

OCEAN: The Open Computation Exchange and Arbitration Network, A Market Approach to Meta computing

Pradeep Padala, Cyrus Harrison, Nicholas Pelfort,
Erwin Jansen, Michael P Frank and Chaitanya Chokkareddy
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, Florida 32611–6120

Email: {ppadala, cdh, npelfort, ejansen, mpf, cchokkar}@cise.ufl.edu

In the Proceedings of the International Symposium on Parallel and Distributed Computing (ISPDC'03)

Abstract—Rapid advancements in processor and networking technologies have led to the evolution of cluster and grid computing frameworks. These high-performance computing environments exploit geographically distributed, diverse resources with the goal of providing efficient computing solutions to all kinds of parallel and distributed applications. OCEAN (Open Computation Exchange and Arbitration Network) provides a scalable market-based infrastructure to such meta-computing frameworks. OCEAN aims to build a marketplace where resources like CPU time, associated memory usage and network bandwidth are the traded commodities. This paper explains the technical challenges faced in the design of OCEAN and discusses our proposed solution. To facilitate finding suitable resources for buyers, we developed efficient matching and evolution protocols for the peer-to-peer matching network. The architecture and various components of OCEAN are described in detail. We implemented OCEAN on Java and .NET platforms and describe results from our preliminary experiments.

Index Terms—Distributed computing, Market-oriented computing, Meta computing, Grid computing, Peer-to-peer, Resource Management.

I. INTRODUCTION

At no other time in human history have so many people had access to powerful computing resources. Proliferation of workstation-class processors and growth of high-speed networks has made Internet computing pervasive worldwide. And yet, it has been observed by numerous sources [1][2][3][4][5], many of these resources lie idle for long periods of time. Legions of computers online are not involved in any compute-intensive tasks, but only tasks like word processing and browsing the Internet, which consume very little computing power. The total computing power in many organizations may often be severely under-utilized, especially outside of peak business hours. Conversely, there are many individuals and organizations that have intense computations to perform, but only have access to limited and restricted resources that are available to execute them. Scientific applications in domains like High Energy Physics, Bioinformatics, Medical Image Processing and Earth Observations often require massive amounts of computation and storage (sometimes of the order of petabytes).

This overwhelming disparity in resource utilization induced the development of cluster[6] and grid computing[7] paradigms. The key to these high-performance computational frameworks is effective management and exploitation of all available computing resources. These infrastructures, often known as meta-computing systems[8], transparently integrate geographically distributed resources to provide coordinated resource sharing. The concept of resource sharing is not simply restricted to file sharing, but rather direct access to computers, software, data, and other resources that belong to multiple institutions and organizations. Hence the need arises for collaborative problem solving and resource-brokering strategies. The computing world can potentially be revolutionized if systems can transparently buy, sell, and use remote computing resources via the Internet. The vision is to greatly increase the overall efficiency of the world's utilization of computing resources, thereby leading to increased productivity.

Achieving this goal is non-trivial due to factors such as resource heterogeneity, cooperation with heterogeneous platforms, distributed ownership with different administrative policies and priorities, wide geographic distribution, and varying conditions like traffic, reliability, availability. Different programming and communication standards, network mechanisms and their associated compatibility issues, present another set of problems. However, these obstacles can be overcome if the concept of resource sharing can be extended to economically beneficial strategies.

OCEAN (Open Computation Exchange and Auctioning Network) provides software infrastructure to support automated commercial buying and selling of dynamic distributed computing resources over the Internet. OCEAN aims to build a marketplace where resources like CPU time, associated memory usage and network bandwidth are the traded commodities. The major components of such a market are the users, the computational resources associated with them, and the underlying market mechanisms that facilitate the trade.

The goals of OCEAN are twofold: (1) Anyone who has underutilized computational resources should be able to easily deploy OCEAN servers which can run other people's computing tasks for profit, and (2) Any user, with a credit card number

(or other means of automated payment), should be easily able to buy resources for his distributed or parallel applications.

In this paper, we explore the technical challenges involved in providing a market-oriented distributed computing infrastructure. First, we provide a motivating example of usage of OCEAN. Then, we review existing distributed computing projects, their mechanisms and how they compare with OCEAN. Next, we describe the architecture of OCEAN. We conclude with details of implementation and experimental results.

II. A MOTIVATING EXAMPLE

In this section, we see an example usage of OCEAN for distributed computing. The key players in OCEAN market are sellers and buyers. The buyers typically have a computation that needs to be performed, while the sellers have access to idle resources that can execute the computation in question. OCEAN's job is to find a match for buyer's sources and provide a list of competing sellers to the buyer. The buyer can then negotiate with various sellers and command OCEAN to execute his computation on the seller machine. Let's see a scenario of how this works.

- A scientist(the buyer), who wants to conduct a High Energy Physics experiment, is willing to pay \$30 for ten pentium 1GHz processor machines with Globus installed for one day.
- Various organizations or users wishing to sell their resources instruct OCEAN to create a seller resource document and wait for incoming requests.
- The buyer uses an OCEAN GUI to create an XML document describing his requirements and submits the request to OCEAN.
- OCEAN's matching network sends the document over the network using efficient protocols (described later) and returns the results to the waiting buyer node.
- The buyer can now choose to auto-negotiate using OCEAN's negotiation protocols or manually select potential seller(s).
- A contract is made with the seller and the buyer requests OCEAN to execute her scientific application. OCEAN spawns required jobs on the remote site and keeps track of them.
- The buyer can monitor the job using OCEAN's GUI. After the computation is completed the results are returned to the buyer.
- OCEAN bills the buyer and credits the seller through a financial transaction to the Central Accounting Server, which maintains financial information.

Note that, the users of the system do not have to be human; in many instances, a "user" is actually a programmed agent acting on behalf of a human user. Figure 1 shows a high level diagram of OCEAN usage.

III. PREVIOUS WORK

Distributed computing has been an active field of research for over two decades. Systems like Amoeba[9] and Sprite[10]

had great impact on distributed programming paradigm. Recently, there has been tremendous research activity in grid computing. The Grid[7], as it is known, enables the sharing, selection, and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems, data sources and specialized devices owned by different organizations administered with different policies.

In the last few years, a number of exciting projects like Globus[11], Legion[12] and UNICORE[13] have developed the software infrastructure and protocols needed for grid computing. Various distributed computing issues have been solved using these tools and libraries. These tools have been very successful in providing high-performance distributed computing for scientific projects like GriPhyN[14](Grid Physics Network) and iVDgL[15] (International Virtual Data Grid Laboratory). But, These projects fall short of providing a framework for trading (buying and selling) of resources.

On the other hand, economic models have been applied to computing in various ways. In 1968, Sutherland demonstrated how auction methods can be used to allocate time to users on the PDP-1 Computer in the Aiken Computation Laboratory at Harvard University[16]. But Sutherland's work failed to address the problem in distributed systems. Drexler and Miller approached this problem in their paper[17], where they describe auction mechanisms for allocating distributed resources. This paper made two important contributions to computational market. It provided market based mechanisms to allocate distributed resources and addressed the problem of stability of pricing in computational market.

More recently, many projects like Spawn[18], Popcorn[4], Enhanced MOSIX[19], JaWS[20], Xenoservers[21], Mojo Nation [22] and Mariposa [23] have explored market based approaches. Unfortunately, many of them are limited to solving problems for specific domains and are monolithic in nature. These projects are also not inter-operable with current grid technologies. Writing applications for some platforms (e.g. Spawn and Popcorn), require a new programming interface. Consequently, developing applications or converting existing applications is more difficult.

Nimrod/G[24] is similar to OCEAN in employing market approaches to grid computing. But it is limited to resource management and doesn't provide a complete infrastructure that can be easily deployed on a user desktop. Efficient, customizable matching protocols for resource matching are missing too.

IV. REQUIREMENTS & DESIGN

The following are the broad requirements that drove the design of OCEAN.

- *Self-evolving, Scalable Matching Network*: The core of our approach consists of a matching network that provides mechanisms for quickly matching the resources. It should be self-evolving and scalable. We have chosen a Peer-to-Peer matching network with efficient evolution and matching protocols.
- *Resource description framework*: A flexible and powerful resource description framework for describing various

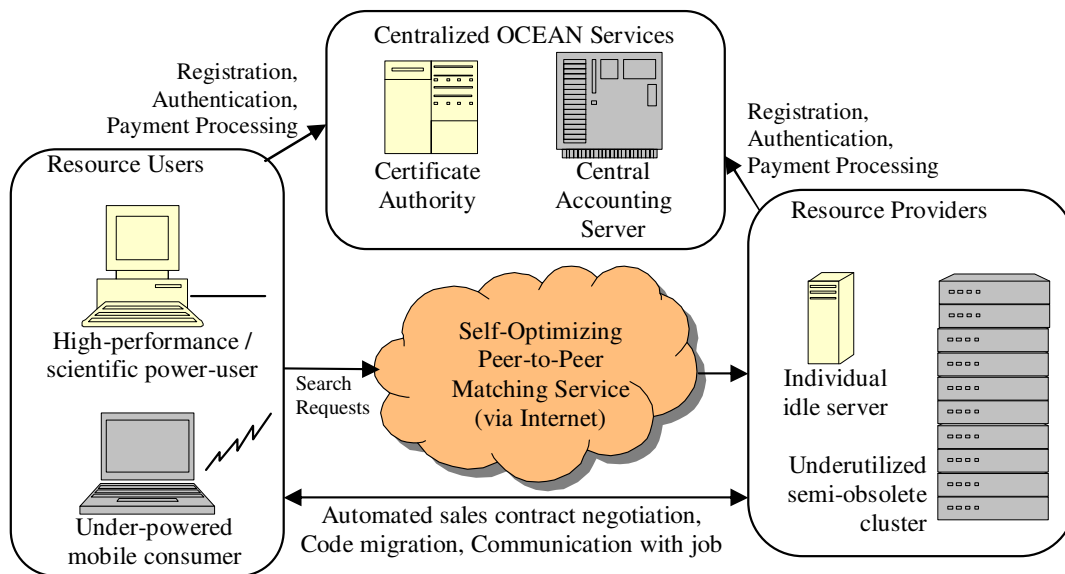


Fig. 1. High Level Overview of OCEAN

resources is required. The framework should provide APIs for writing these descriptions easily. We have a simple and flexible framework for describing resources in XML.

- *Portable, Open Protocols and APIs:* For widespread adoption of OCEAN for resource sharing, a portable and open implementation is required. The APIs should be flexible and powerful enough to exploit various features of OCEAN.
- *Interoperability with existing grid technologies:* Grid middleware like Globus and Legion are powerful and solve various issues in distributed computing. OCEAN can inter-operate with the grid using the existing tools and technologies for functions like data transfer, communication and security. Note that, OCEAN provides a complete system that can work as an independent system with out any grid middleware.
- *Security:* The buying and selling require a secure framework for billing and crediting the users. Mechanisms for encrypting messages sent over OCEAN framework are required. We have developed an XML signature mechanism for this purpose.
- *Easy Deployment:* Since OCEAN will be used by naive users as well as by sophisticated users like scientists, it is essential that deploying an OCEAN node be easy.

V. ARCHITECTURE

The OCEAN is composed of OCEAN nodes. Each node can act as a buyer or seller or both. The primary components of OCEAN can be divided into two parts: Market components and Transport components. The market components, matching, negotiation and accounting provide the market framework for resource sharing. The transport components included mobility, security and communication. These components provide features for transporting messages and jobs securely. Note that, OCEAN is interoperable with existing grid infrastructure like Globus. OCEAN provides a market-oriented framework that

can be used to enhance the services provided by various grid middleware. Figure 2 shows the ocean architecture.

The applications make use of OCEAN market services, which in turn use transport services. The transport services are implemented using existing run time platforms and specific language environments.

A. Interaction Among Components

For a potential buyer, the Matching layer of the Ocean Node serves to locate and rate resources available on the network. It does so by sending search requests out and then collecting search hits it receives. It then organizes these hits based on their possible utility and gives the best ones to the Negotiation Layer.

The Negotiation layer starts a contracting process with the desired host or hosts for the buyer. Negotiation either produces a contract or fails. If a contract is produced, then the Mobility Layer is notified of the task and it can migrate to and spawns the job. If the contract is not produced the Negotiation layer asks the Matching layer for more possible sellers.

Upon completion of the task, transferring of funds is carried out by the Negotiation layer and the central accounting server. Figure 3 shows the interaction of various components.

In the following sections, we see each component in detail.

B. Matching

The matching network is the core of OCEAN providing a framework for buying and selling of resources. Matching component is responsible for maintaining the matching network. The network is a self-evolving and scalable peer-to-peer network. The matching component includes

- A Resource Request/Description Language.
- Resource matching and searching algorithms
- PLUM (Peer List Update Manager) for tracking the list of peers
- Network evolution and optimization algorithms

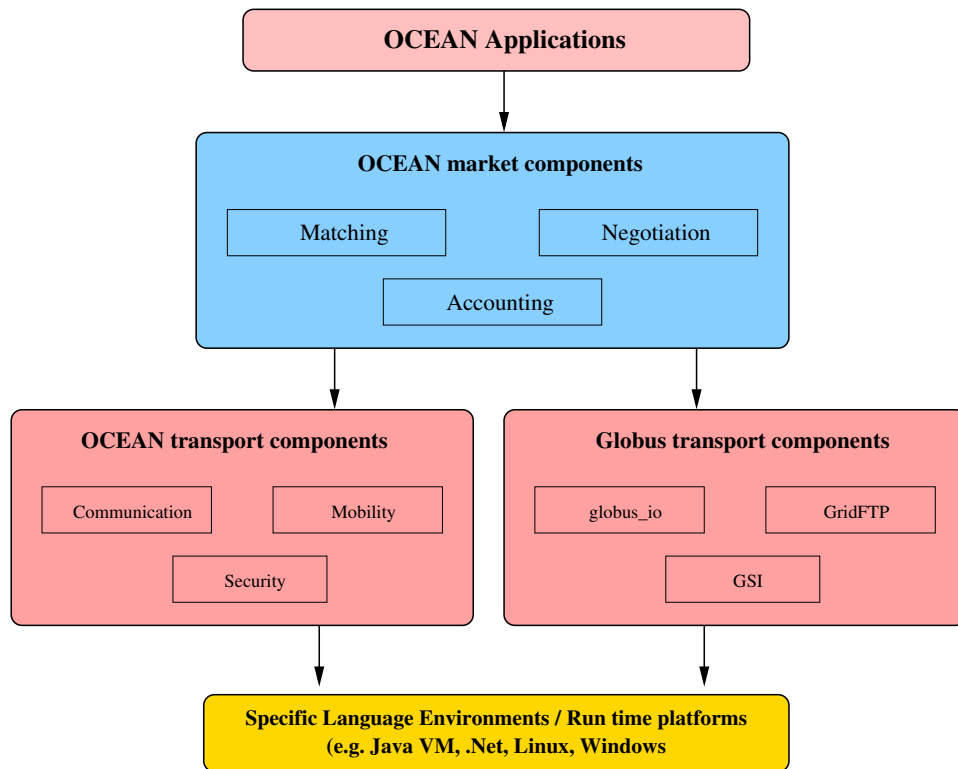


Fig. 2. OCEAN Architecture

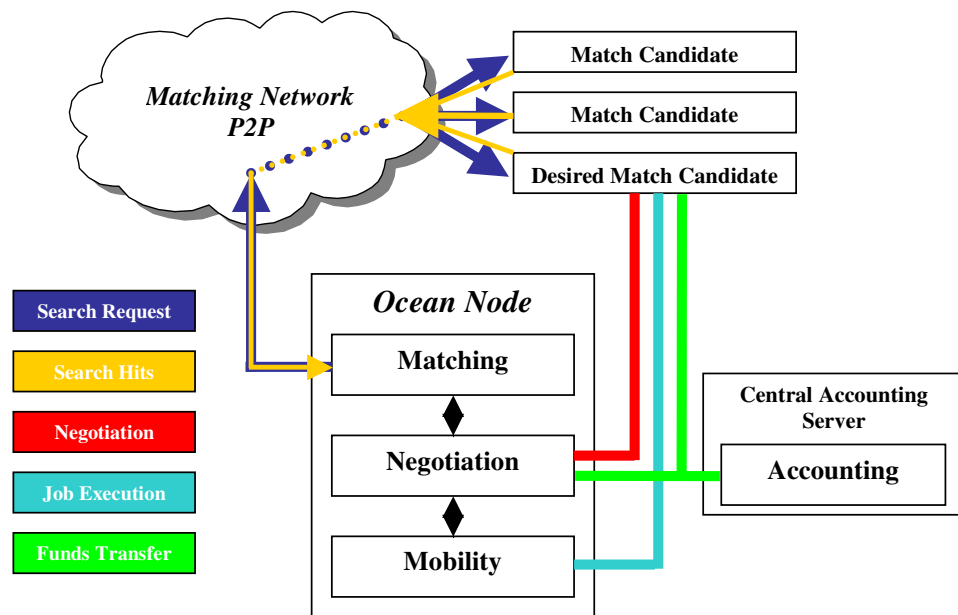


Fig. 3. Interaction among OCEAN components



Fig. 4. Example of a trade proposal

1) *Resource Request/Description Language*: We have defined an XML-based constraint language for matching the needs of resource users to the offerings of resource providers. The language is very general and can be easily extended to describe any type of resource requirements, including constraints on the CPU, memory, disk, network bandwidth, auxiliary hardware, software runtime environment, auxiliary libraries/databases, as well as various constraints on the acceptable sales agreement details, such as price details (currency, units), schedule for usage of resources, limits on use, method of payment (play money, OCEAN account transfer, intra-organizational journal transfer, OFX, PayPal, e-Gold, digital cash, credit card), means of payment (direct transfer, OCEAN-mediated, escrow through OCEAN CAS) and schedule of payments (up-front, on-delivery, post-hoc, periodic). Each application using or providing resources can decide how many constraints to specify up-front in a *trade proposal* for use in the distributed P2P matching system.

To reduce the computational demand on the matching system, the semantics of the constraint language is kept very simple (specifying only a boolean combination of minimum constraints that must be satisfied). More involved (Turing-

universal) custom matching criteria can be applied later by the buyer and seller in the negotiation stage, if desired. The matching process eventually leads to a one-on-one negotiation between buyer and seller, at which point complex custom strategies for deal optimization can be used.

We are exploring various ways of representing the complex requirements of users. Figure 4 shows a request from buyer for an access agreement that allows him to lease a certain package of resources, consisting of

- 1) Exclusive access to a ≥ 2 GHz Intel Pentium4 or better machine running RedHat (v.7.3 or greater) Linux and with Globus from ≥ 3.0 and the linpack library pre-installed
- 2) shared access to a Beowulf cluster of at least 10 machines, also running linux,
- 3) network storage of ≥ 10 GB, and
- 4) unlimited usage of a ≥ 100 GBps Ethernet connecting the machines and the network storage.

The buyer wants to access this system immediately in any desired usage pattern, for a period not less than 2 hours, but no more than 1 day, paying by the hour at a flat rate of not more than US\$1 per hour. Payment will be made via a direct

transfer from his OCEAN account to the seller’s account, but the buyer wants to be able to wait at least 1 day between completion of the resource usage and payment, presumably to give a human user a chance to manually inspect and dispute whether the resources were provided as agreed. Namespace qualifiers and schema URNs are omitted for readability.

Note the use of inequality constraints on the numeric values of various parameters. Such constraints, together with boolean combinations (conjunction/disjunction) thereof, will be automatically processed by the matching system at each node, to determine whether any of the locally-available trade proposals are compatible with the given proposal. For *buy* trade proposals, any unspecified details are assumed to be unconstrained, whereas for *sell* trade proposals, only the explicitly specified details are assumed to be available.

2) *Matching/Searching Protocols*: The matching module propagates the buyer’s trade proposal to a subset of its peers, in a fashion intended to maximize the number of matches found, while limiting the resources consumed in the matching system. A successful match leads to communication of the matching trade proposal back to the buyer, and possible initiation of detailed negotiations between the buyer and seller.

Efficient protocols for message propagation in the matching network are essential for quick matching. We have developed a protocol called MarcoPolo (named after the swimming-pool game) for matching. It is based on undirected peering relationships between nodes, which constitute mutual agreements to directly process and/or forward network search requests from each other. A node joins the network by querying an initial peer, which, if its peer quota is full, pushes the new node further out towards the network edges. A node can conveniently retrieve a list of all nodes that are located within n peering hops (see Figure 5), via a Marco (*Who’s there?*) broadcast, to which all nodes within the given range reply with a Polo (*Hello I’m here*) response.

A node can gradually optimize its location in the P2P network by modifying its set of active peering arrangements, so as to be located nearby (within few hops) of nodes that have a record of past matching success. This is done using the optimization algorithm discussed in the next section.

In this dynamic P2P network, each node is limited to communication with a subset of the total nodes available. Nodes communicate all requests with their direct peers, and the requests are then propagated via these direct links to even more nodes. Each hop exposes one’s requests to a new set of peers. Due to the exponential growth of communication with each hop, many nodes can be reached quickly.

To ensure that messages do not live forever, and to decrease the overall communication load on each node, messages are given a maximum number of hops (time-to-live), which they can travel. Each time a message travels a link its time-to-live is decremented. Messages continue until their time-to-live is zero.

Every node’s request exposure is limited to a fixed subset of the total peers on the network at any time. If the peer is very discriminating, or if it is looking for a rare resource on such a network, many desired matches may exist which are out of its reach. Because of this, it is in the best interest of

a node to maximize the usefulness of this subset. This is the focus of the self evolving nodes.

A node only has direct control of the direct peers it connects to. Choosing these peers wisely based on their utility helps improve overall performance for a node. This utility includes both a direct peer’s ability to provide a service and that peer’s connections to other peers which can provide a service.

There are many possible ways to allow nodes to alter their set of known peers to maximize the overall utility of their peer set. We focused mainly on collecting historical statistical data, and on using this to decide what do with the current peer set.

For a detailed account of MarcoPolo, see [25].

3) *PLUM (Peer List Update Manager) for tracking the list of peers*: For the evolution of the matching network, every node has to maintain information about other peers. The basic requirements are as follows.

- 1) Having each node collect statistics on its peers in order to evaluate their effectiveness at handling requests.
- 2) Forwarding requests first to those peers that are most likely to be able to handle them successfully.
- 3) Reducing the number of hops that messages must traverse between buyers and sellers by allowing nodes to learn about the peers of peers that are particularly effective at handling transactions.
- 4) Allowing statistics to decay over time so as to bias them towards more recent data (so as to more rapidly accommodate changes in performance).
- 5) An incentive system that rewards node operators for tuning their nodes for maximum effectiveness in the distributed algorithm.

The adaptive peer list having these characteristics is managed by PLUM (Peer List Update Manager). It maintains the data structures used by evolution protocols explained in the next section. It also periodically probes the peers and collects information about bandwidth, latency etc.

4) *Evolution and Optimization Protocols*: As we discussed above, it is critical for the network to self-evolve and optimize the set of peers for each node. We explored two evolution algorithms, which are described below.

Wave Algorithm: The algorithm seeks to maximize the effectiveness of direct peers by establishing a rating for each direct peer. This rating is directly based on the historical data of successful searches and the number of hits. This algorithm is a limited implementation of complete evolution algorithm(MPF) described in an earlier paper[26].

The rating is obtained from the following formula:

$$Rating = \frac{NumSuccess+SuccessBias}{NumTrials+TrialBias}$$

Over time the number of successes and attempts are decayed by a fraction to insure that results are biased towards newer data.

The evolution procedure is as follows

- Start each direct peer out with number of successes and number of attempts at zero.
- Each search add one to the attempts counter and each success add one to the successes counter.
- Over Time do the following:

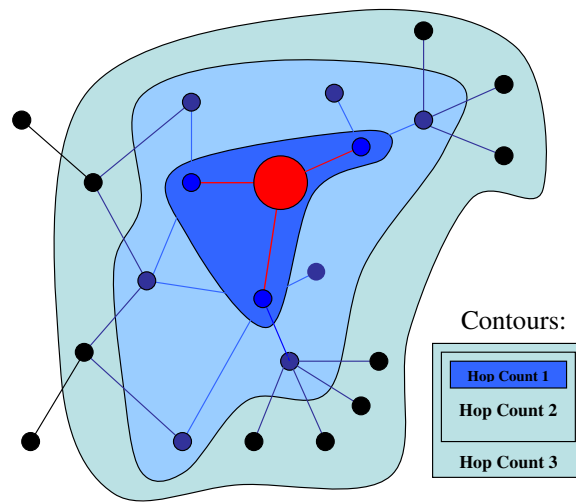


Fig. 5. Hop Contours in MarcoPolo P2P Net

- Decay the number of attempts and successes by a small fraction every few seconds to ensure more recent data is weighted more heavily.
- Every 200 or so decays:
 - * Make a decision to remove the worst performing direct peer.
 - * Obtain the peer list of the best performing peer.

This algorithm has the advantage of being straight-forward and so requires little book-keeping. On the other hand, full MPF algorithm[26] may produce better results at the cost of more time and complexity.

Undertow algorithm: This is another algorithm which seeks to maximize the effectiveness of direct peers. It measures how well each direct node is performing and the usefulness of the path of peers it makes available. The number of peers available through a direct peer at a certain ring is obtained using a hop-targeted Marco Message and then search requests are sent to this ring to determine their effectiveness. Figure 6 shows the first two rings in a possible network. Like previous algorithm, this method obtains possible service providers in the process, however it is a short time history approach.

The evolution procedure is as follows:

- Measure the effectiveness of direct peers by sending a set of search messages to each and obtain their success ratio.
- For each hop up to the time to live:
 - Send out a hop targeted Marco via each direct peer to the current hop. Record Polo responses to obtain the number of nodes available through this peer.
 - Send out a set of hop targeted Search messages via each direct peer to the current hop.
 - If an indirect peer drastically outperforms the direct peer, attempt to make it a direct peer and drop the current direct peer.

This algorithm also has the advantage of being simple and requires little bookkeeping. On the other hand, It takes time and active work to obtain the number of direct peers for each ring, and it also initially limits your searches to only a subset

of the reachable peers.

C. Negotiation

When the matching component returns multiple results from potential sellers, there is a need for negotiation. The Negotiation component provides automated and manual mechanisms for resource negotiation. The automation of negotiations is not a new concept. Maes et al.[27] of MIT media lab put negotiation at the center of the consumer buying behavior (CBB) model for e-commerce. Rosenschein et al.[28] gives many good examples of negotiation strategies. But, a standard set of rules expressed in succinct form for resource negotiation are lacking.

We have developed flexible XML based mechanisms that allow users to dynamically setup rules for negotiation. Figure 7 shows an example of contract. We also implemented two basic negotiation protocols: *yes-no*, and *static bargain*.

In the *yes-no* protocol, following are the sequence of events that happen.

- 1) After buyer receives matching resources, she sends her offer as a contract in the contract specification language to sellers.
- 2) The seller verifies the contract and checks to see if all the elements in the contract match his specifications.
- 3) If everything is in order the seller updates the status of the contract to "Accepted" and sends back the contract to the buyer. If something is wrong, that is, if some elements in the contract don't match, then the seller updates the status of the contract to "Rejected". and includes the rejected items in the contract along with a reason why they were rejected and sends back the updated contract to the buyer.
- 4) The buyer receives the updated contract and checks the status of the contract. If the contract is "Accepted". then the buyer signs the contract and sends the signed contract back to the seller. If the contract is "Rejected", the buyer either goes onto the next seller in the list or tries to take care of the rejected items and sends back an

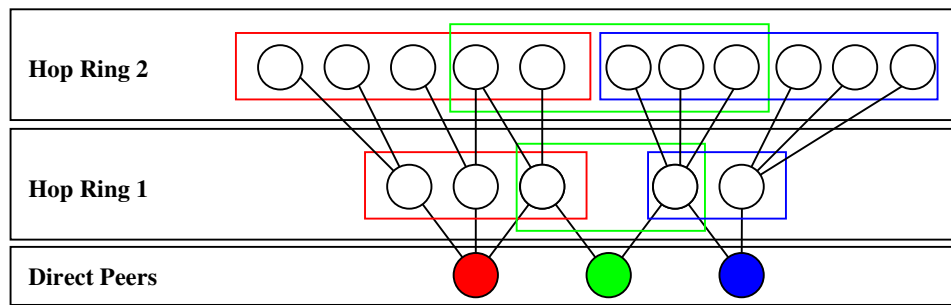


Fig. 6. Undertow algorithm: The colored boxes represent the tree levels related to direct peers measured at each ring

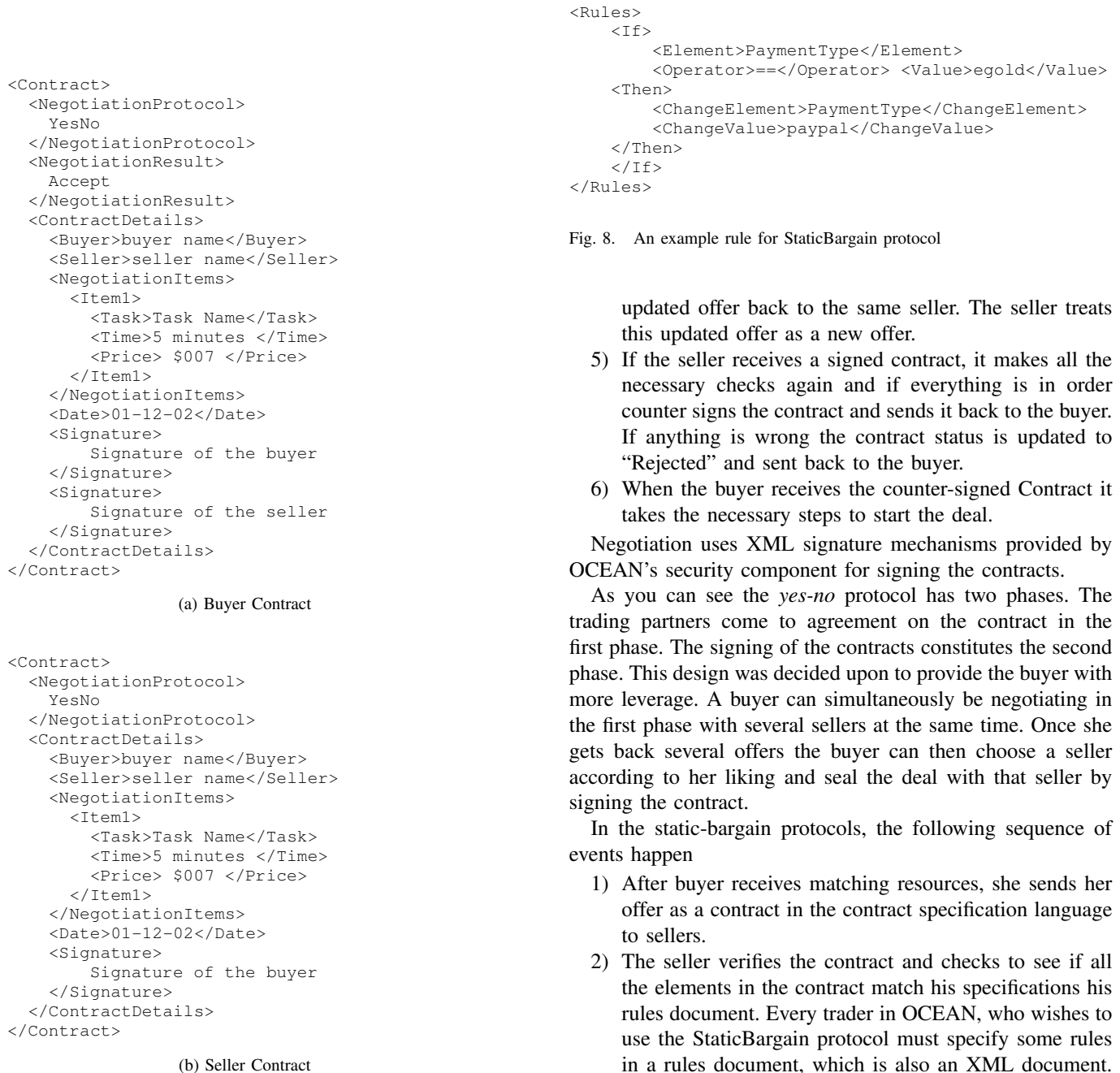


Fig. 8. An example rule for StaticBargain protocol

updated offer back to the same seller. The seller treats this updated offer as a new offer.

- 5) If the seller receives a signed contract, it makes all the necessary checks again and if everything is in order counter signs the contract and sends it back to the buyer. If anything is wrong the contract status is updated to “Rejected” and sent back to the buyer.
- 6) When the buyer receives the counter-signed Contract it takes the necessary steps to start the deal.

Negotiation uses XML signature mechanisms provided by OCEAN’s security component for signing the contracts.

As you can see the *yes-no* protocol has two phases. The trading partners come to agreement on the contract in the first phase. The signing of the contracts constitutes the second phase. This design was decided upon to provide the buyer with more leverage. A buyer can simultaneously be negotiating in the first phase with several sellers at the same time. Once she gets back several offers the buyer can then choose a seller according to her liking and seal the deal with that seller by signing the contract.

In the static-bargain protocols, the following sequence of events happen

- 1) After buyer receives matching resources, she sends her offer as a contract in the contract specification language to sellers.
- 2) The seller verifies the contract and checks to see if all the elements in the contract match his specifications his rules document. Every trader in OCEAN, who wishes to use the StaticBargain protocol must specify some rules in a rules document, which is also an XML document. The rules in the rules document are used to decide what to bargain for and how.
- 3) If everything is in order the seller updates the status of the contract to “Accepted” and sends back the contract

Fig. 7. Example Contracts (signatures are omitted for clarity)

to the buyer. If something is wrong, that is, if some elements in the contract don't match, then the seller updates the status of the contract to "rejected". and includes the rejected items in the contract along with a reason why they were rejected and sends back the updated contract to the buyer.

- 4) The buyer receives the updated contract and checks the status of the contract. If the contract is "Accepted", the buyer signs the contract and sends the signed contract back to the seller. If the contract is "Negotiate", the buyer looks at the rejected items and it's rules document and decides to either go onto the next seller in the list or try to take care of the rejected items and send back an updated offer back to the same seller. The seller treats this updated offer as a new offer.
- 5) If the seller receives a signed contract, it makes all the necessary checks again and if everything is in order counter signs the contract and sends it back to the buyer. If anything is wrong the contract status is updated to "Rejected" and sent back to the buyer.
- 6) When the buyer receives the counter signed contract it takes the necessary steps to start the deal.

In the StaticBargain protocol a rules document, which is an XML document, is used to make dynamic decisions during negotiation. When the user selects StaticBargain protocol as his negotiation protocol in the initial configuration screen, the user will be presented with a sequence of screens where he sets up his negotiation rules. Once the rules have been set up the user makes his negotiation decisions based on the rules document. The Java Expert System Shell, JESS [23], is planned on being used as the interpreter for the rules. Once OCEAN is started up the JESS rule engine also starts up. The rule engine then looks at the rules document and builds facts from the rules document.

When the negotiation takes place, the rule engine will make decisions and takes actions based on the facts. The advantage of using an expert system shell is that we can make the system to learn from previous negotiations. An example rules document is shown in figure 8.

This rule says: If the value of PaymentType in the received contract is e-gold[29], then add PaymentType as a rejected element and set the expected value of PaymentType to PayPal.

D. Central Accounting Server

The purpose of the central accounting server is to maintain OCEAN user accounts in a secure location that is physically controlled by the OCEAN administrators. It also provides a connection point between the OCEAN network and real-world financial networks. It may also publish consolidated information about market activity and prices. In exchange for operating the server, which provides a useful service, the administrator of the OCEAN may collect a small fee on each transaction that is conducted. In addition, it communicates critical information to the accounting system at a local node, to allow real-world payments to actually be processed and archived. However, micro payments for individual OCEAN contracts would be consolidated into larger amounts by the

central accounting system before submitting them to the real financial networks, which have a relatively high overhead.

Currently, we have implemented the accounting server in a centralized manner. This may cause scalability problems as number of OCEAN users and nodes increases. We are exploring mechanisms for developing a distributed accounting server.

E. Transport Components

The communication, mobility and security components provide transport services for market components in OCEAN. These modules can be replaced with similar services provided by existing toolkits like Globus.

1) *Communication*: Communication component provides basic synchronous and asynchronous communication primitives. These basic primitives are wrapped in a service called MailBox, which is used by higher level components. A component or an OCEAN task willing to communicate registers with a local MailBox and instructs it to either send messages or wait for incoming messages. MailBox works in conjunction with Naming (section V-E.2) for providing communication end-points for OCEAN. MailBox also has facilities for plugging in message filters. These filters are useful in reducing communication overhead.

2) *Naming*: The Naming component is responsible for name resolution in the OCEAN system network. The Communication component relies heavily on this component. It is undesirable for components to refer to OCEAN nodes strictly by their IP addresses, since IP addresses can change and be obscured by firewalls. Moreover, in grid frameworks there is a need to locate not just computers in the network, but also other entities such as computing tasks, data, and other resources. Names of OCEAN resources are syntactically defined as extensions to the URL/URI standard. Any resource can be located using a path-naming scheme as shown in figure V-C.

Each hostname or IP address, following the first, may be a private hostname (or IP address on a private intranet) of a machine reachable from the preceding host. Such a naming scheme helps in reaching any target machine or entity, whether or not it has a public, static IP address. Messages are forwarded along the designated path until they reach the eventual destination. This is equivalent to using IP as a link layer protocol, and building a network-routing layer on top of it, to get through the barriers separating different IP address spaces.

3) *Mobility - TSM (Task Spawning and Migration)*: We have developed a light weight task spawning and migration mechanism for OCEAN. The Task Spawning and Migration (TSM) component is responsible for code mobility across remote OCEAN node servers and monitoring task execution. Depending on when a task is scheduled for execution, the TSM subsystem spawns computing tasks on remote servers and monitors their execution. This includes transfer of computational tasks to allocated nodes, setting up of executables, checking access policies and permissions, creating adequate execution environments, initializing task execution, passing arguments, and finally managing task termination.

```
ocean://<public host name or IP address>:<port number>/
      <private host name or IP address>:<port number>/
      .(any additional hosts on private sub-subnets)./
      @<ocean module name>/<jobid>
```

Fig. 9. An Example OCEAN Name

OCEAN tasks, that form the parts of a user job, can be migrated to a remote machine. The code (compressed file), which is migrated, includes a task description, executable classes that comprise the actual work that needs to be performed and authentication information in the form of digital signatures and certificates. In the current version, tasks that are not serializable cannot be migrated.

4) *Security*: Security is one of the major issues discussed during the design of OCEAN. It needs special attention in any distributed system in general and automated computational markets in particular. The Security component in OCEAN works in conjunction with the Matching, Negotiation and Accounting components. It comes into play every time a message is sent out or an incoming message needs to be validated. At present only the contracts are being signed and validated. But in the future versions even the matching request, and the negotiation transactions might be signed for greater security.

We have developed an XML Signature Module (XSM) for security of XML messages sent over OCEAN network. The primary reason for using XML digital signatures in security component for securing transactions, which otherwise can be handled by SSL, is to ensure permanent non-repudiability and impossibility of forgery. XML signatures ensure that application layer attacks such as repudiation of authorship can be prevented. A secondary reason for using XML signatures is to enable the signing of selected elements of a contract.

The XSM consists of two protocols: Registration and Signing and Validation Protocols. The implementation of XSM is carried out in accordance with the recommendations passed the joint proceedings of a W3C and IETF working group. For a detailed account of design process and security issues that are identified and solved, see Sahib's thesis[30]. The thesis also presents an excellent literature survey of existing security mechanisms in distributed systems.

VI. IMPLEMENTATION

We have implemented OCEAN both on Java and .Net platforms. The idea is to show that OCEAN can be built on varying platforms with equal ease. In both the implementations, our goal was to provide a basic framework, on top of which, complex protocols can be developed. Each component in OCEAN can be compiled and run as a single entity. For example, the TSM component can be used by normal applications for migrating jobs without the need of other market components. We are working on exposing the market services as web services.

In both the implementations, we have developed the basic matching, negotiation and evolution protocols discussed in section V. The communication, mobility and security modules all provide the basic services expected of them. More work

needs to be done in migrating running jobs with stateful information. The Accounting module is fairly complete and provides a gateway to the financial world. We have used dummy money for transactions and have also conducted some experiments with e-gold[29].

Figure 10 shows snapshots of OCEAN. Note that these figures are from the .Net implementation of OCEAN.

VII. EXPERIMENTAL RESULTS

We have done simulations using the OCEAN matching network for measuring the performance of our protocols. We created 60 nodes with varying mix of seeker(buyer)s and provider(seller)s. For matching, we used a synthetic matching criteria shown in the figure 11(e). Each seeker searches for a service with the probability distribution shown. The supply and demand for resources in the network are varied as follows.

- 30% seeker to provider ratio - 30% of nodes in the network are seekers
- 70% seeker to provider ratio - 70% of nodes in the network are seekers

All three protocols (network with marcopolo matching with no evolution algorithms), wave and undertow are tested with 50 samples. Results are shown in figure 11.

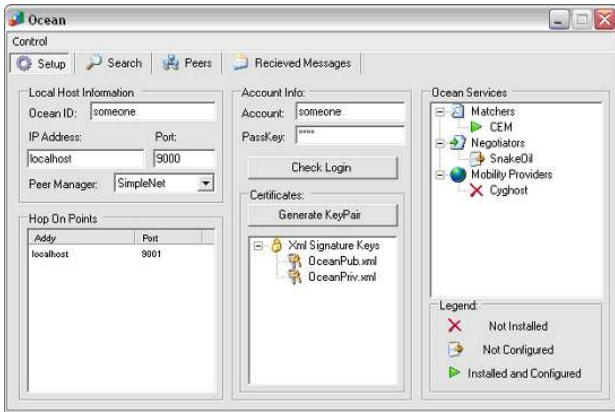
The figures 11(a) and 11(b) show search performance for 30% and 70% mix. Both wave and undertow performed better for 30% mix. This is promising because the most likely mix for OCEAN seems to be many providers and few seekers. On the other hand, wave performed poorly for 70% mix due to low success rate for attempts done by the seeker.

The figures 11(c) and 11(d) show match performance on the provider side. As you can see, there is little change in the performance for either 30% or 70% mix with or without evolution protocols. We are exploring the idea of provider evolution for improving match performance on the provider side. We would like to caution that these results are preliminary and conducting experiments with real machines.

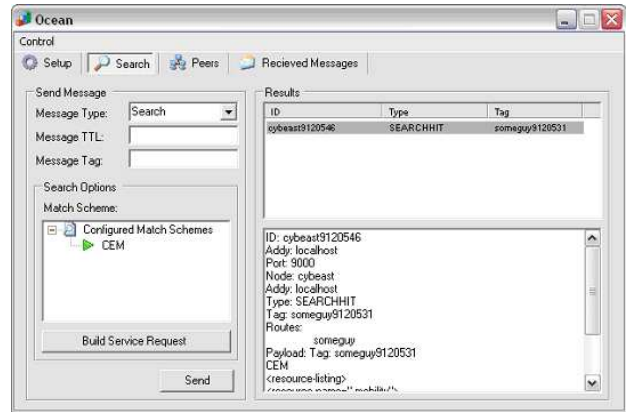
VIII. FUTURE WORK

We briefly discuss concepts that we plan to explore in future versions of the OCEAN. We have provided a basic framework with simple and flexible protocols for exploring market-oriented approaches for distributed computing. We envision development of complex protocols on top of this framework.

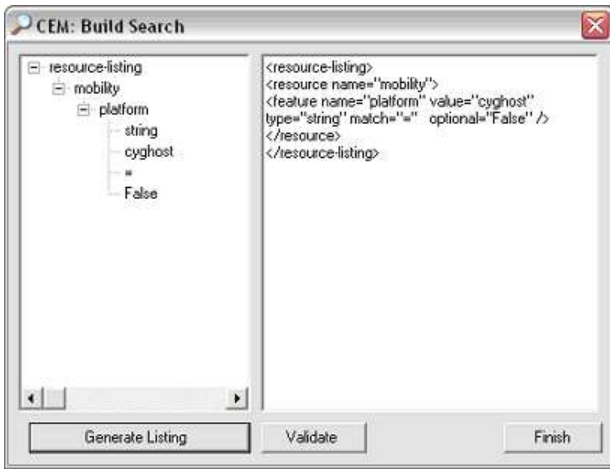
We are exploring a more versatile method of handling resource requests via a match scheme. Match Schemes are simple named protocols which allow reviewers of requests to interpret the data as the searcher intended. Users can have multiple Match Schemes installed and handle requests for all simultaneously. Match Schemes provide both the ability to



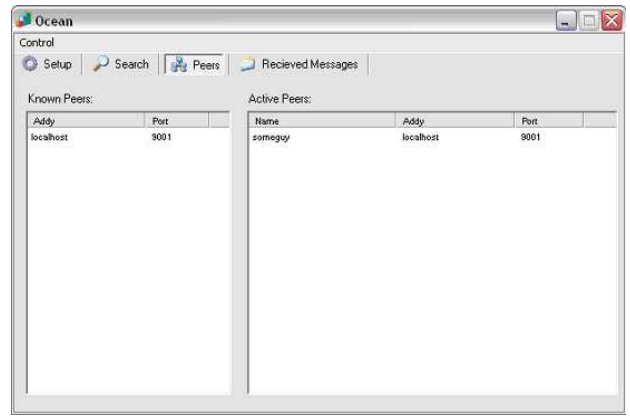
(a) Configuration



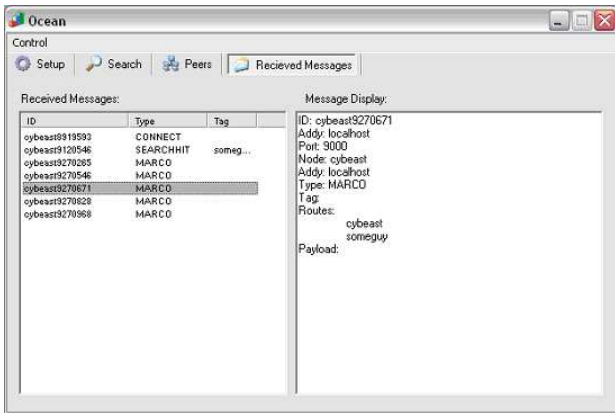
(b) Search Screen



(c) CEM Search Builder

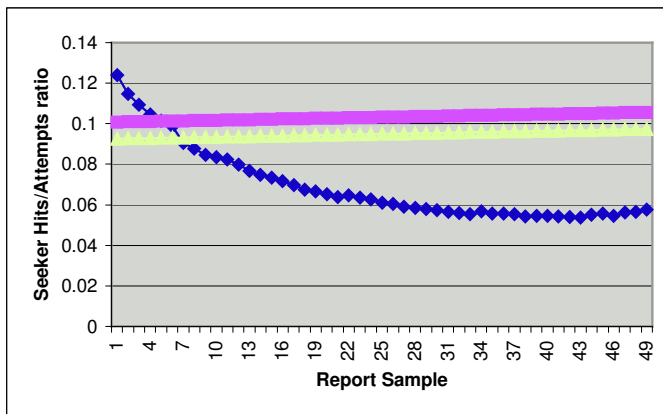


(d) Peers

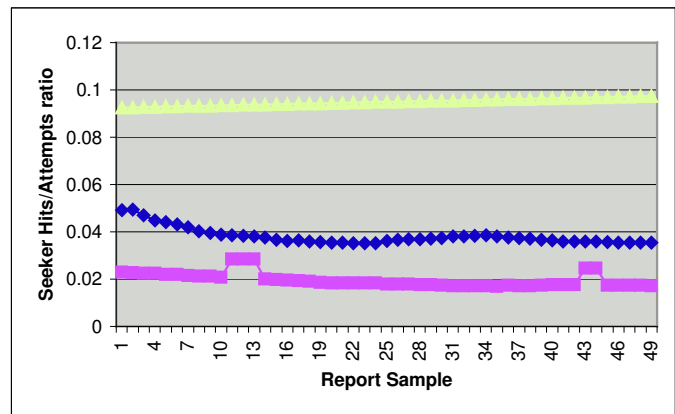


(e) All Received Messages

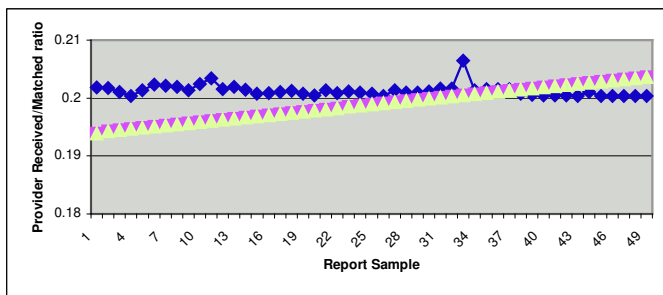
Fig. 10. OCEAN Snapshots



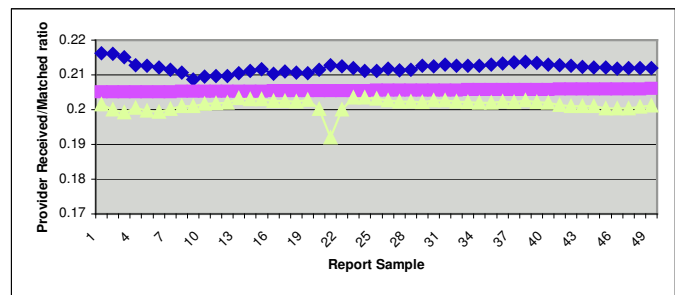
(a) Search Results for 30% Seekers



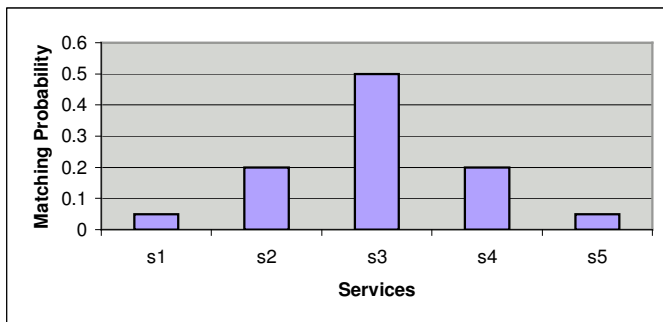
(b) Search Results for 70% Seekers



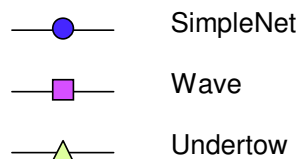
(c) Match Results for 30% Seekers



(d) Match Results for 70% Seekers



(e) Matching Probability Distribution



(f) Legend

Fig. 11. Experimental Results

build search requests for seekers and a method for comparison of local resources and requested requirements for providers.

We are planning to explore an interesting concept called “Peer ranking”. This is similar to rating systems available on Ebay.com, Epinions.com and Resellerratings.com. Each peer ranks the peers that participated in the trade depending on its interaction with them. It is important to have a “collusion-proof” mechanism for the ranking.

We are developing web services on top of OCEAN framework. Currently, the CAS(Central Accounting Server) can be accessed as a web service. We would like to standardize and

provide matching and negotiation web services.

Another important research approach is to develop OCEAN market components using transport components provided by projects like Globus. We already use GridFTP[31] for transferring input and output files. Efforts are also being made to incorporate mobile-agent based subsystems like ADK[32] from Tryllian in the OCEAN framework.

IX. CONCLUSIONS

We have described a framework for a market-based approach to distributed computing. We have identified the requirements and technical challenges in such an approach.

A distributed scalable peer-to-peer matching network with efficient matching and evolution protocols is proposed for finding distributed resource quickly. The architecture and various components that form the basis of OCEAN are described in detail.

ACKNOWLEDGMENTS

The authors would like to thank past and present members of OCEAN for their valuable contributions and comments.

REFERENCES

- [1] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor : A hunter of idle workstations," in *8th International Conference on Distributed Computing Systems*. Washington, D.C., USA: IEEE Computer Society Press, June 1988, pp. 104–111.
- [2] B. Haynes, "Computing science: Collective wisdom," *American Scientist*, vol. 86, no. 2, pp. 118–122, Mar. 1998.
- [3] A. Patrizio, "Discover distributed computing," *Byte.com Magazine*, Sept. 1999.
- [4] N. Nisan, S. London, O. Regev, and N. Camiel, "Globally distributed computation over the internet – the POPCORN project," in *18th International Conference on Distributed Computing Systems (18th ICDCS'98)*. Amsterdam, The Netherlands: IEEE, May 1998.
- [5] "distributed.net." [Online]. Available: <http://www.distributed.net/>
- [6] M. Baker and R. Buyya, "Cluster computing at a glance," in *High Performance Cluster Computing*, R. Buyya, Ed. Upper Saddle River, NJ: Prentice Hall PTR, 1999, vol. 1, Architectures and Systems, pp. 3–47, chap. 1.
- [7] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a Future Computing Infrastructure*. San Francisco, California: Morgan Kaufmann Publishers, 1999.
- [8] M. Baker and G. Fox, "Metacomputing: Harnessing informal supercomputers," in *High Performance Cluster Computing*, R. Buyya, Ed. Upper Saddle River, NJ: Prentice Hall PTR, 1999, vol. 1, Architectures and Systems, pp. 154–185, chap. 7.
- [9] A. S. Tanenbaum and S. Mullender, "An overview of the Amoeba distributed operating system," *Operating Systems Review*, vol. 15, no. 3, pp. 51–64, July 1981.
- [10] J. K. Ousterhout, A. R. Cherenon, F. Douglass, M. N. Nelson, and B. B. Welch, "The Sprite network operating system," *Computer*, vol. 21, no. 2, pp. 23–36, Feb. 1988.
- [11] 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.
- [12] A. S. Grimshaw, W. A. Wulf, and the Legion team, "The legion vision of a worldwide virtual computer," *Communications of the ACM*, vol. 40, no. 1, pp. 39–45, Jan. 1997.
- [13] V. Huber, "UNICORE: A Grid computing environment for distributed and parallel computing," *Lecture Notes in Computer Science*, vol. 2127, pp. 258–266, 2001.
- [14] P. Avery and I. Foster, "The griffyn project: Towards petascale virtual data grids," 2001. [Online]. Available: <http://www.griffyn.org>
- [15] "ivdgl (international virtual data grid laboratory)." [Online]. Available: <http://www.ivdgl.org>
- [16] I. E. Sutherland, "A futures market in computer time," *Communications of the ACM*, vol. 11, no. 6, pp. 449–451, 1968.
- [17] M. S. Miller and K. E. Drexler, "Markets and computation: Agoric open systems," in *The Ecology of Computation*, ser. Studies in Computer Science and Artificial Intelligence, B. A. Huberman, Ed. Elsevier Science Publishers, 1988, pp. 133–175.
- [18] C. Waldspurger, T. Hogg, B. Huberman, J. Kephart, and S. Stornetta, "Spawn: A distributed computational economy," *IEEE Transactions Software Engineering*, vol. 18, no. 2, Feb. 1992.
- [19] Y. Amir, B. Awerbuch, A. Barak, R. S. Borgstrom, and A. Keren, "An opportunity cost approach for job assignment in a scalable computing cluster," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 760–??, July 2000.
- [20] S. Lalis and A. Karipidis, "An open market-based architecture for distributed computing," *Lecture Notes in Computer Science*, vol. 1800, pp. 61–??, 2000.
- [21] D. Reed, I. Pratt, S. Early, and P. Menage, "Xenoservers: Accountable execution of untrusted programs," in *The Seventh Workshop on Hot Topics in Operating Systems: [HotOS-VII]: 29–30 March 1999, Rio Rico, Arizona*, IEEE, Ed. 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA: IEEE Computer Society Press, 1999, pp. 136–141.
- [22] "Hivecache: Distributed enterprise online backups." [Online]. Available: <http://www.mojonation.net/>
- [23] M. Stonebraker, P. M. Aoki, R. Devine, W. Litwin, and M. Olson, "Mariposa: A new architecture for distributed data," in *Proceedings of the 10th International Conference on Data Engineering*, A. K. Elmagarmid and E. Neuhold, Eds. Houston, TX: IEEE Computer Society Press, Feb. 1994, pp. 54–67.
- [24] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An architecture of a resource management and scheduling system in a global computational grid," *HPC Asia 2000, IEEE Press*, Sept. 22 2000.
- [25] C. Harrison, "Self evolving peer to peer networks for service discovery design of marcopolo matching network for the open computation exchange and arbitration network (ocean)," Apr. 2003.
- [26] S. Govindaramanujam, C. Harrison, E. Jansen, S. K. Nallan, S. Singh, and M. Frank, "Locating suitable resources in ocean (poster paper)," Oct. 2002. [Online]. Available: http://www.cise.ufl.edu/research/ocean/hipc02_poster.pdf
- [27] P. Maes, R. H. Guttman, and A. G. Moukas, "Agents that buy and sell," *Communications of the ACM*, vol. 42, no. 3, pp. 81–91, Mar. 1999.
- [28] J. S. Rosenschein and G. Zlotkin, *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. Cambridge, Massachusetts: The MIT Press, 1994.
- [29] "E-gold." [Online]. Available: <http://www.e-gold.com>
- [30] S. S. Wadhwa, "Signing and validating contracts in ocean," Master's thesis, University of Florida, Apr. 2003. [Online]. Available: http://www.cise.ufl.edu/research/ocean/sahib/Wadhwa_S.pdf
- [31] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data management and transfer in high-performance computational grid environments," *Parallel Computing*, vol. 28, no. 5, pp. 749–771, May 2002.
- [32] T. Inc., "Agent development toolkit(adk)." [Online]. Available: <http://www.tryllian.com>