An Adaptive Approach for Downloading Replicated Web Resources

Vittorio Ghini

Dipartimento di Scienze dell'Informazione, Università di Bologna
Mura Anteo Zamboni, 7, 40127 Bologna (Italy)
ghini@cs.unibo.it

Abstract

The success of a Web service is largely dependent on its responsiveness (i.e., its availability and timeliness) in the delivery of the information its users (clients) require. A practical approach to the provision of responsive Web services is based on introducing redundancy in the service by replicating the service itself across a number of servers geographically distributed over the Internet. Provided that the replica servers be maintained mutually consistent, service responsiveness can be guaranteed by dynamically binding the client to the *most convenient* replica (e.g., the nearest, lightly loaded, available replica; the available replica with the least congested connection to the client). Based on this approach, we have developed a software mechanism [GHI01] that meets effectively the responsiveness requirement mentioned above. In essence, this mechanism, rather than binding a client to its most convenient replica server, engages all the available replicas in supplying a fragment of the Web document that client requires. The size of the fragment a replica is requested to supply is dynamically evaluated on the basis of the response time that replica can provide its client with. In addition, the proposed mechanism can dynamically adapt to changes in both the network and the replica servers status, thus tolerating possible replica or communication failures that may occur at run-time. Our mechanism can be implemented either as part of the browser software or as part of a Proxy server. In this paper, we describe the design, the development, and the performance evaluation of both these implementations of our mechanism. The performance results we have obtained from our evaluation exercise illustrate the adequacy of the mechanism we propose, in order to provide responsive Web services.

1. Introduction

The Web is becoming a fundamental technology for most of advanced companies and organizations, and many users rely on the Web for retrieving critical information and for conducting personal businesses. Users are not willing to tolerate too large latency times, and they do not care of complexity of Web infrastructure and technology. They only detect if the response time becomes too high, or if there are many periods of unavailability of the services. Thus, the responsiveness is a crucial issue in the design of those services. Responsiveness can be increased [CAR01] deploying one of the following four approach: *i*) *scaling-up* the single server, *ii*) using so-called *Differentiated Web Services* that provide preferential treatment of classes of users, *iii*) using *local replication,* i.e. locally distributing the Web services among a set of replica servers located into a cluster of workstations and *iv*) *geographical replicating* the servers across the Internet.

Scaling-up. Obviously, the first step to increase the performance of a web service consists of increasing the hardware potential of the servers that implement that service, by adding memory and CPU power to each single server. Unfortunately, the server cannot scale-up indefinitely.

Differentiated Web Services. Recent proposals suggest to provide differentiated scheduling services in order to enable preferential treatments of classes of users. Typically, access to a commercial web service occurs in the form of a session consisting of a sequence of individual requests. The session-based admission control policies proposed in [CHE99] will accept a new session (i.e. a new customer to the site) only when a server has the capacity to process all future requests related to the session, i.e. a server can guarantee the successful session completion. The HP's WebQoS system is based on this approach.

Local Replication. A host of techniques, e.g. [BUN99, CIS96, COL98, DAM97, IYE00, PAI98, WAT98], has been proposed in the literature that addresses specifically the issue of providing highly available Web services. In general, these techniques are based on i) constructing a Web service out of replicated servers, locally distributed in a cluster of workstations, namely a *Web cluster,* and ii) distributing the client request load among those servers, by using a Web router/gateway (with the public IP address of the web service) that acts as a centralized dispatcher. As discussed at length in [ING00], local replication techniques can only partially meet the high availability requirement (e.g., they may be vulnerable to failures of the router/gateway). In addition, they cannot be deployed in order to meet timeliness requirements, such as the client latency time over the network, because these requirements fall beyond the control of these techniques.

Geographical Replication. In order to overcome these limitations, an alternative approach has been proposed in [ING00], and explored further in [CON99, CON01]. According to this approach, a responsive Web service can be provided by replicating servers across the Internet, rather than in a cluster of workstations. In the Internet context, a successful deployment of this approach will depend on the ability of achieving the following two principal goals: i) dynamically binding the client to the *most convenient* replica server, and ii) maintaining data consistency among the replica servers. Unfortunately, neither of these two goals is easy to achieve. Firstly, the dynamic binding of clients to replica servers can turn out to be difficult to implement, owing to the location based naming scheme used in the Web. This scheme provides a one-to-one mapping (i.e., the Uniform Resource Locator - URL) between a name of a resource and a single physical copy of that resource; hence, dynamic binding of a client to distinct replica servers requires that the client-side software be adequately extended in order to be able to select an available replica, at run-time. Secondly, maintaining replica consistency on a large geographical scale can be hard to achieve, without affecting the overall service performance. In addition, the Internet environment is subject to (real or virtual) partitions that can prevent communications between functioning nodes; hence, within this environment, both clients and replica servers may hold mutually inconsistent views of which replica server is available and which is unavailable.

In view of these observations, we have developed a mechanism, that, in order to provide the clients of a geographically replicated Web service with service responsiveness, exploits the parallelism inherent in the replicated servers architecture that implements that service. Specifically, the principal goal of this mechanism is to minimize what we term the User Response Time (URT), i.e. the time elapsed between the generation of a browser request for the retrieval of a Web page, and the rendering of that page at the browser site (issues of replica consistency fall outside the scope of this mechanism). To this end, rather than binding a client to the *most convenient* replica server, as proposed in [ING00], our mechanism intercepts each client browser request for a Web page, and fragments that request into a number of sub-requests for separate parts of that document. Each sub-request is issued to a different available replica server, concurrently. The replies received from the replica servers are reassembled at the client end to reconstruct the requested page, and then delivered to the client browser. The effectiveness of C^2LD mechanism has been validated through both an experimental evaluation over the Internet, and simulation.

This paper is structured as follows. The next section summarizes the design of our C^2LD mechanism and introduce the mechanism. Section 3 discusses both the experimental and the simulative results we obtained from the validation exercise of the C^2LD mechanism we have carried out. Finally, Section 4 provides some concluding remarks.

2. Design Issues

Our mechanism is designed so as to adapt dynamically to state changes in both the network (e.g. route congestion, link failures), and the replica servers (e.g. replica overload, unavailability). To this end, our mechanism periodically monitors the available replica servers and selects, at run-time, those replicas to which the sub-requests can be sent, i.e. those replicas that can provide the requested page fragments within a time interval that allows our mechanism to minimize the URT. As our mechanism implements effectively load distribution of client requests among the available replicas, we have named it Client-Centered Load Distribution (C^2LD). C^2LD can be implemented as an extension of the client-side software (i.e., in the browser) or as a module of a Proxy server to which the clients send their requests.

The design of C^2LD is based on an analytical model that we have developed, and described in [GHI01]. This model is essential in order to determine the size of the page fragment that can be requested to each replica, and the extent of the monitoring period C^2LD is to use in order to execute a Web page request.

The timeliness requirement to be met by our C^2LD mechanism can be expressed by means of a User Specified Deadline (USD), i.e. a value that indicates the extent of time a user is willing to wait for a requested Web page to be rendered at his/her workstation. Owing to this USD time constraint, each replica server that receives a sub-request for a page fragment must honor that sub-request within a time interval that allow the C^2LD mechanism to reconstruct the requested page, out of all the received fragments, before the USD deadline expires. Thus, it is crucial that the C^2LD mechanism assess accurately both the size of the fragment each replica is to supply, and the time intervals within which these fragments are to be received at the client site. The above mentioned analytical model enables these assesments. Access to a Web service from a browser entails that a HTTP GET method be invoked by that browser. The C^2LD mechanism intercepts that HTTP GET invocation, starts the USD timeout and, using the URL in the GET invocation, interrogates the DNS. The DNS maintains the IP addresses of the N_{REP} replica servers that implement a Web service. When C^{2}LD submits a request to the DNS for resolving a URL, the DNS returns the IP addresses of all the replica servers associated to that URL. As the replica servers' addresses are available to the C^{2}LD mechanism, this mechanism interacts with each replica as illustrated by the skeleton code in Figure 1, and summarized below. C^{2}LD invokes a HTTP HEAD method on each replica *i*. The reply from replica *i* to the first HTTP HEAD invocation is used by C^{2}LD to: i) get the size of the requested page, ii) estimate the current data rate that replica *i* can provide ($DR_{i,r}$ in figure 1 below), and iii) assess the size ($PS_{i,r}$ in figure 1) of the first fragment that can be fetched from that replica. C^{2}LD maintains a global variable (*download_done*, in Fig.1) that indicates whether or not all the fragments of a requested page have been delivered. Until a page is not fully downloaded, C^{2}LD uses the Eq. in the rows 9 and 10 to compute the size of the fragment that is to be requested to the replica *i*.

```
1)   within USD do                     /* set USD timeout */
2)   …
3)   HEAD(…)                           /* send HEAD request to replica i */
4)   URT(i) := …                       /* assess URT replica i can provide */
5)   PS(i,1) := …                      /* compute 1st fragment size for replica i */
6)   within S do                       /* set timeout of length S */
7)       if not download_done then     /* check if page download completed */
8)           GET (…)               /* get fragment from replica i */
9)           $DR_{i,r} = \frac{PS_{i,r-1}}{URT_{i,r-1}}$;   /* compute expected data rate, based on URT of previous request */
10)          $PS_{i,r} = DR_{i,r} \cdot S_{i,r}$;   /* compute size of next fragment to be requested */
11)      else
12)          return;
13)      od
14)  …                                 /* handle S timeout exception */
15)  od
```

Figure 1: Implementation of the C^{2}LD Service

In order to adjust adaptively to possible fluctuations of the communication delays that may occur over the Internet, the fragment size is computed each time a fragment is to be requested, based on the value of the *URT* experienced in fetching the previous fragment. Thus, in essence, as the GET request *r-1* directed to a given replica *i* terminates, a new GET request *r* can be issued to the replica *i* with the fragment size value $PS_{i,r}$ computed (see row 10 in the figure 1) on the basis of the $URT_{i,r-1}$ response time. Once the requested fragment size has been calculated, C^{2}LD issues a HTTP GET request to the replica *i*, by specifying the HTTP RANGE option, in order to retrieve the required fragment of that size. Note that some of the replica servers may not respond timely to the HTTP (HEAD and GET) invocations described above (e.g., they may be unavailable owing to network congestion). Thus, C^{2}LD associates a timeout to each HTTP request it issues to each server *i*. If that timeout expires before C^{2}LD receives a reply from a replica server *i*, it assumes that the server *i* is currently unavailable, and places it in a *stand_by* list. Replica servers in that list are periodically probed to assess whether they have become active again. Requests for replicas in the *stand_by* list are redirected to active replicas.

We have developed the implementation of our mechanism using the Java programming language and the Java 2 Software Development Kit. Our mechanism can be incorporated either in the browser software or in a HTTP Proxy server. In the former case, the C^{2}LD implementation intercepts the HTTP GET method invocation issued by the browser, and acts as previously described. In contrast, in the latter case, a Proxy server maintains an instance of the C^{2}LD mechanism for each browser accessing that server. Thus, in essence, a Proxy server dispatches requests from its client browsers to their relative C^{2}LD instances, and delivers responses from those instances to their relative browsers.

3. Validation

The effectiveness of our C^{2}LD implementation has been assessed through the following two separate evaluation exercises. The first of these exercises consisted of experimenting our mechanism using the actual

Internet. In this evaluation, a single browser program, incorporating our mechanism and accessing a geographically replicated Web service, was involved.
The second evaluation exercise was based on simulation; specifically, as part of this exercise, we developed a simulation scenario in which multiple clients accessed the same replicated Web service. In addition, within this exercise, we have evaluated the two C^{2}LD implementations described earlier, i.e., the implementation of the C^{2}LD mechanism within the browser program, and the implementation of our mechanism within a Proxy server. In the following sub-sections we summarize our evaluation exercises, in isolation.

3.1. Single Client Experimentation

In order to carry out experimental evaluation, we have developed a simple Web service implemented by four replica servers located in Italy (Cesena and Trieste), the UK (Newcastle), and the USA (San Diego), and interconnected via the Internet. A large number of experiments (4000, approximately) were carried out during a two-month period. These experiments consisted essentially of a client program (i.e. a browser) downloading Web documents of different size from up to 4 geographically distributed replica servers. These documents were downloaded by the same client program using both the C^{2}LD mechanism, and the standard HTTP GET downloading mechanism, for comparison purposes. Figure 2 depicts the locations of both the client and the replica servers used in our experiments, and the routes between this client and those servers.

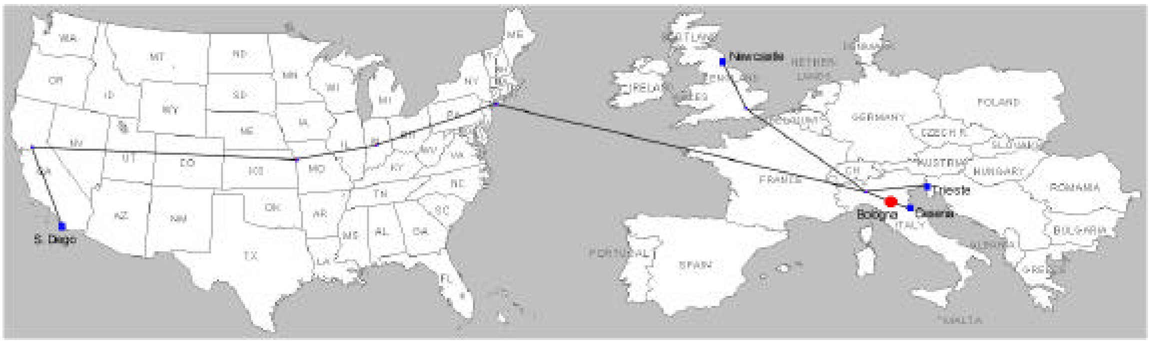

Figure 2: Routes between the client and the Web replica servers

KBytes	USD (secs)	C^{2}LD	S. Diego (HTTP)	Newcastle (HTTP)	Trieste (HTTP)	Cesena (HTTP)
50	120	0 %	0.3 %	0 %	0 %	1.65 %
100	120	0 %	0.25 %	0 %	0 %	5 %
200	180	0 %	0.2 %	0 %	0 %	4 %
500	300	0 %	0.3 %	0 %	0 %	1 %
1000	600	0 %	0.45 %	0 %	0 %	2.5 %

Table 2: Fault Percentage

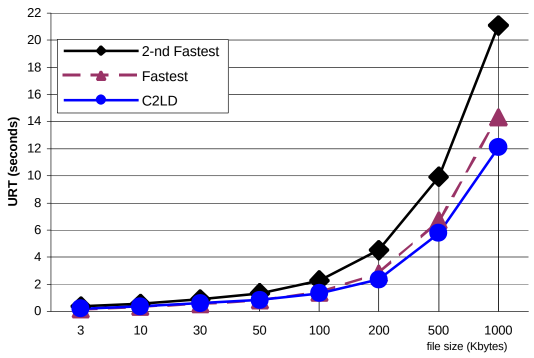

Figure 3: C^{2}LD vs. HTTP

As a first result, Table 2 summarizes the page loss percentage obtained with our mechanism, and that obtained with the standard HTTP GET downloading mechanism (performed with S. Diego, Newcastle, Trieste and Cesena, respectively). It can be seen from this Table that the C^{2}LD mechanism provides a highly available service, since it always guaranties the downloading of the requested document. Instead, the standard HTTP mechanism is not always able to provide its client with that document, as shown by the page loss percentage experienced by the S. Diego and Cesena servers, in particular. Figure 3 summarizes the results of the URT assessment we have obtained. The lowest curve in this Figure represents the URT provided by our C^{2}LD mechanism, as it results from averaging its value over all the performed experiments. The two higher curves, instead, represent the URT values provided by the two fastest Web replica servers, when interrogated with the standard HTTP downloading mechanism. As shown in Figure 3, the performance of the C^{2}LD mechanism and the HTTP mechanism are equivalent when the document size less than 50 Kbytes. Instead, when the document size is larger than 50 Kbytes, the C^{2}LD mechanism outperforms the standard HTTP downloading mechanism. As already mentioned, the C^{2}LD URT values in Figure 3 represent an average URT value, as it results from all the experiments we have carried out. For the sake of simplicity in this figure we do not report the value of the variance of the URT, as measured in our experiments. However, this value for the C^{2}LD mechanism varied from 0.099 s to 5 s, approximately, depending on the size of the requested page. The experimental results indicate that, in general, the C^{2}LD mechanism outperforms each single replica in all cases; the only exception occurs when the C^{2}LD uses only two replica servers, and, out of these two servers, one is very fast (Trieste or Newcastle), and the other is very slow (i.e., San Diego).

3.2. Multiple Client Simulation

The results discussed in the previous sub-section indicate that our C^{2}LD mechanism, when incorporated in a browser's software, can provide a notable speed-up in the communications between that browser and a replicated Web service, compared to the HTTP standard document fetching policy. However, as our experimental evaluation was based on a single client, these results were not sufficient to show the effectiveness of our C^{2}LD mechanism in the general case in which a replicated Web service is accessed by a large number of clients (say, hundreds), concurrently. Thus, as it was impractical to experiment our mechanism in one such real Internet-based scenario, we developed a simulation of that scenario within which we have evaluated the implementation of our mechanism both as part of the client software and as part of a HTTP Proxy server. In our simulation, both the workload that can be experienced by four replica servers, under different distribution of client requests, and the network load have been specified by means of simulation parameters. The simulation model has been specified by using the AEMPA technology [BER00].

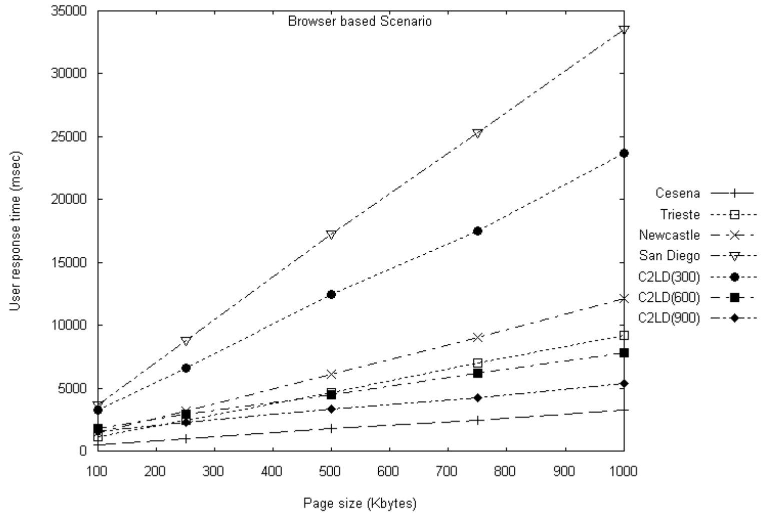

Figure 4. Web Access: URT measurements with multiple-client simulation

A summary of the simulation results we have obtained when the C^{2}LD mechanism is implemented at the browser are reported in Figure 4. We have assessed the responsiveness of our C^{2}LD fetching policy for the following Web page sizes: 100, 250, 500, 750, and 1000 Kbytes. We have used the following monitoring periods: 300, 600, and 900 ms. In addition to the results provided by the C^{2}LD policy this Figure shows the URT provided by each single server, accessed via the standard HTTP mechanism; the related curves are labeled with the name of the corresponding server.

As shown in Figure 4, Cesena achieves the best performance, with the standard HTTP mechanism, followed by Trieste, Newcastle and San Diego. Here, we can note a first discrepancy between these results and those obtained with the single-client experimentation. This discrepancy amounts to the fact that in the simulative scenario the Cesena server (equipped with the HTTP protocol) clearly outperforms the C^{2}LD policy. This is due to the fact that, after our single-client experimentation, the Internet connections provided to interconnect the central site of the University of Bologna with the remote university site of Cesena were upgraded, and the ping program used to set up the simulation parameters was executed after this upgrade. For the same reason, the transmission delays obtained with this method are in general notably lower than those measured in our experiments on the field. In essence, as already observed during the single client experimentation, this is the case when a specific server responds much faster than all the other replica servers. In one such particular situation, it may not be convenient to make use of any load distribution policy among replicated servers. Regardless of this particular situation, however, we can observe that the achieved URT critically depends on the duration of the monitoring period. If the duration is 300 ms, we see a poor performance which is only better than that using the San Diego server alone, with the standard HTTP mechanism. The reason is that, if we examine the configuration parameters, on average the S. Diego server replies to a HTTP request after 400 ms; hence, at the end of each monitoring period of 300 ms, most of the page fragments will be downloaded from the Cesena, Trieste, and Newcastle servers. The URT improves notably when the monitoring period is 600 ms; in this case C^2LD outperforms the San Diego, Newcastle and Trieste equipped with the standard HTTP mechanism, for page sizes greater than 500 Kbytes, and it is even better with a 900 ms monitoring period; in this case, C^2LD outperforms Trieste for page sizes greater than 300 Kbytes.

4. Concluding Remarks

In this paper, we have discussed a mechanism we have developed to construct responsive Web services. We have shown, through both real experiments and simulation, that this mechanism can be extremely effective in order to minimize the URT, if implemented either as part of a browser software or as part of a Proxy server.
We have applied our C^2LD mechanism to the fetching of generic Web resources, such as files and documents; we wish to assess the adequacy of our mechanism when deployed for accessing digital video and audio resources. In addition, we are planning to extend the work described in this paper by addressing the following topics. Firstly, we wish to examine strategies that involve pre-fetching of Web resources from the replica servers. Secondly, we wish to extend our mechanism to deal with dynamic Web services, and the previously described strategies for implement replicated Web servers. Finally, we wish to investigate policies for maintaining data consistency among geographically distributed replica servers.\

References

[BER00] M. Bernardo, P. Ciancarini, L. Donatiello, "*AEMPA: A Process Algebraic Description Language for the Performance Analysis of Software Architectures*", Proc. of the 2nd Int. Workshop on Software and Performance (WOSP 2000), ACM Press, Ottawa (Canada), September 2000, 1-11.

[BUN99] R.B. Bunt, D.L. Eager, G.M. Oster, C.L. Williamson, "*Achieving Load Balance and Effective Caching in Clustered Web Servers*", Proc. 4th International Web Caching Workshop, San Diego, CA, March 1999

[CAR01] V. Cardellini, E. Casalicchio, M. Colajanni, "*A Performance Study of Distributed Architectures for the Quality of Web Services*", Proc. 34th Hawaii International Conference on System Sciences (HICSS34), Maui, HI, January 2001.

[CIS96] "*Cisco Local Director*", CISCO System Inc., White paper, 1996

[CHE99] L. Cherkasova, P. Phaal, "*Session Based Admission Control: a Mechanism for Improving Performance of Commercial Web Sites*", Proc. of Seventh International Workshop on Quality of Service, IEEE/IFIP event, London, May 1999.

[COL98] M. Colajanni, P. S. Yu, D. M. Dias, "*Analysis of Task Assignment Policies in Scalable Distributed Web-Server Systems*", IEEE Trans. on Parallel and Distributed Systems, Vol. 9, N 6, June 1998, 585-600

[CON99] M. Conti, E. Gregori, F. Panzieri, "*Load Distribution among Replicated Web Servers: A QoS-based Approach*", Proc. 2nd ACM Workshop on Internet Server Performance (WISP'99), Atlanta, GA, May 1999

[CON01] M. Conti, E. Gregori, F. Panzieri, "*QoS-based Architectures for Geographically Replicated Web Servers*", Cluster Computing 4, 2001, pp. 105-116, Kluwer Academic Publishers.

[DAM97] P. Damani, P.E. Chung, Y.Huang, C.Kintala, Y. M. Wang, "*ONE-IP: Techniques for Hosting a Service on a Cluster of Machines*", Comp. Net. and ISDN Sys., 29, 1997, 1019-1027

[GHI01] V. Ghini, F. Panzieri, M. Roccetti, "*Client-Centered Load Distribution: A Mechanism for Constructing Responsive Web Services*", Proc. 34th Hawaii International Conference on System Sciences (HICSS34), Maui, HI, January 2001.

[IEEE00] IEEE Network, "*Special Issues – Web performance*", Vol 14, No. 3, May/June 2000

[ING00] D. Ingham, S.K. Shrivastava, F. Panzieri, "*Constructing Dependable Web Services*", IEEE Internet Computing, Vol. 4, N. 1, January/February 2000, 25 - 33

[IYE00] A. Iyengar, J. Challenger, D. Dias, P. Dantzig, "*High-Performance Web Site Design Techniques*", IEEE Internet Computing, Vol. 4., N. 2, March/April 2000, 17-26

[PAI98] V. Pai, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum, "*Locality-aware Request Distribution in Cluster-based Network Servers*", Proc. ASPLOS- VIII, San Jose, CA, October 1998

[WAT98] J. Watts, S. Taylor, "*A Practical Approach to Dynamic Load Balancing*", IEEE Trans. on Parallel and Distributed Systems, Vol. 9, NO. 3, March 1998, 235-248