PAPER

# P2P-Based Approach to Finding Replica Server Locations for Alleviating Flash Crowds

**Masato ASAHARA**[†a)], ***Nonmember***, **Kenji KONO**[†], ***Member***, **Toshinori KOJIMA**[†], ***and*** **Ai HAYAKAWA**[†], ***Nonmembers***

**SUMMARY** Many services rely on the Internet to provide their customers with immediate access to information. To provide a stable service to a large number of customers, a service provider needs to monitor demand fluctuations and adjust the number and the location of replica servers around the world. Unfortunately, flash crowds make it quite difficult to determine good number and locations of replica servers because they must be repositioned very quickly to respond to rapidly changing demands. We are developing ExaPeer, an infrastructure for dynamically repositioning replica servers on the Internet on the basis of demand fluctuations. In this paper we introduce ExaPeer Server Reposition (EPSR), a mechanism that quickly finds appropriate number and locations of replica servers. EPSR is designed to be lightweight and responsive to flash crowds. EPSR enables us to position replica servers so that no server becomes overloaded. Even though no dedicated server collects global information such as the distribution of clients or the load of all servers over the Internet, the peer-to-peer approach enables EPSR to find number and locations of replica servers quickly enough to respond to flash crowds. Simulation results demonstrate that EPSR locates high-demand areas, estimates their scale correctly and determines appropriate number and locations of replica servers even if the demand for a service increases/decreases rapidly.

***key words:*** *replica server repositioning, distributed hash tables, network coordinates*

## 1. Introduction

The Internet has taken an important place in everyday life. Many services rely on it to provide their customers with immediate access to information. To provide a stable service to a large number of customers, a service provider usually positions *replica servers* (or mirror servers) around the world. Heretofore, service providers have analyzed access logs and statically positioned their servers to the areas where there has been a high level of demand. However, it is difficult to statically determine the number and the location of replica servers because the demand for a service often *fluctuates* drastically in a very *short term*. Flash crowds [1], [2] exemplify this. In flash crowds, a web server suddenly receives an avalanche of client accesses. Jung et al. [1] reported that the demand for a service increased five to ten times in a period of 40 seconds to 15 minutes.

We are developing *ExaPeer*, an infrastructure for dynamically repositioning replica servers on the basis of demand fluctuations. Conceptually, ExaPeer is similar to cloud computing [3]; ExaPeer consists of hundreds or thousands of trusted machines all over the Internet, which are lent to service providers to provide their services. Service providers run their replica servers on ExaPeer. Unlike the existing cloud computing environments (such as Amazon EC2 [4]), ExaPeer automatically positions replica servers in high-demand areas without any user support.

In this paper we introduce *ExaPeer Server Reposition (EPSR)*, a mechanism for dynamically and quickly selecting appropriate machines for replica servers. Even if the demand for a service fluctuates rapidly, EPSR automatically locates high-demand areas, estimates the demand scale, and selects appropriate machines in the high-demand areas for running replica servers. EPSR takes a P2P-based approach to quickly respond to rapidly increasing demands; it does *not* rely on global information. Instead, each machine participating in ExaPeer autonomously detects demand fluctuations and launches a replica server, if necessary, on the machine itself.

By taking a P2P-based approach, ExaPeer can reposition replica servers into high-demand areas to meet short-term demand fluctuations as well as long-term ones. Central server approaches [5]–[9] rely on global information, which must be collected from all over the Internet. This process of collecting global information hinders quick response to rapidly changing demands because the global information must be collected at least every 40 seconds to be responsive to flash crowds (as mentioned earlier, the demand for a service increased five to ten times in a period of 40 seconds to 15 minutes). P2P-based approaches enable us to respond to short-term demand fluctuations because they use only local information which can be collected quickly from adjacent machines. Unlike traditional P2P-based approaches [10]–[14], EPSR dynamically positions replica servers into high-demand areas and enables clients to get a service from a near replica server.

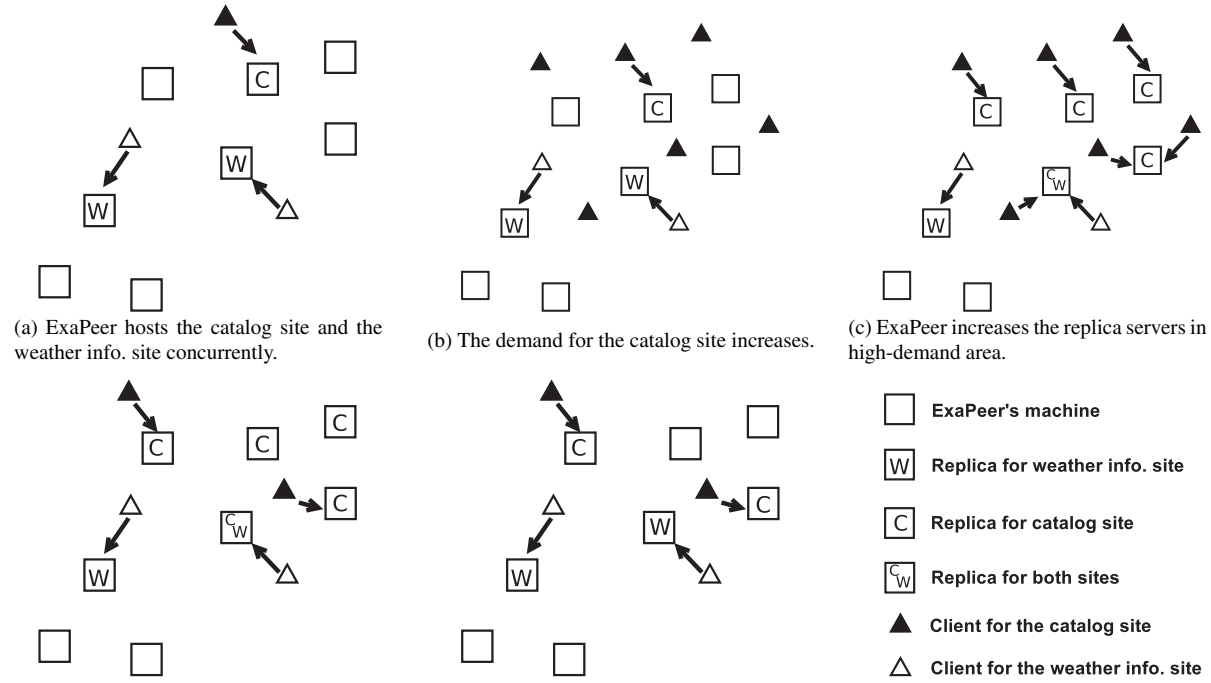
EPSR extends our previous mechanism [15], [16]. Our previous mechanism focuses on locating high-demand areas and lacks the ability of estimating the number of replica servers required to deal with flash crowds. As a result, the previous mechanism alone cannot alleviate flash crowds because it fails to launch the appropriate number of replica servers. EPSR estimates the number of required replica servers for each high-demand area and positions the appropriate number of them into the areas.

Simulation results demonstrate that EPSR is suitable for dealing with flash crowds. In our simulation, EPSR de-

Manuscript received May 29, 2009.
Manuscript revised July 17, 2010.

[†]The authors are with the Department of Information and Computer Science, Keio University, Yokohama-shi, 223–8522 Japan.
a) E-mail: exapeer@sslab.ics.keio.ac.jp

DOI: 10.1587/transinf.E93.D.3027




(a) ExaPeer hosts the catalog site and the weather info. site concurrently.

(b) The demand for the catalog site increases.

(c) ExaPeer increases the replica servers in the high-demand area.

(d) The demand for the catalog site decreases.

(e) ExaPeer stops the wasteful replica servers.

**Fig. 1** ExaPeer behavior. ExaPeer detects demand fluctuations and selects the machines in a high-demand area without any user support. In ExaPeer a client gets a demanded service from the near replica server.

tects demand increase and locates high-demand areas in 25 seconds. And then, it correctly estimates the number of required replica servers and positions the appropriate number of them into high-demand areas; *no* replica server becomes overloaded but all replica servers handle the moderate number of client accesses.

The rest of the paper is organized as follows. Section 2 overviews ExaPeer. Section 3 discusses EPSR design issues. Section 4 summarizes our previous method. Section 5 describes our proposal in this paper. Section 6 shows the simulation results. Section 7 describes related work, and Sect. 8 concludes the paper.

## 2. ExaPeer

The goal of ExaPeer is to allow service providers to provide their services continuously even if the demand for a service increases unpredictably and drastically like flash crowds. ExaPeer consists of hundreds or thousands of trusted machines on the Internet, and it runs replica servers for each service on the machines. To deal with unpredictable and drastic demand increase, ExaPeer dynamically and quickly repositions replica servers on the basis of demand fluctuations for each service.

Without ExaPeer, a service provider must *statically* determine the number and the location of replica servers. To position replica servers in a high-demand area, a service provider has to estimate the scale and the location of the current demand from the access logs of the running servers. But this estimation is ineffective because the demand on the Internet often fluctuates drastically in a short term; as described in Sect. 1, in flash crowds the number of clients suddenly increases five to ten times in several tens or hundreds of seconds. If the scale and the location of the actual demand is different from the estimated demand, replica servers do not work effectively. In the worst case replica servers become overloaded. To avoid becoming overloaded, a service provider positions superfluous replica servers, which costs the service provider.

To solve the problem of static positioning, ExaPeer *dynamically* repositions replica servers; without user analysis of the demand distribution, ExaPeer automatically and transparently repositions replica servers into current high-demand areas. ExaPeer is effective for flash crowds because it takes a *lightweight* approach. ExaPeer does not need to collect the load of all servers or the access pattern of all clients to reposition replica servers.

Figure 1 illustrates the behavior of ExaPeer. To use ExaPeer, a service provider registers its service with ExaPeer. In this example, ExaPeer hosts a catalog site and a weather information site concurrently (Fig. 1 (a)). When the demand for the catalog site increases (Fig. 1 (b)), ExaPeer selects more machines near the clients for the catalog site and runs replica servers on them (Fig. 1 (c)). A client gets the service from a near replica server directly, i.e., after a client finds the near replica server it establishes a direct connection to it. When the demand for the catalog site decreases (Fig. 1 (d)), ExaPeer immediately detects the decrease and stops unnecessary replica servers for the catalog site (Fig. 1 (e)).

In this paper we focus on *ExaPeer Server Reposition* (EPSR), a mechanism for adjusting the number and the location of replica servers to meet the demand. We are also developing other mechanisms of ExaPeer such as those for updating and keeping the consistency of the replicas [17]. These are out of the scope of this paper.

## 3. EPSR Design Issues

EPSR is a mechanism for finding replica server locations in ExaPeer to meet drastic as well as gradual demand fluctuations. To efficiently deal with drastic fluctuations such as flash crowds, EPSR must address the following issues.

*Lightweight Detection of Demand Fluctuations*:

To deal with short-term fluctuations, EPSR must respond quickly to demand fluctuations. Thus, EPSR must be lightweight even if the detection accuracy becomes slightly lowered; otherwise, a short-term fluctuation could not be detected. For example, we cannot take a centralized server approach. If we take this approach, the dedicated server must collect the global information of the current demand distribution. Global information of the demand consists of the client information derived from the access logs of the present replica servers. The size of the global information becomes swollen because the rate of client accesses is drastically increased in flash crowds. Since replica servers are distributed on the Internet, collecting the global information will take tens or hundreds of seconds. Thus, it is difficult to collect the global information frequently enough to identify a short-term fluctuation.

Conventional approaches to dynamic replica server reposition rely on global information of demands such as several kilobytes or megabytes of access logs from all over the Internet [5]–[9]. For example, if we want to detect flash crowds with Globule [7], a dedicated server has to collect client access logs from hundreds or thousands of Internet servers at least each tens seconds because Jung et al. [1] reported that in flash crowds a demand increases five times in a period of 40 seconds. Thus, it would not be a good idea for a centralized server to collect and analyze access logs from the entire network. Actually, several proposed mechanisms that intend to alleviate flash crowds [10]–[14] do not depend on global information of demands. We discuss the difference between EPSR and these proposed mechanisms [10]–[14] in Sect. 7.

*Topologically-aware Detection of High-demand Areas*:

To effectively position replica servers into high-demand areas, EPSR takes a physical network topology into consideration. By leveraging the topological knowledge, EPSR positions replica servers so that the network traffic can be reduced. If a replica server is placed on a machine closer to clients, the network traffic is reduced because the distance over which messages are relayed is shorter. In addition, if a replica server is placed where many access paths from clients cross, it can naturally host many requests without having to relay them to other hosts.

*Capacity- and Load-aware Adjustment of the Number of Replica Servers*:

To increase or decrease the number of replica servers, it is necessary to take the capacity and the current load of a machine into consideration. If the capacities of machines are lower or the load of machines are higher, the more machines should be selected for replica servers. If the capacities of machines are higher or the load of machines are lower, the fewer machines should be selected to reduce the running cost of replica servers.

To host multiple services concurrently, a machine should take the load of each replica server into consideration. If a machine is in overlapping high-demand areas for multiple services, it may host replica servers for each service concurrently. When a machine hosts multiple replica servers, it must control the load of each replica server so that it would not become overloaded.

*Transparent Access to Repositioned Servers*:

To quickly distribute the demand to repositioned replica servers, EPSR must enable a client to access transparently to a new replica server. If the demand for a certain service fluctuates during a short period, EPSR repositions a large number of replica servers within a short period. Therefore, EPSR cannot use the traditional DNS-RR (round-robin) for finding replica servers because the delay involved in updating the DNS database is too large to meet the time constraint. To ensure a transparent access to repositioned servers, ExaPeer must include a mechanism for quickly finding repositioned replica servers.

## 4. Locating High-Demand Areas

EPSR has three steps to find replica server locations for each service. First, EPSR constructs a topologically-aware overlay network. Second, to locate high-demand areas, EPSR selects *marker machine*s, which represent high-demand areas. Finally, to position replica servers within a high-demand area, EPSR selects candidate machines for replica servers around marker machines.

In this section we summarize the method of constructing a topologically-aware overlay network and locating a high-demand area, which is part of EPSR. The detail of this method is described in [15], [16].

### 4.1 Constructing a Topologically-Aware Overlay Network

To monitor the location of clients, EPSR constructs a topologically-aware overlay that combines GNP [18] and CAN [19]. GNP models the Internet as a $d$-dimensional space and assigns a $d$-dimensional coordinate to each machine so that the Euclidean distance between any pair of coordinates approximates the RTT between the machines. CAN, a well-known DHT, assigns each machine a virtual $d$-dimensional coordinate and divides the $d$-dimensional space into *zones*. A machine with a coordinate corresponding

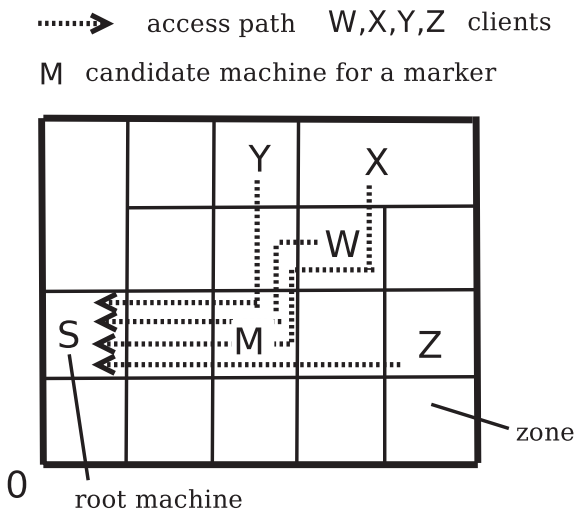



**Fig. 2** Locating high-demand areas on the basis of the degree of access path convergence (APC degree). *M* becomes a marker machine.

to zone *X* stores (key *K*, value *V*) pairs if the coordinate $(h_0(K), \cdots, h_{d-1}(K))$, where $h_i$ is a hash function, is inside zone *X*. If key *K* is given, CAN guarantees that a machine managing key *K* is reached in $(d/4)(N^{1/d})$ hops on average, where *N* denotes the number of machines in CAN.

To build a topologically-aware overlay, EPSR first calculates a GNP coordinate for each machine. The coordinate is used in turn to construct the CAN. Thus, adjacent machines on the constructed overlay are physically closer than other machines because the Euclidean distance between the coordinates approximates the distance in the physical network.

When a service is registered with ExaPeer, EPSR selects a machine, called a *root*, to host the service. The root machine is selected in accordance with the CAN protocol; a machine with a coordinate corresponding to the same zone to which the coordinate calculated from the service name corresponds is selected.

### 4.2 Locating High-Demand Areas

To locate a high-demand area, EPSR selects machines called *markers* which represent the locations of high-demand areas. If the machines around the markers host replica servers, the replica servers are within the high-demand areas.

To select marker machines, EPSR uses the degree of *access path convergence (APC)*. An *access path* is a network path on the overlay along which a query is relayed from a client to a root machine. The APC degree is the number of access paths that cross the machine of interest. Figure 2 shows an example of access paths and APC degrees (for convenience of explanation, a two-dimensional CAN is used). In this figure, ExaPeer hosts service *S*. Initially, EPSR selects machine *S* as a root machine. Client *W* sends out a query for service *S*, which is relayed according to the CAN protocol until it reaches the root. Clients *X*, *Y*, and *Z* also send out the queries. When a machine relays a query message for service *S*, it increments the APC degree for service *S*. As shown in Fig. 2, the APC degree for service *S* becomes four on machine *M*. To locate high-demand areas for each service, each machine maintains APC degree for each service, since ExaPeer hosts multiple services at the same time.

Each machine participating in ExaPeer determines autonomously whether to be a marker machine. If APC degree is increasing, a machine determines the demand around it is increasing and becomes a marker. ExaPeer can use statistical methods, e.g. trend estimation, to determine whether APC degree is increasing or decreasing. To simplify the evaluation of EPSR, the current prototype of ExaPeer uses a simple threshold method.

## 5. Selecting Candidates for Replica Servers

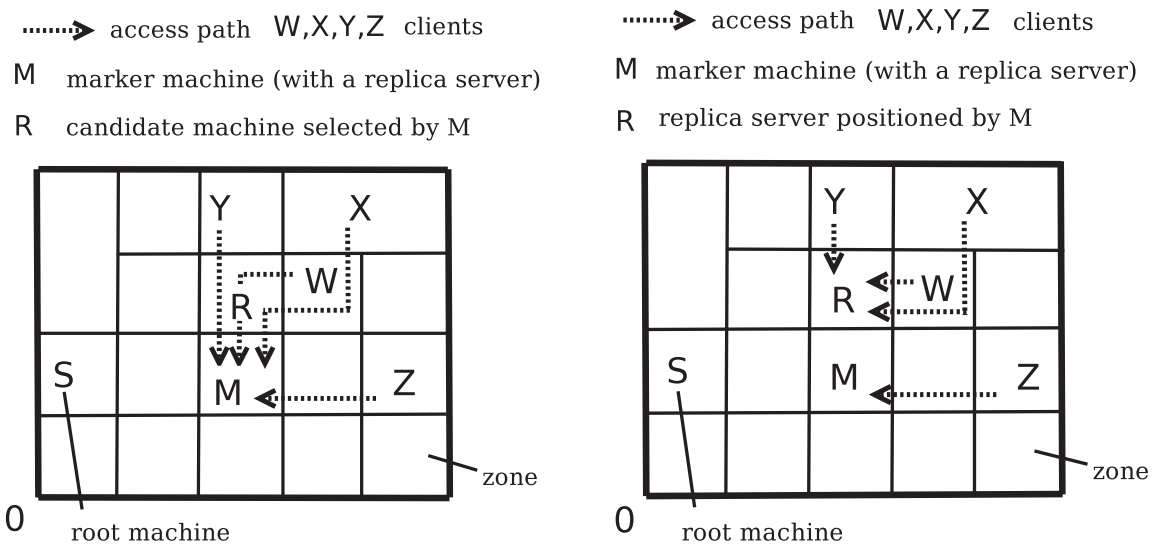
### 5.1 Basic Mechanism

A marker machine indicates that a high-demand area is around itself but does *not* suggest the *scale* of the demand; i.e., the number of required replica servers. If ExaPeer positions replica servers naively on every marker, the number of replica servers is not always sufficient to meet the demand because one marker is selected in each high-demand area regardless of the scale of the demand. In our simulation (the detail is described in Sect. 6), more than 25% of all machines become overloaded even if the adjacent machines are still lightly loaded.

The reason why this naive approach does not work well is that APC degree does not reflect the load and the capacity of a machine. The APC degree tends to increase if the machine is closer to a high-demand area. But it does not reflect the capacity of each machine. For example, if the capacity of a marker machine is substantial, the replica server on the marker can host all accesses from the near clients. If the capacity is not enough, the marker machine should launch other replica servers around itself.

To correctly adjust the number of replica servers to meet the current demand, we use *load indicator* which reflects the current load of a replica server. Load indicator indicates the current load of a replica server generated by its clients. The current prototype of ExaPeer uses the number of network connections as load indicator. The reason why we choose the number of network connections is that some commercial products (e.g., LocalDirector [20], BIG-IP [21]) employ the number of connections as load indicator. If you want to define load indicator more precisely, you can use other indicators as load indicator, e.g. CPU, memory or network bandwidth usages.

To alleviate flash crowds, EPSR takes an autonomous approach to adjusting the number of replica servers. EPSR allows a machine to autonomously determine whether to increase replica servers around itself by monitoring its load indicators. Since ExaPeer constructs a topologically-aware structured overlay, client accesses intend to be converged on the same machine if they are close to each other. Thus, if




(a) Machine *M* handles all client requests and becomes overloaded.

(b) Machine *M* requests machine *R*, whose APC degree is the highest among the neighbors of *M*, to host a replica server.

**Fig. 3** EPSR increases the number of replica servers on the basis of the load indicators and the APC degrees.

EPSR increases replica servers around a heavy-loaded machine, client accesses can be naturally distributed to them. EPSR increases replica servers by the following process. Initially, a marker machine hosts a replica server. When the load indicator of the replica server becomes higher, the machine requests its neighbor machine to launch a replica server. Then, the machine forwards excessive client queries to the neighbor replica server until its own load indicator becomes lower than its capacity.

EPSR ensures that no replica server becomes overloaded. As described above, a heavy-loaded machine requests its neighbor machine to launch a replica server. If the machine which received the launch request has no capacity, the machine forwards the message to one of its neighbor machines. To prevent the forwarded request from looping infinitely, the request is forwarded in the same direction from the machine who sent the launch request to the machine who received the request for the first time.

To allow a client to connect to a near replica server, EPSR starts a new replica server on the machine where many client queries converge. EPSR uses APC degree to select a machine on which a replica server is launched. Since a machine with higher APC degree relays more client queries than other neighbor machines, the replica server would host many clients.

Positioning a replica server on a high APC degree machine also allows a client to find a lightly-loaded replica server. Since a high APC degree machine is on many access paths from the clients, the replica server on such a machine is found by many clients.

Figure 3 illustrates an example where EPSR selects candidate machines for replica servers. *M* is a marker machine which represents the high-demand area consisting of clients *W*, *X*, *Y* and *Z*. Since *M* becomes overloaded if it deals with all the clients (*M*'s capacity is lower than three clients in this example), *M* selects a neighbor machine *R* whose APC degree is the highest of all the neighbor machines (Fig. 3 (a)). *R* hosts a replica server and services clients *W*, *X* and *Y* (Fig. 3 (b)). If the neighbor machine requested to boot a replica server is already heavy-loaded, it forwards the request to its neighbor. In Fig. 3 (b), if *R* has been already heavy-loaded, *R* forwards the request to a machine *W*.

To reduce superfluous replica servers, a machine stops a replica server if no client connects to the replica server during some period. To avoid the oscillation of replica server locations, a machine runs a replica server for a while even if no connection is established.

### 5.2 Improving EPSR Mechanism

Although the basic mechanism of EPSR selects appropriate candidate machines in many cases, it occasionally does not work well. There are three situations in which EPSR selects candidates inappropriately. In the following, we describe the techniques to alleviate these situations.

#### 5.2.1 Preference to High-Demanded Replica Servers

When a machine hosting multiple replica servers is heavy-loaded, it requests its neighbor to launch the most *lightly-loaded* replica server. Since EPSR allows a client to connect to a near replica server, a highly-loaded replica server would be closer to its clients than a lightly-loaded one. Thus, to make a highly-loaded replica server service more clients, EPSR increases a lightly-loaded replica server around the machine and allocates more machine resources to the highly-loaded replica server. In this way, lightly-loaded replica servers, i.e., replica servers which would be distant from a high-demand area, would be repositioned more

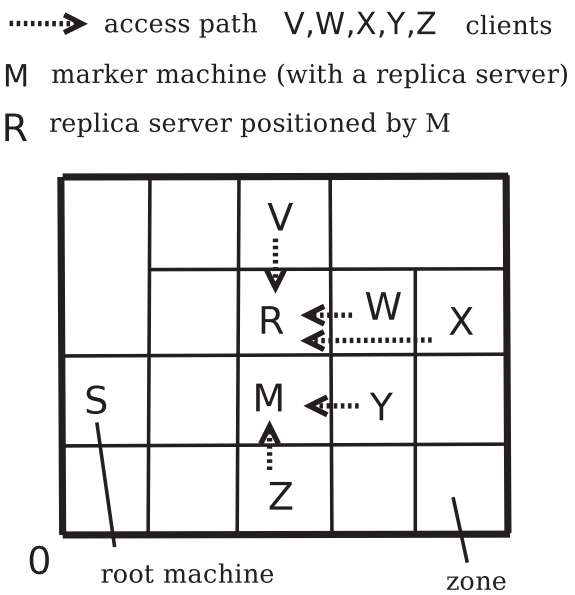



**Fig. 4** Without dummy queries, the number of replica servers will incorrectly decrease. Since *R* hosts clients *V*, *W* and *X* instead of the replica server on *M*, the APC degree of *M* becomes lower and *M* will stop the replica server on it. Then, the load of replica server *R* becomes overloaded if clients *Y* and *Z* connect to *R*.

closer to a high-demand area.

#### 5.2.2 Dummy Query

EPSR increases replica servers around a heavy-loaded replica server and distributes client queries to them. After clients are distributed to the replica servers, the loads of the replica servers decrease. In this situation, a machine may misjudge that its replica server becomes unnecessary because the load indicator of the replica server decreases. As shown in Fig. 4, replica server *R* is launched by marker machine *M* to offload the load of the replica server on *M*. As shown in the figure, *M* handles the accesses from clients *Y* and *Z* after launching *R*. In this situation, *M* will stop the replica server since the APC degree of *M* is low. This is not desirable because *M* is still near the high-demand area.

To solve this problem, when a machine hosting replica servers receives a client request, it sends the marker a *dummy query*, which only contains the coordinate of the service. A dummy query is forwarded on the basis of CAN protocol until it reaches a marker to increment the APC degrees along the access path. As a result, EPSR keeps replica servers alive in a high-demand area. Dummy query is designed so that it increases the load of a machine as less as possible; a dummy query is small and only increments APC degrees.

#### 5.2.3 Dummy Connection

If the enormous number of clients send queries to the same service simultaneously, a machine mistakenly accepts those queries to become overloaded. This is because there is some delay in the load of a replica server being raised. When a client sends a query to a replica server, the load of the replica server is not raised until a network connection is established. If many clients send queries at the same time, the replica server accepts those queries because the current capacity appears to be sufficient to accept the clients. If those clients do establish network connections, the machine becomes overloaded.

To solve this problem, EPSR adds a *dummy connection* to a replica server when the replica server accepts a new client query. Dummy connection adds virtual load to a replica server, and the load of the replica server is raised virtually. Thus, the machine correctly determines whether to accept subsequent client queries. Dummy connection is removed after the client established its connection to the replica server.

## 6. Experiments

### 6.1 Evaluation Methodology

To evaluate the effectiveness of EPSR, we built an ExaPeer prototype on Overlay Weaver [22], a toolkit that enables a structured overlay network to be easily constructed and emulated. We tested EPSR under different load fluctuation conditions and measured three key indicators.

- *Number of running replica servers*, which shows how well EPSR selects candidate machines for replica servers on the basis of client demand. If the number of replica servers changes with fluctuations in demand, we can say that EPSR increases or decreases candidates for replica servers on the basis of demand fluctuation.
- *RTT between an accessing client and the accessed replica server*, which shows how well EPSR selects candidate machines for replica servers near high-demand areas. The lower the RTT is, the physically closer the machines are to the high-demand areas.
- *Number of network connections*, which shows the load of a machine. If the number of network connections is below machine's capacity, EPSR increases the number of replica servers to meet the demand. If the minimum number of network connections is not too low, there are no wasteful servers.

To confirm the behavior of EPSR in a realistic environment, we emulate the Internet environment by introducing Internet topology. In our experiments, we use Transit-Stub (TS) model of the Internet that has a two-level hierarchy of routing domains, i.e., transit domains interconnecting lower-level stub domains [23]. To generate a transit-stub topology, we use the GT-ITM topology generator. To generate the topology, we assign the parameters as follows: 228 transit domains, 5 transit nodes per transit domain, 4 stub domains attached to each transit node, and 2 nodes in each stub domain. We select about 3000 nodes with unique GNP coordinates from about 10,000 nodes. EPSR checks the APC

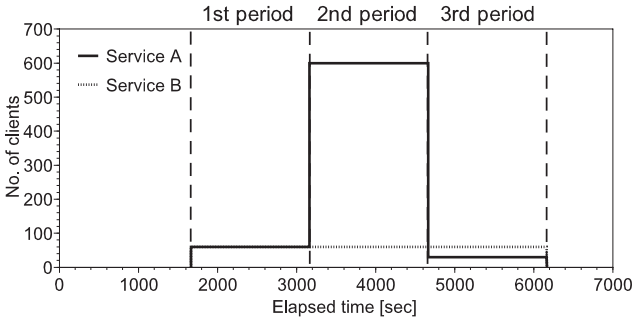



**Fig. 5** The number of clients on service *A* and *B*.

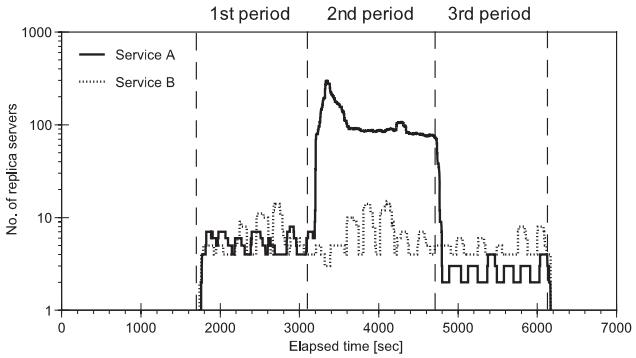



**Fig. 6** The number of replica servers positioned by EPSR when ExaPeer provides two services at the same time.

degrees and the load indicators of a machine every 30 seconds. These parameters are derived from the fact that the access rate increased in 40 seconds to 15 minutes in flash crowds [1].

### 6.2 Simulation Results

EPSR locates high-demand areas and adjusts the number of replica servers to meet the demand even if the demand fluctuates. To confirm this feature, we simulate a localized, high-demand area, in which the RTTs between the clients are less than 4000 msec. A client sends a query to get a service every one minute. To confirm that EPSR can support multiple services simultaneously, we set ExaPeer to host two services *A* and *B* at the same time. The demand for service *A* fluctuates but the demand for service *B* is constant, as shown in Fig. 5. In this simulation, a machine becomes a marker if APC degree is more than 300 and a machine ceases to be a marker if APC degree is less than 200. We set the size of the sliding window for APC degree to 300 seconds. We assume that all machine can accept less than 30 connections, i.e., the initial capacity of each machine is 30 connections. We compare EPSR with the following three strategies.

- *Root_only*, which is only the root machine services all the queries.
- *Marker_only (previous method)*, which launches replica servers only on marker machines. Note that marker_only is our previous method [15], [16].
- *Globule*, which selects machines with Globule method [7], [24], [25]. Globule relies on global information such as access logs of all servers. This global information enables Globule to select nearly optimal machines for replica servers. If RTTs with EPSR are close to those with Globule, EPSR positions replica servers at the nearly optimal location. Due to the cost of collecting the global information and its centralized approach, Globule is not suitable for responding to rapid fluctuations in demand like flash crowds. Since our simulation generates rapidly-fluctuating demands, we calculate the locations for replica servers in off-line.

#### 6.2.1 Number of Running Replica Servers

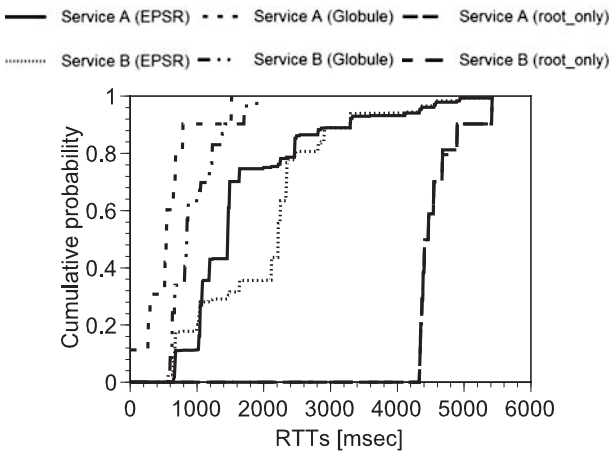
Figure 6 shows the number of replica servers. The x-axis is elapsed time and the y-axis is the number of replica servers in logarithm scale. This result represents that EPSR increases or decreases the number of replica servers on the basis of demand fluctuations. The number of replica servers for service *A* fluctuates with the demands for the service increased or decreased. In the second period of this simulation, the number of clients increases ten times larger than that of the first period as shown in Fig. 5. To deal with this heavy demand, EPSR begins to increase the number of replica servers after 25 seconds. Then, EPSR have increased the number of replica servers for service *A* from 7 to 299 in 166 seconds. After about 300 seconds passed, EPSR stops lightly-loaded replica servers and the number of replica servers decreases to 90. In the third period, the number of replica servers for service *A* is lower than that for service *B*. This is because the number of clients for service *A* is lower than that for service *B* during the third period.

From Fig. 6, we can see that EPSR adjusts the number of replica servers for each service. When the number of clients on service *A* increases from 60 to 600 (from the first to the second period), the number of replica servers for service *A* also increases from 7 to 299. In the second period, the number of replica servers for service *B* is slightly larger than that in the other periods. This is because the machines hosting the replica servers for service *B* becomes highly-loaded due to the increased demand for service *A*; when a machine hosting two or more services has a smaller capacity, it requests its adjacent machines to launch a replica server.

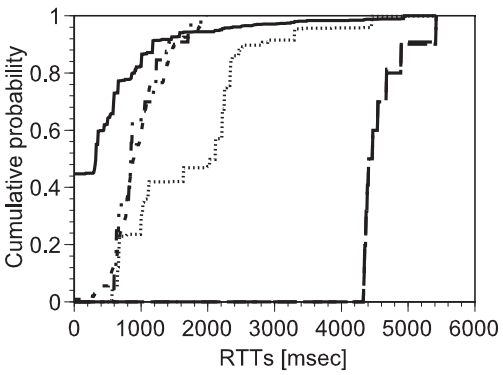
#### 6.2.2 RTT between an Accessing Client and the Accessed Replica Server

To confirm that EPSR positions replica servers near a high-demand area, we compare the RTTs with root_only, Globule and EPSR. Figure 7 shows the cumulative probabilities of RTTs. Figure 7 (a), (b) and (c) show the RTTs in the first, second and third period, respectively.

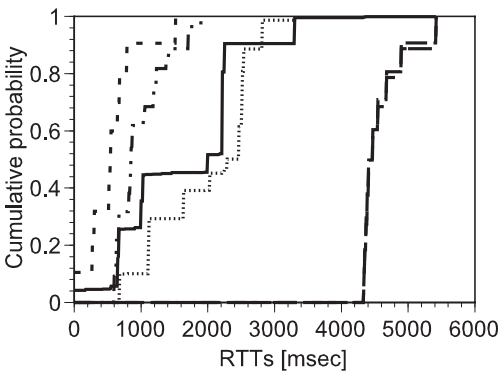
EPSR increases replica servers near the high-demand area under flash crowds. In the second period, the RTTs




(a) 1st period




(b) 2nd period




(c) 3rd period

**Fig. 7** The cumulative probability of RTTs when ExaPeer provides two services at the same time.

for service *A* with EPSR are better than those with Globule. With EPSR, the RTTs for 90% of all client accesses are less than 1180 msec, while those with Globule are less than 1500 msec. This is because EPSR increases the number of replica servers in high-demand areas so that no machine becomes overloaded. With EPSR, since many machines in the high-demand area launch replica servers for service *A*, 44% of all client accesses are handled by the machine nearest the client. With Globule, the RTTs for service *A* are worse than those in other periods. The RTTs for 90% of all accesses for service *A* are less than 1500 msec, while those in the first period are less than 790 msec. This is because Globule forwards some client queries to the machines slightly far from the high-demand area when the machine is overloaded.

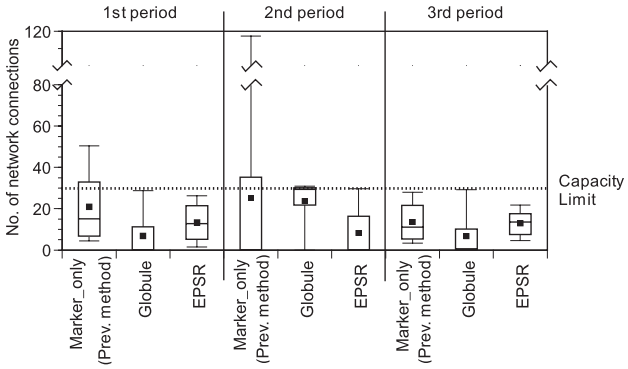



**Fig. 8** The distributions of the number of network connections on machines when ExaPeer provides two services at the same time.

EPSR also positions replica servers near the high-demand area under the normal conditions. In the first period, with EPSR, the RTTs for 90% of all client accesses for service *A* and *B* are less than 3300 msec, while those with root_only are less than 4900 msec. In the third period, with EPSR, those for service *A* and *B* are less than 2250 msec and 2540 msec, respectively. On the other hand, with root_only, those for service *A* and *B* are less than 4900 msec. With Globule, the RTTs for 90% of all client accesses are about quarter of those with EPSR. This is because Globule positions replica servers at nearly optimal locations with global information.

### 6.2.3 Number of Network Connections

To confirm that EPSR prevents a machine from becoming overloaded, we compare the distributions of the number of network connections. Figure 8 shows the box-and-whisker plots of the number of network connections during each period. As shown in Fig. 8, EPSR increases the number of replica servers so that no machine becomes overloaded; the number of network connections of all machines is below the capacity limit 30. On the other hand, marker_only (prev. method) fails to increase them. In particular, the number of overloaded machines increases with the number of clients increased. In the second period of the simulation (the heaviest workload), more than 25% of the machines exceed their capacity limits.

EPSR also reduces wasteful replica servers. With EPSR, in the first and the third period, the number of network connections are more than zero. This is because EPSR stops the replica server with no connection when the scale of demand is low. In the second period, the number of lightly-loaded replica servers increases. This is because EPSR positions more replica servers not to fail under flash crowds. On the other hand, Globule makes many wasteful replica servers in the first and the third period. This is because Globule positions replica servers to meet the highest scale of demand in the second period.

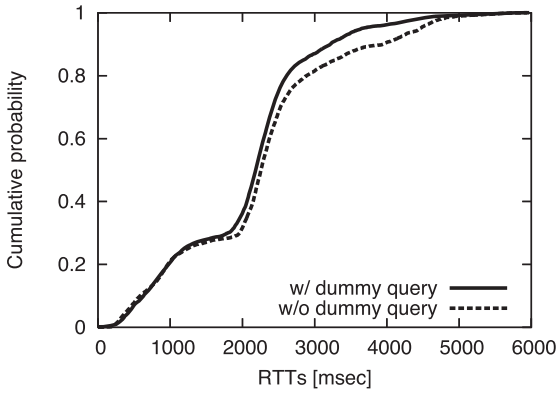



**Fig. 9** The cumulative probability of RTTs with and without dummy queries.

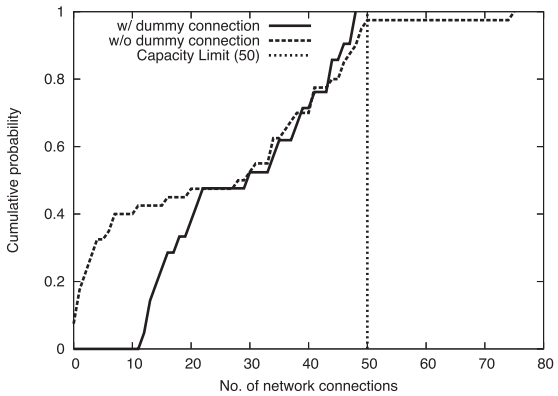



**Fig. 10** The cumulative probability of network connections of machines with and without dummy connections. Note that the machines with no replica server are uncounted.

### 6.2.4 Effectiveness of Dummy Queries/Connections

To confirm that dummy query and dummy connection contribute to selecting machines to launch replica servers, we compared the performance of EPSR with and without these techniques. Since dummy query and dummy connection are effective in heavier conditions, we change some parameters of the simulation. In this simulation, we run 8 services concurrently. A machine becomes a marker if APC degree is more than 300 and a machine ceases to be a marker if APC degree is less than 150. The initial capacity of each machine is 50 connections.

As described in Sect. 5.2.2, dummy queries contribute to keeping replica servers in a high-demand area and avoids placing replica servers outside the high-demand area. This is because dummy queries prevent a machine from misjudging the demand to be lowered around it. Figure 9 shows the cumulative probabilities of RTTs with and without dummy queries (dummy connection is turned on in both cases). For 90% of all client accesses, the RTTs with dummy queries were less than 3220 msec, while those without dummy queries were less than 3950 msec. This result demonstrates dummy query contributes to avoiding placing replica servers distant from clients.

Dummy connections contribute to keeping the number of network connections under the capacity limit of each machine, because they prevent a machine from underestimating the number of network connections in a rush of client accesses. Figure 10 shows the cumulative probabilities of network connections with and without dummy connections (dummy query is turned on in both cases). With dummy connections, EPSR keeps the network connections of each machine under the capacity limit (50 network connections). Without dummy connections, 2.5% of all machines received client requests more than their capacity limits.

Figure 10 also shows that dummy connection reduces the number of replica servers that are processing the small number of connections. Without dummy connections, 40% of all machines handled less than 7 connections, while with dummy connections all machines handled more than 10 connections. This is another effect of dummy connection. When there is a rush in client queries, dummy connections help the queries to be distributed uniformly among replica servers. With dummy connections, the machines request replica servers to be launched before handling the rush of client queries. When the queries actually arrive, they are distributed among replica servers that have been just launched. On the other hand, if dummy connection is not used, most of the queries are handled by overloaded machines and some remaining ones are handled by newly launched servers. This is because replica servers are launched after some machines start handling overwhelming client queries.

## 7. Related Work

Various types of replica placement strategies have been proposed. Many researchers are developing algorithms for web document placement. The algorithms are classified into two types, static and dynamic. Khan and Ahmad [26] compared ten static heuristics-based replication techniques. These techniques statically decide the locations of website contents so as to minimize the average access time perceived by end users. These static positioning algorithms do not consider repositioning replica servers dynamically.

Dynamic algorithms for replica repositioning are also developed. Some dynamic repositioning algorithms are based on simple greedy algorithms. The HotSpot algorithm proposed in [6] positions replicas with high accuracy even though it collects imperfect information about client workload characteristic. Tse [9] proposed the repositioning algorithm which reduces the computing cost to $O(\log N)$, where $N$ denotes the number of machines. Globule [24], [25] with HotZone algorithm [7] enables us to dynamically position replicas so that the latency between a client and the replica is minimized.

These dynamic repositioning algorithms estimate the number and the location of replicas with high accuracy. However, they are not suitable for dealing with flash crowds because they do not address one of the EPSR design issues, lightweight detection of demand fluctuations; they depend

on global information of client demands all over the Internet. For example, Globule collects demand information such as the access frequency and the location of clients from the logs of all the Globule servers all over the Internet. Globule assumes that these pieces of information are collected every several hours.

There are several replica placement strategies with peer-to-peer techniques to alleviate the symptoms associated with flash crowds. However, these strategies do not address all EPSR design issues described in Sect. 3. Backslash [10] makes a replica of the web content on pre-placed machines. Backslash positions a replica along the client access paths on its structured overlay network. Unlike EPSR, Backslash does not consider client locations on the physical network; clients often connect to distant machines to get a replica. In addition, Backslash does not consider machine's capacity when positioning a replica.

Replica Enumeration (RE) [14] enables us to distribute client accesses uniformly to replicas in flash crowds. With RE, a client gets the replica from the closest machine. RE modifies a look-up algorithm of a distributed hash table to consider the distance between a replica and a client. However, RE does not automatically adjust the number of replicas on the basis of demand fluctuations. Thus, unlike EPSR, RE cannot detect flash crowds and reposition replicas to alleviate the flash crowds.

FCAN [11] takes a hybrid approach that combines client-server and peer-to-peer architectures. Once a flash crowd occurs, the original server requests cache proxy servers to form a P2P overlay and to process all the requests instead of the original server. In flash crowds, a cache proxy server finds a replica through the P2P overlay and serves the replica to a client. Unlike EPSR, FCAN does not increase replicas in a topologically-aware fashion.

To deal with short-term fluctuations of demands, most Content Distribution Networks (CDNs) take the cache-based approach. Akamai [27] and other commercial CDNs use DNS redirection to reroute client requests to local clusters of machines by building detailed maps of the Internet through a combination of BGP feeds and their own measurements. When a client accesses to the cluster, these systems cache a content on the cluster from the original server. Thus, these systems do not adjust the location of replicas. EPSR is different from these systems in that it locates a high-demand area and proactively adjusts the number and the location of replica servers.

CoralCDN [13] is a peer-to-peer content distribution network. CoralCDN relies on a hierarchical structure, called "distributed sloppy hashtables," to reduce the load on web servers. In CoralCDN, the node relays client queries to one of the nodes which has the cache of the requested content. If client queries are relayed more than the predefined threshold, CoralCDN concludes a high-demand area is near and generates a new cache of the content. CoralCDN does not consider machine capacity when creating a cache or routing a client query, while EPSR takes machine capacity into account when determining the location of replicas and routing a client query.

Flashback [12] addresses flash crowds by distributing the cache of the web contents on clients and sharing them between clients. With Flashback, a client gets the requested content from other clients instead of the original server. Thus, the load of the server is lowered even with flash crowds. EPSR is different from Flashback in two points. First, Flashback does not take the location of the clients into account. Second, it does not consider machine capacities when the caches are distributed.

Freenet [28] and Winny [29] are P2P file sharing systems based on dynamic cache location. These systems are completely different from EPSR because they construct overlay networks independent of the underlying physical network. Although Freenet and Winny position a cache on a machine close to a client on the overlay networks, they do not always position a cache on a machine close to a client on the underlying physical network. Unlike them, EPSR positions replica servers on a machine close to a client on the underlying physical network as well as on the overlay network. This is because EPSR constructs a topologically-aware overlay.

## 8. Conclusion

In this paper we introduce EPSR, which dynamically and quickly selects candidate machines for replica servers on the basis of demand fluctuations. EPSR dynamically locates a high-demand area in which replica servers should be placed, estimates the scale of the demand, and adjusts the number of replica servers to meet the demand. To deal with short-term demand fluctuations such as flash crowds, EPSR takes a peer-to-peer approach which enables EPSR to immediately locate a high-demand area with low data traffic. To avoid overloading replica servers, EPSR adjusts the number and the location of replica servers on the basis of the scale of each high-demand area. EPSR introduces two heuristics: APC degree and load indicator. APC degree enables each machine to locate a high-demand area autonomously. Load indicator enables each machine to determine whether to increase replica servers in the high-demand area.

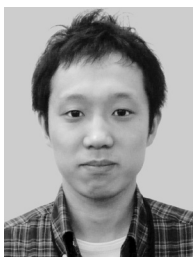
Simulation results demonstrate that EPSR positions replica servers in a high-demand area. The results also show that EPSR increases or decreases the number of replica servers on the basis of demand fluctuations. With EPSR, all of the machines do not become overloaded. For future work, we plan to evaluate EPSR on the Internet using PlanetLab [30].

### References

[1] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites," Proc. ACM Int'l World Wide Web Conf., pp.293–304, 2002.

[2] S. Lorenz, "Is your web site ready for the flash crowd?," 2000. http://www.serverworldmagazine.com/sunserver/2000/11/flash.shtml

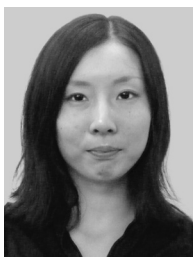
[3] A. Weiss, "Computing in the clouds," ACM netWorker, vol.11, no.4, pp.16–25, 2007.
[4] Amazon.com, "Amazon Elastic Compute Cloud," http://aws.amazon.com/ec2
[5] M. Karlsson and C. Karamanolis, "Choosing replica placement heuristics for wide-area systems," Proc. IEEE Int'l Conf. Distributed Computing Systems, pp.350–359, 2004.
[6] L. Qiu, V.N. Padmanabhan, and G.M. Voelker, "On the placement of web server replicas," Proc. IEEE INFOCOM, pp.1587–1596, 2001.
[7] M. Szymaniak, G. Pierre, and M. van Steen, "Latency-driven replica placement," Proc. IEEE/IPSJ Symp. Applications and the Internet, pp.399–405, 2005.
[8] X. Tang and J. Xu, "QoS-aware replica placement for content distribution," IEEE Trans. Parallel Distrib. Syst., vol.16, no.10, pp.921–932, 2005.
[9] S.S.H. Tse, "Approximate algorithms for document placement in distributed web servers," IEEE Trans. Parallel Distrib. Syst., vol.16, no.6, pp.489–496, 2005.
[10] T. Stading, P. Maniatis, and M. Baker, "Peer-to-peer caching schemes to address flash crowds," Proc. Int'l Workshop on Peer-To-Peer Systems, pp.203–213, 2002.
[11] C. Pan, M. Atajanov, M.B. Hossain, T. Shimokawa, and N. Yoshida, "FCAN: Flash crowds alleviation network using adaptive P2P overlay of cache proxies," IEICE Trans. Commun., vol.E89-B, no.4, pp.1119–1126, April 2006.
[12] M. Deshpande, A. Amit, M. Chang, N. Venkatasubramanian, and S. Mehrotra, "Flashback: A peer-to-peerweb server for flash crowds," Proc. IEEE Int'l Conf. Distributed Computing Systems, 2007.
[13] M.J. Freedman, E. Freudenthal, and D. Mazières, "Democratizing content publication with coral," Proc. USENIX Symp. NSDI, pp.239–252, 2004.
[14] M. Waldvogel, P. Hurley, and D. Bauer, "Dynamic replica management in distributed hash tables," Research Report RZ–3502, IBM, 2003.
[15] M. Asahara, A. Shimada, H. Yamada, and K. Kono, "Finding candidate spots for replica servers based on demand fluctuation," Proc. IEEE Int'l Conf. Parallel and Distributed Systems, 2007.
[16] M. Asahara, A. Shimada, H. Yamada, and K. Kono, "Strategy for selecting replica server spots on the basis of demand fluctuation," IPSJ Trans. Advanced Computing Systems, vol.1, no.1, pp.160–173, 2008.
[17] A. Hayakawa, M. Asahara, K. Kono, and T. Kojima, "Efficient consistency algorithm by speculating the location of the contents on peer-to-peer distributed servers," IPSJ Tech. Report (2008-OS-109), pp.125–132, 2008.
[18] T.S. Eugene Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," Proc. IEEE INFOCOM, pp.170–179, 2002.
[19] S. Ratnasamy, P. Francis, M. Handley, R.M. Karp, and S. Shenker, "A scalable content-addressable network," Proc. ACM SIGCOMM, pp.161–172, 2001.
[20] Cisco Systems, 2002. http://www.cisco.com/
[21] F5 Networks, 2002. http://www.f5labs.com/
[22] K. Shudo, Y. Tanaka, and S. Sekiguchi, "Overlay weaver: An overlay construction toolkit," IPSJ Trans. Advanced Computing Systems, vol.46, no.SIG 4 (ACS 9), pp.33–44, 2005.
[23] E.W. Zegura, K.L. Calvert, and S. Bhattacharjee, "How to model an internetwork," Proc. IEEE INFOCOM, pp.594–602, 1996.
[24] G. Pierre and M. van Steen, "Globule: A platform for self-replicating web documents," Proc. Protocols for Multimedia Systems, Lect. Notes Comput. Sci., vol.2213, pp.1–11, Springer, 2001.
[25] G. Pierre and M. van Steen, "Globule: A collaborative content delivery network.," IEEE Commun. Mag., vol.44, no.8, pp.127–133, 2006.
[26] S.U. Khan and I. Ahmad, "Comparison and analysis of ten static heuristics-based Internet data replication techniques," J. Parallel Distrib. Comput., vol.68, no.2, pp.113–136, 2008.
[27] J. Dilley, B.M. Maggs, J. Parikh, H. Prokop, R.K. Sitaraman, and W.E. Weihl, "Globally distributed content delivery," IEEE Internet Comput., vol.6, no.5, pp.50–58, 2002.
[28] I. Clarke, "A distributed decentralised information storage and retrieval system," tech. rep., University of Edinburgh, 1999.
[29] I. Kaneko, The Technology of Winny, ASCII CORPORATION, 2005.
[30] A.C. Bavier, M. Bowman, B.N. Chun, D.E. Culler, S. Karlin, S. Muir, L.L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating systems support for planetary-scale network services," Proc. USENIX Symp. Networked Systems Design and Implementation, pp.253–266, 2004.



**Masato Asahara** received his B.E. degree from Univ. of Electrocommunications in 2005 and M.E. degree from Keio Univ. in 2007, respectively. He was a Ph.D. student in Keio Univ. from 2007 to 2010. Since 2010 he works for Service Platforms Research Labs., NEC Corp. His research interests include Peer-to-Peer systems, distributed systems, middleware and operating systems. He is a member of the IPSJ, IEEE/CS, ACM and USENIX.

**Kenji Kono** received the BSc degree in 1993, MSc degree in 1995, and Ph.D. degree in 2000, all in computer science from the University of Tokyo. He is an associate professor of the Department of Information and Computer Science at Keio Univ. His research interests include operating systems, system software, and Internet security. He is a member of the IEEE/CS, ACM and USENIX.

**Toshinori Kojima** received his B.E. and M.E. degrees from Keio Univ. in 2008 and in 2010, respectively. Since 2010 he works for Research and Development Headquarters, NTT DATA CORPORATION. His research interests include network coordinates and distributed hash tables.



**Ai Hayakawa** received her B.E. and M.E. degrees from Keio Univ. in 2008 and in 2010, respectively. Since 2010 she works for Nomura Research Institute, Ltd. Her research interests include Peer-to-Peer systems and content distribution networks.