L-015

A Knowledge-Based File Allocation Method for Real-Time Environments

Akiko Nakaniwa[†] Wesley W. Chu[‡] Hiroyuki Ebara[†] Hiromi Okada[†]

1. Introduction

Advances in network technology have made it possible to provide various kinds of content services such as real-time video, audio, etc., which have grown in interests. In distributed networks, one of the most important problems is to efficiently allocate content files to geographically separated database severs in terms of cost, response time, reliability, and other constraints. In such environments where the access patterns from users to contents changes, such as the popularity of video contents, it is also essential to dynamically reallocate files according to the temporal load fluctuation. In this research, we consider the file allocation problem under real-time response requirements.

Conventionally, the file allocation problem has been solved using the 0-1 integer programming method[1][4][5][6]. However, in this method, all the combination of files and servers are considered as the search space and thus difficult for solutions in real-time for large scale problems, even when using greedy algorithms [4][5][6].

In this research, we propose a new knowledge-based file allocation method for real-time environments. We apply several file allocation rules for providing approximate allocation rather than performing complete search for allocation. Moreover, we apply A* algorithm for allocation, which is well-known for its performance in terms of its good balances between the accuracy of the solution and the computing time [3]. We carry out the performance comparison with the conventional 0-1 integer programming method and the results show that our proposed rule-based A* algorithm drastically reduce the computing time.

2. Optimal File Allocation Problem

2.1 Problem Definitions

In distributed networks, one of the most important problems is how to assign files to servers that are geographically separated. Files must be stored in at least one of the servers, as well as be available for any users. It will be preferable to allocate copies of important or popular files to several servers to reduce the communication cost and response time. However, the duplicated storage of the same files in the system also introduces some problems, such as the trade-off between the communication cost for file updates and the additional storage cost, the overhead of keeping the data consistent, and so on. Also, there are trade-offs between cost, response time, and reliability. We should consider the optimal file allocation problem in view of these problems.

In this research, we solve the optimal file allocation problem such that the operating cost (communication and update costs) is minimized subject to a set of constraints, such as the storage capacity of each node, the availability of files and response time requirements of the processing task. Since the storage costs has reduced significantly in recent years and much lower than the communication cost, it can be considered negligible.

2.2 System Model

Let us consider a distributed network that consists of n nodes and l links. Each node is denoted by N_i and each link between nodes N_i and $N_{i'}$ is denoted by $L_{i,i'}$. Let the storage capacity of a node N_i be B_i and the link capacity of a link $L_{i,i'}$ be $C_{i,i'}$. The number of hops between nodes N_i and $N_{i'}$ is denoted by $e_{i,i'}$ and the shortest distance between nodes N_i and $N_{i'}$ can be calculated and is denoted by $h_{i,i'}$. The number of distinct files is m, and each file is denoted by M_k. The size of a file M_k is denoted by F_k. Each node can store these files as long as the total size of files does not exceed the storage capacity. The number of replicated copies of a file M_k is r_k. The access frequency for a file M_k at a node N_i is denoted by a_{ik} and the update probability after each access is P_u. We also assume that the update file size for a file M_k is f_k. When users access each file, there is a response time requirement for each query and the time requirement for accessing file M_k at node N_i is denoted by TR_{ik}. Moreover, the service rate of each node and the failure rate of each node and link is denoted by μ_i, $\lambda_i, \lambda'_{i,i'}$ respectively. This information is useful to estimate system reliability and response time.

2.3 0-1 Integer Programming Method

File allocation problems in distributed networks have been solved using the 0-1 integer programming model[1][4][5][6] as follows.

In the 0-1 integer programming method, we define 0-1 variables on the allocation of files.

$$X_{ik} = \begin{cases} 1 & (\text{the file } M_k \text{ is stored in the node } N_i) \\ 0 & (\text{otherwise}) \end{cases}$$

The optimal file allocation problem is formulated as a 0-1 integer programming model by using these 0-1 variables. In this research, the objective function is the cost and the constraints is the response time and reliability. Then, the cost, response time, and reliability are formulated using X_{ik} respectively.

The advantage of this method is the accuracy of its solutions. In [6], it is shown that the average accuracy is less than 0.3% even when using approximate methods (In the case that the number of nodes is 5 and the number of distinct files is 4). On the other hand, in this method, the search space consists of all the combination of files and nodes, and thus as the number of files and nodes increases, the computing time increases exponentially. Therefore, when using exact methods such as the exhaustive search and the branch and bound, which can both find the exact optimal solution, these methods are only applicable for small-scale systems. Even when using approximate methods such as the greedy algorithm and the tabu search, it is impossible to solve the file allocation problem in large-scale systems during real-time.

[†] Kansai University

[‡] University of California, Los Angeles

3 Knowledge-Based File Allocation Method

In this section, we propose a knowledge-based file allocation algorithm for time-critical environments.

3.1 Rule-Based Selection of File Allocation Plans

Our heuristic algorithm is characterized by rule-based pruning of the original search space. This algorithm prunes the original search space by using a set of rules. As a result, only a subset of file allocation plans are searched, rather than all the possible file allocation plans.

The following is a set of rules that is used in this research for reducing the original search space:

(R1)Select the node to allocate the file such that it has the highest reliability

(R2)Select the node to allocate the file such that it has the most critical time requirement for this file

(R3)Select the node to allocate the file such that it has the highest access frequency for this file

We apply these rules to determine each file allocation plan.

The candidate file allocation plans are the sets of nodes that satisfy one or more of these rules. We may select several nodes for each rule, for example, for the third rule we can select not only the node that has the highest access frequency, but also the second highest one, the third highest one, … , etc. In this case, we need to determine how many cases we allow for each rule. Each file allocation plan is the combination of the nodes that are selected based on these rules.

In the rule-based file allocation method, the more file allocation plans, the more computing time will be needed. It is important to limit the number of file allocation plans. Several ways can be considered to limit it. In this research, we define the cost as the criterion for that. We first create all possible file allocation plans for each file, and then select the limited number of file allocation plans based on the criterion. The appropriate number of file allocation plans to select should be determined in considering the trade-off between the quality of the solution and the computing time.

3.2 A* Algorithm

Our heuristic algorithm applies an A* algorithm [3] to search for the optimal solution in the state-space search tree, which consists of the set of states selected by the rules. The A* algorithm prunes the expanded tree more effectively than the Branch and Bound algorithm because at each step it also uses an estimation of the cost incurred in getting from the step to the final solution of the problem.

The optimal file allocation problem is formulated as a state-space search problem. Each state description is denoted by a node in the state-space search tree. A problem starts from the initial state s_0 and expands states one by one (expanding a state s means generating all immediate successors of each node). In this algorithm, the numerical value $f(s)$ is assigned to each state, consisting of two terms:

$$f(s) = g(s) + h(s)$$

where $g(s)$ is the known cost incurred in getting from s_0 to s, and $h(s)$ is a heuristic function that estimates the additional cost that will be incurred in getting from s to a final state (i.e., the solution of the problem.) To choose which state to expand next, the state s having the smallest value for $f(s)$ is always selected for node expansion. The performance of A* algorithm depends on the choice of $h(s)$. The better this function estimates the cost of the final state, the faster the algorithm. This brings up the issue of admissibility of heuristic functions. The heuristic function $h(s)$ is *admissible* if it always returns a lower bound on the additional cost that will be incurred in getting from s to a final state. If $h(s)$ is admissible, then the A* algorithm is guaranteed to find a final state s_F (leaf node in a state space search tree) such that the cost of getting from $s0$ to s_F is minimal among all final states.

3.3 Applying A* Algorithm to Optimal File Allocation Problems

To formulate the optimal file allocation problem as a state-space search problem, each possible file allocation plan is defined as a state in the search space. Let all file allocation plans for each file M_k be $\mathbf{P_k} = \{P_{k,1}, P_{k,2}, \ldots, P_{k,x_k}\}$, where $P_{k,j}$ is the jth allocation plan for the file M_k and x_k is the number of possible allocation plans for the file M_k. Each allocation plan $P_{k,j}$ is a set of nodes and includes all nodes that store an original or a replicated copy of the file M_k in this plan. That is $P_{k,j} = \{N_{j,1}, N_{j,2}, \ldots N_{j,n(j_k)}\}$ where $N_{j,l}$ is the lth node and $n(j_k)$ is the number of the nodes in this allocation plan. The state-space is constructed as follows:

1. Every state s is an m-tuple $<P_{1,j^1}, P_{2,j^2}, \ldots, P_{m,j^m}>$, where for each k, either $P_{k,j^k} \in \mathbf{P_k}$ (in which case s suggests that a plan P_{k,j^k} is used to allocate a file M_k), or else P_{k,j^k} = NULL (in which case s does not suggest any plan for a file M_k yet). The cost of s, denoted by $g(s)$, is the operating cost required when allocating files according to the allocation plans in the state s.

2. The initial state is the state s_0 = <NULL, NULL, ... ,NULL>, and the states $s_F = <P_{1,j^1}, P_{2,j^2}, \ldots, P_{m,j^m}>$ with $P_{k,j^k} \neq$ NULL, for all k, are the final states.

3. Given a state $s = <P_{1,j^1}, P_{2,j^2}, \ldots, P_{m,j^m}>$, let

$$next(s) = \begin{array}{ll} \min\{k \mid P_{k,j_k} = \mathrm{NULL}\} & \text{if } \{k \mid P_{k,j_k} = \mathrm{NULL}\} \neq 0 \\ k+1 & \text{otherwise.} \end{array}$$

4. Let the state s have at least one NULL entry and a = $next(s)$, then the immediate successors of s include every state $s' = <P_{1,j'^1}, P_{2,j'^2}, \ldots, P_{m,j'^m}>$ satisfying the following properties:

$$P_{k,j'_k} = P_{k,j_k} \quad \text{for } 1 <= k < a;$$
$$P_{a,j'_a} \in \mathbf{P_a};$$
$$P_{k,j'_k} = \mathrm{NULL} \quad \text{for } a < k <= m.$$

The cost of the transition from s to s' is the additional cost needed to process the new plan P_{a,j'^a}, given the (intermediate or final) results of processing the plans in the state s.

3.4 Rule-Based A* Algorithm

In our heuristic algorithm, we first create the state-space search tree which consists of the set of states selected by the rules, and then search this search tree in a similar way as the A* algorithm.

In Fig.1, we show an example of the state-space search tree. Each node in the search tree corresponds to each file allocation plan determined by rules. Here are the steps of our heuristic file allocation algorithm:

1. Create the set of file allocation plans for each file based on rules.

2. Create an array OPEN and insert the initial state s_0 into OPEN as the first factor.

3. Repeat the procedures from 3.1 to 3.3 in order of the file size from largest to smallest until the file allocation for all files is determined.

   3.1 Remove from OPEN the state s that has the smallest value for $f(s)$. If there is more than one such node, then we select the one at the deepest level.

   3.2 Generate the successor states of s, and insert them into OPEN.

   3.3 Check if the restrictive condition of storage capacity is satisfied. If not, remove it from OPEN.

4. Check if all of the restrictive conditions are satisfied. If so, return s as the optimal solution.

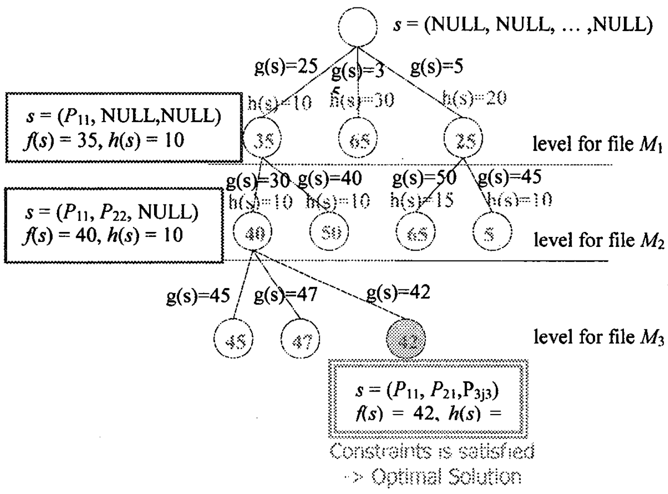

Fig.1 An example of the state-space search tree
(# of distinct files = 3, $M_1>M_2>M_3$)

3.5 Evaluation Function and Constraints

3.5.1 Evaluation Function

As in the A* algorithm, our heuristic file allocation algorithm uses the evaluation function for a state s. Since the objective function is the operating cost, we evaluate $g(s)$ and $h(s)$ by the operating cost function.

The operating cost consists of the communication cost C_t and the update cost C_u. Here, $C_t(h_{i,j})$ is the communication cost coefficient, which depends on the number of hops between nodes, and C_u is the update cost coefficient which is common for all files. The cost function of a jth file allocation plan for a file M_k is as follows:

$$C\left(P_{k,j_k}\right)=\sum_{i=1}^{n}\left(a_{ik}C_t\left(h_{i,j_{\min(i)}}\right)F_k+a_{ik}P_uC_uf_kr_k\right)$$

Here, $j_{\min(i)}$ is the index of the node $N_{j_{\min(i)}}$ which has the minimum number of hops from the node N_i among all the nodes included in the jth allocation plan $P_{k,j}$. Using this cost function, the evaluation function is formulated.

$$g(s)=\sum_{k=1}^{next(s)\ 1}C(P_{k,j_k})$$

$$h(s)=\sum_{k=next(s)}^{m}\min_{j_k}\left[C(P_{k,j_k})\right]$$

3.5.2 Constraints

In the optimal file allocation problem, we also have a set of constraints, such as storage capacity, reliability, and query response time.

1. Storage capacity
   Each node N_i can store files as long as the total size of files does not exceed the capacity of the node, that is

$$\sum_{k:\text{All the files stored in the node } N_i} F_k <= B_i$$

2. Reliability (Availability)
   Let R_k be the reliability (availability) of the file M_k. Here, R_n denotes the reliability of the node and R_c denotes the reliability of the communication channel. Based on the failure rate of each node and link, we can estimate R_n and R_c. We estimate R_n to be the average reliability of all nodes that store the file M_k. On the other hand, R_c is estimated to be the product of the reliability of the channels used to access the file M_k. Let $R_{\min}$ be defined as the minimum required reliability, so that the reliability R_k has to be greater than $R_{\min}$ for all k, i.e., for all the files. According to [2], The availability of the file M_k is:

$$R_k=R_n\left[1\quad\left(1\quad R_cR_n\right)^{r_k}\right]>=R_{\min}$$

3. Query response time
   Based on network information such as the network topology, the access rate, and the update rate, we can estimate the traffic at each node as well as the offered traffic to the network. Let γ denote the total offered traffic to the network and $\alpha_{i,i'}$ denote the total traffic on the link $L_{i,i'}$. Moreover, we assume that $1/\mu'$ is the average file size, t_p is the propagation delay, and t_n denotes the nodal processing time, which is assumed to be the same for all nodes. This information enables us to estimate average network response delay. The query response time T consists of the network response delay T_n and the query processing delay t_q. Let $T_{\max}$ be defined as the maximum allowable time, and then the query response time T has to be less than $T_{\max}$. $T_{\max}$ is given based on the response time requirement for each query TR_{ik}. If the query response time is greater than $T_{\max}$, then we need to increase the replication numbers of certain files that cause traffic to bottleneck to different nodes. According to [2], query response time T = 2×

network response delay T_n+ query processing delay t_q, that is:

$$T = 2\sum_{i=1}^{n\ 1}\sum_{i'=i+1}^{n}\frac{\alpha_{i,i'}}{\gamma}\ \frac{1}{\mu' C_{i,i'}\quad \alpha_{i,i'}} + t_p + t_n \quad + t_q$$

4. Performance Evaluations

We evaluate the performance of our proposed rule-based A* algorithm in comparison with the conventional 0-1 integer programming method. Here, we consider the system parameters as follows. The number of nodes is 5 to 30 (n = 5 – 30) and the capacity of each node is 100 to 200[Gbyte] (B_i = 100000 - 200000). Also, there exist 10 to 200 types of files (m = 10 - 200), and the size of each file is 0.5 to 1[Gbyte] (F_k = 500 - 1000). The average access frequency is 1 to 50[/sec] (a_{ik} = 1 – 50) and the update probability is 0.1 (P_u = 0.1).

The performance comparison in terms of the computing time vs. the number of nodes and distinct files is shown in Figures 2 and 3. We note that in both figures, our proposed rule-based A* algorithm can reduce the computing time drastically.

The accuracy of the solution in terms of operating cost vs. the access frequency is shown in Figure 4. While our proposed method reduces the computing time, the operating cost increases. However, we can improve the operating cost of our proposed method by choosing the number of selected file allocation plans. As mentioned above, we can determine the number of selected file allocation plans by rules in considering trade-off between the accuracy of solution and the computing time.

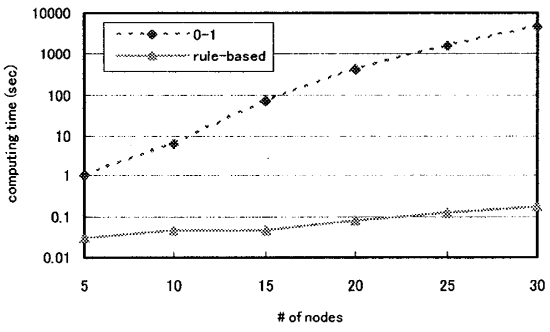

Fig.2 Computing time vs. # of nodes

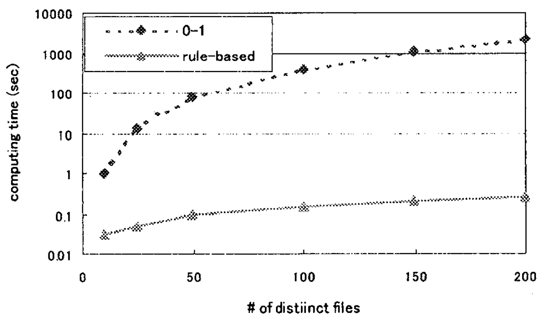

Fig.3 Computing time vs. # of distinct files

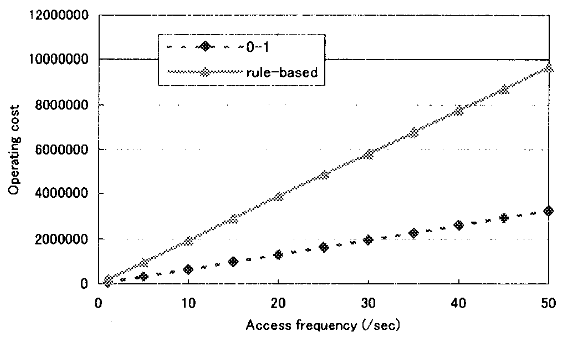

Fig.4 Accuracy of solutions

5. Conclusions

In this research, we have proposed a knowledge-based file allocation method for real-time environments. To reduce search time for solution when environment changes, we have developed file allocation rules as well as applied an A* algorithm to search for file allocation more efficiently. Our study reveals that the proposed rule-based A* algorithm drastically reduces the computing time, in comparison with the conventional 0-1 integer programming method.

References

[1] Wesley. W. Chu, Optimal File Allocation in a Multiple Computer System, *IEEE Trans. Comput.*, Vol.C-18, No. 10, October 1969, pp.885-890.

[2] Wesley W. Chu, Distributed Data Base Systems, in Vick and Ramamoorthy(Ed.), *Handbook of Software Engineering*, 14 (Van Nostrand Reinhold, 1984) pp.283-329.

[3] Y.K.Kwok, K.Karlapalem, I.Ahmad and N.M.Pun, Design and Evaluation of Data Allocation Algorithms for Distributed Multimedia Database Systems, *IEEE Journal on Selected Areas in Communications*, Vol.14, No.7, September 1996.

[4] A. Nakaniwa, M. Onishi, H. Ebara, and H. Okada, “File allocation in distributed multimedia information networks,” Proc.of IEEE Globecom ’98, pp.740-746, Nov. 1998.

[5] A. Nakaniwa, M. Onishi, H. Ebara and H. Okada, “Sensitivity Analysis in Optimal Design for Distributed File Allocation Systems,” IEICE Trans. Commun., vol.E84-B, no.6, pp.1655-1663, Jun. 2001.

[6] A. Nakaniwa, J. Takahashi, H. Ebara and H. Okada, “Reliability-Based Mirroring of Servers in Distributed Networks”, IEICE Transactions on Communications, Vol.E85-B, No.2, pp.540-549, 2002.2.