Design of a Self-evolving Scalable Matching Network for OCEAN

Pradeep Padala and Michael P. Frank
Computer & Information Science & Engineering
University of Florida
Gainesville, Florida 32611–6120
Email: {ppadala,mpf}@cise.ufl.edu

Poster Paper, To be presented at the International Conference on High Performance Computing (HiPC'03)

***Abstract*— OCEAN (Open Computation Exchange and Arbitration Network) provides a market-based framework for grid computing. We developed a self-evolving, scalable matching network for finding suitable resources. In this paper, we explain OCEAN architecture briefly and detail the design choices for the matching network.**

I. Introduction

Grids[1] have become the favourite choice for executing compute intensive and data intensive scientific applications. As more and more people start deploying grid resources, there is a growing need for finding suitable resources. OCEAN (Open Computation Exchange and Arbitration Network) provides a market-based framework for locating resources.

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 grid resources that can execute the computation in question. We have developed a self-evolving matching network that finds a list of sellers matching buyer's requirements. The buyer can then negotiate with various sellers and ask OCEAN to transport and execute his jobs.

Note that an OCEAN node can be deployed by an individual user or by an administrator managing a virtual organization(VO) in a grid. In this paper, we describe OCEAN architecture briefly and the matching protocol in detail.

II. OCEAN 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[2]. OCEAN provides a market-oriented framework that can be used to enhance the services provided by various grid middleware. Figure 1 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.

For a detailed description of interaction of among components, see [3]

III. Locating Desired Peers : Design of 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.

In the following section we describe our motivation and various challenges faced in designing such a network.

A. Network Scenarios

The task of locating a suitable peer base which can provide the required resources is a difficult task. Constructing such a network can be approached in many ways:

- *Client/Server*: The advantage would be that we have a simple network where everyone is easy to locate. Disadvantage is that it is not scalable and has a single point failure. There is a single owner which incurs all the expense.
- *Distributed Servers*: The advantage would be that we have a simple network where everyone is easy to locate. Disadvantage is that it is not scalable and again has a single owner which incurs all the expense.
- *Peer to Peer*: Advantage would be very scalable, distributing the traffic over all nodes and there is no single point of failure. A disadvantage is that it is more difficult to find a desirable peer.

For the reasons of distributed ownership and scalability we have adopted the last approach. Ocean's peer to peer matching system is developed similar to Gnutella. The protocol includes discovery of unknown peers, and provides a flexible search mechanism which supports for any of the three main methods of match arbitration (Resource Listing, Benchmark, and Probe

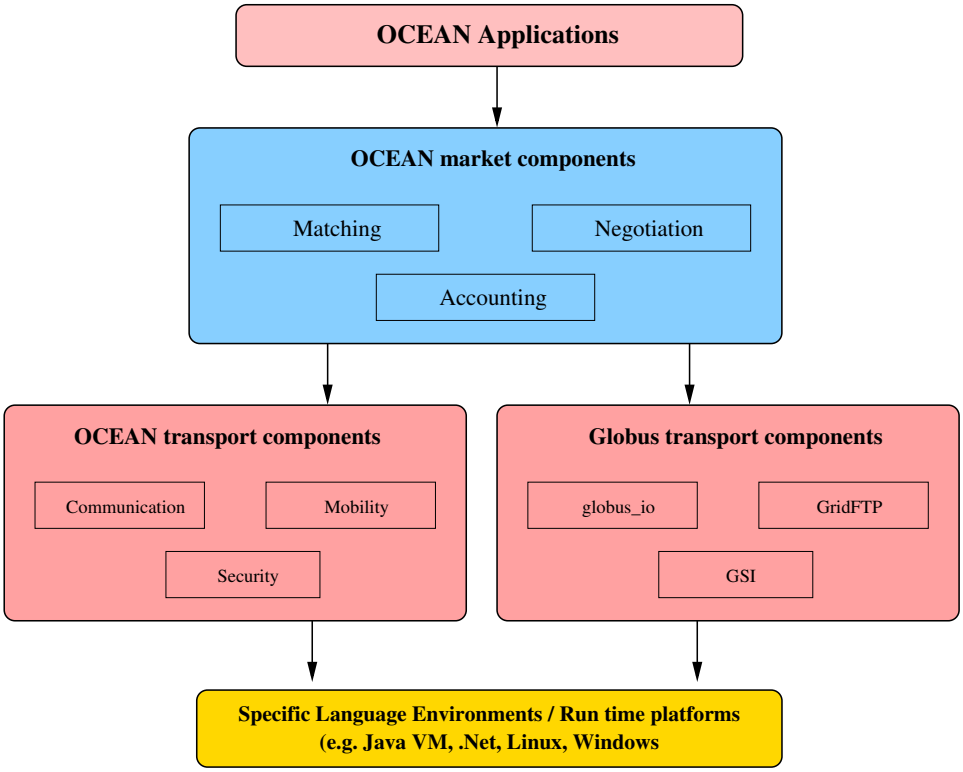

Fig. 1. OCEAN Architecture

Based). It differs from Gnutella in that message routing information travels with the message.

Search messages in the network are broadcast to all nodes with in a certain region of the source node. If a node receives a search request and it matches the request it will send back a complete resources description and a price. The initiator of this search can now establish a contract with this resource for the given price, if the initiator determines that it is worthwhile.

B. Network Extensibility

The protocol is easily extensible to include simple hierarchical peer to peer organization techniques such as super nodes. Super Nodes have been used by LimeWire developers to solve Gnutella scaling issues via the UltraPeer Protocol. The shielding of network traffic and routing optimizations demonstrated with UltraPeer concept[4] can be adapted to enhance the OCEAN Matching system in two important ways:

- *Leaf Node Grouping and Shielding*:
  If leaf nodes join super nodes based on their purpose, traffic shielding optimizations are possible. Buyer nodes can join buyer class super nodes and because buyers are interested in sending but not reviewing search requests, the super node can shield them from these unnecessary messages.
- *Leaf Node Information Caching*:
  If leaf nodes provide static search request info (Resource Listings and Benchmark Results) super nodes can cache this data. This allows the super nodes to respond to simple search requests and also discovery requests for the leaf nodes and limit network traffic.

Like UltraPeers, such super nodes can coexist with regular nodes who do not conform to a leaf protocol. The OCEAN Matching System protocol was built with these considerations to give flexibility to explore many possible network evolution schemes.

IV. Network Evolution

Like similar large distributed peer to peer networks, at any one time an OCEAN node can only know a finite number of peer nodes. We define this as a node's *reach*[5]. Our goal is to propose a scheme that will allow an OCEAN node to learn how to move throughout the network over time to improve the quality of the nodes within the reach restriction. We propose a machine learning algorithm that does the following:

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 a software component called the PLUM (Peer List Update Manager). Let us now describe the algorithm.

A. Data structures

Each node maintains a local, persistent data structure called the peer list. The peer list is simply a list of peer entries that is maintained in a preference order (most-preferred first). Each entry contains the name (i.e., ocean: pathname) of the peer OCEAN node in question, and the following statistics which are locally maintained for that peer:

- *nTrials*: Nominally, this counts the number of resource requests that we (the current node) have forwarded from buyers (including ourselves) to this particular peer. I say "nominally" because the decay procedure (described later) implies that older trials may be counted less heavily.
- *nSuccesses*: Nominally, this counts the number of forwarded resource requests that have actually led to a successful sale of resources.

Other variables mentioned below are constant parameters local to each PLUM that may be configured by the node operator.

B. Peer rating

For each peer in the peer list, its rating is computed according to the following formula:

$$rating = \frac{nSuccesses + successBias}{nTrials + trialBias}$$

The intent of the rating is to predict the probability of success (i.e., of leading to a sale) for sending a request to the given peer. The $successBias$ and $trialBias$ values are constant parameters of our own node that may be set by the node operator, that effectively give the number of successes and trials that the node operator thinks would be typical for a new peer in the absence of any actual data. Note that in the absence of any data, the predicted probability of success is just $\frac{successBias}{trialBias}$. If $successBias = 1$ and $trialBias = 2$, the formula $\frac{nSuccesses+1}{nTrials+2}$ is exactly the success probability that would result from a Bayesian analysis of the data using a Beta distribution which results from a maximum-entropy assumption that any probability of success between 0 and 1 is a priori equally likely, and therefore that the average probability of success is $\frac{1}{2}$. If we want to get some a priori probability of success p other than $\frac{1}{2}$, we simply arrange that $\frac{successBias}{trialBias} = p$. The absolute (as opposed to relative) magnitudes of $successBias$ and $trialBias$ determine how much data is needed to override our bias. For example, if $successBias = 10$ and $trialBias = 20$, then the initial probability of success will still be assessed at $\frac{1}{2}$, but it will take 10 times as many data points, given a particular actual frequency of successes, to achieve a given rating (say 0.9), compared to the case where the bias values are just 1 and 2. (A more principled way to compute $successBias$ and $trialBias$, rather than just picking them out of a hat, might be to just use, say, some fraction of the total $nSuccesses$ and $nTrials$ statistics over all our peers.)

Procedure:

1) Upon initialization after being installed, the local PLUM has exactly 1 peer on its peer list: the pre-programmed OCEAN node at the CAS server, and the data for this peer is initialized to $nSuccesses = nTrials = 0$.
2) Periodically (every $queryPeriod$ seconds), the PLUM will send a query to each of the top $nToQuery$ peers on its peer list. This query requests that the peer insert the current node into its peer list if it's not already there (with 0 $nSuccesses$ and $nTrials$), and also that the peer return back the list of its top $nToReturn$ peers (a parameter of the peer's PLUM). The peers are returned together with their $nSuccesses$ and $nTrials$ values, which are multiplied by $hearsayPenalty$ (a local parameter, for example 0.1 to indicate that we believe our peers' statistics only 0.1 times as much as our own statistics) and then added into our own statistics for those same peers, after multiplying those by (1-$hearsayFactor$, a fraction between 0 and 1), which prevents feedback between ratings from getting out of hand. Also, if the peer is not in our peer list yet, we first add it, with null statistics. If the result is that the peer list size is greater than $maxNPeers$, then the lowest-rated peers (after updating the ratings) are discarded from our peer list until only $maxNPeers$ peers remain.
3) Also periodically (every $degradePeriod$ seconds), we multiply every $nSuccesses$ and $nTrials$ value for every peer on our peer list by (1 - $decayFraction$). The $decayFraction$ is some small number (e.g., 0.01). Doing this guarantees that (a) the values of the statistics don't grow without bound, and (b) that older statistics are weighted less heavily (by a factor of 1-$decayFraction$ per $degradePeriod$) relative to newer statistics. This allows the behavior of the network to adapt more rapidly to changing circumstances.
4) Each time we receive a request for resources (either from our own node, or forwarded from another node), after making sure it is a new request and decrementing and checking its $hopsToLive$, and adding our node to the end of the request's record of the nodes it has traversed, we forward the request to all the top-rated $nToAttempt$ peers on our peer list (in order from highest to lowest rating), and simultaneously increment the $nTrials$ datum for each of those peers.
5) If we can handle the request for resources ourselves, we contact the buyer with the information about our offering, and a copy of the request's travel path, so that the buyer may consider making a transaction directly with us. If $alsoForward = True$, then we also forward the request to our peers (in case the buyer doesn't like our own offer).
6) When the buyer receives an offer from a node, if

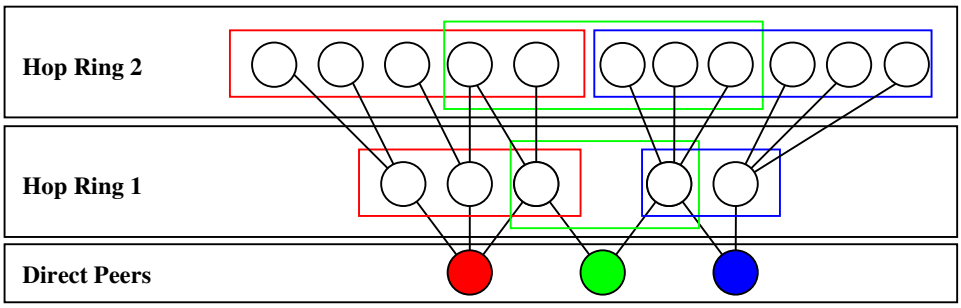

Fig. 2. Undertow algorithm: The colored boxes represent the tree levels related to direct peers measured at each ring

$rememberPathToSeller = True$, it adds the last $nToRemember$ nodes along the travel path directly to its peer list if not there already, and increments their nSuccesses and nTrials values. This way, the buyer has an improved chance of more directly reaching those same nodes in the future.

7) After the buyer has negotiated a successful contract with a seller, it is the buyer node's responsibility to contact the first node along the path that the original request traversed (this will be the buyer node itself), to report the successful sale. This report includes the path that the request traversed.
8) When a node receives a report of a successful sale, it removes itself from the front of the path in the report, and increments the nSuccesses count of the peer that is next in the path (re-adding it to its peer list if necessary). If $rememberPathToSeller = True$, it adds the last $nToRemember$ nodes along the travel path directly to its peer list if not there already, and increments their nSuccesses and nTrials values. Then it forwards the report to the next node in the path.
9) When the transaction is reported to the CAS in order to request that payment be carried out, the payment request is required to include a report of the path that the original resource request had traversed, leading up to the sale. Every node along this path (except buyer and seller) is given a fraction of the finder's fee, which is a portion of the OCEAN transaction fee that is set aside for this purpose. If the buyer had contacted the seller directly, then no finder's fee is levied.

V. Implementation

As we discussed above, it is critical for the network to self-evolve and optimize the set of peers for each node. We have implemented the following two algorithms that are limited version of above discussed methods. The implementation is done both on Java and .Net platforms.

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.

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:
  - Decay the number of attempts and successes by a small fraction every few seconds to ensure more recent data it 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 algorithm 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 2 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 though 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.

Our preliminary results indicate that the evolution protocols perform better than simple matching network with no evolution. More details are can be found in [3].

VI. Conclusions

We have described a market-based infrastructure for meta computing that can be used to buy and sell grid resources. A self-evolving, scalable matching protocol is described in detail. Our experiences with two evolution alogrithms are provided.

References

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

[2] 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.

[3] P. Padala, C. Harrison, N. Pelfort, E. Jansen, M. P. Frank, and C. Chokkareddy, "Ocean: The open computation exchange and arbitration network, a market approach to meta computing," in *Proceedings of the Second International Symposium on Parallel and Distributed Computing*, Oct. 2003.

[4] C. R. Anurang Singla, "Ultrapeers another step towards gnutella scalability," http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm.

[5] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.