Load and Proximity Aware Request-Redirection for Dynamic Load Distribution in Peering CDNs

Mukaddim Pathan, Christian Vecchiola, and Rajkumar Buyya

Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Parkville, VIC 3010, Australia
{apathan,csve,raj}@csse.unimelb.edu.au

Abstract. Peering between Content Delivery Networks (CDNs) endeavors to ensure that each user is served by an optimal Web server in terms of network cost, even under heavy load conditions. Therefore, a dominant factor for the success of peering between CDNs is to perform load distribution to handle highly skewed loads. In this paper, our approach for dynamic load distribution adopts a request-redirection mechanism by taking traffic load and network proximity into account. The load distribution strategy reacts to overload conditions, at a time instance, in any primary CDN server(s) and instantly distributes loads to the target servers, minimizing network cost and observing practical constraints. In this context, we formulate the resulting redirection strategy and perform extensive simulations to demonstrate the novelty of our approach. We show that our approach is effective to handle high load skews, and thus achieve service "responsiveness". We also perform a sensitivity analysis to reveal that our redirection scheme outperforms other alternatives.

1 Introduction

Content Delivery Networks (CDNs) [6][14] emerged to provide fast and reliable Web access services by distributing content to *edge* servers located close to end-users. To operate effectively a CDN is required to either over-provision or to harness external resources on demand. Cooperation between CDNs can reduce costs with over-provisioning and provide users with high quality services. This collaboration, termed as peering between CDNs [15], can be short-term wherein CDNs operate to handle flash crowds, or long-term in which they explore the delivery of specialized services.

The success of peering depends on its ability (i) to perform dynamic load distribution under traffic surges by redirecting requests to optimally underloaded Web server(s), thus binding users to optimal replicas (*timeliness*) and (ii) to exhibit acceptable throughput under overload conditions (e.g. during *flash crowds*). Load distribution to react to overload conditions on multiple inter-CDN Web servers is crucial to achieve scalability. Specifically, it can be stated as:

Given current load information in an overload condition, the peering CDNs system needs to select an optimal server to which a given request is to be redirected. To obtain up-to-date load information, the presence of a method for load monitoring and load index dissemination is necessary.

Many load distribution algorithms [7][8][9][10][18] for Web server and P2P-based systems have previously been proposed and analyzed. However, none of them can be directly applied for dynamic load balancing among distributed inter-CDN servers. This is due to challenges that include virtualization of multiple providers and offloading requests from the primary CDN to peers based on cost, performance and load. In such a cooperative multi-provider environment, requests are directed to sets of servers deployed across CDNs as opposed to individual servers belonging to a single entity. Therefore, request-redirections must occur over distributed sets of servers spanning multiple CDNs, without having complete state information.

In an effective load distribution strategy, a load index of resources needs to be estimated with low computation and communication overhead. In addition, network proximity information (distance between users and servers) is also important for load distribution decision so that requests are directed towards nearby servers. Therefore, an ideal load distribution strategy should realize a redirection scheme that takes traffic load and proximity into account. In this paper, we present strategies that implement dynamic *load distribution* through load and proximity-aware *request-redirection* among distributed Web servers of peering CDNs. In our approach, load indices are obtained through an *asynchronous feedback mechanism* and network proximity is measured using a *pinger logic* with low messaging overhead. A simulation model is developed, capturing key system components, to evaluate the performance of our approach. Experiment results show that our approach exhibits an acceptable level of throughput even under heavy load and the proposed redirection scheme outperforms other alternative schemes. The main contributions of this paper are twofold:

- A dynamic load distribution algorithm realizing request-redirection that reacts to overloaded server conditions in peering CDNs by steering user requests to optimally underloaded servers in terms of traffic load and network proximity.
- Evaluation of the proposed load distribution strategy and demonstration of its benefits by comparing with other alternatives, as well as a sensitivity analysis of the proposed redirection scheme using critical system parameters.

The rest of the paper is structured as follows. In Section 2, an overview of the related work is presented. A brief description of the peering CDNs is provided in Section 3. It is followed by the proposed load distribution algorithm with request-redirection formulation. Simulation methodology is described in Section 5. Results are discussed in Section 6. Finally, the paper is concluded in Section 7.

2 Related Work

IETF proposes the Content Distribution Internetworking (CDI) model [13] assuming a federation of CDNs. However, it does not specify any effective request-redirection mechanism through which load distribution could be performed. While it recommends using a supervision function or an independent third party to supervise and manage all the CDN peers, it does not define or characterize this supervision. Moreover, it does not examine the implications of using an independent third party for load distribution. Barbir et al. [4] present request-routing mechanisms for content networks. However, they do not specify any particular mechanism to redirect requests for dynamic load distribution in peering CDNs domain.

CDN Brokering [5] develops a brokerage system deployed on the Internet on a provisional basis. It presents IDNS, a specified request-routing DNS server, with a proprietary routing mechanism. It is not aimed at an optimal load distribution strategy. It only demonstrates the usefulness of brokering rather than to evaluate the redirection performance for load balancing. Alzoubi et al. [2] present a load-aware IP Anycast CDN architecture, incorporated with a route-controller, which takes server and network load into account to realize anycasting. This work establishes the applicability of anycasting as a redirection technique in CDNs. Although it is appealing, the use of a centralized route controller could be subject to single point of failure. Moreover, it follows a post-processing approach to offload overloaded servers, which may not be applicable for dynamic load distribution in peering CDNs. Shnayder et al. [17] present a trivial request distribution system for PlanetLab, which considers server loads and known proximity information from pre-defined landmarks to route requests. It suffers from a centralized approach for load index dissemination and arbitrary server selections. Moreover, it is non-adaptive to dynamic changes and is not targeted to load distribution under heavy traffic surges in practical context.

Amongst the work on load distribution strategies across geographically distributed Web servers, efforts by Conti et al. [10] and Cardellini et al. [8] could be adverted. The first presents a QoS-based architecture for load distribution among replicated Web servers. However, it does not capture significant parameters such as traffic load or network proximity for user response time estimation. The latter investigates the impact of redirection algorithms for load sharing. It proposes a Web cluster architecture where a DNS dispatcher is integrated with a redirection mechanism based on the HTTP protocol. While their approach might be effective to handle highly skewed load, the focus is particularly on a single domain of clustered Web servers.

In the context of modeling traffic redirection between geographically distributed servers, work of Amini et al. [3] and Ranjan et al. [16] can be mentioned. The first presents a model for intelligent server selections over multiple, separately administrated server pools. It does not show the effectiveness of any particular redirection or load distribution policy. The latter presents WARD—an architecture for redirecting dynamic content requests from an overloaded Internet Data Center (IDC) to a remote replica. It is targeted to IDCs under the control of a single administrative entity. Therefore, it does not virtualize multiple providers for request-redirection. Moreover, it does not provide any mechanism for load distribution between IDCs.

3 Peering CDNs

In the peering CDNs architecture [15], a provider serves requests as long as it can handle the load internally. If load exceeds its capacity, with the aid of the load distribution policy deployed in the Request-Routing System (RRS), the excess requests are offloaded to other optimally underloaded Web server(s). The initiator of each peering negotiation is called a *primary* CDN; while other CDNs who agree to provide their resources are called *peering* CDNs or *peers*. These roles are fluid and at any time a given CDN may be either a primary or a peer. The primary CDN directly manages the resources it has acquired, insofar that it determines what content is served and what proportion of the incoming traffic is redirected.

Figure 1 presents an overview of the load distribution architecture for peering CDNs. User requests for content are made to the RRS of the primary. These requests are then forwarded either directly to its server(s), or to a peer. The RRS is composed of *principle* members—*Peering Agent (PA)* to perform external resource discovery; *Mediator* to perform policy-driven authoritative operations; *Policy Repository (PR)* to virtualize all policies within a peering arrangement, and an *adjunct* member—*Service Repository (SR)* to encapsulate the status of CDN servers. It is also equipped with a request-redirection algorithm that assists in dynamic load distribution within a peering arrangement of CDNs. The PA, Mediator, SR and PR collectively act as a "conduit" for a given primary CDN, and they assist in external resource discovery.

From figure 1, we observe that some user requests of CDN 1 are served by its local servers or the origin server (on cache miss), whereas others are being served by the external Web servers of a peer, CDN 2. It is important to note that depending on the load any CDN can act as a primary CDN in a peering relationship. For instance, CDN 1 acts as a primary when its users are served by the peers' Web servers. Again, in the same peering relationship, CDN 2 plays the role of a primary CDN when its users are served by external Web servers from CDN N.

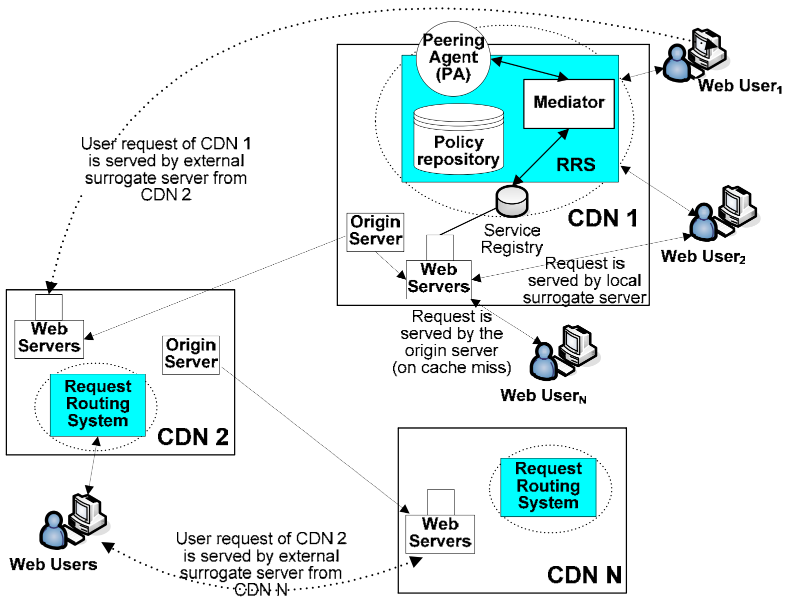

Peering Agent (PA)
Mediator
Policy repository
RRS
Service Registry
CDN 1
Origin Server
Web Servers
Request is served by local surrogate server
Web User$_1$
Web User$_2$
Web User$_N$
Request is served by the origin server (on cache miss)
User request of CDN 1 is served by external surrogate server from CDN 2
Web Servers
Origin Server
Request Routing System
CDN 2
Web Users
User request of CDN 2 is served by external surrogate server from CDN N
Web Servers
Request Routing System
CDN N

Fig. 1. Architecture of the load distribution system in peering CDNs

3.1 Load Distribution in Peering CDNs

Two alternatives could be examined to devise an effective load distribution strategy for peering CDNs. In the first approach, peering could be limited to a part of each CDN. A primary could decide to use only a subset of the surrogates of peers. Then the peers have to define a "virtual CDN" or "subCDN" for use by the primary. They also have to provide a real-time load status of each surrogate in this global subCDN using standard metrics. The primary CDN (or an authoritative entity belonging to it) can compare these load indices with its metrics and choose whether requests have to be redirected to a peer. Practical constraints could be put in place to ensure that redirection minimizes cost, in terms of load and network proximity, and does not overload peers' servers. Alternatively, an independent third party could supervise and manage all the CDN peers for load distribution. However, security, trust, and proprietary issues make it unlikely that a CDN would agree to have an external party to make allocations of its resources according to some load distribution policy.

4 Load and Proximity-Aware Dynamic Load Distribution

Our approach seeks to prevent wide oscillation in the load distribution decisions by selecting optimal server(s) through cost minimization, taking traffic load and network proximity into account. Since request-redirection is critical to our strategy, we first formulate the redirection problem and then present the load distribution algorithm.

4.1 Request-Redirection Formulation

Our system consists of M Web servers from participating CDNs, with N users spread across the system. We define a metric, redirection cost R_C, for serving requests through redirection in overload conditions. R_C depends on a server's traffic load and network proximity (in terms of round-trip response time). The cost of serving requests varies with different servers of the peers. More formally, R_C is defined as:

$$R_c(i, j) = \begin{cases} \infty & \text{if } p_{ij} = \infty \\ l_{ij} p_{ij} & \text{otherwise} \end{cases}$$

where l_{ij} is the incoming traffic load from user i on server j, and p_{ij} is the network proximity between them. It is known that a server's response load indicates the potential load assigned to it, since request and response loads on a CDN server are linearly correlated [2]. Therefore, the request load l_{ij} (traffic volume) can be calculated based on the server load. A Web server j's load is expressed as the product u_jS_j, where u_j is the server utilization in `[0, 1]` as reported by the load monitoring apparatus (mediator and SR) and S_j is its capacity, specified by the maximum number of serviced requests/second as reported in the CDN server's configuration specifications.

To consider the delay caused by inter-CDN redirection, p_{ij} is further defined as:

$$p_{ij} = \begin{cases} rt_{ij} & \text{if } j \text{ is an intra-CDN Web server} \\ rt_{ij} + I_D & \text{if } j \text{ is an inter-CDN Web server} \end{cases}$$

where rt_{ij} is the response time from user i to server j and I_D is the delay.

In order to minimize redirection cost during load distribution, we formulate the redirection problem as follows:

$$\text{minimize} \quad \sum_{i=1}^{N} \sum_{j=1}^{M} R_C(i, j) a_{ij}$$

$$\text{subject to } \sum_{i=1}^{N} a_{ij} = 1, \forall j$$
$$\sum l_{ij} a_{ij} \le S_j, \forall j$$
$$a_{ij} \in \{0,1\}, \forall i, j$$

where a_{ij} is an indicator variable to determine whether the Web server j is in the same CDN as user i; $a_{ij} = 1$ iff server j is an inter-CDN server, and $a_{ij} = 0$ otherwise.

Table 1. Significant properties of the request-redirection scheme

Properties	*Parameters*	*Description*	
Activation	Activation trigger (*when*)	Asynchronous (*on CDN server request*)	
	Activation decision (*where*)	Distributed (*Gateway redirection upon requests from distributed servers)*	
Implementation	Status information	Traffic load (*correlated with server response load = utilization * capacity*)	Alarm (*Asynchronous feedback*)
Redirection policy	Server selection (*how*)	Minimize redirection cost from available server list (*mapping of overloaded and underloaded server lists*)	
	Redirected entities (*what*)	User requests	

```
begin ...
while(true)
 serveRequests(CDN$_i$.WS); /* i∈{1,2,...,N} */
 if receive(request from primaryCDN.WS) = true then
  if WSList = NULL then
   n, WSList = {Available servers from peers};
  else
   ucount,CDN$_i$.uloadedWSList = {Underloaded WS list };
   ocount,CDN$_i$.oloadedWSList = {Overloaded WS list};
  for j = 1 to ucount do
   trafficLoad(CDN$_i$.uloadedWSList[j].loadmetric);
   // Use linear correlation assumption to calculate
   // request load based on server load
   netProximity(CDN$_i$.unloadedWSList[j]);
   R$_c$[i] = CDN$_i$.uloadedWSList[j].trafficLoad*
           CDN$_i$.uloadedWSList[j].netProximity);
   do
    opWS = selectWS(minimize R$_c$);
    targetWSList = add(primaryCDN.WSList, opWS);
    RequestRedirection();
   while primaryCDN.oloadedWSList[k].loadmetric >=
           alarm_threshold    /*k∈{1,2,...,ocount} */
   if |targetWSList| > 1 then
    if targetWSList.avgLoad <= alarm_threshold/2 then
     remove least loaded server in targetWSList;
...
end
```

Fig. 2. Load distribution algorithm (*LD_minCost*)

We can annotate our request-redirection scheme according to the classifications presented by Cardellini et al. [8]. Table 1 summarizes the properties of the redirection scheme used by the proposed dynamic load distribution algorithm. The first property specifies the mechanism in which redirection is activated and where the activation decision process is made. The next property focuses on the implementation, specifying the status information used for redirection. The last property delineates the redirection policy by stating the server selection strategy and the entities that are redirected. We describe these properties in more detail in later sections.

4.2 Load Distribution Algorithm

Figure 2 presents the pseudo-code for the proposed load distribution algorithm, named *LD_minCost*. It does not attempt to perform load distribution when the Web servers of a primary CDN are working under an acceptable load. At a given time, if an overload condition exceeding an alarm threshold is reached, as reported by a primary CDN server, servers' status (response load) is assessed and a list of lightly loaded servers is built from each participating CDNs (line 1 to 8). The traffic load and network proximity for each underloaded server are measured and the redirection cost, $R_C(i,j)$ is calculated in line 9 to 12. The optimal server from the underloaded server list is selected such that $R_C(i,j)$ is minimized upon redirection and the server does not get overloaded due to steered traffic. This optimal server is added to a set of usable server list (`targetWSList)` maintained by the primary CDN to satisfy its content requests. Thus, a mapping is maintained between the requests and a set of suitable servers to serve those requests. As long as there is an overloaded primary CDN server, a new server minimizing $R_c(i, j)$ (except the optimal server selected earlier) is added to this list (line 13 to 17). By using our load distribution strategy, we prevent a Web server from going into an overloaded state and multiple servers can serve a peak demand or a flash crowd situation. Moreover, if `targetWSList` contains several Web servers and their average load decreases significantly, one Web server is removed at a time from the list (line 18 to 20). It ensures that the degree of replication for serving requests does not remain unnecessarily high when requests relinquish over time. Moreover, it also guarantees that sufficient underloaded resources are always available in the peering CDNs system so as to utilize them during load distribution.

4.3 Benefits of Our Approach

A major advantage of our approach over traditional DNS-based redirection systems is that the actual user requests (eyeballs) are being redirected, opposing to the local DNS requests as in DNS-based redirection. Therefore, we can achieve a finer grain redirection. Since load distribution is performed dynamically, any redirection changes take effect instantly. In contrast, due to the IP-address caching in the intermediate name server, the DNS dispatcher loses direct control on subsequent requests for a Time-To-Live (TTL) period following address resolution, and thus causes some delay before redirection changes have an effect [9]. We also seek to achieve high locality with good load balancing, since requests targeted to the primary CDN stay within its domain as much as possible and are redirected to optimally underloaded peers only during highly skewed load conditions. In this way, our solution does not produce widely oscillated outcomes due to load distribution through request-redirection. Finally, our approach seeks to neutralize any load imbalance in the system, since participating CDNs have dynamic nature to act in primary or peering roles.

5 Methodology

While measurement studies in real testbeds could be advantageous in practice, they may not reproduce the problems and scenarios for which the solutions are designed. Hence, we have implemented a simulator, based on Independent Replication Method, using the CSIM/Java[1] simulation toolkit, to conduct *repeatable* and *controlled* experiments that would otherwise be difficult to perform in real CDN testbeds.

5.1 Simulation Model

Figure 3 shows the reference scenario for our simulations. In this scenario, there is an established peering arrangement, consisting of Web servers from four CDNs at different geographical locations across the Internet. Each CDN has its own user request stream and a set of Web servers, but delegates only a subset of them, i.e. virtual CDN or subCDN, to take part in the peering arrangement. To provide the accurate characterization of our scenario, we have simulated the main system entities: (i) Web servers, (ii) mediator, (iii) distributed SR, (iv) network congestions, and (v) end-users. In our simulation, PA and PR have limited functionality given that we would like to emphasize on the load distribution strategy, rather than focusing on external resource discovery through PA and policy enforcement through PR.

We use Transmission Control Protocol (TCP) for disseminating reliable and valid load index and User Datagram Protocol (UDP) for network proximity measurement. The reason for using UDP in the proximity measurement is because there are devices which will prioritize ICMP traffic (in case of TCP) over other traffic, which if there is congestion along the way, could skew things a bit. In addition, some service setups such as firewalls and routers limit ICMP traffic because of various denial of service threats. Moreover, UDP does not require acknowledgement of packets received, which causes less messaging overhead than TCP. Since our goal is to measure proximity under "realistic" traffic, using something closer is deemed more significant.

5.1.1 CDN Web Servers

We implemented the CDN servers as a set of facilities[2] that provide services to user requests. The response load of a Web server (`LoadMetric`) is expressed as a product of its utilization in `[0, 1]` at a given time during simulation and the maximum number of served requests/second (i.e. capacity). We use an asynchronous feedback mechanism, which assists the Web servers to trivially measure their actual loads and periodically update them in the SR. If a server's load exceeds a given alarm threshold, it signals the mediator to perform load distribution. A normal signal is sent when the load returns below the threshold. The use of such an asynchronous feedback mechanism suffice to consider a server as a candidate for receiving requests only if that server has not declared itself critically loaded.

[1] It creates process-oriented discrete-event simulation models. For more information, please check: `http://www.mesquite.com/`.

[2] Each facility is a simulated resource with a single server and a queue for waiting requests.

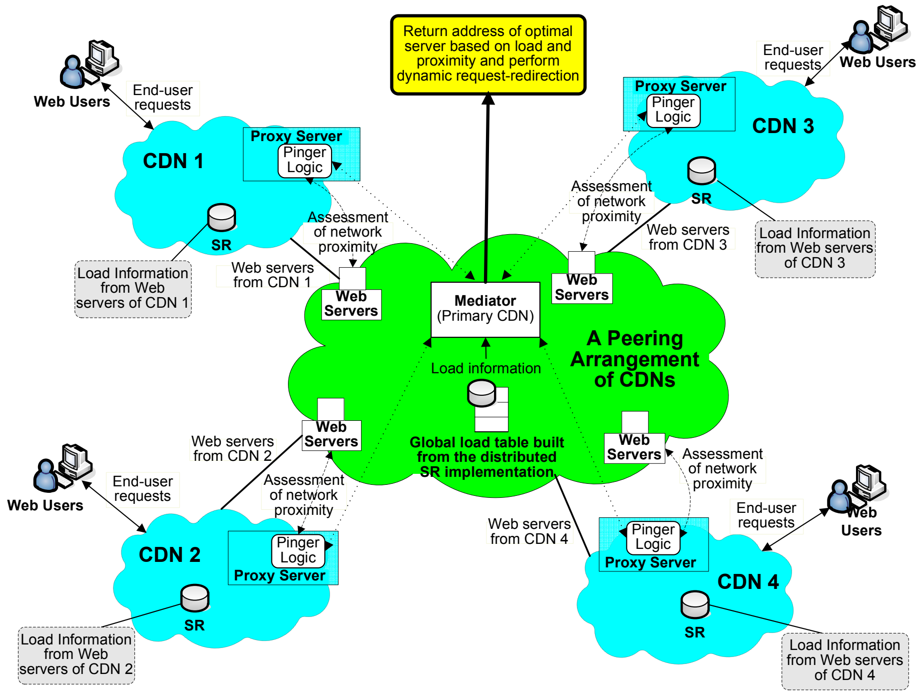

Return address of optimal server based on load and proximity and perform dynamic request-redirection
Web Users
End-user requests
CDN 1
Proxy Server
Pinger Logic
SR
Load Information from Web servers of CDN 1
Assessment of network proximity
Web servers from CDN 1
Web Servers
Mediator (Primary CDN)
Load information
Global load table built from the distributed SR implementation
A Peering Arrangement of CDNs
CDN 3
Proxy Server
Pinger Logic
SR
Load Information from Web servers of CDN 3
Web servers from CDN 3
Web Servers
CDN 2
Web servers from CDN 2
Pinger Logic
Proxy Server
Load Information from Web servers of CDN 2
CDN 4
Web servers from CDN 4
Load Information from Web servers of CDN 4

Fig. 3. Reference scenario for simulation

CDN servers (facilities) have no queuing delay, since they are configured to have high capacity and large bandwidth. This approach leads to the logical implication that servers in the simulation can handle any size of load. This assumption is necessary to deal with request arrivals with very large processing requirements [2]. The load distribution algorithm deployed in the mediator decides, upon receiving an alarm signal, whether a Web server is overloaded or not, depending on its load information from the global load table maintained by the distributed SR implementation.

Table 2 shows the configuration of the list of Web servers from the participating CDNs in the given peering arrangement. We have configured our servers according to the specifications from Fourth Quarter 2006 SPECweb2005 Results [1]. To anonymize this list, we have assigned a numeric identifier in the form $\mathtt{WS}_i$, $i \in \{1,2,...,M\}$ to each Web server. We have used the domain part of the DNS name to distinguish between providers and assigned a provider identifier in the form $\mathtt{CDN}_j$, $j \in \{1,2,...,N\}$ to illustrate which server is acquired from the same CDN provider. It has been found by Amini et al. [3] that a CDN provider may deploy unique IP addresses to its geographically diverse servers. Therefore, our list of simulated Web servers follows such deployments. Column 4 of Table 2 presents the capacity of the Web servers, expressed as their ability to serve the maximum number of requests/second. In order to calculate the capacity for each Web server, we normalize the total number of multiple user sessions (SPECweb2005 reported performance) by the number of connected client machines as reported in the results. We also model the

Table 2. Configuration of the simulated CDN Web servers

Server ID	*Provider ID*	*System*	*Capacity (Requests/second)*	*Service Distribution, PDF and Properties*
WS_1	CDN_1	Dell PowerEdge 2950, Intel Xeon 5160 Processor	906	Pareto, $\alpha k^{\alpha} x^{-\alpha-1}$ $\alpha = 1.25, k = 1$
WS_2	CDN_1	Dell PowerEdge 2950, Intel Xeon X5355 Processor	1052	Pareto, $\alpha k^{\alpha} x^{-\alpha-1}$ $\alpha = 1.25, k = 1$
WS_3	CDN_1	Dell PowerEdge 860, Intel Xeon 3070 Processor	506	Pareto, $\alpha k^{\alpha} x^{-\alpha-1}$ $\alpha = 1.25, k = 1$
WS_4	CDN_2	Fujitsu PRIMERGY TX150 S5, Intel Xeon3060 Processor	200	Log-normal, $\frac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}$ $\mu = 0.9681, \sigma^2 = 1.5846$
WS_5	CDN_2	Fujitsu PRIMERGY TX300 S3, Intel Xeon5355 Processor 8 cores, 2 chips	310	Log-normal, $\frac{1}{x\sqrt{2\pi\sigma^2}} e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}$ $\mu = 0.9681, \sigma^2 = 1.5846$
WS_6	CDN_3	HP ProLiant DL145 G2, AMD Opteron 285 Processor	471	Hyper-exponential, $\sum_{i=1}^{n} P_i \lambda_i e^{-\lambda_i x}$ $\mu = 0.5967, \sigma^2 = 2.6314$
WS_7	CDN_1	HP ProLiant DL380 G5, Intel Xeon 5160 Processor	552	Pareto, $\alpha k^{\alpha} x^{-\alpha-1}$ $\alpha = 1.25, k = 1$
WS_8	CDN_3	HP ProLiant DL585 G2, AMD Opteron 8212 Processor	624	Hyper-exponential, $\sum_{i=1}^{n} P_i \lambda_i e^{-\lambda_i x}$ $\mu = 1.65, \sigma^2 = 3.70$
WS_9	CDN_3	HP ProLiant DL585 G2, AMD Opteron 8220 Processor	843	Hyper-exponential, $\sum_{i=1}^{n} P_i \lambda_i e^{-\lambda_i x}$ $\mu = 1.10, \sigma^2 = 5.65$
WS_{10}	CDN_4	HP ProLiant DL585, AMD Opteron 885 Processor	660	Erlang, $\frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!}$ $\mu = 4.529, \sigma^2 = 0.321$

servers to follow different service distributions. Column 5 specifies the used service distributions along with their Probability Distribution Functions (PDFs) and associated properties.

5.1.2 Service Registry

There is a distributed implementation of the SR in our simulation, wherein each CDN has an SR instance to get the load status of its Web servers. While an SR instance keeps a local copy of the load status of the servers in a CDN, the load information is not necessarily propagated straightaway to the peering CDNs system. Once a peer delegates a set of its servers to define the subCDN, the real-time state of the load of this global subCDN or of each surrogate is passed to the global load table of SR using standard load metrics (`LoadMetric`). The global load table stores the overall state of the servers in the peering arrangement using a tuple

`(Web Server, LoadMetric)`

The asynchronous feedback by the Web servers allows proceeding with the operations without querying them for the updated load information. In this way, the servers are made to decide on the tradeoff between serving requests and updating load information. The overhead of load update messages could be high if a large number of Web servers from the participating CDNs are present in the system. However, a given peering arrangement is not likely to have more than a few hundred or thousand nodes at an instant. Moreover, the load information is updated periodically and the only data that needs to be sent is a few bytes of load index. For example: a total number of `1000` servers in a given peering arrangement, `30s` update frequency and `11bytes` of data transfer for each load table entry "`WebServer_ID LoadMetric`" would lead to only about `22KB/min`, or less than `0.5KBps` for the messaging overhead due to the asynchronous feedback by the servers to the SR.

5.1.3 Mediator

The mediator is simulated to act as an authoritative entity in a given peering arrangement. It monitors the global load table to get Web servers' status and uses the pinger logic to get their network proximity information, in terms of round-trip response time (in milliseconds), assuming that they are directly correlated. The pinger logic uses the User Datagram Protocol to send an `18bytes` proximity measurement query (as UDP packet) of the format "`PING Time CRLF`" to each Web server for the network proximity measurement. `Time` represents the timestamp when query is sent and `CRLF` represents the carriage return and line feed characters that terminate the query message. We also set a timeout period of `1000ms` to check whether the servers are reachable. Thus, along with the proximity measurement, the pinger logic tests a Web server's ability to respond to a proximity measurement query, as well as its level of responsiveness under the current load.

The pinger logic is deemed to be located close to the users so that the mediator ideally can have the same view of the network status as the user's browser. This approach allows estimating, reasonably accurately and possibly offline, the network proximity between the user and CDN Web servers. It may appear that this approach leads to a load distribution system that does not scale enough as it requires an instance of pinger logic to be placed close to each of the numerous number of users in the Internet. However, in practice, it is observed that most of the end-users make use of a proxy server which filters Web accesses through caching. Content which are not available in the proxy server are retrieved from the origin server(s) and stored locally in a cache. Additional requests for the same content are served by the proxy until the expiration time after which the content is considered 'stale'. Therefore, the scalability problem can be solved by placing the pinger logic in the proxy server. This solution is feasible because the pinger logic is activated by the mediator for network proximity measurement only when the requested content is not in the proxy cache.

5.1.4 Network Congestions

Two types of networks delays, namely, *inter-CDN delays* and *intra-CDN delays* are modeled to consider the impact of network congestions on load distribution strategies. In addition, for network proximity measurement we have simulated the UDP packet loss (`LOSS_RATE = 0.3`) due to network congestions. Intra-CDN delays have three components: (i) minimum round-trip time between server and the user browser, (ii) queuing delays at the mediator[3], and (iii) packet transmission time for each link on the user browser and CDN Web server path. While (i) and (ii) are measured through simulation, (iii) is modeled as average network delay of `100ms`. Inter-CDN delays are random variables that model communication latency between different geographical CDN domains. The Inter-CDN delays are `200ms` in average. In the model, we do not characterize the delays spent for address resolution of Web servers.

5.1.5 End-Users

User requests are implemented as CSIM processes[4]. We assume that end-users request content via their own browsers to the CDN, according to the same client-side policy. The hidden load weight [9] is implicitly taken into account through the user distribution to CDNs, as requests to different CDNs are properly weighed and are distributed to the servers in a given peering arrangement.

Alike the Internet access workloads, these user requests exhibit self-similarity. A self-similar process has observable bursts in all time scales. It exhibits long-range dependence, where values at any instance are typically correlated with all future values. This self-similar nature in user requests can be described by using a heavy-tailed distribution [11][12]. Therefore, user requests to each CDN Web server follow a highly variable Pareto distribution with Probability Density Function (PDF),

$$f(x) = \alpha k^{\alpha} x^{-\alpha-1}, \alpha, k > 0, x \geq k$$

where the weight of the tail of the distribution is determined by $\alpha < 2$.

5.2 Schemes and Metrics for Comparison

We experiment with three other load distribution policies and compare their performances. The *LD_RR* policy uses a Round-Robin (RR) approach to redirect excess requests to a server, exceeding the alarm threshold, to all the available underloaded servers of participating CDNs in a cyclic order. In *LD_PRR* policy, request-redirection is performed using a Probability-based Round Robin (PRR) approach. The probability is based on the residual capacity of servers in a given peering arrangement using the latest server load index. The *LD_LL* policy uses a very simple approach that performs load distribution by redirecting excess requests to the least loaded server. Just for comparison purpose, we also consider *no redirection* which tries to assign requests to the closest server, without considering the load.

[3] The queuing delay at the mediator occurs for any possible congestion during load distribution in a given peering CDNs arrangement.

[4] CSIM processes are objects, based on Java threads, which make use of simulated resources.

For performance comparisons we first use the *number of ongoing requests* at each server. A desirable scheme should keep the number below the capacity limit of each server all the time. We use server *utilization* to demonstrate the performance of the proposed redirection scheme. To emphasize the impact of redirection on load distribution of primary CDN servers, we define *maximum utilization* at a given instant as the highest utilization at that instant among all primary CDN servers. Specifically, the major performance criterion is the *cumulative frequency* of maximum utilization, i.e. the probability that the maximum utilization of the primary CDN is below a certain value. By focusing on the highest utilization among all primary CDN servers, we can deduce whether the primary CDN is overloaded or not.

6 Experiment Results

In this section, simulation results are presented to evaluate performance of the proposed request-redirection scheme that is used to perform load distribution in peering CDNs. We also provide a critical assessment of our approach highlighting its benefits. We run our simulations based on the reference scenario in Figure 2, with CDN 1 as the primary and others as the peering CDNs. Our results are obtained from ten simulation runs, where each run is for a duration of `10000s`. In our simulations, we consider `80%` of a server's utilization as the `alarm_threshold`.

6.1 Server Load

We first present the number of connections at each server in different redirection schemes. Figure 4 shows the number of concurrent requests at each server over time. For the clarity of presentation we use two scales—Scale 1 and Scale 2—to show the concurrent flows at servers of primary and peering CDNs, respectively.

Since server load is not taken into account in the no redirection policy and user requests are sent to the closest server, from Figure 4(a) we observe that the load at only a few servers, in the primary CDN domain, grow significantly. For example: throughout the simulation, Server 1 and Server 2 receive more requests than their capacity. Unless they are provisioned with enough capacity to serve more concurrent connections, they will end up dropping many requests. In Figure 4(b), we see that LD_RR performs some load distribution by sending extra requests to the underloaded servers from primary and peering CDNs. However, it does not take cost (in terms of traffic load and proximity) into account and can potentially lead to high redirection cost. Figure 4(c) presents the performance of LD_PRR. Since it assigns some probability to the underloaded servers based on their residual capacity, it redirects more requests to the server(s) with high probability. Therefore, it performs some load distribution but may cause sudden surge to a particular server and thus lead to imbalance load situations. For example: during simulation time 8 and 10 (`x10000s`), Server 1 receives many more requests than its capacity. LD_LL does not perform well as it fails to distribute loads to multiple servers. As for instance, from Figure 4(d), we observe that as simulation time passes, Server 1 receives more requests than its capacity specifically during simulation times 2, 4, 6, and 8 (`x10000s`), and Server 7 constantly receives extra requests than what it can handle. In Figure 4(e), we present the performance of LD_minCost. The main objective of LD_minCost is to redirect extra requests in an imbalanced load situation, minimizing the redirection cost in terms of traffic load and network proximity, without violating the (practical) server capacity constraints. We observe from the figure that none of the primary CDN servers operate beyond their capacity during simulation. In order to prevent wide oscillation in the load distribution decisions, incoming requests to the primary CDN stay within its domain as much as possible and only cross the inter-CDN barrier during excessive load imbalance. Since redirection cost is taken into account, requests are served by the optimally underloaded servers, in terms of network cost. As a result, a few primary CDN servers receive only relatively few requests, while other better located servers run close to their capacity. In addition, service disruptions (or request dropping) are minimized as LD_minCost leads to optimal server selections. Moreover, any dynamic load changes, e.g. sudden load increase in Server 9 at simulation time 5 (`x10000s`), are also taken into account and load is distributed in a timely fashion. Therefore, even under high traffic surges such as flash crowds, our redirection scheme performs well.

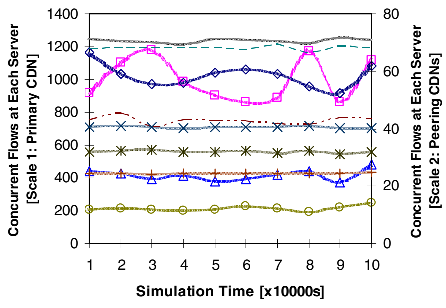
6.2 Server Utilization

Figure 5 presents the utilization of the servers in the peering CDNs arrangement for each request-redirection scheme. Here, we observe that LD_minCost performs better than the other redirection schemes in order to perform load distribution of the overloaded primary CDN servers. With no redirection, most of the primary CDN servers receive excessive traffic and utilization stays over the alarm threshold. While LD_RR and LD_PRR demonstrate similar characteristics to perform some load balancing, none of them reduces utilization of all the primary CDN servers below the threshold. LD_LL policy fails to perform load distribution among multiple servers and always overloads Server 1 and Server 2. On the other hand, LD_minCost exhibits the best performance by reducing all the primary CDN utilization below the threshold. Since requests stay within the primary CDN domain as much as possible to minimize redirection cost, a better located server, e.g. Server 3, may show more utilization than its allies. However, still the primary CDN operates under an acceptable level of load.

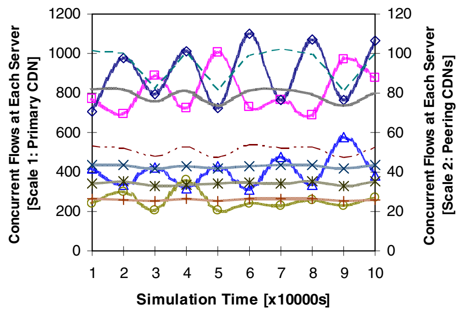
It is also evident from Figure 6, which presents the average utilization of the primary CDN system in each redirection scheme. LD_RR, LD_PRR, and LD_LL do not reduce the average utilization significantly below the threshold. Therefore, with these schemes primary CDN servers may be unable to receive more requests during sudden excessive traffic (flash crowds) in future and thus requests may have to be redirected outside the domain. With LD_minCost the average utilization of the primary CDN system is significantly brought down and makes its servers possible to cope with succeeding traffic outbursts.

6.3 Redirection Performance

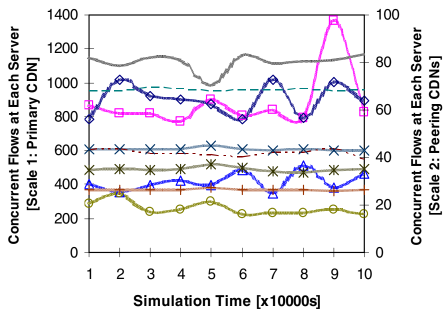
In Figure 7, we show the average redirection percentage in each scheme. For the clarity of presentation we use two scales—Scale 1 and Scale 2—to respectively plot the redirection percentage from LD_minCost and other schemes (LD_RR, LD_PRR, and LD_LL). It shows that our redirection scheme assists in redirecting more requests over time in case of load imbalance. On the other hand, LD_RR, LD_PRR, and LD_LL do not show a high redirection percentage under traffic surge. Therefore, many requests are dropped as incoming requests arrive to a primary CDN server and find that the server operating at its highest capacity. Thus, the inability to redirect more requests in these schemes can lead to significant service disruptions.



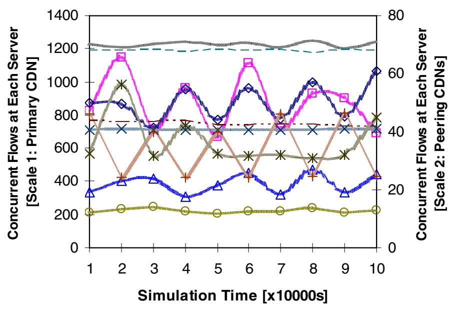
(a) No Redirection



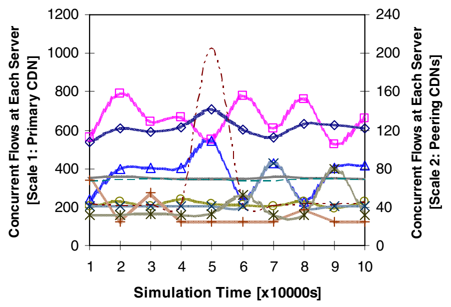
(b) LD_RR

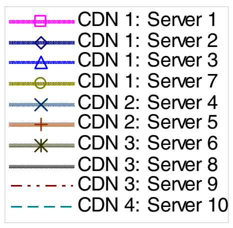


(c) LD_PRR



(d) LD_LL





(e) LD_minCost

Fig. 4. Number of concurrent requests for each scheme

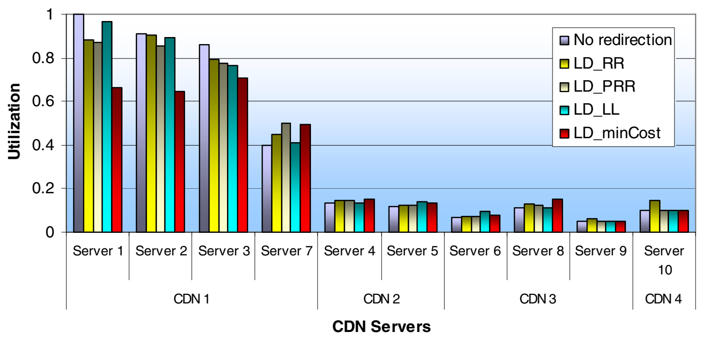


Fig. 5. Server utilization for each scheme

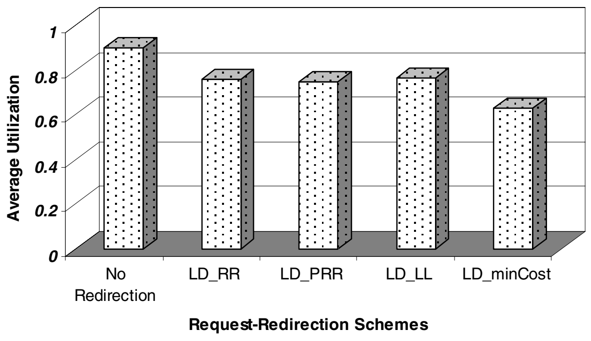


Fig. 6. Average utilization of the primary CDN system for each scheme

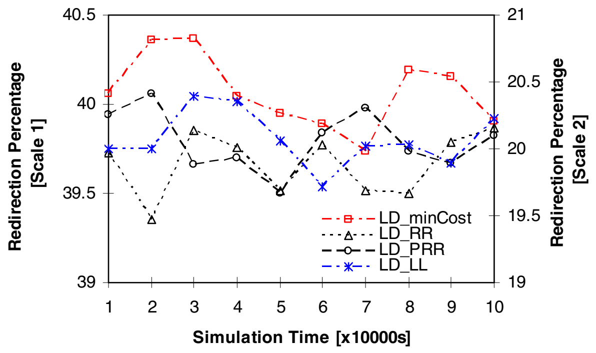


Fig. 7. Redirection percentage in each scheme

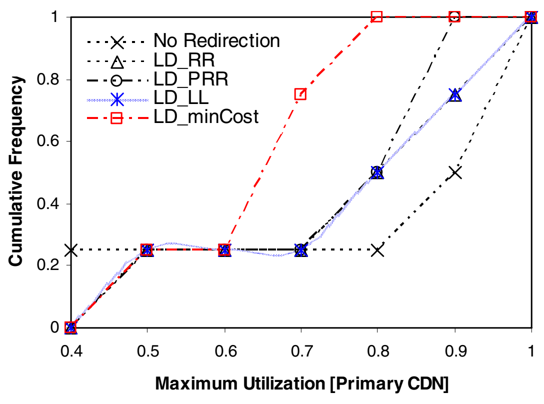


Fig. 8. Comparison of the request-redirection schemes

Figure 8 summarizes the performance of the request redirection schemes in terms of cumulative frequency of maximum utilization of primary CDN servers. It shows that LD_minCost has a probability of 1.0 of not causing any primary CDN server to exceed 80% utilization (threshold). From the figure we can see that other schemes such as LD_RR, LD_PRR and LD_LL do not perform well. Specifically, they exhibit a probability of 0.5 of not causing any Web server to exceed the threshold. Moreover, as predicted, with no redirection there is very low probability of only 0.25 that primary CDN servers will operate under the threshold.

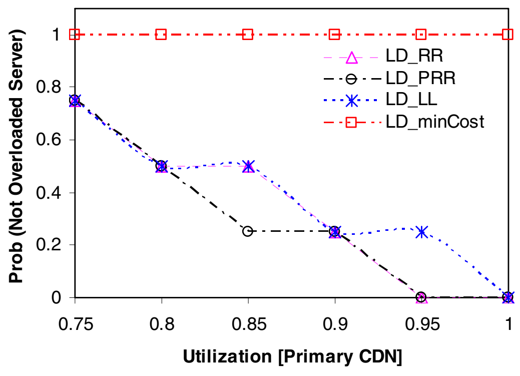


Fig. 9. Sensitivity to system utilization for each scheme

6.4 Sensitivity Analysis

We now show performance comparison of the request-redirection schemes through a sensitivity analysis for the primary CDN, considering critical parameters such as *system utilization* and *end-user request distribution*. We use the probability that no server of the primary CDN is overloaded as the performance metric. Changing other system parameters, such as the average number of requests, total simulation time or user think time do not show noticeable differences among the redirection approaches. Therefore, we do not present those results here.

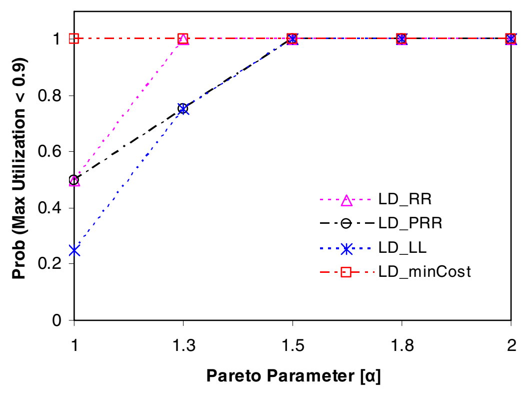


Fig. 10. Sensitivity to end-user distribution for each scheme

In Figure 9, we compare the sensitivity to the system utilization for the primary CDN in each redirection scheme, using the probability that no server in its domain is overloaded as the performance metric. Therefore, we vary the alarm threshold from 0.75 to 1. We observe that our redirection scheme, LD_minCost, shows the best result in all the cases. Analogous conclusion can be drawn when we vary the distribution of users among the CDNs. Figure 10 shows the probability that no server has a utilization higher than 90% as a function of the shape parameter α, which is varied from 1 (high variability) to 2 (moderate variability). We change the performance metric (threshold) from 80% to 90% to show the novelty of our scheme even under highly variable request pattern.

7 Conclusion and Future Work

In this paper, we present a dynamic load distribution algorithm that uses load and proximity-aware request-redirection to alleviate any load imbalance in a peering CDNs system. In our approach, when any CDN Web server in a peering CDNs arrangement reaches an overload condition exceeding the alarm threshold, the load distribution strategy reacts to redirect loads by selecting available optimally underloaded server(s), while not compromising network proximity. We validate our proposal with the aid of simulation, considering practical constraints and significant system parameters; and demonstrate its performance using performance metrics such as number of ongoing connections, server utilization and the cumulative frequency of maximum utilization. We also perform a sensitivity analysis of our redirection scheme by taking critical system parameters into account.

To the best of our knowledge, no prior work in the CDN domain has attempted to perform load distribution in peering CDNs through a request-redirection mechanism which takes network cost (in terms of load and network proximity) into account and cope up with practical constraints in CDN domain. Our approach can alleviate problems with the commonly used DNS-dispatching policies for load balancing, which does not provide sufficient control on user requests, e.g. it can have as little as 5% of requests in many instances [9]. Moreover, DNS-dispatching is less useful for fine-grained server selections. On the other hand, our approach achieves a significant level of granularity, since the actual user requests are redirected during load distribution among the Web servers in peering CDNs. Therefore, it endeavors to produce optimal Web server selections in a given peering arrangement of CDNs, even under degenerated load conditions.

Our future work includes performing experiments in the real-world settings, such as PlanetLab, to validate the methodology presented in this paper. Our future work[5] also includes developing a proof-of-the-concept prototype implementation for peering CDNs, in order to demonstrate the real-time application of our load distribution strategy. We also plan to develop efficient *resource dissemination* and *discovery* algorithms for discovering resources from disparate CDNs. Thus, we will exploit the capabilities of peering CDNs to avoid network hotspots to further enhance our approach. As far as the simulation is concerned, we aim to adopt traces of the Internet traffic for load characterization purposes. The use of actual CDN traces will help us to obtain reasonable load and network proximity estimates, which could be refined over time to improve their accuracy, in response to changes in the network.

Acknowledgement

We would like to thank Marcos Assunção, SungJin Choi, Mustafizur Rahman and Rajiv Ranjan from the University of Melbourne, Australia for sharing thoughts and for making incisive comments and suggestions on this paper. We are also thankful to anonymous reviewers for their valuable and constructive comments to improve the paper's structure, quality and readability. This work is supported in part by the Australian Research Council (ARC), through the discovery project grant and Department of Education, Science and Training (DEST), through the International Science Linkage (ISL) grant.

References

[1] Standard Performance Evaluation Corporation (2008), `http://www.spec.org/`

[2] Alzoubi, H.A., Lee, S., Rabinovich, M., Spatscheck, O., Van der Merwe, J.E.: Anycast CDNs revisited. In: Proc. of the 17th International Conference on World Wide Web, pp. 277–286. ACM Press, New York (2008)

[3] Amini, L., Shaikh, A., Schulzrinne, H.: Modeling redirection in geographically diverse server sets. In: Proc. of the 12th International Conference on World Wide Web, pp. 472–481. ACM Press, New York (2003)

[4] Barbir, A., Cain, B., Nair, R., Spatscheck, O.: Known content network (CN) request-routing mechanisms. IETF RFC 3568 (July 2003)

[5] For more information about our efforts on peering CDNs, please visit the project Web site at www.gridbus.org/cdn.

[5] Biliris, A., Cranor, C., Douglis, F., Rabinovich, M., Sibal, S., Spatscheck, O., Sturm, W.: CDN brokering. Computer Communications 25(4), 393–402 (2002)

[6] Buyya, R., Pathan, M., Vakali, A. (eds.): Content Delivery Networks. Springer, Germany (2008)

[7] Cardellini, V., Colajanni, M., Yu, P.S.: Dynamic load balancing on Web-server sys-tems. IEEE Internet Computing 3(3), 28–39 (1999)

[8] Cardellini, V., Colajanni, M., Yu, P.S.: Redirection algorithms for load sharing in distributed Web-server systems. In: Proc. of 19th IEEE International Conference on Distributed Computing Systems, Austin, Texas, USA (May 1999)

[9] Colajanni, M., Yu, P.S., Cardellini, V.: Dynamic load balancing in geographically distributed heterogeneous Web servers. In: Proc. of the 18th International Conference on Distributed Computing Systems, pp. 295–302 (1998)

[10] Conti, M., Gregori, E., Panzieri, F.: QoS-based architectures for geographically replicated Web servers. In: Cluster Computing, vol. 4, pp. 109–120. Kluwer Academic Publishers, The Netherlands (2001)

[11] Crovella, M.E., Bestavros, A.: A self-similarity in World Wide Web traffic: evidence and possible causes. IEEE/ACM Transactions on Networking 5(6) (1997)

[12] Crovella, M.E., Taqqu, M.S., Bestavros, A.: Heavy-tailed probability distributions in the World Wide Web. In: A Practical Guide To Heavy Tails, pp. 3–26. Birkhauser Boston Inc., Cambridge (1998)

[13] Day, M., Cain, B., Tomlinson, G., Rzewski, P.: A model for content internetworking. IETF RFC 3466 (February 2003)

[14] Pallis, G., Vakali, A.: Insight and perspectives for content delivery networks. Communications of the ACM 49(1), 101–106 (2006)

[15] Pathan, M., Broberg, J., Bubendorfer, K., Kim, K.H., Buyya, R.: An architecture for virtual organization (VO)-based effective peering of content delivery networks. In: UPGRADE-CN 2007, In Proc. of the 16th IEEE International Symposium on High Performance Distributed Computing, California, USA (June 2007)

[16] Ranjan, S., Karter, R., Knightly, E.: Wide area redirection of dynamic content by Internet data centers. In: Proc. of 23rd Annual IEEE Conference on Computer Communications (2004)

[17] Shnayder, V.: Load and proximity aware request distribution. Princeton University Report (2003), `http://citeseer.ist.psu.edu/636009.html`

[18] Zhu, Y., Hu, Y.: Efficient, proximity-aware load balancing for DHT-based P2P systems. IEEE Transactions on Parallel and Distributed Systems 16(4), 349–361 (2005)