Department of Computing Science,
University of Newcastle upon Tyne.
Newcastle upon Tyne, NE1 7RU,
United Kingdom.

Dependable Web Service Provision

A report for Hewlett-Packard Labs, Bristol.

Author: David Ingham
Version: 2.0 (revision 259)
Pages: 34

Printed: Wednesday, 22 January, 1997
Revised: Wednesday, 22 January, 1997
Created: Sunday, 25 August, 1996

Table of Contents

Table of Figures

Introduction

In the context of Web service provision, the term dependable encompasses several different concepts. Firstly, it is desired that Web services be highly available even in the presence of component failures, since unavailability reflects badly on the service provider and may result in lost opportunities. Since access to Web services is international, 24 hour, 7 day availability is desired; there is no appropriate time to remove the service for maintenance purposes. Associated with availability is quality of service (QoS), that is, the perceived responsiveness of a site may degrade as the number of requests it receives increases. From a client's perspective, an overloaded service is perceived as an unavailable service. Therefore, scalable mechanisms are required to manage ever-increasing load requirements, while preserving high availability. Another feature of dependability is data integrity. The nature of Web services is rapidly changing from its original role as an information distribution medium to become a platform for complex services, such as electronic shops, that perform read/write actions on local data. In such systems, maintaining integrity of data in the face of concurrent access and occasional system failures is imperative.

This document first describes how these various aspects of dependability are currently being addressed by today's service providers. The dependability requirements of the CORBAWeb system are then analysed. A number of alternative techniques are presented as suggestions on how the current weaknesses could be addressed.

State of the Art Web Services

The primary task of the majority of today's Web services is to disseminate read-only documentation-based information, for example, an academic site serving information about its teaching and research activities or a company using the Web as an advertising and public relations medium. In addition to serving read-only data, some services perform read/write actions initiated via Web requests. These interactions range from simple data collection tasks, such as filling in a reply-form to register for a conference to more sophisticated services such as an on-line electronic shop, such as CDnow [CDnow].

This section considers two major problems facing major Web service providers, namely, supporting high volume sites and maintaining highly available services. The current approaches are considered and their associated limitations are discussed.

Supporting High Volume Web Services

As the popularity of a Web service increases, the load, i.e., the number of incoming requests per second (rps), can reach a level that cannot be satisfied by a single machine[1]. The primary limiting factor is typically exhaustion of operating system resources, i.e., number of processes and available memory, as a result of high rates of opening and closing TCP/IP connections for long periods of time [Katz94]. To alleviate this situation, load sharing techniques are required to distribute the load among a number of machines.

[1] NCSA analysis revealed that server-class machines either crashed or exhibited severe signs of overloading when experiencing peak arrival rates of approximately 20 rps [Katz94].

There are two main issues to be addressed in order to distribute a Web service across several machines. Firstly, there is data distribution; is the data itself partitioned among the machines or do all of the servers serve copies of the same data and if so is the data replicated or shared? Secondly, how is the load distributed among the machines?

Data distribution in a Multi-Host Web Server

In order to understand the implications of the possible approaches to data distribution, one has to be aware of the Web's native addressing mechanism. Web resources are named via their Uniform Resource Locator (URL), a location-based naming mechanism which specifies the Internet hostname of the machine on which the resource resides and the name of the resource on that machine. The Internet hostname is translated to an IP address which is used by the TCP/IP protocol to communicate with the server machine. This approach has the advantage of low processing and communication overheads but limits the flexibility of the system, by tightly coupling resource names with their physical location.

Partitioning the data

One approach to providing a multi-host Web server is to utilise the inherent distribution of the Web itself, that is, by partitioning the data among the server machines. This means that resources are tied to a particular machine, i.e., the resource name contains the host name of the machine which manages it. This approach is suitable for both read-only and read/write services since resources are not replicated and therefore do not require any additional mechanisms to maintain consistency. Load-distribution is achieved without the use of any special load-sharing techniques but the approach has the disadvantage of unpredictable load distribution. If one set of resources is more popular than others then the machine which manages this set will receive a disproportionate load. Also, redistributing the load, i.e., moving resources between machines, would break both intra-site links and incoming links to the service as a whole, revealing a potentially unmanageable situation using existing off-the-shelf technology.

Serving the same data

The alternative to partitioning the data set is to configure all machines to serve the same data. There are two variants of this approach, either each host possesses its own physical copy of the data set on a local disc or all of the servers share a common data set.

In the case of servers that only support read-only Web transactions, physically replicating the data set is a viable option. However, it is not well suited to read/write servers since data updates will be performed to the individual copies of the data. In certain specific situations, such as a conference registration system, where user data is simply being appended to a log, this may be an acceptable situation, although bespoke tools would be required to merge and synchronise the copies. Furthermore, the fact that multiple copies of the data exist complicates the management task of updating the contents of the server; additional tools (such as the UNIX® tools, `rdist` and `tar`) are required to synchronise the multiple copies of the data.

In order to support general read/write systems and to improve the manageability of the system, it is desirable that all servers share and manipulate a consistent view of the data. There are two ways of supporting this configuration, either through the use of multi-port discs

(only suitable in a closely coupled physical environment) or by using distributed file systems, configured to share a master copy of the data among the servers[2].

Distributing the load in either the replicated or shared-data configurations requires the use of additional techniques. If the Web infrastructure allowed for a loose coupling between resource name and physical address, this would allow support for resource migration, replication or reliability to be introduced at the Web level. Instead, one has to resort to exploiting low-level features of the networking protocol to achieve load sharing.

Load-sharing in a Multi-Host Web Server

The most common form of load sharing currently in use for Web service provision is based upon exploitation of a feature of the Domain Name Service (DNS). The most common implementation of the DNS server software is the Berkeley Internet Name Domain (BIND). BIND version 4.9.3 and above support a *round-robinning* feature which allows a single hostname to be mapped to multiple IP addresses [Albitz92]. For example suppose a Web service is to be hosted by three machines, `www1`, `www2` and `www3`, the details of which are shown in Table 1.

Internet host name	IP Address
`www1.ncl.ac.uk`	`128.240.150.1`
`www2.ncl.ac.uk`	`128.240.150.2`
`www3.ncl.ac.uk`	`128.240.150.3`

Table 1: Illustration of Internet hostname / address mappings

It is possible to create a single name for the three servers by creating special DNS *'A' records* as shown in Table 2.

```
www   IN HINFO WWW-Server    WWW
      IN A     128.240.150.1 ; www1.ncl.ac.uk
      IN A     128.240.150.2 ; www2.ncl.ac.uk
      IN A     128.240.150.3 ; www3.ncl.ac.uk
```

Table 2: Extract from a DNS table configured for round-robinning

When the BIND server is presented with a request to resolve host name, `www.ncl.ac.uk`, it responds with one of the three possible addresses, corresponding to `www1`, `www2`, or `www3`, selected in a round-robin fashion. Therefore two consecutive requests will receive different answers. The result of this technique is that the load associated with `www.ncl.ac.uk` will be shared by the three hosts `www1`, `www2` and `www3`.

There are, however, two main problems associated with this technique. Firstly, the DNS service is organised as a hierarchy; a client passes all resolution requests to a local DNS server, if this server cannot resolve a name it passes the request to another server using well-defined rules. This process continues until the request arrives at the server responsible for resolving the name in question (the *primary*). It is this server that performs the DNS round-robinning. The response is then passed back down the chain to the client's DNS server. To improve efficiency, DNS utilises caching techniques so that each server in the path between

[2] Experiments with this style of configuration by NCSA revealed that NFS was not suitable for such a configuration as access times on the master server were too slow. The optimised client caching in AFS proved a more suitable platform [Katz94, Spasojevic96].

client and server will cache responses from servers further down the chain. The worst case scenario from the perspective of the load sharing is that a DNS server close to the primary caches one of the responses and continues to serve a single IP address thereby resulting in one host receiving a disproportionate percentage of the load. To alleviate this problem, the time-to-live value associated with a DNS entry can be tuned; by shortening the time-to-live the impact of caching can be reduced. However, there is a trade-off associated with this technique, since the lower the time-to-live value, the greater the load on the DNS server[3].

The other main problem is concerned with maintaining service availability in the event of host failures. The DNS service was designed to support data that infrequently changes; it is not well equipped to propagate changes quickly throughout the system of co-operating servers. Therefore, in the event of a server crash, say `www2`, it is not possible to update the whole DNS system, in a timely fashion, to remove `www2` from the server set, therefore, many clients will continue to direct their requests for the service to the deceased machine and after appropriate network-level time-outs will receive a 'service not available' message.

Overcoming this problem requires the use of additional techniques to mask host failures. The next section describes current approaches for achieving this, namely through the use of *High Availability* solutions.

Providing High Availability Web Services

High Availability (HA) solutions were designed to reduce or eliminate single points of failure (SPoF) from systems at a much lower cost than hardware fault-tolerance. HA solutions use a number of mechanisms to protect various aspects of the system, such as RAID and multi-port discs, redundant networks, and uninteruptable power supplies [Sauers96a]. The following description of HA does not aim to provide a comprehensive review of the subject rather it focuses on the specific aspects of HA that are of particular interest to Web service provision. In particular, the mechanisms used for triggering failure masking techniques are not described; they are mentioned briefly later in the section entitled "Limitations of Current Techniques". Furthermore, although most hardware manufacturers offer high availability products, they commonly only provide 'white paper' information rather than detailed technical discussions. This section aims to extract the technical mechanisms underlying such products.

Irrespective of the configuration of the Web server with respect to data distribution, achieving high availability requires the ability for the system to mask failures from clients. This discussion first considers host failures and then application failures. Finally, the implications of the alternative data distribution options are considered.

Masking Host Failures (host failover)

To continue to provide availability in the event of a failure of one of the hosts that comprise the Web service, a second host is required to take over its responsibilities. Since clients access Web resources by directly specifying the IP address of the resources' host machine

[3] HP's main Web site, `<URL:http://www.hp.com/>`, is configured to use round-robin DNS resolution with records having a 10 minute time-to-live value. This compares to 2 hours for the Microsoft site. A typical time-to-live value for the DNS entry of an average host is around 24 hours.

then this takeover has to occur at the network level, i.e., a second host has to takeover the address of the failed machine.

To understand how this takeover can be achieved it is necessary to understand the basic operation of internet routing [Halsall92]. Each autonomous system (e.g., a University network) that comprises the Internet is connected to the core network (the Internet backbone) via an *exterior gateway*. Within each autonomous system there may exist a number of individual local area networks that are interconnected via *interior gateways*. Different routing protocols are employed at each of these three levels. It is the operation of the routing protocol at the local network level that is relevant to the discussion. To enable an interior gateway to deliver datagrams it receives for hosts that are attached to one of its local networks it maintains a mapping between IP `hostid` and the *network point of attachment* (NPA) address for all hosts connected to each of its local networks. In a LAN environment, the NPA corresponds to the *Medium Access Control* (MAC) address. Any host connected to either of an interior gateways' networks can send messages to any other local host by directing the message via the gateway. In old networking hardware, the IP/NPA address pairs had to be entered manually into the gateway whereas modern devices use *the Address Resolution Protocol* (ARP) to dynamically create the mapping tables (the *ARP cache*). To improve efficiency, each host also builds a cache of address pairs to enable direct communication without loading the gateway. In ARP aware hardware, the address pair entries held with the ARP cache are periodically refreshed.

Returning to the previous example of a multi-host Web service comprising three hosts, `www1`, `www2`, and `www3`, Table 3 illustrates an relevant extract from the IP/NPA mapping table, copies of which would be stored at any local interior gateways and at all of the hosts on this and the neighbouring networks.

IP address	NPA	
ip1	mac1	(www1)
ip2	mac2	(www2)
ip3	mac3	(www3)

Table 3: IP/NPA address pairs

Migrating an IP address

Consider the situation when host `www1` fails and `www2` is elected to takeover `www1`'s duties in addition to its own. The networking software on host `www2` can be configured to have `www1`'s IP address in addition to its own. Host `www2` would then respond to any subsequent ARP requests for `ip1` with MAC address `mac2`. However, many ARP caches, including the network's gateways, may maintain the stale mapping from `www1` to `mac1`, for a period of up to tens of minutes, until the cache entry is refreshed [Sauers96b].

Fortunately, this situation can be ameliorated by virtue of a feature of ARP which allows a redirection packet to be broadcast to all listening hosts. The redirection message, which contains a IP/NPA pair (e.g., {`ip1`, `mac2`}), causes the appropriate ARP cache entry to be invalidated and replaced with the address pair contained in the message [Plummer82, Cohen95]. On most UNIX® platforms such a redirection packet can be triggered via the `ifconfig alias` command.

Migrating an IP/NPA address pair

As mentioned briefly earlier, some old interior gateway hardware maintains a static mapping table of IP / NPA address pairs, entered directly into the device. If such devices are present then machine failover cannot be achieved by migrating an IP address alone. Instead both the IP address and the NPA (the MAC address) must be moved.

To support migration of both IP and MAC address requires that all hosts capable of taking over service from others contain additional NPA hardware (network cards connected to the same LAN); one additional card is required for each possible takeover. So, for example, to allow each of the three nodes that comprise our example Web server to each be capable of taking over one other machine, then each host would require two network cards. The secondary network cards must support dynamic configuration of their MAC address (a relatively common feature of current network cards).

In the event of a failover being triggered, the host nominated to takeover its service would configure the MAC address of its secondary network card to be that of the deceased machine and configure its networking software to associate the IP address of the deceased machine to this card This has the effect of transferring service without the use of features of the routing protocol.

Masking Process Failures (application failover)

Service unavailability can equally well be caused by the failure of an application as the host on which it is executes. To address this situation, many HA systems advocate the use of relocatable (application) IP addresses [Sauers96b]. The idea is that unique IP addresses are associated with particular services, and it is via these addresses that clients interact with them. The hosts on which a service executes configures its networking software to respond to the service IP address in addition to its own primary IP address. Applications can be moved between machines by updating the mappings between the application IP address and the NPA, using the ARP aliasing technique previously described. The advantage of employing this technique is the increased manageability it offers, in that an application can easily be moved between hosts, either due to failure, for load-balancing or for maintenance reasons, without causing unavailability of service. The disadvantage of this approach is the potentially large number of IP addresses it may consume.

Data Distribution Options for an HA Web Server

In addition to requiring high availability for the Web server processes, the data which they serve must also be available in the event of failure. The diagram in Figure 1 illustrates two possible data distribution configurations.

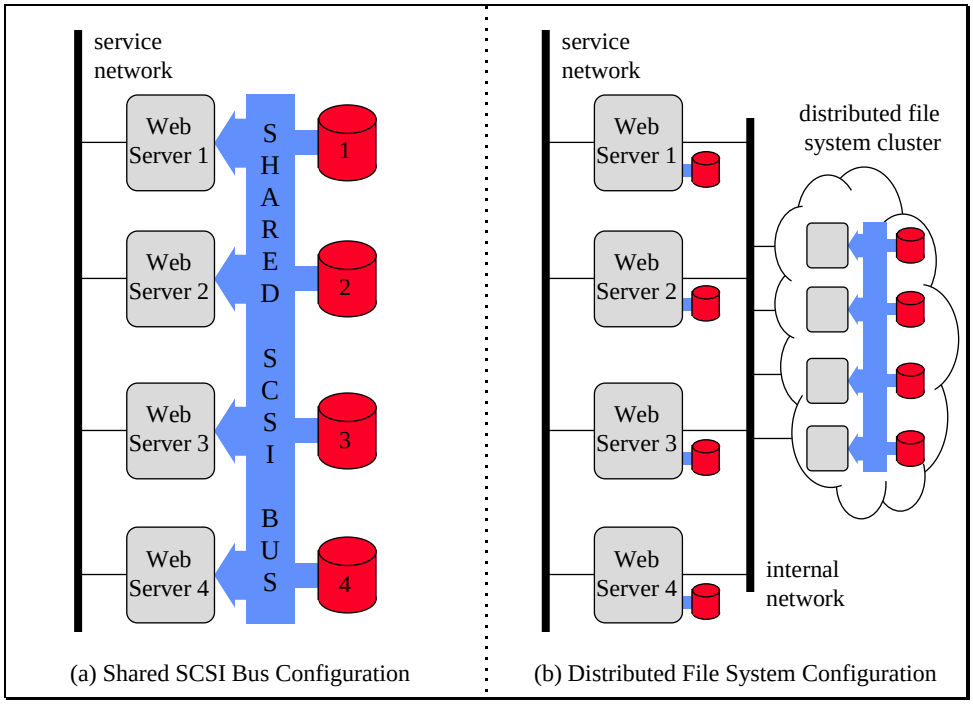

Figure 1: Data distribution configurations for a HA Web server

The shared SCSI bus configuration, shown in Figure 1(a), is a suitable configuration for a server cluster operating with partitioned data sets. Under normal operating conditions each Web server obtains an exclusive reservation on its primary disc, i.e., server 1 serves data from disc 1, etc. In the event of a host failure, say, server 1, then in addition to taking over the IP address of the failed host, as previously described, the designated replacement machine, say, server 2, reserves the disc of the failed host in addition to its own. Server 2 is then capable of serving any requests for server 1's data in addition to its own.

For server clusters that require a shared data configuration, usually necessary for a server that is performing read/write access to data, a two tiered configuration is required as shown in Figure 1(b). The data set is stored within a distributed file system, with the data being distributed across a number of servers as load demands. Each of the Web server hosts mount and serve the same data set from the distributed file system. AFS appears to be best suited to this task by virtue of its client side caching features that allows copies of frequently accessed data to be cached on the local discs of the Web servers. This dramatically improves performance since read-only operations can be performed locally. Write operations are *written-through* to the master copy and cache-consistency mechanisms ensure that *dirty-data* is not accessed. In this configuration the distributed file system is required to be highly available. This can be achieved using the shared SCSI bus techniques described previously.

Limitations of Current Techniques

This section reviews the strengths and weaknesses of the current state-of-the-art techniques for high-end Web service provision and highlights where research techniques could be applied to overcome some current limitations.

Guaranteeing Data Integrity

The HA techniques that have been described are solely concerned with masking process and host failures from clients. For a tightly-coupled Web server that is solely providing read-only access to data, these mechanisms are sufficient. However, the provision of read/write services require additional mechanisms to guarantee integrity of data in the presence of failures during write operations. Similarly, with multiple servers potentially accessing data concurrently, techniques are required to provide serialised access to data to prevent corruption. Programming environments such as Arjuna [Parrington95] greatly simplify the task of implementing such systems.

Alternative Load-Balancing Techniques

The DNS round-robin techniques for load balancing suffer from several shortcomings:

- DNS is not capable of determining the availability or performance of a given server and will continue to send client requests to overloaded or failed servers, hence the requirement for HA solutions to mask such failures.

- Round-robin DNS distributes traffic in a cyclic manner directing equal numbers of requests to each of the alternative servers. In some configurations this may be inappropriate, e.g., if the Web service hosts differ in terms of performance.

- To minimise the effects of caching, DNS entries are configured to have short time-to-live value. This result in increased load on the primary DNS server.

- Round-robin DNS does not scale continuously since records are restricted to 32 entries, translating into a maximum configuration of a 32 host Web server cluster.

A better approach would involve clients being aware of a single *service address* corresponding to the service as a whole[4]. Load balancing could be implemented within the server cluster itself. Such a configuration would be scalable in that more hosts could be added as the demand increased and in the event of a host failure, service could continue without requiring any special address-swapping techniques.

Group Communications

One approach would be to accept all incoming service traffic at a single machine and then relay the request to a group of slave hosts, who would decide among themselves who will service the request. This is a group communications based solution. The main drawback with this approach is the fact that all machines in the cluster will be interrupted with all incoming

[4] Multicasting immediately comes to mind as an appropriate technique to support such a configuration. Some local area network technologies, e.g., Ethernet, support hardware multicasting, and many modern operating systems provide support for its use. Furthermore it is possible to map multicast Ethernet addresses to special group IP addresses (ranging from 224.0.0.0 to 239.255.255.255) [Comer95]. IP multicasting on a LAN environment is therefore a viable proposition. However, to use multicast IP over the Internet requires additional routing support, which is currently in an experimental phase, the most visible large-scale experiment being the mbone. Multicast IP is also a datagram based, whereas HTTP is a connection-oriented protocol, built on top of TCP, so protocol conversion would be necessary, requiring customisation of the protocol stack of the router machine on the server cluster LAN.

requests. Furthermore, there is the system overhead in deciding whether to accept or drop a given packet. Any algorithm used to make this decision must be aware of the membership of the server group. One approach would be for the group to occasionally exchange group membership messages. This would reveal a small window in which some messages may not be answered, i.e., during the time between a host failure and the next membership update round. To achieve 100% service, then group membership would have to be validated for each message, requiring a second round of message transmission, as in the *virtual synchrony* protocols. The overhead introduced by this scheme would likely be prohibitive in many situations, however, it has the advantage of not requiring closely coupled cluster environments.

Network Address Translation

Network address translation (NAT) is a technique for dynamically altering the destination address of a particular IP packet at a network border (i.e., a gateway) [Egevang94]. The mechanism operates by editing the IP headers of packets so as to change the destination address before the IP to NPA translation is performed. Similarly, return packets are edited to change their source IP address. Such translations can be performed on a per session basis so that all IP packets corresponding to a particular session are consistently redirected.

This technology can be applied to Web service load distribution over a host cluster. All clients communicate with the service by specifying a single IP address. At the gateway to the Web cluster network, the gateway can redirect incoming requests to one of a number of slave hosts. Clearly, there are fault-tolerance issues to be addressed in that the gateway is a single point of failure, so redundant networking and gateway hardware is required to overcome this.

Commercial products supporting this technology are just beginning to appear, the *LocalDirector* product from Cisco appears to be the first [Cisco96a]. LocalDirector performs redirection in an intelligent manner by monitoring the response times of the server hosts and directing requests so as to maximise the QoS as perceived by the client. In the event of a host failure, its response time becomes infinite and receives no subsequent requests until it returns to service.

Providing Widely Distributed Services

For services that are inherently centralised a combination of DNS round-robinning and HA systems or the new NAT-based scheme, offer a simple and effective solution to the provision of highly available Web services that are capable of supporting high load. However, these techniques only operate in closely coupled systems, in particular the HA techniques require a common LAN for the IP aliasing mechanisms and a shared SCSI bus to allow takeover hosts to access the data of a failed one. For accurate failure detection, multiple communication paths between servers, for *heart-beat* message exchange, are also required.

There may be installations where this style of server in insufficient. Consider the requirements of a large multinational company’s Web service provision. An ideal installation would consist of multiple Web access points in the various countries in which the company operated. This would have the advantage of improving the quality of service offered to customers, in terms of reduced access latency, since both the processing and communication load would be distributed. In such a scenario it is likely that the various access points would offer some common data and services customised with data of local interest.

Such a configuration raises the same two issues that have been discussed previously in closely coupled environments, that is, data distribution and load distribution.

Data Distribution

To support a widely distributed configuration performance reasons would dictate that the common data would have to be replicated at each of the individual sites with a requirement to maintain consistency in the event of read/write data manipulation.

A system based upon an AFS distributed file system with site-based caching would be one approach to implementing such a system[5]. However, such a scheme is only really capable of supporting read-only Web services since AFS only provides UNIX® file system semantics, it does not enforce serialised access to data under concurrent access or guarantee integrity of data in the event of failure, which is particularly important when updating multiple items of data.

Such a system could be implemented by using a combination of the techniques previously described, to provide high availability and load-balancing at each access point, with distributed computing techniques, such as reliable group communication and atomic transactions, for reliably manipulating a replicated data set.

Load Distribution

Directing clients to the most appropriate server for their particular locality can be achieved in several ways. In many current sites, users are asked to indicate their geographical location or a preference for a particular server and then select an appropriate URL from a list presented on a Web page. DNS round-robinning could also be used to automate distribution but this is not scalable and doesn't take client locality into account with a result that clients may not see the optimum QoS.

A new commercial product from Cisco, the DistributedDirector, aims to perform automatic selection of the optimum server for a particular client by utilising routing information inherent in the network [Cisco96b]. DistributedDirector can operate in two modes, as a DNS server, suitable for redirecting multiple application protocols, and as a HTTP redirector. In DNS mode, it acts as the primary nameserver and replies with a single address of the appropriate server. In HTTP mode, it acts as a Web server, accepting incoming HTTP requests and returning HTTP code 302 (temporarily moved) to redirect clients to the appropriate server.

In order to determine the optimum server for a particular client several different metrics are used, the most interesting use a proprietary protocol to query software agents running on the gateway devices closest to each of the distributed servers. The query contains the client address and the agents use the routing table information to determine the number of hops between the client and the particular server. The DistributedDirector collates the responses

[5] The Web services of the ESPRIT Networks of Excellence are implemented using this scheme, with access points in the UK (Newcastle University), France (LAAS-CNRS at Toulouse), the Netherlands (University of Twente), and Portugal (INESC at Lisbon). Access to data via the Web is read-only.
See <URL:http://www.newcastle.research.ec.org/>.

and chooses the host that is *closest* to the client. In order for this technique to work it requires that the appropriate agent software is running at each of the distributed sites, this naturally requires Cisco gateway systems.

An alternative approach would be to utilise an application-level protocol to monitor the status of the distributed servers and use this information to redirect requests from a master site. Such an approach has the advantage of not requiring any specialist routing hardware.

HP CORBAWeb

System Overview

In the CORBAWeb system, the Web server is typically configured to pass requests for URLs whose base name is "`/orb`" to the CORBAWeb module which runs in the server. The module acts as a CORBA client; the required service is extracted from the remainder of the URL. For example, a request for `<URL:http://www.service.com/orb/bank>` will cause the client to look up the 'bank' gateway in the Nameserver and bind to it. The client then executes the `invoke()` method of the gateway, passing it any associated parameters, typically data that has been entered into an HTML form and passed from the browser using URL-encoding. The gateway performs the necessary processing and returns a response (usually HTML) to the CORBAWeb module which returns it to the browser, thereby completing the request.

A predicted use of this architecture is to provide Web connectivity to CORBA *back-office* applications (or CORBA-wrapped applications). In this situation, the gateway interprets the incoming URL-encoded data, performs the appropriate invocations on the back-office application and provides the necessary presentation logic to wrap the result, creating an appropriate response for the Web client.

The diagram shown in Figure 2 illustrates CORBAWeb architecture from a process-level perspective.

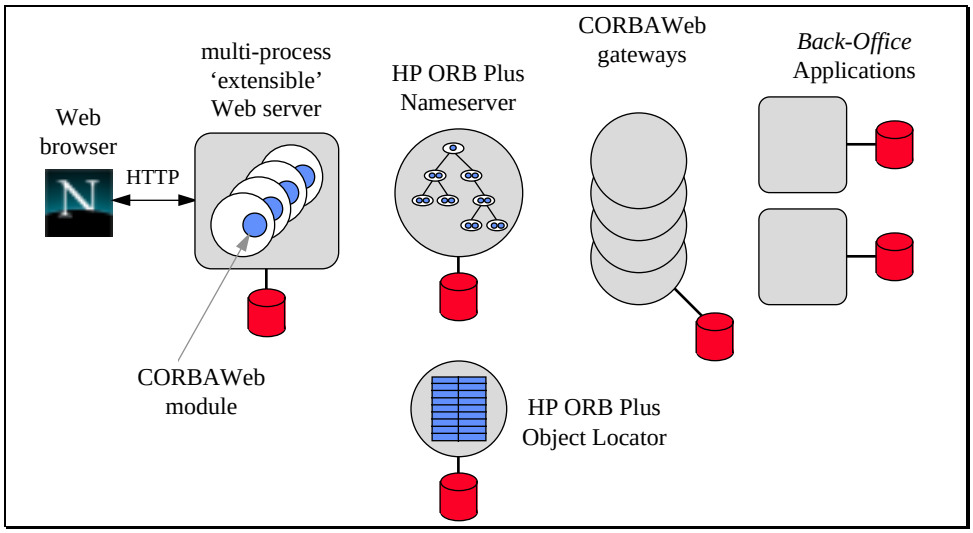

Figure 2: CORBAWeb architecture (process-level view)

Dependability Analysis of CORBAWeb Architecture

Analysing the CORBAWeb architecture from a dependability perspective left to right as illustrated in the diagram in Figure 2

Web Server and CORBAWeb Module

The combination of the Web server and the CORBAWeb module essentially acts as a protocol converter, translating HTTP requests into CORBA object invocations. Although, the CORBAWeb module may hold some state as a performance optimisation, such as

maintaining an object reference cache, no critical, persistent, state is required to be maintained or manipulated. Therefore, from a dependability perspective, the challenge is to provide high availability of the service and possibly support load balancing over multiple hosts should popularity deem it necessary for a particular installation.

Most current web servers, including Apache & Netscape, use a multi-threaded, multi-process architecture operating in a master/slave configuration. The master process accepts connections on the main port (usually 80, the *well-defined* HTTP port) and then passes the request on to one of the slave processes which actually complete the request. Therefore, multiple instances of the CORBA client will exist, i.e., one per server process.

This architecture provides some degree of process-level fault-tolerance, since the master server will recreate processes that terminate abnormally. In addition, the master will intentionally kill a slave process after it has performed a given number of requests, so as to minimise the possible effects of poor programming, notably memory leaks. This simple technique works successfully by virtue of the fact the server and plug-in modules do not maintain persistent state.

CORBA Processes

In the current CORBAWeb architecture, the Object Locator, the Nameserver, and the gateway processes are all potential single points of failure.

The diagram in Figure 2 does not illustrate host boundaries since the architecture allows for many different distribution configurations, for example, there is no requirement for the Object Locator, the Nameserver, or the gateway processes to be co-located with the Web server processes.

The ORB Plus Object Locator

CORBA objects are accessed via an *object reference*, an opaque string which is used by the ORB to locate and bind to distributed objects. CORBA objects may either be *transient*, an object exists for the lifetime of its containing process, or *persistent*, an object lifetime is not restricted to that of the process which created it (its *birth server*). In ORB Plus, persistent objects reside within *virtual servers*. Over the lifetime of a persistent object, its virtual server may reside in many different processes, however, its object reference is guaranteed to provide access to the object for its entire lifetime. In order to support this model, the ORB requires a mechanism for locating persistent objects based on their object reference. In ORB Plus, this is achieved via the *Object Locator* daemon, which resolves requests for object references for persistent objects which can no longer be accessed at their birth server[6]. The locator is also able to start server processes for virtual servers that are not active.

Considering the effect of the failure of the Object Locator daemon responsible for the CORBAWeb location domain. According to ORB Plus documentation, object references that were valid at the time of the failure will continue to provide access to the objects to which

[6] The ORB Plus 2.01 on-line documentation provides details of the information maintained within an object reference and the algorithm used by the Object Locator to track persistent objects. The relevant extracts are included in Appendix B.

they refer for as long as those objects remain in their existing server processes[7]. If a process holding a virtual server terminates (due to inactivity for example) and is subsequently requested then the CORBA client will be unable to bind to the object and the service will therefore be unavailable.

Unlike the Nameserver and the gateway processes, the Object Locator resides at a well known address (hostname and port) for a given location domain. In the event of the failure of the Locator's host, the daemon cannot simply be restarted on an arbitrary host, rather the system will be unavailable until the host recovers.

The ORB Plus Nameserver

ORB Plus provides a Nameserver, compliant to naming service specified in the Common Object Services Specification (COSS) [OMG95]. The Nameserver allows a hierarchy of user defined names to be created which are mapped to object references, thereby removing the need for clients to manipulate object references directly. CORBAWeb uses the Nameserver to map from the service names, as specified in the URLs presented to the Web server, to the object reference corresponding to an object providing the service.

Should the Nameserver process fail, the next time an attempt is made to use it direct communication with the server at its old location will fail and the Locator process will be contacted to determine its new location. The locator will determine that the Nameserver is not currently running and can be configured to restart it, passing the address of the new process back to the client.

Should the Nameserver's host fail, then the process could be restarted on a secondary machine but again since persistent data is stored on the file system then this data must be available on the secondary host for the takeover to succeed. This could be automated by registering intelligent startup scripts with the Object Locator.

Gateway processes

In common with the Nameserver, the gateways are potential single points of failure, albeit with more localised effects, i.e., a gateway process failure will cause only the service which it provides to be unavailable, other services (provided by other gateways) will continue to operate. As with the Nameserver, the Locator will attempt to restart failed servers which, in the case of failed hosts, could be configured to restart on a different host.

More significantly however, gateway failures have the potential to compromise the integrity of data held within back-office applications. In certain configurations, a single Web request may cause a gateway to perform update operations on a number of back-office objects, e.g., a money transfer between two accounts. In the event of a process or host failure during this update the overall system state may be left in an inconsistent state, e.g., the sum of the two account balances in the banking example may be different to that before the transaction took place.

[7] It is assumed that the CORBAWeb modules within each of the Web server processes maintain an object reference cache, typically containing object references for the Nameserver and other gateways that have been recently accessed.

CORBAWeb Availability using HA Techniques

One approach to providing a highly available CORBAWeb system would be to use off-the-shelf HA products to mask process and host failures. Suitable products include HP's MC/ServiceGuard [Sauers96a], Digital's DECsafe [Cohen95] and Veritas' FirstWatch [Veritas96].

To provide a highly available service, it is first necessary to provide failure masking for the Web server and since clients communicate directly by specifying the IP address then IP switching is the only available option. For a particular location domain, the Object Locator process also resides at a specific address (host and port). Therefore IP aliasing is also required to tolerate failure of the Object Locator's host.

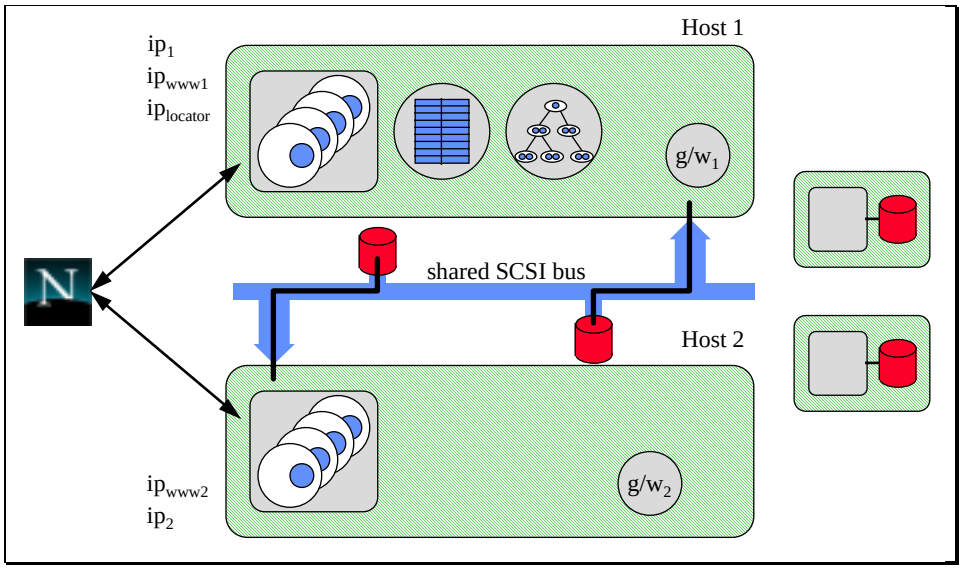

Figure 3: CORBAWeb availability using HA techniques (normal operating condition)

The diagram in Figure 3 illustrates one possible two-host configuration operating under normal, i.e., failure free, conditions. Web server processes are active on both hosts, sharing incoming load. The Object Locator and Nameserver are running on host 1 and gateway processes are running on both hosts, g/w_1 on host 1 and g/w_2 on host 2.

To improve the manageability of the system, the Object Locator and each of the Web servers could be allocated their own IP addresses ($ip_{locator}$, ip_{www1}, ip_{www2} respectively) as shown in the diagram. This would allow graceful migration of services between hosts for maintenance purposes.

The failure of host 1 would trigger the HA system running on host 2 to configure its networking software to takeover the IP address of the services that were running on host 1, i.e., ip_{www1} and $ip_{locator}$, in addition to its own. ARP redirection packets would be broadcast to update these changes to the LAN routers and other local machines, i.e., $\{ip_{www1}, mac_2\}$ and $\{ip_{locator}, mac_2\}$. The Object Locator would then be started locally on Host 2. The Nameserver and gateway process, g/w_1, that were active on Host 1 at the time of its failure are restarted on Host 2 when a CORBAWeb client next requests a binding to them via the Object Locator. The system configuration after the failover is shown in Figure 4.

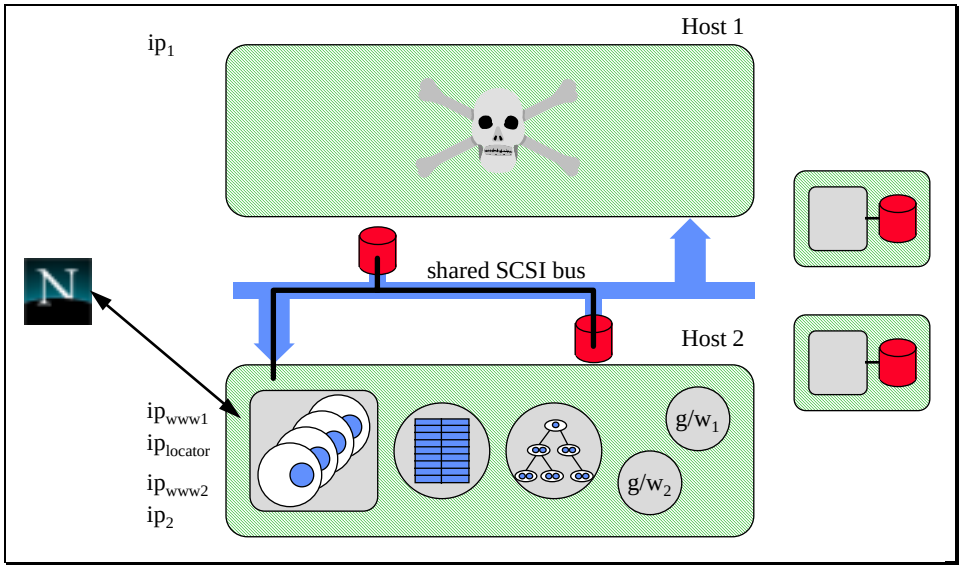

Figure 4: CORBAWeb availability using HA techniques (after failure of host 1)

Since the Object Locator, the Nameserver and optionally, the gateway processes maintain persistent state on the file system then the failover would require this data to be available to the secondary machine, either using a shared SCSI bus (as shown in the diagram) or a distributed file system (which would also be required to be highly available).

There are two primary limitations with this approach, namely, integrity of persistent state and possible message loss.

The Object Locator and Nameserver processes periodically update their persistent data. It is not known whether such updates are atomic or not (i.e., whether they are achieved in a single-block disc write operation). If these updates are non-atomic then it is possible that a host failure during such an operation could result in the persistent data being left in an inconsistent state. Such a situation could prevent the process from starting up and operating correctly on a secondary host. Similarly, the gateway processes would have to be carefully coded so that their persistent state updates would atomically move the state from one consistent view to another.

There is also a possibility that some messages may be lost during a host failover. Consider, as an example, a situation where the Object Locator on Host 1 has initiated the activation of a gateway virtual server on host 2 and then fails. When started, the gateway process will try to register itself with the Object Locator. The registration may fail if the failover is not complete, i.e., the Object Locator has not yet restarted on host 2. It is thought that the virtual server startup algorithm would handle such a situation by terminating the gateway process should its Locator registration fail. However, this assumption would have to be validated and other scenarios may not be catered for.

CORBAWeb Availability using Replicated Processes

An alternative approach to providing a highly available CORBAWeb service would be to use process replication techniques. Somersault [Harry96] is a fault-tolerant distributed system based on active process replication. The Somersault FT Orb is implemented by layering Somersault underneath HP OrbLite to provide a fault-tolerant transport [Harry95].

Using the FT Orb, a virtual server may be actively replicated to form a *recovery unit* (RU) which consists of two processes, the *primary* (to which messages are first delivered) and the *secondary*. Somersault guarantees the consistency of the replica copies and supports very fast failover from primary to secondary in the event of a failure of the primary process. The system strives to maintain the cardinality of the group, so if the secondary process fails, the primary would start a new secondary and if the primary fails, the existing secondary is promoted to primary and then creates a new secondary. During such a failover, the system guarantees that no incoming messages to the RU are lost.

Client access to replicated objects is transparent, that is, they bind to what appears to be a single object by specifying its Object Reference.

Implementing replicated objects using FT Orb requires a few programming changes compared with a normal CORBA application. The first major difference concerns non-deterministic events which have to be identified, isolated and programmed using special non-deterministic choice objects so as to ensure that both the primary and secondary make the same decision in non-deterministic situations. Secondly, state transfer objects must be provided, which are used within the RU to transfer object state between the replica copies.

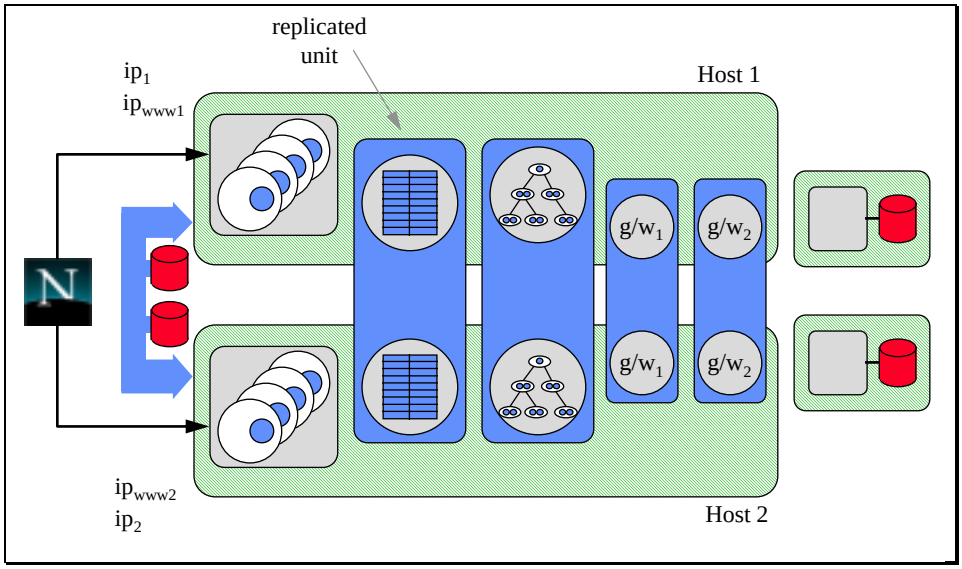

Figure 5: CORBAWeb availability using Somersault FT Orb

Figure 5 illustrates an available CORBAWeb configuration based upon the Somersault FT Orb model[8]. Since it is desired that CORBAWeb be used with standard unmodified Web servers, traditional HA techniques are used to provide high availability of the Web server component of the system, which operates in the same way as in the previous configuration.

The gateway processes are replicated within recovery units so that in the event of a failure of either host they will continue to provide service. Introducing a third host would allow Somersault to continue to maintain two active replica copies in the event of a single host failure.

Data Integrity of Gateways using Atomic Transactions

The gateway processes are potentially the most sensitive part of the CORBAWeb design since they are able to perform read/write operations on back-office objects. It is imperative that the integrity of the data managed by these objects is not compromised.

In the simple case of a CORBAWeb gateway that updates the state of a single back-office object as a result of a Web request. Concurrency control mechanisms are clearly required in order to prevent update conflicts in the event of concurrent access.

A more complex gateway may manipulate the state of multiple objects in response to a single Web request. An example would be an electronic shop that is required to update a number of separate sub-systems in response to an order request, for example, payment processing (using SET-like systems), stock control, product dispatch etc. In such cases, it is necessary to ensure that the consistency of the entire system state (composed of the states of the individual objects) is maintained even in the presence of failures, either of the gateway or the object servers themselves.

The recognised approach to achieve these goals is to implement the system using Atomic Transactions, which guarantee that either all of the work conducted within the scope of an atomic transaction will be successfully performed or it will all be undone. The CORBA Object Transactional Service (OTS) specification describes a system that is able to co-ordinate atomic transactions comprised of operations on CORBA objects so as to provide the aforementioned guarantees.

An OTS service would provide a good fit for the fault-tolerance requirements of the CORBAWeb gateways assuming that the back-office objects could be implemented as transactional objects. Legacy applications could be wrapped by transactional CORBA objects if they themselves were implemented in an open transactional manner or could be made to behave in such a way. This is an important requirement as the wrapper objects must conform to the two-phase protocol as controlled by the OTS service, that is, it must be possible to atomically commit or roll-back state manipulations.

[8] It is not precisely known how the Somersault FTOrb implements replicated equivalents of the Object Locator and Nameserver; the diagram assumes they are implemented as RUs, this requires verification with the Somersault team.

Conclusions and Suggested Further Work

General Web Dependability Issues

Many organisations have a requirement to support Web sites that are able to cope with large numbers of requests while preserving high levels of quality of service (QoS). Although currently the majority of Web requests are read-only and computationally non-intensive operations, the high demands placed on network I/O often means that the load cannot be supported by a single machine. To address this issue many large Web sites have adopted cluster-based Web servers in which the load is distributed across a number of machines.

The most common load distribution technique in use today is the round-robin DNS solution, which effectively distributes the set of IP addresses of the server machines among the various clients. This results in unintelligent load distribution; no account is taken of machine failures or system load when distributing requests. Therefore, in order to provide high levels of QoS, additional High Availability (HA) techniques are required in order to mask system failures from clients. These solutions necessitate that the server cluster shares a common LAN. An alternative approach, based on the concept of Network Address Translation (NAT) is beginning to emerge. Products such as Cisco's LocalDirector operate by performing network-level manipulation of incoming requests, directing them to appropriate machines in an intelligent manner. Although care must be taken to ensure that the routing hardware does not become a single point of failure, this technique is thought to be an improvement over the DNS-based approach for tightly coupled servers.

It is interesting to note that the established distributed computing solutions for providing load balancing and fault-tolerance do not appear to be well suited for current high-load Web services due to the low computation to network I/O ratio. For example, group communications techniques would seem appropriate for a fault-tolerant multi-host Web server, however, the additional messaging cost typically introduced would adversely affect system performance. However, such an approach removes the locality restrictions inherent in the mechanisms previously described, potentially allowing widely distributed servers to be constructed. Furthermore, in future the feasibility of these techniques may increase as the balance between computation and network I/O is expected to change; Web applications are likely to increase in complexity and next generation Web protocols promise to better utilise network resources.

In multi-host servers there are two options for data distribution: data is either replicated or shared. The appropriate choice depends upon the nature of the service. For exclusively read-only services configuring each server with its own copy of the data is feasible although potentially difficult to manage. For read/write services the additional complexity of managing replicated data effectively means that a shared-data configuration is preferred. Distributed file systems offer an appropriate solution; each server accesses a single copy of the data held within the distributed file system. AFS appears well suited to the task due to its high performance achieved through the use of local client (i.e. Web server machine) caching. In order to provide resilience to system failures, the distributed file system should be made highly available.

For complex Web applications that perform write-based operations on back-office system data, concurrency control mechanisms are required to ensure that the integrity of data is

preserved in the event of concurrent access. Furthermore, the use of atomic transaction mechanisms are advised to protect against the corruption of data in the event of system failures during write operations that affect multiple items of data. Systems such as Arjuna or CORBA OTS are appropriate solutions.

HP CORBAWeb-specific Issues

A CORBAWeb installation can be logically divided in to a number of components: the Web servers, the ORBplus Object Locator and Nameserver, the CORBAWeb application gateways, and the back-office applications. Each of these components must be made reliable for the whole system to be considered dependable.

The front-end Web server is a potential single point of failure in the system. The previously described HA mechanisms for providing IP level failover are applicable here.

Our discussion showed how HA techniques could be used to make the Object Locator, the Nameserver and the application gateways tolerant to system failures. In particular IP aliasing could be used to break the mapping between the Object Locator and a physical host. The viability of this approach depends somewhat on the implementation of the ORB Plus components themselves, since recovery relies upon the integrity of the persistent data maintained by the Locator and the Nameserver to be maintained in the event of host failure. In effect, updates to this persistent data must be performed atomically, i.e., in a single disc-block operation. If these semantics cannot be guaranteed then the HA approach is not appropriate. The alternative solution based on the use of replicated processes would not suffer from this limitation. Furthermore, failovers would be completed much faster using replicated processes compared to the HA approach, although it is not clear whether the difference is critical in this application.

The application gateways are a critical part of the CORBAWeb in terms of reliability. The typical role of a gateway is to provide a Web interface to back-office systems. Therefore it is the gateways that instigate manipulation of system state. Complex gateways may manipulate several items of data in response to a single Web request. Therefore it is essential that such gateways are designed to ensure that integrity of data is preserved in the event of system failures. Atomic transactions are best suited to this task. In the CORBA world, the most obvious candidate for an implementation of atomic transactions would be the CORBA Object Transaction Service and its associated components (e.g., persistence, recovery, concurrency control).

Future Directions

This document has focused on the provision of dependable back-end Web systems including providing high availability of service, supporting high load, and ensuring the integrity of back-office data in the event of failures. The next phase of our work will investigate how dependability can be extended to encompass clients. Specifically, we propose to investigate:

End-to-End Reliability

Our previous discussion of transactions for Web-based services has assumed that Web clients only provide the stimuli for transactions that execute solely within the back-office systems, that is, a client is not involved as a party within the transaction. In this scenario, transactions

are being used to maintain the integrity of the overall back-office system state, which may be composed of several distributed items of data. This approach has also been proposed by several commercial organisations [Transarc96].

The next phase of our work will examine the advantages to be gained from incorporating the browser itself as a party within a transaction. We shall consider the options available for allowing the browser to participate in transactional commit protocols. Initial investigations [Little96] reveal a number of issues, including the question of how crash recovery mechanisms can be supported.

The commit protocol of traditional transactional systems specifies a recovery protocol that ensures that all transactions will eventually be resolved after an arbitrary number of system failures. To support these protocols each party is typically required to maintain persistent data detailing the progress of the action and the work carried out within its scope. The lightweight and volatile nature of Web browsers will complicate the task of implementing browser-based transaction objects. For example, although Java allows complex applications to be executed in a browser, security issues may prevent transactional Java applications from accessing the resources of the local machine, for example, to make persistent essential transactional information. These restrictions mean that new approaches must be developed to gain the benefits of end-to-end reliability while placing minimum reliance on the browser itself.

Our work will investigate this, and other, issues, attempting to evaluate the various available options in terms of their relative merits and weaknesses. Part of our evaluation will be an investigation of the ADMS protocol from VISA [Visa96], a non IP based protocol intended to provide end-to-end guarantees for home banking applications.

Inter-Enterprise Transactions

Currently, Web interactions occur between an individual and an organisation. We have described how atomic transactions may be used to co-ordinate and insure the integrity of updates to items of the organisation's back-office system state. One possible progression from this is to consider enterprise to enterprise transactions where system state within both organisations is required to be updated in a transactional manner. Consider the following example:

- an individual within an organisation initiates a transaction with a merchant, for example, to order some stationary;
- the merchant establishes the identity of the client and recognises that he is part of an organisation, for example, Newcastle University;
- the merchant provides a service appropriate to the client's particular organisation, for example, special price deals may have been negotiated at the organisational level;
- the merchant's processing of the transaction may include back-office processing, for stock control, billing etc.;
- in addition, as part of the transaction, the client interacts with the local IT system, for example, to update the local stock control system to register that the items have been ordered;
- it is desired that this entire interaction is protected by atomic transactions to ensure ACID properties.

Logically such systems could be implemented by simply allowing a single transaction to encompass operations on back-office data within both organisations. However, security policy would likely prohibit such operations since they would reveal the internals of the enterprise's IT system. Furthermore, structuring such potentially long-running activities as single transactions may cause locks on resources to be retained unnecessarily. A more appropriate approach would be to *glue* together independent transactions running within each organisation, with a requirement to preserve the atomic properties of the overall transaction.

We will investigate how such systems can be implemented; an initial approach would be to evaluate the appropriateness of transactional workflow techniques as an underlying architecture to support such systems.

Appendix A: Dependability Analysis of Secure Electronic Transactions (SET) Protocol

Introduction

The Secure Electronic Transaction (SET) specification describes a protocol to allow secure payment card transactions to be performed over open networks, such as the Internet [Mastercard95]. The SET protocol has been specifically designed to address this single well-defined transaction case and therefore differs greatly from systems such as the CORBA Object Transaction Service (OTS) and Arjuna which support arbitrary multi-party application-specific atomic transactions. In particular, the protocol has been engineered to remove the requirement for complex recovery protocols in the event of system failures. We now provide an overview of the SET protocol before examining its resilience to system and communication failures.

SET Overview

SET describes a three-party transaction between the cardholder, the merchant, and the acquirer payment gateway (APG), the representation of the merchant's financial institution providing payment card services. To protect the transmission of sensitive financial data over open networks SET specifies detailed encryption mechanisms that are applied to the basic payment protocol to ensure its security. This discussion concentrates only on the payment protocol, the exchange of messages between the three parties is summarised in Table 4, followed by a brief explanation of the various stages of processing.

Cardholder	Merchant	Acquirer Payment Gateway
------PInitReq---->		
<-----PInitRes-----		
--------PReq------>		
	------AuthReq----->	
	<-----AuthRes------	
<-------PRes-------		
-------InqReq----->		
<------InqRes------		
	-------CapReq----->	
	<------CapRes------	

Table 4: SET payment protocol overview

Initialisation

This pair of (optional) messages, PInitReq and PInitRes, between the cardholder and the merchant are used for initialisation of the system. A global transaction identifier is established together with both cardholder and merchant local ids to simplify processing. This exchange also ensures that the necessary authentication certificates are held by both

parties. As with all messages, random challenge variables are included to ensure freshness of messages.

Purchase

The purchase order is the main component of the payment protocol. It confirms the cardholders commitment to buy the goods. PReq consists of two components, the Order Instruction (OI) for the merchant and the Purchase Instruction (PI) which is passed via the merchant to the APG. The response message, PRes, is sent once during the protocol, when the merchant is ready to complete the purchase with the cardholder, most commonly after authorisation, but it may occur before authorisation or after capture.

Authorisation

In the authorisation request, AuthReq, the merchant sends the data about the purchase to the APG together with the PI from the cardholder. Since the hash of the payment amount is included in both the merchant data and the cardholder's PI, the APG can verify that the cardholder and merchant agree on the amount. The PI also contains the card details that the APG uses to authorise the appropriate payment amount using the existing payment card financial network. The response message, AuthRes, indicates whether or not the payment is authorised.

Capture

The pair of messages CapReq and CapRes complete the payment of moneys previously authorised through one or more authorisation transactions.

Inquiry

These (optional) messages may be instigated by the cardholder at any time after PReq has been sent to inquire as to the status of the transaction. The response message, InqRes, contains the same information as PRes. The status of the transaction may either be awaiting authorisation, awaiting capture, or complete.

Dependability Implications of SET

The above description of an SET payment transaction illustrates how the protocol has been engineered to place minimum reliance of the cardholder software component. The sending of the PReq message confirms the cardholder's commitment to the transaction. After receiving this message, control is effectively passed to the merchant. In a failure free transaction, the cardholder will receive an acknowledgment message, PRes, indicating that the transaction has been accepted. It does not, however, indicate that the transaction is complete, it may be awaiting authorisation or capture. If required the cardholder may interrogate the status of the transaction using the inquiry request/response pair. This is an appropriate strategy since the cardholder software will be run on standard PCs and is therefore likely to be the most volatile of the three parties.

To provide resilience to communication failures, which are likely to be common over internet networks, SET incorporates a simple retransmission mechanism with duplicate detection. In effect, all messages are wrapped with an unencrypted header block which contains the global transaction identifier and a request/response pair identifier (RRPId). If a message is sent and no reply is received within a specified time-out period then a bit-wise identical copy of the message is simply retransmitted. In the event of the reception of a duplicate message, the

message processing code simply retransmits the previously-sent response, thereby making message processing idempotent[9].

Although the payment protocol appears to have been designed so as to simplify the task of providing resilience to system failures, the specification omits any discussion of how failure recovery is achieved. Furthermore, it fails to specify a number of implementation considerations that are thought to be essential in order to support successful failure recovery. As an illustration, consider the processing performed by the merchant component, which may be thought of as a state machine as shown in a simplified manner in Figure 6.

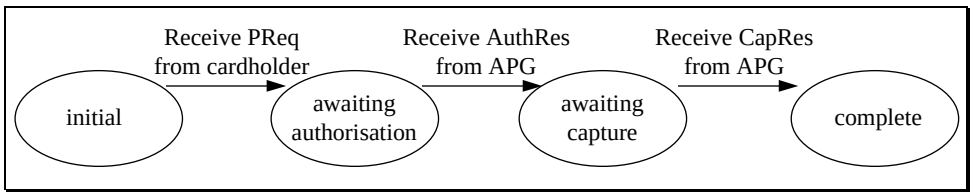

Figure 6: Merchant software state machine

Considering a typical transaction; after the data from PReq has been processed, the transaction enters the awaiting authorisation state. Following the exchange of authorisation messages with the APG, the transaction enters awaiting capture. Finally, after the capture exchange with the APG the transaction is complete. At some installation-specific stage the PRes message is returned to the cardholder.

The specification details precisely how each message should be processed, including which items of data from the message payloads must be stored within the merchant's database. Each transaction exists as a record in the database, keyed by the transaction identifiers that are included in each message. Each transaction can therefore be considered in isolation; a message is received, the appropriate record is retrieved, updated, and written back. In order to provide resilience to system failure it is essential that such database state changes are performed atomically, i.e., in a disc-based system the update of a record must be achieved in a single disc block operation. If state changes are non-atomic then, in the event of a failure, there is an opportunity for updates to only be partially completed, thereby potentially corrupting the database.

The aforementioned message retransmission mechanism that is used to provide tolerance to communication failure also raises issues related to system failures. As an example, consider a transaction that is awaiting authorisation; during the preparation of the AuthReq message the merchant crashes. There are two possible scenarios, the crash either occurred before or after the message was sent, in the latter case, the AuthRes from the APG would naturally be lost. Upon recovery, the transaction's database record would indicate that it is awaiting authorisation. The merchant would then create and send a new AuthReq message. However, if the first message was sent then this second message would break the requirements of the specification since it is in effect a repeated message, but is not bit-wise identical to the original (e.g., it would have different challenge variables.) In this particular case, it is unlikely

[9] There are several exceptions to this, for example, inquiry messages should not be idempotent since multiple distinct inquiry messages can be sent over the lifetime of a transaction. Appendix C of Book 2 of the SET specification expands on this.

that the APG would be confused since it is likely to have updated its own record of the transaction with the result of the first authorisation request. However, in general, errors of this type may cause inconsistencies, especially when multiple errors affecting both parties combine.

To overcome this potential problem, it is necessary for the system to log outbound messages to persistent storage (again in an atomic manner) *before* the message is transmitted. Logically, this can be seen as introducing new states in to the state machine of Figure 6. Upon recovery, an ambiguity would still exist, i.e., in the previous example the merchant would not be sure whether the original request had been sent, but the system now obeys the specification in that any duplicates would now be bit-wise identical to the original.

The cardholder processing is simpler than both the merchant and the APG in that a purchase can be completed using a single message. However, in the event of a failure of the cardholder component during the preparation of the purchase request, there is the possibility that multiple independent requests (PReq) could be sent for the same order. To overcome this, the cardholder should also log outbound messages before they are transmitted. To determine the state of a transaction after a system crash occurring after PReq has been received by the merchant only requires that the transaction identifier is available.

Summary and Conclusions

The SET payment protocol has been designed to place minimum reliance on the potentially volatile cardholder component. In particular, recovery of an in-progress transaction after a system crash does not rely on the participation of the cardholder; this differs from traditional transaction protocols where recovery is a typically a multi-party process[10]. This design is appropriate as there could be a potential financial advantage to be gained from *cheating* such a recovery protocol. Although the protocol does not provide end-to-end guarantees for the cardholder, it is able to interrogate the merchant to determine the status of a transaction after a failure.

Resilience to communication failure is achieved through the use of a message retransmission mechanism, messages are made idempotent through a duplicate detection mechanism based on request/response identifiers. Resilience to system failure necessitates that updates to transaction database records must be performed atomically to avoid the possibility of database corruption. Furthermore, to ensure that duplicate messages arising from system failures are bit-wise identical to the originals as specified, then outbound messages must be logged before they are transmitted.

[10] In traditional transactional systems, part of the commit protocol (e.g., 2-phase commit protocol) specifies the procedures to be carried out in the event of a host crash and subsequent recovery. This crash recovery protocol ensures that any transactions active at the time of the crash will be resolved upon recovery of the failed party and that any data manipulated within the scope of such actions will be restored to a consistent state. These procedures rely on all parties to be mutually trusted to participate honestly in the recovery protocol. Furthermore, to support recovery, each party must maintain persistent data recording the work carried out within the scope of the transaction and the coordinator must record the outcome of the transaction. Another issue is that the commit protocol is blocking, that is, a failure of a participating node could, under certain conditions, block the progress of the transaction.

Although, it has been shown that implementations of the merchant and APG can be made resilient to system failures, it is likely that providing high system availability will be an important consideration. System downtime may result in lost opportunities, with the consequence of loss of revenue. The required atomic nature of the system state updates would make the HA mechanisms described earlier appropriate for SET merchants and APGs.

Appendix B: HP ORB Plus Location Algorithm

What follows are extracts from the on-line documentation that accompanies the public release of HP ORB Plus 2.01. These extracts provide the background information necessary to understand the mechanisms used by the ORB Plus to track persistent objects.

ORB Plus Object Reference

An object reference holds the following information:

- The network address of where your server was running when the object reference was generated (that is, when you called _this to gets its reference or when you created the object reference with HPSOA::VirtualServer::make_object). We'll call this the "server birth address". This address becomes obsolete once your server shuts down and is restarted.

- The address of the server's location domain's Object Locator. We can call this the "locator address". It is guaranteed by the IIOP and DCE-CIOP transports to always be up-to-date.

- Your server's virtual server name (which you specified when you called HPSOA::add_vs).

How does a client send a message to a server?

1. The client tries to send the message to the server birth address. If your server is still running at that address (for example, if it has not been shut down and restarted since the object reference was generated), then the message will be delivered to your server at that address and the Object Locator will not be involved.

2. If the birth address failed, the client sends a message to your server's Object Locator (using "locator address") asking it "Where is the server running right now?" It hands in your server's virtual server name. If that Object Locator knows where your server is currently running, it returns the address. If the server is not running, and the Object Locator knows how to start up your server, it starts up your server and returns the new address. Otherwise, it cannot return a current address for your server.

3. If the client got back a current address for your server, then it sends the message to that address and your server should receive it. If the server is not running but is registered with the Object Locator, the client receives a TRANSIENT exception. If the server is not running and not registered with the Object Locator, the client receives an OBJECT_NOT_EXIST exception. All persistent servers must be registered with the Object Locator.

How does the Object Locator know where a server is currently running?

Whenever your server starts or restarts, you must call HPSOA::add_vs, handing in your server's virtual server name. When you call HPSOA::run, your server will send a message to the Object Locator to tell it where your server is currently running. It hands in the virtual server name and the current address of your server.

The Object Locator records this information. So, when a client asks the Object Locator where the server is running right now, it either returns the server's current address or indicates that the server is not running and could not be started.

The Object Locator puts this information its database so that if you shut down and restart the Object Locator without shutting down and restarting your server, the Object Locator will still know your server's current address. (You can clear this information by using the -ca option when you restart the Object Locator.)

References

[Albitz92] P. Albitz and C. Liu, “DNS and BIND in a Nutshell,” O’Reilly and Associates, 1992.

[CDnow] CDnow Web Site.
<URL:http://www.cdnow.com/>

[Cisco96a] “Cisco LocalDirector,” Cisco Systems, Inc. White paper, 1996.

[Cisco96b] “Cisco DistributedDirector,” Cisco Systems, Inc. White paper, 1996.

[Cisco96c] “Scaling the World Wide Web,” Cisco Systems, Inc. White paper, 1996.

[Cohen95] L. S. Cohen and J. H. Williams, “Technical Description of the DECsafe Available Server Environment,” Digital Technical Journal, 7(4), pp. 89-100, 1995.

[Comer95] D. E. Comer, “Internetworking with TCP/IP, Volume 1: Principles, Protocols and Architecture,” 3rd ed., Prentice-Hall, 1995.

[Egevang94] K. Egevang and P. Francis, “The IP Network Address Translator (NAT),” RFC 1631, Network Information Center, SRI International, May 1994.
<URL:ftp://ds.internic.net/rfc/rfc1631.txt>

[Fleming95] R. A. Fleming, “The next Somersault architecture,” Internal document, Hewlett-Packard Laboratories, Bristol, 8th September 1995.

[Garfinkel96] S. L. Garfinkel, “The Wizard of Netscape,” WebServer Magazine, 1(2), pp. 58-63, July 1996.

[Halsall92] F. Halsall, “Data communications, computer networks, and open systems, ” 3rd ed., Addison-Wesley, 1992.

[Harry95] P. Harry, “FT Orb programmers guide,” Internal document, Hewlett-Packard Laboratories, Bristol, 19th October 1995.

[Harry96] P. Harry, “An Introduction to Somersault (Somersault 95-2),” Internal document, Hewlett-Packard Laboratories, Bristol, 7th March 1996.

[HP96] “HP ORB Plus 2.01 Frequently Asked Questions (FAQ),” Hewlett-Packard Company, January 1996.

[Katz94] E. D. Katz, M. Butler and R. McGrath, “A Scalable HTTP Server: The NCSA Prototype,” Computer Networks and ISDN Systems, 27(2), pp. 155-164, Special Issue Selected Papers of the First World-Wide Web Conference, November 1994.

[Little96] M. C. Little, S. K. Shrivastava, S. J. Caughey and D. B. Ingham, “Constructing Reliable Web Applications Using Atomic Actions,” submitted to 6th International World-Wide Web Conference, December 1996.

[Mastercard95] Mastercard and Visa, “Secure Electronic Transaction (SET) Specification, Book 2: Programmer’s Guide,” June 1996.
<URL:http://www.visa.com/cgi-bin/vee/sf/set/setprog.html>

[OMG95] “CORBAservices: Common Object Services Specification,” Object Management Group, OMG Document Number 95-3-31, March 95.

[Parrington95] G. D. Parrington et al., “The Design and Implementation of Arjuna,” USENIX Computing Systems Journal, 8 (3), pp. 253-306, Summer 95.
<URL:http://arjuna.ncl.ac.uk/papers/designimplearjuna.ps>

[Plummer82] D. Plummer, “Ethernet Address Resolution Protocol: or Converting Network Protocol Addresses to 48-bit Ethernet Address for Transmission on Ethernet Hardware,” RFC 0826, Network Information Center, SRI International, 1st Nov. 1982.
<URL:ftp://ds.internic.net/rfc/rfc826.txt>

[Sauers96a] B. Sauers, “Understanding High Availability,” Hewlett-Packard Company, 1996.

[Sauers96b] B. Sauers, J. Foxcroft, and P. W. Dickerman, “Designing Highly Available Cluster Applications,” Hewlett-Packard Company, 1996.

[Spasojevic96] M. Spasojevic and M. Satyanarayanan, “An Empirical Study of a Wide-Area Distributed File System,” ACM Transactions on Computer Systems, 14(2), pp. 200-222, May 1996.

[Transarc96] “Transarc DE-Light Web Client Technical Description,” Transarc Corporation, February 1996.
<URL:http://www.transarc.com/afs/transarc.com/public/www/Public/ProdServ/WWW/delov.html>

[Veritas96] Veritas Software Corp., “Veritas FirstWatch 2.2 Technical White Paper.”
<URL:http://www.veritas.com/HA/fw22_wp.html>

[Visa96] Access Device Messaging Specification (ADMS), Visa International, 1996.
<URL:http://www.visa.com/cgi-bin/vee/sf/adms.html>