IEICE TRANS. COMMUN., VOL.E92–B, NO.10 OCTOBER 2009

PAPER

Resource Minimization Method Satisfying Delay Constraint for Replicating Large Contents

Sho SHIMIZU[†a)], ***Student Member***, **Hiroyuki ISHIKAWA**[†*], ***Nonmember***, **Yutaka ARAKAWA**[†], **Naoaki YAMANAKA**[†], ***and*** **Kosuke SHIBA**[††], ***Members***

SUMMARY How to minimize the number of mirroring resources under a QoS constraint (resource minimization problem) is an important issue in content delivery networks. This paper proposes a novel approach that takes advantage of the parallelism of dynamically reconfigurable processors (DRPs) to solve the resource minimization problem, which is NP-hard. Our proposal obtains the optimal solution by running an exhaustive search algorithm suitable for DRP. Greedy algorithms, which have been widely studied for tackling the resource minimization problem, cannot always obtain the optimal solution. The proposed method is implemented on an actual DRP and in experiments reduces the execution time by a factor of 40 compared to the conventional exhaustive search algorithm on a Pentium 4 (2.8 GHz).

key words: *content delivery network, replica placement, dynamically reconfigurable processor, exhaustive search*

1. Introduction

Demand continues to grow for downloading rich contents, for example DVD-quality or high definition videos, through the Internet. Two factors are the key to meeting this demand: local content sources and adequate transfer capacity. Optical networks can provide the high-speed and high-capacity pipes needed; they are now commonly used in backbone networks and can handle bandwidth-consuming applications if the transfer distances are reasonable. This paper focuses on the other factor, an shows how to determine where to site content sources.

Identifying the optimum number and location of content sources (servers) involves an understanding the tradeoffs between performance and cost. Using just a few servers is very effective in reducing initial investment costs but the servers will experience extremely high loads since they must deal with simultaneous download requests from many clients. Moreover, the average transfer distance is high which degrades the QoS and indeed overall network performance.

The content delivery network (CDN) was proposed to improve network resource utilization efficiency for large contents distribution [1], [2]. CDN consists of two types of servers: origin server and replica server. The number of origin servers is usually one (for each contents provider), and the many replica servers are spread throughout the service area. Origin server holds the original contents and delivers them to the replica servers as needed to ensure user requests can be satisfied. The contents stored in a replica server are called replicas. CDN promises high-speed downloads since the client downloads the data from the server nearest to the client in terms of network connectivity.

In CDN, replica placement impacts the performance which includes the load on the origin server and the network since data placement decisions must be made on a per content basis and be made dynamically in response to user requests. Minimizing the number of mirroring resources (servers) under a Quality of Service (QoS) constraint is a key issue in CDN, so research in this area has been quite active. A tough problem to select which nodes should host which replicas.

The distance between two nodes is used as a metric for QoS in CDN. A request must be resolved by a server within the distance specified by the request because all clients want to download contents within the allotted time period. Every node knows the nearest replica server that holds the requested data and the request is sent to the replica server that is closest to the client. The goal is to find a replica placement that satisfies all requests without violating any range constraint, and that minimizes the update and storage cost at the same time. This paper emphasizes the optimization of the number of replicas under the delay constraint.

Replica placement problem is derived from the set cover problem which is known to be NP-hard [3]. Therefore, calculation time increases rapidly with network scale. Greedy algorithms have been widely studied since they yield sub-optimal solutions reasonably quickly [4]–[11]. However, it has been proven mathematically that no greedy algorithm can attain the optimal solution [3]. Sub-optimum solutions have higher replicating cost, i.e. the number of replicas, than the optimal solution. The goal then is to secure the optimal replica placement within some practical time.

Our solution to obtaining the optimal solution to the replica placement problem is based on combining advanced processors with suitable algorithms. It is not realistic to obtain the optimal solution with a Neumann-type processor given the number of all solution candidates. To drastically reduce the calculation time, we propose a novel approach that uses an exhaustive search algorithm that suits the parallelism offered by a dynamically reconfigurable parallel pro-

Manuscript received February 10, 2009.
Manuscript revised June 10, 2009.

[†]The authors are with the Graduate School of Science for Open and Environmental Systems, Keio University, Yokohama-shi, 223-8522 Japan.
[††]The author was with IPFlex Inc., Tokyo, 141-0021 Japan.
[*]Presently, with Kansai Electric Power Co. Inc.
a) E-mail: shimizu@yamanaka.ics.keio.ac.jp

DOI: 10.1587/transcom.E92.B.3102

cessor (DRP). The proposal is the marriage of advances in software and hardware.

Our proposal is implemented on a commercially available DRP, DAPDNA-2 of IPFlex Inc [12]. DAPDNA-2 consists of a Digital Application Processor (DAP), a high-performance RISC core, and Digital Network Architecture (DNA), a dynamically reconfigurable two-dimensional matrix. DNA is embedded in an array of 376 Processing Elements (PEs), which are comprised of computation units, memory, synchronizers, and counters. The PE Matrix circuitry can be reconfigured freely into the structure that best suits the current application.

It is not feasible to solve the large-scale replica placement problem on a program counter-based processor. Our proposed algorithm divides the problem in an optimal manner and subjects the pieces to pipeline operation. Whereas the time complexity of conventional exhaustive search using Beeler's algorithm [13] is $O({}_nC_k)$, the time complexity of the proposed algorithm is $O(\sqrt{{}_nC_k})$. Experiments show that the proposed method reduces the execution time by a factor of 40 times compared to conventional exhaustive search using Beeler's algorithm on a Intel Pentium 4 (2.8 GHz).

The rest of this paper is organized as follows. Section 2 details the related work on the replica placement problem and combination algorithm. In Sect. 3, we propose a fast solution to the replica placement problem; it divides the candidates and pipelines them on a DAPDNA-2. Section 4 evaluates the performance of our implementation. Finally, we conclude this paper in Sect. 5.

2. Replica Placement Problem

The network is represented by an undirected graph $G = (V, E)$, where V is the set of servers, and $E \subseteq V \times V$ denotes the set of network links among the servers. Each link $(u, v) \in E$ is associated with a cost $d(u, v)$ that denotes the communication cost of the link between servers u, v. We assume that the graph is connected, so that one server can connect to any other server via a path. We define the communication cost of a path as the sum of the communication cost of the links along the path. Because we assume that each server knows the nearest replica, we define $d(u, v)$ between two servers u, v to be the communication cost of the shortest path between them. Every server u has a storage cost $s(u)$, which denotes the cost to put a replica on server u. Different servers usually have different storage costs.

Figure 1 illustrates replica placement. The numbers in the circles are server indices between 0 and $n - 1$, where n is the total number of servers. The number on a link is the communication cost of the link.

Each server in the network serves multiple clients, although we don't illustrate the clients in Fig. 1. A client sends its request to its associated server, which then processes the request. If the local server has the requested data, the request is processed locally. Otherwise, the request is directed to the nearest server that has the replica. In addition, we ignore the communication cost from clients to servers because it doesn't impact the replication decision.

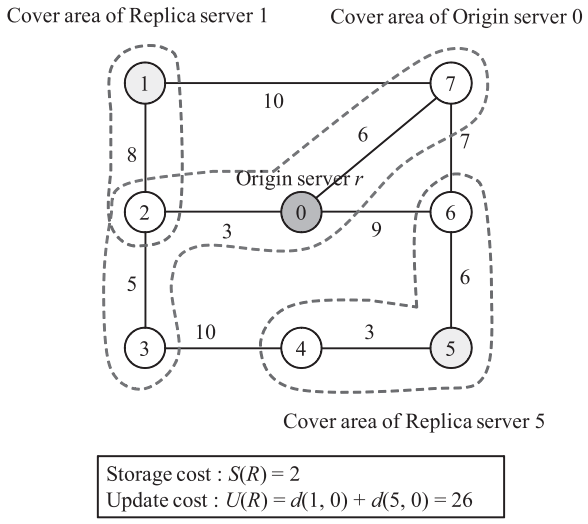


Fig. 1 Origin server and replica servers {1, 5} can cover all nodes when the quality requirement is 8.

Without loss of generality, we assume that server 0 is origin server r. Initially, only the origin server has the contents. A replica server is a server that has replicated contents. A replication strategy, $R \subseteq V - \{r\}$, is a set of replica servers.

We use the replication cost to evaluate replication strategies. The replication cost $T(R)$ of replication strategy R is defined as the sum of storage cost $S(R)$ and update cost $U(R)$.

$$T(R) = S(R) + U(R) \tag{1}$$

Storage cost: The storage cost of replication strategy R is the sum of all storage costs of the replica servers.

$$S(R) = \sum_{v \in R} s(v) \tag{2}$$

Update cost: In order to maintain data consistency, origin server r issues update requests to every replica server. We assume that there is an update distribution tree T, which connects all servers in the network. For example, we use a shortest path tree rooted at the origin server as the update distribution tree. Origin server r multicasts update requests through links on this tree until all replica servers in R receive the update request. Every node receives the update request from its parent and relays these requests to its children according to the update distribution tree.

Let $p(v)$ be the parent of node v in the update distribution tree, and T_v be the subtree rooted at node v. If $T_v \cap R \neq \phi$, link $(v, p(v))$ participates in the update multicast. As a result, the update cost is the sum of the communication costs from these links $(v, p(v))$. For example, if the replication strategy R is {1, 5} in 1, then the update cost is 11 + 15 = 26.

$$U(R) = \sum_{v \neq r, T_v \cap R \neq \phi} d(v, p(v)) \tag{3}$$

Every server u has a service quality requirement $q(u)$.

The requirement mandates that all requests generated by u will be served by a server at less than $q(u)$ communication cost. We assume that every server in the network knows the replica server nearest to itself. If a request is served by the nearest replica server within $q(u)$, the request is satisfied, otherwise, the request is violated. If all requests in the system are satisfied, the replication strategy is called feasible.

$$\min_{w \in R \cup r} d(v, w) \leq q(v) \tag{4}$$

The replica placement problem is to find the feasible replication strategy that minimizes the replication cost in Eq. (1) [10]. As an example, we assume that the quality requirement is 8 for all servers and the replication strategy is {1, 5} in Fig. 1. We can verify that the replication strategy together with the origin server can satisfy all requests within the network. The replication strategy {1, 5} covers all nodes in Fig. 1. The replica placement problem is derived from the set cover problem which is known to be NP-hard [3]. The definition of the set cover problem is as follows.

Minimum Weight Set Cover Problem: Let U be the universal set and S be 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 NP-hard because the minimum weight set cover problem is known to be NP-hard. Several greedy algorithms have been proposed to decrease the calculation time [4]–[11]. Johnson proposed a greedy algorithm against the minimum weight set cover problem [4]. This algorithm is a straightforward heuristic. The time complexity is proportional to n. In [5], [8], fan-out based replica placement algorithms were proposed. They put replicas on servers in descending order of server degree. Kangasharju et al. proved that their target replica placement optimization problem is NP-complete, and proposed some heuristic algorithms [9]. Tang et al. investigated QoS-aware replica placement problems to elucidate QoS requirements, and proposed the l-Greedy-Insert and l-Greedy-Delete algorithm [10]. They showed that the QoS-aware placement problem for replica-aware services was NP-complete. Wang et al. proposed a heuristic algorithm called Greedy-Cover [11]. Experiments indicated that the proposed algorithm found near-optimal solutions effectively and efficiently. Karlsson et al. provided a framework for evaluating replica placement algorithms [7], and compared several replica placement algorithms [6]. [6] also provides a comprehensive survey of replica placement algorithms. However, note that it has been proven mathematically that no greedy algorithm can obtain the optimal solution [3]. Therefore, to get the optimal solution, a fast exhaustive search algorithm is required.

Exhaustive search algorithms generally consist of the following three procedures.

1. Generate all solution candidates, in other words all replication strategies
2. Check each solution candidate as to whether all nodes are covered
3. Calculate the replicating cost of each solution candidate

The above procedures are executed over all of replication strategies, and the optimal solution is selected. The parallelization of procedures 2 and 3 is easily achieved because the replication strategies are completely independent in these procedures. However, the parallelization of procedure 1 is not easy, so procedure 1 is likely to become a bottleneck. Therefore, we focus on a solution candidate generation scheme to speed up the exhaustive search algorithm in this paper.

Exhaustive search algorithms to solve the Boolean Satisfiability Problem (SAT), which is an NP-hard problem as well as the set cover problem, have been implemented on FPGAs [14]–[17]. Instance-specific hardware is employed to reduce the execution time in these implementations. Thus, we have to re-generate instance-specific hardware for each problem instance, i.e. the hardware compilation, which is a significant overhead, is required. In addition, the problem instance is limited in the implementations of [15], [17] since it was assumed that the forms of the boolean expressions they contained were limited.

3. Proposed Method

Combinatorial algorithms can be applied to problems derived from the set cover problem, such as the replica placement problem. The calculation time of the replica placement problem increases rapidly with network scale. We propose a new method that generates and tests all combinations rapidly to obtain the optimal solution in a feasible time. Our proposed method divides combinations into different groups which are executed in parallel. The first data of each group are entered per clock cycle following pipeline operation. We implemented Beeler's algorithm [13], which can generate all combinations in ascending order, on DAPDNA-2.

Figure 2 shows the pipeline operation when ${}_6C_3$ is divided into 4 groups. 1st, 6th, 11th and 16th data are input data because 20 combinations are divided into 4 groups. DNA matrix outputs Data2, Data7, Data12 and Data17, which are the next input data in Fig. 2. The result of the last group is delayed by 3 clocks compared to that of the first group. The overall execution time is about 75 percent shorter than the original execution time.

There are two problems that need to be solved. First, how can we calculate the first data of each group when the

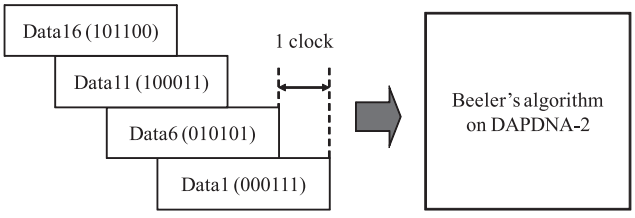


Fig. 2 First data of each group are entered per clock cycle by pipeline operation. DNA matrix outputs Data2, Data7, Data12 and Data17, which are the next input data.

combinations are divided into different groups? Beeler's algorithm can generate all combinations in ascending order but there are data dependencies. It's difficult to calculate any order pattern because the difference between neighboring patterns is not constant. We solve this problem by proposing an algorithm that can generate patterns in any order.

Second, what is the optimal number of divisions in terms of minimizing the overall number of calculation clocks needed? Increasing the number of divisions decreases the overall calculation clock number but there is a lower limit beyond which the overall clock number starts to increase. The optimal number of divisions depends on the number of combinations and the calculation clocks of Beeler's algorithm. In order to solve this problem, we tackled the theory behind the optimal number of divisions.

3.1 Beeler's Algorithm and Any-Order Pattern Algorithm

M. Beeler et al. proposed an algorithm that generates all combinations and picks k outcomes from n possibilities [13]. These combinations can be expressed in n-digit binary form. For example, 010110 represents $(2, 3, 5)$ when $n = 6$. Combinations can be ordered as follows; $(2, 3, 5) < (2, 4, 5)$ because $010110 < 011010$. Beeler's algorithm can generate all combinations from 000111 to 111000 in order. The details of the algorithm are as follows.

1. Let S_1 be the pattern in which all bits are unset except for the least significant bit of combination X.
2. $R = X + S_1$
3. Let S_2 be the pattern in which all bits are unset except for the least significant bit of combination R.
4. $S_3 = (S_2/S_1) \gg 1 - 1$
5. $Y = R|S_3$ is next to X.

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

1. $S_1 = 000010$
2. $R = X + S_1 = 010000$
3. $S_2 = 010000$
4. $S_3 = (S_2/S_1) \gg 1 - 1 = 001000 \gg 1 - 1 = 000011$
5. $Y = R|S_3 = 010011$

We propose a new algorithm that generates any order pattern in combinations sorted in ascending order. The following equation is generally true.

$$ {}_nC_k = \sum_{i=k-1}^{n-1} {}_iC_{k-1} \tag{5} $$

If you want to get m-th pattern, find x_1, which is the smallest value among the values of x satisfying Eq. (6). ${}_iC_{k-1}$ corresponds to the number of the patterns whose i-th bit is the most significant bit, and where the number of 1's between 0 and the $(i-1)$-th bit is $k-1$. Therefore, x_1 means the patterns that include the m-th pattern, and the highest bit to be set at 1 of the patterns is the x_1-th bit.

$$ \sum_{i=k-1}^{x} {}_iC_{k-1} \geq m \quad (k-1 \leq x_1 \leq n-1) \tag{6} $$

${}_{x_1}C_{k-1}$ means the number of the patterns whose x_1-th bit is the most significant and the number of 1's between 0 and $(x_1 - 1)$-th bit is $k - 1$ because the number of 1's is k in total. Hence, the x_1-th bit of the m-th pattern is 1. The m-th 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.

$$ m \to m - \sum_{i=k-1}^{x_1-1} {}_iC_{k-1} \tag{7} $$

Next, find x_2, which is the smallest value among the value of x satisfying the following inequality.

$$ \sum_{i=k-2}^{x} {}_iC_{k-2} \geq m \quad (x \leq x_1 - 1) \tag{8} $$

${}_{x_2}C_{k-2}$ represents the patterns whose highest bit to be set at 1 is the x_2-th bit and there are $k-2$ 1's between 0 and $x_2 - 1$. Hence, the x_2-th bit of the pattern is 1. $x_1, x_2, \cdots, x_k$ can be obtained by repeating k times in a similar way. Setting the corresponding bits to 1 yields get the m-th pattern.

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

$$ {}_6C_3 = {}_2C_2 + {}_3C_2 + {}_4C_2 + {}_5C_2 = 1 + 3 + 6 + 10 \tag{9} $$

Apply the Eq. (5) to ${}_4C_2$ because ${}_4C_2$ includes the 6th pattern. Hence, $x_1 = 4, m \to 2$.

$$ {}_4C_2 = {}_1C_1 + {}_2C_1 + {}_3C_1 = 1 + 2 + 3 \tag{10} $$

Apply the Eq. (5) to ${}_2C_1$ because ${}_2C_1$ includes the 2nd pattern. Hence, $x_2 = 2, m \to 1$.

$$ {}_2C_1 = {}_0C_0 + {}_1C_0 = 1 + 1 \tag{11} $$

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

3.2 Optimal Number of Divisions

Let a be the number of clocks taken to calculate any order pattern and b be the number of clocks to execute Beeler's algorithm. $b\,({}_nC_k - 1)$ clocks are required to generate all combinations and pick k outcomes from n possibilities. ${}_iC_j$ is the number of j-selections from i elements, where i, j are nonnegative integers. When we divide the combinations into 2 groups, $a + \frac{b({}_nC_k-1)}{2} + 1$ clocks are required. When we divide the combinations into 3 groups, $2a + \frac{b({}_nC_k-1)}{3} + 2$ clocks are required. When we divide the combinations into x groups in a similar way, y clocks are required as follows.

$$ \begin{aligned} y &= (x-1)a + \frac{b({}_nC_k - 1)}{x} + x - 1 \\ &= \frac{b({}_nC_k - 1)}{x} + (a+1)x - a - 1 \end{aligned} \tag{12} $$

According to a relationship between arithmetic mean and

geometric mean,

$$\begin{aligned} y &= \frac{b({}_nC_k - 1)}{x} + (a+1)x - a - 1 \\ &\geq 2\sqrt{\frac{b({}_nC_k - 1)}{x}(a+1)x} - a - 1 \\ &= 2\sqrt{b({}_nC_k - 1)(a+1)} - a - 1 \end{aligned} \tag{13}$$

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

$$x = \sqrt{\frac{b({}_nC_k - 1)}{a+1}} \tag{14}$$

This is the optimal number of divisions.

3.3 Implementation on DAPDNA-2

Let n be the number of nodes except for the origin server and $k(\leq n)$ be the number of replicas. In our implementation, $n \leq 32$ because the word size of PE is 32-bits long. For example, we generate all combinations from 0000011 to 1100000 when $n = 7, k = 2$. Each node is represented as 32-bit data. Let the i-th bit be 1 if this node covers node i. In Eq. (4), the v-th and w-th bits of node w are 1 because node w covers v. This information is called the cover data of node w. If OR between the cover data of all replica servers and that of the origin server yields 1111111, the replication strategy covers all nodes. For example, the replication strategy is node {1, 5} when the combination is 0010001. The following equations are true in Fig. 1.

$$\begin{aligned} &d(2,0) \leq q(2), \quad d(3,0) \leq q(3), \quad d(7,0) \leq q(7) \\ &d(2,1) \leq q(2), \quad d(4,5) \leq q(4), \quad d(6,5) \leq q(6) \end{aligned}$$

One strategy is node 0 (1000110), node 1 (0000011), and node 5 (0111000). This replication strategy covers all nodes because the result of the OR operation between the cover data of the these nodes equals 1111111. If several replication strategies cover all nodes, we choose the minimum-cost replication strategy.

After calculating the optimal number of divisions, our proposed algorithm consists of following 3 processes.

1. Calculate the first replication strategy of each group by using the algorithm described in Sect. 3.1.
2. Execute Beeler's algorithm.
3. Using the corresponding cover data, check that all nodes can be covered.

The result of process (1), which is executed by DAP, is stored in main memory. DNA reads this result from main memory and executes processes (2) and (3) in pipeline manner. The hardware compilation for each problem instance is not required since our implementation is not instance-specific, but application-specific. In addition, it can be generally applied to combinatorial optimization problems including the set cover problem.

To support network with more than 32 nodes, we have to make a small modification to the implementation; the algorithm remains basically the same. Several words are required to express a replication strategy and cover data. Therefore, several words are treated as one data unit in the implementation for over 32 nodes.

4. Performance Evaluation

In this section, we compare the execution time of a DAPDNA-2 (166 MHz) with that of a Pentium 4 (2.8 GHz). Let k be the number of replicas and n be the number of nodes, except for the origin server, and d be the number of partitions.

Figure 3 shows the execution time to generate all combinations when $k = 8$, in other words, 25 percent of all nodes hold replicas. This percentage is derived from the result shown in [11]. This reference shows that the average number of replicas is 25 percent. Black plots represent the conventional exhaustive search using Beeler's algorithm on the Pentium 4, and white plots represent the proposed method on the DAPDNA-2. Circle plots represent the theoretical execution time, and square plots represent the experimental execution time. Figure 3 has some margin of error between theoretical and experimental times, but both demonstrate the same overall tendency. In the proposed method, the execution time increases slowly with n because DAPDNA-2 calculates in parallel using a pipeline operation. When $n = 30$, DAPDNA-2 reduces the execution time by a factor of 40 compared to Pentium 4. It is noted that the clock frequency of DAPDNA-2 is only 1/17th that of the Pentium 4. Such large performance gain cannot not be attributed to only the difference in processor architecture. The performance gain is the result of combining the parallel processing of DRP with the proposed algorithm.

Figure 4 shows the theoretical execution time when k is 25 percent of the number of nodes, n. Let a be the calculation clock of any-order algorithm and b be the calcula-

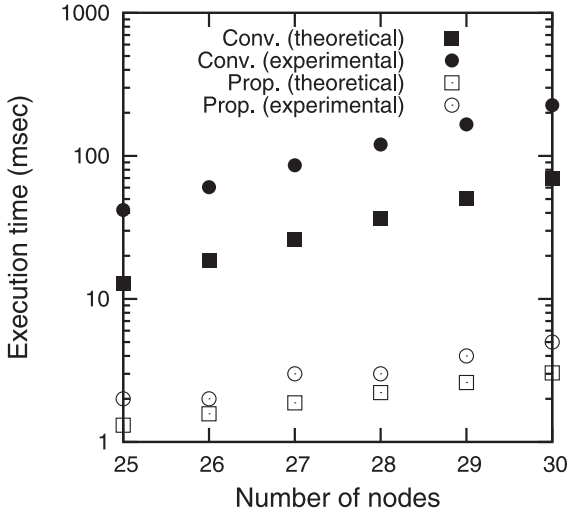


Fig. 3 DAPDNA-2 can reduce the execution time by 40 times compared to Pentium 4 when the number of nodes is 30.

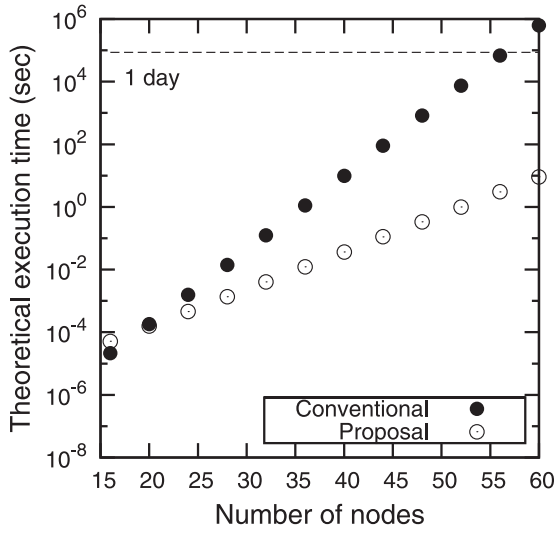


Fig. 4 Theoretical execution time versus the number of nodes when the number of replicas k is 25 percent of the number of nodes n.

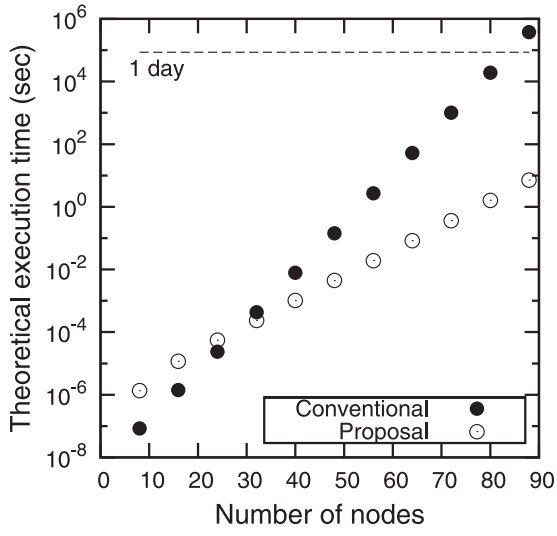


Fig. 5 Theoretical execution time versus the number of nodes when the number of replicas k is 12.5 percent of the number of nodes n.

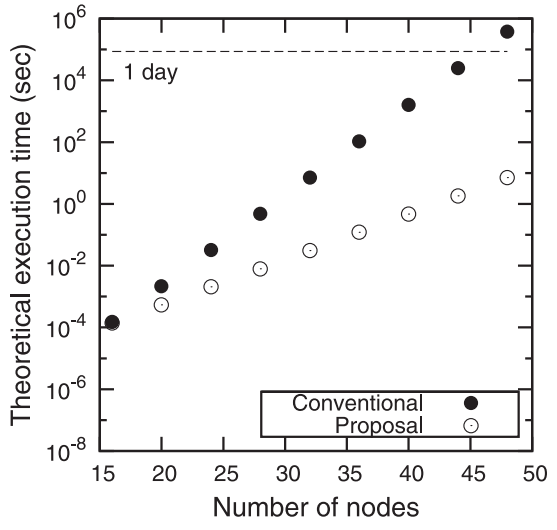


Fig. 6 Theoretical execution time versus the number of nodes when the number of replicas k is 50 percent of the number of nodes n.

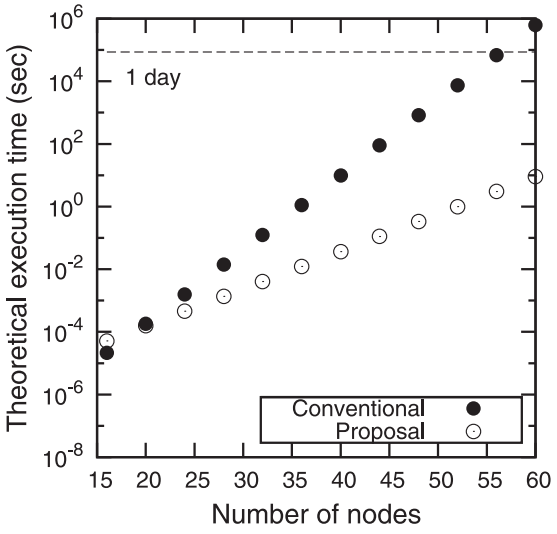


Fig. 7 Theoretical execution time versus the number of nodes when the number of replicas k is 75 percent of the number of nodes n.

tion clock of Beeler's algorithm. The measured values are $a = 330$, and $b = 33$. Let t_c be the theoretical execution time of the Pentium 4 and t_p be the theoretical execution time of the DAPDNA-2. t_c, t_p are as follows.

$$t_c = \frac{b({}_nC_k - 1)}{2.8 \times 10^9}(sec) \tag{15}$$

$$t_p = \frac{2\sqrt{b({}_nC_k - 1)(a+1)} - a - 1}{166 \times 10^6}(sec) \tag{16}$$

While the Pentium 4 requires about 7 days to generate all combinations when the number of nodes, n equals 60, the execution time of the proposed method is about 9 seconds. This is because the time complexity of proposed algorithm is $O(\sqrt{{}_nC_k})$. As a result, the proposed algorithm is scalable against the number of nodes, n. Figures 5, 6, 7 show the theoretical execution time when k is 12.5 percent, 50 percent, and 75 percent of the number of nodes, n, respectively. The dashed lines in the figures correspond to the value of 1 day. When k is equal to 12.5 percent of the number of nodes and $n = 88$, the conventional method requires over 4 days to generate all combinations. On the other hand, the execution time of proposed method is about 7 seconds. If k is 50 percent of the number of nodes and $n = 48$, the conventional method takes about 4 days, while our proposal takes only 7 seconds. The result when k is 75 percent of the number of nodes equals that when n is 25 percent of the number of nodes. This is because the execution time is a function of ${}_nC_k$ and the equation ${}_nC_k = {}_nC_{n-k}$ is always true. Until k reaches 50 percent of the number of nodes, the execution time increases. Therefore, when the value of k is small, we can extract the optimal replica placement for a large network within a certain value of the execution time.

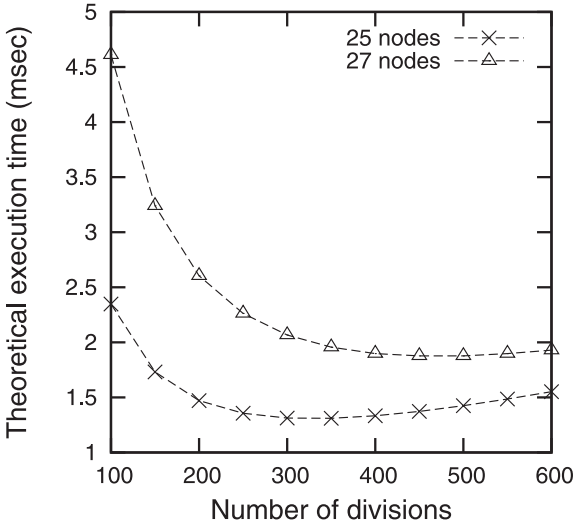


Fig. 8 Theoretical execution time versus the number of partitions.

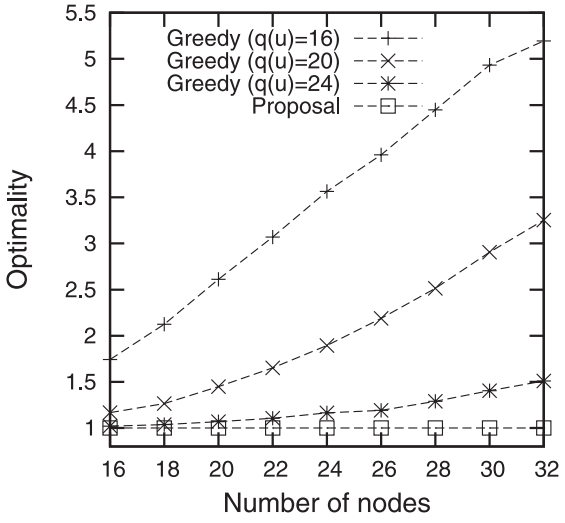


Fig. 9 Comparison of the optimality of Greedy-Cover and the proposed algorithm.

Figure 8 shows the execution time of the DAPDNA-2 versus d when $k = 8$. Cross plots represent 25 nodes and triangular plots represent 27 nodes. Optimal number of divisions is calculated by Eq. (14). $d = 328$ when $n = 25$ and $d = 472$ when $n = 27$. The execution time increases if d exceeds the optimal value.

Next, we compare the optimality of the replica placements yielded by a greedy algorithm and the proposed algorithm. The greedy algorithm employed in the evaluation is the algorithm proposed in [11], Greedy-Cover algorithm. We conducted simulations on 10000 different topologies. The topologies were generated by NetworkX library [18], and we used a random graph model (gnm_random_graph in NetworkX). It is assumed average degrees of a node is 4, and service quality requirement $q(u) = 16, 20, 24$. The cost of a link is uniformly distributed between 1 and 15. Figure 9 compares the average optimality of Greedy-cover to that of the proposed algorithm. In the evaluation, optimality is defined as follows.

$$\text{optimality} = \frac{s}{o} \tag{17}$$

where s is the number of replicas obtained by the replica placement algorithm, and o is the optimal number of replicas. From Fig. 9, the optimality of the Greedy-cover algorithm tends to increase with the number of nodes. On the other hand, the optimality of the proposed algorithm is always 1 since our proposal is based on exhaustive search, and can always obtain the optimal solution. When $q(u)$ is large, the optimality of Greedy-cover algorithm is getting near that of the proposal. It is because in case that the cover area is large the total number of replicas is decreasing in both of Greedy-cover and the proposed algorithm.

Figure 10 shows the execution time of Greedy-Cover and the proposed algorithm. The execution time is the average value over 10000 topologies and the parameters are as same as those used in the simulation of Fig. 9. The execution time of Greedy-Cover algorithm is always less than

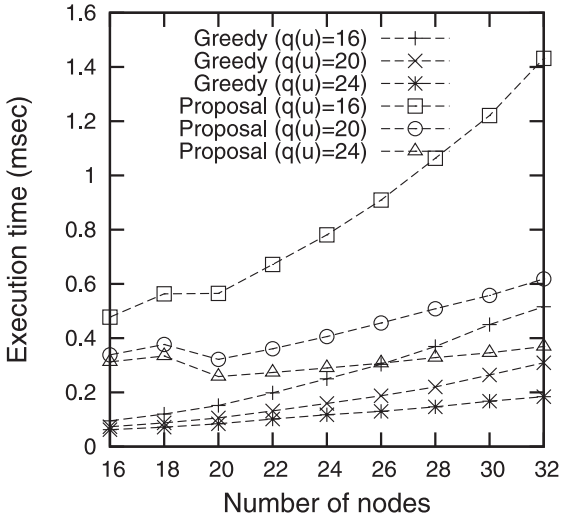


Fig. 10 Comparison of the execution time of Greedy-Cover and the proposed algorithm.

that of the proposed algorithm for the same $q(u)$. The execution time of the proposed algorithm is only 1.6 to 3.9 times as large as that of Greedy-Cover algorithm even though the proposed algorithm can always obtain the optimal solution. The factor of the execution time decreases as the number of nodes is increased. Thus, according to Fig. 9 and 10, the proposed algorithm is effective when the number of nodes is large or $q(u)$ is small.

5. Conclusion

The distribution of large contents is a promising application in the era of optical networks, but care is needed to keep the costs feasible. The content delivery network can achieve high resource efficiency in large content distribution if the placement of replica servers is optimal. In order to obtain

the optimal solution, we have developed a novel approach that is based on the use of DRPs while the conventional approaches are based on sequential processors. We have also proposed a fast calculation method for exhaustive search that well suits the DRP by fully utilizing the parallelism offered by this type of processor. Our proposed method optimally divides the combinations and subjects the pieces to pipelined processing. We propose a new algorithm that generates any order pattern in combinations that are sorted in ascending order, and derived the optimal number of divisions theoretically. In addition, we implemented the proposed algorithm on a commercially available DRP, DAPDNA-2, developed by IPFlex Inc. While the time complexity of conventional method is $O({}_nC_k)$, the time complexity of the proposed algorithm is $O(\sqrt{{}_nC_k})$.

Experiments have showed that the execution time of the proposed algorithm increases slowly as n increases because DAPDNA-2 calculates in parallel using pipeline operations. When $n = 30$, DAPDNA-2 reduces the execution time by a factor of 40 compared to that needed by a Pentium 4.

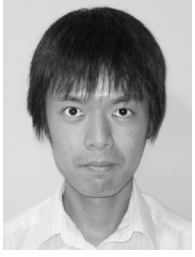
Acknowledgment

The authors would like to thank Tomomi Sato and other staff for helping with implementation on DAPDNA-2 (IPFlex Inc). This work is 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, and Grant-in-Aid for Scientific Research (19360178). One of the authors appreciates Yoshida Scholarship Foundation for its financial support.

References

[1] "Akamai." http://www.akamai.com/
[2] "Mirror imge." http://www.mirror-image.com/
[3] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.
[4] D.S. Johnson, "Approximation algorithms for combinatorial problems," J. Comput. Syst. Sci., vol.9, pp.256–278, 1974.
[5] S. Jamin, C. Jin, A.R. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the internet," INFOCOM 2001, 2001.
[6] M. Karlsson, C. Karamanolis, and M. Mahalingam, "A framework for evaluating replica placement algorithm," Tech. Rep., HP Laboratories Palo Alto, Aug. 2002.
[7] M. Karlsson and M. Mahalingam, "Do we need replica placement algorithms in content delivery netowrks?," The International Workshop on Web Content Caching and Distribution (WCW), pp.117–128, Aug. 2002.
[8] P. Radoslavov, R. Govindan, and D. Estrin, "Topology-informed internet replica placement," Comput. Commun., vol.25, no.4, pp.384–392, March 2002.
[9] J. Kangasharju, J. Roberts, and K.W. Ross, "Object replication strategies in content distribution networks," Comput. Commun., vol.25, no.4, pp.376–383, March 2002.
[10] X. Tang and J. Xu, "Qos-aware replica placement for content distribution," IEEE Trans. Parallel Distrib. Syst., vol.16, no.10, pp.921–932, Oct. 2005.
[11] H. Wang, P. Liu, and J.J. Wu, "A qos-aware heuristic algorithm for replica placement," Grid Computing 7th IEEE/ACM International Conference, pp.96–103, Sept. 2006.
[12] "IPFlex dynamically reconfigurable processor, DAPDNA-2." http://www.ipflex.com/, 2005.
[13] M. Beeler, R.W. Gosper, and R. Schroeppel, "Hakmem," http://www.inwap.com/pdp10/hbaker/hakmem/hakmem.html, Sept. 1972.
[14] M. Platzner and G.D. Micheli, "Acceleration of satisfiability algorithms by reconfigurable hardware," 8th International Workshop on Field Programmable Logic and Applications (FPL98), pp.69–78, 1998.
[15] P. Zhong, M. Martonosi, P. Ashar, and S. Malik, "Using configurable computing to accelerate boolean satisfiability," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.18, no.6, pp.861–868, June 1999.
[16] M. Abramovici and J.T.D. Sousa, "A SAT solver using reconfigurable hardware and virtual logic," J. Automated Reasoning, vol.24, no.1-2, pp.5–36, Feb. 2000.
[17] T. Suyama, M. Yokoo, H. Sawada, and A. Nagoya, "Solving satisfiability problems using reconfigurable computing," IEEE Trans. Very Large Scale Integr. Syst., vol.9, no.1, pp.109–116, Feb. 2001.
[18] "NetworkX." http://networkx.lanl.gov/

Sho Shimizu received the B.E. and M.E. degrees from Keio University, Japan, in 2005 and 2007, respectively. He is currently working toward the Ph.D. degree in Graduate School of Science and Technology, Keio University, Japan. His research interests include network architecture and traffic engineering on the next generation optical network. In 2007, he became a research assistant of Keio University Global COE Program, "High-level Global Cooperation for Leading-edge Platform on Access Spaces" by Ministry of Education, Culture, Sports, Science and Technology, Japan. He is a student member of the IEEE and the OSA.



Hiroyuki Ishikawa received B.E. and M.E. degrees from Keio University, Japan, in 2006 and 2008, respectively. His current research interests are network algorithms and their hardware implementation on dynamically reconfigurable processors. He is currently working for Kansai Electric Power Co. Inc.

Yutaka Arakawa was born in Fukuoka, Japan, in 1977. He received his B.E., M.E., and Ph.D., from Keio University, Japan, in 2001, 2003, and 2006, respectively. Since 2001, he has been researching optical network architectures and traffic engineering, especially for Optical Burst Switching. From 2004 to 2006, he was a research assistant of Keio University COE Program in "Optical and Electronic Device on Access Network" by Ministry of Education, Culture, Sports, Science and Technology, Japan. He is now assistant in the Yamanaka Lab. in Keio University, and is mainly engaged in research on photonic network architectures, traffic engineering, and network applications. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) of USA.

Naoaki Yamanaka graduated from Keio University, Japan where he received B.E., M.E. and Ph.D. degrees in engineering in 1981, 1983 and 1991, respectively. In 1983 he joined Nippon Telegraph and Telephone Corporation's (NTT's) Communication Switching Laboratories, Tokyo Japan, where he was engaged in the research and development of a high-speed switching system and high-speed switching technologies for Broadband ISDN services. Since 1994, he has been active in the development of ATM-based backbone networks and systems including Tb/s electrical/optical backbone switching as NTT's Distinguished Technical Member. He is now researching future optical IP networks, and optical MPLS router systems. He is currently professor of the Department of Information and Computer Science, Faculty of Science and Technology, Keio University. He has published over 112 peer-reviewed journal and transaction articles, written 82 international conference papers, and been awarded 174 patents including 17 international patents. Dr. Yamanaka received Best of Conference Awards from the 40th, 44th, and 48th IEEE Electronic Components and Technology Conference in 1990, 1994 and 1998, TELECOM System Technology Prize from the Telecommunications Advancement Foundation in 1994, IEEE CPMT Transactions Part B: Best Transactions Paper Award in 1996 and IEICE Transaction Paper award in 1999. Dr. Yamanaka is Technical Editor of IEEE Communication Magazine, Broadband Network Area Editor of IEEE Communication Surveys, Former Editor of IEICE Transaction, TAC Chair of Asia Pacific Board at IEEE Communications Society as well as Board member of IEEE CPMT Society. Dr. Yamanaka is an IEEE Fellow.

Kosuke Shiba received the bachelor degree in Science in Mathematics from University of Tsukuba, Japan, in 1984, received the master degree in Information Science and Electronics from the University of Tsukuba, in 1986. From 1986 to 1992, he worked at Casio Computer Co., Ltd. He was an engineer at Rohm Co., Ltd, from 1992 to 1997, being responsible for the development of a VLSI for MPEG1 decoders and image processors. He worked as a manager at Mentor Graphics Japan Co., Ltd. from 1997 to 2000. He was a director of IPFlex Inc, being responsible for the management of development activities on dynamically reconfigurable processors, application development framework software, and applications for the processors. He is also responsible for the research of dynamically reconfigurable architectures. He is a member of IEEE.