Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2011, Article ID 707592, 11 pages
doi:10.1155/2011/707592

Research Article

Fast Optimal Replica Placement with Exhaustive Search Using Dynamically Reconfigurable Processor

Hidetoshi Takeshita,[1] Sho Shimizu,[1] Hiroyuki Ishikawa,[1] Akifumi Watanabe,[2] Yutaka Arakawa,[1] Naoaki Yamanaka,[1] and Kosuke Shiba[2]

[1] *Department of Information and Computer Science, Faculty of Science and Technology, Keio University, Yokohama, 223-8522, Japan*
[2] *IPFlex Inc., Tokyo 141-0021, Japan*

Correspondence should be addressed to Hidetoshi Takeshita, takeshita@yamanaka.ics.keio.ac.jp

Received 4 April 2011; Revised 16 September 2011; Accepted 19 September 2011

Academic Editor: Khoa Le

This paper proposes a new replica placement algorithm that expands the exhaustive search limit with reasonable calculation time. It combines a new type of parallel data-flow processor with an architecture tuned for fast calculation. The replica placement problem is to find a replica-server set satisfying service constraints in a content delivery network (CDN). It is derived from the set cover problem which is known to be NP-hard. It is impractical to use exhaustive search to obtain optimal replica placement in large-scale networks, because calculation time increases with the number of combinations. To reduce calculation time, heuristic algorithms have been proposed, but it is known that no heuristic algorithm is assured of finding the optimal solution. The proposed algorithm suits parallel processing and pipeline execution and is implemented on DAPDNA-2, a dynamically reconfigurable processor. Experiments show that the proposed algorithm expands the exhaustive search limit by the factor of 18.8 compared to the conventional algorithm search limit running on a Neumann-type processor.

1. Introduction

Content delivery networks (CDNs) [1–3] are being developed to improve the user's experience when downloading voluminous files such as music and videos, a rapidly growing component of the traffic on the Internet. A CDN consists of two types of servers: origin server and replica server. The original data is stored in the origin server and then copied to the replica servers, which are geographically distributed. A user requesting content is connected to a replica server automatically selected by the network, which then sends the content to the user. Replica selection is based on the distance between the server and the user, and usually, the closest server is selected [4].

One important issue in CDN performance is replica placement [5]. The problem is in deciding which servers are to hold which replicas. Replica servers cache the original servers' contents to prevent traffic congestion and to maintain user performance. They allow CDN providers to minimize the capital expenditures (CapEx) and operational expenditures (OpEx). Note that cache size is restricted; no replica server can hold all of the contents held by the origin server. For each content, we must pick those servers, not all, that will hold the replica. In addition, in order to achieve adequate user performance, each replica server must have a limited delivery area. The delivery area is expressed as the distance from the replica server. Each user must lie within the delivery area of at least one replica server. For maximizing resource utilization, the number of replicas holding a content should be minimized while satisfying the delivery area constraint.

Selecting a combination that satisfies the constraint conditions in all combinations for replica placement is called the set cover problem. The replica placement problem is an extension of the set cover problem [6, 7], which is known to be NP-hard [8]. Several greedy algorithms have been proposed with the aim of decreasing the calculation time, since it rapidly increases when the number of servers is large [5, 9–12]. Greedy algorithms are widely used because of their simplicity. However, it has been mathematically proved that no greedy algorithm can guarantee the optimal solution. The optimal solution is the minimum replica-server set that satisfies the constraints, for example, quality of services. Our simulations also show that over 90 percent of the solutions output by the greedy algorithm are not optimal; we must search the entire solution space to get the optimal solution. There is a report to try exhaustive search [13] with liner programming, but they define sever groups and fixes location of servers and reduce the number of combinations. This requires an impractically long time if the network is large, especially when the calculations are run on a Neumann-type sequential processor.

In this paper, we propose a fast calculation method that uses parallel processing for the replica placement problem. Users' demands for guarantees of the quality of services trigger constraints such as the latency for content access and download time. The demands of providers, on the other hand, focus on maintaining the service level within minimum replica placement. Since contents and user requests are dynamically changing, it is necessary to rework replica placement dynamically. Shorter computation time contributes to satisfying the user and provider demands without delay.

The replica placement problem is to find the minimum replica-server set that satisfies all constraints such as quality. Our algorithm is not only for split Beeler's algorithm into parallel tasks. Our proposal is to expand the exhaustive search limit within reasonable calculation time by extract the maximum hardware performance with our architecture. Our proposed algorithm divides all patterns of replica server selection into groups. Each group is run in parallel using the pipeline technique. We prove how many groups are optimal. This algorithm suits the architectures of current dynamically reconfigurable processors, most of which have many processor elements (PEs). While the time complexity of the conventional method is $O({}_nC_k)$, that of the proposed algorithm is $O(\sqrt{{}_nC_k})$, where ${}_nC_k$ is the combination number, n is the number of servers in the network, and k is the number of selected servers in that search. In addition, we implement the proposed algorithm on DAPDNA-2, a commercial dynamically reconfigurable processor developed by IPFlex Inc. [14]. Experiments show that our proposed algorithm reduces the execution time by a factor of 18.8 compared to the conventional method on an Intel Pentium 4 2.8 GHz.

The rest of the paper is organized as follows. Section 2 describes the replica placement problem and related work. Section 3 explains our proposed algorithm for fast solution of the replica placement problem. Section 4 describes DAPDNA-2 implementation and performance evaluation results of our experiments. Finally, Section 5 concludes the paper.

2. Replica Placement Problem

Let $G = (V, E)$ be an undirected graph, where V is the set of servers and E is the set of links between any two servers. Link $(u, v) \in E$ is associated with cost $c(u, v)$, which denotes the cost of the link between servers u and v. It is assumed that the graph is connected so that any server can connect to any other server via a path. The cost of a path is defined as the sum of the costs of the links along the path and the cost of nodes between a user and a replica server. $d(u, v)$ is the cost of the shortest path between u and v.

The origin server, which at the start is the only server holding the data, is expressed as r. A replica server is a server that is assigned a replica. Each client is connected to a local server and sends its request to this server. If the server has the data requested, the request is processed locally. Otherwise, the request is redirected to the closest server that has a replica of the data requested. The client then obtains the data from that replica server. Every server u has a distance constraint, $q(u)$, to guarantee users' performance such as response time and download speed. A request from a client associated with server u must be serviced by a replica server within distance constraint $q(u)$. If the closest replica server lies outside $q(u)$, the request cannot be resolved. Therefore, each server must be connected to at least one replica server that can satisfy the constraint condition.

The replica placement problem is to decide which servers are to be selected as replica servers while satisfying the distance constraint, see (1), and minimizing the number of replica servers. A binary bit pattern of server selection is called a replication strategy and is expressed as a set of replica servers, $R \subseteq V - \{r\}$. If a replication strategy meets the constraint requirement, it is called feasible

$$\min_{v \in R \cup r} d(u, v) \leq q(v) \quad \text{for } \forall v \in V. \tag{1}$$

Figures 1 and 2 illustrate examples of replica placement. The numbers in the rings are the index number of the server; they lie between 0 an $n - 1$, where n is the total number of servers. The number on a link is the sum of the cost of the link between nodes and both nodes. In both figures, the origin server is server 0, and the distance constraint is 8, so the replication strategies shown are feasible.

The replica placement problem is derived from the set cover problem, which is known to be NP-hard. The definition of the set cover problem is as follows.

Minimum Weight Set Cover Problem. Let U be the universal set and S the family of U. The solution is subfamily $\mathbf{S}$ such that the weight is minimized and $\bigcup_{S \in \mathbf{S}} S = U$ is satisfied.

The replica placement problem is also NP-hard, since it has been proved that the minimum weight set cover problem is NP-hard. The computational complexity of replica placement is extremely high, especially when the number of servers in the network is large. Some greedy algorithms have been proposed [5, 9–12]. Johnson proposed a greedy algorithm for the minimum weight set cover problem [15]. The algorithm is a straightforward heuristic algorithm, requiring time $O(n)$. However, it has been mathematically proven that no greedy algorithm can guarantee the optimal solution. For example, Figure 1 shows replica placement with the greedy algorithm that selects the server that has the largest cover area. It is not the optimal solution, since the number of replica servers is 2, as shown in Figure 2. In addition, our simulation results show that about 90% of the replication strategies output by the greedy algorithm are not optimal as shown in Figure 3. In Figure 3, node degree represents the

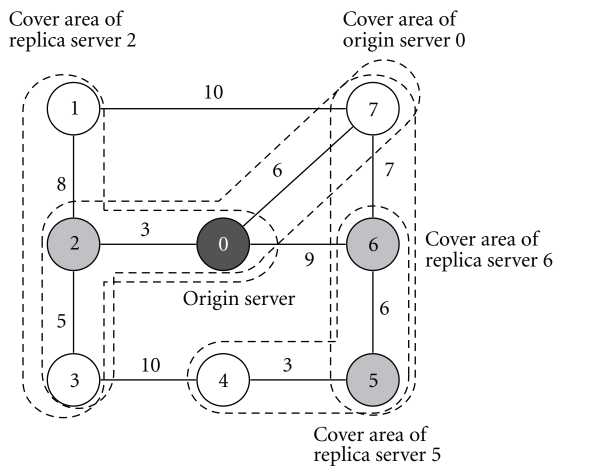

Cover area of replica server 2
Cover area of origin server 0
Cover area of replica server 6
Origin server
Cover area of replica server 5

Figure 1: Greedy algorithm cannot obtain the optimal solution, where the distance constraint, $q(u)$, is 8. The number of replica servers is 3.

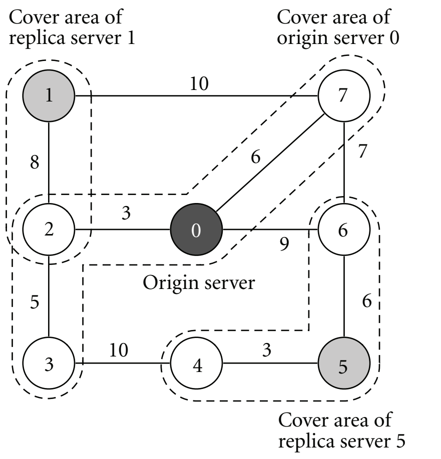

Cover area of replica server 1
Cover area of origin server 0
Origin server
Cover area of replica server 5

Figure 2: The optimal solution, where the distance constraint $q(u) = 8$. The number of replica servers is 2.

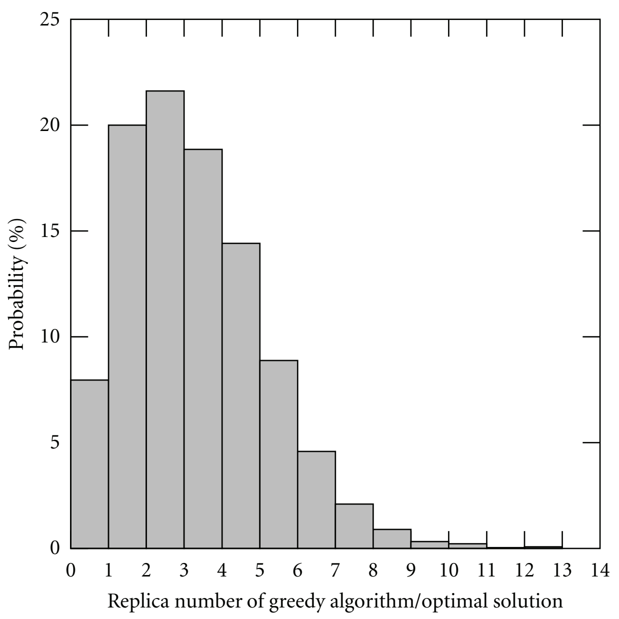


Figure 3: Probability of approximate ratio of the greedy algorithm, where the number of nodes is 32, and the average node degree is 3, the cost of a link is uniformly distributed between 1 and 15, and the number of simulations are 5000.

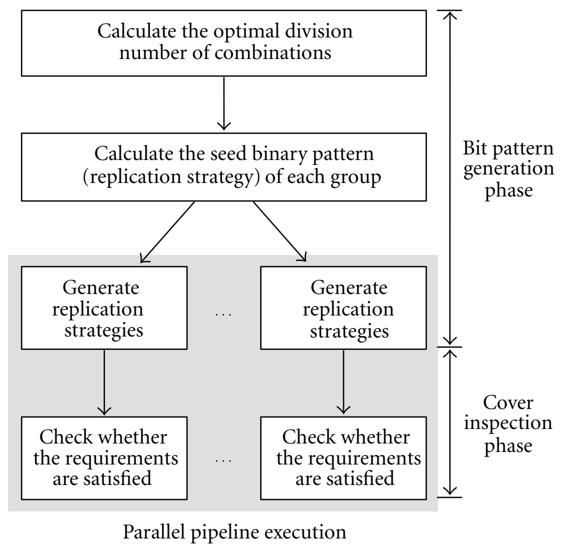

Calculate the optimal division number of combinations
Calculate the seed binary pattern (replication strategy) of each group
Generate replication strategies
Generate replication strategies
Check whether the requirements are satisfied
Check whether the requirements are satisfied
Bit pattern generation phase
Cover inspection phase
Parallel pipeline execution

Figure 4: Overview of the proposed algorithm.

number of links connected to a neighbor node. Greedy algorithms are realistic, since suboptimal solutions are obtained within realistic time. However, many of the solutions deviate from the optimal solution by over 100%. If we can obtain the optimal solution within realistic time, we do not need to pay the cost penalty of the suboptimal solutions.

In order to obtain the optimal solution, a fast exhaustive search algorithm is required. This approach is unrealistic if implemented on a Neumann-type sequential processor, since the number of replication strategies is too large.

3. Proposed Method

3.1. Overview. In this paper, we propose a fast calculation method to obtain the optimal solution for replica placement. The proposed architecture is based on exhaustive search, because only exhaustive search can obtain the optimal solution. Figure 4 overviews the procedures of our proposal. It consists of two phases: generation phase and inspection phase. In the generation phase, all replication strategies are calculated. The inspection phase checks whether each replication strategy can cover all servers and satisfy the distance constraint. Finally, the replication strategy with the minimum number of replica servers is selected as the optimal solution.

In the generation phase, calculation of the optimal division number of combinations and the seed replication strategies are complicated and not repeatable, so software is suitable for implementation. Generation of replication strategy with binary bit patterns (replication strategies) and inspection of cover area are simple and repeatable, so hardware is suitable for implementation. Therefore, our proposed architecture needs both hardware for software execution and binary bit operation.

Candidate hardware for implementation is FPGA (field programmable gate array) and a reconfigurable processor. It is difficult to create software-execution function on FPGA, because we need private compiler. So, we selected a reconfigurable processor. Though there are many reconfigurable processors, we selected DAPDNA-2 because of that DAPDNA-2 combines high-performance RISC (reduced instruction set computer) processor and high-capacity process elements, and hardware configuration can be changed by one clock. We implemented the proposed algorithm on DAPDNA-2 closely cooperated with hardware DAPDNA-2 and software and drew out the maximum performance of it. DAPDNA-2 is commercially available dynamically reconfigurable processor developed by IPFlex Inc. [14, 16].

In order to employ parallel pipeline execution to speed up exhaustive search, the replication strategies are divided into several groups in the generation phase. The replication strategies of each group are simultaneously generated using Beeler's algorithm [17, 18], which generates the next replication strategy from the previous replication strategy. Beeler's algorithm is a sequential algorithm, so it is not suitable to obtain the *m*th replication strategy directly. To divide all replication strategies into groups, we propose a new algorithm that calculates the *m*th replication strategy directly. In addition, determining the division number that minimizes the calculation clock total is a problem to be solved. We derive the optimal number of divisions theoretically in Section 3.2.4.

3.2. Bit Pattern Generation Phase

3.2.1. Bit Pattern Notation. First, the replication strategy notation is introduced. A replication strategy can be expressed as an n-bit bit pattern, where the ith bit corresponds to the server indexed as i and n is the total number of servers in the network. 1 on the ith bit means server i is selected as a replica server. By using this notation, when k $(\leq n)$ servers are picked from n servers, k-bits of the n-bit bit pattern are set to 1, where k is the number of original and replica servers. For example, 010110 represents the replication strategy $\{1, 2, 4\}$ for 6 servers. These bit patterns allow us to sort replication strategies in ascending order. Figure 5 shows the bit pattern order when $k = 3$ servers are selected from $n = 6$ servers. It seems that these values increase irregularly, since the differences of neighboring values are not constant.

[Replication strategy $\{0, 1, 5\}$ covers all servers] [Replication strategy $\{0, 3, 7\}$ does not cover all servers].

3.2.2. Beeler's Algorithm. Our proposed algorithm aims to minimize the hardware operation clocks to minimize calculation time. A simple counter that generates ascending values needs many clocks to find the combination binary patterns because the "1" bit-combination patterns in ascending values are random, and there is no regularity. On the other hand, Beeler's algorithm ensures regularity and needs fewer clocks than a simple counter. Accordingly, we utilize Beeler's algorithm.

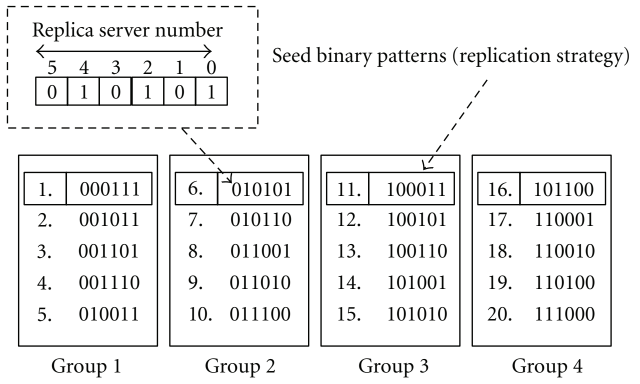


Figure 5: Example of replication strategy (ordering bit patterns), where $n = 6$ and $k = 3$, and division yields 4 groups.

Beeler proposed an algorithm that generates all combinations, where k outcomes are picked from n possibilities in ascending order [17, 18]. These combinations can be expressed as n-digit binary sequences. For example, the binary bit pattern "010101" represents replica server numbers (0, 2, 4) when $n = 6$, as shown at seed binary bit pattern of group 2 in Figure 5. Combinations can be ordered in ascending order. Replica server numbers (0, 2, and 4) are smaller than replica server numbers (1, 3, and 4), because the binary bit pattern "010100" is smaller than the binary bit pattern "011010". Beeler's algorithm can generate all bit patterns from "000111" to "111000" in order. The algorithm is explained as follows.

There are five steps to generate the next binary bit pattern-Y from the original binary bit pattern-X.

Beeler's Algorithm. There are five steps to generate the next binary bit pattern-Y from the original binary bit pattern-X.

(1) Let S_1 be given, as all bits are unset except for the least significant bit 1 in binary pattern-X.

(2) $R_1 = X + S_1$.

(3) Let S_2 be given, as all bits are unset except for the least significant bit 1 in binary pattern-R_1.

(4) $R_2 = (S_2/S_1) \gg 1 - 1$ [$\gg i$: shift right i-bit].

(5) $Y = R_1 | R_2$, and Y is next to X.

When $n = 6$, $k = 3$, $X = 001110$, for example, Y is calculated as follows:

(1) $S_1 = 000010$,

(2) $R_1 = X + S_1 = 010000$,

(3) $S_2 = 010000$,

(4) $R_2 = (S_2/S_1) \gg 1 - 1 = 001000 \gg 1 - 1 = 000100 - 1 = 000011$,

(5) $Y = R_1 | R_2 = 010011$.

With Beeler's algorithm, we can generate all bit patterns, representing replication strategies, in ascending order. For example, when $k = 3$, bits are selected from $n = 6$ bits; we start from 000111 as the first bit pattern, and from this, we get the next bit pattern 001011. Consequently, by sequential calculation, 19 iterations of Beeler's algorithm produces all bit patterns shown in Figure 5.

Parallel pipeline execution is a technique to speed up exhaustive search. However, Beeler's algorithm in itself is not applicable to parallel pipeline execution, since it has data dependency; we need the previous bit pattern to calculate the next bit pattern. In parallel pipeline execution, we divide all replication strategies into groups, and the bit patterns of each group are simultaneously generated from the first value of each group. The first replication strategy of a group is called the group's seed. Therefore, an algorithm that can calculate the bit pattern of a seed directly is required for parallel pipeline execution. Unfortunately, Beeler's algorithm is not suitable for this purpose, and no known algorithm is useful either. We propose here an algorithm to calculate the bit patterns of the seeds directly.

3.2.3. Direct Calculation of a Seed Bit Pattern. Our proposed algorithm that generates any-order patterns in sequences that are sorted in ascending order. More generally, a nonordered pattern is generated by the following equation. We use the characteristic of the following equation to calculate the bit pattern representing the mth replication strategy. Here, ${}_nC_k$ is the bit pattern number, the bit pattern representing the mth replication strategy is called the mth pattern

$$ {}_nC_k = \sum_{i=k-1}^{n-1} {}_iC_{k-1}. \tag{2}$$

To get the mth pattern, find the smallest x_1 that satisfies

$$\sum_{i=k-1}^{x_1} {}_iC_{k-1} \geq m \quad (k-1 \leq x_1 \leq n-1), \tag{3}$$

${}_{x_1}C_{k-1}$ means the bit patterns whose most significant bit with value "1" is the x_1th bit; there are $k-1$ "1"s between the 1st bit and the x_1-1th bit because there are k "1"s in total. Hence, the x_1th bit of the mth pattern is "1". The mth pattern corresponds to $m - \sum_{i=k-1}^{x_1-1} {}_iC_{k-1}$-th in ${}_{x_1}C_{k-1}$. Replace m as follows:

$$m \longrightarrow m - \sum i = k - 1^{x_1-1} {}_iC_{k-1}. \tag{4}$$

Next, find the smallest x_2 that satisfies

$$\sum_{i=k-2}^{x_2} {}_iC_{k-2} \geq m \ (x_2 \leq x_1 - 1), \tag{5}$$

${}_{x_2}C_{k-2}$ means the patterns whose most significant bit with value "1" is the x_2th bit; there are $k-2$ "1"s between 1st and (x_2-1)th bit. Hence, the x_2th bit of the pattern is "1". $x_1, x_2, \ldots, x_k$ can be obtained by repeating k times in a similar way. Setting the corresponding bit to "1" yields the mth pattern.

For example, the 6th pattern ($m = 6$) in ${}_6C_3$ can be obtained as follows:

$${}_6C_3 = {}_2C_2 + {}_3C_2 + {}_4C_2 + {}_5C_2 = 1 + 3 + 6 + 10. \tag{6}$$

Apply (2) to ${}_4C_2$, because ${}_4C_2$ includes the 6th pattern. Hence, $x_1 = 4$, $m \rightarrow 2$.

$${}_4C_2 = {}_1C_1 + {}_2C_1 + {}_3C_1 = 1 + 2 + 3. \tag{7}$$

Apply (2) to ${}_2C_1$, because ${}_2C_1$ includes the 2nd pattern. Hence, $x_2 = 2$, $m \rightarrow 1$.

$${}_2C_1 = {}_0C_0 + {}_1C_0 = 1 + 1. \tag{8}$$

The 1st pattern corresponds to ${}_0C_0$. Hence, $x_3 = 0$. Set the corresponding bit to 1 which yields the 6th pattern, 010101.

After dividing all bit patterns, pick k-bits from n-bits, into d groups, and then calculate the bit patterns of the seeds, such as $1,\ {}_nC_k/d+1,\ 2\cdot{}_nC_k/d+1,\ \ldots,\ (d-1)\cdot{}_nC_k/d+1$th bit patterns using the above algorithm.

3.2.4. Optimal Number of Groups. Determining the number of groups is a problem that must be solved prior to parallel pipeline execution of exhaustive search. As the number of groups increases, the calculation clocks needed to generate the bit patterns of the seeds increases. The optimal number of groups depends on the number of all replication strategies and the calculation clocks of Beeler's algorithm. In order to solve this problem, we introduce here the theoretical optimal number of groups.

Let a be the number of clocks needed to calculate the bit pattern of a seed and b the number of clocks needed to execute Beeler's algorithm. $b({}_nC_k - 1)$ clocks are required to generate all combinations created by picking k outcomes from n possibilities. When we divide all replication strategies into 2 groups, $a + b({}_nC_k - 1)/2 + 1$ clocks are required. When we divide them into 3 groups, $2a + b({}_nC_k - 1)/3 + 2$ clocks are required. More generally, when we divide them into x groups, y clocks are required as follows:

$$\begin{aligned} y &= (x-1)a + \frac{b({}_nC_k - 1)}{x} + x - 1 \\ &= \frac{b({}_nC_k - 1)}{x} + (a+1)x - a - 1. \end{aligned} \tag{9}$$

According to the relationship between arithmetic mean and geometric mean,

$$\begin{aligned} y &= \frac{b({}_nC_k - 1)}{x} + (a+1)x - a - 1 \\ &\geq 2\sqrt{\frac{b({}_nC_k - 1)}{x}(a+1)x} - a - 1 \\ &= 2\sqrt{b({}_nC_k - 1)(a+1)} - a - 1. \end{aligned} \tag{10}$$

The equality is satisfied if and only if $b({}_nC_k - 1)/x = (a+1)x$. Hence,

$$x = \sqrt{\frac{b({}_nC_k - 1)}{a+1}}. \tag{11}$$

This is the optimal division number.

3.3. Cover Inspection Phase. In the inspection phase, we check whether each replication strategy is feasible or not. The cover area of each replica server represents the distance constraint, and it can be calculated using a shortest path algorithm such as Dijkstra's algorithm [19]. In this paper, it is assumed that the data representing the cover areas for all servers are calculated with Dijkstra's algorithm before the inspection phase. The data is expressed as a bit pattern as in the replication strategies. In a network with n servers, we use an n-bit bit pattern to represent the cover areas. The n-bit of the bit pattern corresponds to the cover area of one server. The value of 1 on the ith bit means that the server indexed as i is covered. For example, 010110 indicates that servers $\{1, 2, 4\}$ are covered.

Figure 6 shows the procedure of the cover phase when the replication strategies are $\{0, 1, 5\}$ and $\{0, 3, 7\}$ in the network shown in Figure 2. The data of the cover area of the servers where the replicas are placed is looked up using the bit pattern of a replication strategy. Next, all looked-up data is subjected to Bit-wise OR operation. If all n-bits of the result are 1, the replication strategy covers all servers. Otherwise, it does not cover all servers.

4. DAPDNA-2 Implementation and Evaluation

4.1. DAPDNA-2 Implementation. DAPDNA-2 is a heterogeneous dual core processor. It consists of two cores: DAP (digital application processor), and DNA (distributed network architecture). These processors have different architectures. DAP is a 32-bit RISC core for controlling DNA; DNA is a reconfigurable parallel data-flow machine. Typically, the main data processing of an algorithm is run on DNA. DNA consists of 376 processor elements (PEs), each comprising computation units, internal memory, synchronizers, and counters; the PEs are arranged in a grid pattern. We can design the connections between PEs when implementing an algorithm on DAPDNA-2 to yield a parallel data-flow machine. Each structure is called a configuration. DNA can keep three configurations in its own internal cache. These configurations can be switched within one clock. Thus, this chip combines the advantages of the high-speed processing of hardware and the flexibility of software. Details of its architecture are described in [16]. Let n be the number of nodes and k $(\leq n)$ the number of original and replica servers. In our implementation, $n \leq 32$, since PE word size in DNA is 32-bits long.

Figure 7 shows a block diagram of the DAPDNA-2 implementation design, and Figure 8 shows a block diagram of the cover inspection unit as implemented on DNA; see Figure 7. We must generate and check huge number of combinations to solve the replica placement problem. So, we focus on reducing the calculation time for generation phase and inspection phase with parallel and pipeline computation and increasing the number of parallel computation. So, we considered four major issues in implementation.

(1) We designed Beeler's algorithm unit and inspection unit in Figure 7 to operate with minimum clocks and minimum PEs.

(2) We set the depth of the pipeline operation for Beeler's algorithm unit in Figure 7 to 18. This means that we can get the first output after 18 clocks and the next outputs arrive one by one at each clock until the pipeline is empty.

(3) We implemented inspection unit in Figure 7 as four divided cover-area-data tables with 8-bit indexing to reduce the PE resources and operation clocks. Because direct 32-bit indexing needs 4(32-bit) $\times$ 2^{32} bytes of memory and operation clock is 1, the bit-position-number search method, shown in Figure 6, needs many clocks to find bit-position-number and to index cover-area-data tables. Therefore, we produced the cover-area-data tables shown in Figure 8; they need only four memory tables with 1(8-bit) $\times$ 256 byte memories and operation clock is 4.

(4) We could minimize the PE resources for Beeler's algorithm unit and inspection unit and create four blocks for parallel calculation in DNA.

Calculation is executed in the following 8 steps.

Step 1: DAP calculates the seed binary bits patterns (replication strategies) using the algorithm described in Section 3.2.

Step 2: DAP writes seed replication strategies into external memory.

Step 3: DNA reads seeds replication strategies in external memory.

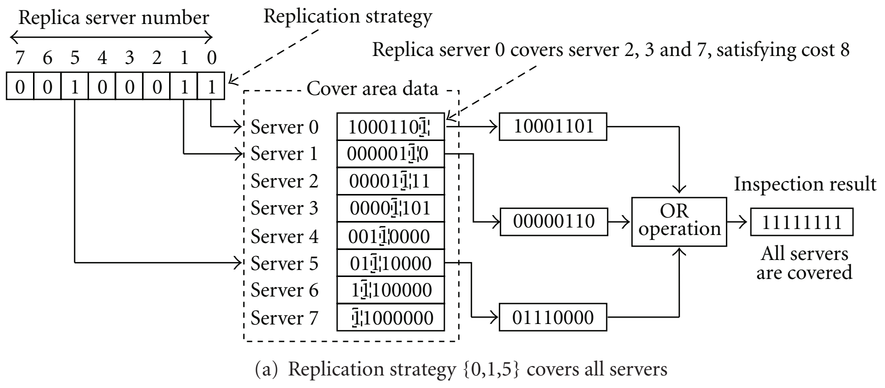
Step 4: Beeler's algorithm unit in DNA executes Beeler's algorithm by pipeline calculation. The depth of pipeline calculation in Beeler's algorithm calculation unit is 18; we can get the first output after 18 clocks, and the next outputs arrive one by one at each clock until the pipeline is empty. So, we divide all replication strategies into 18 groups per calculation to minimize the calculation time.

Step 5: Inspection unit in DNA checks the cover area, which is the output of Beeler's algorithm unit. The principle of the inspection algorithm is described in Figure 8 shows the implementation of the inspection unit. The bit pattern representing a replication strategy, which is 32-bits long, was divided into 4 data blocks; each block is 8-bits long, and the cover area is looked up using the 8-bit data. The 4-bit patterns of the looked-up cover area are then subjected to the bitwise OR operation. Finally, the result of the OR operation is compared to 0xFFFFFFFF.

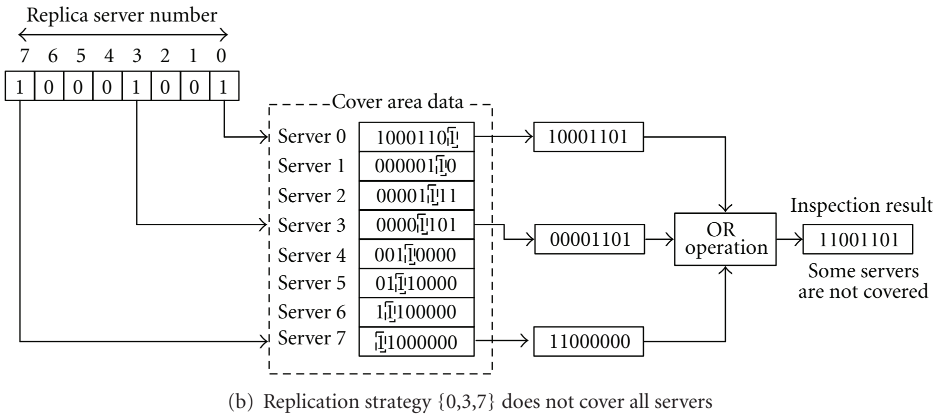
Step 6: Inspection unit in DNA writes the inspection results to external memory.

Step 7: DAP reads the inspection results in external memory.

Step 8: DAP checks the inspection results and decides replica placement.

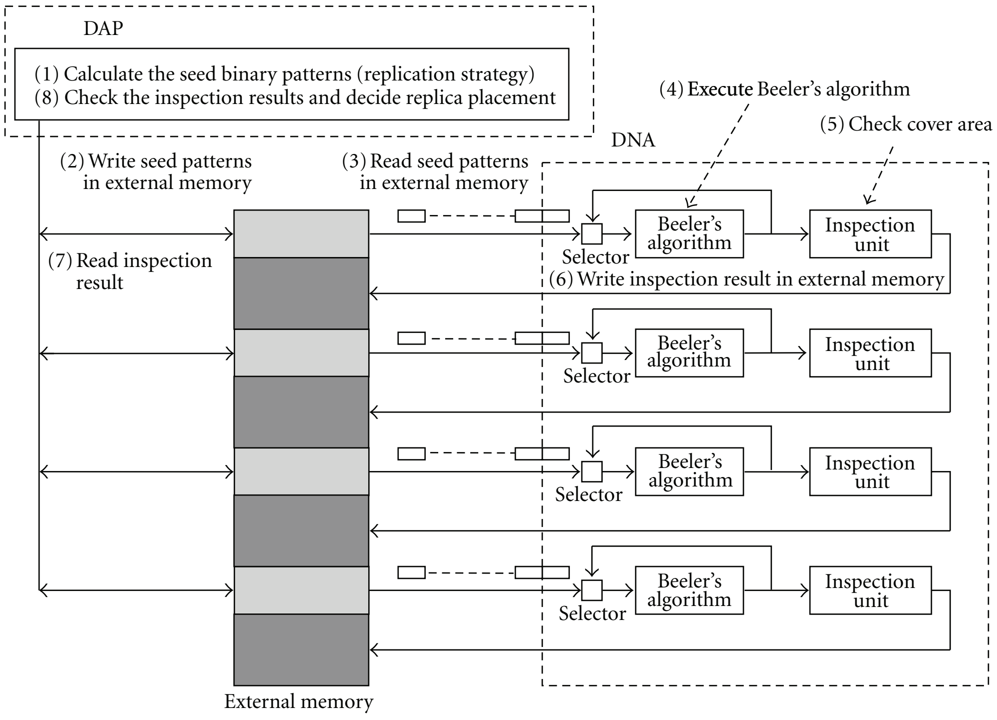


(a) Replication strategy {0,1,5} covers all servers



(b) Replication strategy {0,3,7} does not cover all servers

Figure 6: The cover inspection of replication strategies satisfying cost 8.



DAP
(1) Calculate the seed binary patterns (replication strategy)
(8) Check the inspection results and decide replica placement
(4) Execute Beeler's algorithm
(5) Check cover area
DNA
(2) Write seed patterns in external memory
(3) Read seed patterns in external memory
Beeler's algorithm
Inspection unit
Selector
(6) Write inspection result in external memory
(7) Read inspection result
External memory

Figure 7: Block diagram of DAPDNA-2 implementation and execution steps.

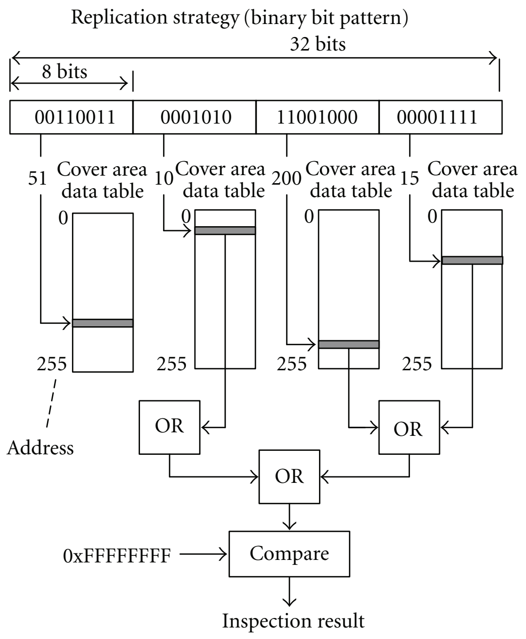


Figure 8: Implementation of the inspection unit on DNA in Figure 7.

4.2. Evaluation. We conducted experiments to evaluate the performance of our proposal; values of k were increased until a solution was found, where k is the number of replica and original servers. Three types of network models were considered: grid network ($n = 4 \times w$ $(=3, 4, 5, 6, 7, 8)$), ring network ($n = 4 \times w$ $(=3, 4, 5, 6, 7, 8)$), and nation-wide network ($n = 31$). Figure 9(a) shows the nation-wide network model. The distance constraint is expressed as the maximum number of hops h. We compare the performances of our proposed approach and the conventional approach. The implementation mentioned above is used as the proposed approach, and the conventional approach, sequential Beeler's algorithm, was executed on a Intel Pentium 4 3.6 GHz processor. In the proposed approach, 4 pairs of Beeler's algorithm unit and the inspection unit were used as shown in Figure 7. The conventional approach was implemented as a C program and compiled with GNU C Compiler (GCC) with "-O3" optimization option.

Figure 10 shows the execution time of the proposed and the conventional algorithm approach for the ring, grid, and nation-wide network models, where the distance constraint $h = 1$. The grid and ring network model each have $n = 32$ nodes, while the nation-wide network model has $n = 31$ nodes. The result shows that the proposed algorithm has much shorter execution times than the conventional algorithm for all network models. It is over 16 times, 14 times, and 12 times faster for the ring, grid, and nation-wide network models, respectively. This is because the proposed algorithm makes good use of the parallel and pipeline calculation provided by DAPDNA-2. The conventional algorithm is a strictly sequential algorithm, and a lot of replication strategies must be generated and searched. Note that execution time is strongly dependent on topology. For example, the proposed approach takes 363 milliseconds for the ring model but only 9.5 milliseconds for the nation-wide network model. This is due to the difference in average node degrees of the topologies. The average node degree of ring network model is 2, grid network model is 3.25, and nation-wide network model is 4.1. So, the cover area of a replica server in the ring network model is smaller than that of in the grid network model, and that of grid network model is smaller than that of nation-wide network model. When the cover area is small, an increased number of binary combination patterns must be generated and inspected. Therefore, the execution time increases.

Next, we investigate the characteristics of our proposed algorithm. Figure 11 shows the execution time of the proposed algorithm versus the node cost constraint. We evaluated node cost, where link cost is constant and node cost is variable by hops, h on the grid and ring networks, where h is the number of hops; the number of nodes $n = 32$. As h increases, the execution time decreases. This is because the cover area of a replica server is large when h is large; in other words, the latency constraint is relaxed. Larger cover areas can reduce the execution time and reduce the number of replica servers.

Figure 12 shows the execution time of the proposed algorithm versus the number of nodes on the ring and grid networks. The execution time increases as the number of nodes and the number of replica servers increases. The execution time is proportional to the number of combinations created by the number of nodes and the number of replica servers, because the generated combination binary patterns and inspection patterns increase. The calculation load greatly increases with the number of nodes.

We can theoretically estimate the execution time when the number of nodes. The experiment showed that the execution time per bit pattern of DAPDNA-2 and Pentium 4 was 1.5 nanoseconds and 28 nanoseconds, respectively. If the execution time per combination binary patterns is expressed as t, the total execution time T is given by

$$T = t \times \sum_{k=1}^{f} {}_{n}C_{k}, \tag{12}$$

where f is the minimum number of replica servers for which the replication strategy is feasible. To identify f, we assume the grid network model as shown in Figure 9(b) with $n = p \times q$ nodes. The ideal value of f when $h = 1$ is given by

$$f_{\text{ideal}} = \left\lceil \frac{n}{d_{\text{ave}}} \right\rceil, \tag{13}$$

where d_{ave} is the average node degree of the network. In the grid network, the average degree

$$\begin{aligned} d_{\text{ave}} &= (2 \times 4 + 3 \times (p-2) \times 2 + 3 \times (q-2) \times 2 \\ &\quad + 4 \times (p-2) \times (q-2)) \div (p \times q) \\ &= \frac{2(2p \times q - p - q)}{(p \times q)} = 2\left(2 - \frac{1}{p} - \frac{1}{q}\right). \end{aligned} \tag{14}$$

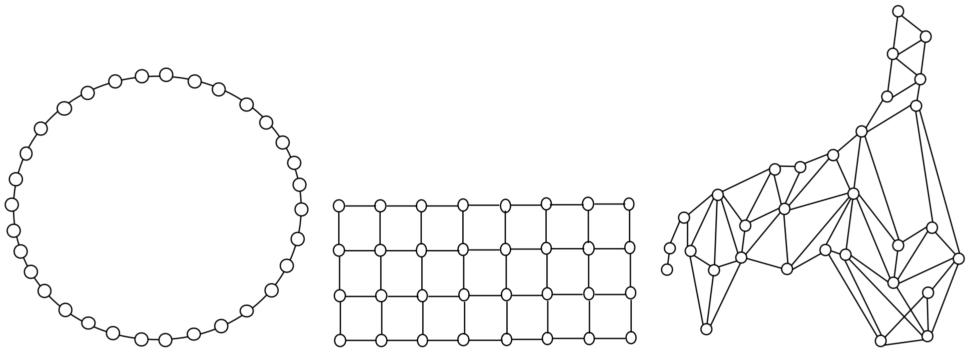

(a) Ring network model (32 nodes) (b) Grid network model (32 nodes: $p = 4$, $q = 8$) (c) Nation-wide network model (31 nodes)

FIGURE 9: Network model topology.

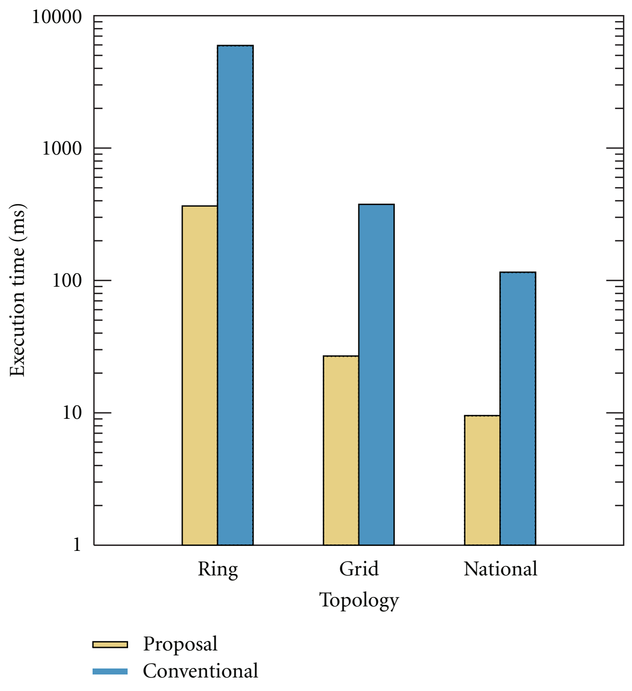


FIGURE 10: Execution time of the proposed algorithm and the conventional algorithm on ring, grid, and nation-wide server networks.

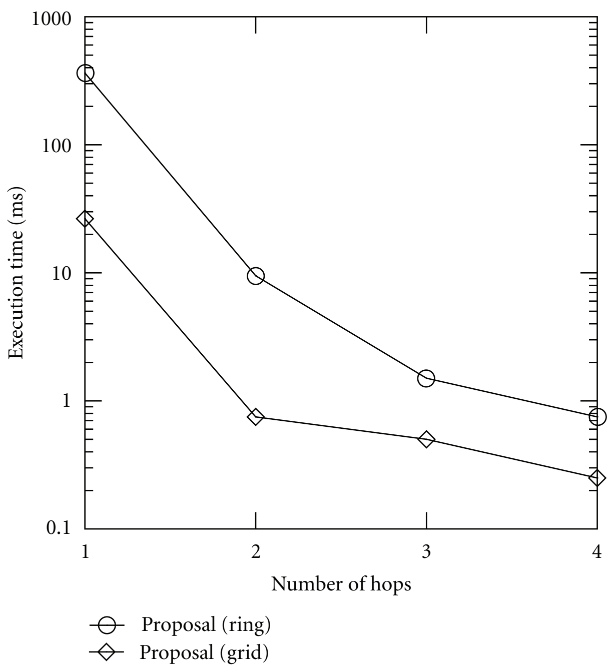


FIGURE 11: Execution time of the proposed algorithm versus the number of hops on the ring and grid network model.

From (12), (13), and (14), and the execution time per binary combination pattern of DAPDNA-2, and Pentium 4, we can estimate the execution times even if the number of nodes $n > 32$.

Figure 13 show the execution time of the proposed algorithm and of the conventional algorithm derived from the above equations, where $h = 1$, $p = 4$, and $q =$ from 2 to 32. The execution time of the proposed algorithm is 18.8 times faster than that of the conventional algorithm for all cases examined.

5. Conclusion

This paper proposed a fast calculation method to solve the replica server placement problem; exhaustive search is implemented in parallel form to discover the optimum solution in practical time. To achieve fast exhaustive search, we employ parallel pipeline execution. Exhaustive search consists of two phases: the generation phase and inspection phase. In order to be able to use parallel pipeline execution, we have introduced the following two points:

(1) calculation method that directly determines the mth replication strategy expressed as a bit pattern,

(2) theoretical proof of the optimal number of groups.

In addition, we implemented our proposed method on DAPDNA-2, a dynamically reconfigurable processor. The results of experiments on ring, grid, and nation-wide networks showed that our proposed method is 18.8 times faster than the conventional method on an Intel Pentium 4. This is

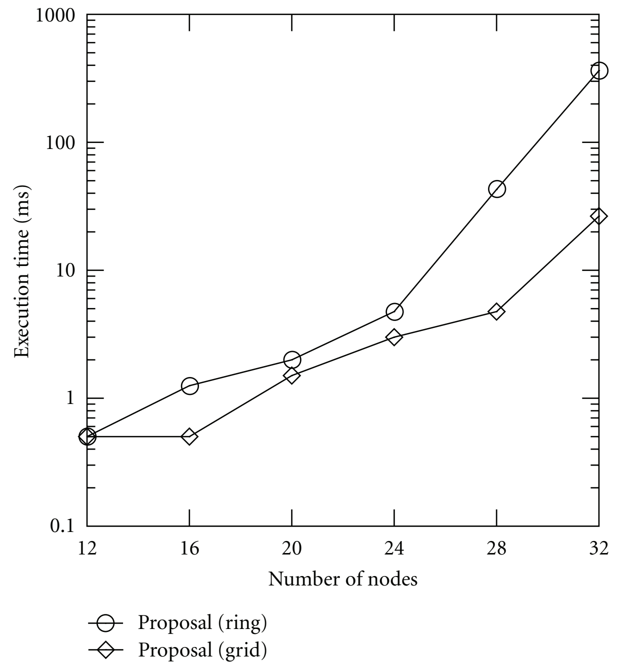


Figure 12: Execution time of the proposed algorithm versus the number of nodes on the grid and ring network model.

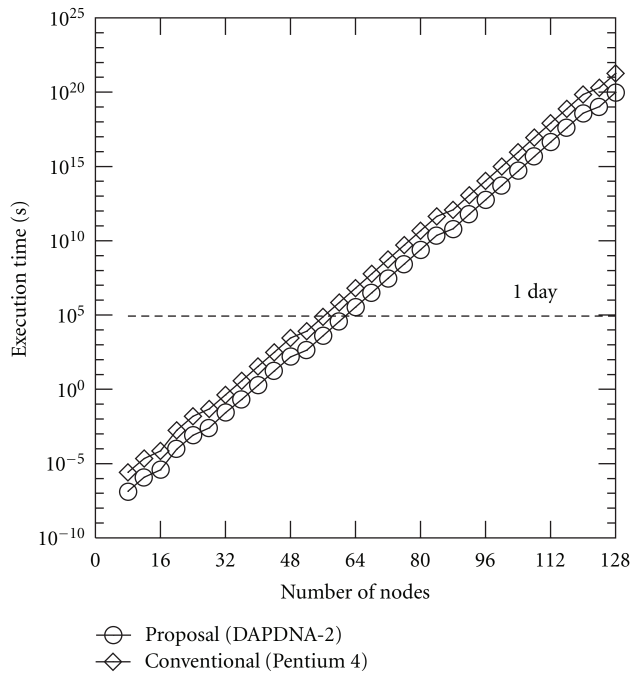


Figure 13: Execution time versus the number of nodes on the grid network model (theoretical estimation).

because the proposed method can realize close cooperation between hardware (DAPDNA-2) and software, and thus fully utilize the performance of DAPDNA-2. Therefore, the proposed algorithm expands the exhaustive search limit by a factor of 18.8 compared to the conventional algorithm limit running on a Neumann-type processor.

Acknowledgments

This work is supported by the PREDICT program of the Ministry of Internal Affairs and Communications (MIC) of Japan. This work is also supported in part by a Grant-in-Aid for the Global Center of Excellence for high-Level Global Cooperation for Leading-Edge Platform on Access Spaces from the Ministry of Education, Culture, Sport, Science, and Technology in Japan.

References

[1] "Akamai," http://www.akamai.com/.
[2] "Level 3," http://www.level3.com/.
[3] "Mirrorimge," http://www.mirror-image.com/.
[4] F. L. Presti, C. Petrioli, and C. Vicari, "Dynamic replica placement and user request redirection in content delivery networks," in *Proceedings of the 4th Annual IEEE International Conference on Communications (ICC '05)*, vol. 3, pp. 1495–1501, May 2005.
[5] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '01)*, vol. 3, pp. 1587–1596, April 2001.
[6] M. Karlsson and C. Karamanolis, "Bounds on the replication cost for QoS," Tech. Rep., Hewlett Packard Labs, 2003.
[7] C. Toregas, C. ReVelle, and L. Bergman, "The location of emergency service facilities," *Operations Research*, vol. 19, pp. 1363–1373, 1971.
[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.
[9] K. Jain, M. Mahdian, and A. Saberi, "A new greedy approach for facility location problems," in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pp. 731–740, May 2002.
[10] J. Kangasharju, J. Roberts, and K. W. Ross, "Object replication strategies in content distribution networks," *Computer Communications*, vol. 25, no. 4, pp. 376–383, 2002.
[11] X. Tang and J. Xu, "QoS-aware replica placement for content distribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 10, pp. 921–932, 2005.
[12] H. Wang, P. Liu, and J.-J. Wu, "A QoS-aware heuristic algorithm for replica placement," in *Proceedings of the 7th IEEE/ACM International Workshop on Grid Computing*, pp. 96–103, September 2006.
[13] S. Zaman and D. Grosu, "A distributed algorithm for the replica placement problem," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 9, pp. 1455–1468, 2011.
[14] T. Sato, H. Watanabe, and K. Shiba, "Implementation of dynamically reconfigurable processor DAPDNA-2," in *Proceedings of the IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test (VLSI-TSA '05)*, pp. 323–324, April 2005.
[15] D. S. Johnson, "Approximation algorithms for combinatorial problems," *Journal of Computer and System Sciences*, vol. 9, no. 3, pp. 256–278, 1974.
[16] T. Sugawara, K. Ide, and T. Sato, "Dynamically reconfigurable processor implemented with IPFlex's DAPDNA technology," *IE-ICE Transactions on Information and Systems*, vol. E87-D, no. 8, pp. 1997–2003, 2004.
[17] M. Beeler, R. W. Gosper, and R. Schroeppel, "HAKMEM ITEM175," http://www.cl.cam.ac.uk/~am21/hakmemc.html.

[18] M. Beeler, R. W. Gosper, and R. Schroeppel, in *C: A Reference Manual*, S. P. Harbison and G. L. Steele Jr., Eds., p. 176, Prentice-Hall, 2nd edition, 1978.

[19] E. W. Dijkstra, “A note on two problems in connection with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.