

Locating Suitable Resources in OCEAN

Sama Govindaramanujam, Cyrus Harrison, Erwin Jansen
Sriram Kumar Nallan, Sahib Singh, Michael P Frank

October 6, 2002

Abstract

We propose a learning algorithm used by individual matching nodes on the OCEAN Network to help evolve the network with goals of increased speed and quality of search results.

1 Introduction

There are currently millions of people and organizations whose computing resources are connected via the Internet. It has been observed by numerous sources that these resources are frequently idle [4, 2]. Many computers that are online are not involved in any compute-intensive tasks. The total computing power is underutilized. The computing world can potentially be revolutionized if systems can transparently buy, sell, and use remote computing resources via the Internet.

OCEAN (Open Computation Exchange And Arbitration Network) is a computational market in which CPU time, memory, network bandwidth etc are tradable commodities. OCEAN is a layered system that exposes its different layers as web services accessible via SOAP. Within a single node we have a matching service, a negotiation service and a mobility service.

The mobility service can be any grid architecture, like globus[3] for example, that allows interaction with the negotiation layer. The mobility layer requests a location where it can execute a task from the negotiation. The negotiation layer then contacts the matching layer to locate a set of possible hosts. When this list is obtained the negotiation layer can establish a contract with one of these hosts, file the contract with the accounting service and give the mobility layer a location where it can execute its task.

The task of the matching layer is to find a set of suitable hosts where a task can be executed. The matches should be ordered according to the interest of the negotiation layer. The matching layers of OCEAN nodes in the network create Peer to Peer network for discovery of suitable hosts. The details of the matching service layer are discussed in this paper.

Since OCEAN is a peer to peer network searching is being done in a broadcast fashion, similar to Gnutella. This means that it can happen that a node is not able to find a good match since the other node is out of reach[5]. We are developing methods to combat these scalability issues.

2 Defining Requirements

If we want to distribute a task to a different node on the OCEAN network we must be able to specify the requirements of our task. This means that a buyer must be able to extract the minimum requirements for its task to execute. This information will be used to find all the potential resources that are compatible with the requirements.

The buyer needs to figure out the quality of the resources that are available. There are several techniques that could be used to identify the quality of a resource:

Resource Listing: The resource is defined in an XML document. Based upon the description we assign a quality to the resource. This will take a limited amount of overhead, but might not provide accurate or reliable information for every task.

Benchmark: The results of the sellers machine on a well known benchmark. This can be used by the buyer to get an indication of the quality of the resources of the seller. These benchmarks could be included in the OCEAN distribution.

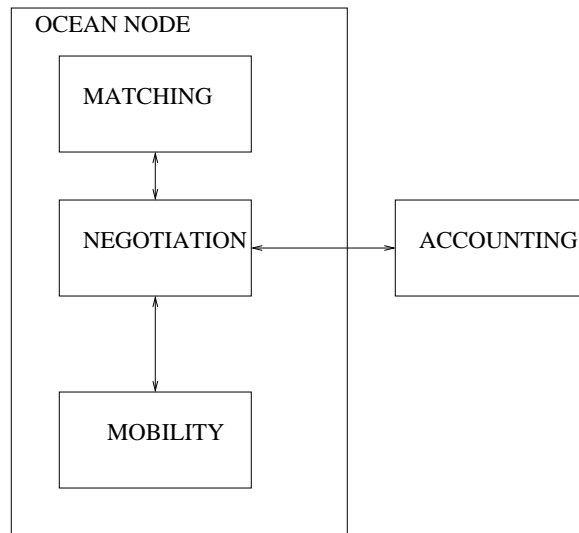


Figure 1: OCEAN Service Layers

Probe: This allows a buyer to run custom code on potential sellers at the time in question to get accurate indication of the capabilities of the resources offered. The probe is tuned to represent the task at hand and will be limited in the time it can execute and its communication abilities. An example of this would be a probe that renders only 1 frame of a movie scene. The task is clear representation of the entire, but will consume only a limit amount of time. This method is the most time expensive and would be most useful in very important or ultra compute-intensive tasks.

If we have these requirements for the possible sellers we can make an ordering based upon the resources that are offered for a given price. The negotiation component can now select the best offer and setup an agreement. Our matching system is built to provide buyers with enough exposure sellers to obtain good resources at competitive prices.

3 Locating Desired Peers : Design of Matching Network

3.1 Network Scenarios

The task of locating a suitable peer base which can provide the required resources its 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 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.

3.2 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[1] 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, super nodes can coexist with regular nodes who do not conform to the leaf protocol. The OCEAN Matching System protocol was built with these considerations to give flexibility to explore many possible network evolution schemes.

4 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*. 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.

4.1 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.

4.2 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 *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.

5 Discussion

The finder's fee system provides the incentive for nodes to "play nice," and redistribute requests to the peers that are most likely to be able to serve them. Meanwhile, the system for tracking and reporting paths allows nodes to properly maintain the success statistics that allow them to evaluate their peers' probability of success. Also the system for adding nodes to the peer list ensures that information about new sellers propagates through the system, and that the most direct paths between frequent buyers and sellers that effectively meet their needs quickly become available. Since the buyer for a short near-term computation is likely to accept the first offer that meets his needs, the statistics will tend to favor nodes that led to an offer as quickly as possible. This could be nodes that were often available, or nodes that forwarded requests to a large number of other nodes.

In any case, we expect that this system will lead to an adaptive, self-optimizing behavior, both at the level of the internal PLUM operation (which intelligently rewards its best-performing peers by sending them more business), and at the level of the human operators who are tuning the parameters of their own PLUM, since the operators who will be financially rewarded the most by the system will be the ones who do the best job of forwarding requests in a way that is most likely to lead to a successful transaction.

Since forwarding requests is cheap, there is little reason for a node operator not to do it, especially if the operator's node has already had a chance to send its own offer (if applicable) to the buyer, so that the buyer will see our own offer before any from our peers to which we have forwarded the offer.

We also expect to see a sort of emergent "clustering" behavior, where a set of nodes that frequently request and/or provide a given category of resources (say, a certain CPU) tend to evolve high ratings for each other, but not for other nodes, whereas another set of nodes that typically trades a different category of resources (say, machines with lots of memory) will likewise evolve high mutual ratings for each other, but lower ratings for other types of nodes. In other words, we expect that the network will autonomously tend to partition itself into distinct communities that trade highly efficiently within their specific category of trades.

We hope to see that these behaviors help evolve the network for the benefit of all nodes that want to participate.

References

- [1] Christopher Rohrs Anurang Singla, *Ultrapeers another step towards gnutella scalability*, <http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm>.

- [2] Rajkumar Buyya and Sudharshan Vazhkudai, *Compute power market: Towards a market-oriented grid*, Proceedings of 1st IEEE International Conference on Cluster Computing and the Grid, CCGRID2001, IEEE, May 2001.
- [3] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit*, The International Journal of Supercomputer Applications and High Performance Computing **11** (1997), no. 2, 115–128.
- [4] Ian Foster, *The anatomy of the Grid: Enabling scalable virtual organizations*, Lecture Notes in Computer Science **2150** (2001).
- [5] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, *A measurement study of peer-to-peer file sharing systems*, Proceedings of Multimedia Computing and Networking 2002 (MMCN '02) (San Jose, CA, USA), January 2002.