Océano – SLA Based Management of a Computing Utility

K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, and B. Rochwerger
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, New York 10598, USA
Contact address: gsg@us.ibm.com

Abstract

Océano is a prototype of a highly available, scaleable, and manageable infrastructure for an e-business computing utility. It enables multiple customers to be hosted on a collection of sequentially shared resources. The hosting environment is divided into secure domains, each supporting one customer. These domains are dynamic: the resources assigned to them may be augmented when load increases and reduced when load dips. This dynamic resource allocation enables flexible Service Level Agreements (SLAs) with customers in an environment where peak loads are an order of magnitude greater than the normal steady state.

Keywords

Cluster Management, Service Level Agreements, Network and System Monitoring, Computing Utility

1. Introduction

Océano is a prototype of a highly available, scaleable, and manageable infrastructure for an e-business computing utility. It enables multiple customers to be hosted on a collection of shared resources. At any point of time each resource is assigned to only a single customer. That is, the hosting environment is divided into secure domains, each supporting one customer. These customer domains are dynamic: the resources assigned to them may be augmented when load increases and reduced when load dips. This dynamic resource allocation reduces hosting costs while providing a mechanism to guarantee contracted Service Level Agreements (SLAs).

In e-business hosting, customers increasingly require support for peak loads that are an order of magnitude greater than those experienced in normal steady state [16]. A common commercial hosting model is colocation model, in which dedicated resources are permanently assigned to each customer. In this model, supporting peak workloads requires significant resource overprovisioning. Océano provides a more cost effective alternative by automatically and dynamically reassigning resources to meet demands as they occur.

0-7803-6719-7/01/$10.00 (C) 2001 IEEE

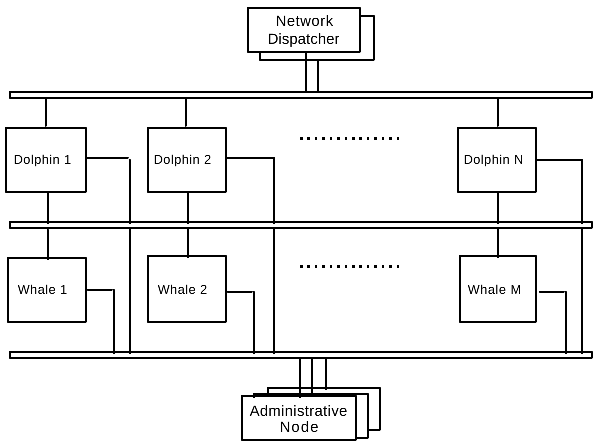

Figure 1: Overview of one domain in Océano

Océano manages the resources of the computing utility so that each hosted customer is furnished the resources necessary to provide a contracted level of service as specified by a Service Level Agreement (SLA) [5]. Monitoring agents issue events when thresholds are exceeded. Events are correlated to identify root causes, which are reported to a central *Resource Director* responsible for planning resource allocation and recovery actions. The main controls available are load sharing via dynamic allocation and de-allocation of servers, and throttling incoming requests. Underlying subsystems provide the mechanisms for managing resources and shifting them from one customer domain to another in real time (minutes) without compromising security requirements.

Other approaches to sharing resources among multiple hosted customers or applications [3,7,14,15,16] focus on sharing the CPU cycles of single servers. In contrast, Océano focuses on sequential sharing at the granularity of whole servers, and the management of a whole farm of servers. Approaches that also sequentially allocate whole servers allow only static server allocation. These approaches, unlike Océano, make no attempt to modify the computing environment to satisfy the allocation (for example, by installing an operating system or by changing the network configuration). Rather, they allocate the server as is. A matchmaking algorithm [13] must ensure that the environment is suitable. In general, these approaches create virtual domains without providing network isolation. IcorpMaker [15] does provide isolation via virtual private networks; Océano does so via virtual LANs. Finally, the Galaxy project [18] focuses on providing a variety of tools to build Windows-NT clusters for multiple different purposes. It does not provide SLA monitoring or automatic resource reprovisioning in response to performance bottlenecks, only in response to failures. Océano provides a unique and more comprehensive combination of technologies to address a number of issues ignored by these approaches, including SLA driven monitoring, event correlation, network topology discovery, and automatic network reconfiguration.

In this paper, the term customer refers to a third party hosted on an "e-business computing utility." The utility is composed of a server farm that is managed by the Océano distributed software. Customers are supported by dynamically dividing the physical infrastructure into customer domains. The implementation of Océano described here assumes that the physical infrastructure consists of 3 tiers: (1) front-end IP sprayers for load balancing (e.g. Network Dispatchers [9]), (2) a large pool of allocable *"Dolphin"* servers, and (3) a pool of fixed *"Whale"* servers, all interconnected by a switched network. Figure 1 shows one domain. Each domain contains the servers currently allocated to a particular customer. The Dolphins can be allocated to a domain and later reallocated to another. On the other hand, the Whales are permanently assigned to a domain. In practice, a customer's database would reside on the fixed Whale servers. All servers are connected to an administrative, or management, domain. A small set of dedicated nodes execute Océano management applications. Management and monitoring agents may reside on customer assigned Dolphin and Whale servers.

The rest of this paper provides an overview of Océano. Section 2 describes how Océano uses SLAs to identify metrics to monitor, how they are monitored and how monitored events are correlated. Section 3 describes the Resource Director. The mechanisms to manage bandwidth, servers, and customer data and applications are described in Section 4. Section 5 describes the underlying infrastructure and configuration information. Section 6 reports on the status of our prototype implementation and reports some initial performance statistics and identifies some areas of future research. Finally, Section 7 concludes.

2. SLA based Management

Key to all resource management in Océano, is the management of customer SLAs. This is achieved by managing sets of metrics derived from customer service level contracts. Two components, shown in Figure 2, the Aggregator, and the SLMonitor, implement this function. The Aggregator builds customer Service Level **(**SL**)** meta-events from the data received in events generated by monitoring agents distributed throughout the Océano infrastructure. The SLMonitor monitors the state of each customer domain by correlating system events and SLA violation events.

Correlation scenarios identify root causes and other potential future implications of the events. If application performance or other metrics fall below contracted limits, the SLMonitor notifies the Resource Director (described in Section 3) to initiate corrective actions. The Resource Director uses customer domain and system state data to decide when, and if, to dynamically shift resources between customers. In this way resources that are not currently needed to meet a specific customer SLA, can be reallocated to another customer experiencing high workload.

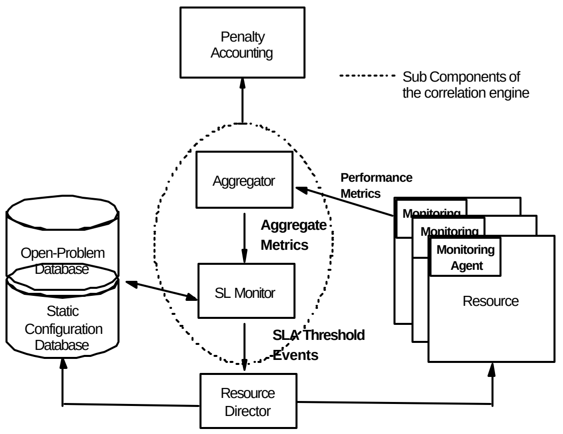

Figure 2: Relationships between SLA monitoring and enforcement components.

2.1. SLA Metrics

Service level contracts specify requirements and goals for availability, response time, server load, assigned resources, and output bandwidth. These specifications generate threshold values that are used as event triggers. There is an important distinction between what the violation of a requirement and a goal implies. Requirement violations are used to trigger resource allocations, and initiate penalty charging. Goal violations trigger allocations, but do not cause penalties to accrue. SLAs also specify a priority class (such as Gold, Silver, or Bronze) that indicates a customer domain's priority in relation to other customer domains. These priorities can be used when resource allocation conflicts arise.

The specific metrics to be utilized are determined by the SLA requirements/goals specified in the contract. The SLA metrics chosen should be measurable and reliable. For example, we may use a *DB Response Time* threshold if most DB queries are similar in terms of processing time. On the other hand, if base query processing time varies widely the *DB Response Time* metric should not be used to trigger resource allocations. Table 1 shows the predefined metrics that are currently used.

Many of the metrics are aggregated over a customer domain. To account for capacity differences among servers, performance metrics are normalized. The polling interval and the interval over which metrics are aggregated can be set individually for each domain and metric. A simple smoothing algorithm is used to remove temporary peaks in the data. Note that the *Overall Response Time* metric is computed using the response time data collected from the customer domain's active server set in combination with any wait time spent in the incoming throttling queues and TCP stacks.

2.2. Monitoring Agents

Monitoring agents local to the managed servers collect server load and performance metrics. These agents may monitor system level information or may monitor application or middleware specific metrics. For example, a monitoring agent for the Apache Web Server was implemented as a “plugin”. Currently the agent is used to obtain HTTP request response time (URL filters identify requests of interest) and output bandwidth. Monitoring behavior can be dynamically altered by the SLMonitor by resetting such parameters as thresholds for monitored metrics, changes in periodic event reporting intervals, and so on. Agents issue event notifications to the SLMonitor (see Figure 2) whenever thresholds are exceeded.

Table 1: Predefined metrics in Océano

Metric	*Definition*
Active Connections/server	The average number of active connections per normalized server across a domain
Overall Response Time	Average time it takes for any request to a given domain to be processed
Output Bandwidth	The average number of outbound bytes per second per normalized server for a given domain
DB Response Time	Average time it takes for any request to a given domain to be processed by the back-end DB
Throttle Rate	A percentage of connections disallowed to pass through Océano on a customer domain
Admission Rate	The complement of domain throttle rate, i.e. 1-T
Active Servers	The number of active normalized-servers which service a given customer domain

2.3. Correlation Engine

Océano contains a state based rule engine that correlates network and SLA events. For scalability, the server farm can be broken up into multiple segments allowing multiple engine instances to correlate different parts of the farm. Further, engine instances can be organized into a hierarchy so decisions that cannot be made locally can be forwarded to a higher level. Engine instances could also be split by problem type. In this case, low-level problems can be handled by one instance while another handles high-level problems. Communication between the levels is done using internal events and the open-problem database.

The correlation engine [1] supports a layered model-based language. Some of its features include:

1. A scenario set that encapsulates individual component behavior is developed for each abstract entity defined.
2. Entities communicate by publishing and subscribing to internal events that propagate through the implicit dependency chain.
3. Scenario event sequences can contain interspersed method invocations. This allows for the collection of additional data, early initiation of corrective action, or the analysis of state information before the complete set of required events have arrived.
4. Rules that represent alternate solutions to the same set of events are grouped and ranked in priority order.
5. SLA violation detection is integrated into the correlation rule sets. Low-level network faults can be easily correlated with high-level SLA violations.
6. A built in problem database provides support for long-lived error conditions. Dependent problems are automatically canceled when the root problem is canceled.
7. A built in event type that collapses multiple events of the same type to a single event specification is available. This can be used to recognize when some % of resources in a resource-set generate a specified event in a given time interval.
8. Ordered and unordered event arrival requirements are supported.
9. Multiple rearming methods are provided.

The scenarios written for the Océano server farm are generic and do not depend on the customers being hosted or the size of the farm. Although one of our major goals was to make scenario writing as simple as possible, an Océano environment can be installed and run without the need to develop any custom scenarios. A large set of default scenarios are built into the correlation engine. Farm configuration and customer specific information referenced in the scenarios is pulled from the configuration database when needed.

To avoid resource allocation thrashing, new allocations cause resource deallocations, for the given customer domain, to be temporarily suspended. This is achieved by ignoring minimum threshold events for a period of time. There is a global default for the length of the deallocation suspension, but it can be overridden for each customer domain. The reverse function is also provided; allocations are temporarily suspended after a deallocation occurs. The effectiveness of these approaches has not yet been studied in detail.

3. Resource Director

The Océano Resource Director is an event driven system responsible for selecting corrective actions when a service level threshold is exceeded or when hardware or software failures occur. Possible actions include modifying server-set assignments, throttling incoming request streams, initiating recovery actions, and issuing

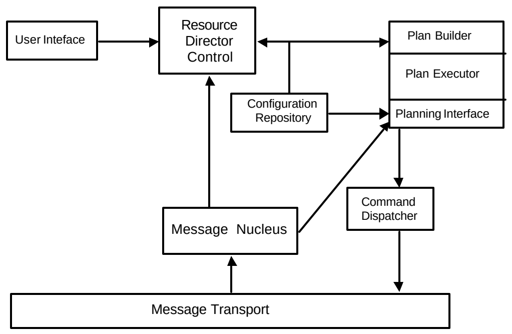

Figure 3: Overview of the Resource Director architecture

administrator alerts. An overview of the Resource Director’s architecture is presented followed by a description of its planning technique.

3.1. Architecture

The Resource Director is an event-driven system: events spawn the instantiation of configuration or recovery plans, which execute coordinated commands to other Océano components. A broad architectural overview of the Resource Director is given in Figure 3. Océano components generate events, sent as messages, to the *Message Transport*, a pub/sub communication layer. On receiving a message, the transport hands the message to the *Message Nucleus*, which determines to which component of the Resource Director to route it. The *Resource Director Control* coordinates both automated and user initiated planning, and initiates recovery or reconfiguration activity based on messages from the Message Nucleus.

The *Planning Component* is the central unit that builds and executes plans for system recovery and SLA-based resource planning. It has three subunits: the *planning interface*, the *plan builder*, and the *plan executor.* The planning interface is the intercept point for orders that a plan needs to be built. These orders are usually generated by events received by the Resource Director Control by way of the event correlator. The plan builder responds to information that either systems recovery or SLA-based reconfiguration planning needs to be done, and constructs a plan. Finally, the plan executor executes and manages concurrent plans.

As plans execute, commands are dispatched to other Océano subsystems through the *Command Dispatcher*. Command completion messages are received through the Message Nucleus. A Configuration Repository, a cache of system-wide configuration information from the Océano configuration database, supports planning. Critical information supplied by the Configuration Repository includes identification of all customers, available resources, and SLA thresholds. Finally, the *User Interface* shows the progress of all plan activity occurring in the Resource Director.

3.2. Recovery Planning

Plans take the form of graph hierarchies, wherein the nodes represent distinct configuration activities, e.g. add a Dolphin, set admission rate, etc., and the arcs represent precedence dependencies. The construction of these hierarchical phases is based on a set of scripts along with the circumstances or needs of the system.

For example, plans for failed system resources, including Océano components, are largely based on a set of well-known recovery scripts. For instance, for the recovery of a system resource such as a Dolphin, a replacement needs to be found and put in place. This type of planning is relatively straightforward.

Resource configuration planning due to performance anomalies is more intricate. Depending on the bottleneck type, the Resource Director devises a recovery action appropriate and optimal for the customer. A number of factors are taken into account for the recovery action. Examples of the factors checked by the planning system include: customer domain priority, minimum and maximum customer domain input request rates, minimum and maximum number of allocated servers, and customer domain input queue length.

To illustrate, assume the event correlator identifies a performance bottleneck, and that overloaded Dolphins are the root cause of this particular performance problem. The Resource Director selects the response actions to be executed. A number of corrective actions are then considered. The Resource Director will check if the customer's maximum outbound bandwidth is being approached, in which case a change in the admission rate may be necessary. If this is not the case, a new Dolphin will be allocated. If no free Dolphin can be identified, an underutilized one in another customer domain might be identified and reassigned. A fourth option is to throttle the incoming request stream to prevent Dolphin overload.

Generally, there is an ordered list of preferred remedies to a given problem. Once a potential solution is selected, the Resource Director will initiate the necessary actions to implement this selection. Planning responses to SLA performance constraints is more difficult. This is because performance metrics generally differ from system resource management metrics. For example, the customer contract may use response time or active connections as metrics for defining SLA constraints. However, the "knobs" that the Resource Director controls are throttling percentages of new incoming requests and server allocation. The mapping between performance and configuration parameters is not direct. The Resource Director uses a simple but effective method of estimating resources needed from the given performance metrics. The details of this method are beyond the scope of this paper.

4. Resource Management

We now describe the general mechanisms that are used to manage network bandwidth, servers, and the applications and data that are needed for particular customers.

4.1. Bandwidth Management

To maintain satisfactory performance, it may be necessary to limit the number of requests that a customer domain is asked to handle. This may occur if additional servers can no longer be allocated to a customer domain. For example, in the case where the maximum specified by the contract have already been allocated. It may also be necessary to avoid back end server overload. In general, a portion of the incoming requests to a particular customer domain can be denied, via a modification to the front-end IP sprayer.

To prevent back end overload, we use content based throttling to deny requests that specifically require the back end resources. For example, application-specific agents selectively admit requests based on their URL (the content of the request). Priorities can be assigned to each URL enabling fine-grained control. When content based throttling is turned on, the application specific agent intercepts requests and determines, based on the URL, its priority and current system load, whether or not to admit the request. If it can be admitted, the agent lets the request proceed normally. Otherwise, an application specific error message is returned to the client. The Resource Director could determine the thresholds for the acceptable level of load and use content-based throttling agents to prevent overload.

4.2. Server Management

The Dolphin servers in an Océano farm are typically stateless. Their local disks are chiefly used for the local operating system and for temporary working space. This facilitates fast reassignment of Dolphins on demand and simplifies recovery of service following hardware failures. A server manager keeps track of free Dolphins and coordinates their allocation with the Resource Director.

The server manager maintains a pool of available Dolphins. When a server is to be allocated to a particular customer domain, an available server, which meets customer requirements, is selected from this pool. An operating system is installed on the server, which is then primed with customer applications and data (see Section 4.3). Once a server is primed, the network infrastructure is reconfigured so that the server becomes part of the customer domain (see Section 5).

When the Resource Director decides to remove a Dolphin from a customer domain, the server manager places it into a *dirty* state (see Figure 4) by reconfiguring the front-end workload balancers to stop forwarding new requests to the server. Once the server completes its current work, it can be *scrubbed*: the server is removed from its current customer domain, all applications are shut down, and customer data is removed from the disk. The server is then returned to the free pool. Note that a server in the dirty state may be quickly reallocated to the same customer domain because it still contains the applications and data necessary to carry out its assignments. The server manager may use such a server instead of allocating one from the free pool.

4.3. Application and Data Management

When a Dolphin has been assigned to a particular customer domain, the task of installing and configuring all the relevant applications and data can be time consuming. This problem is addressed by managing all customer data (including application binaries) in a shared file system. Installation and configuration of applications onto the shared file system is done offline. The process of priming a server is then reduced to mapping one or more subtrees in a shared file system to the local file system of the server. For some applications, this mapping process is straightforward: several symbolic links need to be created. However, for applications that require system configuration changes, symbolic links are not enough. For example, applications that make use of unshared read-write files such as lock files. Such files must actually be copied from the file server to the local disk. To provide fast and secure data access Océano uses AFS [4], which reduces network traffic and server load by aggressive client caching of files and by proactive invalidation of cached data. In a standard AFS setting, these features provide shared data with performance almost as if the data were local. Finally, AFS caching allows read-only data to be easily replicated among servers for load balancing and scalability.

A potential problem caused by dynamic reallocation of Dolphins is that a sudden increase in load may cause a site to quickly grow in computing power. However, the number of back-end file servers stays the same for longer periods of time. They may therefore become a bottleneck. This is particularly serious when many Dolphins are allocated at the same time. When they are primed, it is likely that they will request access to the same data at the same time from the Whales. Choking the back end servers is prevented by pre-loading "hot" cache data on newly allocated Dolphins, from "older" Dolphins. In addition, multicasts are used to "push" this hot data to all new nodes simultaneously. Additional details are provided in [2].

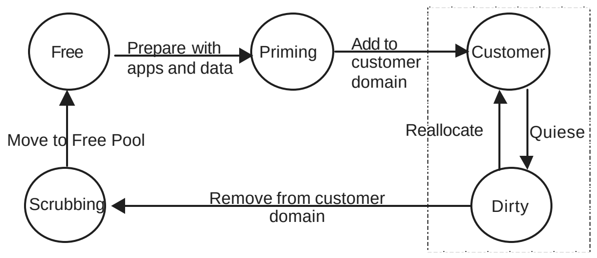

Figure 4: Server states

5. Network Infrastructure and Configuration

Underlying Océano are the network infrastructure and a configuration database. The network infrastructure provides discovery, verification, failure detection and configuration services to Océano. The configuration database is a repository of both static and dynamic configuration data used by Océano to intelligently plan actions, identify future implications of current events, and execute commands.

5.1. Network Infrastructure

The Océano network infrastructure provides a number of services. First, it provides a mechanism for identifying all network components and for monitoring their up/down status. Second, it verifies the discovered network configuration against the expected configuration. For example it verifies that all servers allocated to a particular customer can communicate with each other but that no other servers can do so. Third, it provides the mechanisms to physically partition itself into secure domains. This mechanism is used by the Resource Director to ensure customer domains are independent. Finally, the infrastructure supports load balancing via front end IP sprayers.

Océano "discovers" the physical configuration of the servers, and monitors this configuration on an ongoing basis. This discovery and monitoring process is the result of a distributed algorithm, which is executed by agents on all servers. In it, each agent discovers its link layer neighbors. These neighbors form groups of communicating network adapters. Group membership information is forwarded, through a hierarchy, to a central authority responsible for three tasks: (1) correlating network status events to identify the up/down status of network components and sending these events to other Océano components, (2) verifying the discovered configuration with the expected configuration, and (3) reconfiguring the network when servers are moved from domain to domain.

Océano monitors the availability of network adapters and servers by monitoring the traffic on the various components using an efficient heartbeating algorithm [8]. Failures are reported to other Océano components through the central agent as described above.

The Océano infrastructure physically connects servers using a switched network. The switches used support virtual local area networks (VLANs). Servers connected to a VLAN can freely communicate with each other. However, the switch prevents servers outside the VLAN from communicating with them except via a router. No such router is provided. The network infrastructure programmatically reconfigures the VLANs using SNMP commands when a server is moved from one customer domain to another. The infrastructure can also provide further security by programming firewall settings when customer domains are defined.

5.2. Configuration Database

Without data about the resources and their configuration in an Océano farm, Océano is unable to make informed decisions about how to reconfigure itself in the face of changing access patterns or failures. Océano's configuration database contains both static and dynamic data. Static data changes very slowly and only as a result of human interaction. For example, manual installation of hardware, the addition of new customers, or the renegotiation of a SLA changes this type of data. Examples of static data include hardware (e.g. CPU speed), contractual SLA data, and the software environment (OS, applications) for every customer. Dynamic data captures the changing states of the resources, including aggregated performance data and assignment of servers to customer domains.

6. Status, Performance and Future Work

A demonstration prototype of Océano has been developed and deployed on a testbed containing 55 Linux and AIX nodes. These nodes are interconnected using 100 Mbps switched Ethernet. The prototype is written largely in Java. Initial versions of the Event Correlator, the Resource Director, and managers for network bandwidth, servers and customer applications and data have been developed. Three fictitious customer domains have been defined and deployed in the testbed environment, to illustrate possible applications: (1) static web pages, (2) dynamic web pages using a servlet engine and a back end database, and (3) a video stream server.

6.1. Preliminary Performance Results

Some preliminary measurements of performance were conducted to determine the time to allocate a server to a customer domain. The reported numbers are from a few runs, rather than from a rigorous performance study. Further, the numbers do not reflect the many optimizations that are possible.

First, we measured the elapsed time from a Resource Director decision to allocate a server to the completion of the allocation to be 130 *sec*. Of this, 116-120 *sec* were devoted to priming the application environment of the server. The remaining 8-12 *sec* were for all other tasks including reconfiguring the network infrastructure (configuring VLANs and Network Dispatcher) and information exchange between Océano components. This time was found to be the same for the three customer domains. It does not include the time to pre-fetch customer data. The additional cost to pre-fetch customer data was between 140 *sec* and 200 *sec* depending on the customer domain (the data transferred was 112MB and 162MB, respectively). Thus, the cost to prepare a server, without an operating system install, is between 270 and 330 *sec*.

To install Linux, we use LUI [10] and must reboot the system twice. After the first reboot, the operating system is installed (measured to take 240 *sec*). The second reboot prepares it for use (measured to take 180 *sec*). Thus 420 *sec* are required to prepare a server with a clean running Linux. This time is a function of LUI, the install image size, and the network bandwidth. The image we used has not been optimized for each customer domain; it is the same for all domains. In our experiments, it was 250 MB in size. We suspect it is larger than is necessary because it has been built for development, rather than speed.

In Summary, in the worst case, server allocation in our prototype takes between 710 and 750 *sec*. In practice, the operating system install takes place as a part of the server allocation only when the free pool is empty. Otherwise, the operating system is installed in parallel with other activities; it is pre-installed. In this case, server allocation takes 270 to 330 *sec* (4.5 to 5.5 minutes).

6.2. Future Work

Océano was designed to provide availability for hosted customers, be scalable and enhance manageability. Availability for hosted customers results directly from the overall approach. When server failures occur, they are detected. The SL monitor may identify this failure as a violation of the customer's SLA. In response, the Resource Director allocates new servers. Implicit in this description are two assumptions that require further research. First is the assumption that there are sufficient free resources to meet the demands of all customers. The second assumption is that the Océano infrastructure itself is available. With regards to the first issue, we are developing pricing models, including penalty charges, as a mechanism to handle such circumstances. A second approach we are working on is how to evaluate new SLAs to determine, in advance, how well they can be supported by an existing infrastructure. Finally, we are looking at applying customer priorities to resource allocation decisions when there are no servers in the free pool. With regards to the second question, the current implementation did not focus on guaranteeing component availability. As with availability, customer domains are scalable. However, we need to demonstrate that the components of Océano are scalable. For example, the event correlator and the network infrastructure are designed to work in a hierarchical manner. Further, multicasts are used to prime servers. To study the success of these design choices, we are increasing the size of our testbed and actively looking for larger environments in which to test the system. Further work will consider server heterogeneity.

7. Conclusions

The Océano prototype demonstrates the ability of an e-business computing utility to "manage" itself without human intervention, dynamically reallocating resources as needed to meet the demands of a shifting load, while enforcing the security and isolation requirements of a shared computing utility. The architecture is quite flexible, for supporting different platforms or implementing differing service level agreements, and preliminary performance measurements have been very encouraging. The work to date has been focused on aspects of the system which are relatively straightforward to monitor and control: allocation of servers for increased processing capacity; and throttling of network bandwidth when additional servers are unavailable or won't help (e.g., because the bottleneck is at a "back-end" database). Future work will explore more complex "tuning knobs", such as interactions with the schedulers and workload managers embodied in the kernels of the various individual systems, or with logical partition managers on systems that support flexible partitioning.

References

[1] Appleby K., Goldszmidt G., and Steinder M., "Yemanja – A Layered Event Correlation Engine for Multi-domain Server Farms", Proceedings of the Seventh IFIP/IEEE International Symposium on Integrated Network Management, 2001.

[2] Azagury, A., Goldszmidt, G., Koren Y., Rochwerger B., and Tal A.,"Khnum – Data Management for a Dynamically Scalable Computing Utility", unpublished draft, January 2001.

[3] Bruno, J., Gabber, E., Ozden B., and Silberschatz A. "The Eclipse Operating System: Providing Quality of Service via Reservation Domains", *Proceedings of the 1998 USENIX Annual Technical Conference*., pp. 117-130, June 1998.

[4] Campbell, R., Managing AFS: the Andrew File System, Prentice Hall, 1998.

[5] Cunha, J., da Silva, F., Goldszmidt, G., and Appleby, K. "An Architecture to Define, Store, and Monitor SLAs in Server Farm", unpub. draft, February 2001.

[6] Devlin, B., Gray, J., Laing, B., and Spix G., "Scalability Terminology: Farms, Clones, Partitions, and Packs: RACS and RAPS", *Microsoft Research Technical Report*, December 1999.

[7] Ensim Corp., “ServerXchange” (White Paper), http://www.ensim/com. Moutain View, California.

[8] Fakhouri, S., Goldszmidt, G., Gupta, I., Kalantar, M., and Pershing J., “GulfStream – A System for Dynamic Topology Management in Multi-domain Server Farms”, IBM Technical Report, 2001.

[9] Goldszmidt G., and Hunt G., “Scaling Internet Services by Dynamic Allocation of Connections”, *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management (IM 1999)*, pp. 171-184, May 1999.

[10] Linux Utility for cluster Install (LUI). http://oss.software.ibm.com/developer/ opensource/linux/projects/lui

[11] Nick, J.M., Moore, B.B., Chung, J.Y., and Bowen, N., “S/390 cluster technology: Parallel Sysplex”, *IBM Systems Journal*, Vol. 32, No. 2, 1997.

[12] Pfister, G.F., In Search of Clusters, The Ongoing Battle in Lowly Parallel Computing, Prentice Hall, 1998.

[13] Raman, R., Livny, M., and Solomon, M. “Matchmaking: An extensible framework for distributed resource management”, *Cluster: Journal of Software, Networks and Applications*, 2(2), 1999.

[14] Reumann, J., Mehra, A., Shin, K.G., and Kandlur, D. “Virtual Services: A New Abstraction for Server Consolidation”, *Proceedings of the 2000 USENIX Annual Technical Conference*, pp. 117-130, June 2000.

[15] Rooney, S. “The IcorpMaker: A Dynamic Framework for Application-Service Providers”, *Proceedings of the IEEE Workshop on IP-oriented Operations and Management*, Cracow, Poland, Sept 4-6, 2000.

[16] Sun Microsystems Inc., “Sun Enterprise 10000 Server: Dynamic System Domains” (White Paper), Palo Alto, California.

[17] Squillante, M. S., Yao, D. D., and Zhang, L. “Web Traffic Modeling and Web Server Performance Analysis”, *Proceedings of the 39th IEEE Conference on Decision and Control*, December 1999.

[18] Vogels W., and Dumitriu D.M., “An Overview of the Galaxy Management Framework for Scalable Enterprise Cluster Computing”, *Proceedings of the IEEE International Conference on Cluster Computing: Cluster-2000*, Chemnitz, Germany, December 2000.

Acknowledgments

Many people have contributed to the design and implementation of Océano. Among them: J. Lorrain, J. Fong, A. Tal, R. Ferri, I. Gupta, M. Steinder, J. Cunha, Y. Koren, T. Eilam, M. Mei, B. White, and V. Aharonian. J. Sanmugaraja, J. Norris, and M. Frissora spent many hours supporting the hardware and software in our test farm. Finally, we would like to thank a few of the managers that strongly supported this project, A. Azagury, N. Bowen, T. Chandra, A. Fleishman, and A. Krishna.