Fine Grained Resource Reservation and Management in Grid Economies

Kris Bubendorfer* and Peter Komisarczuk and Kyle Chard and Ankur Desai
School of Mathematics, Statistics and Computer Science
Victoria University of Wellington,
PO Box 600, Wellington 6001, New Zealand,
Ph: +64 4 463 6484, Fax: +64 4 463 5045
Email: {kris.bubendorfer, peter.komisarczuk}@mcs.vuw.ac.nz

***Abstract*— *Researchers have been actively looking at utilising new models for network based services. For enterprises and third party service providers can enable their infrastructure based on utility computing and next generation Internet technology to create dynamic virtual organisations. Small fixed and mobile, pervasive and ubiquitous devices will also benefit from networked services and computation provided by utility computing providers and the virtual organisations that lease resources from them. Additionally, we believe that it is critical that the virtual organisation and the mobile, pervasive or ubiquitous devices be able to dynamically manipulate their resource specifications when obtaining services and resources from the utility computing enabled network infrastructure. There are a number of research issues that need to be explored and overcome to achieve this potential future scenario, two of these are explored in this paper. Firstly this vision requires a simple, manipulatable, and preferably modular resource specification structure. Secondly the vision requires the provision of a resource utilisation framework that can efficiently enable resource allocation and optimise operational cost.*

Keywords: resource reservation, virtual organisations, utility computing.

I. Introduction

Researchers have been actively looking at utilising new models for network based services based on the provision of overlay networks and utility computing resources. One such model is the Internet Virtual Organisation (iVO), where the iVO creates services on demand utilising resources leased dynamically from Utility Computing Providers (UCP) [1]. The basic UCP model is further extended by including the leasing of communications services, resulting in Utility Computing Communications Providers (UC^{2}P). The UC^{2}P infrastructure could be heavily based on developments in Grid computing.

A currently evolving model of utility computing is based on grid computing [2], and the GRASP [3] project is exploring the commercial grid application paradigm based around the Application Service Provider model. However the UC^{2}P infrastructure, as envisaged, demands the NGG - Next Generation Grid which needs to encompass more resources and provide greater flexibility in terms of mobility, resource allocation and economy based resource allocation than the current Grid implementations. The predominant Grid toolkit is Globus [4], which has come some way to providing an NGG through the Open Grid Service Architecture (OGSA) and enabling web services though the Open Grid Service Infrastructure, Web Service Resource Framework (OGSI, WSRF). However the communication services model requires extension in order to encompass the potential services that could be required in a general purpose UC^{2}P scenario, and its resource allocation does not provide for speedy resource allocation and optimal charging mechanisms, which are required in iVO operations.

Within a large scale distributed systems, such as UC^{2}P systems, efficient negotiation for and allocation of resources plays an important role in the performance of the system. Recently much emphasis has been placed on the computational advantages of large scale grid systems which provide mechanisms to share a wide pool of resources such as computers, software, and peripherals amongst many users and organisations [5].

Describing the resources offered by a UC^{2}P distributed system and those requested by client applications is vital in ensuring efficient allocation of the resources, providing quality of service (QoS) guarantees and therefore maximising the overall performance of the system. However, providing such descriptions is only part of the task – quickly and efficiently matching requested resources to distributed system components is also critical to the efficiency of the system. Another potential reason for low system performance in large-scale distributed systems is related to under-utilisation of negotiated resources. The negotiation process introduces latency and uncertainty in the system. Applications apply for the resources they require, yet by the time the application actually uses the resources, considerable time may have elapsed. In effect, the host must tentatively reserve the resources that are under negotiation for the entire negotiation process, even if the negotiation later terminates without agreement. This is less of a problem within the traditional Grid model, of large long term computations, where resources are acquired in advance. However, when generalising the Grid model as a basis for the deployment of virtual organisations operating within computing utilities, we must consider smaller, more dynamic negotiations that would support mobile devices and the provision of on demand services. In this context the utilisation and negotiation latency of resources will become a significant performance bottleneck. As an additional factor, we believe that it is critical that for mobile, pervasive or

*Presenting Author.

ubiquitous devices to be able to dynamically manipulate their resource specifications when obtaining services from the UC^{2}P network. This requires a simple, manipulatable, and preferably modular resource specification structure.

This paper falls naturally into two major parts. The first concenrates on presenting an efficient resource specification mechanism called the Resource Description Graph (RDG). The RDG is a single rooted directed acyclic graph (DAG) that is passed between hosts in a simple textual format. The RDG encodes resource requirements, constraints and availabilities thorough a sequence of edges. An application can provide an RDG, to a host or group of hosts, that represents a combination of resources required for execution. A RDG can be efficiently formed and manipulated by all potential entities within a UC^{2}P system. With the ability to describe resources and negotiate combinatorical packages of resources established, this paper then introduces the coallocation and oversubscribtion resource allocation (CORA) Architecture. The CORA architecture utilises a combination of existing and new techniques to provide the infrastructure in which to carry out distributed economic resource exchange and management in Grid style economies.

CORA, along with the RDG was developed within the Nomad [6] middleware system. Nomad is a mobile agent system, that utilises a economic management model as a basis for an open system. Many of the lessons we learnt developing the Nomad system are applicable to the wider Grid community. Like the Grid, Nomad is a distributed computational system, which consists of a collection of loosely coupled cooperating virtual machines (depots) that are capable of hosting distributed applications. Many of the aims of Nomad are shared by the CONOISE [7] project, however our approach and focus differ. Nomad utilises an economic resource management model, using an auction mechanism for optimal resource management and provision of lowest priced resources to applications. The representation of resources is critical to the versatility and optimality of the auction process and without such a mechanism there cannot be an efficient auction mechanism. Thus some of the key criteria behind the RDG design are versatility and enabling low latency, high throughput performance.

II. Resource Description

The RDG specification consists of a sequence of edges terminating at a vertex marked as an *accept state*. There may be many resource combinations leading to a single accept state representing a set of compromises or alternatives from the view of the application requesting the resources. The RDG has several advantages; in particular expressiveness and versatility; for example, it can be decomposed into subgraphs as processes split or migrate between hosts and can also express mobile objects aggregation.

The majority of recent related resource specification work is in the area of grid systems. Two resource specification schemes are mentioned here specifically: the Condor classAd language and the Globus Resource Specification Language (RSL).

Condor uses the classified advertisement language (classAd) to describe jobs, workstations, and other resources [8]. The classAd language is a symmetric, semi structured declarative language designed to allow easy matching of resources and requests in order to correctly execute jobs on the Grid. A classAd is a mapping from attribute names to expressions and two classAds match if each has attribute requirements that evaluate as true. Similarly to the RDG, customer agents use the classAd to request resources and a machine (a resource) can use a classAd to advertise its resources. Matching is performed by the Condor central manager to determine the compatibility of jobs and workstations. The classAd language is available as a stand alone package for other applications and currently it has a native and an XML form.

Currently Globus (the de facto standard for Grid computing) specifies an XML based language called the Resource Specification Language (RSL) to describe resource requirements [9]. The RSL was changed from the Globus Toolkit 2 to an XML based language in Globus Toolkit 3 to ensure optimal portability. The RSL is used by Brokers and Globus Resource Allocation Manager(GRAM), Globus Architecture for Reservation and Allocation (GARA) [10] is a mechanism designed to enable end-to-end QoS guarantees supporting resource discovery and dynamic (a reservation for some future time) or immediate reservation. GARA extended the generic Globus resource management architecture by introducing a generic resource object, which encompasses network flows, memory blocks, disk blocks, and other entities. XML based resource specifications are encumbered by the processing overheads incurred by parsing the XML resource specifications. This is a minor inconvenience when considering the large relatively static scientific applications that were envisaged by the designers.

All three resource specification systems, classAd, Globus RSL v3 and the Nomad RDG are all sufficiently descriptive and flexible to be used by systems other than those for which they were developed.

III. Resource Description Graph

From the viewpoint of an application, an RDG describes a group of tasks (encoded as accept states), each with a set of resource combinations that represent resources required to satisfy each task. The edges of the RDG encode the resources and the vertices the state. There may be alternative resource allocations for each task — these represent viable alternative sets of resources that can be utilized to perform the required application function. A path from the root to an accept state is called a sentence, and an accept state may therefore be traversed via multiple sentences.

Anytime that a set of resources needs to allocated over a set of providers we face an *NP*-complete optimization problem [11]. By constraining the combinations of resources within the RDG we minimize the *NP*-complete Combinatorial Allocation Problem (CAP)[1] within a system. In an economic

[1]To allocate items in order to maximize total utility

resource allocation system, such as NOMAD, each sentence is valued, and the preferred (lowest cost sentence) for each accept state is chosen.

The resources available at a host are likewise encoded in an RDG, which we have termed a *Resource Profile*. A resource profile does not utilise accept states — but is instead used to record the current availability of resources and any future resource commitments at a host. The point here is, that certain resources are only available in set combinations — but with variation in their weightings, e.g. both disk space and disk IO are required to store data, the exact weighting between the two resources depends on the application. The RDG is a good fit for describing the resource constraints of both applications and hosts - a handy situation, as resource allocation is essentially an exercise in constraint satisfaction.

A. RDG Structure

Figure 1 shows part of a simple content distribution virtual organisation RDG. The iVO RDG specifies a requirement for a number of servers with Constant CPU rate (CCU), memory and disk to perform, say, video stream serving.

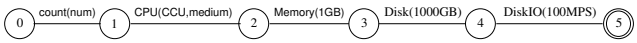

Fig. 1. Example iVO RDG, double circles indicate an accept state

It is possible that the iVO could also utilise a slightly different set of resources to provide its services. Figure 2 shows an RDG that encodes two different sets of resources that can be used interchangeably by the iVO. A resource provider, may satisfy either the resource sentence (a) {0,1,2,4,5,6} or (b) {0,1,3,4,5,6}. The alternative sentence (b), utilizes a Variable CPU rate (VCU), and so requires additional memory for caching. The principle here, is that an accept state relates to a service, and not the resource sentences by which that service can be provided.

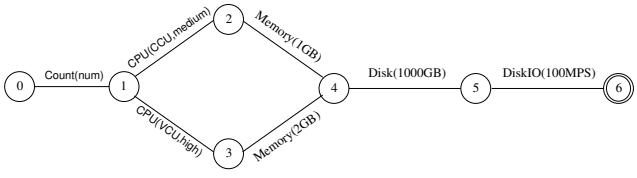

Fig. 2. Alternative iVO RDG, double circles indicate an accept state

Different structures in the graph map to logical combinations of AND (sequences), OR (Branches to separate accept states) and XOR (branches that terminate in the same accept state). If an edge belongs to two different sentences that terminate in different accept states, then the host must provide multiple units of that resource if it wishes to host both tasks. However, if the two sentences terminate in the same accept state, as in figure 2, then only one unit of each resource is required.

B. Edge Expressions

Classes of service such as those used in networking (e.g. ATM CBR, VBR, ABR, UBR[2]) are used to represent the expected behaviour of a resource consumer. Processes do not usually require constant CPU, Memory, and I/O. For this reason classes have been defined for CPU, memory, disk, as well as bandwidth. The defined classes for CPU are Constant, Variable, and Available CPU Utilisation (CCU, VCU, ACU). The class of service is encoded in the value parameter of the edge, for example, constant CPU bursting of 200 MFLOPS is encoded as {CPU(CCU, 200)} or can represent non specific values, such as high, medium or low CPU power.

C. Graph Modularity

The structure of an RDG makes composition and decomposition relatively straightforward as subgraphs may be removed and manipulated as required. An example would be an application migrating some part of its functionality to another host. The application can remove the associated part of its RDG to create a sub RDG to send with the migrating process. Similarity if the process returns, the RDGs can be rejoined.

Another advantage is that common shapes of resource requirements (or *modules*) can be created and used by applications to quickly characterise a common problem. For example when expressing a simple computation CPU, Memory, and disk will all be required. The common modules can be combined to form a complete RDG for a given application at a reduced computational cost.

D. Resource Matching — Modules

The ease of joining and separating graph components makes the use of modules a viable option for improving matching efficiency. The biggest limitation of the RDG is the potential for inefficiency when graphs become large. One solution to this is to break large graphs down into smaller subgraphs. We have identified three orthogonal resource categories into which a host's resource profiles are decomposed and then compared against an application's RDG. **Computational** resources may be categorised as performance or availability resources. Performance resources are measurable like CPU, exchanged disk I/O and exchanged memory I/O, these resources are generally related to a required application or mobile object demand. Availability resources are the amount of a resource required for execution, e.g. an amount of memory or number of licenses. **Network** resources include parameters such as, bandwidth, jitter, loss rate and locality. Other networking aspects can also be considered, such as firewalls, provision of Application Level Gateways, access to certain network bearers, network load sharing features and redundancy etc. **Existential** resources include the availability of J2ME, CODECs, particular software libraries, or licenses. Hardware existential resources include

[2]Asynchronous Transfer Mode, Constant Bit Rate, Variable Bit Rate, Available Bit Rate and Unspecified Bit Rate

properties such as CPU architecture (e.g. SMP - symmetric multiprocessor architecture), printers and other peripherals.

E. Experimental Results

The two key variables that can be expected to impact negatively on performance when matching RDGs are size and complexity. The following result is taken from [12] and illustrates the performance of the RDG with respect to path length. Increased path length is a direct result of a corresponding increase in RDG complexity. Figure 3 shows the relationship between path length and computational time. The first algorithm called *path comparison* (PC) extracts each path (or sentence) and then compares the individual paths. This simple algorithm results in multiple comparisons against certain resources. A solution to this was to utilise the modules outlined in section III-D — giving the *module comparison* (MC) algorithm. Details of these algorithms can be found in [13]. This graph illustrates the improvement of the MC algorithm over the PC, and provides a feel for the performance of RDG matching.

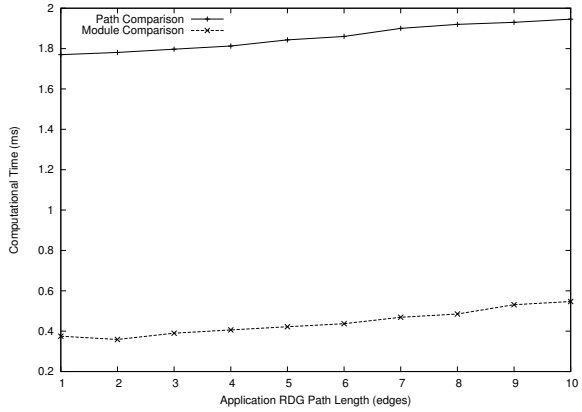

Fig. 3. Application path length vs computational time

The results exhibit a small linear increase with length, the dominant cost is that of computing the resource profile paths. Additional results are published in [12]. These results show that we can achieve computational times for average sizes of RDGs and Resource Profiles of less than 1ms. This computational time is acceptable in most real situations, take for example a wide area distributed system — the latency required to transport the RDGs would far exceed the computational time required to evaluate them.

IV. CORA

Within larger collaborative distributed systems the efficient description of resources plays an critical role in the performance of the system. However, distributed resource allocations can also result in lower resource utilisation owing to the delay involved in negotiation, the delay in taking up the agreed resources, and the tentative allocation of resources during the negotiation process. Resource oversubscription allows for better utilisation of resources in distributed systems, however, this must be done in a controlled way to ensure that the resulting allocations can be fulfilled. In the CORA (Coallocation, Oversubscribing Resource Allocation) architecture, resource providers (hosts) delegate all or part of their resources to a selected broking agent(s) that then negotiates on their behalf.

A. Economic Resource Management

With the wide-spread use of large-scale, distributed platforms such as Globus Grids [14] , and Planet-Lab [15], it is important to improve the performance of the system as a whole. For example, when hosts are solely responsible for managing and allocating their own resources, it results in efficient local resource allocations. However, it will not necessarily result in globally efficient resource allocations over all hosts within a system.

For many excellent reasons auctions are touted as an efficient solution to the problem of distributed resource allocation in both economic [6], [16] and noneconomic [17] resource allocation systems. However, while on one hand auction protocols are an ideal mechanism for determining the optimal allocation of resources, and for determining the market price of a good, auctions compound the problem of resource utilisation. In particular an auction generally has a single winner, and multiple m losers. While the winner gains the eventual contract[3], there is no such compensation for the m losers of the auction process, and any resources r put aside during the auction will decrease the net utilisation of the system by mr. In addition, the length of time an English auction is unbounded, Dutch auctions are bounded by the bid decrement rate, while sealed bid auctions are of fixed duration. The duration of an auction protocol limits the application of such resource allocation systems to larger longer lived entities within the system. This is reasonable considering the inherent cost of remote execution, shorter lived resource demands must remain the responsibility of the local host and scheduler.

The major goal of the CORA architecture is to address the multiplicative decrease in utilisation within an auction based resource allocation architecture. The techniques the we have identified and adopted within CORA as significant steps towards meeting this goal are *coallocation* [4] and *oversubscription* [18]. The techniques that we have developed within CORA to move further towards the goal are *just-in-time allocations* and a *progressive contract structure*. All of these techniques require some additional entities within the system, with a greater resource horizon[4] than individual hosts, yet with a smaller scope than say, a system scheduler. We introduce within CORA the idea of broking agents, to which hosts delegate responsibly for resource negotiation. The broking agents then interact with a Marketplace (equivalent to a Globus GARA) that manages resource allocations over administrative boundaries. In a little more detail:

[3]The result of a resource negotiation is a contract.

[4]It is very difficult to achieve oversubscription and coallocation within the resource scope of an individual machine. A greater view of the resource allocation landscape is called for.

Coallocation Resource allocation often requires making allocations in a coordinated fashion across virtual organisation boundaries. This form of allocation is known within the Grid community as coallocation. CORA introduces the broking agent role into the Nomad architecture, where the broking agent can act for a group of resource providers and allocate resources based on evaluation of allocation constraints over an ad-hoc resource group.

Oversubscription Controlled oversubscription of resources improves resource efficiency and availability when rights to allocated resources can be lost or left idle. CORA introduces an oversubscription mechanism by distinguishing between the granting of soft-state and hard-state resource rights to applications.

Flexible Contract Structures A progression from soft to hard state contracts as the system becomes more certain about the set of resources being allocated. That is, contracts harden as they progress through the various stages of negotiation.

Just in Time Allocation Introducing the caching of availability knowledge for an ad-hoc group of resource providers allows the Just in Time allocation of resource allocation contracts to resource providers thereby reducing the latency that is inherent in the auctioning process.

B. CORA Overview

Figure 4 gives an overview of the CORA architecture within Nomad. The featured CORA components are the Broking Agents, the Reputation Service, and the Agent Finder. The Nomad components are included for completeness, but are not part of the CORA system per se. A depot is simply a resource provider, that hosts application components, a federation is a collection of depots defined by ownership or administrative domain, and the marketplace is a interfederation market for resources. The agent pool is a collection of broking agents whose membership is defined by registration within the agent finder. There is no intent to describe ownership, administration or physical location of the broking agents.

C. Resource Negotiation

Many systems utilise auction protocols, without regard of the shortcomings of the protocol. The Vickrey auction protocol has long been a favourite in the automation of computing economies for it inherent properties of a single, truthful bid. However, the Vickrey and all other auction protocols, have known problems that limit the applicability of the protocols in practice. Some of these problems are bidder collusion, the lying auctioneer, lying non-private value auctions, lower revenue than alternative protocols, and wasteful counter-speculation. A number of these problems were documented in the original research work [19]. In addition to this, several problems related to untruthful bidding and counter-speculation in presence of local uncertainty in computational multi-agent systems were discovered by Sandholm [20].

With the delegation of resource allocation from resource providers to mid-level resource brokers, we have discovered that a new form of bidder collusion becomes inevitable with

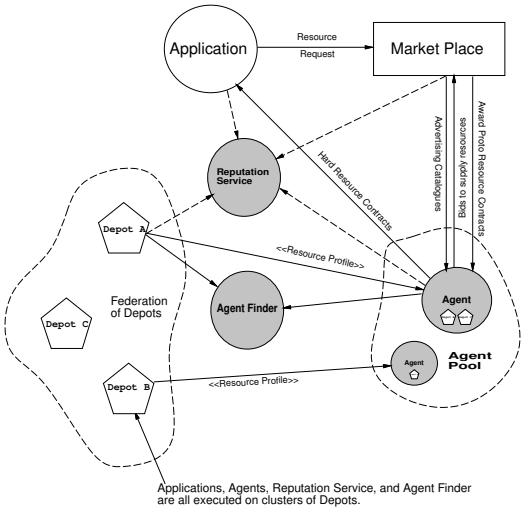

Fig. 4. *CORA architecture integrated within Nomad. CORA introduces the entities shaded in gray.*

delegation and multiple bidding when using the Vickrey auction protocol. We have named this problem **self enforced bidder collusion via delegation**. For the following argument, assume that a higher bid is a better bid, as in a conventional auction.

Consider a system in which there are broking agents. Each agent represents a number of resource providers that have delegated the allocation of their resources. In an auction it makes sense for the broking agent to determine the bids for all the providers that it represents and then only forward the highest two bids. All lower bids will not succeed and therefore would result in pointless network traffic. We need the second bid for the second price Vickrey protocol - which will otherwise either abandon the auction, or determine the price using the wining bid. The obvious ploy for the agent to increase its revenue, is to fabricate the second bid to the lowest possible value. Now if only one broking agent bids, the second price mechanism will result in the seller receiving only the false second price.

All auction protocols are susceptible to bidder collusion, however, only the English auction protocol and the Vickrey auction protocol self-enforce any collusive agreement [20]. The problem with self enforced bidder collusion via delegation only applies to the Vickrey protocol due to the need for a second price. The result is that the dominant strategy of truthful bidding in Vickrey auctions becomes: *make the first bid truthfully, and make the second bid as low as possible*. In situations where only one agent bids on one item — price paid is that of the falsified second bid. This change in dominant strategy also has the effect of reintroducing counterspeculation in coallocative systems — due to the need to determine the probability of the success of each individual bid, within a coallocative auction.

There are two possible partial solutions; only allow one bid per agent (reducing the number of bids available in coallocation situations), or ignore multiple bids from the winner when

determining the second price (only applies when there is only one bidding agent). Neither of these is a complete solution to this problem, as both somewhat reduce the optimality of the auction process.

The CORA architecture is not tied to any particular auction protocol, however, the prototype implementation does use the Vickrey auction protocol. A number of careful design choices in the implementation avoid the limitations of the protocol with only a small reduction in flexibility and optimality. For example, the auctioneer (Market) is a trusted system service, and all auctions limited to being private value auctions (no reselling of contracts) [6]. The best choice of auction protocol depends on the context in which the system is ultimately deployed.

D. CORA Resource Contracts

CORA introduces the notion of proto resource contracts (PRC) and hard resource contracts (HRC). A PRC represents soft-state resource rights and is generated by the Marketplace, and returned to the requesting application *and* the winning Agent as the initial result of a negotiation. Being soft-state, a PRC does not guarantee that resources are available, but rather that they may be available upon redemption. For this level of guarantee the broking agent must first *harden* the PRC into a HRC, after considering the current resource situation. The key idea is that an Agent assigns resources from its pool of depots to satisfy a given PRC. Agent generates an HRC if and only if it is able to find a depot for the resources in the PRC. The resources on which the original bids were based may no longer be available, or a better choice may have since become available. In these cases, the depot listed as providing the resources in the PRC may be substituted by another depot in the HRC.

This two level approach is inherently sensible, as a top level allocation entity such as a Marketplace, can not and should not attempt to provide resource guarantees when considering the inherent latency in negotiation. Such approaches would not scale. The primary advantage of the PRC stage is that the negotiation can be cheaply aborted at this early stage if the available resources within the system suddenly change.

Consider the situation if HRC contracts were issued by the marketplace instead of PRC. To prevent rejections of contracts on redemption at the depot, more resources would have to be reserved (by both winning and losing bidders), decreasing overall utilisation. If on the other-hand, we indulged in the same degree of oversubscription — then more contracts would be unsatisfiable on redemption, causing more serious and immediate difficulties to the applications.

E. Two Phase Contracts and Commit

Essentially, the resource contracts work on the principle of a two-phase commit mechanism: a technique based on well-known two-phase commit protocol [21], [22] to solve the distributed commit problem of maintaining consistency of distributed data. The key idea of this two-phase commit protocol is to maintain the atomicity of transactions applied on distributed data, that is, either all participants commit or all participants abort. Of course, the goal of this mechanism is different in the context of CORA resource contracts. Using this mechanism, we intend to ensure that an application is not faced with a situation in which, the contracts presented by it is refused by a depot(s) unless in exceptional circumstances. We believe that it is in an application's best interest to re-initiate an auction rather than faced with a rejection of contracts at redemption time.

One of the positive side effects of this two phase contract mechanism, is that it neatly caters for both single and coallocative negotiations. Figure 5 and Figure 6 are the flow charts of first and second phase of two-phase commit mechanism. Rectangles represent actions and solid circles represent the states for entities such as Market, Agent, and Application. Dashed circles represent the transition from one phase to another. Dashed lines represents message exchanges. Solid lines represents state transitions. The Marketplace acts as a coordinator and broking agents act as participants. Marketplace's job is to find out if all winning broking agents are in a position to honour the contracts or not. Clearly for a noncoallocative negotiations, there is only one broking agent, so the process is completed quickly. With coallocative negotiations, the mechanism ensures that all the sets of coallocated resources are available prior to any PRCs being hardened into HRC contracts.

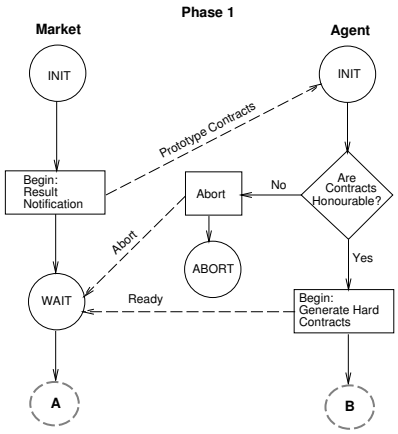

Fig. 5. *A two-phase commit mechanism for resource contracts – Phase 1*

The first phase, as shown in Figure 5, begins with the Marketplace sending PRCs to each winning broking agent and waiting until a positive `ready` response from each Agent is received. When a broking agent receives a PRC, it checks if it can honour it. If Agent can honour the contracts, then it sends a `ready` message the Marketplace and enters the `READY` state. If Agent cannot honour the contracts, then it sends an `abort` message to the Marketplace and enters the `ABORT` state.

The second phase is shown in Figure 6. If Marketplace receives any `abort` messages, it concludes that generation of hard contracts will fail for some broking agent(s) and aborts. Otherwise, if there are no `abort` messages the Marketplace sends `ready` messages to all participants, sends the PRC

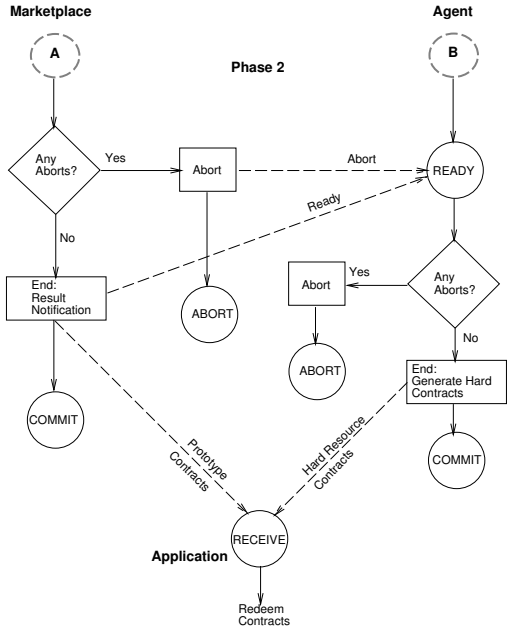

Fig. 6. *A two-phase commit mechanism for resource contracts – Phase 2*

contracts to application, and enters the `COMMIT` state. Each broking agent, upon receiving its `ready` message, generates the HRCs.

F. CORA Coallocation

Coallocation is a technique of simultaneously allocating resources in predetermined capacities over an ad-hoc group of resource providers. This technique is widely used in Grid computing paradigm and several recent research efforts have taken various approaches to solve this [23]–[26]. Coallocation is highly desirable for many applications that demand an adequate level of QoS and parallelism such as content distribution in multimedia applications and scientific applications.

In CORA, Agents can allocate resources over an ad-hoc group of depots for applications requiring coallocation services. In order to distinguish ordinary allocation requests from coallocation requests, application needs to describe its resources accordingly in a RDG when placing an auction. For this purpose we have adopted the *count* parameter from RSL [27] as used for coallocation in computational grids [24] and utilise this as an edge expression in the RDG.

A coallocative negotiation is treated as a normal by the Marketplace, except that the best *count* bids are used to generate PRCs and the two-phase mechanism from section IV-E is used to ensure that all or none of the resource allocations are made.

G. Just-in-time Allocation of Resources

Depots periodically communicate their resource profiles to the broking agents, which then allocate resources based on those profiles. This is effectively caching of availability knowledge for an ad-hoc group of depots, and allows an Agent to make allocation decisions just before sending the actual contracts to application. That is, in the step between PRC and HRC contracts. This technique of making allocation decisions at the last moment before hardening of contracts is effectively just-in-time allocation.

Consider the situation in which the PRC received by a broking agent is for depot A. However, during the interval between when the bids were generated and the time at which the PRCs where generated by the Marketplace, depot A's resource availability changes. This could reflect a change in the set of resources delegated to the broking agent, be a result of oversubscription, or failure. When the broking agent receives the PRC, depot A is no longer able to provide the resources. If the broking agent can still satisfy the contract utilising resources from another depot, then it may substitute the depot for another when hardening the PRC into a HRC. This overcomes many of the problems introduced by the latency in negotiation.

H. Oversubscription of Resources

CORA broking agents use the controlled oversubscription of resources to improve the resource utilisation depots. broking agents use the same resources in multiple bids to increase the chance of winning an auction, relying on the low probability of winning all such auctions.

This does not alter the valuation of the bids, but raises the spectre of contracts being rejected through a lack of resources. Obviously the degree of oversubscription and the probability of winning an auction have a direct bearing on the likelihood of rejection. These factors are a combination of agent policy and market environment. As discussed in Section IV-E, the resources in a PRC issued by the Marketplace are not guaranteed until the second phase of the two-phase commit mechanism, when the agent hardens the contract. In the worst case, if the agent can't find sufficient resources, then application will have to initiate a new resource auction.

I. The Agent Finder Service

The agent finder service of CORA provides the ability to query for and locate a suitable broking agent. The agent finder maintains a record of all broking agents, any specialities that they have and a metric of their recent performance (success). In the Nomad implementation, the agent finder is a subfunction of the Marketplace, although this need not be the case in other implementations.

J. The Reputation Service

A broking agent is a largely autonomous entity, it is not a system provided service in Nomad. In these cases, depots need some form of feedback system to determine if they wish to deal with a particular broking agent. Therefore, the concept of Reputation is introduced to check whether entities in a system is providing their services as per the expectations. The reputation service of CORA is an authority that assigns and manages the *reputation* of entities in a system. Any entity in the system can *lodge* complaints against other entities using reputation service. For example, the Marketplace can

lodge a complaint against a broking agent that is causing excessive delays in hardening contracts. A depot may complain that a broking agent is ignoring its profile and issuing an excessive number of HRCs, or ignoring its pricing policies. Such complaints reduce the *reputation* of a broking agent.

K. CORA Status

CORA exists as a functioning prototype. We are currently extending the mechanisms to cope robustly with failure and adding more scheduling and bidding intelligence to the broking agents. We are also investigating the use of cryptographic techniques that will permit entities to prove that they have acted correctly in a transaction - which will then provide a better basis for recording reputation within the system.

V. Conclusions

This paper has focused on the extension of grid technology to form a platform suitable for virtual organisations that dynamically negotiate resource contracts from utility computing providers. This vision of the future of grid computation requires a more dynamic approach to resource exchange and management with smaller more frequent negotiations by smaller less powerful, but more numerous entities.

We have explored and developed in this paper, two critical components needed for such a paradigm shift. The first component is the resource description graph (RDG). The RDG is an modular and efficient resource specification mechanism, flexible enough for use in supercomputers through to computationally limited devices such as PDAs. Our results have demonstrated that matching resources encoded within an RDG is quick and poses little overhead on the resource negotiation system. The CORA architecture addresses the problem of low resource utilisation due to latency in the distributed negotiation of resources. CORA utilises oversubscription and just in time allocations to reduce tentative resource reservations, and a progression from soft to hard contractual agreements as the certainty of the resource allocation increases. The two phase contractual commit process neatly caters for both single and coallocative negotiations.

References

[1] P. Komisarczuk, K. Bubendorfer, and K. Chard, "Enabling Virtual Organisations in Mobile Networks," *IEE 3G2004 Conference, London, October 18-20 2004*, 2004.

[2] P. Eerola, B. Konya, O. Smirnova, T. Ekelof, M. Ellert, J. R. Hansen, J. L. Neilsen, A. Waananen, A. Konstantantinov, and F. Ould-Saada, "Building a Production Grid in Scandinavia," *IEEE Internet Computing*, vol. 7, no. 4, pp. 27–35, July-August 2003.

[3] T. Dimitrakos, D. M. Randal, F. Yuan, M. Gaeta, G. Laria, P. Ritrovato, B. Serhan, S. Wesner, and K. Wulf, "An Emerging Architecture Enabling Grid Based Application Service Provision," in *Seventh International Enterprise Distributed Object Computing Conference (EDOC'03)*, Brisbane, Queensland, Australia, September 2003, pp. 240–251.

[4] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Lecture Notes in Computer Science*, vol. 2150, 2001.

[5] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan and Kaufmann, 1999.

[6] K. Bubendorfer, "Nomad: Towards an architecture for mobility in large scale distributed systems," Ph.D. dissertation, Victoria University of Wellington, New Zealand, 2001.

[7] T. J. Norman, A. Preece, S. Chalmers, N. R. Jennings, M. Luck, V. D. Dang, T. D. Nguyen, V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian, "Conoise: Agent-based formation of virtual organisations," in *Proceedings of AI2003, the Twentythird SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, 2003, pp. 353–366.

[8] M. Solomon, "The ClassAd Language Reference Manual Version 2.1," October 2003, last checked 1/9/04. [Online]. Available: http://www.cs.wisc.edu/condor/classad/refman/

[9] Globus, "The Globus Resource Specification Language RSL v1.0. ," http://www.globus.org/gram/rsl_spec1.html, 2000, last checked 1/9/04.

[10] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation," in *Proceedings of the International Workshop on Quality of Service*, 1999.

[11] K. Bubendorfer, "NOMAD: Towards an Architecture for Mobility in Large Scale Distributed Systems," Ph.D. dissertation, Victoria University of Wellington, December 2001.

[12] K. Bubendorfer, P. Komisarczuk, and K. Chard, "Efficient Dynamic Resource Specifications," in *(To appear) in Proceedings of MDM 2005, 6th International Conference on Mobile Data Management*, Ayia Napa, Cyprus, May 2005.

[13] K. Chard, "Efficient resource descriptions," 2004.

[14] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 2, pp. 115–128, 1997.

[15] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," in *Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, October 2002.

[16] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger3, "Economic models for resource management and scheduling in Grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1507–1542, 2002.

[17] T. W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard, "Enterprise: A Market-like Task Scheduler for Distributed Computing Environments," in *The Ecology of Computation*, H. B.A, Ed. Elsevier Science Publishers (North-Holland), 1988, pp. 177–205.

[18] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat, "Sharp: an architecture for secure resource peering," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM Press, 2003, pp. 133–148.

[19] W. Vickrey, "Counterspeculation, Auctions, and Competitive Sealed Tenders," *The Journal of Finance*, vol. 16, no. 1, pp. 8–37, March 1961.

[20] T. Sandholm, "Limitations of Vickrey Auction in Computational Multiagent Systems," in *In Proceedings of Second International Conference on Multiagent Systems (ICMAS-96), Kyoto, Japan*, December 1996, pp. 299–306.

[21] A. L. Risnes, "Asynchronous Lock Distribution in the New File Repository," Master's thesis, Department of Computer Science, University of Tromsø, Norway, 2000.

[22] A. S. Tanenbaum and M. V. Steen, *Distributed Systems - Principles and Practice*. Prentice Hall, New Jersey, USA, 2002, ch. Chapter 7: Fault Tolerance, pp. 393–398.

[23] S. Anand, S. Yoginath, G. von Laszewski, B. Alunkal, and X.-H. Sun, "Flow-based Multistage Co-allocation Service," in *The 2003 International Conference on Communications in Computing*, Las Vegas, Nevada, USA, June 2003.

[24] K. C. andI. Foster and C. Kesselman, "Resource Co-Allocation in Computational Grids," in *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, 1999, pp. 219–228.

[25] F. Azzedin and M. Maheswaran, "A Co-allocation Mechanism for Multimedia Enabled Grids," in *Proceedings of 13th IASTED International Conference on Parallel and Distributed Computing Systems (PDCS '01)*, August 2001, pp. 27–32.

[26] F. Azzedin, M. Maheswaran, and N. Arnason, "A Synchronous Co-Allocation Mechanism for Grid Computing Systems," *Cluster Computing*, vol. 7, no. 1, pp. 39–49, 2004.

[27] K. Czajkowski, I. T. Foster, N. T. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 1998, pp. 62–82.