resource allocation stage of grid models and does not provide a decentralization scheme for other grid operations.

Abbes and Louati [9] propose a rollback-recovery protocol, which is based on checkpoints designed for the decentralized desktop grid systems. It provides fault tolerance for grid applications and ensures the termination of the execution of applications in a transparent way to users. However, the protocol only solves the problem of fault tolerance in a decentralized environment, thus its functionality needs to be further improved.

To identify an appropriate resource share mechanism for decentralized grid jobs, Shaikh et al. [10] propose a semantic resource discovery model by using semantic similarity threshold values and extended ontologies in a decentralized resource discovery model of grid computing. This model can improve success probability for complex jobs and reduce communication overheads. However, the model is only suitable for the resource share stage and cannot be extended for more generic applications for decentralized grid systems.

The authors in [12] present a communication model for decentralized meta-scheduler in grid environments, which employs two types of agents: one for resource management (Broker), and the other manages the user task requests (Agents). Moreover, the paper describes the communication protocol between agents and the proposed structure for agents. However, this model does not consider the specifying protocol to the grid nodes, which is difficult for more generic extends.

In the methods that are discussed above, the decentralized model of the grid is insufficient in terms of performance, efficiency, and versatility. In contrast to available immature methods, we focus on designing a universal and functional decentralized grid model, which can realize the operations of joining, leaving, searching and acquiring resources of the grid nodes.

III. AGENT-BASED DECENTRALIZED GRID MODEL–ADGM
In this section, we first formulate an agent-based grid structure for further decentralized research. Then, the principles of AGCA are described, in which the group membership protocol, consistency protocol and view change protocol are elaborated in detail.

A. AGENT-BASED GRID STRUCTURE
Agent technology is widely used in grid systems [13], [14]. An agent can sense the environment, take appropriate actions to establish its own code of conduct and influence environmental change, which provides an effective tool for solving
network distributed application problems [15]–[17]. Thus, agent technology is introduced for improving the scalability, stability and efficiency of grid systems in this model.

Accordingly, in this part, we first abstract the operating mode of the grid systems. Then, we present the agent-based grid structure, in which the behavior and role of the agent are introduced.

1) VIEW

The correct nodes in the grid observe the changes of the nodes in the group in a consistent form, and a group that has reached such a consensus is called the view[18], which can be numbered and recorded as \( v \). In the agent-based grid model, a node is configured as the primary node (primary) according to \( v \), while other nodes are backup nodes (backups). All operations are based on the rotation of the view. Before the system starts, the system administrator assigns an initial view and determines the member nodes in the grid. We initialize \( v = 0 \), and we define a finite non-empty set \( R \) for storing the member nodes, which are numbered sequentially by \( i \) in the view. For example, if there are \( n \) nodes in the system, then \( R = \{ r_1, r_2, \ldots, r_n \} \). Member \( R_i \) in the view can be recorded as \( R^i \); hence, the initialization condition is as follows:

\[
R^i = \begin{cases} 
R & R_i \in R \\
\text{Undefined} & R_i \notin R
\end{cases}
\]  

(1)

2) GRID STRUCTURE

As illustrated in Fig. 1, in the agent-based grid model, the agent is the medium for information transfer between the grid nodes; hence, each grid node provides the information that is required by the agent and simultaneously obtains the execution result of the agent. In each view, primary organizes backups to complete the negotiation process. The behaviors of the agent are as follows:

1) Autonomously migrate tasks from one node to another.
2) Interact with other agents to realize resource management and adaptation.
3) Move bi-directionally between grid nodes and transfer information such as resources, loads, traffic, and task execution sequences.
4) Intelligently judge the scenario of the management domain and handle it accordingly.
5) Be created, copied, and deleted by grid nodes.

B. AGENT-BASED GRID CONSENSUS ALGORITHM

In a centralized grid system, the central node must be notified to handle related transactions when the grid undergoes structural and functional changes. To realize the decentralized grid system, the algorithm should ensure the autonomy and independence of the grid nodes so that the grid nodes can reach a consensus on the grid structure and actions without control. Therefore, AGCA constrains the behaviors of nodes by designing protocols so that they negotiate the completion of node joining, node leaving, resource sharing, and processing of failed nodes.

The group membership protocol, consistency protocol, and view change protocol are core elements of AGCA. Their structure is illustrated in Fig. 2, in which the dotted line indicates that the group member protocol and the consistency protocol are completed on the basis of the view change protocol, while the solid line indicates that the group member protocol and the coherency protocol are serially completed to perform functional work. Next, their roles and principles are described.

1) GROUP MEMBERSHIP PROTOCOL

The group member protocol is used for information registration of all nodes in the grid system, which includes a node joining protocol and a node leaving protocol, and the node leaving protocol is further divided into an active leaving mode and a negotiated leaving mode. The operation of this protocol only updates the current system state and does not trigger a view change.

When an outer node \( q \) wants to join the grid system, the node joining protocol is triggered. Assume that the joining node is valid, the number of nodes in the grid is \( 3f + 1 \), and the node can only be added once (a node that rejoins after exiting
is regarded as a brand new node). The negotiation process of the protocol can be described as follows.

As shown in Algorithm 1, the node \( q \) that requests to join sends the Ask_in agent to all the grid nodes, and a joining request, which includes the timestamp \( t \), the node public key \( pk \) and the certificate \( cert \), is sent to the grid node where the agent is located. After receiving the joining request, primary confirms the certificate \( cert \) and assigns a new number, namely, \( new_id \), for \( q \). Then, it broadcasts the node joining message, namely, \(<\text{new_in}>\). If no view change has occurred, grid node \( i \) receives and acknowledges \(<\text{new_in}>\) and broadcasts the acknowledgement message, namely, \(<\text{ack_in}>\). When node \( i \) receives and acknowledges message \(<\text{ack_in}>\) from \((2f + 1)\) nodes, a commit-to-join message, namely, \(<\text{commit_in}>\), will be broadcasted. After \( i \) verifies \(<\text{commit_in}>\) from \((2f + 1)\) nodes, and if \( new_id \) is not used in the local list, a \(<\text{reply}>\) message is returned to Ask_in and the view state of the node is updated to \( R'_i = \{r_1, r_2, \cdots, r_n, q\} \). Ask_in will migrate to \( q \). If \((f+1)\) Ask_in agents migrate from nodes to return information to \( q \), the view state is updated to \( R'_q = \{r_1, r_2, \cdots, r_n, q\} \).

The node leaving protocol is divided into an active leaving method and a negotiated leaving method. The former refers to the process in which a node actively leaves the grid system, and its algorithmic process is similar to the joining process. The latter refers to the process in which a node is detected and removed from the grid system due to an error. The reasons why node \( q \) triggers the node leaving protocol are as follows:

1. \( q \) is a correct node in the grid and has submitted a leaving request to leave the grid system;
2. \( q \) is the failed backup and has submitted a leaving request to leave the grid system;
3. \( q \) is the failed primary. After the view has changed, other nodes that monitored this node should leave the grid system.

Cases a) and b) can be classified as active leaving methods, and case c) can be classified as a negotiated leaving method. The negotiation process of the protocol is described as follows:

### Algorithm 1 Node Joining Protocol

```plaintext
1 Define and initialize the related parameters: the node requests to join \( q \), the primary node primary, nodes set \( R_{\text{nodes}} \), and the number of all nodes \((3f + 1)\), etc.;
2 Node \( q \) sends the Ask_in agent to all the grid nodes;
3 Ask_in agents sends a joining request to its located grid node;
4 Primary broadcasts message \(<\text{new_in}>\); if No view change occurs then
5 Grid nodes broadcasts \(<\text{ack_in}>\);
6 else
7 Grid nodes ignore all request messages;
8 end
9 for Each grid node \( i \) in \( R_{\text{nodes}} \) do
10 if \( i \) received \(<\text{ack_in}>\) from \((2f+1)\) nodes then
11 Node \( i \) broadcasts \(<\text{commit_in}>\);
12 end
13 end
14 for Each grid node \( i \) in \( R_{\text{nodes}} \) do
15 if \( i \) received \(<\text{commit_in}>\) from \((2f+1)\) nodes then
16 Node \( i \) sends a \(<\text{reply}>\) message to Ask_in agent and updates its view state;
17 end
18 end
19 for \((f+1)\) Ask_in agents return its information to \( q \) then
20 Node \( q \) updates its view state;
21 Update and return \( R_{\text{nodes}} \);
22 //\( R_{\text{nodes}} \) is the updated grid nodes set by the algorithm
```

### Algorithm 2 Node Leaving Protocol

```plaintext
1 Define and initialize the related parameters: the primary node primary, the process launching node \( q \), nodes set \( R_{\text{nodes}} \), and the number of all nodes \((3f + 1)\), etc.;
2 Node \( q \) sends the Ask_out agent to all the grid nodes;
3 Ask_out agents sends \(<\text{rqt_out}>\) to its located grid node;
4 for Each grid node \( j \) in \( R_{\text{nodes}} \) do
5 if primary is invalid then
6 The Ask_out agent located on node \( j \) judges the identity of the leaving node;
7 if the node actively requests to leave then
8 Node \( j \) broadcasts \(<\text{ack_out}>\);
9 end
10 if primary is invalid then
11 The grid node \( j \) wait for other nodes to discover that primary is invalid;
12 if \((f+1)\) nodes find primary is invalid then
13 Node \( j \) broadcasts \(<\text{ack_out}>\);
14 end
15 end
16 for Each grid node \( j \) in \( R_{\text{nodes}} \) do
17 if Ask_out agent located on node \( j \) receives at least \((2f+1)\) \(<\text{ack_out}>\) then
18 Node \( j \) broadcasts \(<\text{commit_out}>\);
19 end
20 end
21 for Each grid node \( j \) in \( R_{\text{nodes}} \) do
22 if Ask_out agent located on node \( i \) receives at least \((2f+1)\) \(<\text{commit_out}>\) then
23 The Ask_out agent sends \(<\text{success}>\) message to node \( j \);
24 Node \( j \) updated its view state;
25 end
26 end
27 Update and return \( R_{\text{nodes}} \);
28 //\( R_{\text{nodes}} \) is the updated grid nodes set by the algorithm
```
The pseudo-code of the node leaving protocol is presented in Algorithm 2. As can be seen, first, a node sends the Ask_out agent to all the grid nodes. Moreover, a leaving message, namely, <rq_out>, which contains the id of the leaving node, is sent to the located grid node by the Ask_out agent. The grid node checks the correctness of the received <rq_out> message. Then, Ask_out agent judges the identity of the leaving node by verifying the id. If \( i = 1 \), which means that \( i \) actively requests to leave, the node the agent located broadcasts the acknowledgement leaving message, namely, <ack_out>. If \( i = \text{primary} \), the Ask_out agent continues to wait for other nodes to discover that \( \text{primary} \) is invalid. After receiving \((f + 1)\) messages that \( \text{primary} \) fails, the message <ack_out> is broadcast by the node the Ask_out agent located. When Ask_out agent receives at least \((2f + 1)\) <ack_out> from different nodes, a commit-to-leave message, namely, <cmt_out>, is broadcasted to other nodes. If Ask_out receives at least \((2f + 1)\) <cmt_out> messages from nodes, a successful flag, namely, <success>, is sent to the node where it is located. After the node receives <success>, the view state \( R_i \) is modified. If \( q \neq r_i \), \( q \) is removed from the view. If \( q = r_k \), then

\[
R_i = \begin{cases} 
  r_1 \ldots r_i \ldots r_{k-1}, r_{k+1} \ldots r_n & i < k \\
  r_1 \ldots r_{k-1}, r_{k+1} \ldots r_i \ldots r_n & i > k 
\end{cases}
\]

2) CONSISTENCY PROTOCOL
The consistency protocol enables nodes in the grid to agree on resource sharing. In this paper, a complete set of resource information of other nodes is maintained in all nodes in the grid system. If a grid external node successfully joins the grid system, the node will broadcast its own resource information to other nodes so that the other nodes can update their resource information tables. The operation of this protocol only updates the current system state, and it does not trigger a view change. The negotiation process of the protocol is described as follows.

The pseudo-code of the consistency protocol is shown in Algorithm 3. As can be seen, first, the node \( q \) that is newly added in the grid sends the ResourceShare agent to all grid nodes. \( \text{Primary} \) calculates the hash value according to the information in ResourceShare and saves the result to the set \( \text{MA}_\text{Dight} \). Then, \( \text{primary} \) sends a message, namely, <pre-prepare>, which includes \( \text{MA}_\text{Dight} \) and the id of \( q \), which is denoted as \( \text{MA}_\text{id} \), to \( \text{backups} \). \( \text{Backups} \) performs the same calculations as \( \text{primary} \) after receiving ResourceShare, and \( \text{MA}_\text{Dight} \) of it is compared with the value in the received <pre-prepare>. If they are equal, a message, namely, <prepare>, which includes \( \text{MA}_\text{id}, \text{MA}_\text{Dight} \), and \( v \), is sent to the other nodes. After receiving <prepare>, the node checks whether it is correct and in the current view. If yes, <prepare> is written to the log. When <prepare> messages are received from \( 2f \) nodes, the node is in the prepared state and a <commit> message will be sent. The node receives <commit> and checks whether it is correct and in the current view. If yes, <commit> is written to the log. When <commit> is received from \((2f + 1)\) nodes by the node, the resource information is written into the resource information table of the node. \( \text{Primary} \) stores the updated table in the agent. Agents that are located at other nodes migrate to \( q \). When \( q \) receives \((f + 1)\) correct agents that have the same \( \text{MA}_\text{id} \), the resource information table is saved locally. A graphical representation of the consistency protocol algorithm is shown in Fig. 3.

Algorithm 3 Consistency Protocol

1. Define and initialize the related parameters: the primary node \( \text{primary} \), the newly added node \( q \), backup nodes set \( \text{R}_{\text{backups}} \), and the number of all nodes \((3f + 1)\), etc.;
2. Node \( q \) sends the ResourceShare agent to all the grid nodes;
3. \( \text{Primary} \) calculates \( \text{MA}_\text{Dight} \) and sends a <pre-prepare> message to \( \text{R}_{\text{backups}} \);
4. For Each grid node \( i \) in \( \text{R}_{\text{backups}} \) do
   5.   Node \( i \) calculates \( \text{MA}_\text{Dight} \);
   6.   If \( \text{MA}_\text{Dight} \) calculated by \( i \) is equal to it calculated by \( \text{Primary} \) then
      7.      Node \( i \) broadcasts <commit> message;
   8. end
9. end
10. For Each grid node \( i \) in \( \text{R}_{\text{backups}} \) do
11.   Node \( i \) writes <prepare> to its log;
12.   If node \( i \) receives at least \((2f+1)\) <prepare> then
13.      Node \( i \) broadcasts <commit> message;
14. end
15. end
16. For Each grid node \( i \) in \( \text{R}_{\text{backups}} \) do
17.   Node \( i \) writes <commit> to its log;
18.   If node \( i \) receives at least \((2f+1)\) <commit> then
19.      Node \( i \) writes resource information into the resource information table;
20. end
21. end
22. \( \text{Primary} \) stores the updated table in the agent;
23. Agents located at other nodes migrate to \( q \);
24. If \( q \) receives \((f + 1)\) agents then
25.   The resource information of agents is saved into the resource information table of \( q \);
26. end
27. Update and return \( \text{R}_{\text{nodes}} \);
28. //\( \text{R}_{\text{nodes}} \) is the updated grid nodes set by the algorithm

3) VIEW CHANGE PROTOCOL
The view change protocol is used to overcome the problem that system fails to operate due to \( \text{primary} \) failure or node disagreement. The view change protocol refers to the process of switching the view from \( v \) to \( v + 1 \). During the view change, all actions in the previous view must be stopped before the next view can be converted, and a graphical representation of
the view change protocol is shown in Fig. 4. There are two cases in which the view change protocol will be triggered:

1) An error occurs in primary of view $v$.
2) Backups are always waiting and time out.

As shown in Algorithm 4, after primary fails, backups generate the ViewChange agent that contains part of the information of the node’s log. Then, the ViewChange agent migrates to all nodes in view $v$ and sends a message, namely, $\langle$view-change$\rangle$, which includes $h$, $v+1$, $P$ and $i$, where $h$ is the number of the last stable checkpoint of the current node, $P$ is a table in which each element is a triple {$n$, $d$, $v$}, $n$ is the id of the node that is already in the prepared state, and $d$ is a summary of the message. After the other grid nodes in the view $v$ receive $\langle$view-change$\rangle$ from node $i$ and $(2f + 1)$ confirmations of the message, $\langle$view-change$\rangle$ is recorded into the set $V$. When the number of $\langle$view-change$\rangle$ in the set $V$ reaches $(2f + 1)$, primary generates a set $O$ for storing $\langle$pre-prepare$\rangle$ messages. First, primary selects the smallest $h$ in $\langle$view-change$\rangle$ of the set $V$, which is denoted as $\min$, and selects the largest $n$ in the set $P$, which is denoted as $\max$. Then, we let $n \in (\min, \max)$, and we generate $\langle$pre-prepare$\rangle$ messages, which are numbered by $n$. After the process, primary puts the set $V$ and the set $O$ into ViewChange and enters the $v + 1$ view stage. ViewChange sends a $\langle$new-view$\rangle$ message, which contains $v + 1$, $V$ and $O$, to all nodes in view $v + 1$. After receiving $\langle$new-view$\rangle$, each node verifies the correctness of the message, puts $\langle$pre-prepare$\rangle$ in $O$ into the log, and enters the $v + 1$ view stage if the message is correct. The nodes continue the operations that were being conducted prior to the view change.

4) COMPUTATIONAL COMPLEXITY
The complexity of the proposed protocols is analyzed in this part. We suppose that there are $n$ grid nodes in the grid system. Due to each grid node must send messages to other $n - 1$ nodes in each process of reaching consensus, the complexity of the algorithm is $O(n^2)$. However, the initialization of the agent may consume extra computing time, which is hard to predict. Thus, to obtain the time performance of the algorithm more accurately, the response time of the proposed algorithm is evaluated in Section V.

IV. PROTOCOL SPECIFICATION BASED ON PI CALCULUS
The grid system is a concurrent system, and its network configuration may change during the calculation process, which makes it difficult to model by ordinary methods. In theoretical computer science, Pi calculus is a formal language for describing and analyzing concurrent systems [19], [20], which uses the reduction in calculus to represent the dynamic evolution of inter-process communication [21]. Moreover, in Pi calculus, the channels can be passed as data in other
channels between processes, which makes the expression of Pi calculus to be very powerful [22]. Thus, Pi calculus is utilized for describing the communication mechanism and data processing logic of the decentralized grid model in this part, so that facilitating rigorous mathematical verification. The key specifications are presented and analysed as follows.

**A. NODE JOINING PROTOCOL SPECIFICATION**

According to Fig. 5 and Eq. (2), we use three processes in parallel to complete the tasks of the node joining phase: NewNode, a mobile agent (Ask_inAgent), and all grid nodes (GridNode) in view $v$.

NodeJoin
\[
\text{def} \quad \text{NewNode}|\text{Ask}_\text{inAgent}|\text{GridNode} \quad (2)
\]

NewNode($\overline{r}$)
\[
\text{def} \quad \text{SendInProcess}|\text{AcceptInProcess} \quad (3)
\]

Ask_inAgent($\overline{i}$)
\[
\text{def} \quad \text{inner}(t, q, pk, cert).\overline{trans} < \text{request} > \cdot \text{trans}(\text{node}).([\text{node}.\text{view} = = \text{node}.i] \overline{agent} < \text{new}_\text{in} > |\text{NewInProcess}) \quad (4)
\]

GridNode
\[
\text{def} \quad \text{migrate}, \text{trans}(\text{request}, R^t). < \text{node}_\text{in}, \text{node}, \text{success} > \\
\cdot (\text{trans}(\text{request}).[\text{op} == \text{ask}_\text{in}]\overline{trans} < \text{node} > \cdot (\text{trans}(\text{success})\text{UpdateViewState}(R^t))) \quad (5)
\]

Remark 1:

1) In the NewNode process, two processes are executed in parallel, such as in Eq. (3). The former is SendInProcess, which sends the joining request of NewNode to each node in the view and waits for a response. The latter is AcceptInProcess, which receives the results of the grid nodes and determines whether the joining is successful according to the results.

2) Ask_inAgent is used to indicate the exchange of information between NewNode and GridNode and the negotiation of messages between the agents. The node information of NewNode’s joining request is obtained from the inner channel, and the request is sent to GridNode from the trans channel and GridNode’s information is obtained. If the node where the agent is located is primary in the current view, a <new_in> message is sent to other agents to wait for the result to be returned, as expressed in Eq. (4).

3) GridNode specifies the operation of each grid node in the current grid system in the node joining phase. The Ask_inAgent that has migrated through the migration channel is received and interacted with. If the received message is <ask_in>, a number, namely, new_in, is assigned, which will be sent to the mobile agent with node information that is stored in the node set. At the same time, the view state is updated.

**B. NODES LEAVING PROTOCOL SPECIFICATION**

According to Fig. 6 and Eq. (6), we use three processes in parallel to complete the tasks of the node leaving phase: RequestNode, a mobile agent (Ask_outAgent), and other grid nodes (OtherNode) in view $v$.

QuitGrid
\[
\text{def} \quad \text{RequestNode}|\text{Ask}_\text{outAgent}|\text{OtherNode} \quad (6)
\]

RequestNode($\overline{t}$, leave, view, $q$, $t$, $id$)
\[
\text{def} \quad (\text{CreateAgent}(
\text{Ask}_\text{out}, \text{inter} < \text{view}, q, t, id > \text{Ask}_\text{outAgent})|\text{leave} < \text{Ask}_\text{outAgent}, i >) \quad (7)
\]

Ask_outAgent
\[
\text{def} \quad \text{inter}(\text{view}, q, t, id).\text{nodeout} < \text{rqt}_\text{out} > .(\text{nodeout}(\text{Message}).([\text{Message} == \text{Wait}]
\text{CalculateCount}(\text{rqt}_\text{out})).(\text{true})\overline{agent} < \text{ack}_\text{out} > +\text{false}().\text{CalculateCount}(
\text{ack}_\text{out})) + [\text{Message} == \text{Confirm}]
\overline{agent} < \text{ack}_\text{out} > ).\text{AckOutProcess}() \quad (8)
\]

OtherNode($\text{nodeout}, \text{rqt}_\text{out}, \text{id}, q, \text{quit}, \text{Confirm}, \text{wait}$)
FIGURE 7. Process channel diagram of the ResourceShare phase.

\[
def \triangleq \text{nodeout}(rqt\_out).(\text{[id} == q]\text{nodeout} < \text{Confirm} > +[\text{id} == \text{view} - 1]\text{nodeout} < \text{Wait} >) \]
\[
\text{|nodeout}(\text{quit}).\text{UpdateViewState}(R^v)
\tag{9}
\]

Remark 2:

1) In the RequestNode process, two tasks are executed, as expressed in Eq. (7). RequestNode creates a mobile agent, namely, Ask\_outAgents, and sends basic information about the leaving node to it through the inner channel. RequestNode copies multiple Ask\_outAgents to migrate to OtherNode through the leave channel.

2) In the Ask\_outAgent process, after receiving the basic information of the leaving node, Ask\_outAgent migrates to OtherNode and sends the leaving message <rqt\_out> from the node-out channel. Then, it judges the identity of the leaving node according to the node where the agent is located and performs various operations. As expressed in Eq. (8), if the node that is leaving is primary, Ask\_outAgent waits to receive the <rqt\_out> messages that are sent by the agents that are located at OtherNode. If the number of <rqt\_out> messages meets the threshold, Ask\_outAgent sends an <ack\_out> message to the other agents through the agent channel. If other scenarios caused backups to leave, the <ack\_out> message is sent through the channel agent directly.

3) OtherNode completes the judgment of the identity of the leaving node. If primary failure causes primary to leave, the node returns a <wait> message to the corresponding agents. If other reasons cause the node to leave, a <confirm> message is returned to the corresponding agents. After receiving the <quit> flag, the update process of the view state is completed by OtherNode, as expressed in Eq. (9).

C. CONSISTENCY PROTOCOL SPECIFICATION

According to Fig. 7 and Eq. (10), we use three processes in parallel to complete the task of the consistency protocol. They are expressed in Eq. 17 in terms of the new joining node (NewJoinNode), the mobile agent (ResourceShareAgent), and other grid nodes (OtherNode) in the view v.

\[
\text{ResourceShare} \triangleq \text{NewJoinNode}\{\text{ResourceShareAgent} | \text{OtherNode} \}
\tag{10}
\]

\[
\text{NewJoinNode}(n) \triangleq \text{SendResourceProcess} | \text{Accept} - \text{ResourceProcess}
\tag{11}
\]

\[
\text{ResourceShareAgent}(i) \triangleq \text{inner}(\text{reply\_new\_id}, \text{ResourceInfo, view}),\text{trans} < \text{ResourceInfo} > .\text{trans}(\text{MA\_Digest, view, i}),(\text{[view} == i]\text{agent}),\text{Pre} - \text{Prepare}(i)
\tag{12}
\]

\[
\text{OtherNode}(o) \triangleq (\text{trans}(\text{ResourceInfo}), \text{CalculateHash})(\text{ResourceInfo}, \text{trans} < \text{MA\_Digest, view, i}>) \text{|(trans(updateResource), UpdateResource(ResourceInfo, viewTable)} | \text{migrate} \text{ResourceShareAgent})
\tag{13}
\]

Remark 3:

1) In the NewJoinNode process, two processes are executed in parallel, as expressed in Eq. (11). The former is SendResourceProcess, which creates several ResourceShareAgent instances and assigns its resource information to them. Then, these agents migrate to other nodes of the grid through the migration channel. The latter is AcceptResourceProcess, which receives the return resource information of the agent and checks the contents. If the contents are correct, the resource information table is updated and the agent is released.

2) ResourceShareAgent obtains the resource information of NewJoinNode after being created, migrates to OtherNode, and performs resource interaction through the trans channel.

3) OtherNode calculates the digest value according to the received resource information of NewJoinNode and returns the digest value to the agent. The local resource information table is updated according to the negotiation result, and the agent that is located at the node is sent to NewJoinNode to update its resource information, as expressed in Eq. (13).

D. VIEW CHANGE PROTOCOL SPECIFICATION

According to Fig. 8 and Eq. (14), we use three processes in parallel to complete the task of the view change protocol: the view change node (ViewChangeNode), the mobile agent (ViewChangeAgent), and the update view (NewView).

\[
\text{ViewChange} \triangleq \text{ViewChangeNode}\{\text{ViewChangeAgent} | \text{NewView} \}
\tag{14}
\]
H. Kang et al.: ADGM

V. SIMULATION

In this section, we first implement ADGM by using nPict language and then we set up experimental items by analyzing the characteristics and requirements of the grid systems. Moreover, to validate the AGCA, we compare its performance with some comparison algorithms.

A. ENVIRONMENT AND SETUPS

The CPU of the computer used for the experiments is i3-4150 and the RAM is 8 GB. We implement the operations of joining, leaving, searching and acquiring resources of the grid nodes by employing the nPict language [23], [24] on a Linux virtual machine with 628 MB of memory and 20 G of storage. Moreover, Raft [25], practical byzantine fault tolerance (PBFT) [26] and proof of work (PoW) [27] are introduced as the comparison algorithms, and each algorithm is run for 20 times. In each time, the numerical statistics results are recorded and presented.

As mentioned above, there is no unified standard for quantifying the performance of decentralized grid models. Thus, we discuss the performance indicators of the decentralized grid in this part. In the grid system, the response time refers to the elapsed time from the start of the execution to the termination of the execution, which can intuitively reflect the running time performance of the algorithm on the system [28]. The resource occupancy refers to the average memory usage when the program completes a task, which can better reflect the spatial performance of the algorithm on the grid system. The number of grid nodes has a substantial influence on the response time and the resource occupancy of each stage; hence, the number of grid nodes is regarded as an independent variable. For comparison, we initialize the system to four grid nodes and set the initial information for each node. Then, the number of grid nodes is continuously increased, and the related data of the response times and resource occupancies of the four algorithms in node joining, resource sharing, and node leaving are recorded.

Moreover, in practical applications, the grid member nodes may have Byzantine errors that are caused by malicious attacks [29], and nodes with Byzantine errors are called Byzantine nodes. Therefore, in the simulation, we select a grid system with 30 grid nodes, and we continuously increase the number of Byzantine nodes in the system, simulate the scenario of Byzantine errors in the grid in a practical application, and record the response times of the four algorithms. The fault tolerance of the algorithm is discussed based on the response time.

B. EXPERIMENTAL RESULTS AND DISCUSSION

1) RESPONSE TIME

The response time of nodes joining phase is plotted in Fig. 9. As can be seen, Raft, which sacrifices part of the fault tolerance and is relatively lightweight, always realizes superior time performance [30]. When the number of parallel nodes is small, the execution efficiencies of algorithms AGCA, PBFT, and PoW do not differ substantially. However, as
FIGURE 9. (a), (b), (c) show the response time changes of the algorithms in node joining phase, node leaving phase and resource sharing phase, respectively. (d) presents the resource occupation of different methods. (e) provides response time of different algorithms in resource sharing phase with Byzantine error. (f) compares the comprehensive ability of different methods.

the number of parallel nodes increases, the response times of PoW and PBFT increase significantly, and the trend is strong. The initialization of the agent results in less nodes and, hence, non-ideal time performance of AGCA. However, as the number of nodes increases, AGCA realizes satisfactory time performance.

Fig. 9b shows the response time of nodes leaving phase. It is observed that the response times of AGCA, PBFT, PoW and Raft increase significantly with the number of grid nodes. The response times of Raft and AGCA do not differ substantially; however, they have strong advantages over PoW and PBFT, and the advantages increase as the number of nodes increases. The addition of the agent enables AGCA to realize superior time performance in the node departure phase as well.

The response times in the resource sharing phase are compared in Fig. 9c. The response times of AGCA, PBFT, PoW and Raft increase significantly with the number of grid nodes. According to the rising trend, PBFT has the largest growth compared with the other three algorithms; hence, PBFT does not possess satisfactory stability. According to the response times, Raft and AGCA differ minimally in terms of execution efficiency, and they have large advantages over PoW and PBFT. The introduction of the agent and the light weight of Raft accelerate the resource sharing of the grid.

2) RESOURCE OCCUPANCY
As shown in Fig. 9d, as a lightweight protocol, Raft has lower resource occupation, and PoW’s proof mechanism causes it to have higher resource occupation than the other three. When the number of grid nodes is low, the application of agent in AGCA causes it to occupy more resources. However, as the number of grid nodes increases, the amounts of resources that are occupied by AGCA and PBFT do not differ substantially. AGCA realizes superior time performance because it does not occupy many resources.

3) FAULT TOLERANCE
In the resource sharing phase, the number of Byzantine nodes is continuously increased and the response time is recorded. As plotted in Fig. 9e, the response times of algorithms AGCA, PBFT and PoW increase minimally with the number of Byzantine nodes, while Raft shows a very strong upward trend. AGCA, PBFT and PoW can resolve the Byzantine error interference and have robust fault tolerance; hence, they are suitable for grid environments that are prone to malicious access.

4) COMPREHENSIVE ANALYSIS
To the best of our knowledge, no studies have been conducted on a comprehensive quantitative standard for decentralized grid models. Therefore, we normalize the performance of these algorithms to obtain their performance scores in terms of time performance, space performance, and fault tolerance by using max-min normalization [31]. Moreover, we employ the linear weighting method to obtain the comprehensive score of these algorithms, in which the weights of each term are equal. As shown in Fig. 9f and Table 1, the comprehensive performance of PoW is low, and its work proof mechanism consumes a substantial amount of resources; hence, it is not suitable for grid environments. The Raft algorithm has
excellent time performance and space performance, but it cannot solve Byzantine errors; hence, it is not suitable for grid environments with fault-tolerance requirements. PBFT has relatively balanced time performance, space performance, and fault tolerance, but it has lower performance compared to AGCA in most terms. AGCA has a higher comprehensive performance and can overcome the interference problem of Byzantine nodes.

In summary, for grid systems with the requirements of fault tolerance and short response time, the AGCA can better decentralize the grid.

VI. CONCLUSION

The decentralization of grid systems plays an important role in increasing the efficiency of joining, leaving, searching and acquiring resources of the grid nodes and in enhancing the fault tolerance of grid systems. Aiming at overcoming the problems that are encountered due to the decentralized model of the grid, such as low efficiency, low decentralization and poor fault tolerance, a decentralized grid model, namely, ADGM, with generality and functional integrity is proposed. First, the view is defined and explained, and the basic structure of the ADGM grid model and the behavior of the agent are discussed. Then, the principle and internal structure of the AGCA are described. The principle of action and the topologies of the group member protocol, consistency protocol, and view change protocol are described in detail. Furthermore, based on Pi calculus, the three protocols are formally specified. Finally, the ADGM is implemented using nPict language. In the experiments, AGCA, PBFT, Raft, and PoW are compared in terms of response time, resource occupancy, fault tolerance, and comprehensive performance. The results demonstrate that AGCA realizes higher comprehensive performance and more symmetrical time performance, space performance and fault tolerance than algorithms such as PBFT, Raft and PoW via effective decentralization of the grid environment.

The ADGM model guarantees that there is no content loss or malicious tampering in the process of the message digest calculation. However, it must be strengthened in terms of security. In future work, we will add encryption algorithms to the model to render it more secure.

REFERENCES

[1] X. Xu, C. Zhu, Q. Wang, X. Zhu, and Y. Zhou, “Identifying vital nodes in complex networks by adjacency information entropy,” Sci. Rep., vol. 10, no. 1, Feb. 2020, Art. no. 2691.

[2] I. Foster, “The anatomy of the grid: Enabling scalable virtual organizations,” in Proc. 1st IEEE/ACM Int. Symp. Cluster Comput. Grid, 2001, pp. 6–7.

[3] M. Chauvin, G. Mathieu, S. Camarasu-Pop, A. Bonnet, M. Bardies, and I. Perseil, “Enabling large scale data production for OpenDose with GATE on the EGI infrastructure,” in Proc. 19th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput. (CCGRID), May 2019, pp. 658–665.

[4] S. Camarasu-Pop, T. Glatard, and H. Benoit-Cattin, “Simulating application workflows and services deployed on the European grid infrastructure,” in Proc. 13th IEEE/ACM Int. Symp. Cluster Cloud Comput., May 2013, pp. 18–25.

[5] Q. Ju, G. Sun, H. Li, and Y. Zhang, “Collaborative in-network processing for Internet of battery-less Things,” IEEE Internet Things J., vol. 6, no. 3, pp. 5184–5195, Jun. 2019.

[6] A. A. Monrat, O. Schelén, and K. Andersson, “A survey of blockchain from the perspectives of applications, challenges, and opportunities,” IEEE Access, vol. 7, pp. 117134–117151, 2019.

[7] L. Kang and D. C. Parkes, “A decentralized auction framework to promote efficient resource allocation in open computational grids,” in Joint Workshop on The Economics of Networked Systems and Incentive-Based Computing, New York, NY, USA: Association for Computing Machinery, 2007.

[8] X. Wang and L.-F. Kong, “Resource clustering based decentralized resource discovery scheme in computing grid,” in Proc. Int. Conf. Mach. Learn. Cybern., 2007, pp. 3859–3863.

[9] H. Abbes and T. Louati, “PastryGridCP: A decentralized rollback-recovery protocol for desktop grid systems,” in Algorithms Architectures for Parallel Processing, vol. 8285. Cham, Switzerland: Springer, 2013, pp. 143–152.

[10] A. K. Shaikh, S. M. Alhashmi, and R. Parthiban, “A semantic impact in decentralized resource discovery mechanism for grid computing environments,” in Proc. Int. Conf. Algorithms Archit. Parallel Process. Berlin, Germany: Springer, 2012, pp. 206–216.

[11] J. Wu and X. Xu, “Decentralised grid scheduling approach based on multi-agent reinforcement learning and gossip mechanism,” CAAI Trans. Intell. Technol., vol. 3, no. 1, p. 21, Apr. 2020.

[12] F. Pop, “Communication model for decentralized meta-scheduler in grid environments,” in Proc. Int. Conf. Complex. Intell. Softw. Intensive Syst., 2008, pp. 315–320.

[13] J. Lee, S. Choi, T. Suh, and H. Yu, “Mobility-aware balanced scheduling algorithm in mobile grid based on mobile agent,” Knowl. Eng. Rev., vol. 29, no. 4, pp. 409–432, Sep. 2014.

[14] J. Qin, Q. Ma, Y. Shi, and L. Wang, “Recent advances in consensus of multi-agent systems: A brief survey,” IEEE Trans. Ind. Electron., vol. 64, no. 6, pp. 4972–4983, Jun. 2017.

[15] J. J. Chung, D. Miklíd, L. Sabattini, K. Tumer, and R. Siegwart, “The impact of agent definitions and interactions on multiagent learning for coordination in traffic management domains,” Auto. Agents Multi-Agent Syst., vol. 34, no. 1, p. 21, Apr. 2020.

[16] O. Amir, F. Doshi-Velez, and D. Sarna, “Summarizing agent strategies,” Auto. Agents Multi-Agent Syst., vol. 33, no. 5, pp. 628–644, Sep. 2019.

[17] M. A. Mahmoud, M. S. Ahmed, and S. A. Mostafa, “Norm-based behavior regulating technique for multi-agent in complex adaptive systems,” IEEE Access, vol. 7, pp. 126662–126678, 2019.

[18] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, “Enhancing bitcoin security and performance with strong consistency via collective signing,” in Proc. 25th USENIX Secur. Symp., 2016, pp. 279–296.

[19] R. Milner, J. Parrow, and D. Walker, “A calculus of mobile processes. I,” Inf. Comput., vol. 100, no. 1, pp. 1–40, Sep. 1992.

[20] R. Milner, Communicating Mobile System-Pi-Calculus. Cambridge, U.K.: Cambridge Univ. Press, 1999.

[21] M. Abadi, B. Blanchet, and C. Fournet, “The applied pi calculus: Mobile values, new names, and secure communication,” J. ACM, vol. 65, no. 1, pp. 1–41, 2017.

[22] S. Sakulin, A. Alfimtsev, E. Tipsin, V. Devyatkov, and D. Sokolov, “User interface distribution method based on pi-calculus,” Int. J. Distrib. Syst. Technol., vol. 3, no. 3, pp. 1–20, Jul. 2019.

[23] B. C. Pierce and D. N. Turner, “Pict: A programming language based on the pi-calculus,” in Proof, Language, and Interaction, Essays in Honour of Robin Milner. Cambridge, MA, USA: MIT Press, 2000, pp. 455–494.

[24] P. Sewell, P. T. Wojciechowski, and A. Unyapoth, “Nomadic pict: Programming languages, communication infrastructure overlays, and semantics for mobile computation,” ACM Trans. Program. Lang. Syst., vol. 32, no. 4, pp. 1–63, 2010.

[25] D. Ongaro and J. Ousterhout, “Search of an understandable consensus algorithm,” in Proc. USENIX Annu. Tech. Conf., 2014, pp. 305–319.
[26] Y. Wang, S. Cai, C. Lin, Z. Chen, T. Wang, Z. Gao, and C. Zhou, “Study of blockchains’s consensus mechanism based on credit,” IEEE Access, vol. 7, pp. 10224–10231, 2019.

[27] H. Cho, “ASIC-resistance of multi-hash proof-of-work mechanisms for blockchain consensus protocols,” IEEE Access, vol. 6, pp. 66210–66222, 2018.

[28] Q. Ju, G. Sun, H. Li, and Y. Zhang, “Latency-aware in-network computing for Internet of battery-less things,” in Proc. IEEE 88th Veh. Technol. Conf. (VTC-Fall), Aug. 2018, pp. 1–5.

[29] S. Chinnathamby, A. Santhanam, J. Rajarathinam, and M. Senthilkumar, “Scheduling and checkpointing optimization algorithm for Byzantine fault tolerance in cloud clusters,” Cluster Comput., vol. 22, no. 6, pp. 14637–14650, 2019.

[30] D. Huang, X. Ma, and S. Zhang, “Performance analysis of the raft consensus algorithm for private blockchains,” IEEE Trans. Syst., Man, Cybern. Syst., vol. 50, no. 1, pp. 172–181, Jan. 2020.

[31] L. Munkhdalai, T. Munkhdalai, K. H. Park, H. G. Lee, M. Li, and K. H. Ryu, “Mixture of activation functions with extended min-max normalization for forex market prediction,” IEEE Access, vol. 7, pp. 183680–183691, 2019.

HUI KANG received the M.E. and Ph.D. degrees from Jilin University, in 1996 and 2007, respectively. She is currently an Associate Professor with the College of Computer Science and Technology, Jilin University. Her research interests include grid computing, information integration, and distributed computing.

JIAHUI LI received the bachelor’s degree in software engineering from Jilin University, Changchun, China, in 2018, where he is currently pursuing the master’s degree in computer science and technology. His current research interests include edge computing and microservice.

LIN ZHANG received the bachelor’s degree in electrical engineering and automation from the Changchun University of Technology, in 2013, and the master’s degree in computer science and technology from Jilin University, in 2016. He is currently an Assistant Engineer with Financial Department, Jilin University. His research interest includes wireless sensor networks.

TIE FENG received the M.S. and Ph.D. degrees from Jilin University, in 1994 and 2007, respectively. He is currently an Associate Professor with the College of Computer Science and Technology, Jilin University. His research interests include software maintenance and evolution, reverse engineering, software refactoring, and automated software engineering.

QIUWEN YIN received the bachelor’s degree in computer science and technology and the master’s degree in computer application technology from Jilin University, Changchun, China, in 2015 and 2018, respectively. Her research interests include mobile communication and network and Pi calculus.

YUJUN LIU received the bachelor’s degree in software engineering from Jilin University, Changchun, China, in 2018, where she is currently pursuing the master’s degree in computer science and technology. Her current research interests include mobile crowdsensing and Lyapunov optimization.

* * *