분산 시스템의 동적 파일 할당 연구

서 필 교[†]

요 약

분산 시스템에서 파일 할당 문제는 시스템의 운영비용을 최소화하기 위하여 파일 및 그의 복제물의 최적 위치를 결정하는 것이다. 정적인 파일 할당 문제는 분산 시스템의 각 노드에서 특정 파일에 발생하는 질의 및 갱신에 관련된 비용이 일정하다는 가정에서 출발하고 있다. 그러나 실제로는 시간이 지남에 따라 질의 및 갱신 등에 관련된 매개변수들은 변하게 마련이다. 이 연구에서 다루고 있는 동적인 파일 할당 문제는 변화하는 매개변수들을 고려하고 있으며, 또한 시스템 내에 파일이 한 종류만 있는 경우뿐만 아니라 여러 종류가 있는 경우도 다루고 있다. 동적인 파일 할당 문제는 혼합 정수계획법으로 모형화 되었으며 문제 해결을 위해 Lagrange 함수를 이용한 분단탐색법(branch-and-bound method) 알고리즘이 개발되었다. 제시된 알고리즘은 포트란으로 프로그램화되었으며, 여러 종류의 문제 해결을 통하여 그의 유용성을 보여주고 있다.

Dynamic File Allocation Problems In Distributed Systems

Pil-Kyo Seo[†]

ABSTRACT

In a distributed system, the simple file allocation problem determines the placement of copies of a file, so as to minimize the operating costs. The simple file allocation problem assumes the cost parameters to be fixed. In practice, these parameters change over time. In this research, dynamic file allocation problems for both single and multiple files are considered, which account for these changing parameters. A model for dynamic file allocation problem is formulated as a mixed integer program for which Lagrangian relaxation based branch-and-bound algorithm is developed. This algorithm is implemented and its efficiency is tested on medium to large test problems.

1. Introduction

Contemporary organizations are tending towards increased delegation of decision making responsibilities and functional specialization, manifest in the form of decentralized organizational structures [1]. This, coupled with growing geographic dispersion and rapid sophistication in computing and communications technology, has led to the emergence of Distributed Computing Systems(DCS). A DCS (defined as a set of computers inter-connected by telecommuni-1 cations facilities) attempts to disperse the data processing function in order to adhere more closely to the organizational structure. In addition to its ability to provide a logically integrated information system for geographically dispersed corporations, DCS realize

※본 연구는 1997학년도 명지대학교 학술연구비 지원에 의해 수행된 것임
† 정 회 원:명지대학교 경영정보학과
논문접수:1997년 2월 3일, 심사완료:1997년 6월 2일

benefits of lower communications cost, quicker access to data, higher system reliability, and ease of incremental growth [2].

The evolution of DCS has been followed almost immediately by the concept of distributed databases. A major function of a distributed database system is to provide easy, cost effective sharing of information between geographically dispersed locations, that are connected by telecommunication network. In a distributed architecture, it is not necessary to place a copy of a file at each location that needs it. Files can also be accessed from other locations via telecommunication channels or links.

Maintaining file copies in several locations increases storage and file update costs, while keeping them in fewer locations increases query cost. The problem of determining (a) the number of file copies, and (b) where to keep each copy, in order to minimize the overall cost, is known as the Simple File Allocation Problem (SFAP). In the SFAP queries read information from a single file copy which is placed at a minimum cost node. Updates, on the other hand, must write information into all copies of the file to maintain data consistency and integrity.

In the SFAP, it is assumed that all parameters of the system (query and update rates, cost of storage, etc.) are fixed and are known a priori. The SFAP is applied to the problems where all parameters of the system does not change significantly, or the change does not affect the performance of the distributed system. In practice, these parameters change over time. These changes can occur due to several reasons. New users added to a computer system or node can increase the query and update rates from that location, while a decrease in the number of users would similarly decrease the query and update rates from that node. Also, the query and update pattern can change over time, both seasonally and non-seasonally, due to changes in information needs. Thus, in each case, the file allocation determined by the SFAP would no longer be optimal over an extended time horizon. Applying the SFAP repeatedly over shorter time spans would not necessarily be optimal over the entire time span due to significant setup costs incurred in each time period. Setup cost includes file migration cost from a remote node. The SFAP treats setup cost and storage cost simultaneously i.e., each time a copy of a file is placed at a node, a setup cost and storage cost are incurred. However, when the problem is extended to several time periods, the setup cost will be incurred at a node in time period t, only if a copy of the file is to be kept at that time period, but was not present in the immediately preceding time period. Thus, in actuality, the setup cost in a given time period is a function of file placements in the previous time period.

A significant component of change in file access patterns is seasonal in nature, i.e. a pattern that keeps repeating itself. This is usually due to the nature of the query and update. This phenomena can be illustrated using an example borrowed from Gavish and Sheng [3]. This example concerns an airline database used to maintain all flight and reservation data. The semi-annual transaction frequencies of the airline database depicted in Table 1 indicate significant seasonal variations for the reservation activities in Tucson and Syracuse. To reduce remote communication in transaction processing, a sensible allocation could be to assign a copy of the database to Chicago, Denver and Syracuse during the summer season. This allocation requires file reallocations to take place when the season changes. For example, copies of the database have to be moved to Nashville and Syracuse during summer. There is a setup cost, which include a file migration cost, associated with this movement. However, this setup cost, would not be incurred if copies of the database already resided in Nashville and Syracuse from November through April. On the other hand, additional costs, involving the storage costs of the additional file copy and the update overhead to keep it consistent with other copies, are incurred. Such tradeoffs are inherent in such dynamic file allocation problems.

This seasonal change in the query and update pattern can be estimated from historical data. In this research, an extension of the SFAP model is proposed that captures this seasonal change. The dynamic model proposed provides file placements for each season by incorporating the setup costs explicitly. Therefore such a model provides the overall optimal solution for the time period that covers all the seasons.

This single file model is extended to multiple files with capacity restriction at each node. For instance, a database can be fragmented when it is distributed to computing sites. For example, transactions generated in Tucson will most often reference information on flights departing or arriving at Tucson. In order to reduce the amount of remote communications, the fragment of the airline database consisting of local (Tucson) flight information could reside in Tucson. In allocating fragmented airline database, a fragment of a database can be regarded as a file. Usually each computing site has its storage capacity. In this case single file model can be extended to multiple file model with capacity restriction at each site.

This research is organized as follows. In section 2, the literature on file allocation problems is reviewed. In section 3, models for dynamic file allocation problem for single and multiple files are developed. In section 4, solution procedures for the two models are described. Section 5 provides the computational results and the conclusion can be found in section 6.

〈Table 1〉 Seasonal query and update frequencies of the airline database

Periods / City	November-April		May-October	
	Queries	Updates	Queries	Updates
Chicago	100	50	100	50
Denver	80	20	60	15
Nashville	60	15	80	20
Syracuse	20	5	80	20
Tucson	80	20	20	5

2. Literature Review

The primary issue in the file allocation problem is to determine the number of file copies that must be maintained in the distributed system, and the location of each file copy. The SFAP was initially studied by Chu [4]. In this model, multiple file copies are allocated with the objective of minimizing operating costs. Chu's model take into account the memory size restrictions of individual processors and an average delay constraint for queries. Casey [5] investigated simpler versions of the SFAP where capacity restriction on individual processors and delay constraints for on-line queries were ignored. Levin and Morgan [6] proposed a comprehensive model that explicitly took into consideration the dependencies between program and data files. However, the proposed solution procedure is capable of solving relatively small problems only [7]. The SFAP has also been studied in conjunction with the network design problem [7-13]. In addition to system operating costs, these models include non-cost performance measures such as attainment of an acceptable level of file availability, processor location, device capacity, and average delay. The overall architectural design of a distributed system, that includes database (and computer) allocation, along with the assignment of report generation sites have been studied in [9, 14].

In contrast to the static SFAP, the dynamic file allocation problem (DFAP) involves the determination of file reallocation policies over time. Segall [15] made a unique attempt in solving the dynamic file allocation problem, permitting a file copy to move at any point in time, by applying dynamic programming techniques. The purpose of this paper was to find optimal policies for the dynamic allocation of files in a distributed system, which works under time-varying operating conditions. It is assumed that the demand patterns in all computers are known to the controller.

Levin and Morgan [16] relaxed the static/one period assumption by permitting the access rates to vary over time. They also took into account the file transfer cost involved in reallocating a file from one time period to the next. They used a dynamic programming formulation and presented a procedure to reduce the state space at each stage to make a potentially large dynamic programming problem computationally feasible. No computational results on how these solution procedures performed are reported. Levin [17] reported the study of an adaptive strategy for allocating programs and data files in a computer network, when only imperfect information on access request rates are available. This adaptive system is based on a set of computer procedures that collect access statistics, estimates frequencies, and finds optimal file assignment based on expected improvements in performance. The file is then redistributed accordingly over the network nodes. Gavish and Sheng[3] presented a Markov decision model for optimizing the file migration policy in a distributed system. The file migration operations involve only a single copy of a file.

3. Model Formulation

In this research, two closely related problems are considered. The first problem, the Dynamic File Allocation Problem for Single File (DFAPS), extends the SFAP to time periods. It is assumed that the query and update rates associated with each node are known for each time period. Further, a setup cost is incurred at each node in time period t, if a copy of the file is to be kept in that time period, but was not present in the previous time period. An optimal solution procedure is developed for this problem. The second problem, the Dynamic File Allocation Problem for Multiple Files (DFAPM), extends the first problem, the DFAPS, to many files. Storage capacities at each node place additional restrictions on where to place copies of each file in a given time period. Heuristic solution procedures are considered for this problem. In this section, integer programming formulations for the problems, the DFAPS, and the DFAPM, are presented. For both DFAPS and DFAPM, it is assumed that the distributed system is homogeneous. Each site in the system is equipped with computing, storage and telecommunication devices. Copies of a file can exist at each node in each time period. File access requests originate due to query and update transactions. A query needs to access only one copy of a file while an update needs to access all the copies of a file in order to maintain data consistency. Query and update request rates can change over time periods.

3.1 The dynamic file allocation problem for single file (DFAPS)

The following notations is used in the presentation of the DFAPS.

I: Index set of user nodes

J: Index set of file nodes

T: Index set of time period ($t = 1, 2, \cdots, s$)

S_j: Storage/maintenance cost associated with storing a copy of a file in location j. ($).

V_t: Average data volume transmitted per query in period t (megabyte).

N_{it}: Total number of queries generated at node i in period t

C_{ijt}: Unit transmission cost between i and j in period t ($/megabyte)

$V_t \cdot N_{it} \cdot C_{ijt}$: Query transmission cost satisfying the need of users in location i entirely from location j, in period t.

M_{it}: Total number of updates originating from node i in period t

W_t: Average data volume transmitted per update in period t (megabyte).

$\sum_{i \in I} C_{ijt} \cdot M_{it} \cdot W_t$: Update cost in location j in period t

F_j: Setup cost in location j.

$Y_{jt}=1$: if the file is present at node j in period t
$=0$: otherwise

X_{ijt} : proportion of need for file that is satisfied by location j, for the user in location i in period t.

$P_{jt}=1$: a setup cost is incurred at node j in period t
$=0$: otherwise

The DFAPS can be stated as 0/1 integer programming formulation as follows:

(DFAPS)

$$Z=\text{Min}\sum_{t\in T}\sum_{j\in J}S_j\cdot Y_{jt}+\sum_{t\in T}\sum_{i\in I}\sum_{j\in J}V_t\cdot N_{it}\cdot C_{ijt}\cdot X_{ijt}+\sum_{t\in T}\sum_{j\in J}\left(\sum_{i\in I}C_{ijt}\cdot N_{it}\cdot W_t\right)\cdot Y_{jt}+\sum_{t\in T}\sum_{j\in J}F_j\cdot P_{jt}$$

Subject to

$\sum_{j\in J}X_{ijt}=1 \quad \forall\ i, t$ .... (1)

$X_{ijt}\le Y_{jt} \quad \forall\ i, j, t$ .... (2)

$P_{jt}\ge Y_{jt}-Y_{jt-1} \quad \forall\ j, t\ (t\ne 1)$ .... (3)

$P_{j1}\ge Y_{j1}-Y_{js} \quad \forall\ j, t$ .... (3)

$X_{ijt}\ge 0 \quad \forall\ i, j, t$ .... (4)

$Y_{jt}, P_{jt}\in\{0, 1\} \quad \forall\ j, t$ .... (5)

In the objective function, the first term represents the storage cost. The second and the third terms represent the communication cost incurred in processing query and update transactions on remotely stored files, respectively. The fourth term represents the file set up cost. Constraint set (1) describes the demand constraint that each node i needs to access the file, in each period t. Constraint set (2) ensures the existence of a copy at node j in order to satisfy query from node i. In constraint set (3), Variable P_{jt} is associated with setup cost. Setup cost is incurred only when the file is placed the node j in time period t, but it was not kept in the (t−1)th time period. Constraint set (3) ensures this relation.

3.2 The dynamic file allocation problem for multiple files (DFAPM)

The DFAPM can be considered to be the DFAPS, extended to f number of files. However, since more than one file type may reside at the same node, the storage capacities at each node place additional restrictions on where to place copies of each file in a given time period. The following additional notations will be used in the presentation of the DFAPM.

D : Index set of file type ($d=1, 2,\cdots,f$)

S_j^d : Storage/maintenance cost associated with storing a copy of a file d in location j. ($)

V_t^d : Average data volume transmitted per query to file d in period t.(megabyte).

N_{it}^d : Total number of queries to file d generated at node i in period t

$V_t^d\cdot N_{it}^d\cdot C_{ijt}$: Query transmission cost of file d satisfying the need of users in location i entirely from location j, in period t.

N_{it}^d : Total number of updates to file d originating from node i in period t

W_t^d : Average data volume transmitted per update to file d in period t.(megabyte).

$\sum_{i\in I}C_{ijt}\cdot M_{it}^d\cdot W_t^d$: Update cost of file d in location j in period t

F_j^d : Setup cost of file d in location j.

K_j : Storage capacity of node j(megabyte).

B^d : Size of file type d (megabyte)

$Y_{jt}^d=1$: if the file d is present at node j in period t
$=0$: otherwise

X_{ijt}^d : proportion of need for file d that is satisfied by location j, for the user in location i in period t.

$P_{jt}^d=1$: a setup cost for file d is incurred at node j in period t
$=0$: otherwise

The DFAPM can be stated as 0/1 integer programming formulation as follows:

(DFAPM)

$$
\begin{aligned}
Z = \text{Min} & \sum_{d \in D} \sum_{t \in T} \sum_{j \in J} S_j^d \cdot Y_{jt}^d + \\
& \sum_{d \in D} \sum_{t \in T} \sum_{i \in I} \sum_{j \in J} V_t^d \cdot N_{it}^d \cdot C_{ijt} \cdot X_{ijt}^d + \\
& \sum_{d \in D} \sum_{t \in T} \sum_{j \in J} \left(\sum_{i \in I} C_{ijt} \cdot M_{it}^d \cdot W_t^d \right) \cdot Y_{jt}^d + \\
& \sum_{d \in D} \sum_{t \in T} \sum_{j \in J} F_j^d \cdot P_{jt}^d
\end{aligned}
$$

Subject to

$$\sum_{j \in J} X_{ijt}^d = 1 \quad \forall\ d, i, t \cdots\cdots\cdots\cdots \ (6)$$

$$X_{ijt}^d \le Y_{jt}^d \quad \forall\ d, i, j, t \cdots\cdots\cdots\cdots \ (7)$$

$$P_{jt}^d \ge Y_{jt}^d - Y_{jt-1}^d \quad \forall\ d, j, t(t \ne 1) \cdots\cdots\cdots\cdots \ (8)$$

$$P_{j1}^d \ge Y_{j1}^d - Y_{js}^d \quad \forall\ d, j, t \cdots\cdots\cdots\cdots \ (8)$$

$$X_{ijt}^d \ge 0 \quad \forall\ d, i, j, t \cdots\cdots\cdots\cdots \ (9)$$

$$Y_{jt}^d, P_{jt}^d \in \{0, 1\} \quad \forall\ d, j, t \cdots\cdots\cdots\cdots \ (10)$$

$$\sum_{d \in D} B_d \cdot Y_{jt}^d \le K_j \quad \forall\ j, t \cdots\cdots\cdots\cdots \ (11)$$

In the objective function, the first term represents the storage cost of f number of files. The second and the third terms represent the communication cost incurred in processing query and update transactions on file type d, which are remotely stored, respectively. The fourth term represents the set up cost of f number of files. Constraint sets in (DFAPS) are extended to f number of file types, such as constraint set (6) through (10). Constraint set (11) ensures that all the files placed at node in time period t should not exceed the storage capacity of node j.

4. Solution Procedures

4.1 The dynamic file allocation problem for single file (DFAPS).

A zero-one integer programming problem quite clearly has a finite number of solutions that need to be considered in identifying an optimum. But it is easy to see the computational effort involved in total enumeration, i.e., by trying all those finitely many possibilities, would soon become prohibitive. For problem DFAPS, a branch-and-bound method is developed to solve it optimally.

The branch-and-bound technique involves a well-structured systematic search of the space of all feasible solutions of constrained optimization problems that have a finite number of feasible solutions. Usually the space of all feasible solutions is repeatedly partitioned into smaller and smaller subsets (branching) and a lower bound (for a minimization problem) is calculated for the cost of solutions within each subset (bounding). After each partitioning, those subsets with a bound that exceeds the cost of a known feasible solution are excluded from all further partitioning. Thus, large subsets of solutions may be excluded from consideration without examining each solution in these subsets. A procedure for branching to a set of subproblems and bounding them is called branch-and-bound algorithm [18]. The principal feature of the algorithm developed here is the use of Lagrangian relaxation as a means of obtaining lower bounds at each node of the branch-and-bound tree.

4.1.1 Lagrangian Relaxation and Dual Problem

Lagrangian relaxation, a scheme for obtaining superoptimal bounds to integer programming problems, has been successfully applied to many combinatorial optimization problems [19]. Lagrangian relaxation of a particular problem can be obtained by dualizing a set of complicating constraints whose removal would make the problem relatively easy to solve.

Erlenkotter [20] developed a highly efficient algorithm to solve the uncapacitated plant location problem. In this problem, facilities need to be placed in m possible sites, with the objective of minimizing the total cost for satisfying demands specified at n locations. The costs include a fixed charge of opening a facility at each location and a cost associated with satisfying the demand of location j from facility i. In his approach, a tight linear programming formulation of the location problem is considered. Empirical studies

[21, 22] have found that the LP relaxation of such a formulation frequently yields natural integer solutions or provides bounds close to the optimal. He exploited this observation and considered the dual of the LP relaxation. Although the dual, a linear program, can be solved using a simplex procedure, it exhibits a special structure consisting of only one kind of a dual variable. A very efficient dual ascent procedure is used to solve this dual problem. This procedure begins with any dual-feasible solution and repeatedly cycles through the demand locations one by one, attempting to increase dual variable to the next higher value. When all the dual variables are blocked from further increases, the procedure terminates. To derive feasible primal solutions from dual solutions, complementary slackness relationships for the optimal linear programming solutions are used. If the integer primal solution obtained exhibits no complementary slackness violation, then it is optimal. If not, the gap between the primal and dual solutions is closed by using a branch-and-bound procedure.

For (DFAPS), if constraints (3) are ignored, then an uncapacitated plant location problem structure is obtained for each time period. Each such problem can be solved by Erlenkotter's efficient algorithm. Thus, the constraints (3) are treated as the complicating constraints. To dualize these constraints, we first define non-negative multipliers u_{jt} and add the non-positive term $u_{jt} \cdot (Y_{jt} - Y_{jt-1} - P_{jt})$ for $t \neq 1$ and $u_{jt} \cdot (Y_{j1} - Y_{js} - P_{j1})$ for $t = 1$ to the objective function of (DFAPS) and define

(LR$_t$(u))

$$Z_D(u) = Min \sum_{t \in T} \sum_{j \in J} S_j \cdot Y_{jt} +$$
$$\sum_{t \in T} \sum_{i \in I} \sum_{j \in J} Q_{ijt} \cdot X_{ijt} +$$
$$\sum_{t \in T} \sum_{j \in J} U_{jt} \cdot Y_{jt} + \sum_{t \in T} \sum_{j \in J} F_j \cdot P_{jt} +$$
$$\sum_{j \in J} u_{j1} \cdot (Y_{j1} - Y_{js} - P_{j1}) +$$
$$\sum_{\in T(t \neq 1)} \sum_{j \in J} u_{jt} \cdot (Y_{jt} - Y_{jt-1} - P_{jt})$$

Subject to

$$\sum_{j \in J} X_{ijt} = 1 \quad \forall\ i, t \cdots\cdots (1)$$
$$X_{ijt} \leq Y_{jt} \quad \forall\ i, j, t \cdots\cdots (2)$$
$$X_{ijt} \geq 0 \quad \forall\ i, j, t \cdots\cdots (3)$$
$$Y_{jt}, P_{jt} = \{0, 1\} \quad \forall\ j, t \cdots\cdots (4)$$

where $Q_{ijt} = V_t \cdot N_{it} \cdot C_{ijt}$ and
$$U_{jt} = \sum_{i \in I} C_{ijt} \cdot M_{it} \cdot W_t.$$

The objective function of LR$_t$(u) can be alternately expressed as follows;

$$Z_D(u) = Min$$
$$\sum_{t \in T(t \neq 1)} \sum_{j \in J} ((S_j + U_{jt}) + u_{jt} - u_{jt+1}) \cdot Y_{jt} +$$
$$\sum_{j \in J} ((S_j + U_{js}) + u_{js} - u_{j1}) \cdot Y_{js} +$$
$$\sum_{t \in T} \sum_{i \in I} \sum_{j \in J} Q_{ijt} \cdot X_{ijt} +$$
$$\sum_{t \in T} \sum_{j \in J} (F_j - u_{jt}) \cdot P_{jt}$$

For a fixed value of multiplier ujt, we can solve for the variables, X and Y, usingErlenkotter's algorithm. After we get the file location Yjt for all j and t, the optimal values of P_{jt} can be obtained as follows;

$$P_{jt} = 1 \quad \text{if} \quad F_j - u_{jt} < 0$$
$$= 0 \quad \text{otherwise}$$
$$\forall\ j, t$$

It is clear that for a fixed value u ($u \geq 0$), $Z_D(u)$ gives a lower bound on the optimal value of (DFAPS) because we have merely added a non-positive term to the objective function of (DFAPS). In order to get the tightest lower bound, corresponding Lagrangian dual problem (D) is solved.

$$(D)\ Z_D = \underset{u \geq 0}{\underset{u}{Max}} \quad Z_D(u)$$

In order to obtain a good multiplier (u) that nearly solves (D), Subgradient method is used as a dual heuristic.

4.1.2 Dual Heuristic

We propose using a Subgradient method [19] for obtaining optimal or near optimal solutions to the dual problem. This method begins at an initial point u^k and generates a sequence u^k by the rule

$$u^{k+1} = u^k + t_k \cdot d^k,$$

where t_k is a positive scalar step size and $d^k = \|d^k_{jt}\|$ $\forall$ j, t, is a direction vector. In DFAPS, the direction vector is given by

$$d^k_{jt} = Y_{jt} - Y_{jt-1} - P_{jt} \quad \forall j \in J,\ t \in T (t \neq 1) \text{ and}$$
$$d^k_{j1} = Y_{j1} - Y_{js} - P_{j1} \quad \forall j \in J$$

A formula for tk that has proven effective in practice is

$$t_k = \frac{\Gamma_k (Z^* - Z_D(u^k))}{\|dk\|^2}$$

Here Γ_k is a scalar satisfying $0 < \Gamma_k \le 2$, and Z^* is an upper bound on DFAPS, which can be calculated by the primal heuristic in section 4.1.3.

In the course of searching for a better multiplier which will give a better lower bound, a sequence of k is determined by setting $\Gamma_0 = 2$ and halving Γ_k whenever $Z_D(u^k)$ fails to increase in some fixed number of iterations.

4.1.3 Primal Heuristic

The Lagrangian relaxation developed here can also be used to provide good incumbent solutions with little additional effort. Given any fixed value for Y_{jt} and u_{jt} found by the dual solution, we can get the upper bound on DFAPS. The feasible P_{jt} values for all j and t, which satisfy the constraint (3), are given by

$$P^*_{jt} = 1 \quad \text{if} \quad Y_{jt} = 1 \quad \text{and} \quad Y_{jt-1} = 0$$
$$= 0 \quad \text{otherwise}$$

where $t \neq 1$, and

$$P^*_{jt} = 1 \quad \text{if} \quad Y_{j1} = 1 \quad \text{and} \quad Y_{js} = 0$$
$$= 0 \quad \text{otherwise}$$

The upper bound on DFAPS, which we denote Z(P) is therefore given by

$$Z(P) = Z_D - \sum_{t \in T(t \neq 1)} \sum_{j \in J} u_{jt} \cdot (Y_{jt} - Y_{jt-1} - P_{jt})$$
$$- \sum_{t \in T} \sum_{j \in J} u_{j1} \cdot (Y_{i1} - Y_{js} - P_{j1})$$

4.1.4 Branch-and-Bound Procedure

To solve a particular DFAPS, we first apply the primal and dual heuristic described in sections 4.1.2 and 4.1.3. If we find that the lower bound and an upper bound are equal, we stop as the DFAPS has been solved optimals. Otherwise, the search for an optimal solution can be completed with a branch-and-bound algorithm that use the primal and dual heuristic at each node of the branch-and-bound tree. For this purpose, we have employed a conventional 0-1 branch-and-bound algorithm based on implicit enumeration of the $2^{|Y|}$ possible value of Y_{jt}. The branch-and-bound procedure for the DFAPS observes the following rules.

(1) Branching Rule

The idea behind the branching rule used here is to reduce the difference between the lower bound and upper bound calculated at each node from which we are about to branch. The gap between a primal and dual solution occurs when the value of

$$\sum_{t \in T(t \neq 1)} \sum_{j \in J} u_{jt} \cdot (Y_{jt} - Y_{jt-1} - P_{jt}) +$$
$$\sum_{j \in J} u_{j1} \cdot (Y_{j1} - Y_{js} - P_{j1})$$

is not equal to zero. This implies that some of the complementary slackness conditions are violated. Two different branching schemes are used. One works at finding a good upper bound quickly and the other at finding a better lower bound. For the first scheme, the coefficient of Y_{jt} whose value is the largest is selected. The reason is that this variable is most likely to be 0 in the true optimal solution. The other scheme is associated with the complementary slackness violation. Each Y_{jt} appears exactly two times in the objective function of $LR_t(u)$. The Y_{jt} which appears most often among the complementary slackness violation terms is selected as a branching variable. In both the schemes, fixing these branching variable 0 or 1 can be easily done by enforcing the coefficient of the branching variable to a large positive or negative number.

(2) Lower Bounding Rule

At each node of the branch-and-bound tree, a lower bound is derived from the Lagrangian relaxation and dual heuristic.

(3) Fathoming Rule

The incumbent solution is used for the fathoming test. An incumbent solution is the value of the objective function of DFAPS for the best feasible solution identified thus far. If the lower bound obtained at a node in the branch-and-bound tree exceeds the incumbent solution, then that node if fathomed.

(4) Searching Strategy

In the branch-and-bound tree, a depth-first search is used.

Now the complete branch-and-bound procedure can be stated as follows:

STEP 1: Calculate an incumbent solution and a lower bound without fixing any variables.
STEP 2: If there is no node to be visited, then stop. Otherwise, go to step 3.
STEP 3: Branch on a variable Y_{jt}, using a branching rule.
STEP 4: Find the lower bound, using the lower bounding rule.
STEP 5: Update the incumbent solution
STEP 6: Do fathoming test using the fathoming rule.
STEP 7: If the node is not fathomed, then go to step 2. Otherwise go to step 8.
STEP 8: IF the node is fathomed, then backtrack. Go to step 2.

4.2 The dynamic file allocation problem for multiple files (DFAPM)

Since the DFAPM is an extension of DFAPS to f number of files with capacity constraints, the DFAPM can be solved based on the solution procedure for the DFAPS. Two heuristic procedures are proposed to obtain a good solution for the DFAPM.

4.2.1 Using the DFAPS procedure for all file type.

In the DFAPM, if we ignore the constraints (11), then the DFAPM can be solved by solving the DFAPS for each file type d. If the solution procedure to solve the DFAPS is applied to each file type, then the file placement for each time period and each file type will be given. Given these file placement, a procedure to satisfy the capacity constraints is presented as follows:

For each time period, perform the following procedure.

STEP 1: Find the nodes which do not satisfy the capacity constraints.
STEP 2: Drop the files from those nodes until the capacity constraints are satisfied.
STEP 3: Check whether all the file types are placed in the network, and define the missing file set.
STEP 4: If the missing file set is null, then stop. Otherwise go to step 5.
STEP 5: Sort the missing files by the file size.
STEP 6: Pick one file which is the largest among the missing file set.
STEP 7: Find the available nodes which can keep the file.

STEP 8 : Among them, pick one nodes which use that file most frequently. Place that file at that node.

STEP 9 : Subtract that file from the missing file set. If the missing file set is null, then stop. Otherwise, go to step 6.

4.2.2 Nested Lagrangian Relaxation Procedure

Another approach to obtain a good solution for the DFAPM uses a nested Lagrangian relaxation. In the DFAPM, if the constraints (11) are dualized, the Lagrangian relaxation can be solved by calling the DFAPS solution procedure as a subroutine. The Lagrangian relaxation is stated as follows:

$(LRM_t(v))$

$$M_D(v) = \text{Min} \sum_{d \in D} \sum_{t \in T} \sum_{j \in J} S_j^d \cdot Y_{jt}^d +$$

$$\sum_{d \in D} \sum_{t \in T} \sum_{i \in T} \sum_{j \in J} V_t^d \cdot N_{it}^d \cdot C_{ijt} \cdot X_{ijt}^d +$$

$$\sum_{d \in D} \sum_{t \in T} \sum_{j \in J} \left(\sum_{i \in I} C_{ijt} \cdot M_{it}^d \cdot W_t^d \right) \cdot Y_{jt}^d +$$

$$\sum_{t \in T} \sum_{t \in T} \sum_{j \in J} F_j^d \cdot P_{jt}^d +$$

$$\sum_{j \in J} \sum_{t \in T} v_{jt} \cdot \left(\sum_{d \in D} B^d \cdot Y_{jt}^d - K_j \right)$$

Subject to

$$\sum_{j \in J} x_{ijt}^d = 1 \quad \forall \quad d, i, t \cdots\cdots\cdots\cdots (6)$$

$$X_{ijt}^d \le Y_{jt}^d \quad \forall \quad d, i, j, t \cdots\cdots\cdots\cdots (7)$$

$$P_{jt}^d \ge Y_{jt}^d - Y_{jt-1}^d \quad \forall \quad d, j, t(t \ne 1) \cdots\cdots\cdots (8)$$

$$P_{j1}^d \ge Y_{j1}^d - Y_{js}^d \quad \forall \quad d, j, t \cdots\cdots\cdots\cdots (8)$$

$$X_{ijt}^d \ge 0 \quad \forall \quad d, j, t \cdots\cdots\cdots\cdots (9)$$

$$Y_{jt}^d, P_{jt}^d \in \{0, 1\} \quad \forall \quad d, j, t \cdots\cdots\cdots\cdots (10)$$

At a fixed v_{jt} value, the (LRMt(v)) can be solved by the DFAPS solution procedure. The feasible solution can be obtained by enforcing the violated capacity constraints. The procedure to satisfy the capacity constraints used in section 4.2.1 can be applied to find feasible solution to the DFAPM.

5. Computational Results

A set of computational experiments for the DFAPS was conducted. It was designed to test the performance of the Lagrangian relaxation based branch-and-bound algorithm. This algorithm was coded in Fortran and experiments were performed on the IBM 3090 computer. A number of test problems were generated, which were used in the experiment. The characteristics of these test problems are described in Table 2. In the DFAPS, the number of user nodes (i) and the possible location of the file (j) are equal. The number of nodes were varied from 5 to 15. For a given number of nodes, the number of time periods were varied from 2 to 6. In the DFAPS problem, three kinds of costs are involved. They are the query transmission cost Q_{ijt}, the update cost U_{jt}, and the setup cost F_j. In the problem data set, these three costs were randomly generated within a given range. In general, query costs are higher than update cost in each time period, and setup costs are between the range of query and update cost. The cost ranges used reflect this.

In the Table 3, average CPU times and average gaps between upper bounds and lower bounds at the parent node of the branch-and-bound tree are described. Average values were obtained by running three different cases in each data set. In the problem data set, setup cost lies between the range of query cost and update cost. For most test problems, the gap between the lower bound and upper bound were found to be within 0.1 percent. It was observed that with higher setup cost, the gap between upper and lower bound tended to be bigger. However, for those problem where a gap existed, it was found to be low. Also, it was observed that average CPU time spent at parent node was relatively short in spite of the increase of the number of variables. From the results, the Lagrangian relaxation for the DFAPS can be expected to perform efficiently in terms of the CPU time. Also, the gap between the upper bound and lower bound can be expected small. In most cases, we can expect to get the optimal solutions at the parent

<Table 2> Input Data Characteristics

Data set No.	No. of Nodes	No. of Time Periods	No. of Integer Var	No. of Real Var	Cost range
1	5	2	20	50	$200 \leq Q_{ijt} \leq 500$
2	5	4	40	100	$150 \leq U_{jt} \leq 350$
3	5	5	50	125	$100 \leq F_j \leq 200$
4	5	6	60	150	
5	10	2	40	200	
6	10	4	80	400	
7	10	5	100	500	
8	10	6	120	600	
9	15	2	60	450	
10	15	4	120	900	
11	15	5	150	1125	
12	15	6	180	1350	

node itself. In general, since the gap is small, we can expect the number of nodes visited in branch-and-bound tree to be small.

As explained earlier in section 4.1.4, two different branching schemes were used in branch-and-bound tree. Branching scheme I is based on the cost coefficients of Y_{jt} variables, and branching scheme II is based on complementary slackness violation. Table 4 shows the comparison of scheme I and scheme II. The data set 2, 6, and 7 show the difference in number of node visited in the branch-and-bound tree and total CPU time. From the results, it seems that scheme II is better than scheme I in terms of the computational time taken. Although there exists a gap between the upper bound and lower bound at the parent node, branching scheme II showed a faster way to get the optimal solution. In scheme II, the number of node visited in branch-and-bound tree was less than 5. Also, total CPU time taken in branch-and-bound tree was less than 35.3 seconds. These results clearly show that the Lagrangian relaxation based branch-and-bound algorithm with branching scheme II is a very efficient algorithm, which is capable of solving large DFAPS problems in reasonable time.

<Table 3> Average CPU time and gap at parent node

Data Set No.	Avg. CPU time(Sec) at parent Node	Avg. Gap as % of lower bound
1	0.06	0.00
2	0.11	5.30
3	0.25	0.00
4	0.52	0.00
5	0.13	0.00
6	1.94	3.23
7	6.47	2.63
8	8.22	0.00
9	0.56	0.00
10	1.05	0.00
11	1.89	0.00
12	2.11	0.00

<Table 4> Comparison of Scheme I and Scheme II.

Data	Scheme I		Scheme II	
Set No.	No. of Nodes Visited	CPU Time (Sec.)	No. of Nodes Visited	CPU Time (Sec.)
1	1	0.06	1	0.06
2	5	5.43	3	2.32
3	1	0.25	1	0.25
4	1	0.52	1	0.52
5	1	0.13	1	0.13
6	33	128.9	5	35.3
7	3	23.9	3	23.9
8	1	8.2	1	8.2
9	1	0.56	1	0.56
10	1	1.05	1	1.05
11	1	1.89	1	1.89
12	1	2.11	1	2.11

6. Conclusion

Distributed database systems have become superior substitutes for centralized systems, and they offer economic and operational advantages over centralized systems. Simple file allocation problems (SFAP) determine the placements of copies of a file, so as to minimize the storage, setup, and transmission costs. The SFAP treats query and update rates as fixed parameters. In practice, these parameters change over time. One of the significant components of the change is seasonal in nature. In this research, the seasonal change is incorporated while allocating the file. Different query and update rates in each season are considered to optimize the file placements for all the seasons.

The dynamic file allocation problem for a single file (DFAPS) incorporates the setup cost, incurred between the seasons, explicitly. The DFAPS is formulated as an integer programming problem. To solve DFAPS, a branch-and-bound algorithm is developed. A Lagrangian relaxation method is used to find lower bounds in the branch-and-bound tree. The lower bounds are obtained in a relatively short time. In most cases, the gap between the upper bound and lower bound at the parent node is less than 0.1 percent. Two branching schemes were considered in the branch-and-bound algorithm. The scheme I was based on the cost, and the scheme II was based on complementary slackness violation. Branching scheme II was found to be more effective. Thus the Lagrangian relaxation based branch-and-bound algorithm with branching scheme II can be considered as an efficient way to solve the DFAPS problems.

The DFAPS is directly extended to a multiple file allocation problem (DFAPM). Since the DFAPM has capacity constraints, heuristic solution procedures are developed for this problem. In future research, this method will be implemented. The Lagrangian relaxation based branch-and-bound algorithm of the DFAPS and DFAPM will be compared with an LP-based branch-and-bound algorithm in the computational study.

References

[1] J. L. King, "Centralized vs. Decentralized Computing: Organizational Considerations and Managerial Options," ACM Computing Surveys Vol. 15, No.4, pp.319-349, 1983.

[2] H. Katzan, 'Distributed Information Systems,' New York, Petrocelli, 1979.

[3] B. Gavish and O. Sheng, "Dynamic File Migration in Distributed Computer Systems," Communications of the ACM, Vol.33, No.2, pp. 177-189, Feb. 1990.

[4] W. Chu, "Optimal file allocation in Multiple Computer Systems," IEEE Transactions on Computers, C-18, pp.885-889, October 1969.

[5] R. Casey, "Allocation of Copies of a file in an Information Network," AFIPS Conference Proceedings, 41, pp.617-625, 1972.

[6] K. Levin and H. Morgan, "Optimal Program and Data Locations in Computer Networks," Comm. of ACM, Vol.20, No.5, pp.315-321, 1976.

[7] B. Gavish and H. Pirkul, "Allocation of Databases and Processors in a Distributed Computer Systems," Management of Distributed Data Processing, J. Akoka, ed., pp.215-231 Amsterdam, North Holland, 1982.

[8] P. Chen and J. Akoka, "Optimal Design of Distributed Information Systems," IEEE Trans. on Computers, Vol.29, No.12, pp.1068-1080, 1980.

[9] B. Gavish and H. Pirkul, "Computer and Database Location in Distributed Computer Systems," IEEE Trans. on Computers Vol.35, No.7, pp. 583-590, 1987.

[10] D. Ghosh, I. Murthy, and A. Moffet, "File Allocation Problem: Comparison of Models with Worst Case Average Communication Delays," Operations Research, Vol.40, No.6, pp.1074-1085,

1992.

[11] K. B. Irani and N. G. Khabaz, "A Methodology for the Design of Communications Networks and the Distribution of Data in Distributed Supercomputer Systems," IEEE Trans. on Computers, Vol.31, No.5, pp.419-414, 1982.

[12] L. J. Laning and M. S. Leonard, "File Allocation in a Distributed Computer Communications Network," IEEE Trans on Computers, Vol.32, No.3, pp.232-244, 1983.

[13] I. Murthy and D. Ghosh, "File Allocation Involving Worst Case Response Times and Link Capacities: Model and Solution Procedure," European Journal of Operational Research, vol. 67, pp.418-421, 1993.

[14] R. Ramesh and B. Ryan, "Optimal File Allocation and Report Assignment in Distributed Information Networks," Naval Research Logistics, Vol.37, pp.165-187, 1990.

[15] A. Segall, "Dynamic File Assignment in a Computer Network," IEEE Transactions on Automatic Control, Vol. AC-21, pp.161-173, April 1976.

[16] K. Levin and H. Morgan, "A Dynamic Optimization Model for Distributed Databases," Operations Research, Vol.26, N0.5, pp.824-835, 1978.

[17] K. Levin, "Adaptive Structuring of Distributed Databases," National Computer Conference, 1982.

[18] B. Gillett, Introduction to Operations Research-A computer oriented algorithmic approach, McGraw-Hill Book Co., 1976.

[19] M. Fisher, "The Lagrangian Relaxation Methods for Solving Integer Programming Problem," Management Science, Vol.27, No.1, pp.1-17, January 1981. pp.1-17.

[20] D. Erlenkotter, "A Dual-Based Procedure for Uncapacitated Facility Location," Operations Research, Vol.26, No.6, pp.993-1009, November-December 1978.

[21] C. ReVelle and R. Swain, "Central Facilities Location," Geographical Anal., Vol.2, pp.30-42, 1970.

[22] L. Schrage, "Implicit Representation of Variable Upper Bounds in Linear Programming," Math. Programming Study, Vol.4, pp.118-132, 1975.

서 필 교

1981년 연세대학교 경제학과 (학사)

1990년 루이지애나 주립대 경영정보학과 (석사)

1994년 루이지애나 주립대 경영정보학과 (박사)

1994년~현재 명지대학교 경영정보학과 조교수

관심분야: 데이터베이스, 정보공학, OR