Fast Replica Allocation Method by Parallel Calculation on DAPDNA-2

Hiroyuki ISHIKAWA*, Sho SHIMIZU*, Yutaka ARAKAWA*, Naoaki YAMANAKA* and Kosuke SHIBA[†]

*Department of Information and Computer Science, Faculty of Science and Technology, Keio University,

3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223-8522, Japan

Email: ishikawa@yamanaka.ics.keio.ac.jp

[†]IPFlex Inc. Kamiosaki 2–27–1, Sun felista Meguro 6F, Shinagawa-ku, Tokyo, 141–0021, Japan

Email: shiba@ipflex.com

Abstract—Replica placement problem is to select a subset from a group of potential nodes to put replicas in Content Distribution Network (CDN). It is derived from the set cover problem which is known to be NP-hard. So it is difficult to calculate the large-scale replica placement problem on a program counter-based processor. Several greedy algorithms are proposed in order to decrease calculation time. However, it was proved mathematically that no greedy algorithm can obtain the optimal solution.

This paper proposes a fast calculation method of the replica placement problem, which is implemented on reconfigurable processor DAPDNA-2 of IPFlex Inc. Our proposed method divides the combination optimally and performs pipeline operation. Beeler's algorithm can calculate all combinations in ascending order but it has data dependence. It's difficult to calculate any pattern because each data increases irregularly. In order to solve this problem, we propose the new algorithm that generates any order pattern. In addition, the optimal number of partitions depends on the number of combinations and calculation clocks of Beeler's algorithm. In order to solve this problem, we think about the optimal division number in theory.

While the time complexity of conventional method is proportional to the number of combinations, that of proposed method is proportional to the square root of the number of combinations. Experimental results show that the proposed algorithm reduces the execution time by 40 times compared to Intel Pentium 4 (2.8GHz).

I. INTRODUCTION

CDN has been used to deliver the contents from the origin server to geographically distributed nodes, which send contents to the clients. In CDN, it's important to reduce the load on the origin server and network. In order to deliver contents efficiently, CDN copies contents and places replicas on several servers. The placement decision has to be made on per content and be made dynamically according to user requests.

One of the decision problems is replica placement, which selects a subset from a group of potential nodes to put replicas. Replica placement is usually done by a system administrator and many CDN service providers tend to acquire as many replication sites as possible.

The distance between two nodes is used as a metric for quality assurance in CDN. A request must be answered by a server within the distance specified by the request because all clients want to download contents during the allotted time period. Every node knows the nearest server that has the replica and the request takes the shortest path to reach the server. The goal is to find a replica placement that satisfies all requests without violating any range constraint, and minimize the update and storage cost at the same time. If storage cost is 1 for all nodes, the number of replicas must be minimized.

Replica placement problem is derived from the set cover problem which is known to be NP-hard [1]. Therefore, calculation time increases rapidly when the network scale is large. Several greedy algorithms are proposed in order to decrease calculation time [2]. However, it was proved mathematically that no greedy algorithm can obtain the optimal solution.

In this paper, we propose a fast calculation method of the set cover problem, which is implemented on reconfigurable processor DAPDNA-2 of IPFlex Inc [3]. DAPDNA-2 consists of DAP (Digital Application Processor), a high-performance RISC core, and DNA (Digital Network Architecture), a dynamically reconfigurable twodimensional matrix. The DNA is embedded in an array of 376 PE (Processing Elements), which are comprised of computation units, memory, synchronizers, and counters. The PE Matrix circuitry can be reconfigured freely into the structure, which is suitable for application on demand.

It is difficult to calculate the large-scale replica placement problem on a program counter-based processor. Our proposed algorithm divides the combination optimally and performs pipeline operation. While the time complexity of conventional method is $O({}_{n}C_{k})$, the time complexity of proposed algorithm is $O(\sqrt{{}_{n}C_{k}})$. Experimental results show that the proposed algorithm reduces the execution time by 40 times compared to Intel Pentium 4 (2.8GHz).

The rest of this paper is organized as follows. Section II denotes the related work of this research about replica placement problem and combination algorithm. In Section III, we propose a fast calculation method of the replica placement problem which divides all combinations optimally and performs pipeline operation on DAPDNA-2. Section IV evaluates the performance of our implementation. Finally, we conclude this paper in Section V.

II. RELATED WORKS

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 two servers u, v. We assume that the graph is connected, so

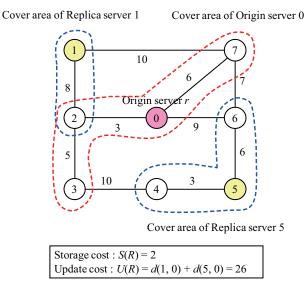


Fig. 1. Origin server and Replica servers $\{1, 5\}$ can cover all nodes when the quality requirement is 8.

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), that denotes the cost to put a replica on server u. The storage cost on different servers may be different.

Figure 1 shows the illustration of 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 services multiple clients, although we don't illustrate clients in Figure 1. A client sends its requests to its associeted server, then the server processes the request. If the client's requests can be served by the server, i.e., the local server has the requested data, the requests will be processed locally. Otherwise, the request will be directed to the nearest server that has the replica. In addition, because the communication cost from the clients to servers doesn't affect the replication decision, we ignore the communication cost from clients to servers.

There is the origin server r in Figure 1. Without loss of generality, we assume that server 0 is the origin server. Initially, only the origin server has the data. A replica server is a server that has a copy of the original data. 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 a replication strategy R is defined as the sum of the storage cost S(R) and the update cost U(R).

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

Storage cost: The storage cost of a replication strategy R

is the sum of all storage cost of the replica servers.

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

Update cost: In order to maintain data consistency, the 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. The origin server r multicasts update requests through links on this tree until all replica servers in R receive the update requests. Every node receives update requests 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 $T_v \cap R \neq \phi$, the link (v, p(v)) participates 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 Figure 1, then the update cost is 11 + 15 = 26.

$$U(R) = \sum_{v \neq r, T_v \cap R \neq \phi} d(v, p(v))$$
(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 within q(u) communication cost. We assume that every server in the network knows the nearest replica server from 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.

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

The replica placement problem is to find the feasible replication strategy such that the replication cost in Equation (1) is minimized [4]. For example, we assume that the quality requirement is 8 for all servers and the replication strategy is $\{1, 5\}$ in Figure 1. We can verify that the replication strategy together with origin server can satisfy all requests within the network. The replication strategy $\{1, 5\}$ covers all nodes in Figure 1. Replica placement problem is derived from the set cover problem which is known to be NP-hard. The difinition 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 S such that the weight is minimized and $\bigcup_{S \in S} S = U$ is satisfied.

Replica placement problem is NP-hard because Minimum Weight Set Cover Problem is known to be NP-hard. Some greedy algorithms have been proposed.

David S.Johnson proposed the greedy algorithm for Minimum Weight Set Cover Problem [5]. This algorithm is a straightforward heuristic, requiring time proportional to n. Several greedy algorithms have been proposed in order to decrease calculation time [2]. However, note that no greedy algorithm can obtain the optimal solution.

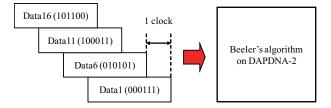


Fig. 2. First data of each group are entered per clock cycle by pipeline operation. DNA matrix outputs Data2, Data7, Data12 and Data17, which are the next input data.

III. PROPOSED METHOD

Combinatorial algorithm can be applied to the problem which is derived from the set cover problem, such as replica placement problem. The calculation time of replica placement problem increases rapidly when the network scale is large. We propose the new method that generates all combinations fast because no greedy algorithm can obtain the optimal solution. Our proposed method divides the combination into different groups which are executed in parallel. First data of each group are entered per clock cycle by pipeline operation. We implemented Beeler's algorithm, which can generate all combinations in ascending order on DAPDNA-2.

Figure 2 shows the pipeline operation when ${}_{6}C_{3}$ 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 Figure 2. The result of the last group get delayed in 3 clocks compared with the one of the first group. The whole execution time is about quarter compared with original execution time.

There are two problems that need to be solved. First, how can we calculate the first data of each group when combination is divided into different groups? Beeler's algorithm can generate all combinations in ascending order but it has data dependence. It's difficult to calculate any order pattern because each data increases irregularly. In order to solve this problem, we propose the new algorithm that generates any order pattern.

Second, what is the optimal division number which minimizes whole calculation clocks? The more division number increases, the more whole calculation clocks decreases. However, whole calculation clocks increases the other way around when the division number exceeds a certain value because the results get delayed in a clock compared with the one of previous group. The optimal division number depends on the number of combinations and calculation clocks of Beeler's algorithm. In order to solve this problem, we think about the optimal division number in theory.

A. Beeler's algorithm and Any-order pattern algorithm

M.Beeler, R.W.Gosper, R.Schroppel proposed an algorithm that generates all combinations picking k outcomes from n possibilities [6]. These combinations can be expressed in n-digit binary form. For example, 010110 represents (2,3,5) when n = 6. Combinations can be ordered in this way. (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 is as follows.

- 1) Let S_1 be what all bits are unset except for the least significant 1 of a combination X.
- 2) $R = X + S_1$
- 3) Let S_2 be what all bits are unset except for least significant 1 of R.
- 4) $S_3 = (S_2/S_1) >> 1-1$
- 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) $S_3 = (S_2/S_1) >> 1 1 = 001000 >> 1 1 = 0000011$
- 5) $Y = R|S_3 = 010011$

We propose the new algorithm that generates any order pattern in combinations which are sorted in ascending order. Generally, the following equation is true.

$${}_{n}C_{k} = \sum_{i=k-1}^{n-1} {}_{i}C_{k-1}$$
(5)

If you want to get m-th pattern, find the smallest x_1 which satisfies the following inequation.

$$\sum_{k=k-1}^{x_1} {}_i C_{k-1} \ge m \quad (k-1 \le x_1 \le n-1) \tag{6}$$

 x_1C_{k-1} means the patterns whose most significant one is x_1 bit and there are k-1 ones between 0 and x_1-1 because there are k ones in total. Hence, x_1 bit of m-th pattern is 1. m-th pattern corresponds $m - \sum_{i=k-1}^{x_1-1} {}_iC_{k-1}$ -th in x_1C_{k-1} . Replace m as follows.

$$m \to m - \sum_{i=k-1}^{x_1-1} {}_i C_{k-1}$$

Next, find the smallest x_2 which satisfies the following inequation.

$$\sum_{i=k-2}^{x_2} {}_i C_{k-2} \ge m \quad (x_2 \le x_1 - 1)$$

 x_2C_{k-2} means the patterns whose most significant one is x_2 bit and there are k-2 ones between 0 and x_2-1 . Hence, x_2 bit of the pattern is 1. x_1, x_2, \dots, x_k can be obtained by repeating k times in a similar way. Set corresponding bit to 1, and you can get the m-th pattern.

For example, 6th pattern (m = 6) in ${}_{6}C_{3}$ can be obtained as follows.

$$_{6}C_{3} = _{2}C_{2} + _{3}C_{2} + _{4}C_{2} + _{5}C_{2} = 1 + 3 + 6 + 10$$

Apply the equation (5) to ${}_{4}C_{2}$ because ${}_{4}C_{2}$ includes 6th pattern. Hence, $x_{1} = 4, m \rightarrow 2$.

$$_{4}C_{2} = _{1}C_{1} + _{2}C_{1} + _{3}C_{1} = 1 + 2 + 3$$

Apply the equation (5) to ${}_{2}C_{1}$ because ${}_{2}C_{1}$ includes 2nd pattern. Hence, $x_{2} = 2, m \rightarrow 1$.

$$_{2}C_{1} =_{0} C_{0} +_{1} C_{0} = 1 + 1$$

1st pattern corresponds $_0C_0$. Hence, $x_3 = 0$. Set corresponding bit to 1, and the 6th pattern can be obtained, 010101.

B. Optimal division number

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

$$y = (x - 1)a + \frac{b({}_{n}C_{k} - 1)}{x} + x - 1$$
$$= \frac{b({}_{n}C_{k} - 1)}{x} + (a + 1)x - a - 1$$

According to a relationship between arithmetic mean and geometric mean,

$$y = \frac{b({}_{n}C_{k} - 1)}{x} + (a+1)x - a - 1$$

$$\ge 2\sqrt{\frac{b({}_{n}C_{k} - 1)}{x}(a+1)x} - a - 1$$

$$= 2\sqrt{b({}_{n}C_{k} - 1)(a+1)} - a - 1$$

We have equality if and only if $\frac{b(aC_k-1)}{x} = (a+1)x$. Hence

$$x = \sqrt{\frac{b(nC_k - 1)}{a+1}} \tag{7}$$

This is the optimal division number.

C. 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 one word is 32-bit length in PE. For example, we generate all combinations from 0000011 to 1100000 when n = 7, k = 2. Each node is represented by 32-bit data. Let *i*-th bit be 1 if this node covers node *i*. In expression 4, v, w-th bit of node w is 1 because w covers v. If OR between 2 replica and the origin server equals 1111111, the replication strategy covers all nodes. For example, the replication strategy is node $\{1, 5\}$ when the combination is 0010001. Now, the following equations are true in Figure 1.

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

Node 0 represents 1000110, node 1 represents 0000011, and node 5 represents 0111000. This replication strategy covers all nodes because OR between 3 data equals 1111111. If some replication strategies covers all nodes, we choose the minimum-cost combination.

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

1) Calculate *m*-th pattern according to the optimal division number.

- 2) Execute Beeler's algorithm.
- 3) Using 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 execute process (2), (3) by pipeline operation.

IV. PERFORMANCE EVALUATION

In this section, we compare the execution time of DAPDNA-2 (166MHz) with that of Pentium 4 (2.8GHz). 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. Black plots represent conventional method on Pentium 4, and white plots represent proposed method on DAPDNA-2. Circle plots represent theoretical execution time, and square plots represent experimental execution time. Figure 3 has a margin of error between theoretical and experimental time but increasing tendency is almost the same. In the proposed method, the execution time increase slowly as n increases because DAPDNA-2 calculates in parallel using a pipeline operation. When n = 30, DAPDNA-2 reduces the execution time by 40 times compared to Pentium 4.

Figure 4 shows the execution time of DAPDNA-2 versus d when k = 8. Cross plots represent 25 nodes and triangular plots represent 27 nodes. Optimal division number is calculated by expression 7. d = 328 when n = 25 and d = 472 when n = 27. The execution time reduces with increasing of d but increases when d is larger than the optimal number. This is because data are output per clock cycle.

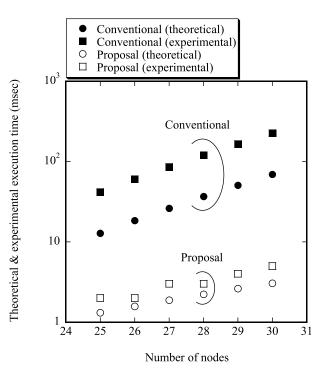


Fig. 3. DAPDNA-2 can reduce the execution time by 40 times compared to Pentium 4 when the number of nodes is 30.

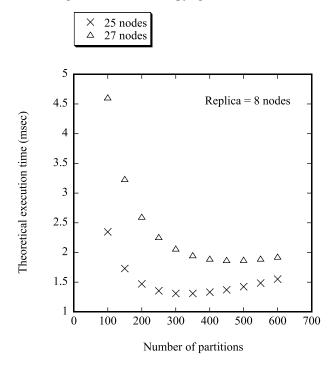


Fig. 4. Theoretical execution time versus the number of partitions

V. CONCLUSION

In order to obtain the optimal solution of Replica Placement in CDN, we have proposed a fast calculation method with reconfigurable processor DAPDNA-2 of IPFlex Inc. Our proposed method divides the combination optimally and performs pipeline operation. We have proposed the new algorithm that generates any order pattern in combinations which are sorted in ascending order and derived the optimal division number in theory. While the time complexity of conventional method is $O({}_{n}C_{k})$, the time complexity of proposed algorithm is $O(\sqrt{{}_{n}C_{k}})$.

Experimental results have showed that the execution time of the proposed algorithm increases slowly as n increases because DAPDNA-2 calculates in parallel using a pipeline operation. When n = 30, DAPDNA-2 reduces the execution time by 40 times compared to 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 Grantin-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

- M.R.Garey, D.S.Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, 1979.
- [2] Hsiangkai Wang, Pangfeng Liu, Jan-Jan Wu, "A QoS-Aware Heuristic Algorithm for Replica Placement," Grid Computing 7th IEEE/ACM International Conference, pp.96-103, September 2006.
- [3] IPFlex Inc. (http://www.ipflex.com)
- [4] Xueyan Tang, Jianliang Xu, "QoS-Aware Replica Placement for Content Distribution," IEEE Transactions on parallel and distributed systems, vol.16, No.10, pp.921-932, October 2005.

- [5] David S.Johnson, "Approximation algorithms for combinatorial problems," Journal of Computer and System Science, pp.256-278, 1974.
- [6] M.Beeler, R.W.Gosper, R.Schroeppel, HAKMEM
- (http://www.inwap.com/pdp10/hbaker/hakmem/hakmem.html)