OCEAN: A Liquid Market For Grid Computation

http://www.cise.ufl.edu/research/ocean

Pinkeshkumar Desai <pbdesai@cise.ufl.edu>, Sama Govinda Ramanujam, Rajesh Devarajan, Michael P. Frank, Nachiket Acharya, Nitin Chawla, Chaitanya Chokkareddy, Sriram Kumar Nallan Chakravarthula, Murali Nagendranath, Pradeep Padala, Heeyong Park, Gargi Sur, Mark Tobias, Chi-Kin Wong, Bong Ju Yu

Department of Computer & Information Sciences & Engineering
University of Florida

Abstract

Distributed computing technologies have enormous promise for facilitating a wide variety of massively-parallel and mobile-agent applications. A very crucial advantage of distributed computing over traditional computing is its potential efficiency, making use of every spare moment your computer's processor would otherwise be idle. However, at present, these benefits are largely thwarted by the lack of any effective incentive system to encourage organizations to make their under-utilized resources available to run other organizations' distributed applications.

The OCEAN project aims to implement the total automation of remote, distributed computation in a way that promotes an extremely high level of liquidity (or equivalently, very low overhead and very high versatility), and furthermore, to do so in a way that will encourage the technology to become as widespread as possible and as quickly as possible in order to maximize its total economic benefit. Applications within OCEAN can purchase and utilize remote distributed computation resources on demand, while compensating the providers of these resources via a free-market mechanism. The OCEAN is based on a peer-to-peer distributed double-auction mechanism we have developed that can quickly find suitable matches among large numbers of computing resources and bidders for those resources. OCEAN is specifically designed to facilitate grass-roots growth and rapid, widespread adoption so as to hopefully become the de-facto standard platform for most grid-computing and mobile-agent applications.

Keywords: Distributed Computing Frameworks, Distributed agents and intelligent networks, Internet Computing, Computational Markets, Peer-to-Peer Technologies, Mobile Agent, System Security

1. Background

At no other time in human history have so many people had access to powerful computing resources. And yet, many of these resources lie idle for long periods of time as observed by many sources [1, 2, 3, 4, 5, 6]. 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. Conversely, there are many individuals and organizations who have intense computations to perform, but do not have immediate access to the resources that are required to execute them.

However, the Internet is increasingly omnipresent. The computing world can potentially be revolutionized if systems can transparently buy, sell, and use remote computing resources via the Internet. The system we envision should greatly increase the overall efficiency of the world's utilization of computing resources, thereby leading to increased productivity in many industries, since information technology and its applications currently drive a significant and growing fraction of the world's economy.

Achieving this goal is non-trivial, however, due to factors such as resource heterogeneity, cooperation with heterogeneous platforms, distributed ownership with different administrative policies and priorities, wide geographic distribution, and varying traffic/reliability/availability conditions. Different programming and communications standards, network and communication mechanisms, and the associated compatibility issues present another set of problems. In the rest of the paper, we present the architecture of OCEAN, and discuss various design issues. We conclude with a summary of the future work we are planning, to carry out OCEAN's mission.

2. About OCEAN

OCEAN (Open Computation Exchange & Auctioning Network) is a major ongoing project [3] at the University of Florida's Computer & Information Science & Engineering department, to develop a fully functional infrastructure supporting the automated, commercial buying and selling of dynamic distributed computing resources over the internet. The OCEAN project was started by a group of MIT students and Stanford alumni in 1997; in 1999 it moved, along with its founder's career, to U.F.

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 its users, the computational resources and tasks associated with them, and the underlying market mechanisms that facilitate trade.

The users of the system do not need to be human; in many instances, a “user” is actually a programmed agent acting on behalf of a human user [7,8]. Users of computational markets may take the roles of buyers or sellers. 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. When buyer and seller agents have negotiated a contract, the program requiring the computation (and associated state information) will be transported to the seller's machine, and the computation will be performed. The computational market’s role in all of this is to provide an environment in which these interactions may take place in an automatic way.

3. Related Work

There are many projects that are roughly similar to OCEAN, in that they provide infrastructures for distributed computing. Most of these, however, differ from OCEAN in that they do not provide an automated, open market having sufficiently low barriers to entry to encourage very widespread adoption (in our opinion).

GriPhyN [9], Globus [10], Beowulf [11], SETI@Home [12], Distributed.Net [1], Gnutella [13], IBM Aglets [14], Sun Grid Engine [15], XtremWeb - [16], Condor [17], Legion [18], OceanStore [19], and Ubero [26] are all examples of distributed computing systems that currently lack a market mechanism to compensate resource providers, which in our view severely limits the potential growth of their installed-base of deployed servers, as well as risking loss of resources to other systems, like OCEAN, that will offer compensation.

United Devices [21], Entropia [22], Parabon [24], Porivo [25] are examples of ventures that do sell distributed computing resources, but they are closed markets (controlled by these companies), and often may not give the best possible compensation to the resource provider. They also typically present a high barrier to entry to application developers, in terms of not providing a free, open, standard API that any developer can target and test. These systems are
also much less flexible than OCEAN, since resource usage must be typically contracted in advance, and cannot be purchased on-demand dynamically by arbitrary applications. They are also less scalable than OCEAN, because of the centralized nature of their resource management, as contrasted with OCEAN's peer-to-peer distributed auction system.

Out of all the current projects we know about, the Economy Grid [20] / Compute Power Market (CPM) [23] projects of Rajkumar Buyya and colleagues is the most similar to OCEAN in its goals and spirit. However, even this project currently lacks an auction mechanism to set prices. We have completed a prototype implementation of an auction mechanism, and have initiated discussions with Buyya's group about possible collaboration to make OCEAN's auctioning framework available as a market mechanism for CPM, and also hopefully to cooperate with CPM on market and API standards development.

4. Support for multiple grid environments:

Various Grid technologies in the market offer different (and often competing) solutions to certain basic problems, such as transfer of code to new hosts, and communication between distributed components of an application. However, OCEAN is designed to be compatible with all of them. On top of the grid technologies sit numerous technology-specific grid applications (consuming resources) and compute servers (making resources available to the grid). But, from within any of the grid environments, developers will be able (using web-service model) to access any desired features of OCEAN’s infrastructure for peer discovery, resource auctioning, automated negotiation and payment accounting. So, OCEAN can then be used as a huge automated clearinghouse for distributed resources, which can then be utilized by applications using any grid technology. Refer Figure [1].
**GLOBUS:**

The primary goal of the Globus project is to provide basic technology that enables entirely new classes of applications. The Globus Toolkit is used by many organizations to build computational grids that can support their applications. The open source Globus Toolkit includes tools and libraries for solving problems in the areas of Security, Communication, Information, Infrastructure, Fault Detection, Resource Management, Portability and Data Management.

Globus software development has resulted in the Globus Toolkit, a set of services and software libraries to support Grids and Grid applications. The Toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. Grid builders are using Globus services to create production Grid computing environments.
.NET Platform Support

The .NET framework released by Microsoft in the first quarter of 2001 is a runtime environment that provides a simple programming model, safety and security, powerful tools support, and help with deployment, packaging, and other support. The .NET Framework has two main components: the common language runtime and the .NET Framework class library. The Common Language Runtime is an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict safety and accuracy of the code. The .NET Framework class library is a comprehensive, object-oriented collection of reusable classes that can be used to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET and Web Services.

ADK Platform Support

ADK (Agent Development Kit) is a commercially available tool kit developed by Tryllian, a company focused on the development of mobile agents and mobile agent environments, providing products, solutions and services. The Tryllian ADK allows application programmers to define the components required to build an agent-based application.

The ADK consists of two parts, the Agent Foundation Classes (AFC) and the Agent Runtime Environment (ARE). Some of the features provided by ADK are

- An integrated set of sophisticated tools that enables users to develop and deploy mobile agent applications;
- Provides libraries of standard behavior in the form of pluggable tasks;
- Includes software for running and managing complete mobile agent infrastructures;
- Features autonomous task execution.

OCEAN Grid Runtime Environment:

The OCEAN Runtime component is logically comprised of the Task Spawning & Migration (TSM), Security and Communication components. Upon successful negotiation of a contract for the execution of a computation, the TSM component spawns or migrates client tasks
to remote nodes in secure fashion, using services provided by the Security and Communication components.

The Security component’s role is to ensure the integrity of the contract document, data and code as well as preserve the confidentiality of the seller’s data files, private keys and other sensitive information that can not be comprised. The component performs its role in a transparent manner, and its implementation is a logical abstraction that is invisible to higher layers. Hence the Security component forms a part of the core services provided by OCEAN, and the application developer is freed from concern over providing transmission of OCEAN tasks through insecure networks. The authentication and integrity checks of the buyer’s identity and executable code ensure a safe environment within which an OCEAN task executes, incapable of unexpected or hostile activity while running on an OCEAN host.

The Communication component is primarily responsible for data delivery to the other OCEAN node(s), and configurable through the Node Configuration / Maintenance Operator Interface.

5. OCEAN Architecture

System Architecture

Coularis et al. [27] define a distributed system as "one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages." The OCEAN software system fits this description. The most common type of architecture for a distributed system is the client-server model.

The high level interactions among nodes in the OCEAN system are depicted above in Figure [2]. As can be seen from the figure, all nodes are connected to one another in one of two ways. Either they are connected directly via the Internet, or indirectly through a node that operates as a gateway to other nodes in an intranet behind a firewall. Note that specific types of node indicate the role that each one is playing at a given point in time. For example, a node that is acting as an auction server at some point in time can act as a
server performing a computation at another time. The only exception to the completely distributed nature of the OCEAN system is the presence of a centralized accounting system.

Figure [2]: The OCEAN System

A client-server type of architecture for OCEAN would employ a server process, or many processes to handle all auctioning, negotiation and accounting between buyers and sellers (clients). The potential volume of users that are able to participate in the Network is enormous; anyone with a computer may join. Thus, the likelihood of encountering scalability problems is a huge factor in the system design. A system is described as scalable if its effectiveness is not compromised upon significant increases in the number of resources and the number of users [27]. The client-server model can present scalability problems for two reasons. One, the server may be
a bottleneck when the user volume is large. Two, a server can act as a single point of failure. For these reasons, a client-server type of architecture is unsuitable for the OCEAN system.

While the Compute Power Market [23] approach improves fault tolerance over a Client-server approach, it does not achieve maximum scalability. The fact that there are a finite number of market servers has implications—if the number of users increased dramatically, it is possible that there are not enough market servers to keep the market operating efficiently.

The OCEAN system attempts to improve scalability by implementing a distributed, peer-to-peer based architecture. The OCEAN software system, developed in the Java™ programming language, is packaged as a single unit, and can be installed on any computer with a Java™ Virtual Machine (JVM), thereby causing the machine to become a node in the Network. Any node can perform any function in the Network. Thus any node can hold an auction, execute a buyer's computation, serve as an application development environment, or an application launch point.

**Node Architecture**

This section discusses each component of the OCEAN node architecture as shown in below Figure [3]. As of this writing, only the Auction component has been implemented in the current OCEAN prototype generation. We are currently planning to finish the implementation of the other components by December 2001.

As shown in the Figure[3], there are three external APIs: the Trade Proposal API, the Application Programmer API and the Node Configuration and Operation API. These provide human traders the mechanisms necessary to interact with the OCEAN node software and with the system as a whole. All these APIs are examined further.
Trader Component:

This component acts as an interface between the human participants and the rest of the node components. Note that human interaction is accomplished through use of the external APIs. In effect, the Trader subsystem acts on behalf of the human users to allow them to communicate with the Negotiation, Auctioning, and Task Spawning and Migration subsystems. The Trader component is used as a Buyer, through the Trade Proposal API, by an application-specific task...
which wishes to spawn new remote tasks, or to migrate itself to a remote node. In this case, the node is acting as a launch point for the remote tasks. The Trader service enters a bid into the auction system, negotiates a contract with a suitable seller, and then initiates task spawning or migration to utilize the purchased resources.

The Trader component is also used as a Seller, through the Node Configuration/Operation API, by built-in (or custom) node operation software. Once configured with an appropriate sales policy, the Trader component advertises the node's availability to the auction system, negotiates sales, and accepts tasks to execute.

Auctioning Component

The main focus of this the auctioning process is to match traders with the best possible trading partners. For buyers this means finding sellers who not only offer the best prices, but also provide the resources that the buyer is requesting. For sellers, the Auctioning system is responsible for finding the highest paying buyers, without exceeding the resource capabilities of the seller.

It has been noted [23, 31] that the trade proposals in a computational market can be complex. The buyer may have strict requirements regarding hardware and software resources that must be satisfied. For example, a buyer may require a secure environment to protect sensitive data. Or, they may require a high-speed network connection because a large volume of data is to be transmitted with a task. A buyer may also need certain hardware, a sophisticated graphics board, for example. Similarly, sellers may offer many differing types of services; they may have a vast array of hardware, software and data resources at their disposal.

In order to handle the complexity of trade proposals, expressive data description languages were developed in the Extensible Markup Language, or XML. These languages were developed according the XML Schema specification [32], defining the format that trade proposals assume. Application Programming Interfaces (APIs) were developed for the Java™ programming language that allow OCEAN traders to define the resources they request or provide. In addition,
the APIs allow traders to express the prices at which they are willing to trade. In XML vernacular, these APIs generate the XML content that defines a trade proposal. The auctioning system is the component that consumes the XML-based trade proposals. It parses all relevant trading information for each incoming proposal, and attempts to find a matching trader(s). Distributed algorithms were implemented to perform this function. The auctioning process occurs in a distributed, peer-to-peer framework among many nodes in the OCEAN system. Potential trading partners, before being matched, must have their resource descriptions checked to verify that the traders are compatible in terms of resource request and provision.

Finally, a simulator was developed to test the performance of the auctioning system relative to configurable parameters.

**Peer List Update Manager Component (PLUM)**

The main purpose of the PLUM is to ameliorate the problem of directory lookup operations mentioned by Parameswaran et. al. [29]. More specifically, in the distributed system it is impractical for nodes to communicate with one another in a broadcast fashion. Instead, they must communicate with a selected few peer nodes. It is the responsibility of the PLUM component to determine a list of peer nodes with which a node will communicate.

![Figure 4: The PLUM Component](image)

The PLUM maintains a list of addresses of other OCEAN nodes (peers) that it knows about, and associated status information about that node (e.g. present & average accessibility,
availability, intercommunication bandwidth & latency, etc.). Based on the node administrator's preferences, the PLUM periodically updates its peer list.

**Negotiation Component**

The Negotiation component allows traders to automate the process of negotiation. The Auctioning system determines who are potential trading partners. However, the negotiation system provides the means for traders to agree on the terms of a contract. This includes resolving conflicts that arise when a trader has many potential trading partners. The negotiation component tries to automate the process of negotiation as much as it can. In any event, it also provides methods so that humans can intervene and take over the job of negotiation, if necessary, for a high-value transaction.

Like most node components, it uses the security and communications component for communications to other nodes on the network. Like all node components, it supports a configuration & maintenance interface for use by the node operator. Its network communications are primarily with the Negotiation components of other nodes. One possible procedure for negotiation is as follows, though others may be explored as well.

1. The negotiation component receives a list of peers that are viable candidates for negotiations from the Auction component via the Trader Component.
2. Based on a policy set by the Trader layer, the negotiator chooses which peer to negotiate with. In extreme, high-value cases, a human user may be prompted to manually choose the node with which negotiation is to be done.
3. Negotiations take place between the nodes. All the negotiations are in the form of XML documents. The XML schemas for negotiation will be part of the OCEAN standards.
   a. If the buyer is satisfied with the seller's initial trade proposal he sends a contract to the seller. If he is not satisfied, he sends an XML document to modify a proposal to the seller based on the proposal it receives from the seller.
b. The seller looks at the modified proposal document and decides if it can do business with the buyer based on the new proposal. If so, it sends back a modified proposal to the buyer. If not, it sends a document to reject the business to the buyer.

c. If the buyer receives a reject business document, it moves onto the next node in the list. If it receives a new proposal from the seller according to its specifications then it sends a contract document which contains details about the deal and which has to be signed by the seller.

d. When the seller receives the contract document it checks it to validate it and if everything is fine, it signs the contract and sends it back to the buyer. If the validation of the contract fails, the seller sends a document, which indicates that the contract is wrong and specifies the mistakes found by the seller in the contract.

e. If the buyer receives a signed contract we move onto step 5. If we receive a contract-failed document we either correct our mistakes if there are any or we send a reject business document and move to step 4.

4. If the nodes do not reach an agreement, we move onto the next most-favored node in the list of potential trading partners and repeat step 3.

5. If the nodes reach an agreement, we return the name of the seller and the signed contract to the Trader component so that it can instruct the Task Spawning and Migration component to migrate the job.

For the first version of the product there will be just a simple version of the Negotiation Component. In the future versions more research will be done on conducting multiple simultaneous negotiations among multiple nodes at the same time. Also, research will be done on including intelligence in the negotiation process so that the negotiation component learns from its previous negotiations. Some jobs may take a lot of time and might involve large investments, and require sophisticated negotiation. On the other hand, some deals may be routine short day-to-day activities involving meager investments, and may require only very simple negotiation policies.
Task Spawning & Migration Component

The Task Spawning and Migration (TSM) component is responsible for the proper dissemination and execution of computing tasks in the Network. When a contract for the execution of a computation is successfully negotiated, the TSM subsystem will spawn computing tasks on remote servers through the Security and Communication layers at each node. When the server receives the task(s) it is to execute, they must be started and run to completion. Also, that task may spawn new tasks and so on. A task may also migrate from one node to another, if necessary.

The TSM obtains the Task Sender information in the OCEAN Task Object from the Trader Layer subsystem and migrates the OCEAN tasks from the origin (Task Sender, likely Resource Buyer) to the destination(s) (Task Receiver, Resource Seller) through the Communication/Security Layer. This code (compressed file) includes a task description, the executable classes (Java byte code in a jar file, or possibly a .NET CLR assembly) that comprise the actual work that needs to be performed and authentication information in the form of digital signatures and certificates. The TSM on the receiver side sets up a secure sandbox environment and allocates computer resources in the OCEAN node for running the code.

The TSM internal system has both a Sender and a Receiver component. The Receiver component of the TSM Layer essentially runs in an infinite loop, listening for incoming requests like a server or daemon. The Sender component of the TSM Layer (at the Task Sender Node) prompts the Receiver component of the TSM Layer (at the Task Receiver Node) to download the java class files from the Sender. After receiving the jar file, TSM will extract the compressed file and then start execution of the task. The communication of any computational results back to the task's originator is the responsibility of the application program.

Security Component

Security is a major concern in computational markets. Both the parties to a transaction should have confidence in the integrity of the documents. If a buyer has potentially sensitive
information associated with its computation, the sellers must ensure that the integrity of data is maintained. On the other side, the security of the seller’s data files, private keys and other sensitive information must not be compromised. The OCEAN infrastructure requires that tasks should be able to communicate securely (if at all) with any other tasks that are part of a given distributed job. However, the security aspect of the communication between peer nodes is hidden from the higher layers of the OCEAN architecture.

The security mechanism uses symmetric and asymmetric cryptographic techniques to maintain confidentiality of information; CA certificates, digital signatures and Message Authentication Codes (MAC) or hash functions to ensure data integrity and authentication and security managers to implement access control functions and node security.

The Auction, Negotiation, PLUM, Communication systems and Node Configuration and Operator API are the consumers of the services offered by the Security system. The security system offers various levels of security, which can be configured by the Node Configuration and Operator API depending on the quality of service required.

**Communication Component**

The Communication Component is responsible for any communication to and from any OCEAN node. The component directly interacts with Security and Naming components on the same nodes and the Communication Components on other nodes. The component is primarily responsible for data delivery to the other OCEAN node(s). The communication component can be configured through the Node Configuration / Maintenance Operator Interface.

The Security component is the most direct customer for the service offered by the Communication component. As a general rule, whatever is received as data, the communication system passes it to the Security system. The communication system uses the services from the Naming system but does not provide any services to the Naming system. The job of the communication component is to take data and URL from the security system, pass the URL to the
Naming system and obtain the immediate destination, and pass the data on to the IP network to the destination.

Figure [5]: Inter-process Communication

The communication system can handle secure and non-secure communication for the security system. The node configuration and operation API can configure the Communication component to block/unblock a node, set the maximum bandwidth allowed to the OCEAN software, and obtain information on packets coming in and going out of the node.

Various technologies like Juxtapose (JXTA) [34], SOAP [36] and UDDI [37] are potential technologies that may be used in implementing Communication Component needs in OCEAN.

Naming Component

The Naming component is responsible for name resolution in the Network. The Communication component relies heavily on this component. In particular, it is undesirable for components to refer to OCEAN nodes strictly by their IP addresses. The reason for this is that IP addresses can change, not to mention that they can be obscured by firewalls. In addition, it is desirable to not only locate computers in the Network, but it is also necessary to identify the locations of other entities so that they may be contacted. These include computing tasks, data, and resources. The Naming and Communication subsystems provide this functionality: referring to and contacting any OCEAN entity on the Network. Names of OCEAN entities are syntactically
defined as extensions to the URL/URI standard. Any resource or process can be located with a URL. This is achieved by using a path-naming scheme as below:

```
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>
```

Each hostname or IP address after the first one may be a private hostname (or IP address on a private intranet) of a machine reachable from the preceding host. By using names like these, the communication component uniquely identifies the target machine and target entity, whether or not it has a public, static IP address. Messages are forwarded along the designated path until they reach the eventual destination. Essentially, we are 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.

**Central Accounting Server**

It is the responsibility of the Central Accounting Server (CAS) to log transactions of the business that occurred between the buyer and seller. Every trader on the OCEAN system has an account on the CAS having a stable, fault-tolerant storage capacity, which is similar to a conventional bank account. Many transactions may be too small to be worth executing on traditional financial networks (for example, if a user invokes an application that just runs for 10 seconds on several machines to perform a quick calculation), so micro-payments are accrued to or deducted from the trader’s account each time a successful transaction takes place, without accessing the external financial networks. This account information is used to make real world transactions with the existing financial networks to debit or credit the traders only periodically or when the balance in the trader’s account exceeds a particular limit. A trader is not allowed to participate in any transaction as a buyer if the trader's balance after the transaction is below a certain (negative) credit limit.
Though most of OCEAN is distributed, the CAS is centralized to simplify the financial operation of the system. If this function were distributed amongst OCEAN nodes, every trader would need a direct merchant account connection with the financial institutions, which might significantly reduce the number of participating traders. Also, there is the issue of logging all the transaction history. There needs to be some secure location where all the account information and transaction history can be stored for archival purposes in case a dispute arises over payment. When there is a centralized control over all the payment transactions, the OCEAN administrators, maintaining the CAS, may ask for a small fee for each transaction. There is no need to implement the CAS functionality on a single server. In fact, suitable fault tolerance methods can be used and the CAS can be implemented on a cluster of computers for acceptable performance, security and availability.

**Local Accounting Component**

It is the job of the local accounting component to ensure that every successful negotiation and subsequent transaction gets logged on to the Central Accounting System. It also maintains the node’s own transaction history.

**OCEAN Client Task API**

The OCEAN Client Task API provides the primary interface to OCEAN resource buyers. The API will be developed in stages, with the preliminary version concentrating on essential, deliverable, functionality such as submission of tasks; task and account state querying; and task communication and control. Rather than allow the application to access subsystem functionality directly, the API will provide interfaces that abstracts the underlying subsystem interfaces. For example, rather than allow tasks to access, directly, the functionality provided by the Communication system, the API will, instead, route all communications through the Communication system. This creates a level of abstraction that facilitates modularization and hides the internal OCEAN core architecture from applications, in case the internal architecture needs to be changed later.
As OCEAN applications need to generate auction proposals, the API provides functionality that allows developers to generate XML documents that describe their resource requirements (buyers) and available resources (sellers). These documents will then be propagated as the buyer or seller's trade proposal. The API also provides functionality for an application to query its account state (balance, line of credit, etc.) so it can determine, dynamically, how it chooses to precede.

**OCEAN Simulator**

The OCEAN Simulator is an environment that replicates the functionality of the OCEAN architecture and supports its Client Task API, while running applications on a single machine. This gives OCEAN application developers a service that permits them to test their applications in an OCEAN-like environment where they may determine if their applications function properly, without fear of delinquent jobs request an inordinate amount of resources on behalf of their owner. This protects the owner’s line of credit on the OCEAN system.

Essentially, the Simulator creates a pseudo-distributed environment by replicating nodes as threads on a single machine. Thus, a developer may test his application on the Simulator to verify results, optimize execution, and estimate an upper bound on the cost of running the application on OCEAN by gathering information on the number of tasks generated, and their run times. The Simulator replicates the OCEAN Client Task API, so applications may be ported to the Simulator without any significant changes.

Since the Simulator is not part of the OCEAN system proper, it can be developed independently. This grants tremendous freedom for the designers to extend the Simulator concept to provide a truly distributed environment, and not just a simulated one. There are applications that simply cannot be executed on one machine due to, for example, extreme memory or real-time performance requirements. Future development of the Simulator might allow for applications to provide a list of machine addresses, to which the Simulator may automatically distribute tasks, coordinate communications, and gather results, etc. The OCEAN developers recognize the
possibility of several competitive distributed computing markets, and, as such, the Simulator is a
service intended to aid an application developer and, in the process, make OCEAN a more
attractive distributed computing market.

6. Conclusion

Currently OCEAN is in its design and prototype implementation stage at the Computer &
Science Information & Engineering department at the University of Florida. The prototype
auction component has been nearly completed, but other components remain in an early stage of
implementation. However, we intend to complete the prototype implementation and release a beta
version of the software for public testing by the end of the second quarter of 2002.

Like Napster [30], peer-to-peer communities can grow by themselves as noted by
Parmeswaran et al. [29]. The distributed nature of the OCEAN system provides for maximum
scalability and robustness for its users. The growth in the user community doesn’t raise scalability
issues, as the number of service nodes also increases. The architecture of the OCEAN system
aims to address scalability and fault-tolerance issues by implementing a distributed peer-to-peer
architecture. The core OCEAN system is currently developed in Java and packaged as a single
unit. The system can be installed on any machine with JVM with ease. However, Microsoft's
.NET framework is also being considered as a possible platform technology for future releases of
OCEAN, especially since visual J# allows Java apps to run on .NET.

On the one hand there are legions of computers with idle computing power, and on the
other end there are technologies like mobile devices whose growth is hampered by lack of
computing power. The OCEAN aims to help achieve a balance in the computation paradigm.

Seller trade proposal document

<Seller>

<SellerID>seller26346</SellerID>

<SaleableResourceUnits>

<!-- SaleableUnit with one CPU -->

<SaleableUnit>
<Resource>
  <CPU>
  <Platform>
    <Architecture value="x86"/>
    <OS value="Linux" version="2.4"/>
  </Platform>
  <ClockSpeed value="900" units="MHz"/>
  </CPU>
  </Resource>

<!-- The Ask price is
  0.01 US Dollars per second
-->

<AskPrice>
  <TimeDependentPrice>
    <CurrencyAmount value="0.01" Currency="USD"/>
    <Timespan>PT1.0S</Timespan>
  </TimeDependentPrice>
</AskPrice>

</SaleableUnit>

<!-- end SaleableUnit -->

</SaleableResourceUnits>

</Seller>

Buyer trade proposal document

<Buyer>
  <BuyerID>buyer73908</BuyerID>
  <BidItemType>
  <TaskList>
<TaskListItem>

<!-- A Bid on one Task -->

<Task>

<!-- unique task id as specified by

    OCEAN naming system -->

<TaskID>www.node1.com:400/jobid/taskid</TaskID>

<!-- Task is requesting a CPU and Memory -->

<RequestedItems>

<Resource>

<CPU>

<Platform>

<Architecture value="x86"/>

<OS value="Windows" version="2000"/>

</Platform>

<ClockSpeed value="800" units="MHZ"/>

</CPU>

</Resource>

<Resource>

<Memory value="256" units="MB"/>

</Resource>

</RequestedItems>

</Task>

<!-- Bid Price is .02 US Dollars per second -->

<BidPrice>

<TimeDependentPrice>

<CurrencyAmount value="0.02" Currency="USD"/>

<Timespan>PT1.0S</Timespan>

</TimeDependentPrice>

</BidPrice>

</TaskListItem>
6. References


[12] The SETI@home homepage, http://setiathome.ssl.berkeley.edu


[22] Entropia Inc. – http://www.entropia.com


[33] www.cise.ufl.edu/research/ocean/language/schemas

[34] Juxtapose – http://www.jxta.org

