

A Replicated Resource Architecture for High Performance Network Service*

Colin Allison, Martin Bramley, Jose Serrano

Division of Computer Science,
University of St Andrews,
Scotland, KY16 9SS

Abstract

Distributed Learning Environments represent the hope that communications and information technology can improve and widen access to education while maintaining and improving its quality. Such environments consist of network applications and services. Good interactive response time is crucial to their success. Slow responses can quickly dissuade teachers and learners alike from investing their time in the use of these services. Responsiveness timings taken across 155Mb/s IP/ATM networks have exposed traditional monolithic server performance as the main bottleneck in interactive response time. A strategy of providing bigger and faster monolithic server hardware in response to each occurrence of system slow down is not a good solution as it is expensive and inflexible. Cluster computing has proven a successful and cost effective alternative to conventional supercomputing and it would now seem to be appropriate to investigate its application to the problem of high performance network service provision. In order to research this issue a replicated resource architecture has been designed to harness the combined power of multiple independent computers. The architecture is outlined and an initial implementation of its core component, a coherence server, is described. Results are presented which indicate that this approach is viable within the context of Distributed Learning Environments.

1. Introduction

The deployment of high speed wide area networks such as 155Mb/s IP/ATM has raised expectations of network services and created new challenges for distributed computing. We are involved in building a specific type of service, a Distributed Learning Environment (DLE), which exemplifies many of the challenges now facing providers operating in modern high speed networks. A DLE service is typically hosted on a single server node and front-ended by a web server thereby enabling the exploitation of the ubiquitous web browser as client software. (See Figure 1).

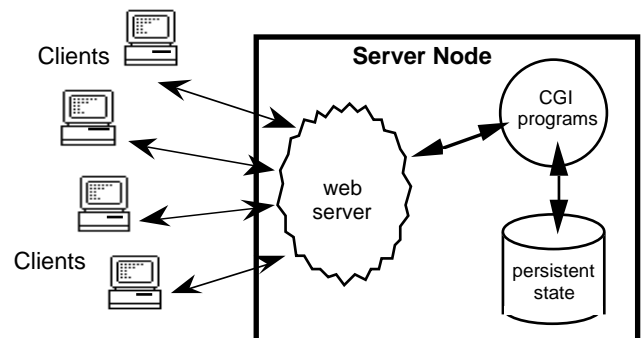


Figure 1:
A Single Server Distributed Learning Environment

A high performance service provided by a DLE must meet the primary objectives of *responsiveness* and *scalability*.

- *responsiveness* – good interactive response time is crucial to the success of a DLE. Users expect response times

* This work is supported by the SHEFC UMI, Phase II.

which are commensurate with personal computer applications and may see a DLE as an unproductive use of time if it is perceived to be slow. A cost model for decomposing the time taken to fulfill interactions between client(s) and server(s) has been developed and measurements based on the model have been made across the Scottish IP/ATM networks [1]. These clearly identify server performance as the main bottleneck in interactive response. The cost model has shown that within a server node the overhead formed by the web server component forms only a small part of the overall delay. The significance of these results is that neither the performance of the network nor the front-end web server are significant problems when compared with the execution speed of the program suite which constitutes the core service.

• *scalability* – DLEs may be used intensively for short periods by scheduled classes of students and need the ability to maintain responsiveness as the load on the service increases. The cost model has been applied to concurrent loads and shows a logarithmic increase in response times proportionate to the number of users. As previously stated this delay depends mainly on overall server performance. Initially provisioning a monolithic system of sufficient capability to meet peak demand is not a good solution as there is a very large difference between average and peak loads. Furthermore small changes in the scheduling or sizes of classes can quickly invalidate predicted peaks. Scalability can better be met by involving multiple nodes to provide the service and spreading the load.

In addition to these two primary performance objectives a DLE should also be robust and distribution transparent. In terms of robustness it should be reliable, available and fault tolerant. In terms of transparency the client should be unaware of the distributed nature of service provision. To meet these combined objectives a system that comprises of multiple independent nodes containing replicated copies of DLE resources would appear to be a promising approach. Independence and replication can be used to support load sharing and scalability, and also provide a good basis for work on fault tolerance. Cluster computing – the cost-effective exploitation of networked groups of commodity computers as a single computational resource – has proved successful as an alternative to conventional supercomputing [2,3], and it now seems appropriate to investigate the further exploitation of clusters for high performance network service provision. Wide area clusters [4] are of particular interest as their flexible distribution and dynamic constitution reflect the

distributed nature and wide load variance of a DLE user population. This paper outlines the design of a replicated resource architecture for wide area clusters and reports experiences with an initial implementation of its core component, the coherence server.

2. A Replicated Resource Architecture (RRA)

Figure 2 illustrates a conceptual outline of the RRA in the context of a four node DLE service. The layers are labeled 1 - 6. Layers 3, 4 and 5 are partitioned into four nodes. The other layers are strictly abstract.

1. Web browsers are used as client software.
2. Connections to a service pass through a load balancing mechanism. This masks the actual number of nodes in service, and may trigger attempts to recruit new nodes, or to shrink the pool of active servers, dependent on load. The purpose of this layer is support the abstraction of a single uniform service.
3. One conventional web server runs on each node.
4. One or more CGI programs, and program invocations run on each node.
5. Exactly one coherence server runs on each node. The coherence server is the framework for registering and implementing a per-resource type synchronisation requirement and coherence mechanism.
6. The persistent state of the DLE is identical when viewed from any node. In practice the state is physically replicated but should appear identical on every node *when there are no outstanding messages in the system*.

The issues raised by the core layer (6) are at the heart of the problems facing the successful construction of an RRA. If the time taken to stabilise an update, which may occur at any node, is too high then the illusion of a common state will not be maintained. On the other hand, if a synchronisation mechanism is too optimistic then it runs the risk of leaving the system in an inconsistent state. The adoption of a single uniform coherence mechanism is unlikely to prove efficient in meeting all requirements because, by dint of its universality, it must at the very least meet the strictest requirements, which are likely to create the greatest delays. Accordingly we propose to adopt a “policy-per-resource” approach, as described in the next section.

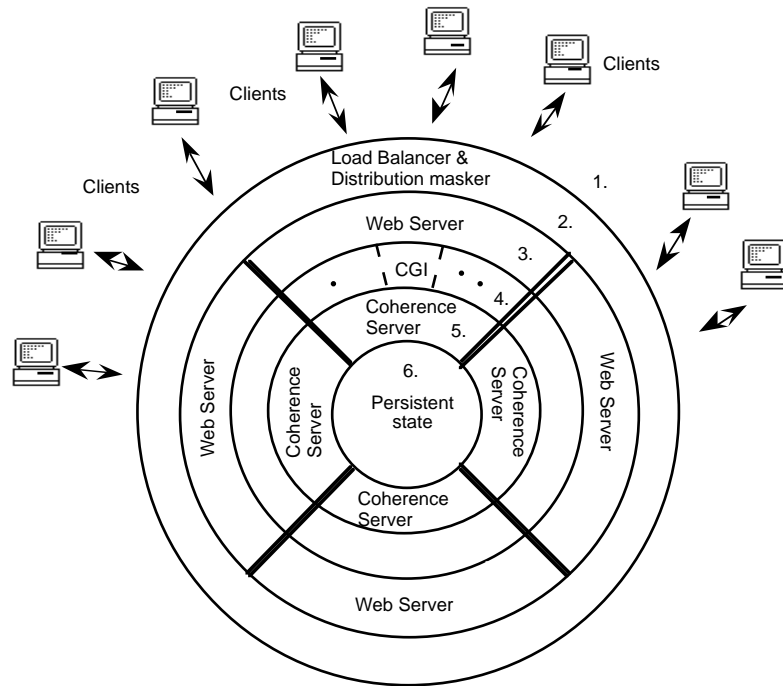


Figure 2: A Conceptual Outline of a Replicated Resource Architecture on Four Nodes

3. DLE Resource Synchronisation and Replication

DLEs consist of subject-specific components embedded in a general framework that supports *resources*, *events*, *users*, and *groups*. DLE resources are deliberately not defined rigorously as that would obstruct pedagogical imperatives. Example resource types include computational objects, interactive multimedia channels and simple web pages. Resources are often shared by groups of users and require synchronisation policies to handle concurrent access. The mechanisms which implement these policies should reflect and exploit high-level resource-specific features to minimise interactive response times.

Different types of shared resource require different types of synchronisation so it is important to support a diversity of such needs. What approach should be taken? In order to maximise performance we propose to treat the synchronisation requirements for each DLE resource individually. The scope for re-use and generality within a replicated resource architecture for DLEs will therefore be discovered experimentally. In order to support experimentation local synchronisation mechanisms have

been implemented in the context of a coherence server (CS). A CS can be more effective in implementing concurrency control than programs invoked directly by the web server because there is only ever one instance of the CS running, while there may be multiple invocations of other programs.

One of the main aims of the system architecture is to exploit cost-effective commodity based clusters so it must support the creation of shared resources with different synchronisation policies which will work on both single and replicated instances of these resources. This means the CS must be able to maintain global as well as local coherence. Figure 3 shows the role of the CS in providing global as well as local coherence. Three server nodes are collaborating to providing a uniform service for multiple clients. The exact means for achieving global coherence will depend on the type of resource. Extending a local synchronisation mechanism to the distributed scenario may not always be possible e.g. file-locking, or desirable e.g. because of lengthened response times. The coherence server therefore supports multiple mechanisms in order to support various policies in as efficient a manner as is possible.

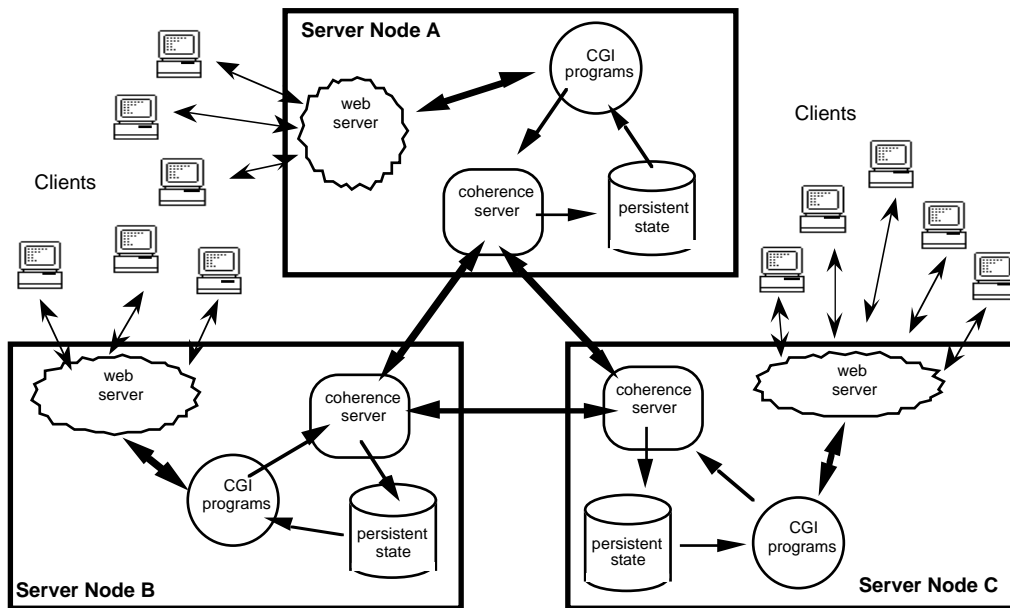


Figure 3: A Three Node Service

4. Case Study: A Replicated Notebook Resource

Finesse [5] is an example of a DLE where the focus is on financial management. Shared resources in Finesse include a lightweight groupware tool called a *notebook* which integrates bulletin board and mailing list functions, and an *investment portfolio* which is managed by a group of users which has access to a database of current share prices. In both cases a *many readers / many writers* semantics must be defined and supported even for a single resource instance on a single node. In the case of the notebook concurrent updates are either additions (new notes) or deletions. Concurrent additions should be applied (order is not important) but multiple deletions of the same entry must be detected and only one applied. There is no need to ever refuse a user request – two users concurrently deleting the same message should see that message deleted. By contrast, any concurrent updates to a portfolio may be in conflict and mediation or cancellation necessary. For example, if two members of a group attempt to sell the same shares at the same time then the system cannot allow both requests to be processed. This example also raises the shared context issue: the needs of co-operative working where *awareness* of others actions is important versus traditional database atomicity where users actions are

isolated and *hidden* from each other. In the case of a group working to manage a portfolio support for shared awareness is highly desirable in addition to the integrity enforced by the synchronisation mechanism.

The case study reported here used the notebook, which is the simplest type of resource within the system. The target response time for a notebook update to take place and return to the user is one second. Local synchronization (concurrency control) requirements are minimal – the notebook data is write-locked while being updated and released on completion. Concurrent additions are serialised on the write-lock. If there are concurrent attempts to delete a note the notebook application must ensure that only one delete is applied. This is achieved in a single server DLE by allocating each note a unique serial number and ignoring attempts to delete a non-existent note. Global synchronisation (replication) requirements for a notebook include the need to avoid duplicate deletions being applied, but also mandate that all copies of a notebook are identical when there are no outstanding update messages. Both these synchronisation requirements can be met by replacing the assignment of locally unique serial numbers with globally unique numbers. Message processing order and serial number maintenance are based on logical clocks.

4.1 Notebook Coherence Mechanism

An adaptation of Lamport's logical clock algorithm [6] is used to ensure that each note is assigned a globally unique serial number, and that serial numbers are totally ordered. Messages between nodes include a unique message identifier consisting of a server identifier and the value of the logical clock in the originating server. On receipt of a message a server checks the clock value of the message timestamp and compares it with its local clock. If its local clock is less than the received timestamp it is incremented to be greater than the received timestamp. This ensures that messages arrive after they have been sent in terms of global logical time and provides the basis for lost message detection and a total ordering of notes. The following steps are involved in a notebook update on multiple servers:

- The user submits the update (addition or deletion) to a web server
- The web server invokes the CGI notebook application.
- The application validates the request and passes it to the CS
- The application optimistically returns the new notebook image to the web server
- Meanwhile the CS multicasts an update request to the full group of servers (including itself)
- Each server processes the request and multicasts an acknowledgement.
- Nodes wait until they have seen all acknowledgments before committing the new notebook image to disk.

If there are concurrent updates at a single node or across the system they are serialised by the CS based on globally unique serial numbers. Note that this resource-specific solution is more efficient than e.g. an atomic transaction which would require a more complex and time consuming agreement protocol, and impede concurrent processing.

4.2 Results

Table 1 lists the costs of replicating the notebook on multiple servers. The components are as follows:

<i>Server</i>	Time taken by the point-of-contact coherence server to respond to the client.
<i>Request Processing</i>	Time taken by each server to carry out the request.
<i>Client</i>	The lifetime of the client.
<i>Coherence</i>	Time from arrival of request at the point-of-contact server until all servers have

processed the request and confirmed with each other.

Network Time from arrival of request at point-of-contact until all servers see the request.

servers	Server (ms)	request processing (ms)	Client (ms)	Coherence (ms)	Network (ms)
1	0.71	0.42	4.49	0.71	0.00
2	6.04	1.08	6.07	4.05	2.30
4	9.25	0.91	5.87	11.39	2.42
8	22.70	0.44	3.98	38.18	2.77

Table 1: Timing Results

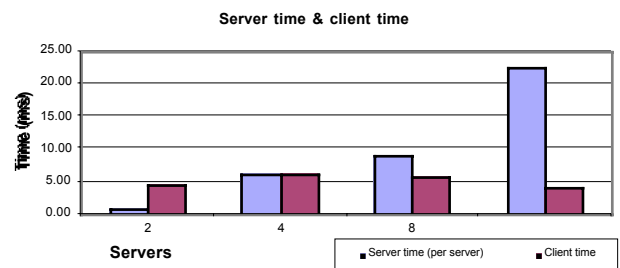


Figure 4: Server and Client Time

Figure 4 shows the relative amounts of time (ms) spent in the coherence server and the client CGI program. Not surprisingly, the CS time increases with server numbers. Note that with eight servers the target interactive response time of less than one second is still easily achievable.

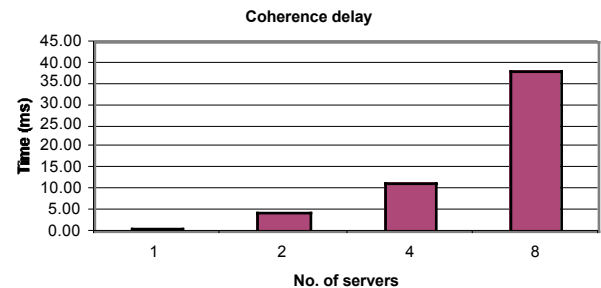


Figure 5: Coherence Delay

Figure 5 shows the increase in delay (ms) as server numbers increase. Again, the target response time is well within reach.

5. Related Work

Replication strategies and high performance network service are both active, although usually independent research topics in the field of distributed processing. Schneider [7] describes a fundamental conceptual approach to replication based on state machines. Distributed database systems [8] support global coherence based entirely on atomic transactions. Synchronisation toolkits such as Isis [9] aim to cater for all possible situations through group communication semantics and virtual synchrony. In all these cases fault tolerance and data integrity are the primary objectives and there is little or no concern for service performance in terms of real time responsiveness. A further drawback with adopting a uniform approach such as atomic transactions to all synchronization requirements is that it precludes the exploitation of higher level application semantics for performance, as demonstrated in the notebook case study.

[10] presents an high-level, software architecture for providing network services using a cluster of workstations. Issues of load balancing, fault tolerance and scalability are considered. At the same time, distillation of web page content is also used as a means of improving the performance of web servers. The type of service being provided is mainly read-only and consists of repeated multiple downloads in response to multiple simple requests. This is quite different from a DLE where there is a high degree of interactivity and the service is actively computing on behalf of the user.

[11] suggests the implementation of specialised operating systems for servers as a set of libraries on top of a custom-built operating system, the exokernel, providing applications with direct, protected access to hardware resources. This approach focuses on server efficiency and resource utilisation. It is not suitable for cluster computing however as the exokernel cannot readily be used as commodity software.

[12] proposes a model for the construction and management of highly available services. It is based on two protocols for replica synchronisation and group communication, and adopts the state machine approach of Schneider [7]. The key DLE issue of responsiveness is not considered.

The SWEB server [13] is a cluster-based server designed for the Alexandria Digital Library project and provides high performance HTTP service by using a cluster of workstations with embedded load balancing algorithms to maximise performance. The extra functionality of the

HTTP server is achieved by adding appropriate modules to the HTTP demon.

In summary there appears to be little other work specifically focusing on the use of clusters for high performance service provision. Nonetheless much of our work is grounded in the growing body of research into replication and synchronisation in distributed systems.

6. Conclusions and Future Work

Interactive response time is a key measure of performance in Distributed Learning Environments. The programme of work described in this paper has been motivated by measurements made across relatively fast wide area networks which have shown that server performance is the single most significant bottleneck in interactive response times. The decision to opt for a cluster-based solution reflects the nature of Distributed Learning Environments – high variance in loads and widely distributed user populations. Clusters are traditionally employed in the area of high-performance numerical computation, where a single user, single program may run for a few minutes or hours and then terminate. A Distributed Learning Environment makes different demands on a cluster in that the persistent state of shared resources must be maintained indefinitely in a dynamic multi-user situation where interactive response times are the measure of performance.

In order to exploit clusters for DLEs a replicated resource architecture has been sketched. The approach taken to RRA design and construction has been to support DLE resource types in their individual needs rather than attempt to create a system which supports all conceivable synchronisation requirements. In order to support experimentation and re-use a core mechanism, the coherence server, has been implemented and tested in the context of an 8-node server system. We have described the case study of a simple resource, the notebook. Results have shown the coherence server concept to be valid and beneficial. The next stage of this work will develop the coherence server to support replication of other, more complex, types of resource.

Future work includes implementations of mechanisms to dynamically manage the pool of available servers and provide fault tolerance. The former will adapt [14] and the latter will draw on theoretical work found in [15, 16] but will still maintain cost-effective high performance network service provision for DLEs as the primary objective.

7. References

- [1] Allison, C, Bramley, M, Serrano J, "The World Wide Wait: Where Does the Time Go?", to appear in Proc. Euromicro 98 Conference "Engineering Systems and Software for the Next Decade", IEEE Press, Vasteras, Sweden, August 1998.
- [2] D. Ridge, D. Becker, P. Merkey, T. Sterling, Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs Proceedings, IEEE Aerospace, 1997
- [3] CULLER D, et. al., Parallel Computing on the Berkeley Now, In JSPP'97 (9th Joint Symposium on Parallel Processing), Kobe, Japan.
- [4] Allison C. Virtual Networks and Quality of Service for Cluster Computing, in *Proc. 6th Euromicro Workshop on Parallel and Distributed Processing*, IEEE Press, Madrid, January 1998. pp.127-132.
- [5] Finesse: Finance Education in a Scalable Software Environment <http://tullamore.accountancy.dundee.ac.uk/> 1994.
- [6] LAMPORT L., Time, clocks, and the ordering of events in a distributed system, *Commun. ACM* 21, 7 (1978),558–565.
- [7] Fred B. Schneider "Replication Management using the State-Machine Approach", in Sape Mullender (Ed), "Distributed Systems", Addison-Wesley, 1993
- [8] Oszu & Valduriez, "Principles of Distributed Database Systems", Prentice-Hall, 1991.
- [9] Birman, K & Van Renesse R, "Reliable Distributed Computing with the Isis Toolkit", IEEE Press,
- [10] Fox, A. et al. "Cluster-Based Scalable Network Services", in Proceedings of the 16th ACM Symposium on Operating System Principles, France, 1997, pp78-91.
- [11] Kaashoek, M. F. et al. "Server Operating Systems", in Proceedings of the 7th European ACM SIGOPS Workshop, Ireland, 1996.
- [12] "Construction and management of highly available services in open distributed systems", *Distrib. Syst. Engng* 5 (1998), 29-45
- [13] Daniel Andresen et al. "Towards a Scalable Distributed WWW Server on Workstation Clusters", Proceedings of the 10th IPDS, Hawaii, April 1996.
- [14] Allison C, Harrington P, Huang F, & Livesey M.J, "Scalable Services for Resource Management in Distributed and Networked Environments" in SDNE'96, ed. P. Honeyman, IEEE Press, June 1996, pp98-105.
- [15] HADZILACOS V. & TOUEG S., Fault Tolerant Broadcasts and Related Problems in *Distributed Systems*, ed. Mullender,, Addison-Wesley (1993) 97-145.
- [16] BABAOGU O, DAVOLI R, GIACHINI, L & BAKER M.G, RELACS: A Communications Infrastructure for Constructing Reliable Applications in Large-Scale Distributed Systems, in *HICSS'28*, Vol II, IEEE CS Press (1995) 612-621.