

# Meeting Interactive Response Targets in Distributed Learning Environments

Colin Allison, Martin Bramley, Jose Serrano

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

*Extended abstract for "The Dynamic Web"*

## 1. Introduction

Distributed Learning Environments (DLEs) represent the hope that communications and information technology can improve and widen access to education while maintaining and improving its quality. Such multi-user environments consist of web-oriented networked applications and services. Our experiences so far have shown that 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. Students expect the types of real-time response they get from computer games, while staff expect the immediacy typical of single-user applications on a personal computer. In general, a key goal of a DLE is to deliver a response to the users' screen in less than one second. This abstract describes some of the work carried out to test, analyse and improve interactive response times in Finesse [1], a DLE for finance education.

## 2. Finesse

Finesse 1.0 is run on a single web server serving four universities connected by a 34Mb/s IP/ATM infrastructure across distances of up to 120 Km. Due to a large and diverse base of client machines it is not possible to make any assumptions other than that they have basic web-browsing capabilities. To this end the entire application has been written in interactive CGI scripts. Users may work individually or as part of group, asynchronously, or synchronously. The typical pattern of usage is that a user-initiated request will result in a CGI-generated response, which may contain some real-time information about the shared state of the system or a shared group context. The core focus for group collaboration is a virtual portfolio of stocks and shares which is managed by the group in a highly realistic trading environment. The single most common user request is for the current share price list. These prices reside in a database on the server which is updated daily from the London Stock Exchange. The CGI program that handles these requests generates an HTML table of up to seven columns and a thousand rows, and can reach about 180 Kbytes in size. We cite this as an example because it seems to stress both browser and server-side processing to the limits. Although the raw data size is quite modest this frequently requested page is the source of much delay and poor response. In some cases delays can be measured in minutes rather than seconds. Accordingly, because it is our worst case in terms of delay, we use it as the target for reducing response times.

### 3. A simple structured timing model

Poor response when using a web-based application often leads to speculation. Service providers and consumers may blame the network, the platform they are using, hard disk performance etc. Such guesswork is not useful – significant improvements can only come from informed decisions. Most work to-date on performance has focused on improving web server performance or web communication protocol efficiency [2,3,4,5,6,7,8] with little regard for the users perspective. Accordingly, because our response targets are based entirely on user perception, we have designed a simple cost model to guide analysis of the total delay and implemented a number of lightweight methods for measuring the constituent parts of that delay. The model covers the entire period from the user-initiated selection of a URL, to the resolution of all its associated links within the scope of an HTML page. We refer to this period as the closure over a URL, or *CURL*. A *CURL* may terminate correctly or incorrectly. When a *CURL* terminates correctly all images, files, applets and other components that are associated with a URL are located or generated, retrieved, displayed and/or activated. A *CURL* involving an HTTP URL is initially broken down into one or more instances of a top-level three-phase pattern:

- A) client-side HTML parse time and request generation
- B) server-side request processing: response generation may be CGI or static HTML.
- C) client-side rendering time consisting of parse time and display update

Network transit times, including source and destination protocol processing times, are represented by  $N_{CS}$  and  $N_{SC}$ , where  $N_{CS}$  denotes the time taken during the communication phase from client to server, and  $N_{SC}$  the delay in the other direction. Phase C may result in further requests and there is usually an overlap between request generation and other HTML parsing.

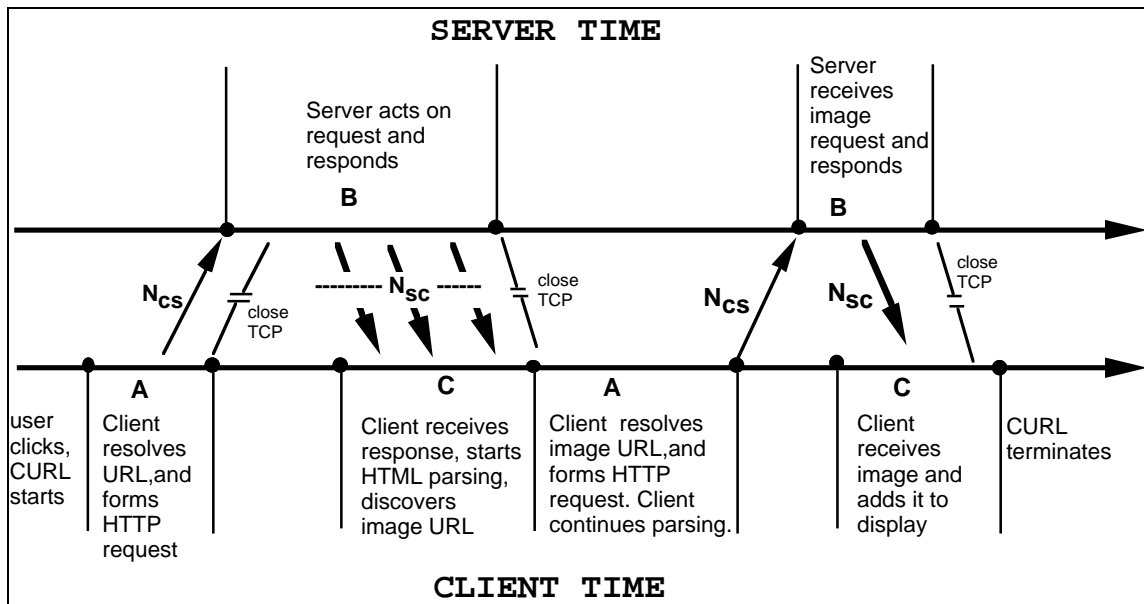


Figure 1: Phases of a typical web interaction

Figure 1 shows an example CURL consisting of a URL which results in the retrieval of an HTML page which in turn contains a single image reference. The top level phases can of course be broken down further and the model is designed to support stepwise refinement of the analysis once the main problem areas are identified. The model is described in more detail in [9] and is still evolving but has already proved useful in guiding performance tuning, as is described in the following section.

## 5. Case study

Methods for implementing the model are described in [9]. The tables below show a comparison made between the response times from our first measurements and the current state of the system. To recap, the object being requested, built and returned, is a table of current stock information that consists of seven columns and a thousand rows, and is about 180 Kbytes in size. The figures in brackets are the original timings, the non-bracketed figures represent the times after some performance tuning. Table 1a shows times in the case of a single client, Table 1b shows the case for four concurrent accesses. There are some surprising features shown by these measurements e.g. how little time was spent on the network, and how much client-side time can be taken by certain combinations of browsers, operating systems and platforms to render a large table. Methods used to tackle the problems are described in [9] and include server-side distillation, server-side function optimisation and client-side recommendations on choice of browsers.

	Mac	Sun	PII W95 Netscape 4.04 4.5	PII W95 Explorer 4.x	486 Netscape 4.04 4.5	486 W95 Explorer	<i>Server</i> (Sun)
total time (CURL)	(60) 42	(45) 15	(38) 7	(24) 3	(285) 79	(103) 63	(43) 16
submit time ( Phases A+B - overlap )	0	0	0	0	0	0	0
CGI runtime overlap - Phase C	(26) 7	(22) 4	(18) 2	(19) 1	(56) 22	(28) 2	(21) 1
minimum network & client parse time (Phase C)	(34) 34	(23) 11	(20) 5	(5) 2	(229) 57	(75) 61	(22) 15

Table 1a: CURL constituent timings in seconds for single client access

	Mac	Sun	PII W95 Netscape 4.04 4.5	486 W95 Netscape 4.04 4.5
total time (CURL)	(130) 50	(81) 16	(85) 8	(283) 70
submit time ( Phases A+B - overlap )	0	0	0	0
CGI runtime overlap - Phase C	(70) 7	(58) 4	(62) 2	(79) 23
minimum network & client parse time (Phase C)	(60) 43	(23) 12	(23) 6	(204) 47

Table 1b: CURL constituent timings in seconds for 4 concurrent accesses

## 6. The next stage: a replicated resource architecture

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. Table 1b shows the measured increases in response times proportionate to the number of four concurrent users. In practice classes of size twelve or greater use the system concurrently. These increases in delay depend largely 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. We believe that scalability can better be met by involving multiple nodes to provide the service and spreading the load.

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 comprising 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 [10, 11], and it now seems appropriate to investigate the further exploitation of clusters for high performance network service provision. Wide area clusters [12] are of particular interest as their flexible distribution and dynamic constitution reflect the distributed nature and wide load variance of a DLE user population. Figure 2 illustrates an initial design of a replicated resource architecture (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.

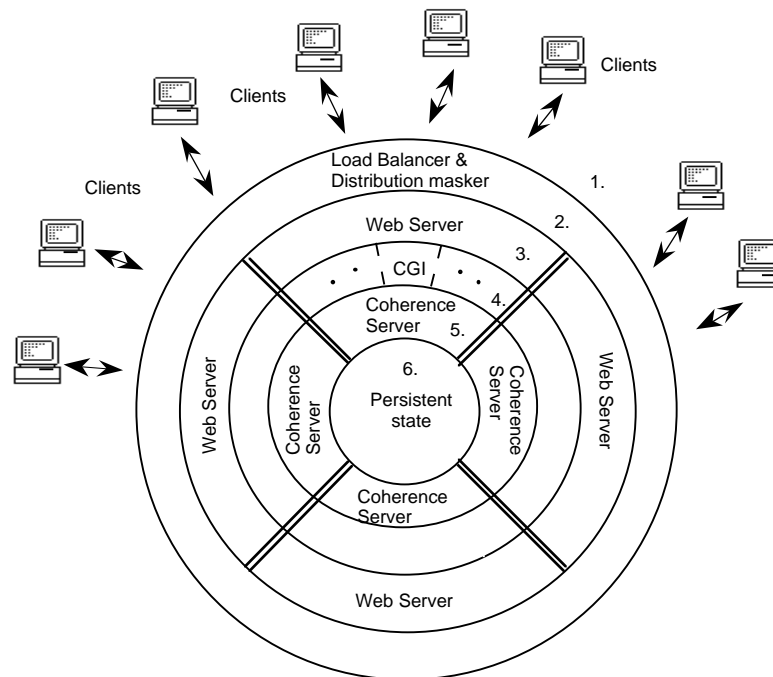


Figure 2: A Conceptual Outline of a Replicated Server Architecture on Four Nodes

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.
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 investigate a “policy-per-resource” approach.

## 7. Summary and future work

We have related some experiences of tuning a web-based Distributed Learning Environment to minimise response times. This has involved the design of a cost model which has been

implemented with lightweight methods. Results from the application of the model have been used to guide decisions on where effort and resource should be concentrated to gain improvements. At time of writing we have just observed our first one second response times for the full share prices table in a class of 12 students, thereby meeting the general target response for the worst case scenario.

We believe that further considerations regarding scalability and robustness favour the development of a Replicated Resource Architecture to provide a uniform set of services and we are now setting up multiple client / multiple server testbed to develop this idea.

## 8. References

- [1] Finesse: Finance Education in a Scalable Software Environment <http://tullamore.accountancy.dundee.ac.uk/> 1998.
- [2] N. Edwards, O. Rees. "Performance of HTTP and CGI", <http://www.ansa.co.uk/ANSA/ISF/1506/APM1506.html>
- [3] V. N. Padmanabhan, J. C. Mogul. "Improving HTTP Latency", Proceedings of the 2nd International WWW Conference. <http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html>
- [4] Spero, S. E. "Analysis of HTTP Performance Problems" <http://sunsite.unc.edu/mdma-release/http-prob.html>
- [5] Touch, J, J. Heidemann, K. Obraczka. "Analysis of HTTP Performance" <http://www.isi.edu/lam/publications/http-perf/>
- [6] Fielding et al. "Hypertext Transfer Protocol HTTP/1.1" RFC 2068.
- [7] Hu J.C., Semedh M., Schmidt D. C. "Techniques for Developing and Measuring High-Performance Web Servers over ATM Networks", submitted to InfoCom '98.
- [8] M. F. Arlitt, C. L. Williamson. "Internet Web Servers: Workload characterization and implications", IEEE/ACM Transactions on Networking, Vol. 5. No. 5, 631-644, October 1997
- [9] Allison, C Bramley, M, Serrano J, "The World Wide Wait: Where Does the Time Go?", in Proc. Euromicro 98 Conference "Engineering Systems and Software for the Next Decade", IEEE Press, Vasteras, Sweden, August 1998.
- [10] D. Ridge, D. Becker, P. Merkey, T. Sterling, "Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs" Proceedings, IEEE Aerospace, 1997
- [11] Culler, D. et. al., "Parallel Computing on the Berkeley Now", In JSPP'97 (9th Joint Symposium on Parallel Processing), Kobe, Japan.
- [12] 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