A Negotiation Mechanism for Advance Resource Reservation using the Alternate Offers Protocol

Srikumar Venugopal, Xingchen Chu, and Rajkumar Buyya
Grid Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
Email:{srikumar, xchu, raj}@csse.unimelb.edu.au

***Abstract*—Service Level Agreements (SLAs) between grid users and providers have been proposed as mechanisms for ensuring that the users' Quality of Service (QoS) requirements are met, and that the provider is able to realise utility from its infrastructure. This paper presents a bilateral protocol for SLA negotiation using the Alternate Offers mechanism wherein a party is able to respond to an offer by modifying some of its terms to generate a counter offer. We apply this protocol to the negotiation between a resource broker and a provider for advance reservation of compute nodes, and implement and evaluate it on a real grid system.**

I. Introduction

Grids [1] are evolving from a collection of computing, data and networking resources to systems of services that are discovered and invoked by scientific and business applications. These services interface with the physical resources and may integrate different functions offered by these to present unified capabilities required by the applications. Users may have Quality-of-Service (QoS) requirements covering metrics such as deadlines, fidelity, security and budget associated with service invocation. The user QoS requirements, as well as the rewards and penalties for achieving and violating them respectively, are encoded in Service Level Agreements (SLAs) that are negotiated between the providers and the users [2]. Therefore, SLAs enable guaranteed provisioning of service (or resource) capability.

This promise of SLA has motivated research and development into formulating, negotiating and establishing of such agreements between providers and users. Many grid systems have also incorporated SLA specification, exchange and monitoring for resource brokering and task scheduling [3], [4], [5], and for resource provisioning [6], [7]. The Open Grid Forum has also arrived at a standard for creation and specification of SLAs called WS-Agreement [8]. WS-Agreement follows the well-known Contract Net protocol [9] for exchange of SLAs between the user and the provider.

In this paper, we introduce a protocol for negotiating SLAs based on Rubinstein's Alternating Offers protocol [10] for bargaining between agents. This protocol allows either party to modify the proposal or to provide counter proposals so that they can arrive at a mutually-acceptable agreement. We illustrate its usage by implementing it to enable a resource consumer to reserve nodes on a shared computing resource in advance. The consumer side of the protocol is implemented in the Gridbus broker [11] and the provider side of the protocol is implemented within a .NET-based enterprise grid system called Aneka [12]. We experimentally evaluate this system using reservation requests with a range of strict to relaxed requirements, and present results. Thus, we propose, implement and evaluate a software infrastructure necessary for enabling SLA-based resource allocation and scheduling in a real grid resource management system.

The next section presents an overview of the related work. Then, the negotiation protocol is presented in the succeeding section. Following that, we detail the implementation of the advance reservation system on both the provider and the consumer side. We then present the results of experimental evaluation of the system and finally, conclude the paper.

II. Related Work

Negotiation has been employed in real-time systems to manage tasks that have different levels of priority and different QoS demands [13]. Negotiation and resource reservation has also been used in multimedia systems [14] for processing media streams with different levels of criticality and QoS requirements.

Czajkowski, et al. [6] introduced the Service Negotiation and Allocation Protocol (SNAP) for managing SLAs in distributed systems. In SNAP, task SLAs specify the resource requirements of the tasks, resource SLAs specify the amount of resources available for the user and binding SLAs represent the allocation of resources to tasks. Dumitrescu and Foster [4] present a distributed brokering architecture that takes into account SLAs between resources and Virtual Organisations (VOs) while selecting the best site to submit a job. Elmroth and Tordsson [5] also describe a brokering architecture that is able to make advance resource reservations and create SLAs using the WS-Agreement standard. These systems share a common feature – they follow the Contract Net protocol for negotiating SLAs.

Contract Net [9] is a popular protocol for agent communication that has been employed for negotiating SLAs for utility computing [15] and for grid resource management. Contract Net also forms the basis for the communication between the entities in the OGF WS-Agreement Standard. In Contract Net, agents that are contractors evaluate task announcements from other agents acting as managers and bid for executing the tasks that they are interested in. The bidding process has only two outcomes: the bid is accepted or rejected in its entirety. Therefore, Contract Net is suitable for implementing multilateral processes such as auctions. However, there is no feedback mechanism that allows the SLA proposer or receiver to modify the terms of the proposed SLA in order to converge to an agreement acceptable to both parties. Therefore, the protocol is not suitable for implementing a bilateral process such as one presented in this paper. Recently, a concurrent bilateral negotiation model, similar to the one followed in this paper, was evaluated through simulation for grid resource management by Li and Yahyapour [16]. They conclude that this model is feasible for use in grids and enables the user to obtain guaranteed QoS from grid resources.

The advent of grid computing introduced the need for advance reservation mechanisms in order to coordinate resource sharing between autonomous partners. One of the first works in this regard was the Globus Advance Reservation Architecture (GARA) [17], [18]. GARA provided a user with the capability to reserve resources such as bandwidth and compute nodes in advance. However, there was no negotiation capability within this system. Also, in GARA, reservation was separated from resource allocation and was enforced only by a late binding of successful reservations to resource objects. Another notable advance reservation architecture is that of SHARP (Secure Highly Available Resource Peering) [19] wherein cryptographically secure tickets – representing the right to access the resource at a specified future time – are generated by the resource management system. These can be further subleased to other consumers and are bound to actual resources only when they are claimed. However, while the tickets themselves can be bartered, it is not possible to directly negotiate with the system for a lease at a particular time.

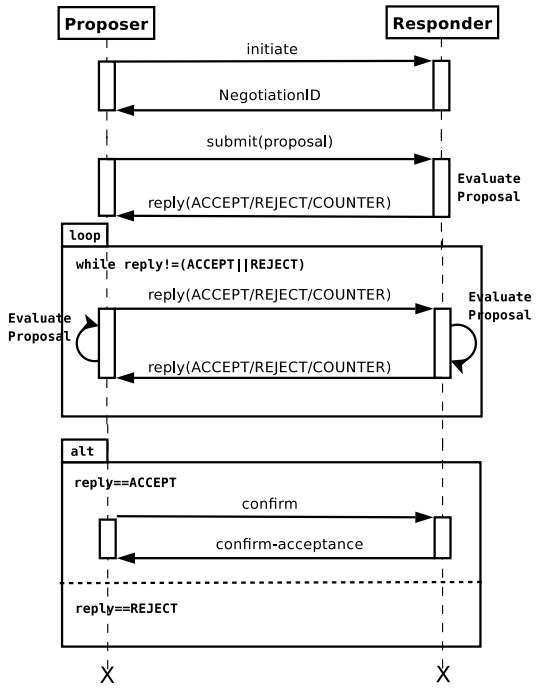

Fig. 1. Alternate Offers Protocol.

In contrast to these architectures, our system provides the ability to conduct bilateral negotiations in order to gain guaranteed reservations of resources in advance. The resource management system has the ability to generate alternate offers to consumers in case their original request cannot be fulfilled. The broker, acting as the resource consumer, has the ability to generate its own counter proposals as well. To our knowledge, no other real grid resource management system has these capabilities in its implementation.

III. The Negotiation Protocol

Figure 1 shows the alternating offers-based protocol for SLA negotiation. It is a bilateral protocol consisting of the *proposer* who initiates the process and the *responder* who replies to the proposal. The proposer starts the negotiation process by sending the `INITIATE` message to which the responder replies with a unique negotiation identifier (*negotiationID*). The initiate call

may be accompanied by an exchange of credentials so that both parties are able to verify each other's identity. The proposer then presents a proposal using the `submitProposal` message. The responder can accept or reject the offer in its entirety by sending an `ACCEPT` or a `REJECT` message as a reply. The responder can also reply with a counter-offer by using the `COUNTER` reply accompanied by the counter proposal. In this case, the proposer has the same options and therefore can reply with a counter proposal of its own. Either party can signal its dissatisfaction and abort the negotiation session by sending a `REJECT` message. In case either party is satisfied with the proposal, it can send an `ACCEPT` message to the other. To seal the agreement, the other party has to send a `CONFIRM` message and receive a `CONFIRM-ACCEPTANCE` message in reply.

IV. Implementing Resource Reservation

An advance reservation is a commitment made by a resource provider to provide a guaranteed share of a computing resource to a resource consumer at a definite time in the future [17]. An advance reservation mechanism therefore, allows a consumer to provision enough resources to meet requirements such as deadlines, in environments such as grids where availability of shared resources varies from time to time. Since an advance reservation is also a commitment by the provider, it may be made in lieu of a reward or payment specified by the provider. Failure to meet this commitment may result in the provider having to pay a penalty. Therefore, a reservation represents an instantiation of an SLA.

A provider with a profit motive would aim to maximise his revenue while minimising the risk of penalties [20]. Likewise, a consumer would like to gain the maximum amount of guarantees for meeting his QoS requirements but at the lowest cost possible. A number of strategies can be adopted by both the provider and the consumer depending on their individual needs and situations. As a result of these, a consumer's plan for resource usage may not be favored by a provider. However, the provider can indicate its expectations by changing the relevant parts of the proposal and returning it to the consumer. In this manner, proposals can be exchanged back and forth until both parties reach an agreement or decide to part ways.

In the following subsections, we describe the implementation of negotiation for advance reservation of resources using Aneka and the Gridbus Grid resource broker. Aneka is a .NET-based resource management system for enterprise grids composed of computers running Microsoft Windows operating system. Therefore, it acts as the resource provider in this implementation. For a given user application, the Gridbus broker discovers appropriate resources for executing the application, schedules user jobs on the resources, monitors their execution and retrieves the results once they are completed. Negotiation for advance reservations is therefore performed by the Gridbus broker as a resource consumer on behalf of the user. Currently, the only user QoS requirement that this system aims to achieve is that of the application execution meeting a certain deadline.

A. Aneka

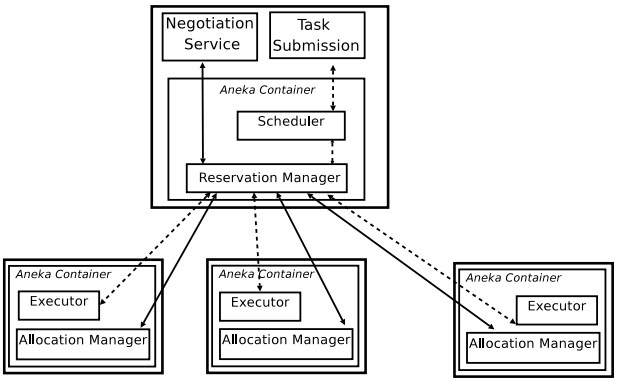

Fig. 2. The Aneka resource reservation architecture.

In Aneka, the capabilities of each node in the system are determined by the functionality offered by the services hosted in a service container that provides common security, message handling and communication functions. For example, hosting a task executor service in the container enables a node to execute independent tasks. Any number of such services may be hosted thereby, potentially allowing the same node to execute applications implemented using different programming models. A node functions as a scheduler for an application if it hosts the scheduler service corresponding to the application's programming model (e.g., task scheduler for the task farming model). Executors in a Aneka grid register with or are discovered by a specific scheduler service which then allocates work units across them.

Figure 2 shows the architecture for resource reservation in Aneka. The advance reservation capability in Aneka is enabled by two components, the Allocation Manager at the executor end and the Reservation Manager at the scheduler end. The Allocation Manager underlies all the executor services on a node. It determines which of the executors are allowed to run, and the share of the node that is allowed for each. The Allocation Manager therefore takes care of allocating and enforcing reservations on a single node. The Allocation Manager

is associated with a policy object that encodes the utility function of the node. For example, this may specify a maximum duration that can be specified for a reservation request at the node level.

The Reservation Manager is co-located with a scheduler and is able to perform reservations across the nodes whose executors are registered with the scheduler. The Reservation Manager determines which of the reservation requests coming from users are to be accepted based on factors such as feasibility, profitability or improvement in utilisation. For this reason, it is associated with a QoS Policy object that represents the reservation policy at the level of the entire system. For example, this object may specify a minimum reward for considering a reservation request. External applications interface with Aneka's resource reservation system through the Negotiation Service. This web service implements the negotiation protocol presented in Section III and interfaces with the Reservation Manager for realising the reservation requests that arrive from external entities. The web service was required in order for non-.NET programs to interface with the Aneka system. The task submission is also mediated by the resource reservation architecture. Tasks that arrive with a valid reservation ID, assigned in case of successful requests, are scheduled on to the nodes that are associated with that ID.

```
<xml-fragment
xmlns:ws="http://www.gridbus.org/negotiation/">
  <ws:Reward>1000.0</ws:Reward>
  <ws:Penalty>0.0</ws:Penalty>
  <ws:Requirements>
    <ws:ReservationRecordType>
         <ws:ReservationStartTime>
                2007-11-17T18:24:37.078+11:00
         </ws:ReservationStartTime>
         <ws:Duration>150000.0</ws:Duration>
         <ws:NodeRequirement>
                <ws:Count>4</ws:Count>
         </ws:NodeRequirement>
         <ws:CpuRequirement>
             <ws:Measure>Ghz</ws:Measure>
                  <ws:Speed>2.5</ws:Speed>
         </ws:CpuRequirement>
    </ws:ReservationRecordType>
  </ws:Requirements>
</xml-fragment>
```

Fig. 3. The proposal document.

As per the protocol presented in Section III, when the broker sends an initiate message, the Aneka Negotiation Service returns a globally unique identifier for the session. The broker then submits a proposal to the Negotiation Service in the XML format shown in Figure 3. The proposal is parsed and converted to a reservation requirement object that is sent to the Reservation Manager.

```
    At the Reservation Manager
1.  for each incoming reservation request do
2.      if (QoS Policy is violated) then
3.          send(REJECT)
4.      Get available nodes from Information Service
5.      Filter the nodes as per requirements
6.      if (requested nodes < available nodes) then
7.          send(REJECT)
8.      Broadcast requested timeslot to all available
        nodes
9.      Wait for response
10.     if (agreed nodes ≥ required nodes) then
11.         send(ACCEPT)
12.     else
13.         Find the common timeslot among the largest
            number of nodes ≥ number of required
            nodes
14.         if (timeslot is found) then
15.             send(COUNTER,new_timeslot)
16.         else
17.             send(REJECT)
18.     end
19. end
    At the Allocation Manager
20. for each incoming request do
21.     if (reservation policy is violated) then
22.         send(REJECT)
23.     else if (timeslot is available) then
24.         send(ACCEPT)
25.     else
26.         send(COUNTER,new_timeslot)
27. end
```

Fig. 4: Handling resource reservation in Aneka.

The current algorithm for handling resource reservation requests in Aneka is shown in Figure 4. A *timeslot* is the period for which the reservation is required. Lines 2-3 control the admission of requests as per the policy specified in the QoS Policy object. Once the request is approved, the request is broadcast to all the available nodes in the grid. At the node, the Allocation Manager checks if its reservation policy is violated. If not, and the node is free for the requested timeslot, the request is accepted. If the node is not free, then an alternate time slot is provided to the Reservation Manager. If the Reservation Manager is not able to find the required number of nodes for the timeslot asked by the user, then it tries to find a new timeslot of the same duration for the same number of nodes requested by the user. This is then sent as a counter proposal to the consumer.

When a proposal is finally accepted, the Reservation Manager executes a two phase commit to finalise the

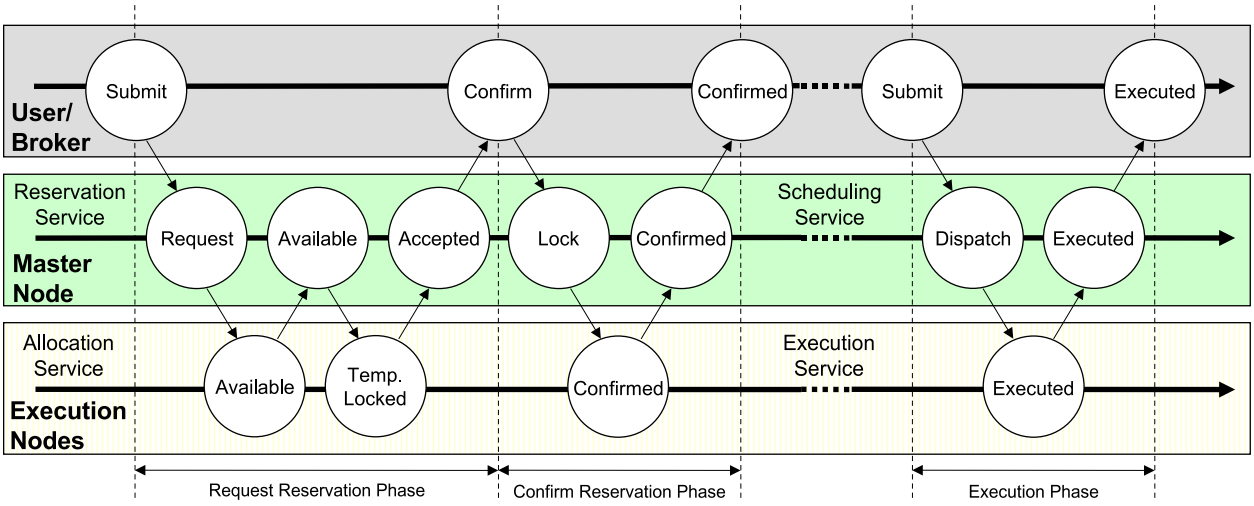

Fig. 5. Control flow for a successful resource reservation.

reservation . In the initial phase, it requests the respective Allocation Managers to "soft" lock the time slot for that particular request. A soft lock in this case is an entry for the time-slot in the Allocation Manager database which is removed if a confirmation is not received within a certain time-interval. Once all the nodes successfully acknowledge that this operation has been performed, the reservation manager then sends an ACCEPT message to the broker. If the broker then sends a CONFIRM message, the Reservation Manager asks all of the Allocation Managers to commit the reservation. On receiving their acknowledgement, a CONFIRM ACCEPTANCE message is returned to the broker. The negotiation session identifier is then used as a reference for the resource reservation by subsequent tasks. This process is shown in Figure 5.

B. Gridbus Broker

The Gridbus broker has been used to realise economy-based scheduling of computational and data-intensive applications on grid resources [21]. Advance reservations are a means for the broker to provide guarantees for meeting the user's QoS requirements for the execution, such as deadline and budget. The required abilities for negotiation within the broker are brought about by a *negotiation-aware* scheduler and a *negotiation client*.

The negotiation client is the interface to the corresponding service on the remote side (e.g. the Negotiation Service of Aneka). It is not implementation-specific and can support any other middleware that implements the protocol. The scheduler is aware of the negotiation client only as a medium for submitting proposals and receiving feedback from the remote side. However, separate schedulers may be required for different SLA negotiation protocols, as certain features (e.g., presence or absence of a counter-proposal method) may impact negotiation and scheduling strategies.

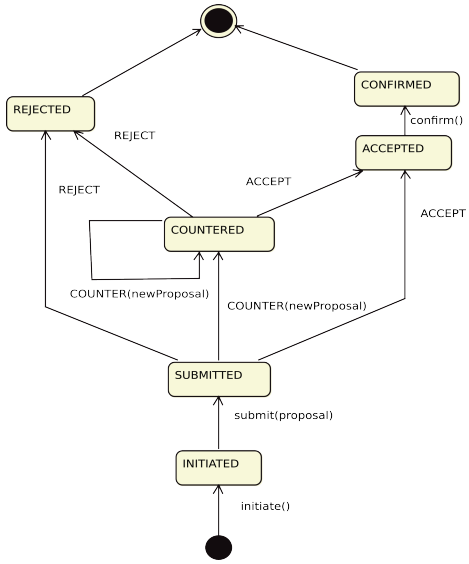

Fig. 6. Negotiation state machine.

The broker keeps track of the negotiation process through a state machine detailed in Figure 6 and implemented using the State software design pattern. The actions are encoded in the State objects which prevents the broker from performing invalid actions in certain states, say for example, replying to a REJECT message with a CONFIRM message. The transition between the states is guided by the broker's strategy and the responses

from the provider.

```
1.  Get user's QoS and application requirements
2.  Nodes ← (∑ Est(j)) / (f × (deadline − start_time))
3.  Create proposal for Nodes
4.  Choose a provider based on attributes such as cost
5.  repeat
6.      Submit proposal to the provider
7.      repeat
8.          if (state is COUNTERED) then
9.              if (counter proposal is within deadline)
                then
10.                 send(ACCEPT)
11.             else if (f < 1) then
12.                 Increase f
13.                 Recalculate Nodes
14.                 Create new proposal for Nodes
15.                 send(COUNTER,proposal)
16.             else
17.                 send(REJECT)
18.         end
19.         if (state is ACCEPTED) then
20.             send(CONFIRM)
21.         end
22.     until (a final state is reached)
        //Final state is REJECTED or CONFIRMED or
          FAILED
23.     if (previous state was REJECTED or FAILED)
        then
24.         Find another provider to repeat the process
25. until (enough nodes are obtained)
26. Wait until reservation start time
```

Fig. 7: The broker's negotiation strategy

A broker is associated with a single distributed bag-of-tasks application. The deadline is provided for each application as a whole. The expression in Line 2 calculates the number of nodes that are required for executing the distributed application within the deadline. The estimated time for completing a job is provided by the user. The broker adds to this an additional estimate for staging the jobs on to the remote machine, invoking it and collecting the results for the job. The total estimated time for each job is added up to obtain the maximum time required to execute the application (i.e. its sequential execution time on a single remote processor). This is the numerator in the expression in Line 2.

The denominator is the wallclock time available to execute the application. This is time difference between the deadline and the starting time for the reservation. The starting time is estimated as the time when the negotiations would have likely concluded and the job scheduling can commence. Since the broker's utility lies in executing the users' job as quickly as possible, the time available is further reduced by multiplying against an aggression factor, denoted by f, where $0 < f \leq 1$. However, the smaller the time available, the larger is the number of nodes required.

The broker creates a proposal and choses one out of a list of resource providers – based on factors such as resource price or capability – to initiate a negotiation session and submit the proposal. If the proposal is accepted straightaway, then a confirmation message is returned to the provider. If a counter proposal is received, then it is evaluated to see whether the counter reservation is still within the deadline. If so, then it is accepted by the broker. If not, then the aggression factor is increased to reduce the number of nodes required. This is done on the assumption that requests for smaller number of nodes have better chances to be accepted or found more acceptable (earlier) counter time slots. This continues until the aggression factor is increased upto 1 which is the maximum latitude available to broker. If the counter proposal from the resource provider does not satisfy the deadline requirements, the proposal is rejected and the session closed.

V. Experimental Evaluation

The negotiation architecture described previously was evaluated using a grid testbed constructed by installing Aneka on 13 desktop computers running Microsoft Windows XP in a local area network. One instance of Reservation Manager service was installed on the node acting as the scheduler and the others ran the Allocation Manager service. This means that upto 12 nodes could be reserved by brokers by interacting with the sole Reservation Manager using the negotiation protocol described in previous sections.

In order to emulate multiple clients with different applications that have different deadlines, a set of brokers was created with different deadlines generated using a uniform random distribution. The deadlines were chosen so as to reflect different levels of urgency - from a strict deadline for a high-urgency application to a relaxed deadline for a low-urgency application. The urgency was calculated from the following ratio

$$r = \frac{deadline - start\_time}{max\_execution\_time}$$

where $start\_time$ is the estimated start time for the reservation and $max\_execution\_time$ is the maximum time estimated for executing the complete application. In this evaluation, the sequential execution time is considered as the maximum execution time for the application. The deadline is considered *very strict* when $r < 0.25$, *moderately strict* when $0.25 < r < 0.5$, *relaxed* when $0.5 < r < 0.75$, and *very relaxed* when $r > 0.75$.

The maximum execution time was same for all the applications in this evaluation.

According to the algorithm in Figure 4, when the broker makes a request and Aneka is not able to provide the required number of nodes at the requested start time, the latter finds an alternative start time when the nodes can be provided. The difference between the alternative start time and that requested originally is termed as the *slack*. The slack allowed for reservation start time is a function of the urgency of the deadline, and indicates the relaxation allowed in the broker's requirements.

The brokers were launched at closely-spaced intervals from two computers that were part of the same local area network but separate from the grid nodes. This created the effect of different requests with different deadlines arriving simultaneously at the Reservation Manager. The objectives of this exercise are to measure the impact of deadlines on the responses adopted by both the broker and the Reservation Manager.

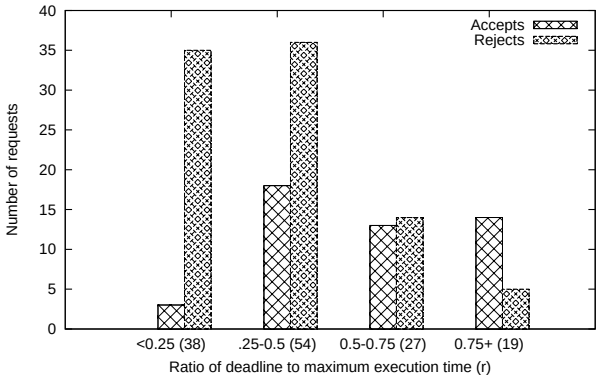

Fig. 8. Distribution of decisions against deadline urgency

Figure 8-10 shows the results of an evaluation that involved 138 advance reservation requests arriving at the Aneka Reservation Manager in the space of 4 hours. The numbers in parantheses against a point on the x-axis show the total number of requests corresponding to that data point. Nearly 17% of the total requests were decided in the first round itself (i.e., a straightaway accept or reject decision from Aneka) while the rest were decided after multiple rounds of negotiation between the broker and the Manager. In all, 35% of the requests were accepted while 65% of the requests were rejected. Since the evaluation covered a scenario where the demand for computing nodes would exceed their supply, it is only to be expected that a majority of the requests will be rejected. However, the system was still able to generate alternatives for 83% of the requests.

Figure 8 shows the distribution of the accepted and rejected requests against the urgency of application deadlines. It can be seen that the proportion of accepted requests increases when the deadlines progress from very strict to very relaxed. When normalised against the number of requests for each data point, the percentage of accepted requests increases from 8% for strict deadlines to 74% in the case of very relaxed deadlines. This is because the broker is more willing to accept a delayed reservation when the deadlines allow more slack. Also, due to the strategy adopted by the broker (Figure 7), applications with urgent deadlines require more nodes for a shorter duration than those with relaxed deadlines. Aneka was therefore able to generate better counter offers for requests involving lesser number of nodes, even if their duration is longer.

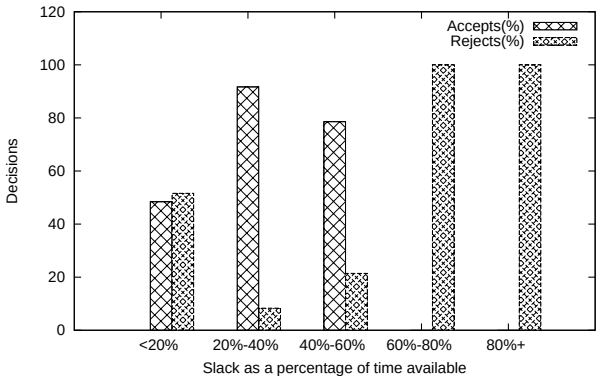

Fig. 9. Distribution of decisions according to delay in reservation start time.

This inference is supported by the graphs in Figure 9 which show the percentage of accept and reject decisions according to the slack allowed in the reservation start time. The slack is indicated as a percentage of the time available (i.e. deadline minus original start time) for the broker to execute the application. It can be seen here that the broker is willing to accept counter-offers with up to 60% slack in reservation start time. Indeed, 90% of the counter-offers with up to 40% slack are accepted by the broker. However, counter-offers with more than 60% slack are unacceptable. A significant amount of proposals are rejected by the Reservation Manager without counter-offers (zero slack time) as they require more nodes than what is available. These are included in the data point corresponding to offers with $<$20% slack at the far left of Figure 9.

A request-response pair between the broker and the Aneka Reservation Manager is termed as a *round* of

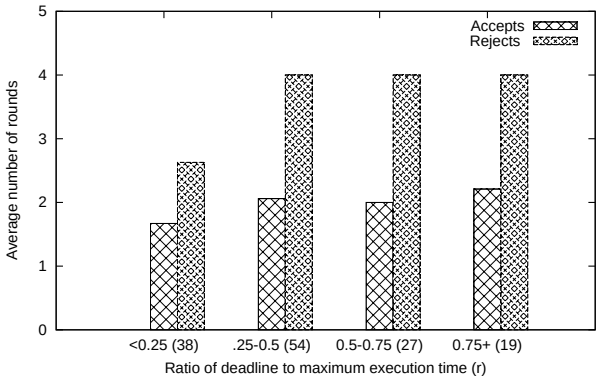

Fig. 10. Number of rounds against urgency of deadlines.

negotiation. Figure 10 shows the average number of negotiation rounds taken to obtain a result for requests with different deadlines. For this evaluation, the aggression factor was set to 0.5 and then increased by 0.25 for every round. Therefore, including the submission request, a maximum of 4 rounds (3 offers each and a final decision) was possible for this evaluation. For very strict deadlines, many of the offers were rejected or accepted in the first round itself. Therefore, the average number of rounds is the least in this case. For more relaxed deadlines, the broker is willing to negotiate for the maximum number of rounds before the request is rejected.

A. Discussion

The important result here is that the broker was able to fulfil its QoS requirement without having to reveal its deadline preference to the provider by choosing an acceptable counter proposal whenever possible. Therefore, by modifying the proposal suitably, both parties were able to convey feedback without revealing their preferences. This prevents providers from taking undue advantage or playing consumers against each other in scenarios where different brokers may be competing for access to the same set of resources.

The implementation involved tackling some significant challenges. In the first iteration, synchronous web service calls were used between the broker and Aneka. However, this was replaced with asynchronous calls to increase the request handling ability of Aneka Negotiation Service. Another significant problem encountered was the clock skew between Aneka nodes. This acutely affected the performance of the reservation system which was based on the assumption of a globally synchronised clock mechanism to guarantee a particular timeslot. While the current solution is to ensure that the nodes are synchronised through Network Time Protocol, it is still a challenge to detect clock skew.

VI. Conclusion and Future Work

This paper presented a bilateral negotiation mechanism based on the Alternate Offers Protocol in which each party has the opportunity to submit a counter proposal so that a mutually acceptable agreement can be arrived at. The protocol was used to enable the advance reservation of nodes in an enterprise grid system called Aneka, by the Gridbus grid resource broker. The results of the evaluation show that brokers with relaxed requirements benefited from the counter proposals as they were able to accommodate delays in starting reservations. Thus, this shows the potential of the system to facilitate the implementation of SLA-based resource allocation and scheduling strategies.

The system currently supports only the negotiation for timeslots and number of resources. Therefore, more sophisticated strategies that take into account other attributes such as rewards and penalties are planned in the future for both the broker and Aneka Reservation Manager. In addition, we would also like to explore malleable reservations wherein both the timeslots and the number of nodes are changed simultaneously. In this respect, we would like to realise some of the ideas that have been explored in this space in the recent past [22], [23]. Also, at present, the broker depends on user estimates of application execution times to compute the number of nodes required. It is well-known that such estimates are inherently inaccurate [24]. In the future, we would like to explore scheduling strategies that can hedge against errors in estimates by overprovisioning nodes or by continually re-evaluating the requirement for computing nodes as the application execution proceeds.

Acknowledgements

We thank Chee Shin Yeo for creating Figure 5. We would also like to thank Marco Assuncao for his comments on the paper. This project was funded by Australian Research Council and Dept. of Education, Science and Training under Discovery Project and International Science Linkage grants respectively.

References

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a Future Computing Infrastructure*. San Francisco, USA: Morgan Kaufmann Publishers, 1999.

[2] R. J. Al-Ali, K. Amin, G. von Laszewski, O. F. Rana, D. W. Walker, M. Hategan, and N. Zaluzec, "Analysis and provision of qos for distributed grid applications," *Journal of Grid Computing*, vol. 2, no. 2, pp. 163–182, June 2004.

[3] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, and K. Krishnakumar, "A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing." in *Proceedings of the 2005 European Grid Computing Conference (EGC 2005)*, 2005, pp. 651–660. [Online]. Available: http://dx.doi.org/10.1007/11508380_66

[4] C. L. Dumitrescu and I. Foster, "Gruber: A grid resource usage sla broker," in *Proceedings of the 11th International Euro-Par Conference on Parallel Processing, Lisbon, Portugal*, ser. LNCS, no. 3648. Springer-Verlag, Berlin, Germany, August 2005.

[5] E. Elmroth and J. Tordsson, "A grid resource broker supporting advance reservations and benchmark-based resource selection." in *State-of-the-art in Scientific Computing*, ser. LNCS. Springer-Verlag, Berlin, Germany, 2006, vol. 3732, pp. 1061–1070.

[6] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems," in *Proceedings of the 8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2002), Edinburgh, Scotland*. Springer-Verlag, Berlin, Germany, 2002, pp. 153–183. [Online]. Available: http://www.springerlink.com/content/1e8rjyg17bep4nvc

[7] R. Ranjan, A. Harwood, and R. Buyya, "Sla-based coordinated superscheduling scheme for computational grids," in *Proceedings of the 8th IEEE International Conference on Cluster Computing (Cluster 2006), Barcelona, Spain*. IEEE CS Press, Los Alamitos, CA, USA, 2006.

[8] A. Andrieux *et al.*, "Web services agreement specification (ws-agreement)," Open Grid Forum, Tech. Rep. GFD.107, 2007.

[9] R. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on Computers*, vol. C-29, no. 12, pp. 1104–1113, 1980.

[10] A. Rubinstein, "Perfect equilibrium in a bargaining model," *Econometrica*, vol. 50, no. 1, pp. 97–109, January 1982.

[11] S. Venugopal, R. Buyya, and L. Winton, "A grid service broker for scheduling e-science applications on global data grids," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 6, pp. 685–699, May 2006.

[12] X. Chu, K. Nadiminti, C. Jin, S. Venugopal, and R. Buyya, "Aneka: Next-Generation Enterprise Grid Platform for e-Science and e-Business Applications," in *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing (e-Science 2007), Bangalore, India*. IEEE CS Press, Los Alamitos, CA, USA., Dec. 2007.

[13] T. Abdelzaher, E. Atkins, and K. Shin, "Qos negotiation in real-time systems and its application to automated flight control," *Transactions on Computers*, vol. 49, no. 11, pp. 1170–1183, 2000.

[14] J. Huang, P.-J. Wan, and D.-Z. Du, "Criticality- and qos-based multiresource negotiation and adaptation," *Real-Time Systems*, vol. 15, no. 3, pp. 249–273, Nov. 1998. [Online]. Available: http://dx.doi.org/10.1023/A:1008044430932

[15] M. J. Buco, R. N. Chang, L. Z. Luan, C. Ward, J. L. Wolf, and P. S. Yu, "Utility computing sla management based upon business objectives," *IBM System Journal*, vol. 43, no. 1, pp. 159–178, 2004.

[16] J.Li and R.Yahyapour, "Learning-based negotiation strategies for grid scheduling," in *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006), Singapore*. IEEE CS Press, Los Alamitos, CA, USA, May 2006.

[17] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in *Proceedings of the 7th International Workshop on Quality of Service (IWQoS '99)*. London, UK: IEEE CS Press, Los Alamitos, CA, USA, Mar. 1999.

[18] I. Foster, A. Roy, and V. Sander, "A quality of service architecture that combines resource reservation and application adaptation," in *Proceedings of Eight International Workshop on Quality of Service (IWQoS '00), Pittsburgh, PA, USA*. IEEE CS Press, Los Alamitos, CA, USA, June 2000, pp. 181–188.

[19] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat, "Sharp: an architecture for secure resource peering," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 133–148, 2003.

[20] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing Risk and Reward in a Market-based Task Service," in *Proceedings of the 13th IEEE international Symposium on High Performance Distributed Computing(HPDC-13)*. Honlulu, USA: IEEE CS Press, Los Alamitos, CA, USA, June 2004.

[21] S. Venugopal and R. Buyya, "A Deadline and Budget Constrained Scheduling Algorithm for e-Science Applications on Data Grids," in *Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2005)*, ser. Lecture Notes in Computer Science, vol. 3719. Melbourne, Australia.: Springer-Verlag, Berlin, Germany, Oct. 2005.

[22] M. A. Netto, K. Bubendorfer, and R. Buyya, "Sla-based advance reservations with flexible and adaptive time qos parameters," in *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC 2007), Vienna, Austria.* Springer-Verlag, Berlin, Germany, Sept 2007.

[23] J.Li and R.Yahyapour, "A negotiation model supporting co-allocation for grid scheduling," in *Proc. of 7th IEEE/ACM International Conference on Grid Computing (Grid 2006), Barcelona, Spain*. IEEE CS Press, Los Alamitos, CA, USA, Sept 2006.

[24] A. Mu'alem and D. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 6, pp. 529–543, 2001.