A Negotiation Protocol for Fine-Grained Accountable Resource Provisioning and Sharing in e-Science

Zeqian Meng · John Brooke · Junyi Han ·
Rizos Sakellariou

Received: 20 December 2018 / Accepted: 21 February 2020 / Published online: 20 May 2020
© The Author(s) 2020

Abstract With the increasing demand for dynamic and customised resource provisioning for computational experiments in e-Science, solutions are required to mediate different participants' varied demands for such resource provision. This paper presents a novel negotiation protocol based on a new collaboration model. The protocol allows e-Scientists, the manager of an e-Scientist's collaboration, and resource providers to reach resource provisioning agreements. By considering the manager of an e-Scientist collaboration for negotiation decisions, the protocol enables fine-grained accountable resource provision on a per job basis for e-Scientist collaborations, without binding the e-Scientist collaboration to resource providers. A testbed built with the protocol is also presented, making use of a production e-Science gateway, use cases, and infrastructures. The testbed is experimentally evaluated, via designed scenarios and comparison with existing production tools. It demonstrates that the proposed negotiation protocol can facilitate accountable resource provision per job, based on resource sharing rules defined and managed by e-Scientist collaborations.

Keywords Negotiation protocol · e-Science · Resource sharing · Fine-grained accounting

1 Introduction

E-Science is a collaborative, computationally- or data-intensive research activity across all disciplines, throughout the research lifecycle, facilitated by infrastructures [1]. As a collaborative activity, e-Science inherently enables e-Scientists, often from different universities, institutions, or companies to share resources, data, and expertise. This sharing activity requires e-Scientist collaborations, which can be in the form of research groups. A research group requires a group manager to manage distributed resource sharing among distributed members [2], while resources can be provided by different infrastructures, e.g., Clouds, Grids, local Clusters. In such e-Science collaborations, resource management for computational experiments involves: (i) resource sharing management for members in a research group; and (ii) resource provisioning management for infrastructures as resource providers.

To satisfy the demands of dynamic and customised resource provisioning, Cloud computing has been increasingly employed by e-Science computational experiments. Open markets, such as the Helix Nebula Science Cloud (HNSciCloud) [3] and EGI Marketplace [4], have been proposed and established to

Z. Meng (✉) · J. Han · R. Sakellariou
University of Manchester, Manchester, UK
e-mail: zeqian.meng.academic@gmail.com

J. Brooke
Freelance Consultant and Researcher, Manchester, UK

harness the power of Cloud platforms and offer high-performance computing for scientific experiments. Meanwhile, Clouds, Grids, and local Clusters have all been explored for collaborative resource provisioning to maximise throughput [5–7]. The on-going standardisation will accelerate this trend but, at the same time, it will also increase the separation distance between research groups and resource provisioning infrastructures. The reason is that standardisation facilitates dynamic resource provisioning via open standards, without binding a research group with resource providers. This evolution can be observed from the changes in the definition of what a Virtual Organisation (VO) is. In the early days of the Grid, to facilitate e-Science collaborations, Foster et al. regarded a VO as a collaboration of resources and users [8]. Nearly fifteen years later, in the EGI Marketplace, the concept of a VO has been interpreted as a collaboration of e-Scientists while resource providers are regarded as more independent entities from a VO [9].

This evolution requires solutions to searching for and mediating resource provisioning from different resource providers according to e-Scientists' specific demands while considering resource sharing management for a research group at the same time. The latter scenario needs to be examined as a research group may pay for the resources consumed by its group members and have a limited budget for computational resources which it needs to spend effectively. A mistake in a submitted job, for instance, an infinite loop, will lead to essentially infinite execution and may cost a significant amount of money [10]. Mechanisms to manage the complete resource provisioning lifecycle between independent research groups and resource providers are also required. To address such requirements, not really met by existing e-Science tools, this paper proposes an e-Science collaboration model and a negotiation protocol based on the model. The collaboration model, named the Alliance2 model, distinguishes a research group from resource providers and shifts resource sharing management to a research group. This shift makes fine-grained accountable resource sharing feasible, while resources can be supplied from any infrastructures, including Clouds, Grids, and local Clusters, via negotiation. The negotiation is enabled by the Alliance2 protocol. The proposed model and protocol take advantage

of the increasing standardization of production infrastructures as studied in [11], making negotiable and accountable resource provisioning achievable.

To evaluate the functionalities contributed by the Alliance2 protocol a testbed has been built. This testbed: (i) extends the Application Hosting Environment 3 [12] for negotiable and fine-grained accountable resource provisioning; and (ii) utilises Amazon Web Services (AWS) and a local Cluster managed by the University of Manchester as resource provisioning infrastructures for two use cases. The performance of the developed automatic negotiation procedures has also been measured and evaluated.

In summary, the main contributions of this paper are:

1. A negotiation protocol based on a novel e-Science collaboration model. The paper provides a detailed introduction to the proposed model and protocol.
2. A prototype implementation of the proposed negotiation protocol, contributing to a novel broker. The implementation demonstrates the integration of the protocol with production e-Science gateway, infrastructures, and use cases. It also illustrates that the protocol is interoperable with existing infrastructures.
3. A set of experiments to evaluate the functionalities and performance of the protocol for negotiable resource provision for e-Scientists and fine-grained accountable resource sharing for research groups.

The rest of this paper is organised as follows. Section 2 discusses other protocols for resource provisioning for e-Science computational experiments. Section 3 presents the Alliance2 model, which demonstrates the relationships and demands of the involved negotiating entities. Section 4 illustrates a high-level state machine, which can manage a complete resource provisioning lifecycle covered by the Alliance2 protocol. The protocol is described in Section 5, which also presents a detailed analysis and comparison between the Alliance2 protocol and the related work discussed in Section 2. The testbed shown in Section 6 illustrates an implementation and evaluation of the protocol. Finally, Section 7 summarises the main contributions of the paper.

2 Related Work

Negotiation has been considered important for dynamic and SLA-based resource provisioning in e-Science [13, 14]. The semantic framework developed in [13] is capable of selecting negotiation protocols and satisfying resources to form agreements between users/agents and providers. The work in [14] presents an automatic system for SLA-based Cloud service provisioning, targeting self-adaptive SLA attainment for loosely coupled Cloud infrastructures. However, the negotiation procedures, namely how negotiating entities negotiate and reach an agreement, are not in the scope of the work in [13, 14].

One of the early protocols to enable negotiable resource provisioning for application execution is the Negotiation and Acquisition Protocol (SNAP) [15], which manages and composes Service Level Agreements (SLAs) dynamically and asymmetrically. SNAP enables a client to negotiate and re-negotiate an SLA with a resource provider, while the resource provider responds according to the received requests. The protocol developed in [16] enables service selection based on an application's QoS criteria specified by a user, by querying available services. The specification presented in [16] illustrates how to coordinate communication between different functional components to select resources for an application's execution, including negotiation with a user. However, it does not give detailed specifications on how the negotiation proceeds. WS-Agreement Negotiation is presented as a symmetric protocol in [17] even though asymmetric implementation is possible. WS-Agreement Negotiation supports distributed web service provisioning by alternating offers and counter-offers between a service provider and a service consumer. In order to form a contract or terminate a negotiation, WS-Agreement Negotiation needs to be combined with WS-Agreement [18] or WS-Disagreement [19], respectively. WS-Agreement Negotiation considers re-negotiation to alter an existing agreement, which however is allowed in the Agreement layer and not specified in detail in the specification. The EAlternating offer protocol in [20] provides mechanisms to enable users to choose the best proposals from trading partners via two-phase asymmetric negotiation. The asymmetric negotiation, which allows a requester to make a final decision for negotiation, makes it

impossible to interoperate with some existing infrastructures, as they do not allow a requester to make a decision for resource provisioning. In [21], a solution to conduct payment for resource provisioning after successful resource usage negotiation is proposed. However, this solution cannot be applied for pay-as-you-go resource provision, namely Cloud services, or jobs enabling interaction during runtime that may change resource demands. The reason is that, in such cases, the number of resources to be used cannot be known when negotiating.

It is noted that protocols in [15–17, 20, 21] are based on Grids, and the negotiation between a requester and a provider is restricted. They assumed that: (i) a research group has reached a resource provisioning collaboration with a provider; and (ii) authorization and accounting are conducted by the provider. These assumptions are not true anymore for dynamic resource provisioning collaborations between independent research groups and resource providers and are not valid for an open market either. For these two scenarios, resource providers are not concerned about or want to be burdened with a research group's internal resource sharing management. For instance, a group can define different roles with various resource sharing rules within the group, while a provider would prefer to provision and manage resources according to roles, rather than individuals. Also, these assumptions indicate that previously unused resources cannot be negotiated dynamically, as collaborations between research groups and providers need to be established beforehand. This resource management lifecycle of Grids has been criticised by Demchenko et al. [22], who propose that the existing resource management lifecycle should be changed for dynamic resource provisioning requirements.

In summary, the protocols proposed by [15–17, 20, 21] in isolation are not suitable for the scenario considered in this paper: dynamic and fine-grained accountable resource provisioning without binding research groups and resource providers. Fine-grained accountable resource provisioning indicates that a research group is aware of the number of resources to be provided and actually consumed by a job submitted by a group member. Moreover, none of the protocols discussed above applies formal verification to validate the protocols' properties. Formal verification is critical for large-scale distributed systems, as it enables a

protocol to be checked for desired properties before implementation. For instance, the correctness properties of communication using the nonblocking primitives of the Message Passing Interface (MPI) have been verified by formal verification [23].

3 A Collaboration Model: Alliance2 Model

Based on the different entities' varied requirements and responsibilities in e-Science collaborations, we extend the Alliance model [24], which is composed of a *Resource Requester* and a *Resource Provider*, with a *Resource Manager*. The Resource Manager and Resource Requesters are in the same administration domain, which is independent from the domain of a Resource Provider or a collaboration of Resource Providers. The extended model is the Alliance2 model.

The three types of entities included by the Alliance2 model represent: (i) e-Scientists; (ii) a group manager for the e-Scientist collaboration; and (iii) resource provisioning infrastructures. They are considered the main participants in e-Science resource provisioning for computational experiments.

Resource Requesters may come from different organisations or institutions but have interests in the same research area. They share and take advantage of large numbers of data, expertise, knowledge, computing and storage resources. Such distributed and collaborative sharing should be managed by certain rules, such as e-Scientists' access priorities and balances to the shared resources. These rules should be defined and managed by a Resource Manager [3].

The introduction of the Resource Manager makes the membership and resource sharing management of a research group independent from resource provisioning infrastructures as Resource Providers. This independence enables shifting fine-grained accounting and resource sharing management to a research group, which can be facilitated by the Resource Manager of the group. Clearly, such independence and shift require an evolutionary change to the existing e-Science resource sharing management lifecycle, as shown in Fig. 1. In the new lifecycle, resource provisioning requests from members of a group should be authorised by a group manager, while the group manager can apply limitations for resource usage required by members, according to resource sharing

Fig. 1 e-Science resource management lifecycle evolution introduced by Alliance2 model

rules within the group. A group manager will be in total control of authorisation, resource sharing management, and accounting of members of the group. This mechanism contributes to fine-grained management, which is achievable in the Alliance2 model as the number of members and policies for a research group is on a manageable scale, compared to the number that needs to be managed by an infrastructure. In this scenario, Resource Providers are only concerned with coarse-grained resource provisioning management. For example, role-based management can be applied by providers for simplifying resource provisioning management [25] by reducing the mapping complexity between Resource Requesters' identities and a Resource Provider's local access management. In this way, a Resource Manager can manage local membership and resource sharing policies with fine-granularity, while a Resource Provider allocates resources according to a requester's global role.

The communication between a Resource Manager and a Resource Provider regarding access control simplifies payment procedures, as the Resource Manager will be the entity to pay for resources consumed by its group members. In addition, distinguishing a research group from resource providers helps make Alliance2 interoperable with existing infrastructures, including Clouds, Grids, and Clusters. This interoperability can be achieved by regarding existing infrastructures as

Fig. 2 The high-level state machine for each participant for complete resource provisioning lifecycle

independent resource providers and taking advantage of existing functionality, as discussed in [24]. Following this principle, the testbed presented in Section 6 shows a solution to enable interoperation.

4 Alliance2 High-Level State Machine

A high-level state machine has been designed to capture the lifecycle that the Alliance2 protocol covers. The high-level state machine of each entity in the Alliance2 protocol is shown in Fig. 2. Two high-level sessions are designed: a negotiation session and a termination session. A negotiation session ends with one of the two final states: *contracted* and *uncontracted*. A *contracted* state indicates that the negotiation is successful and a contract is formed, while an *uncontracted* state means that no agreement is reached among the negotiating entities. To complete the negotiation procedures, a state *negotiating* is introduced to represent the situation where a valid negotiation has been initiated, but no agreement has been reached.

In a business setting, terminating a contract before normal completion may allow the innocent party to claim a monetary penalty, while to end negotiation before contract formation has no such effects. Based on this, we distinguish different final states for contract termination from negotiation termination. The (contract) termination session captures the outcome of formed contracts. In the normal completion of the resource provisioning lifecycle, the state *completed* indicates that a job has been executed successfully under the agreed conditions in a contract. In addition, three states are introduced to represent the cases where an entity initiates the termination of a contract before job completion: *proTerminated* (contract termination initiated by the Resource Provider), *reqTermi-*

nated (contract termination initiated by the Resource Requester or the Resource Manager), and *terminated* (contract termination initiated by the Resource Provider and the Resource Requester/Resource Manager at the same time). Note that as resource sharing management for a research group is independent from resource provisioning management in providers' domains, the termination session enables a research group to track the resource usage of each job.

Furthermore, termination can also be caused by successful re-negotiation. Successful re-negotiation will update the previous contract to a *terminated* state, meaning that the negotiating entities agree with the termination of the previous contract. Figure 2 shows the states for both the negotiation session and the termination session. This state machine can be used for re-negotiation.

5 A Negotiation Protocol: Alliance2 Protocol

The Alliance2 protocol is extended from [24], which was designed upon UK contract law and the European Union Electronic Commerce Directive. The protocol proposed by [24] allows forming and terminating legally-binding contracts for dynamic e-Science collaborations between independent research groups and resource providers. The Alliance2 protocol inherits the law-based negotiation features from the protocol in [24] to form valid contracts. This approach is named contract-oriented negotiation in this paper. Contract-oriented negotiation contributes to the effectiveness of negotiation results, enforcing independent entities to fulfil the conditions agreed. The key difference between the Alliance2 protocol to the protocol in [24] is the introduction of the Resource Manager. The Resource Manager contributes to: (i)

enabling communication between a Resource Provider and a Resource Manager for access control decisions before contract formation; (ii) notification of negotiation results and job execution status from a Resource Provider to a Resource Manager to manage a complete resource provisioning lifecycle; and (iii) negotiation termination and contract termination initiated by a Resource Manager. These contributions are enabled by the negotiation procedures designed in the Alliance2 protocol, as shown in Fig. 3. Based on the Alliance2 model, the independence between research groups and resource providers allows dynamic collaborations between research groups/e-Scientists and resource providers, according to different resource provision features. Furthermore, race conditions, which are not addressed in [24], will also be discussed in this protocol.

5.1 Terminology

The three entities defined in the Alliance2 protocol correspond to those designed in the Alliance2 model: Resource Requester, Resource Manager, and Resource Provider. The *lifecycle* of resource provisioning considers not only negotiation for resource usage between a Resource Requester and a Resource Provider but also access control, negotiation result, and job execution status notification between a Resource Manager and a Resource Provider.

Negotiation is a way to resolve differences and reach an agreement among negotiation entities, usually with multiple rounds of communication to reach an agreement. *Re-negotiation* is the process where an entity of a formed contract wants to change conditions in that contract. Successful re-negotiation will terminate the re-negotiated contract and lead to the formation of a new contract.

Our work is consistent with the principles defined in [26] that a *contract* for an e-Science collaboration contains both technical terms and non-technical terms. The technical terms consider the practical effectiveness of the hardware and software infrastructures that are being created to enable collaborations in e-Science. The non-technical terms may include information for intellectual property and competition policy. This paper only discusses technical terms with specific application execution features for e-Scientists' application execution demands. We also suppose that a contract can be formed by combining the dynamic technical details with static non-technical conditions. This can be applied to negotiation that is carried out for resource provisioning requests under collaborations already formed between a research group and resource providers.

5.2 Assumptions and Rules

The following assumptions and rules identify the boundaries of the designed protocol:

1. It defines negotiation entities involved, their messaging behaviours, and message types.
2. It includes a negotiation protocol for contract formation and a contract termination protocol.
3. It identifies race conditions that each entity may encounter during negotiation, with corresponding agreed outcomes.

In addition:

1. Notification messages for job termination/completion will be handled by the proposed contract termination protocol. Other notification messages during job execution will not be discussed, such as messages to inform a requester

Fig. 3 Negotiation enabled by Alliance2 protocol

that the specified input file has been transferred or deleted. Such functions are infrastructure- or application-specific.

2. The protocol does not include mechanisms to deal with concurrent communication, where an entity has to deal with multiple messages from different sources at the same time.
3. There is no mechanism to deal with multi-peer consensus, where a group of users want to reach an agreement.
4. The negotiation protocol concentrates on negotiation procedures and message types. It does not consider law-related contract contents. However, contract templates for e-Science collaborations are available. For instance, the FitSM templates used by EGI pay-by-use experiment for IT service provision management can be applied [27].

5.3 The Protocol

The negotiation messages are grouped by different phases of negotiation: pre-negotiation, negotiation, and (contract) termination. The negotiation phase is subdivided according to functionality: resource negotiation, access negotiation, and revocation. The different negotiation phases and the corresponding messages are presented in Table 1.

Considering that different negotiating entities have varied requirements and responsibilities, the Alliance2 protocol is designed as an asymmetric protocol. In the Alliance2 protocol, an *Offer* message can only be sent from a Resource Requester to a Resource Provider, and the decision to accept or reject the *Offer* is made by the Resource Provider. This mechanism makes the protocol compatible with existing infrastructures, which enable only providers to make resource provisioning decisions. It also allows resource providers to provide resources according to the availability of local resources. Meanwhile, *QuoteRequest* messages and *Quote* messages are designed for Resource Requesters and Resource Providers, respectively, to express the intention for collaborations. They are not legally bound to final contracts. If a Resource Requester wants to change offer contents before forming a contract, revocation can be activated by sending a revoke request (*RevokeReq*). When a Resource Provider wants to alter the contents of an offer, a *Reject* message can be replied with the conditions that the provider prefers. *AcceptAck* is a message

designed to meet the legislative requirement: the acceptance must be communicated to the offeree, namely the Resource Requester in this protocol. An *AcceptAck* message informs the Resource Provider that the acceptance has been communicated to the Resource Requester.

Access negotiation allows a Resource Manager to control and track resource allocation. Combined with the high-level state machine in Fig. 2, a positive access negotiation result will keep the negotiation in a *negotiating* state. Restrictions on this resource provision can be contained in an *AccessSucceed* message, such as the maximum number of resources or the amount of money that can be consumed by this job. The restrictions can be specific for each member of the group, which should be defined and managed by the group manager. Along with contract termination and completion notification, access negotiation contributes to managing the complete resource provisioning lifecycle and fine-grained accountable resource provisioning. A negative access negotiation result will terminate the negotiation, leading to an *uncontracted* state. A negative access negotiation result indicates that: (i) the requester is not allowed to access the required resource(s) according to his/her priority in the group; or (ii) the requester does not have sufficient balance to run the job. Both situations stop the negotiation from proceeding further, reaching an end state of *uncontracted*.

As discussed in the protocol proposed by Parkin [24], access negotiation is regarded as a stateless simple request-response messaging model. This indicates that access negotiation can happen any time during negotiation. The Alliance2 model demonstrates that dynamic and accountable resource provisioning requires an access decision for a resource provisioning decision. It indicates that access negotiation should happen before a contract is formed. Based on this, access negotiation can happen in two possible scenarios in the Alliance2 protocol: during the resource negotiation phase or the pre-negotiation phase. The decision for implementation should depend on demands in practice. For instance, if an access decision depends on complex policies, this indicates that to make an access decision will probably take longer than the time consumed by transferring messages for the access negotiation procedures. In this scenario, access negotiation that is activated during resource negotiation is preferable, that is, after

Table 1 Negotiation messages and features

Negotiation phase	Negotiation sub-phase	Message	Sender	Receiver	Features
Pre-negotiation	Resource Negotiation	QuoteRequest	Requester	Provider	Both messages are without legal effects.
		Quote	Provider	Requester	This message is without legal effects.
		Offer	Requester	Provider	
		OfferAck	Provider	Requester	
		Accept	Provider	Requester	This can only happen after the Requester's access privilege has been approved by a corresponding Manager.
		AcceptAck	Requester	Provider	It closes the negotiation.
		Reject	Provider	Requester	
		Terminate	Requester	Provider	It closes the negotiation. Provider may send a Terminate to inform Manager the termination.
			Manager	Provider	It closes the negotiation. Provider may send a Terminate to inform Requester the termination.
			Provider	Requester & Manager	It closes the negotiation. Provider may send a Terminate to inform Manager the termination.
Negotiation	Access Negotiation	Contracted	Provider	Manager	It closes the negotiation.
		Uncontracted	Provider	Manager	It closes the negotiation.
		AccessReq	Provider	Manager	
		Accessing	Provider	Requester	
		AccessSucceed	Manager	Provider	
	Revocation		Provider	Requester	
		AccessDenied	Manager	Provider	It closes the negotiation.
			Provider	Requester	It closes the negotiation.
		RevokeReq	Requester	Provider	
		RevokeAcc	Provider	Requester	
Contract Termination		RevokeRej	Provider	Requester	
		ReqTerminate	Requester	Provider	It closes the negotiation. Provider should send a ProTerminate to Manager, which should indicate that the termination is required by the Requester.
			Manager	Provider	It closes the negotiation. Provider should send a ProTerminate to Requester, which should indicate that the termination is required by the Manager.
		ProTerminate	Provider	Requester & Manager	It closes the negotiation.

the requester has selected an *Offer* from all *Quotes* for further negotiation. This is because the *Offer* has been preliminarily selected, which can avoid processing policies for a large number of resources for an

access decision. This can also increase the success rates of negotiation.

Any of the three entities can initiate termination during negotiation (that is, before contract formation)

via a *Terminate* message, ending the negotiation with an *uncontracted* state. Contract termination, as captured by the termination session in Fig. 2, can be initiated by a *ReqTerminate* and a *ProTerminate* message. A *ReqTerminate* can be sent by a Resource Requester or a Resource Manager, while a *ProTerminate* can be sent by a Resource Provider. A termination decision made by a Resource Requester/Resource Manager should be passed by a Resource Provider to the corresponding Resource Manager/Resource Requester. In this situation, a *ProTerminate* should contain information to specify that the termination is required by a Resource Manager/Resource Requester.

5.4 Race Conditions and Solutions

It is essential to discuss race conditions that may happen during negotiation and propose solutions accordingly so that it is guaranteed that the three negotiating entities reach the same final negotiation states. A race condition is a messaging situation where a Resource Requester and a Resource Provider, or a Resource Provider and a Resource Manager, send messages that cross each other on the network. As we already consider the case that a Resource Requester/Resource Manager and a Resource Provider send a message to terminate a contract at the same time, the discussion of race conditions will focus on the negotiation phase. The following summarises the principles applied when designing solutions for race conditions.

Before contract formation or negotiation termination, race conditions may occur because of the following three reasons: (i) a Resource Requester can send a *RevokeReq* message or a *Terminate* message during negotiation anytime; (ii) a Resource Provider can send a *Terminate* message during negotiation anytime; and (iii) a Resource Manager can send a *Terminate* message anytime after receiving an *AccessReq* message.

Any solution to address race conditions should be able to mediate between the messaging behaviours of negotiating entities to continue the current negotiation or to reach the same final state. To maintain the law-based features of the protocol, they should also avoid disputes over negotiation results. Correspondingly, the proposed solutions are: (i) when an entity receives/sends a termination request, the local negotiation state should be updated to *uncontracted*. This enables both negotiation entities to determine that a

race condition has happened when they receive further messages after sending/receiving a termination request and check with local negotiation states; (ii) after sending a *RevokeReq* message, if the message received is not a termination request, the Resource Requester should stay in a *negotiating* state and wait for further information from the corresponding Resource Provider.

5.5 Comparison with Related Work

The Alliance2 protocol inherits the law-based features from the protocol in [24], which considers all situations that may happen during contract-oriented negotiation: invitation to treat, advertisement, counter-offer, negotiation termination, and acceptance communication. To complete the resource provisioning lifecycle, the Alliance2 protocol also considers contract termination and re-negotiation, where contract termination can be initiated by a group manager. Furthermore, to enable a group manager to apply resource sharing rules in the group to resource provisioning decisions, access negotiation requires access decisions from the group manager before contract formation. The Alliance2 protocol distinguishes the negotiation between a requester and a provider from access negotiation between the provider and the corresponding manager. This asymmetric mechanism allows different security credentials to be applied to different negotiation procedures. For instance, negotiation between e-Scientists and providers can use usernames and passwords, while access negotiation between group managers and providers can require digital certificates. Also, as authorisation of resource provisioning requests is conducted during access negotiation, other authorisation solutions, e.g. proxy certificates, are not required for authorisation purposes. This not only enables lightweight credentials from e-Scientists permitting lightweight clients for e-Scientists, but also sustains the restrict security requirements from existing infrastructures contributing to interoperation. The acknowledgement for contract formation from requesters allows them to collect all required resources, working as a two-phase commit procedure, as discussed in [20, 21]. Table 2 summarises the features of the Alliance2 protocol and related protocols.

It is our view that the earlier work in [15, 16, 20] does not provide a detailed specification of the pro-

Table 2 Summary of negotiation protocols

	SNAP	G-QoS	WS-agreement negotiation	EAlternating offer protocol	HPC negotia- tion Protocol	Alliance2 protocol
Invitation to treat	—	—	✓	—	✓	✓
Advertisement	—	—	—	—	✓	✓
Access negotiation	✓	—	—	—	✓	✓*
Revocation	—	—	—	—	✓	✓
Negotiation termination	—	—	—	✓ (by provider)	✓ (by requester or provider)	✓ (by requester or provider or manager)
Counter offer	—	✓	✓	✓	✓	partly (only by requester)
Acceptance communication	—	—	—	—	✓	✓
Contract termination	✓ (by requester)	—	✓ (by requester or provider)	—	—	✓ (by requester or provider or manager)
Re-negotiation	✓	—	—	—	—	✓
Negotiation result notification	—	—	—	—	✓ (for payment)	✓*
Job termination notification	✓	—	—	—	—	✓*
Symmetric/ Asymmetric	asymmetric	symmetric	symmetric (can enable symmetric)	asymmetric	asymmetric	asymmetric
Co-allocation	—	—	—	✓	✓	✓

*These are additional features of the Alliance2 protocol, compared to the protocol proposed in [24].

posed protocols. A detailed specification should be able to clearly define the engaged entities' messages and messaging behaviours, ensuring consistent and valid negotiation states. This is especially important for contract-oriented negotiation between independent parties, to navigate negotiation between participants correctly and effectively.

In summary, compared to the discussed related work, the Alliance2 protocol considers all situations that may happen during contract-oriented negotiation. It manages complete resource provisioning lifecycle, from negotiation to contract satisfaction, between independent research groups and resource providers. It gives a detailed specification of the designed messages and messaging behaviours. The correctness of the Alliance2 protocol has been verified formally

by a formal model checker, Spin [28], by simulating the negotiation entities, messages, and messaging behaviours as designed in the protocol. The correctness here indicates that all negotiating entities can reach the same negotiation states via state exploration.

6 A Testbed for Experimental Evaluation

A testbed has been designed to demonstrate the properties of our proposal. The architecture of the testbed, including its main contribution, the Service Broker, is illustrated in Fig. 4. The Service Broker extends AHE3 with the capability of negotiation and fine-grained accounting. AHE3 is built upon the Software as a Service (SaaS) concept on the

Fig. 4 Testbed architecture: the modules within the extended AHE3 are the extensions; arrows marked with 1 and 2 show communication for resource provisioning in Use Case 1 and job execution in Use Case 2, respectively

top of infrastructure resources, focused on providing an easy-to-use gateway for e-Scientists in diverse application domains with high-performance resource supply. To achieve this, AHE3 is proposed to manage job submission and execution for e-Scientists to various infrastructures based on the demands for application execution. This indicates that AHE3 can be extended for the two use cases with the two different types of infrastructures. However, as accounting is not the focus of AHE3, resource consumption updates for a job submission or execution have not been enabled in AHE3. Also, resource sharing management for research groups has not been considered in AHE3. To enable negotiable and accountable resource provision for computational application execution, three main extensions have been built upon AHE3 for:

1. Negotiation for e-Scientists to search for satisfactory resources to conduct computational experiments in collaborated infrastructures.
2. Accountable resource provision on a per job level for fine-grained resource sharing management for research groups. This is achieved by cooperating with the resource management model and ontologies and programs developed in [29].
3. Job submission management for applications to be executed in Clouds and Clusters.

These extensions are shown in Fig. 4. The resulting software, which is a version of the AHE3 extended by the above three extra functions, is called **Service Broker**.¹ The Service Broker functions as a Resource Provider as specified in the Alliance2 protocol, to provide resource details and negotiate for resource provision on behalf of collaborated infrastructures. It comprises the functional components for negotiation

and resource sharing management. The main functions enabled are negotiation, user access control, resource matchmaking, accounting, platform credential management, job submission and execution management.

The two use cases implemented are: automatic data-driven computational steering deployed on AWS as Use Case 1; and resource sharing management in a local Cluster facilitated in the University of Manchester as Use Case 2, as shown in Fig. 4. Automatic data-driven computational steering has been proposed to solve the issues caused by runtime steering and real-time simulation monitoring [30]. It enables assigning more computing resources automatically, typically by increasing the number of CPUs, to shorten execution time and ensure finishing execution in time. Combined with the usage of AWS, the Service Broker conducts searching and negotiating for instance(s) with the required number of CPUs and provides the required information of the contracted instance(s) to the application during runtime in this use case. AWS allows on-demand resource provision, and computational steering allows e-Scientists (or agents) to take control of a program running. The combined use of the two introduces the possibility of infinite resource consumption caused by mistakes [10]. We empower the group manager to set a maximum cost/time for application execution for each user in this use case.

In Use Case 2, two types of jobs are supported in the university's local Cluster: serial jobs and parallel jobs.² As jobs in the Cluster are queue-based, users have no control over the exact time when the application execution will be activated and completed. This testbed allows e-Scientists to specify the deadline and the way (i.e., serial or parallel) for job execution in the local Cluster, to shorten the research lifecycle. In Use

¹The source code of the Service Broker is available at <https://github.com/ZeqianMeng/ServiceBroker>.

²For detailed information about the local Cluster: <http://ri.itservices.manchester.ac.uk/csf/>

Case 2, we also enable a group manager to define two different priorities for members to execute serial jobs and parallel jobs, respectively. All e-Scientists of the group can require applications to be executed serially by one CPU, while prioritised members can require a job to be executed in parallel with more than one CPU. This testbed also assumes that the research group has reached an agreement with the Cluster provider about the total number of CPU time that can be consumed by members for serial jobs and parallel jobs. In this way, the group manager can define fine-grained priorities for group members, while the Cluster provider only needs to be concerned with the total number of resources consumed by any member of the group for serial jobs or parallel jobs.

Designed to facilitate automatic negotiation for Use Case 1 to ensure the effectiveness of steering results, a Client Service³ has been developed to enable negotiation on behalf of e-Scientists. Meanwhile, the Service Broker stores and tracks negotiation and accounting related information, in addition to enabling job submission to the local Cluster in Use Case 2.

With the use cases, this testbed aims at verifying that the Alliance2 protocol is capable of:

- enabling e-Scientists to request a customised application's execution environment automatically during runtime in Use Case 1, namely to satisfy dynamic and customised resource provisioning demands via negotiation and re-negotiation.
- enabling the specification of a deadline for an application's execution, namely to meet e-Scientists' customised application execution demands in Use Case 2.
- facilitating accountable resource provisioning and fine-grained resource sharing per job for a research group for both use cases.
- being interoperable with existing infrastructures for resource provisioning via negotiation.

6.1 Implementation Assumptions

Negotiation-based resource provision is a different way to form and dissolve collaborations for resource supply, compared to the mechanisms applied by existing infrastructures. As a result, it is not feasible to enable the Alliance2 protocol in production environments.

³The source code of Client Service is available at <https://github.com/ZeqianMeng/ClientService>

Instead, the protocol can follow the brokering approach in [31] to enable negotiable resource provision from collaborated providers. This has been achieved by developing a broker, i.e. the Service Broker, to negotiate resource supply on behalf of infrastructures in this paper. Also, we extend an e-Science gateway, AHE3, to facilitate the negotiation procedures designed. In this way, our testbed can also demonstrate that the protocol can interoperate with existing infrastructures while meeting real demands of production use cases.

Apart from negotiation messages and messaging behaviours, other functions need to be taken into consideration for negotiation and negotiation-based resource management in practice, including negotiation decision-making strategies, resource allocation mechanisms, and concurrent communication management. The principal aim of this testbed is to verify the feasibility of the designed protocol to facilitate dynamic, customised, and accountable resource provisioning from different infrastructures. Focused on this aim, simple negotiation decision-making strategies, matchmaking strategies, and communication management are implemented in the Service Broker. In a real application, these mechanisms would vary according to e-Scientist, projects, and collaborations, etc.

Two interfaces are enabled for the Client Service so far, for negotiation and re-negotiation. For demonstration purposes only, pre-negotiation, resource negotiation, and access negotiation procedures have been enabled in the Client Service. Other negotiation functions can be developed and added if required.

The testbed focuses on negotiable contract contents for application execution or the demanded computing resource. The negotiable features for resource provisioning are tailored to the use cases enabled in this testbed. However, the features enabled (i.e., the required number of CPUs, the deadline required for application execution, the maximum cost/CPU time that can be consumed by a member in a group) can be used by other applications or use cases immediately. Also, other features can be enabled easily by using the developed ontologies and reasoning programs, as discussed in [29].

6.2 Service Broker

The Service Broker has been built using Java, with Restful web services for negotiation and accounting

The ontologies applied are based on a resource management model extended from the Grid Laboratory Uniform Environment 2.0 (GLUE 2.0) [32]. The developed ontologies and programs are deployed in the Service Broker currently, responsible for managing resource sharing policies in fine-granularity on behalf of a Resource Manager and managing coarse-grained resource provisioning policies for Resource Providers. However, as the functions for negotiation and accounting are independent, and ontologies are lightweight and can be accessed via a web link, separate software specifically for a Resource Manager and a Resource Provider can be enabled from existing functions if needed.

Combining the above resource management features with the negotiation capability enabled by the Alliance2 protocol, the Service Broker facilitates both: (i) dynamic and customised application execution demands for e-Scientists via negotiation; and (ii) fine-grained accountable resource provisioning and sharing for a research group. These two functions have not been enabled by AHE3 so far, and are shown in Fig. 5 as Negotiation and Accounting, respectively.

AHE3 was built as Restful web services, indicating that the communication can only be paired (a reply

The diagram illustrates the Amazon UoM computing cluster architecture, showing the flow of data and control between various components and external services.

Steps and Interactions:

- 1. Application Start/Stop/Steer Request:** This request is sent to the **Negotiation** module.
- 2. Startup/Shutdown:** This request is sent to the **AHE Runtime** module.
- 3. Get:** This request is sent to the **API** module.
- 4. Prepare:** This request is sent to the **API** module.
- 5. App-State created and workflow initiated:** This state is created and initiated within the **API** module.
- 6. Submit (Start job execution):** This request is sent to the **API** module.
- 7. As part of the workflow, the job is submitted to contracted infrastructures:** This step involves the **Connector Module**, **Storage Module**, **Security Module**, and **Extension Points**.
- 8. Data is sent to the designated location once completed:** This data is sent to the **Storage Module**.

Internal Components and Interactions:

- Negotiation** and **Accounting** modules interact with each other.
- Accounting** interacts with **Ontology files**.
- AHE Runtime** interacts with **App Registry** and **JBPM Workflow, Main Logic and API**.
- App Registry** and **JBPM Workflow, Main Logic and API** interact with the **App Instance** and **AHE Engine**.
- App Instance** and **AHE Engine** interact with the **Service Broker Database**.
- App Instance** and **AHE Engine** interact with the **Connector Module**.
- App Instance** and **AHE Engine** interact with the **Storage Module**.
- App Instance** and **AHE Engine** interact with the **Security Module**.
- App Instance** and **AHE Engine** interact with the **Extension Points**.
- App Instance** and **AHE Engine** interact with the **Service Broker Database**.
- App Instance** and **AHE Engine** interact with the **Service Broker Database**.
- App Instance** and **AHE Engine** interact with the **Service Broker Database**.

External Services and Data:

- Ontology files** are used for accounting and negotiation.
- Service Broker Database** is used for app state and workflow management.
- UoM computing Cluster** is used for job execution.
- amazon web services Collaborated Providers** are used for infrastructure and data storage.

corresponds to a request) [33]. Considering this, the two messages for acknowledgement (*OfferAck* and *Accessing*) were not enabled in this testbed. These two messages do not affect the validity of contract formation. Also, negotiation termination initiated by a group manager has not been implemented yet, because the two use cases enabled do not require manager termination during negotiation. Apart from that, other negotiation messages and messaging behaviours as presented in the protocol have been implemented and evaluated. More specifically, pre-negotiation, negotiation, revocation, negotiation termination initiated by resource requesters and providers, contract termination initiated by all three entities, and re-negotiation have been realised. The testbed implements access negotiation within the resource negotiation phase.

6.2.2 Job Management

AHE3 facilitates job execution management with a full package of functions to support job submission to infrastructures. These functions include file staging, application upload, and result fetching, which however are not available in the applied infrastructures. Instead, a job submission management workflow has been created and connected to the use cases. In this way, both, well-established job execution management procedures and simple job submission procedures, are available in the Service Broker and can be used according to different requirements from applications/projects. In addition to job submission management, job execution environment changes for dynamic steering demands, job completion notifications, and job termination requests are managed by the Service Broker with corresponding web service APIs.

6.2.3 Accounting Strategies

Based on the different measurement mechanisms applied by AWS and the local Cluster: (i) the service cost and CPU time are used to measure resource consumption in the testbed; and (ii) hours as units for AWS and seconds as units for the local Cluster are applied for duration measurement of application execution.

Accounting functionalities that have been enabled in the testbed are: (i) checking the cost or CPU time agreed between the group and providers with the requester's balance during access negotiation to ensure the throughput of negotiation, as the total cost

or the total number of CPU time that will be consumed cannot be known during job submission in both cases. In the testbed, this value has been set the same as the maximum value defined by the group manager for the requester per job; (ii) reducing the balances of the requester and the contracted resource(s) with the maximum value after successful negotiation. This is to avoid over-expenditure caused by follow-up jobs as e-Scientists may submit another application execution request before the current one is completed. When a job has been completed, the difference between the reduced amount and the actually consumed amount will be calculated and added back to the balances of both the requester and resource(s); (iii) the job execution can be managed by the maximum value defined by the group manager in Use Case 1 or by the deadline specified by an e-Scientist in Use Case 2 (note that when the Service Broker detects that the cost/CPU time consumed is approaching the maximum value/deadline, it will send a job termination request to the collaborated infrastructures); (iv) updating the balances of both the requester and resource(s) with the actually consumed amount when a job has been completed or terminated.

These accounting strategies can work with the constructed negotiation functionalities, contributing to accountable resource provisioning. The accounting properties facilitated by the Service Broker are based on the resource management model enabled in [29]. In addition, functions have also been developed to facilitate e-Scientists with balance queries and job status queries. These two functions are accessible via web service APIs requiring job IDs.

6.3 Evaluation and Results

Data-driven computational steering involves dynamic resource changes during runtime, while job execution in a local Cluster is typically queue-based. This means that job execution duration for both use cases is unpredictable during job submission. As a result, it is difficult to benchmark and evaluate the duration of the resource provisioning lifecycle for such dynamic resource provisioning use cases. Also, currently, no single accepted benchmark for large-scale scientific computing exists [34]. Furthermore, negotiation differs from existing approaches that enable resource provisioning in e-Science, making it very challenging to evaluate the full potential for performance

of a negotiation protocol in an existing infrastructure [21]. For these reasons, only the negotiation and the related accounting functions enabled as well as the performance of the developed automatic negotiation are evaluated. During the evaluation, the Service Broker and Client Service were deployed in two different instances on AWS, a t2.medium instance and a t2.micro instance, respectively.

6.3.1 Functionality Evaluation and Results

The purpose of the evaluation is to verify that the proposed protocol not only enables dynamic and customised resource supply via negotiation as expected but also enables access control and resource sharing management per job for a research group without a heavyweight management layer.

To combine the Alliance2 protocol with the two implemented use cases, the testbed has been designed to enable e-Scientists in a research group to form and dissolve resource provisioning contracts with existing

collaborated infrastructures. An existing collaboration indicates that the total number of resources to be provisioned by the resource provisioning infrastructures to a research group has been agreed. Meanwhile, resource provisioning is tracked by policies that can be defined by the group manager, while accounting for resource usage is on a per job basis. Different scenarios have been designed and applied to evaluate the proposed protocol and its implementation for different expected functions, as shown in Table 3. After evaluating these scenarios, the experiment results showed that:

1. Negotiation functions worked as designed for all negotiation phases and negotiation state updates.
2. Accounting methods functioned as designed: balances were managed and updated accordingly and correctly for unsuccessful (re-)negotiation, successful (re-)negotiation, job termination, and job completion.
3. Job submission and execution management worked properly. Job submission and execution

Table 3 Evaluated scenarios

Scenario	Scenario details
1	Successful negotiation is conducted and the job completes. The requester has sufficient balance to run the specified application and the group has sufficient balance for the resources contracted between the requester and the provider.
2	Successful negotiation is conducted and the job is stopped by the deadline specified by the requester in Use Case 2. After successful negotiation and job submission, the Service Broker confirms that the submitted job has not been completed when the deadline specified by the requester is approaching.
3	Successful re-negotiation or new negotiation is conducted in Use Case 1. After successful negotiation and job submission, the running application needs more CPUs to ensure that the application can be completed in a time frame. Also, the balance of the requester and the balances for the group on the available resources are sufficient to continue job execution.
4	Negotiation is successfully conducted with rejection as a result. The rejection is caused by insufficient balance on resources for the group.
5	Negotiation is successfully conducted with rejection as a result. The rejection is caused by insufficient balance of the requester.
6	Negotiation is successfully conducted with rejection as a result. The rejection is caused by the requester requesting a resource with a higher priority than she/he is allowed to access.
7	Termination is required by the requester during negotiation before an <i>AcceptAck</i> is sent.
8	Successful negotiation is conducted with a requester termination during job execution. The requester sends a contract termination request to stop the application execution.
9	Successful negotiation is conducted and job execution is controlled by the maximum CPU time/cost set by the group manager. This scenario assumes that the application would be executed immediately after submission. After job submission, the CPU time/cost of application execution approaches the maximum limit set by the group manager, or the requester's balance or the contracted resource's balance for the whole group approaches 0.

states were updated correctly for different scenarios: successful and failed job submission, successful job completion, and job termination required by users or the manager.

6.3.2 Automatic Negotiation Performance Evaluation and Results

Automatic negotiation is achieved by integrating negotiation procedures with the ontologies and programs developed for resource matchmaking as presented in [29]. The two types of matchmaking enabled are application-oriented and resource-oriented matchmaking. Application-oriented matchmaking searches for satisfactory resources according to specific demands for application execution, such as a deadline and the required CPU number. It indicates that a research group has established a collaboration with

an infrastructure for resource provisioning, and customised execution environment has been deployed if it is required. Resource-oriented matchmaking is activated by a full package of information for application execution, including the required CPU model, CPU speed, memory size, cost per hour. Such information is encoded in ontologies and will be fetched automatically from ontologies. Both matchmaking procedures consider resource sharing rules specified and managed by a group manager. Such rules include information on the balance for each group member, the maximum number of CPU time or cost for each application execution for each member, and each member's privilege in the group.

The negotiation performance between the Service Broker and Client Service has been evaluated. This evaluation takes advantage of the automatic negotiation capability enabled in the Client Service,

Table 4 Automatic negotiation performance

Performance evaluation scenario	Negotiation average (ms) with standard deviation	Round-trip average (ms) with standard deviation
1. Successful negotiation: all 4 contracted instances could meet demands and one instance was selected to form a contract	2166±92.2	0.9±0.1
2. Successful re-negotiation: pure re-negotiation procedures were measured	1778±171.3	1.0±0.2
3. Successful negotiation with other contracted instances after failed re-negotiation	4017±126.5	1.0±0.1
4. Successful negotiation after failed re-negotiation and failed negotiation with the other 3 contracted instances: this scenario was evaluated by resource-oriented matchmaking with 10 AWS instances	2157±82.3	0.9±0.1
5. Successful negotiation after failed re-negotiation and failed negotiation with the other 3 contracted instances: this scenario was evaluated by resource-oriented matchmaking with 50 AWS instances	2166±76.9	0.9±0.1
6. Successful negotiation with rejection, because of no sufficient balance for the requester	348±31.3	1.0±0.1
7. Successful negotiation with rejection, because of no sufficient balance for all contracted instances: resource-oriented matchmaking was disabled in this case	440±26.7	1.0±0.1
8. Successful negotiation with rejection, because of no satisfactory instance: none of the contracted 4 instances can provide the required number of CPUs; resource-oriented matchmaking was disabled for this scenario	586±148.0	1.0±0.1

avoiding unmeasurable manual procedures. Accordingly, the ontologies applied contained information for AWS instances. Information about 4 instances was checked for application-oriented matchmaking, while 10 and 50 instances were checked for resource-oriented matchmaking. The latter case aimed at measuring the scalability of the negotiation capability of the Service Broker, regarding negotiation with different numbers of resources. The number of members considered for resource sharing management within a research group was 15.

In this evaluation, negotiation requests were sent from a client program running on Eclipse. The requests were sent to the Client Service, which activated negotiation procedures. Then, the duration of the negotiation procedures was measured.

AWS does not provide a benchmark or tools to measure the real-time network performance of the instances used. Instead, we used the ping command to measure the real round-trip time for communication between the Client Service and the Service Broker, to give a hint of the network performance during negotiation [35]. The ping command was activated in the Client Service before the first negotiation message was sent to the Service Broker. Each ping command execution was repeated 10 times, and the round-trip average duration and the deviation of duration were fetched, as shown in Table 4. The negotiation of each scenario was repeated 100 times. Then, average and standard deviation were calculated for the duration of negotiation, as also shown in Table 4.

The performance data shown in Table 4 excludes the first enquiry to the Service Broker. The first enquiry requires initiation of the web services, including establishing a connection with the database, and takes longer than enquiries afterwards. The experiment verifies that after initiation, the duration of negotiation is not affected by different enquiries.

The scenarios designed in Table 4 aim at evaluating all enabled automatic negotiation procedures: negotiation with contracted instances, which involves application-oriented matchmaking for resource searching; negotiation with un-contracted instances, which involves re-negotiation and resource-oriented matchmaking; and negotiation with rejection as results. We also evaluated the scalability of the negotiation procedures by measuring the negotiation performance with different numbers of instances.

Table 4 shows that Scenario 3 consumed more time than the other scenarios. The longer negotiation duration was caused by the three negotiation procedures included: successful negotiation with one collaborated instance, failed re-negotiation with the contracted instance, and successful negotiation with another collaborated instance. Scenario 4 and Scenario 5 involved combination of sub-offers while negotiating with collaborated instances. The request asked for 5 CPUs while the 4 collaborated instances each could only provide 1 CPU. As a result, the matchmaking processes figured out that the 4 collaborated instances could not collaboratively provide the required 5 CPUs, and activated resource-oriented matchmaking. So far, the developed programs return all satisfactory offers, including all satisfactory combinations of sub-offers. In practice, the returned offers and the algorithms of sub-offer combinations can be determined by specific demands from applications, projects, e-Scientists, etc., which may contribute to varied performance.

In summary, as shown in Table 4, duration of the developed complete automatic negotiation over a distributed network was only a few seconds at most. This suggests that the developed automatic negotiation can be applied where the job duration is anything over a few seconds. It also shows that the negotiation with more resources did not increase negotiation time significantly, by comparing the performance of Scenario 4 and Scenario 5.

6.3.3 Comparison with Other Approaches

Comparison with existing solutions to resource provisioning via negotiation has been discussed in the related work. Here, we compare the Service Broker with some widely applied production tools, as shown in Table 5. The comparison is focused on the support for application management and accounting granularity for research groups, as these production tools have not facilitated negotiable resource provisioning at the moment. Application management and accounting are considered as the two other main contributions of the testbed. The information model enabled for resource management and authentication credentials required from e-Scientists are also discussed, which are the other two advantages of the Service Broker.

As shown in Table 5, application management can be realised by developing an additional layer on top

Table 5 e-Science tools functionality comparison

	Service Broker	UNICORE 7	GWpilot	Meta-Broker	EGI Marketplace	AWS
Tool type	broker	(web) gateway	broker	broker	academic Cloud platform	commercial Cloud platform
Application management	yes	yes	yes	yes	partly	no
Accounting (for manager)	per job	per VO	—	—	per VO	per account /per service
Information model	extended GLUE 2.0	GLUE 2.0	GLUE-style tags	JSDL-compatible	GLUE 2.0	provider specific
Requester credential	username+pwd /certificate	certificate	certificate/insecure mode	certificate	username+pwd certificate	—

of Grid middleware. Many tools are available for this purpose, the UNICORE gateway [38], GWpilot broker [36], and Meta-Broker [37] being three examples. They allow an e-Scientist to specify the application to submit and select a resource, rather than giving resource details. This is similar to AHE3 and the Service Broker. Additionally, a generic web interface is also provided by UNICORE to allow e-Scientists access resources in a lightweight manner, using usernames and passwords. The EGI Marketplace is an instance of an academic Cloud platform. It provides application management tools and an application database to help e-Scientists set up execution environments efficiently [39]. However, when the required execution environment is not available, it needs to be developed and deployed from scratch. Similarly, application management needs to be realised by software developers manually if needed when using AWS. As a result, the authentication credentials required for an e-Scientist to access deployed resources will be project-specific for applications deployed on AWS.

Regarding the accounting granularity, all tools for VO-based collaborations, namely the UNICORE gateway and Marketplace, support resource management at a VO level. This means that a report that contains information of the number of resources consumed by all members in a VO for a certain time period can be provided by these tools. For instance, a VO manager in the Marketplace can view the total CPU

time, the computation monetary cost, the memory, etc., that have been consumed by all members in the VO per month [40]. Even though AWS aims to provide detailed accounting information for service usage, it can only show the following accounting data to a group manager [41, 42]: (i) the total cost consumed by a member, if he/she has an AWS account, which includes all services consumed by this member; and (ii) the total cost consumed by a service, which may be contributed by multiple/all members of the group. Accounting for resource consumption is not discussed in GWpilot [36] and Meta-Broker [37].

Table 5 also shows that apart from the Meta-Broker and AWS, the resource management model implemented in all other tools is GLUE 2.0 or GLUE2.0-compatible. As studied in [11], JSDL (Job Submission Description Language) that is used by Meta-Broker is compatible with GLUE 2.0, while the resource management model applied in the Service Broker has also been used for management for services by AWS. Based on these, the resource management functions available in the Service Broker are naturally interoperable with all other tools in Table 5.

The comparison concludes that the Service Broker can realise not only dynamic and customised resource provision via negotiation, but also fine-grained resource sharing for research groups. It enables research groups to manage resource sharing among group members while resources can be provided from different

infrastructures, via a single tool. These functions are not available in existing infrastructures but are considered by this paper to be highly demanded. The reasons are: (i) the increasingly growing demands of using resources from different infrastructures according to e-Scientists' requirements; and (ii) the increasing application of virtualisation for dynamic and customised resource provision for computational experiments. Both are separating research groups from resource providers, requiring dynamic and accountable resource provisioning management.

7 Conclusion

The e-Science community is striving to find a solution to satisfy e-Scientists' dynamic and customised resource provisioning demands while considering the requirements of a research group to manage resource sharing among its members (i.e. the e-Scientists). In this paper, we have presented a negotiation protocol as a solution. Via communicating with the Resource Manager for resource provisioning decisions, the proposed protocol enables fine-grained resource sharing on a per job basis for a research group. Meanwhile, infrastructures only need to be concerned about coarse-grained resource provisioning management for the whole group. Distinguishing a research group from resource provisioning infrastructures also makes it possible to form collaborations with different infrastructures without affecting the inner organisation structure of the research group.

A testbed has been built with a production e-Science gateway AHE3, integrating production use cases and infrastructures. The evaluation of the tested verified that access, controlled by a group manager of an e-Scientists' collaboration for negotiation decisions, can contribute to accountable resource provisioning per job, without binding the research group with specific infrastructures. An analysis and comparison of the support for application management, accounting, resource management modelling, and authentication credentials from e-Scientists have also been conducted between the Service Broker and other available production e-Science tools. We safely conclude that the proposed negotiation protocol can satisfy e-Scientists' dynamic and customised resource provisioning demands while managing fine-grained resource sharing for a research group.

Funding Information Open access funding provided by University of Manchester (UoM)

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Marina, J., Charlotte, P.L., Gary, M.O.: Supporting scientific collaboration: Methods, tools and concepts. *Computer Supported Cooperative Work (CSCW)* **22**(4-6), 667–715 (2013)
2. Piotr, B., Krzysztof, B., Rafał, K., Grzegorz, M.: Advancements in UNICORE accounting. In: *UNICORE Summit*, vol. 2013, pp. 83–94 (2013)
3. Rachida, A.: Report on the open market consultation and the results Zenodo. <https://doi.org/10.5281/zenodo.51592> (2016)
4. EGI: Marketplace - e-Infrastructure Services for Research. <https://marketplace.egi.eu/>. Cited 15 Dec 2018 (2018)
5. Fernández-Quiruelas, V., Blanco, C., Cofiño, A.S., Fernández, J.: Large-scale climate simulations harnessing clusters, grid and cloud infrastructures. *Future Gener. Comput. Syst.* **51**, 36–44 (2015)
6. Krasovec, B., Andrej, F.: Enhancing the grid with cloud computing ARC-CC: ARC cluster in the cloud. *J. of Grid Comput.* **17**(1), 119–135 (2019)
7. Altunay, M., Avery, P., Blackburn, K., Bockelman, B., Ernst, M., Fraser, D., Quick, R., Gardner, R., Goasguen, S., Levshina, T., Livny, M.: A science driven production cyber-infrastructure — the open science grid. *J. of Grid Comput.* **9**(2), 201–218 (2011)
8. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. *The Int. J. of High Perform. Comput. Appl.* **15**(3), 200–222 (2001)
9. Solagna, P.: AAI in EGI Current Status. <https://documents.egi.eu/public/ShowDocument?docid=2185>. Online; Accessed 15 Dec 2018 (2015)
10. Calheiros, R.N., Kasim, H., Hung, T., Li, X., Lu, S., Wang, L., Palit, H., Lee, G., Ngo, T., Buyya, R.: Adaptive execution of scientific workflow applications on clouds. In: Terzo, O., Mossucca, L. (eds.) *Cloud Computing with e-Science Applications*, pp. 93–108. CRC Press (2015)
11. Meng, Z.: Negotiation protocol for accountable resource sharing in e-science collaborations. Ph.D. thesis, chapter 2, pp. 69–72 (2018)

12. Zasada, S.J., Chang, D.C., Haidar, A.N., Coveney, P.V.: Flexible composition and execution of large scale applications on distributed e-infrastructures. *J. of Computational Sci.* **5**(1), 51–62 (2014)
13. Comuzzi, M., Kritikos, K., Plebani, P.: A semantic based framework for supporting negotiation in service oriented architectures. In: 2009 IEEE Conference on Commerce and Enterprise Computing, pp. 137–145. IEEE (2009)
14. Kertész, A., Kecskemeti, G., Brandic, I.: An interoperable and self-adaptive approach for SLA-based service virtualization in heterogeneous Cloud environments. *Future Gener. Comput. Syst.* **32**, 54–68 (2014)
15. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: Workshop on Job Scheduling Strategies for Parallel Processing, pp. 153–183 (2002)
16. Ali, R.J.A., Rana, O.F., Walker, D.W., Jha, S., Sohail, S.: G-QoS: Grid service discovery using QoS properties. *Computing and Inform.* **21**(4), 363–382 (2012)
17. Waeldrich, O., Battré, D., Brazier, F., Clark, K., Oey, M., Papaspyrou, A., Wieder, P., Ziegler, W.: WS-Agreement negotiation version 1.0. Technical Report GFD-R.193 Open Grid Forum (2011)
18. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web services agreement specification (WS-Agreement). Technical Report GFD.192 Open Grid Forum (2007)
19. Merzky, A., Papaspyrou, N.W.A.: WS-Disagreement. Technical Report GFD.199 Open Grid Forum (2012)
20. Adabi, S., Movaghar, A., Rahmani, A.M., Beigy, H., Dastmalchy-Tabrizi, H.: A new fuzzy negotiation protocol for grid resource allocation. *J. Netw. Comput. Appl.* **37**, 89–126 (2014)
21. Zasada, S.J., Coveney, P.V.: A distributed multi-agent market place for HPC compute cycle resource trading. *arXiv* (2015)
22. Demchenko, Y., De Laat, C., Lopez, D.R., García-España, J.A.: Security services lifecycle management in on-demand infrastructure services provisioning. In: Cloud Computing Technology and Science (CloudCom), pp. 644–650 (2010)
23. Siegel, S.F.: Model checking nonblocking MPI programs. In: International Workshop on Verification, Model Checking, and Abstract Interpretation, pp. 44–58 (2007)
24. Parkin, M.S.: Lightweight client organisations for the computational grid. Ph.D. thesis, University of Manchester (2007)
25. Brooke, J.M., Parkin, M.S.: Enabling scientific collaboration on the Grid. *Future Gener. Comput. Syst.* **26**(3), 521–530 (2010)
26. David, P.A., Spence, M.J.: Towards institutional infrastructures for e-Science: The scope of the challenge SSRN. <https://doi.org/10.2139/ssrn.1325240> (2003)
27. Radecki, M., Szymocha, T., Szepieniec, T., Różańska, R.: Improving PL-Grid operations based on FitSM standard. In: Bubak, M., Kitowski, J., Wiatr, K. (eds.) *eScience on Distributed Computing Infrastructure*, vol. 8500. Springer, Cham (2014)
28. Holzmann, G.J.: The SPIN model checker: Primer and reference manual. Massachusetts Boston (2004)
29. Meng, Z., Brooke, J., Sakellariou, R.: Semantic accountable matchmaking for e-Science resource sharing. In: 2016 IEEE 12th International Conference on e-Science, pp. 282–286 (2016)
30. Han, J., Brooke, J.: Hybrid computational steering for dynamic data-driven application systems. *Procedia Comp. Sci.* **80**, 407–417 (2016)
31. Riedel, M., Laure, E., Soddemann, T., Field, L., Navarro, J.P., Casey, J., Litmaath, M., Baud, J.P., Koblit, B., Catlett, C., Skow, D.: Interoperation of world-wide production e-Science infrastructures. *Concurrency and Computation: Prac. and Experience.* **21**(8), 961–990 (2009)
32. Andreozzi, S., Burke, S., Ehm, F., Field, L., Galang, G., Konya, B., Litmaath, M., Millar, P., Navarro, J.P.: GLUE Specification v. 2.0. Technical Report GFD.147 Open Grid Forum (2009)
33. Fielding, R.: Architectural styles and the design of network-based software architectures. University of California. Ch. Representational state transfer (REST), Ph.D thesis (2000)
34. Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: A performance analysis of EC2 cloud computing services for scientific computing. In: International Conference on Cloud Computing, pp. 115–131 (2009)
35. Jiang, H., Dovrolis, C.: Passive estimation of TCP round-trip times. *ACM SIGCOMM Computer Communication Review* **32**(3), 75–88 (2002)
36. Rubio-Montero, A.J., Huedo, E., Castejón, F., Mayo-García, R.: GWpilot: Enabling multi-level scheduling in distributed infrastructures with GridWay and pilot jobs. *Future Gener. Comput. Syst.* **45**, 25–52 (2015)
37. Karoczkai, K., Kertész, A., Kacsuk, P.: A meta-brokering framework for science gateways. *J. Grid Comput.* **14**(4), 687–703 (2016)
38. Benedyczak, K., Schuller, B., Petrova-El Sayed, M., Rybicki, J., Grunzke, R.: UNICORE 7 - Middleware services for distributed and federated computing. In: 2016 International Conference on High Performance Computing & Simulation, pp. 613–620 (2016)
39. EGI: EGI applications database. <https://appdb.egi.eu/>. Online; Accessed 15 Dec 2018 (2018)
40. EGI: EGI accounting portal. <https://accounting.egi.eu/>. Online; Accessed 15 Dec 2018 (2018)
41. AWS: Monitoring your usage and costs. <http://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/monitoring-costs.html>. Online; Accessed 15 Dec 2018 (2018)
42. AWS: Paying bills for multiple accounts using consolidated billing. <http://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/consolidated-billing.html>. Online; Accessed 15 Dec 2018 (2018)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.