Distributed Computing: Moving from CGI to CORBA

James FitzGibbon
TargetNet.com Inc.
Tim Strike
TargetNet.com Inc.

Abstract

In this paper, we document the evolution of a banner ad delivery system from a simple CGI script written in Perl running on a single host into a distributed computing application using CORBA.

While CORBA has an established history in the enterprise-computing world, it is only recently that the Open-Source® community has begun to embrace it. Starting without any RPC programming experience, it took TargetNet a little less than half a year to integrate CORBA into the Apache web server and convert all their CGI programs into CORBA servers.

Performance of the system increased from 50 transactions per second to over 400 per second. Thanks to the cross-platform capabilities of CORBA, future components can be developed on virtually any operating system and programming language. By adding inexpensive servers, the capacity of the system scales in a near-linear fashion. Most importantly, the switch to CORBA didn't require a change of operating system or development environment – everything runs on a free operating system using OpenSource components.

Introduction

In 1997, TargetNet built and deployed "The Datacom Ad Network", a banner ad delivery system. The CGI script that selected and delivered ads was written in Perl, ran on a Pentium 200 machine, and could deliver just one ad per second. Rewriting in C and migration to faster processors took performance to 10 ads per second, then 30, then 50. Further optimization proved futile: we had reached the CGI performance barrier.

Increasing performance by adding processors or hosts was not feasible: the architecture of the existing delivery system was limited to running on a single host, and was single-threaded. Worse, the code used a large number of flat files on disk, and so spent a large percentage of its time performing system calls or waiting for file locks. Increasing processor power provided a brief respite, but we could not afford to upgrade server hardware forever.

To remain competitive, ad delivery performance needed to increase to roughly 400 ads per second plus allow for multiple hosts to share the load of the network. It was clear that a new architecture was required that could overcome the limitations of standard CGI scripts. The base requirements of the new system were laid down before any research began. The new system would have to:

- offer single host performance several times that of the existing CGI script.
- utilize a distributed computing model without arbitrary limits on the number of hosts.
- allow multiple hosts to share the network load, preferably with load balancing and redundancy.
- be called from a web browser like a CGI script but without the inherent limitations of CGI.
- scale to handle anticipated growth over the next four to five years without major architectural changes.
- allow gradual integration of commercial hardware and software without massive re-coding.

Most importantly, the system had to be cost effective. Our server platform was Apache running on FreeBSD, and all software in use was either freely available or developed in-house. While commercial solutions to our problem existed (Oracle Parallel Server running on a commercial UNIX), financial constraints dictated that we find a free solution or develop one in-house.

Breaking the CGI speed barrier

The problem of CGI performance is not new. Over the last several years, many solutions that remove the forked-CGI bottleneck from web applications have come to light.

CGI enhancement wrappers like [FASTCGI] allow an existing CGI script (especially those written in interpreted languages like Perl) to run much faster and remain resident between invocations. Still, these wrappers extend the existing CGI standard, sacrificing flexibility for compatibility. They are limited, naturally, to tasks that you would usually use a CGI script for. Other client/server tasks need to be addressed separately.

Integrating the functionality of a CGI script directly into the web server provides the benefits of a CGI wrapper, and gives developers access to the internals of the web server. Not only does this technique share the same limitations as CGI wrapper toolkits, but developers have to lock themselves into a particular web server architecture, choosing to develop an ISAPI, NSAPI, or Apache module. Similarities between these architectures are few: moving a complex module from one web server product to another could require a complete rewrite.

Clustered computing solutions promise transparent scalability simply by adding hosts. Unfortunately, most clustering systems are commercial, costly, and tied to a particular line of hardware (though there are alternatives, such as Beowulf clusters running on Linux). Using commodity hardware would have addressed the cost issue, but would have required us to switch from FreeBSD to Linux. In doing so we would be giving up the significant investment we already had in FreeBSD servers and knowledgeable personnel. In short, we felt that the immediate benefits of clustering were outweighed by the commitment that one has to make to a particular vendor and/or operating system.

Remote Procedure Call (RPC) solutions do not share the aforementioned limitations. Most RPC solutions are available for multiple operating systems and hardware. They are abstracted from the application layer, and do not require adherence to one vendor's API. As it is not directly tied to the CGI model, RPC can be used to replace traditional client/server applications as well. The only issue surrounding RPC is which architecture to use.

ONC RPC, developed by Sun Microsystems, is already in wide use on UNIX® systems. ONC RPC is at the heart of NIS, NIS+ and NFS. The limitations of the original ONC RPC have been documented and exploited for as long as they have been in use. ONC RPC+ addresses many of these limitations and provides for encrypted communication but is not as widely available as the original implementation.

The Distributed Computing Environment (DCE) from the Open Group provides almost every distributed computing tool one could need, but is as complex as it is complete. Suited best for large projects, the administration of DCE can be a monumental task. Only one free implementation of DCE is available, limiting improvement through vendor competition.

We found many of the elements we required in RPC, but we did not find an implementation that provided all of them in one package. We attempted to build our own middleware, without much success. The issues that undoubtedly plagued the developers of ONC RPC and DCE proved too much for our small team of developers. Returning to the research arena, we began to look at CORBA.

We had dismissed CORBA early in the design phase, believing it to be geared towards enterprise computing and unsuitable for our use. An in-depth examination showed promise. CORBA offered everything that we were looking for: unlimited cross-platform capability, several free implementations, and an aggressive development model that promises to keep the technology alive in the future. As discussed in [MODZ97], CORBA also offers features not found in RPC or DCE, including interface portability, dynamic interface invo-

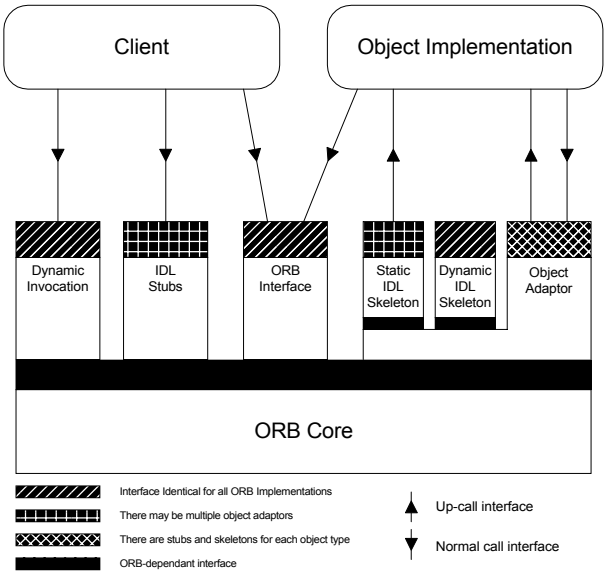

Figure 1: the CORBA Infrastructure

cation and platform independent data types. Only one question remained: how easy would CORBA be to integrate into a system built solely from OpenSource software and in-house code?

Back to School

The first order of business was to learn about CORBA. Starting with no more than a conceptual understanding of RPC mechanisms, we needed to deliver a proof-of-concept application in short order. To learn about CORBA, the best place to start is the Object Management Group[OMG]. The OMG produces the CORBA specification, but unlike ONC RPC and DCE, does not provide an implementation. This distinction prevents the specification group from monopolizing the implementation.

CORBA allows applications to communicate with each other, without regard to where they are located, the operating system they run on, or the programming language they were developed in. CORBA allows the developer to concentrate on the function that the application performs as opposed to the mundane details of protocol design and network transport. At the heart of a CORBA application is the ORB, or Object Request Broker. The ORB is responsible for intercepting client calls for remote methods, locating the host that provides the implementation of that method and the packaging of parameters and return values over the wire. The OMG also specifies the Internet Inter-Orb Protocol (IIOP), which is protocol that ORBs use to talk to each other. Figure 1 (from the OMG's web site) illustrates the architecture.

CORBA allows clients and servers to be written in any programming language for which a CORBA mapping has been defined. To create a CORBA server, the procedures and methods of the application are described using the Interface Description Language (IDL), which is roughly analogous to a C++ class definition. The OMG specifies both the syntax of IDL and the mapping from IDL data types to language specific data types. Presently, there are language mappings for C, C++, Ada, COBOL, Java, Smalltalk and Python. There are several independent language mappings for Perl, though they have yet to be ratified by the OMG.

An IDL compiler (part of the CORBA development environment) is used to create language-specific code from an IDL definition by generating client-side stubs and server-side skeletons. When writing a server, you add your implementation code to the generated skeletons. When writing a client, you call the functions from the generated client stubs. In either case, the finished binary is linked against a CORBA runtime object, usually in the form of a shared library or DLL.

Choosing an ORB

At the onset of this project, TargetNet's server base consisted of FreeBSD machines running on Intel hardware. The number of freely available ORBs is significantly lower than those available for commercial UNIX systems or for Windows.

Our ideal ORB would support both C and C++, use POSIX threads for maximum server performance, and support the latest CORBA specification. We evaluated the five ORBs that were readily available for FreeBSD:

Table 1: ORBs available for FreeBSD

ORB	Supported languages	True CORBA ORB?	Performance[1]	License Type / Cost	Comments
mico	C++	yes	0.1 ops/sec	GPL	slow performance in testing
ILU	Many	no	not tested[2]	BSD-like; free	reads IDL
ORBacus	C++, Java	yes	not tested[3]	$3000 per seat	SSL support
ORBit	C, C++ coming	yes	20 ops/sec	GPL	basis for GNOME
omniORB	C++, Python	yes	1250 ops/sec	GPL	multithreaded servant

[1] We measured performance by invoking a "null" function which took no parameters and returned no output
[2] The feature set of ILU did not meet our criteria, and was removed as a candidate prior to testing.
[3] We deferred testing of ORBacus due to its cost; the performance of omniORB finalized our decision before testing of ORBacus became necessary.

- mico [MICO], which supported only C++.

- ILU [ILU], which supported more languages than the OMG has language mappings. ILU is not a true CORBA ORB, but it is conceptually similar and can read IDL definitions.

- ORBacus [ORBACUS], which supports C++ and Java and provides an SSL interface, but is not freely available.

- ORBit [ORBIT], which is the technology at the heart of the GNOME project. ORBit supports only C, but a C++ binding is being developed.

- omniORB [OMNIORB], which supports C++ and Python.

The results of our evaluation are summarized in Table 1.

Our primary concern was standards conformance, followed by performance. Inexperienced as we were with CORBA, we didn't want to complicate matters with a non-compliant ORB. ILU failed the conformance portion early on - though it understands IDL, the language mapping is not the same as the CORBA specification.

Our test suite consisted of a simple function that took no parameters and returned no output (the intent was to measure ORB overhead). In retrospect, further research into existing benchmarks would have been beneficial; [SCHM96] describes modifications to the popular ttcp network benchmark program to test CORBA performance.

mico's performance was extremely slow, taking up to 10 seconds for each transaction. ORBit worked well as a client, but its server implementation serialized remote requests, restricting performance to 20 transactions per second.

omniORB took first place for performance – its server implementation is fully multithreaded, creating a new thread for each remote request while using several "housekeeping" threads for connection management. omniORB was able to handle 1,250 transactions per second. The only downside to omniORB was that it did not provide a C language binding.

Having found a free ORB that met our performance criteria, we did not test the performance of ORBacus. Its US $3000 price per developer scared us away; we may take another look at it if we require the extra features it provides.

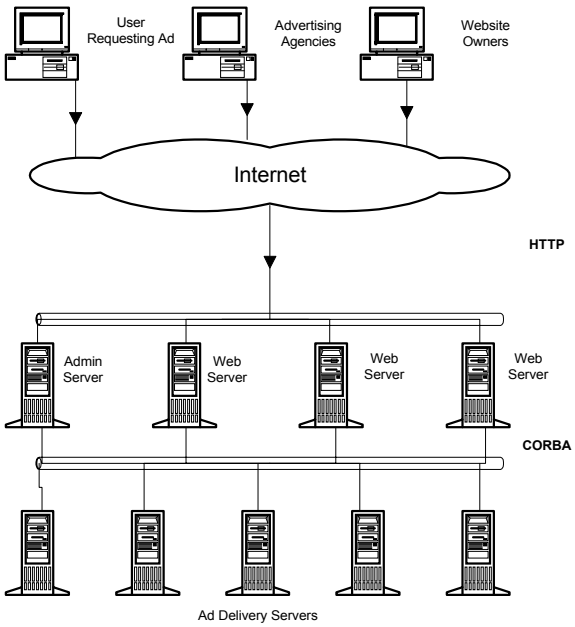

Figure 2: Datacom Mk2 Overview

Having completed our tests, we were unable to find all of the desired features in one product. In the end, we chose to use two ORBs: omniORB for C++ client and server development, and ORBit for C client development. Due to the performance limitations of the ORBit server model, we decided to restrict server development to C++. With our development tools in hand, we began the design and planning of the new ad delivery system.

Datacom Mk2

Our original vision for the new ad delivery network is outlined in figure 2. There are two distinct groups of users who interact with the system over the Internet. The first group includes clients (advertising agencies) and members (the owners of the web sites that display our ads), who log into the administrative server to query how many hits and clicks have been recorded. The second group is the users who request ads as they visit member web sites. These users need to contact one of several ad servers, which select an appropriate ad to display based upon targeting criteria, and return that ad to the user.

As figure 2 illustrates, CORBA would only be used for communication between the administrative server and the ad servers. The main drawback to this model is that it requires an HTTP daemon to be installed and configured on each ad server, and that each ad server needs to be directly connected to the Internet. This precludes the

ad servers using services that are not safe to run on a public network, including file sharing or an RDBMS. In the event that an ad server goes down, users who continue to use a cached IP address will see broken images when they attempt to connect.

Our solution was to abstract the ad server communication with the Internet. Rather than talking directly to web browsers, the ad delivery machines would use CORBA to communicate with a proxy machine. The remote users would use HTTP to talk to the proxy machine, which could select the best ad server to handle the request based upon server load.

An HTTP to CORBA Proxy

To allow a server to interact with a user on the Internet without being directly connected to the Internet, the server must make use of a proxy or gateway of some type. A firewall is an example of a proxy with which most people are familiar. Most firewalls perform simple address translation without regard to the protocol that the IP packets carry. What Datacom called for was a proxy server that could convert an HTTP request into a CORBA call, then take the result, if any, and convert it back into an HTTP response.

Using such a proxy (which we named the "Dispatch Server") allowed us to connect the dispatch machines to the Internet, leaving all other hosts connected to a private LAN only. Converting from HTTP to CORBA was only the initial goal. With careful planning, the dispatch server could act as a gateway between other protocols as well. Future goals include gating between HTTP and a Generic NQS queue [GNQS] or an Enterprise JavaBean gateway[JAVABEAN].

Figure 3 illustrates how the dispatch server integrates into Datacom Mk 2.

The dispatch server is implemented as an Apache module, written in C, using ORBit as its ORB. When a web browser requests an ad, it contacts the dispatch server and requests a URL such as

http://dispatch.TargetNet.com?target=ad&dir=bi&member=jfitz

This URL represents a request for a bi-directional communication with the ad delivery service, passing the single key-value pair "member=jfitz" to the server. To service this request, the dispatch server makes a connection to the ad delivery server and invokes a remote procedure named "DispatchRequest". The name-value pairs are passed as parameters. The response from the procedure is returned to the web browser as if it had come from a CGI script. The web browser is not aware that the response it receives back from the dispatch server was actually generated on a different machine. Thanks to the cross-platform capabilities of CORBA, it does not even know if the response was generated on a UNIX machine in the same data center or on a Windows NT box halfway around the world.

For requests that do not require a response to be returned to the client, the dispatch server calls the remote procedure "Dispatch::unidirectional", which takes the same input parameters but returns no output.

Abstracting the Server interface

Earlier, we mentioned that for a client to invoke a remote CORBA call, it must be linked with the client stub code generated by the IDL compiler. Imagine for a moment that the ad delivery network expands such that there are fifteen different types of servers in the system. If each server defines a separate interface then the dispatch server would have to link in fifteen different client stubs in order to dispatch requests to all the servers. Every time a new type of server was added, the dispatch server would have to be re-compiled and deployed across the network. This limits the scalability of the dispatch architecture.

There are two techniques that avoid this. The first uses the Dynamic Invocation Interface (DII) features of

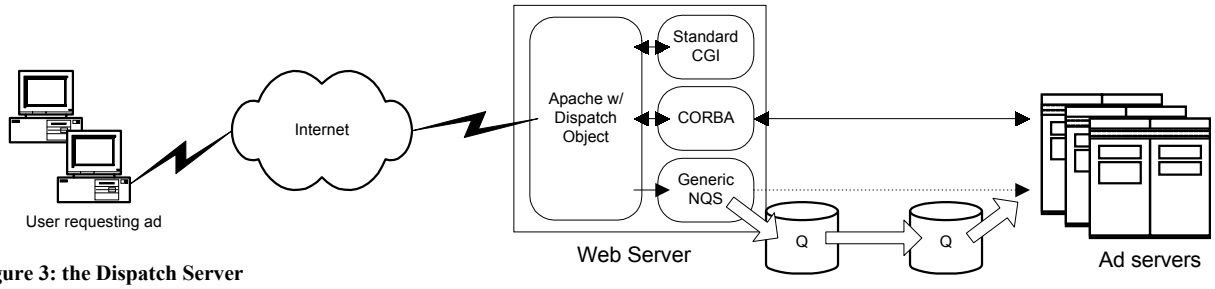

Figure 3: the Dispatch Server

```
Module Dispatch {

        typedef sequence<octet> stream;

        interface unidirectional : Control {
                long DispatchRequest(
                        in string id,
                        in string indata,
                        in string inattr
                );
        };

        interface bidirectional : Control {
                long DispatchRequest(
                        in string id,
                        in string indata,
                        in string inattr,
                        out stream outdata,
                        out string outattr
                );
        };

};
```

Figure 4: the Dispatch IDL

CORBA. DII allows a client to create references to a server at run time instead of at compile time. This is an elegant approach, but it is an advanced CORBA topic that we were (at the time) uncomfortable using. To invoke operations on arbitrary servers while still using static invocation, we chose to define a base interface from which all servers would inherit.

Recall that IDL is conceptually similar to a C++ class declaration. An interface can inherit from another interface and methods in a derived interface can override methods from their base interface. Multiple interfaces may be grouped into a module. The definition of the Dispatch interface is in Figure 4.

The Dispatch module contains two interfaces: unidirectional and bi-directional. Each of these interfaces defines one member function called DispatchRequest. Each of the CORBA servers has its own interface, which is derived from the unidirectional or bi-directional interface. As in C++, a reference to a base interface object can be used to refer to a derived interface object. This allows the dispatch server to invoke the DispatchRequest function on the remote server without actually knowing which type of server it is talking to. As we become more experienced with CORBA, we may abandon this method in favor of DII. For now, our inherited base-interface technique provides the functionality we need with a minimum of complexity.

Locating remote servers

One important step that we have not discussed is the method by which an ORB determines the specific server that provides a given interface. Most ORB implementations assign an ephemeral listening port upon server startup. For the ORB to make a connection to a remote host, it needs to know the IP address of the host, the port number and the name of the interface.

To represent this information in a platform-independent way, CORBA uses an Interoperable Object Reference, or IOR. The IOR is a text string that encodes the values of the host, port and interface for a given server. Upon server start-up, the ORB generates an IOR. The client ORB takes this IOR, decodes the values, and establishes a connection to the remote host. How, then, is the IOR communicated between the server and the client?

The CORBA spec includes interfaces to CORBA helper applications. One of these is the CosNaming service, which is a directory service that allows an ORB to register and lookup IORs. The CosNaming service provides an effective method for looking up IORs but requires that all servers use a single host as their naming service. Neither omniORB nor ORBit provides a facility for the naming service on one machine to share information with the naming service on another machine. If the machine providing the naming service goes down, servers will not be able to register their IORs, nor will clients be able to look up IORs.

There are efforts underway to replace the CosNaming service with other directory services, such as LDAP. [RFC2714] describes an LDAP schema for storing

```
ad-delivery.iiop.datacom.rpc          IN   SRV  0 0 0 ior.ad-delivery.ad-01.datacom.rpc
                                      IN   SRV  0 0 0 ior.ad-delivery.ad-02.datacom.rpc

ior.ad-delivery.ad-01.datacom.rpc     IN   TXT  "IOR:72616c2d30312e746f722e646174"…
ior.ad-delivery.ad-02.datacom.rpc     IN   TXT  "IOR:6d2f52656369657665723a312e30"…
```

Figure 5: the Dynamic DNS zone

IORs, which would eliminate many of the problems with the CosNaming service. This solution still has a single point of failure, as all updates must be made to the LDAP directory master. Rather than wait for these efforts to bear fruit, we developed our own IOR directory, building in load balancing, replication and redundancy. We call this directory daemon the Service HeartBeat Daemon.

The Service HeartBeat Daemon

The Service Heartbeat Daemon, or HBD, performs several important tasks:

- accepts registrations from servers and keeps them in an internal table
- performs regular checks on all recognized servers to make sure that they are answering requests
- synchronizes its internal table with those of the HBDs running on other hosts.
- synchronizes a DNS zone with its internal table using Dynamic DNS updates.
- listens on a socket for clients wishing to know the IOR for a given service.

The architecture of the Service HeartBeat Daemon is conceptually similar to that of IGOR [MODZ97], though we were not at the time aware of this work. Unlike IGOR, the HeartBeat daemon does not store object references in persistent storage; rather it relies on a peer network in which daemons on different hosts keep each other up-to-date.

Upon start-up, a server determines its IOR, which is then sent to the HBD running on the local host. The HBD performs a quick check to ensure that the server is really up, echoes the registration information to other HBDs using multicast IP and then inserts the registration information into its internal table. The HBD regularly synchronizes its internal table with a DNS zone using Dynamic DNS updates. SRV records in DNS represent servers which are up and the IORs are stored using TXT records. If ad delivery servers were started on the hosts ad-01.targetnet.com and ad-02.targetnet.com, the DNS zone would resemble that in figure 5.

A client can find the IOR for a service using two methods: DNS or local socket lookup. To use DNS, the client asks a nameserver for the SRV records for a given service name. The client then uses the algorithm

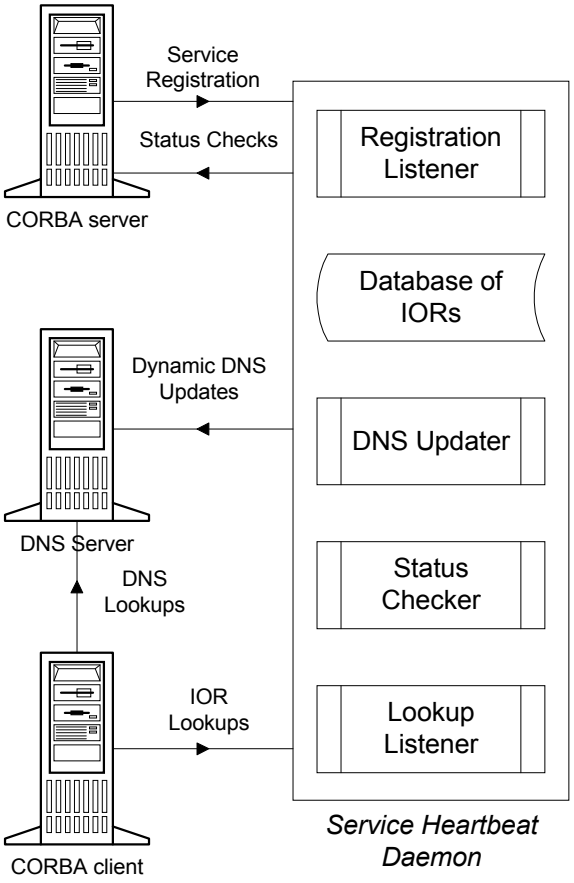

Figure 6: the Service HeartBeat Daemon

described in [RFC2052] to select among the multiple records returned. We deviate slightly from the RFC in that the target field of the SRV record is not the host on which the service can be found but the hostname whose TXT record holds the IOR to be used.

The DNS method of lookup has two shortcomings: it requires that the HBD keep a DNS server synchronized with the internal list of tables, and due to the 255 character limit of DNS TXT resource records, it cannot be used to store an IOR in excess of this length (In practice, omniORB does not generate IORs in excess of 255 characters, but several of the other ORBs that we tested did). In such cases, the client can make a direct socket connection to the HBD. The client requests the IOR for a given service (in which case the "best" IOR is returned using the same algorithm as the DNS lookup method) or for a specific service and host combination.

Figure 6 illustrates the function of the Service Heartbeat Daemon.

Adding load balancing and redundancy

The DNS zone in figure 6 uses zeros for the preference, weight, and port values in the SRV record. As mentioned previously, the port for a CORBA server is ephemeral, making it of little use to us. However, the preference and weighting values can be very useful.

When a server registers itself with the HBD, it can specify a priority and weighting. Priorities in SRV records are used just like the priority field of a MX record – first you sort by priority lowest to highest and then you select from all servers with the same priority value. The weight value is used in the SRV record algorithm and allows a server to ask for a smaller or larger proportion of the network load. We select a weight value for each server based upon its capacity, allowing us to take full advantage of each server as we roll in more powerful machines.

To provide a common interface for determining if a server is up or down, we defined a Control interface that the Dispatch interface inherits from. This interface defines administrative procedures to bring services up, take them down, or query their status. When a server is asked for its status, it may simply return a TRUE value, or it may perform complex checks of the resources that it uses (databases, free disk space, etc.) before deciding if it is really up and ready to handle requests.

For the HBD to detect when a server has crashed, it must regularly check the status of each server in its internal table. If one of these checks fail, it sends a multicast IP alert to the other HBDs on the network. When an HBD receives such an alert, it performs its own check of the service, and if is still found to be down, the server is removed from the internal table. This eliminates the race condition between one HBD marking a service as down and another HBD multicasting an update saying that the same service is up.

There is a degree of latency between a server crashing and all HeartBeat Daemons on the network removing the server from their internal lists; for this reason CORBA clients must be prepared to soft-fail if the attempt to connect to the server fails. As discussed in [MODZ97] certain commercial ORBs (such as Visigenic's Visibroker) offer the ability to transparently fail-over to another application, but this feature is not available in omniORB.

An interesting approach to transparently providing redundancy is that employed by the Eternal system [NARA97]. Eternal intercepts the IIOP protocol communication between CORBA clients and servers, redirecting the communication to a reliable multicast group of servers in such a way that multiple servers are given the opportunity to respond to the request. Eternal intercepts the IIOP stream at the operating system level, and as such can be plugged into any ORB that supports IIOP. Unfortunately, Eternal is still under development at the University of California, Santa Barbara and is not yet available to the public.

Security

We mentioned that all of the CORBA communication takes place over a private LAN. The decision to isolate CORBA traffic allowed us to deploy Datacom Mk2 without building a security infrastructure on top of CORBA. Although several ORB vendors offer CORBA over SSL solutions, they are proprietary, invalidating one of CORBA's biggest selling points. We instead used an open source IPSEC tunnel program to extend this private network, allowing Datacom to use multiple data centers for performance and redundancy. In the future, we hope to move from running software-based tunnels on our servers to using a firmware solution, such as Cisco's VPN product running on our routers.

Performance

The success of our conversion was tied to tangible results: the new system had to meet or exceed our requirements for it. Using CORBA increases processor and network overhead and our initial performance tests did not simulate the load that the system would be under in the real world. Throughout our development cycle, a large question mark loomed overhead: might we have done all this work only to run into yet another performance barrier?

Our target was 200 transactions per second on each 600Mhz Pentium III machine in the network. After all the servers were running in the lab, we were able to max out a 400Mhz Pentium II machine at more than 400 transactions per second. The design of the system allows for near-linear performance increases as hosts are brought online, so we expect that this model will serve us for the next several years.

Looking to the Future: CORBA 3

The benefits that CORBA has provided to our application are numerous, but there is an even bigger bonus yet to come: CORBA 3. This collection of specifications addresses many of the annoyances present in the CORBA 2.3 specification and lays the groundwork for integrating CORBA with other component technologies, like ActiveX and JavaBeans.

The new features in CORBA 3 fall into three categories: Internet Integration, Quality of Service (QoS), and the CORBAcomponent architecture. Internet Integration allows CORBA to communicate easier over the Internet by defining a firewall protocol that allows for client-side callbacks over a single TCP/IP connection.

Internet Integration also defines IIOP over SSL, which will allow ORBs from multiple vendors to talk to each other securely. Another big change is the Interoperable Naming Service which lets a client find an IOR using a new URL syntax, like this:

iioploc://ns.targetnet.com/AdDelivery

While this feature might seem to render the Service Heartbeat Daemon obsolete, it still has a single insertion point for updates. Until a hybrid solution that provides a redundant IOR directory appears, we expect that the HBD will continue to be an essential part of our system.

Also in the new spec are QoS extensions, including asynchronous messaging and queue priorities. Specifications for Minimum, Fault-tolerant and Real-time CORBA will make CORBA a viable solution for many applications that were unable to use CORBA version 2.

The CORBAcomponent architecture is intended to separate CORBA into components, and defines the integration of these components with popular scripting languages. This will allow a developer to freely mix and match CORBA 3 components, ActiveX controls, and Enterprise JavaBeans, choosing the best technology for each part of their problem without becoming mired in interoperability issues. This could be the single most important part of CORBA 3, as it will break the present exclusive development model that CORBA 2 forces developers to use.

The CORBA 3 specification will not be published until mid-to-late 2000. Even then, we are not sure how quickly or how much of CORBA 3 we will use. Having spent the time building CORBA into the heart of the ad delivery system gives us the freedom to migrate parts of the system to CORBA 3 at a comfortable pace. Had we chosen a different architecture, we might have been forced to move the entire system in one piece.

Concluding Remarks

The conversion of Datacom from CGI to CORBA took us a little more than four months. Making the journey from no CORBA knowledge to full implementation was not without its pitfalls – we cut some corners and skipped over those topics deemed "too advanced" at the time. Nevertheless, we succeeded in creating a system that exceeds its design requirements several times over and allows for almost limitless expansion.

We could have chosen to outsource a solution, but when faced with financial constrictions, solving the problem in-house is often the only option. With a little creative thinking and some investment in learning new technologies, it is possible to deploy a distributed computing model without a large capital investment.

CORBA does not have to remain in the enterprise-computing arena. Examination of projects like Datacom or industry software like the Dents name server [DENTS] and the GNOME environment [GNOME] proves that CORBA/OpenSource integration is viable today. Whether you use CORBA implicitly by contributing to these projects or explicitly as we have, it is clear that CORBA has a bright future in the Open-Source community. We hope that our experience encourages OpenSource developers to seriously consider CORBA as part of their projects.

References

[FASTCGI] http://www.fastcgi.com/

[MODZ97] Brent E. Modzelewski and David Cyganski, "Interactive-Group Object-Replication Fault Tolerance for CORBA," *Proceedings of the Third USENIX Conference on Object-Oriented Technologies and Systems*, Portland, Oregon (June 1997).

[OMG] http://www.omg.org/

[MICO] http://diamant.vsb.cs.uni-frankfurt.de/~mico/

[ILU] ftp://parcftp.xerox.com/pub/ilu/ilu.html

[ORBACUS] http://www.ORBacus.com

[ORBIT] http://www.labs.redhat.com/orbit/

[OMNIORB] http://www.orl.co.uk/omniORB/omniORB.html

[SCHM96] Douglas C. Schmidt, Tim Harrison and Ehab Al-Shaer, “Object-Oriented Components for High-speed Network Programming”, *Proceedings of the USENIX Conference on Object-Oriented Technologies*, Monterey, California (June 1995).

[GNQS] http://www.gnqs.org/

[JAVABEAN] http://java.sun.com/beans/faq/faq.enterprise.html

[RFC2714] V. Ryan, R. Lee, and S. Seligman, *Schema for Representing CORBA Object References in an LDAP Directory*, RFC2714 (October 1999), ftp://ftp.isi.edu/in-notes/rfc2714.txt.

[RFC2052] A. Gulbrandsen and P. Vixie, *A DNS RR for specifying the location of services (DNS SRV)*, RFC2052 (October 1996), ftp://ftp.isi.edu/in-notes/rfc2052.txt

[NARA97] P. Narasimhan, L.E. Moser and P.M. Melliar-Smith, “The Interception Approach to Reliable Distributed CORBA Objects”, *Proceedings of the Third USENIX Conference on Object-Oriented Technologies and Systems*, Portland, Oregon (June 1997).

[DENTS] http://www.dents.org/

[GNOME] http://www.gnome.org/