IEICE TRANS. COMMUN., VOL.E95–B, NO.7 JULY 2012

PAPER Special Section on Future Internet Technologies against Present Crises

On Tackling Flash Crowds with URL Shorteners and Examining User Behavior after Great East Japan Earthquake*

Takeru INOUE[†a)] ***and*** **Shin-ichi MINATO**[††,†]**,** ***Members***

SUMMARY Several web sites providing disaster-related information failed repeatedly after the Great East Japan Earthquake, due to *flash crowds* caused by Twitter users. Twitter, which was intensively used for information sharing in the aftermath of the earthquake, relies on *URL shorteners* like bit.ly to offset its strict limit on message length. In order to mitigate the flash crowds, we examine the current Web usage and find that URL shorteners constitute a layer of *indirection*; a significant part of Web traffic is guided by them. This implies that flash crowds can be controlled by URL shorteners. We developed a new URL shortener, named *rcdn.info*, just after the earthquake; rcdn.info redirects users to a replica created on a CoralCDN, if the original site is likely to become overloaded. This surprisingly simple solution worked very well in the emergency. We also conduct a thorough analysis of the request log and present several views that capture user behavior in the emergency from various aspects. Interestingly, the traffic significantly grew up at previously unpopular (i.e., small) sites during the disaster; this traffic shift could lead to the failure of several sites. Finally, we show that rcdn.info has great potential in mitigating such failures. We believe that our experience will help the research community tackle future disasters.

key words: *flash crowd, URL shortener, CDN, disaster, Great East Japan Earthquake*

1. Introduction

On March 11th 2011, a great earthquake and tsunami hit eastern Japan. After that, several web sites, especially those providing helpful disaster-related information like shelters or radiation levels, were overloaded by *flash crowds* (we roughly define flash crowds as sharp traffic growth that overwhelmed a web site). They had been previously unpopular sites with only small traffic, but massive numbers of users, most of whom were likely to be Twitter users, visited them in a short period of time. Twitter provides a handy reposting function called *retweet*, which allowed users to quickly spread their favorite *tweets* (Twitter's short messages). This function probably drove the users to jump on the sites. In normal times, Twitter flash crowds are not a serious problem, since most tweets cover trivial topics in Japan [5]. On March 11th, however, the number of tweets increased by 80% and 72% of the increased total number was related to the disaster [5]. Since the disaster information was considered to be essential to surviving the crisis, the site failures became a crucial issue.

There are two common solutions to flash crowds, but they were not useful in the emergency.

- **Scaling web sites.** Web sites can be scaled up by upgrading server machines or adding new machines. This is now a reasonable solution thanks to scalable cloud technologies. Some sites increased their capacity through the help of hosting companies that provided their service for free (only for the sharing of disaster information). However, it is difficult for most sites to react quickly in the aftermath of an earthquake (some site administrators might be stuck in shelters themselves).
- **Mirroring pages.** Flash crowds can be mitigated by posting URLs of mirror sites. There is no difficulty in using mirror sites normally. However, in the chaos following a disaster, we cannot imagine that people will stay cool enough to replace the original URLs with those of the mirror sites before posting messages.

Since Twitter limits its message length to just 140 characters, *URL shorteners* like bit.ly are often used to shorten long URLs**. Twitter client software automatically replaces a long URL in a tweet with a short URL created by a URL shortener. By clicking the short URL, the user is redirected to the original site via the URL shortener. The use of URL shorteners is transparent to the Twitter user, manual replacement of URLs is not needed.

We launched a new URL shortener named *rcdn.info* just after the disaster; rcdn.info redirects users to a replica created on a CDN (content delivery/distribution network), if the original site can be overloaded. We chose CoralCDN [25] as the target CDN. CoralCDN is a research network designed to mitigate flash crowds, and has capacity sufficient to handle tens of millions of requests per day [24]. This is a quite simple solution, but its simplicity must be guaranteed to work in an emergency. Our solution is free from the problems stated above; rcdn.info can be introduced quickly without the intervention of site administrators, and its use is transparent to Twitter users.

We summarize our contributions as follows.

- We find that URL shorteners constitute a layer of *indirection* [35] in the current web. Even though the web

Manuscript received October 17, 2011.
Manuscript revised February 15, 2012.

[†]The authors are with JST ERATO Minato Discrete Structure Manipulation System Project, Sapporo-shi, 060-0814 Japan.
[††]The author is with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060-0814 Japan.

*An earlier version of this paper appeared in [27].

a) E-mail: takeru.inoue@ieee.org

DOI: 10.1587/transcom.E95.B.2210

**Technically, URL (uniform resource locator) should be replaced with URI (uniform resource identifier) as defined by [14], but we use the term URL in this paper since the shortening services are usually called URL shorteners.

contains several layers of indirection, this is the only indirection that can be deployed quickly in an emergency. Since indirection is allowed to route traffic, flash crowds can be diverted from overloaded sites on the layer. We design rcdn.info based on this idea. (Sects. 4 and 5)
- We thoroughly analyze the HTTP request log collected at rcdn.info. This log has extremely valuable information in capturing user behavior in the emergency; to the best of our knowledge, no work has analyzed user behavior in an emergency with HTTP requests. We quantitatively discuss the behavior from various aspects, such as users' purpose and regionality. We also show that the traffic significantly grew up at previously unpopular sites in the aftermath of the earthquake. (Sect. 6)
- We show that rcdn.info can greatly reduce the number of requests received by the original sites. Moreover, though rcdn.info increases the latency due to the indirection, we confirm that the increase is quite small and it is acceptable. (Sect. 7)

We believe that it is worth sharing this very rare experience as a way to prepare for future disasters[†].

The rest of this paper is organized as follows. Section 2 reviews the Great East Japan Earthquake, URL shorteners, and CoralCDN. After describing the requirements in Sect. 3, Sect. 4 shows our basic idea, and Sect. 5 discusses the design of rcdn.info. Section 6 analyzes the log to unveil user behavior, and Sect. 7 evaluates the performance of rcdn.info. Section 8 summarizes related work, and finally Sect. 9 concludes with lessons learned from the disaster.

2. Background

2.1 Great East Japan Earthquake

A great earthquake of magnitude 9.0 occurred off the eastern coast of Japan (Fig. 1), at 14:46 JST on March 11th 2011. It was the most powerful known earthquake to have hit Japan, and one of the five most powerful earthquakes in the world since modern record-keeping began in 1900. The earthquake triggered extremely destructive tsunami waves along the eastern coast of Japan. In addition to the significant loss of life and destruction of the infrastructure, the tsunami caused a number of nuclear accidents at the Fukushima Daiichi nuclear power plant; several hydrogen explosions at the reactors scattered low level radioactive materials around the local area. The nuclear incidents greatly disturbed people living in a wide area that included Tokyo. They rushed onto the web searching for helpful information and this public action formed flash crowds.

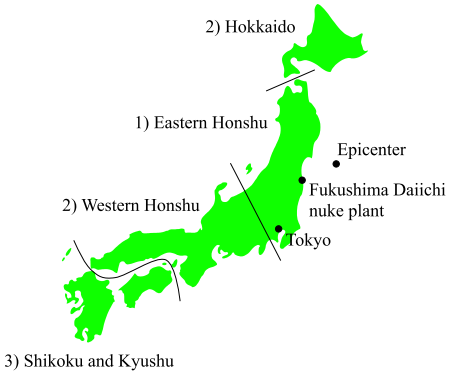

Fig. 1 Map of Japan showing the epicenter of the earthquake and the Fukushima Daiichi nuclear power plant.

2.2 URL Shorteners

The first URL shortener, "Make A Shorter Link", was launched in 2001 to eliminate the frustration of copying very long URLs [8]. Currently, URL shorteners are mainly used by short messaging services like Twitter, which severely limit the number of characters in a message. A URL shortener, which usually has a short domain name, generates a unique alphanumeric key for each URL. The domain name and unique key form a short URL like `http://bit.ly/v1Anp`, which redirects visitors to the original URL. A substantial amount of web traffic currently goes through URL shorteners. One report indicated that short URLs on bit.ly, the largest URL shortener, were accessed 2.1 billion times in November 2009 [3].

We show an example of message sequences with *shortening* a long (original) URL and *expanding* a short URL in Fig. 2. We omit DNS name resolution of the URL shortener's domain in the example.

- **Shortening original URL.** We show an example

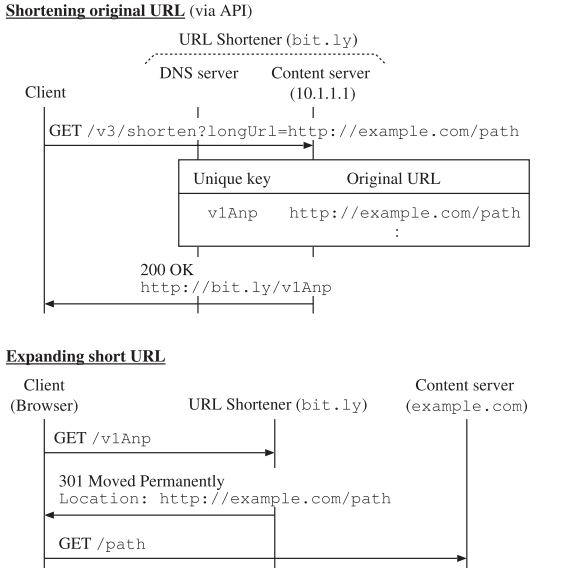

Fig. 2 An example of shortening an original URL and expanding a short URL at a URL shortener (DNS queries are omitted).

[†]Our anonymized data is now publicly available online at `http://rcdn.info/data.html`.

of shortening a URL via an API. A client makes an HTTP request for shortening a long URL (e.g., `http://example.com/path`). Upon receiving the request, the URL shortener generates a unique key (e.g., `v1Anp`) for the URL, and stores the key with the original URL in its database. The URL shortener, then, responds with the short URL (e.g., `http://bit.ly/v1Anp`).

- **Expanding short URL.** By entering or clicking this short URL, a client, or a web browser, makes an HTTP request for the short URL. After receiving the request, the URL shortener retrieves the associated URL from its database. The shortener, then, returns a response with 301 status code and a `Location` attribute. Finally, the client is redirected to the original URL.

The use of URL shorteners is transparent to Twitter users, because Twitter clients retrieve a short URL in the background and the original URL is automatically replaced with the short URL. URL shorteners are usually open service; users are allowed to use them without prior registration and authorization.

2.3 CoralCDN

CoralCDN is a research network developed by Michael Freedman in 2004 [25]. It was designed to mitigate flash crowds; for example, CoralCDN distributed large quantities of amateur videos of the Indian Ocean tsunami in 2004. CoralCDN is fully open and requires no prior registration or authorization. It can be used just by adding `.nyud.net` to the domain name in the original URL, e.g., `http://example.com/path` → `http://example.com.nyud.net/path`. URLs including `.nyud.net` are called "Coralized URLs". CoralCDN can be used easily but its use is not transparent to users since the replacement of URLs must be done manually.

CoralCDN is deployed on PlanetLab [19] and typically runs on 300–400 servers, spread over 100–200 sites worldwide [24]. It has sufficient capacity to handle 40–50 million requests per day. CoralCDN consists of DNS name servers and proxy servers; DNS servers maintain "A" records for `.nyud.net` and the proxy servers keep replicas of the original pages. Each replica is mapped to several proxy servers by Sloppy DHT technology [23], but the replica can be duplicated on other proxies depending on popularity (the number of requests). Replicas are updated after the expiry time, which is specified by the response header given in retrieving the original page. They are kept for at least five minutes, even if `No-Cache` directives are set in the response, and are removed within twenty-four hours at most.

Figure 3 shows an example of a message sequence in CoralCDN. Upon receiving a DNS query for `example.com.nyud.net`, a DNS name server in CoralCDN responds with an IP address of a proxy server. The client sends an HTTP request to the proxy server, which searches for a replica of the requested page in CoralCDN by using DHT. If the replica is not found, the proxy retrieves the requested page from the original site by issuing a DNS query and an HTTP request. Finally, the proxy returns the page to the client.

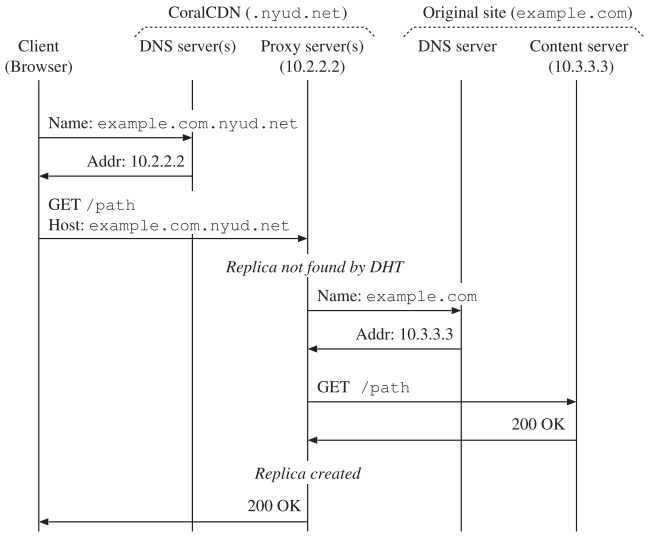

Fig. 3 An example of CoralCDN's behavior.

CoralCDN servers are mainly located in North America and Europe, which are far from Japan. Moreover, CoralCDN does not well consider the distance in choosing a proxy server as far as we know. Consequently, users in Japan would experience poor latency with CoralCDN. A latency optimization mechanism called DONAR has been recently introduced [36], but most requests are still handled by the original CoralCDN mechanism.

3. Requirements

This section presents our requirements for preserving small but useful web sites in the aftermath of the earthquake. The first three were discussed in Sect. 1, and the remaining three are general requirements for all online services.

- **Mitigating flash crowds.** This is our primary goal. To prevent the failure of web sites, our solution must tame flash crowds and reduce the request rate to the ordinary level. If our solution works well, more users could access useful information on the sites.
- **Rapid deployment.** Our solution must be deployed quickly, since information access is considered to be imperative for surviving any crisis.
- **Transparent to use.** We should not load the user with extra operations to make a posting, since such inefficient approach will fail in an emergency.
- **Better performance.** We should reduce latency as far as is possible, though load balancing generally comes at the cost of increased latency.
- **Open to many sites.** We should save as many web sites as possible. This means that our solution should not be limited to a particular site.
- **Secure to use.** Our solution should be secure to users,

even if it is open.

4. Basic Idea

This section describes our basic idea for satisfying the requirements presented in the previous section. We first compare the two solutions described in Sect. 1. The first solution, scaling web sites, requires the help of site administrators, which is often infeasible in an emergency and prevents rapid deployment. Moreover, this solution is limited to a particular web site; we have to cover any and all sites that should be preserved. We think that these problems are too difficult to fix. The second solution, mirroring pages, does not have these problems, but it is not transparent to users and forces detours to replicas. These issues, however, can be solved to some extent. We discuss our idea, based on the second solution, that addresses the remaining issues. Section 4.1 discusses how to guide users to replicas, and Sect. 4.2 shows where to create replicas.

4.1 How to Guide Users to Replica

The current web relies on several layers of indirection [35], which decouples senders (clients) from receivers (servers). These layers include IP, DNS, and HTTP, as shown in Table 1; for example, DNS allows a client to communicate with a web server without knowing the location (IP address) in advance. Since indirection can be used to divert traffic to some other place, it is often used for load distribution as in round-robin DNS.

Figure 4 illustrates the three layers of indirection seen in accessing a short URL. First, a URL shortener redirects a client to the original site (e.g., `http://example.com/path`). Next, the DNS server of `example.com` responds with an IP address of a reverse proxy [20] in one of several linked datacenters. Finally, the reverse proxy chooses a content server in the datacenter selected. In addition to the traditional layers of indirection, such as DNS and reverse proxying (and IP anycast but omitted in the figure), URL shorteners constitute a new layer of indirection by using HTTP redirection. While it may be difficult to quickly deploy the traditional indirection since site administrators' help is required, URL shorteners are not tied to any particular site and can be deployed independently. Though existing URL shorteners redirect clients only to the original URLs, technically they can point to replicas on a CDN by rewriting the URLs.

Table 1 Layers of indirection.

Layer of indirection	Techniques to rendezvous
IP	IP anycast
DNS	Round-robin DNS
HTTP	Proxying and redirection

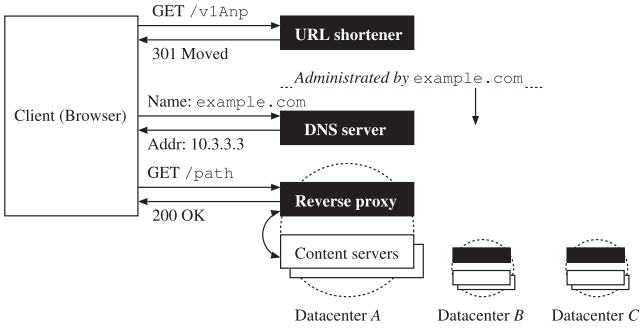

Fig. 4 An example of layers of indirection (black boxes) experienced in accessing a short URL.

URL shorteners are transparent to Twitter users as described in Sect. 2.2. They should not redirect users to a replica that includes malicious contents. We will discuss this issue in Sect. 5.4.

4.2 Where to Create Replicas

Several places are possible for replicating original pages. Commercial CDNs provide large capacities for replication, but it is difficult to make a contract with them because we cannot estimate the number of pages to be replicated in the emergency. Page cache of search engines is already used as a replica when the original page cannot be accessed. However, the cache is not guaranteed to be up to date even if the expiry time exceeds. Information should be the latest in an emergency.

We choose CoralCDN as the place for replication. CoralCDN can be used without prior registration or authorization. The replicas are updated after the expiry time.

Unfortunately, users in Japan would experience poor latency with CoralCDN, as described in Sect. 2.3. We resolve this performance issue as follows; Coralized URLs are accessed by rcdn.info before clients to create a replica on CoralCDN, or clients are redirected to the *original* site if the site seems to have sufficient capacity (i.e. not overloaded).

5. Design and Implementation of rcdn.info

This section describes the design and implementation of rcdn.info in detail. We first show the usage in Sect. 5.1. Section 5.2 presents the system configuration and Sect. 5.3 describes procedures of shortening and expanding. Section 5.4 discusses implementation matters.

5.1 Usage

We briefly describe the usage of rcdn.info here. The APIs shown in Table 2 are designed following bit.ly APIs [2].

Shortening original URL. We provide three ways to shorten a long URL.

- **API.** A Twitter client sends a URL to the "shorten API" in Table 2, and rcdn.info returns a short URL in a specified format.
- **HTML form.** A user submits a URL by the form on

Table 2 rcdn.info APIs.

Operation	URL
Shorten	`http://rcdn.info/api/shorten?longUrl={originalURL&}format={json|xml|text}`
Expand	`http://rcdn.info/api/expand?shortUrl={shortURL&}format={json|xml|text}`

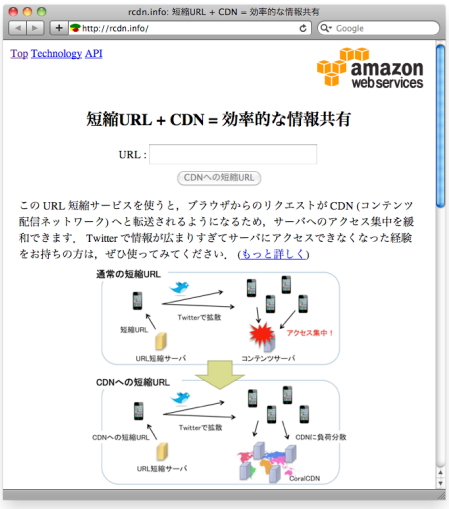

Fig. 5 The top page of rcdn.info; the title says "short URLs and CDN offer effective information sharing", the button says "(make) a short URL to CDN", and a short description accompanies the figure.

the top page, and rcdn.info then returns with a short URL (Fig. 5)[†].

- **Bookmarklet.** A user clicks the rcdn.info bookmarklet installed on the browser, which pops up a short URL of the current page.

Twitter clients that accept any URL shortener, such as TweetDeck [7] or YoruFukurou [10], provide transparency to users through the APIs, by setting rcdn.info as the default URL shortener. The HTML form and the bookmarklet are offered to users of other clients.

Expanding short URL. We provide two ways to expand a short URL.

- **Redirection.** A user clicks a short URL, and rcdn.info then redirects her to CoralCDN or the original site.
- **API.** A Twitter client sends a short URL to the "expand API" in Table 2, and rcdn.info returns a Coralized URL or original URL without redirection.

5.2 System Configuration

Figure 6 shows the system configuration of rcdn.info. The application server executes URL shortening and URL expanding. The database maintains tuples of a unique key, an original URL, and a Coralized URL. A list of "large capacity sites" is also maintained (not shown in the figure) to determine where to redirect, i.e., to CoralCDN or the original site as described in Sect. 4.2. The list includes some domains in the Alexa top 10 in Japan, since they have large capacity and are unlikely to be overloaded. We also require a DNS name server that maintains an "A" record of rcdn.info.

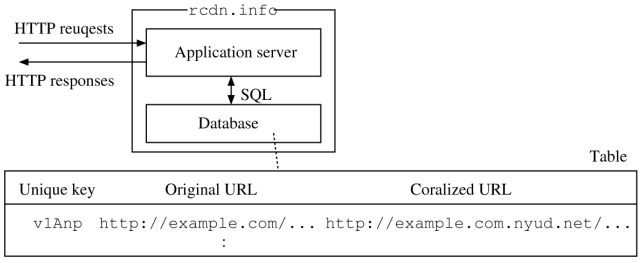

Fig. 6 The system configuration of rcdn.info.

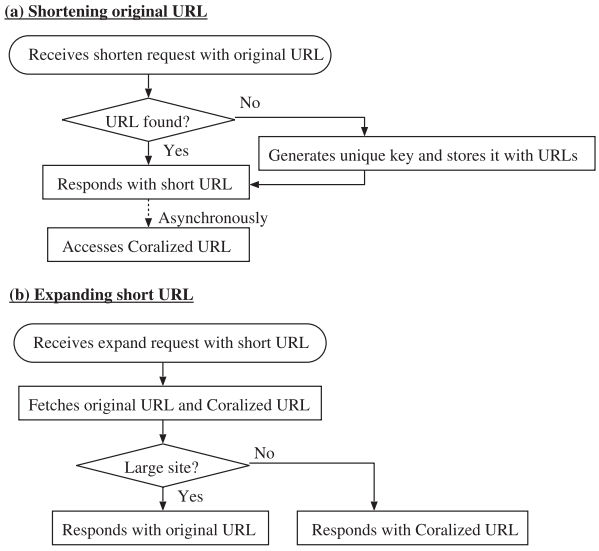

Fig. 7 Procedures of shortening an original URL and expanding a short URL in rcdn.info application server (error handling processes are omitted).

5.3 Procedures of Shortening and Expanding

The message sequences are common between rcdn.info and existing URL shorteners (Fig. 2), except that clients can be redirected to CoralCDN in rcdn.info. We describe only the internal procedures executed within the application server. We also explain the latency improvement techniques introduced in Sect. 4.2.

Shortening original URL. Figure 7(a) shows a procedure of URL shortening. Upon receiving a request for shortening, the application server validates the requested URL, e.g., whether it has a correct URL format beginning with `http://`. The server then searches its database for the re-

[†]We offer only Japanese pages on rcdn.info, because very few people do not speak Japanese and they are unlikely to cause flash crowds to disaster information (in general, majority is most likely to cause flash crowds).

quested URL. If the URL is not found, the server generates a unique key for the URL by using the SHA1 hash function (the hash value is incremented if conflicted with existing keys), and stores a tuple of the unique key, the original URL, and the Coralized URL, into the database. Finally, the server returns the short URL that consists of rcdn.info domain and the unique key.

In order to resolve the poor latency issue described in Sect. 4.2, the application server asynchronously makes a request for the Coralized URL. Since this request populates a DHT cache, a DNS cache, and a replica of the requested page in CoralCDN, subsequent requests experience less latency. This request should be issued *asynchronously* so as not to keep the client waiting long, while it should be issued *just after* receiving the shortening request in order to create caches and replicas as soon as possible.

Expanding short URL. Figure 7(b) shows a procedure of URL expanding. Upon receiving a request for expansion, the application server extracts the unique key from the short URL, and retrieves the corresponding tuple from the database (if the tuple is not found, 404 status code is returned). If the domain name in the original URL is found in the list of large capacity sites, the server returns the original URL for better latency; otherwise, the server returns the Coralized URL to mitigate flash crowds.

We show the URL expansion process without and with the asynchronous request in Fig. 8. The expansion process redirected to the original URL by the large capacity list has the same procedures with Fig. 2. The process without the asynchronous request (the first access we call) involves one more round-trip than that with the asynchronous request (the second access). The process to the original URL (the origin access) has the same number of round-trips to the second access, but the origin access is usually faster if the original site is much closer to clients than CoralCDN.

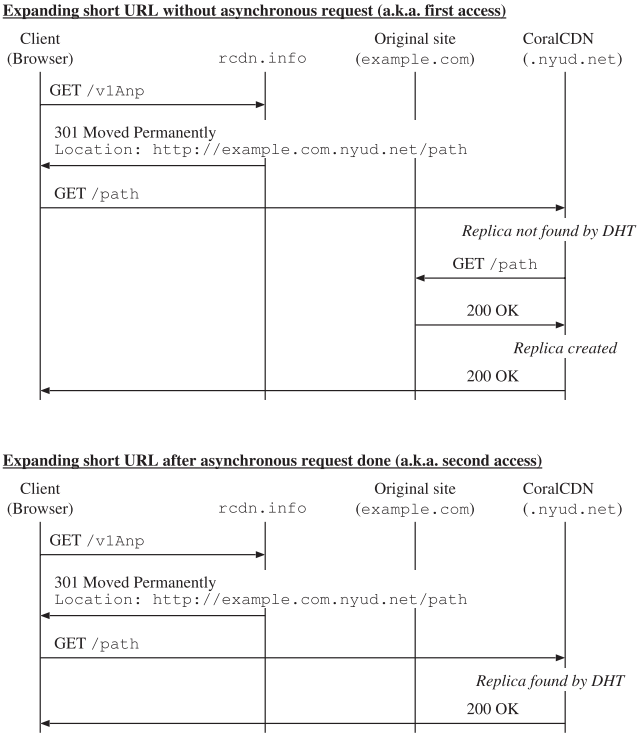

Fig. 8 Examples of expanding a short URL at rcdn.info, without and with the asynchronous request (DNS queries are omitted).

5.4 Implementation

We implemented rcdn.info on the LAMP (Linux, Apache, MySQL, Perl) stack [32]. The server machines were donated by Amazon Web Services [1]; Linux runs on an EC2 virtual machine (micro instance) in Tokyo region, and the "A" record of rcdn.info is maintained by a "Route 53" name server. The application server required 541 lines of Perl script. Since a unique key consists of six figures in base 62 (i.e., `0-9A-Za-z`), a short URL looks like `http://rcdn.info/EXnXN5`. The request log is monitored in real time (using "tail -f" command), and an asynchronous request is made just after a shortening request is found in the log.

We experienced no trouble related to the throughput of the server, since the procedures require no heavy operations; database access involves no complicated join operation, and the HTTP response has just a short URL for shortening API or "empty" body for redirection.

We did not give priority to detecting malicious contents, because CoralCDN maintains a global blacklist of specified domain names [24]. Our rcdn.info shows no sign of "redirecting to replica page" before taking users to CoralCDN, because there has been no serious problem in the history of CoralCDN as far as we know.

6. Log Analysis

This section analyzes HTTP requests logged at rcdn.info. Section 6.1 overviews the request log. Sections 6.2 to 6.5 answer questions, such as "what was rcdn.info used for", "who used it", "how it was used", and "from where was it accessed". We finally examine the user behavior from the two aspects tightly linked to flash crowds, namely URL popularity and request peaks, in Sects. 6.7 and 6.6.

6.1 Overview

We began developing rcdn.info on March 13th 2011, just two days after the earthquake. After the test operation on March 14th, we launched rcdn.info on March 15th. There were 24,959 HTTP requests at rcdn.info from March 15th 2:00 JST to 18th 22:23 JST (the log after March 18th was unfortunately lost due to mis-configuration in a rush job; remember that rcdn.info was not a well planned research project). Each log entry includes date and time, request method, path name, source IP address, user agent, referring page, and so on.

One of the authors, Takeru Inoue, announced the

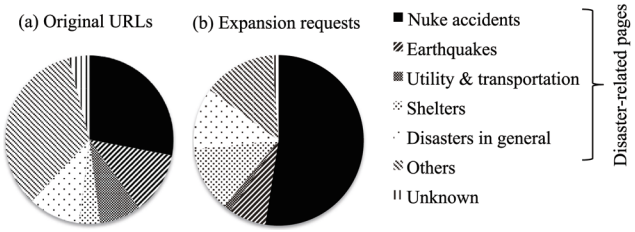

Fig. 9 Topics of top 50 popular URLs at rcdn.info, and those of expansion requests for these popular URLs.

launch of rcdn.info on Twitter, on March 15th. His Twitter account is `@takemaru_jp` and had 365 followers at that time. His followers mainly lived in the Tokyo metropolitan area, which includes Tokyo and the four neighboring prefectures, but the use of rcdn.info was not limited to the Tokyo area. Our rcdn.info appeared on several major online media like Yahoo! Japan [9] and livedoor [4][†] on March 16th. We confirmed that rcdn.info was accessed from all prefectures in Japan.

We examined the March log in June and July, 2011. First, source IP addresses were resolved into domain names. We then removed requests other than shortening and expanding, e.g., requests for the top page, images, style sheets, and so on. We also removed requests from automated agents, whose user agent name includes "bot", "crawler", "slurp", or "AppEngine-Google", or whose domain name ends with "amazonaws.com". Finally, we obtained 4,543 requests, which include 474 shortening requests and 4,069 expansion requests. The dataset is not that large, but its information is extremely valuable in capturing user behavior in the emergency. To understand the behavior well from the dataset, we provide several views in the following subsections. We think the dataset has no strong bias because the data was collected nation-wide, and samples of thousands give a good estimation to grasp a overall picture.

6.2 Topics

During the logging period, 299 unique URLs were shortened. We manually categorize the top 50 popular URLs (pages) based on their topics in Fig. 9(a). We also categorize the 3,582 expansion requests that were made for these top 50 URLs in Fig. 9(b). Since these requests cover 88.0% (3,582/4,069) of all requests, we can understand "what was rcdn.info used for" by examining just this data.

We first discuss Fig. 9(a). Our purpose in developing rcdn.info was to mitigate flash crowds for disaster information, but rcdn.info was used in a variety of ways that differed from our original purpose. While 31 pages in the top 50 included disaster information, 17 pages were related to other topics. We could not retrieve valid content from the two URLs in the "unknown" category.

As shown in Fig. 9(b), 85.1% of requests that were made for the top 50 were issued for disaster-related pages. As expected, "nuclear accidents" garnered a high level of interest, more than half of the requests. This pie chart shows that rcdn.info was mainly used to share disaster information.

Interestingly, both charts have different ratios. This difference might imply that people often want to spread what they do not want.

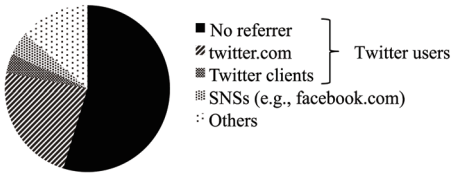

Fig. 10 Referring pages in expansion requests.

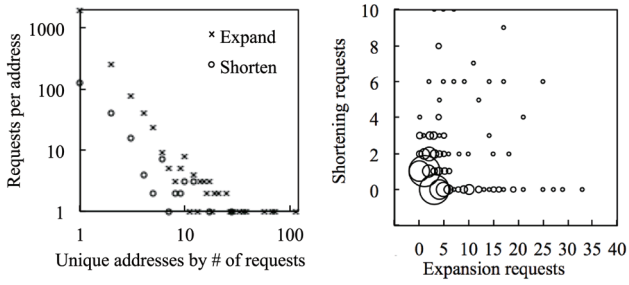

Fig. 11 Requests per unique IP address (left), and correlation between shortening and expansion requests by each address (right).

6.3 Referring Pages

Figure 10 shows the referring pages of the expansion requests. More than half of the requests have no referring page. We regard these "Referer-less" requests as issued by Twitter clients, since Twitter clients usually set no Referer header in the request. That is, 81.7% of expansion requests were made by Twitter users. Social networking services (SNSs) accounted for a small part of the requests, because their users do not need to shorten long URLs. We can say that rcdn.info was mostly used by Twitter users.

6.4 Request Types

We found 2,443 unique source IP addresses in the log. Figure 11 (left) shows a distribution of requests per unique address, which roughly follows a Zipf-like distribution for both shortening and expansion requests.

We next examine the correlation between the number of shortening requests and that of expansion requests for each address. Figure 11 (right) is a bubble chart of the correlation; circle size represents the number of addresses at that point. We see no clear correlation in the figure; some addresses issued many more shortening requests than expansion ones.

[†]Yahoo! Japan has been the most popular site in Japan for more than ten years, and livedoor occupied the 8th Alexa ranking in Japan as of July, 2011.

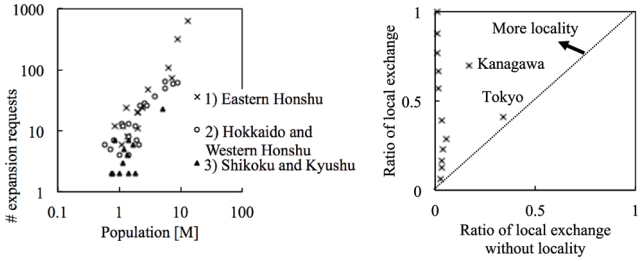

Fig. 12 Expansion requests versus population for each prefecture (left), and locality of information exchange (right).

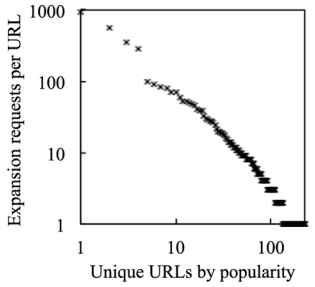

Fig. 13 Expansion requests per unique URL.

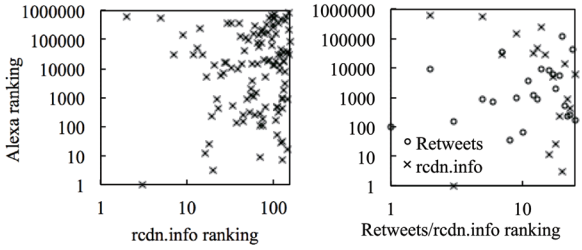

Fig. 14 Domain rankings between rcdn.info in the emergency and Alexa at ordinary times (left), and rankings between retweets and Alexa both at ordinary times (right).

6.5 Regionality

We classified source IP addresses into Japan's 47 prefectures, in order to examine the regionality of requests. We could classify 1,315 of 2,443 unique addresses, and we examined 2,108 requests that have those classified addresses.

Figure 12 (left) plots the number of requests against the population of each prefecture. We created three prefecture groups according to the distance from the epicenter, 1) Eastern Honshu, 2) Hokkaido and Western Honshu, and 3) Shikoku and Kyushu, as shown in Fig. 1. Comparing prefectures with similar populations, we see a weak trend in which prefectures closer to the epicenter made more requests in Fig. 12 (left). This trend is reasonable because people closer to the epicenter are more interested in the disaster and rcdn.info is mainly used to share disaster information as mentioned in Sect. 6.2.

We discuss the locality of information exchange among users by using Fig. 12 (right). The vertical axis is the ratio of "local exchange" measured at rcdn.info, while the horizontal axis is the same ratio with no locality assumption. We explain "local exchange" by using a mark of Kanagawa prefecture labeled in the figure. This mark means that URLs shortened by Kanagawa users were expanded by Kanagawa users with probability of 69.5% (see the vertical axis), while expansion requests made by Kanagawa users account just for 16.9% of total requests (see the horizontal axis). We see strong locality in terms of information exchange among Kanagawa users, since 69.5% is much greater than 16.9%. The strong locality is found in all prefectures other than Tokyo in the figure (the figure shows 13 prefectures that made more than 4 shortening requests). We consider that this strong locality come from local contents and/or user relations.

6.6 URL Popularity

Figure 13 plots the distribution of expansion requests per unique URL. The plot roughly follows a Zipf-like distribution, as is common among web caching and proxy networks [16]. Certain URLs are very popular, such as the most popular one received nearly 1 K expansion requests, while a large number of URLs received only a few requests (72 URLs received no expansion request).

We found 154 unique domains in the 299 original URLs that were shortened at rcdn.info. Figure 14 (left) compares domain rankings between rcdn.info in the middle of March and Alexa in June 2011 (i.e., Alexa at normal times)[†]. We see no clear correlation between these rankings. Surprisingly, 41 of 154 domains are not found even within the Alexa *1 M* ranking, and so they are not shown in the figure. This result means that massive requests were made for normally unpopular sites; these sites must have been less popular before the earthquake. We think this is the reason why many web sites went down in the aftermath of the earthquake.

For comparison, we examine the ranking of retweeted domains at ordinary times with the Alexa ranking. The retweeted domain ranking was calculated by the authors from a retweet dataset between June 20th and June 23rd, 2011, in Japan [6]. The top 25 domains in the retweet ranking have better Alexa ranks than those in rcdn.info, as shown in Fig. 14 (right). Moreover, only 2 of the top 25 retweeted domains are not found in the Alexa 1 M ranking, while 7 of the top 25 rcdn.info domains are not found in Alexa. We know that Alexa traffic ranks (bit/s) and retweet ranks (impressions) cannot be compared directly with our click ranks, but these results still support our conclusion that the request growth, which could be flash crowds, occurred at previously unpopular sites in the middle of March.

[†]The Alexa ranking can be different between March and June, but this gap does not upset our conclusion, because domains providing disaster information must have been even less popular before the earthquake.

Table 3 Top three popular URLs.

URL # of total requests	Title (originally in Japanese) # of requests to origin	Date/time of shortening
`http://earthsense.info/`	List of radiation levels	March 17th, 7:59pm
`http://www.atomin.go.jp/atomin/...`	Radiation and human body	March 15th, 11:53am
`http://maps.google.com/maps/...`	Shelters in Rikuzen-Takata City	March 16th, 7:10pm

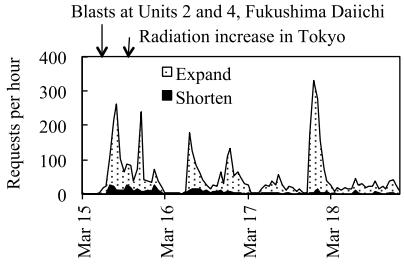

Fig. 15 Total requests per hour (stacked lines).

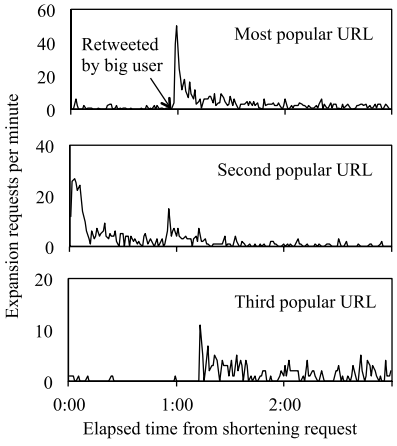

Fig. 16 Expansion requests per minute for the top three popular URLs.

6.7 Request Peaks

Our advantage over ordinary tweet analysis is that rcdn.info can count actual clicks with accurate time stamping, while conventional tweet analysis provides only impressions. Figure 15 shows the number of requests per hour. We see several sharp peaks as well as a periodic day-and-night pattern in the figure.

To investigate these peaks more closely, we examined request rates for the top three popular URLs at a finer timescale. Table 3 presents details of the top three URLs. Figure 16 shows the number of expansion requests per minute for these URLs. The horizontal axis is time elapsed since the shortening request was issued for each URL. The most popular URL was retweeted by an influential Twitter user (more than 20 K followers) about one hour after the shortening process was established. We see a sharp peak just after the retweet. The second popular URL was tweeted directly by another influential user (more than 100 K followers), and this tweet also triggered a sharp peak (this user also retweeted the fourth most popular URL which yielded a sharp peak). The third popular URL was retweeted by tens of users, but none of them had strong influence and we see no sharp peak. We found such drastic rate shifts for most of the popular URLs. These peaks grew on the order of minutes, not seconds or hours, as is common among flash crowds studied in CoralCDN [24]. The peaks are a bit small to cause server failures, but this mechanism (or the combination) could trigger larger peaks that hit some previously unpopular sites just after the earthquake.

7. Performance Evaluation

This section evaluates performance of rcdn.info through simulations using the log analyzed in Sect. 6. Section 7.1 investigates the request reduction effect that would be measured at origin servers by replaying the requests. Section 7.2 evaluates the latency increase, which is caused by rcdn.info as discussed in Sect. 4.2, assuming requests collected in the log. We do not discuss the throughput of rcdn.info, because we had no trouble about that and we can utilize good distributed data storage [21], [22], [31] if needed.

7.1 Request Reduction

We examine the number of requests that original sites received. We first count such requests for the top three URLs. As described in Sect. 2.3, CoralCDN makes a request for an original site if it has no replica or the replica is stale. Since we do not know the expiry time of each replica, we assume that CoralCDN updates replicas every five minutes (a conservative assumption). Figure 17(a) shows the number of requests that the original site would have received under the assumption. For the most popular URL, the original site received only 17.9% of requests (166/925); furthermore, it received just one of 144 requests within the busiest five minute period (Fig. 17(b)). We also see the great suppression of requests for the second popular URL, while the third URL's site received slightly more requests since it has a long-lasting request pattern as shown in Fig. 16. We calculated the total number of requests that *all* original sites received, assuming that rcdn.info redirected all requests to CoralCDN without the large site list. The result is that the original sites received only 32.5% of the requests made (1,321/4,069). That is, rcdn.info offers great potential to reduce the request traffic, and the reduced rate, a single request per five minutes, is considered not over the ordinary level for most sites. Web sites, however, may receive other traffic from users not using rcdn.info; we briefly discuss this issue in the conclusion.

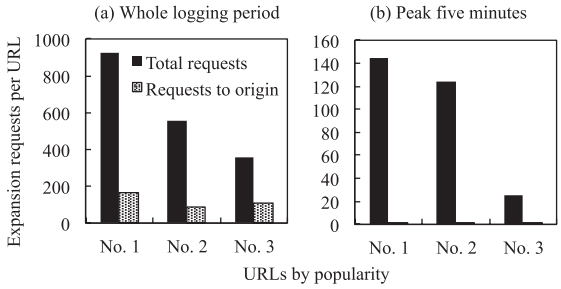

Fig. 17 Total requests and requests to origin (a) during the whole logging period and (b) in the peak five minutes for each URL.

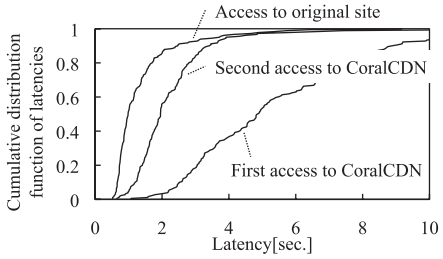

Fig. 18 Latencies for shortened URLs.

7.2 Latency Increase

We conducted experiments to evaluate the latency improvement techniques introduced in Sect. 5.3. The experiments were made at a host located in Japan, in a day in July 2011. Though some web sites might have rather different conditions from the emergency, we can still understand intrinsic impacts of the techniques. The latency and its variance could be larger in March due to heavy server loads and detoured networks, but the order relation among accesses would remain unchanged.

We first selected 181 rcdn.info short URLs whose original pages were unlikely to be replicated in CoralCDN at that time; i.e., they were not accessed within twenty-four hour period via rcdn.info, and the corresponding Coralized URLs were not found by Google search. The following process was then executed for each selected URL. We accessed the Coralized URL *twice* via rcdn.info, and compared the latencies between the first and second accesses (we assume that CoralCDN creates a replica with the first access, and simply returns the replica in the second access). We also accessed the original URL via rcdn.info and measured the latency. DNS cache was cleared at the experiment host before every access for fair comparison.

Figure 18 shows the latencies. The latencies are greatly improved between the first and second accesses (typical RTT between Japan and US is more than 120 msec.). This improvement is brought by the asynchronous access to the Coralized URL in the shortening procedure. The origin access has shorter latencies than the second access does. Users who access a large capacity site enjoy the better latency with the original site thanks to the large site list. Assuming the requests collected in our log, we calculated the average latencies weighted by the popularity of URLs; the averages were 6.30, 2.26, 1.48 seconds for the first, second, and origin accesses, respectively. The pages had weighted average size of 132 KB (excluding images and etc.). The additional latency of rcdn.info, that is the small gap between the second and origin accesses, can be considered as a necessary cost to prevent the long latency caused by flash crowds.

Since it takes some time to complete the asynchronous access, a user may suffer from the long latency if clicking an rcdn.info short URL before the completion. This, however, cannot be a serious problem in practice, because the asynchronous access usually takes less than ten seconds as shown in Fig. 18 (the asynchronous access includes the same operations with the first access).

8. Related Work

Flash crowd mitigation has been studied through the design and operation of CoralCDN. Reference [24] shows sample configurations for web servers to redirect flash crowds to CoralCDN, but this technique does not work in an emergency since it needs the support of site administrators. Other research papers focused on flash crowd mitigation [12], [18], [29], [37], but they also rely on the involvement of web sites.

Several research papers [13], [26], [28] studied anomalous traffic including flash crowds, but none of them focused on the traffic during an emergency; we have detailed the great shifts seen in site popularity.

The roles of indirection were extensively investigated in [35]. The review paper of CoralCDN [24] also discussed indirection in terms of its naming technique. To the best of our knowledge, no research work has addressed the layer of indirection constituted by URL shorteners, though shorteners have been widely used. This paper is the first work to discuss the indirection and to unveil the potential to flash crowd mitigation.

There are many studies that focus on network latency [17], [34], but no work has dealt with the latency caused by the combination of URL shorteners and CDNs, since this is a new challenge introduced by our idea.

Short URLs were recently examined in [11], which crawled many short URLs and analyzed their popularity. This work, however, could not investigate user behavior from various aspects, because crawled data includes no timestamp, no address, and no HTTP headers. We have presented various views to examine the behavior by analyzing the HTTP log collected at our URL shortener.

Several research papers studied information dissemination on Twitter. The dissemination is known to be quite fast; it takes less than thirty minutes for every hop [30]. Current news and events are likely to be retweeted [15]. Reference [33] also examined frequently tweeted topics. These findings are consistent with the fast development of flash crowds after a disaster.

9. Conclusions

This paper tackled the flash crowds seen in the aftermath of the Great East Japan Earthquake (March, 2011). We found that URL shorteners constitute a layer of indirection that is easily reconstructed even in the midst of an emergency. We developed a URL shortener named rcdn.info and confirmed that it can greatly mitigate flash crowds. Our solution is quite simple, but it is based on a deep insight into current Web usage. We also found request growth at previously unpopular sites during the disaster. Future work includes additional data analysis or more controlled experiments to compare users in an emergency with those in normal situations.

It is worth noting that rcdn.info is supposed to work with other CDNs if redirection URLs are given by the CDNs (or can be generated like CoralCDN), though the techniques for latency improvement strongly depends on positions of users, rcdn.info, and CDNs.

The more users rely on rcdn.info, the more flash crowds can be mitigated. However, rapid promotion is not easy in the emergency. One idea is that rcdn.info would be implemented in the existing shorteners stealthily in normal times, and it would be activated in the emergency.

Finally, we summarize lessons learned from the disaster. In terrible disasters, we should take into account that the traffic pattern will drastically change. We should develop a layer of indirection that is highly independent, in order to quickly control the traffic.

Acknowledgement

We would like to thank Prof. Michael Freedman for developing an excellent content distribution network. We also wish to acknowledge energetic support in software development by Dr. Norihito Yasuda and Mr. Masaaki Nishino. We would like to thank Dr. Tatsuya Mori for his valuable advice in the log analysis. We would like to thank Mr. Yuichi Yoshida for designing the pages of the shortener. We wish to acknowledge kind support about Amazon Web Services by Dr. Yasuhiro Araki, Ms. Miki Takata, and Mr. Keiichi Okabe.

References

[1] Amazon Web Services. http://aws.amazon.com/
[2] bitly-api. http://code.google.com/p/bitly-api
[3] Goo.gl challenges bit.ly as king of the short — NYTimes.com. http://bits.blogs.nytimes.com/2009/12/14/googl-challenges-bitly-as-king-of-the-short/
[4] livedoor news. http://news.livedoor.com/article/detail/5417888/ (in Japanese).
[5] Press Release — NEC Biglobe. http://www.biglobe.co.jp/press/2011/0427-1.html (in Japanese).
[6] Retweeter! http://retweeter.unicco.in/ (in Japanese).
[7] Tweetdeck. http://www.tweetdeck.com/
[8] We want 'em shorter. — MetaFilter. http://www.metafilter.com/8916/
[9] Yahoo! news. http://headlines.yahoo.co.jp/hl?a=20110316-00000003-rbb-sci (in Japanese).
[10] Yorufukurou. http://sites.google.com/site/yorufukurou/home-en
[11] D. Antoniades, I. Polakis, G. Kontaxis, E. Athanasopoulos, S. Ioannidis, E.P. Markatos, and T. Karagiannis, "we.b: The web of short URLs," ACM WWW, pp.715–724, 2011.
[12] I. Ari, B. Hong, E. Miller, S. Brandt, and D. Long, "Managing flash crowds on the Internet," IEEE/ACM MASCOTS, pp.246 – 249, 2003.
[13] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," ACM IMW, pp.69–73, 2001.
[14] T. Berners-Lee, R.T. Fielding, and L. Masinter, "Uniform resource identifier (URI): Generic syntax," IETF RFC 3986, 2005.
[15] D. Boyd, S. Golder, and G. Lotan, "Tweet, tweet, retweet: Conversational aspects of retweeting on Twitter," IEEE HICSS, pp.1–10, 2010.
[16] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," IEEE INFOCOM, vol.1, pp.126 –134, 1999.
[17] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," USENIX USITS, p.18, 1997.
[18] X. Chen and J. Heidemann, "Flash crowd mitigation via adaptive admission control based on application-level observations," ACM Trans. Internet Technol., vol.5, pp.532–569, 2005.
[19] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An overlay testbed for broad-coverage services," SIGCOMM Comput. Commun. Rev., vol.33, pp.3–12, 2003.
[20] B. Davison, A web caching primer, IEEE Internet Computing, vol.5, no.4, pp.38–45, 2001.
[21] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," Proc. twenty-first ACM SIGOPS symposium on Operating systems principles, pp.205–220, Oct. 2007.
[22] B. Fitzpatrick, "Distributed caching with memcached," Linux J., vol.2004, no.124, p.5, Aug. 2004.
[23] M. Freedman and D. Mazières, "Sloppy hashing and self-organizing clusters," Peer-to-Peer Systems II, LNCS, vol.2735, pp.45–55, Springer, 2003.
[24] M.J. Freedman, "Experiences with CoralCDN: A five-year operational view," USENIX NSDI, 2010.
[25] M.J. Freedman, E. Freudenthal, and D. Mazières, "Democratizing content publication with coral," USENIX NSDI, 2004.
[26] K. Ingham and H. Inoue, "Comparing anomaly detection techniques for HTTP," Recent Advances in Intrusion Detection, LNCS, vol.4637, pp.42–62, 2007.
[27] T. Inoue, F. Toriumi, Y. Shirai, and S.-i. Minato, "Great east Japan earthquake viewed from a URL shortener," Proc. Special Workshop on Internet and Disasters, SWID'11, pp.8:1–8:8, 2011.
[28] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," ACM CCS, pp.251–261, 2003.
[29] R. Kurebayashi, K. Obana, H. Uematsu, and O. Ishida, "A Web Access SHaping method to improve the performance of congested servers," IEICE/IEEE APSITT, pp.120–125, 2008.
[30] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?," ACM WWW, pp.591–600, 2010.
[31] A. Lakshman, P. Malik, and K. Ranganathan, "Cassandra: A structured storage system on a P2P network," Proc. 2008 ACM SIGMOD international conference on Management of data, Products Day #1, June 2008.
[32] J. Lee and B. Ware, Open source Web development with LAMP: using Linux, Apache, MySQL, Perl, and PHP, Addison-Wesley, 2002.
[33] J. Letierce, A. Passant, J. Breslin, and S. Decker, "Understanding how Twitter is used to spread scientific messages," WebSci, 2010.
[34] H.F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H.W. Lie, and C. Lilley, "Network performance effects of HTTP/1.1,

CSS1, and PNG," ACM SIGCOMM, pp.155–166, 1997.
[35] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, Internet indirection infrastructure, ACM SIGCOMM, pp.73–86, 2002.
[36] P. Wendell, J.W. Jiang, M.J. Freedman, and J. Rexford, "DONAR: Decentralized server selection for cloud services," ACM SIGCOMM, pp.231–242, 2010.
[37] X. Yang and G. de Veciana, "Service capacity of peer to peer networks," IEEE INFOCOM, vol.4, pp.2242–2252, 2004.

Takeru Inoue received the B.E., M.E., and Ph.D. degrees from Kyoto University, Kyoto, Japan, in 1998, 2000, and 2006, respectively. He joined NTT Laboratories in 2000. He is also a researcher at Japan Science and Technology agency. His research interest includes design and control of network systems. He received the best paper award from Asia-Pacific Conference on Communications in 2005. He also received the research awards of the IEICE Information Network Group in 2002 and 2005. He is a member of IEEE.

Shin-ichi Minato is a Professor at Graduate School of Information Science and Technology, Hokkaido University. He also serves as a Research Director of ERATO MINATO Discrete Structure Manipulation System Project, executed by JST. He received the B.E., M.E., and D.E. degrees in Information Science from Kyoto University in 1988, 1990, and 1995, respectively. He had been working for NTT Laboratories since 1990 until 2004. He was a Visiting Scholar at Computer Science Department of Stanford University in 1997. He joined Hokkaido University as an Associate Professor in 2004, and has been a Professor since Oct. 2010. He published "Binary Decision Diagrams and Applications for VLSI CAD" (Kluwer, 1995). He is a member of IEEE, IPSJ, and JSAI.