Systems communications delays obstruct optimal file allocation and migration, but heuristics provide practical, approximate solutions.

File Placement on Distributed Computer Systems

Benjamin W. Wah, Purdue University

Advances in large-scale integrated logic and communication technology coupled with the applications explosion have led to distributed architectures. A *distributed computer system* is basically an interconnection of processing elements, each having certain capabilities, communicating with other elements through a network, and working on a set of related or unrelated jobs. DCS architecture is implemented in global systems such as Arpanet, local computer networks such as C_m*, and office information systems.

One important DCS characteristic is localized information processing by a subset of the processing elements. A piece of information may be processed repeatedly within a local subsystem before it changes locality. For example, a seat assignment file in an airline reservation system is very heavily used first at the airport where passengers are preparing to board. Then, as the plane flies to another airport, the use locality changes. To maximize the efficiency of the system, the file should be distributed so that it is accessible at the locality of first use and should be allowed to migrate as the need changes.

The distribution of DCS information is the distribution design problem.[1] In general, information is partitioned into fragments (files) and placed in appropriate locations. In some systems, the files are uniquely defined, and no partitioning is necessary. But often databases must be partitioned horizontally, when the instances of an object are divided into fragments, or vertically when the attributes are divided into possibly overlapping fragments. After the fragments are partitioned, the FAP, or (optimal) file allocation problem, distributes files and fragments on a DCS to satisfy system objectives, such as availability, reliability, and delay constraints, and if the constraints and needs change dynamically, the files are allowed to migrate. The optimal file migration problem adjusts the file migration sequence to assure efficient adaptation to changing needs.

Solutions to the distribution design problems depend on the operations performed on the information. In the simplest form, an operation is a file access from a specified origin. In a database distributed on a DCS, an operation or query may originate in a program located anywhere in the system. It may access multiple files assembled at a single node before it is processed, or the query and the intermediate results created in each step may be sent sequentially through the files. A combination of the two strategies is also possible. In addition, query processing is related to file placement, concurrency control, and communication network design, as shown in Figure 1, since files are partitioned and placed according to the network characteristics and query characteristics in the system. It is very difficult to solve these problems as whole units for real databases, so the designer usually decomposes them into independent problems.

The FAP was originally isolated and investigated by Chu[2] who studied it with respect to multiple files on a multiprocessor system. Current work lies in the integration of the query processing, file partitioning, concurrency control, and network design problems with the file placement problem. This article examines recent developments in these areas.

Simple file allocation

Problem formulation. The basic file placement problem is the allocation of multiple copies of a single file. It assumes that average query and update rates can be assessed for each node, that each query accesses only one copy of the file, and that all copies must be accessed by an update. Queries, updates, and data storage are represented as system costs, which must be minimized. The

0018-9162/84/0100-0023$01.00 © 1984 IEEE

problem addresses the most basic attributes of query, update and storage costs, and omits operational features, such as the delay of return traffic and system reliability. This was called the file allocation problem, but after studies with different design requirements for optimization, it was renamed the simple file allocation problem. There is a corresponding simple file migration problem.

The solution can be formulated as a linear integer program. The cost function is expressed as a combination of query, storage, and update costs over the multiple copies of the single file when the nearest file copy is accessed.[4]

The problem is to find an index set

$$I = \{Y_k = 0 \text{ or } 1, k = 1, \ldots, n\}$$

such that the cost function is minimized, where $Y_k = 1$ if a copy exists at node k (0 otherwise) and n is the number of nodes. This is an NP-hard problem[3]: it belongs to a class of problems for which there is no known optimal algorithm, and computation time increases at least exponentially with the size of the problem.[5] The computation times for all known optimal algorithms for this class increase at least exponentially with the problem size, so if n represents the size of the problem, then the computation time increases at least as fast as k^n where $k > 1$. These algorithms are, therefore, very expensive to run in realtime.

For large problems, heuristics offer "reasonable" polynomial time search strategies, which, however, do not guarantee precision. They are generally interactive algorithms that generate feasible solutions. A decision algorithm then decides whether or not to improve the solution and how to improve it. Since heuristics do not always generate optimal solutions and the analytical average and worst-case behaviors are difficult to evaluate, evaluations are generally made by simulating sample cases.

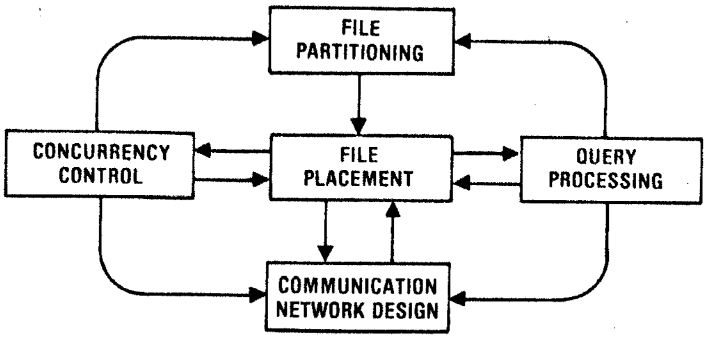

Figure 1. Problems related to file placement. Arrows indicate information flow.

Table 1.
Mapping between simple file allocation and single-commodity warehouse location.

SIMPLE FILE ALLOCATION PROBLEM	SINGLE COMMODITY WAREHOUSE LOCATION PROBLEM
Locations of computers	Possible warehouse sites
Locations of file	Locations of warehouse
Access for a file	Commodity flow
Unit cost of communicating one query unit from *j* to *k*	Unit cost of shipping commodity from plant to warehouse *j* and from warehouse *j* to customer *k*
File storage cost + multiple update cost for file at node *k*	Fixed cost of opening a warehouse at site *k*
File migration	Warehouse relocation
Cost of migraing a copy of the file from *j* to *k*	Cost of relocating a warehouse from site *j* to site *k*

Isomorphism in simple file allocation. The SFAP and SFMP have been shown to be isomorphic to two well-known problems in operations research, the Single Commodity Warehouse Location Problem and the Single Commodity Dynamic Warehouse Location Problem.[4] In the SCWLP, a set of warehouses has to be located for given sets of factories and customers so that the fixed and operational costs of the system are minimal. In the dynamic version of the same problem, plant or warehouse locations are allowed to change over multiple periods to adapt to the changing demands of customers. SFAP and SCWLP mapping parameters are shown in Table 1.

Isomorphism permits many techniques developed for the SCWLP and SCDWLP to solve SFAP and SFMP. Optimal algorithms developed for the SCWLP belong to two types: the branch-and-bound type,[6] which exhaustively enumerates possible solutions to obtain the optimal allocations; and the direct-search or implicit-enumeration type, which allows the search to be terminated with predefined conditions. Heuristic solutions, on the other hand, are usually of the add-drop type and alter feasible allocation in order to generate a better solution.[7] And dynamic programming[4] assigns dynamic warehouse locations with specified access requirements at fixed time intervals. Conversely, SCWLP and SCDWLP can be solved by techniques developed for SFAP and SFMP—the hypercube technique;[8,9] the clustering technique;[10] and the dynamic programming method,[9] which is used to solve for the optimal migration sequence.[4]

Similar to SFAP is the Stone problem of allocating processes to interconnected network of computers.[11] There is a fixed allocation cost, and the objective is to minimize assignments and communications costs. The Stone problem differs from SFAP in that it has multiple processes of one copy each and its process communication is initiated from another process, not from a fixed computer (in simple file allocation, there are multiple copies of a single file, and accesses and updates are initiated from fixed locations). The process allocation problem can be shown to be isomorphic to the single commodity quadratic assignment problem.[4] In the SCQAP, there is a set of possible plant locations and plants that require the exchange of fixed quantities of a single type commodity. The problem is to assign the plants to locations to minimize the total cost of the system. The isomorphism is in associating processes with plants, accesses with commodity flows, and in locating computers and plants.

Branch-and-bound algorithms. All feasible SFAP solutions are successively partitioned into smaller and smaller subsets by branch-and-bound algorithms.[6] Both lower and upper bounds are calculated for solutions within each subset. Newly created subsets are rejected if their lower bounds lie above the least upper bound of all subsets. Then the partitioning continues until the process finds a solution with a value equal to or less than the lower boundary of any subset.

The state of the partitioning process at any time can be represented as a partial tree. Each terminal node in the tree represents a partition, called a subproblem, which branches into even smaller partitions. Three search strategies are available to select a branching node: the breadth-first search, the depth-first search, and the best-first search.

- In a breadth-first search, tree nodes are examined at various levels. Nodes at a lower depth will always be examined before nodes at a higher depth. This search always yields a goal node nearest the root. However the sequence of nodes examined is predetermined, so the search is "blind."
- The depth-first search is similar, except that it generates complete subtrees before generating and examining others. In both algorithms, the next node to be examined is known, so the unique state of the parent node is easily found. Since the memory space required for storing the state is very small, these two algorithms save space.
- The best-first search stores all active subproblems as intermediate data. The total number of nodes expanded is minimal, since branching operations governed by this strategy are also performed under other strategies, provided the lower bounds are unique. However, it has the disadvantage of requiring large storage space.

Once a node has been selected for partitioning, a problem parameter with an undetermined value must also be selected, alternative parameters defined, and multiple subproblems created. In SFAP, the alternative parameters are a set of unassigned computers. In expanding a subproblem, the algorithm selects an unassigned computer and creates two alternatives, one with an allocated copy and another without. Expansion parameters are usually chosen to suit specific problem needs, and the computer with the highest query traffic can be selected for branching.

The lower boundary is usually furnished by the same problem with relaxed constraints: The integrality constraint on Y_k is disregarded, and the overall cost optimization is solved as a linear program. A closed form has been derived by Efroymson and Ray.[6] The upper bound is updated when a feasible solution is found.

To simplify computation of the simple file allocation, rules have been developed to decide, without any enumeration, whether a copy of the file should be placed at a node. Table 2 informally lists the four principal rules, which can be applied during the evaluation of the branch-and-bound algorithm to reduce the complexity of search algorithms.[4]

Casey's five-node example[8] illustrates the algorithm in the best-first search strategy. The lower bound of a subproblem is computed by Efroymson and Ray's equation.[6] Suppose the following matrix represents the query and update costs for a five-node system:

$$\text{query costs} = \text{update costs} = \begin{bmatrix} 0 & 6 & 12 & 9 & 6 \\ 6 & 0 & 6 & 12 & 9 \\ 12 & 6 & 0 & 6 & 12 \\ 9 & 12 & 6 & 0 & 6 \\ 6 & 9 & 12 & 6 & 0 \end{bmatrix}$$

Let query rates = [24,24,24,24,24],
update rates = [2,3,4,6,8], and
storage costs = [0,0,0,0,0].

The branch-and-bound tree is shown in Figure 2, where the state vector shows the state of the allocation. The state of a node i can be 0 ($Y_i=0$), 1 ($Y_i=1$), or U (Y_i=unassigned). The number below each vertex is the linear-programming lower bound. The corresponding conditions of Table 2 are shown on the edges. (See Ramaroorthy and Wah[4] for calculations.) The order of traversal is shown within each vertex. Since the minimum lower bound criterion is used, (UU011) is expanded. Then, there are three subproblems, (U0011), (U1011), and (UU111) with lower bounds 693, 657, and 567, respectively. Subproblem (UU111) with the minimum

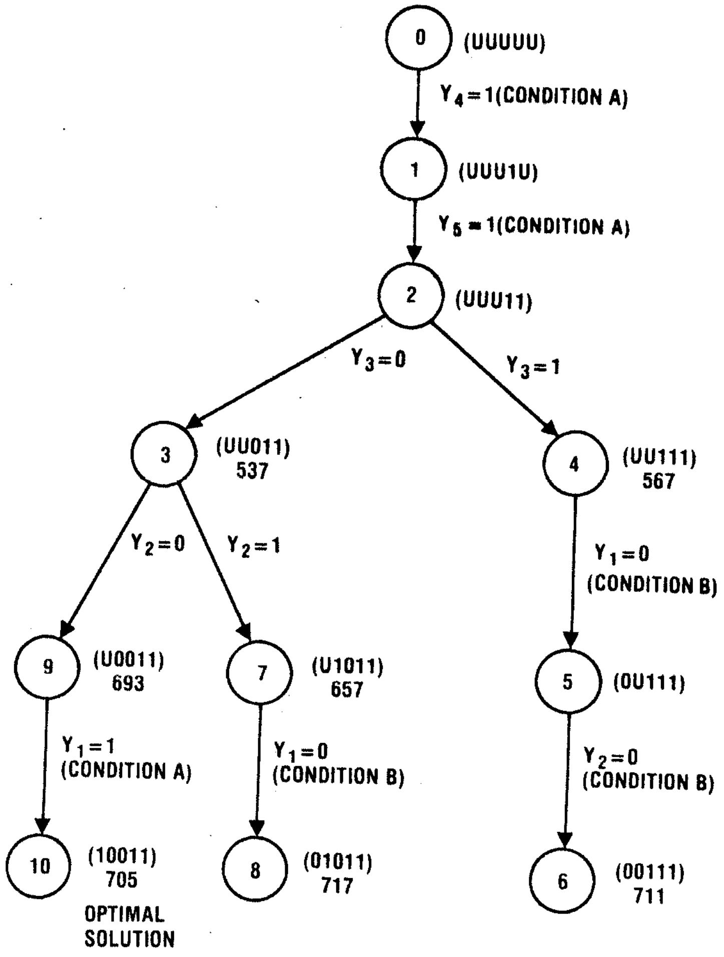

Figure 2. Application of branch-and-bound algorithm and conditions of Table 2 for Casey's five-node example. *U* indicates that a node is unassigned; the integer outside each node is the lower bound or actual cost.

lower bound is selected for expansion in the next iteration. The vertex obtained after step 9 gives the optimal solution.

Besides branch-and-bound algorithms, other enumerative approaches are also available. For example, Casey has developed a technique that shows all possible assignments in a hypercube.[8] He found that, if the total system cost decreases when a new copy is introduced, it is worthwhile to introduce one, but unnecessary to introduce further copies otherwise. Although the size of the enumerations is reduced, the complexity of the problem still increases exponentially. In other words, enumerations are impractical when the problem is large.

Fast heuristics for simple file allocation. Branch-and-bound and other enumerative algorithms are time-consuming because they backtrack on previous decisions, once it is determined that the current path may not lead to an optimal solution. To speed up processing, backtracking is limited or eliminated, but at the risk of reducing the quality of the solution.

A fast heuristic can be extended from the branch-and-bound algorithm discussed earlier. The set of unassigned nodes K_2 is selected, one node at a time, and the current assignment is extended by the selected node. There are two possibilities: either to assign or not to assign a copy of the file. Then there are $2\ |K_2|$ possible extended assignments. The linear programming lower bounds are calculated for each of these. A criterion for deciding which node to extend is to select the extended assignment with the minimum lower bound. The effect of the heuristic applied on Casey's five-node example is shown in Figure 3.

Another effective heuristic is the add-drop algorithm,[7] which begins by finding a feasible allocation. It then tries to reduce the total system cost by successive addition or deletion of file copies. When a feasible, lower cost solution is found, the algorithm adopts it as a new starting solution, and the process continues. Eventually, it reaches a local optimum in which addition or deletion does not reduce the cost. The whole procedure can be repeated with a different starting solution to produce another local optimum. The final solution is obtained by taking the minimum over all the local optima. The heuristic is illustrated in Figure 4, which shows an initial copy assignment to every node. Copies are successively dropped until costs can no longer be reduced. Then copies can be added to the assignment to test for further improvement.

Extensive evaluations of the first heuristic on sample SFAP and SCWLP cases confirm its reliability.[4] Of the 22 cases evaluated, 14 show optimal assignments. As a comparison, the application of the add-drop algorithm results in nine optimal assignments, but the error percentage for the add-drop algorithm is smaller (0.10 vs. 0.22 percent). Since both heuristics have polynomial time complexity,—$O(n^4)$ in both cases—they can be applied simultaneously to take advantage of the benefits of each.

Although the NP-hardness produces exponential time optimal algorithms, some special cases can be solved optimally in polynomial time. Coffman et al.,[9] for instance, derived a simple formula to achieve the maximum READ throughputs where there are infinite READs and updates arriving stochastically. Whitney[10] considered the allocation of a fixed number of copies without the update effects on a network modeled by a linear graph and found that the computational complexity is $O((n-q)^q)$ where q is the given number of copies. Ghosh[11] developed polynomial time algorithms for distributing a database so that multiple segments satisfying a query can be retrieved in parallel. These special cases are hard to generalize, however, and suboptimal heuristics prevail in the general file allocation problem.

General file allocation

FAP, the general file allocation problem, allocates single or multiple files with design requirements such as delay, storage capacity, parallelism, and availability, and in the presence of program accesses and specialized hardware. Its objective functions are storage and communication costs. The communication cost is further broken down for queries and updates. The optimization problem is to locate copies that (1) minimize the cost function and (2) satisfy all the accesses. More complicated cost functions can also be written. For example, the cost of a query can be separated into the cost of the request and the cost of data return.

In the absence of other constraints, the optimization problem can be decomposed into multiple optimization problems—one for each file, since all the files are independent. Techniques described in the previous section can solve individual optimization problems. When delay, availability, and capacity constraints are taken into account, their effects are usually represented in constraint equations. Then one file may affect the allocation of other files and preclude decomposition.

An interesting decomposition problem was investigated by Morgan and Levin.[9] In their model, queries and

Table 2.
Summary of conditions for placement and non-placement of a file at an unassigned node *i*.

CONDITION	RULE	USE
A	$Y_i = 1$ if the cost of a local copy at node i is smaller than the smallest possible access cost increase without a copy at node i.	Preassignment in each iteration
B	$Y_i = 0$ if the cost of a local copy at node i is greater than the maximum possible access cost increase.	Preassignment in each iteration
C	The number of possible users who can access a copy at an unassigned node should be reduced by 1 if it is more cost-effective to access an already existing copy.	Evaluating linear programming lower bound
D	$Y_i = 0$ if there exists a "better" site nearby.	Initial preassignment

updates were processed by the corresponding programs before they were sent to databases. Query traffic from programs to databases could not be determined for each node, however, because the locations of programs are not fixed. And if the problem assigns negligible costs to program storage, programs can be duplicated in every node. For a given placement of a file, the route taken to access the file from a node through a program was fixed. Because of the fixed routing, file independence, and freedom from delay and storage constraints, the optimization could be handled as multiple problems, one for each file.

Delay constraints. The delay in accessing is an important factor in determining file placement, but individual access or update delays are not analyzed. Rather, many simplifications are made, and an average delay for all transactions is calculated. We examine some of the methods here.

Chu modeled the traffic between two points of a network as an M/D/1 queueing process.[2] By evaluating the arrival and service rates of transactions and by neglecting the overhead of sending requests to files, he found a closed formula for the average delay. He calculated the average delay for sending file *f* from node *j* to node *k* and added it as a constraint to the optimization problem so that it was less than the maximum allowable retrieval time.

In their model, Mahmoud and Riordon[12] assumed that the query and query-return traffic for a request originating from the same node are equally divided among all the copies in the system. By using a fixed routing, the traffic along a link in the network could be evaluated. The delay along a link was modeled by an M/M/1 queueing process. It was assumed that all intermediate messages were independent and exponentially distributed. The average delay in the network was the average delay for all messages.

Irani and Khabbaz[13] produced the most accurate analytical evaluation of delay. They used the same M/M/1 queueing model for a link as Mahmoud and Riordon, but they accounted for the effects of routing and congestion more accurately. They showed that the flow along a link depends on the placement of copies, network connectivity, and diameter. Adaptive routing schemes were used, and flow was directed along paths with the least congestion.

Since analytical delay equations may not fully represent the effects of adaptive routing procedures and flow control techniques, Laning and Leonard developed a simulation model to evaluate the delays in each step of an iterative heuristic.[14] Their evaluation time may be long, so it is difficult to apply this technique in real time.

In updating multiple copies of a file, the updates can be issued sequentially or in parallel. When multiple, concurrent updates are issued by different users, all except one has to be blocked, then reissued. Queries are also delayed when updates are made. Detailed update effects are excluded, first, because the model was already fairly complicated and second, because the dynamic, interaction of updates would probably not be a significant factor in determining the allocation.

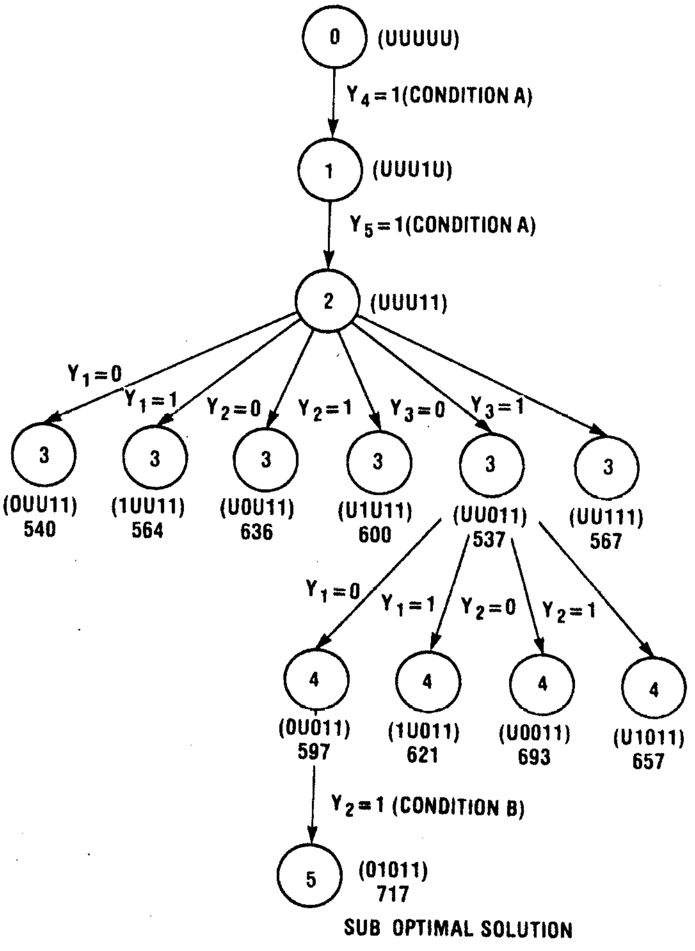

Figure 3. Application of heuristic and conditions of Table 2 on Casey's five-node example. *U* indicates that node is unassigned; the integer outside each node is the lower bound or actual cost.

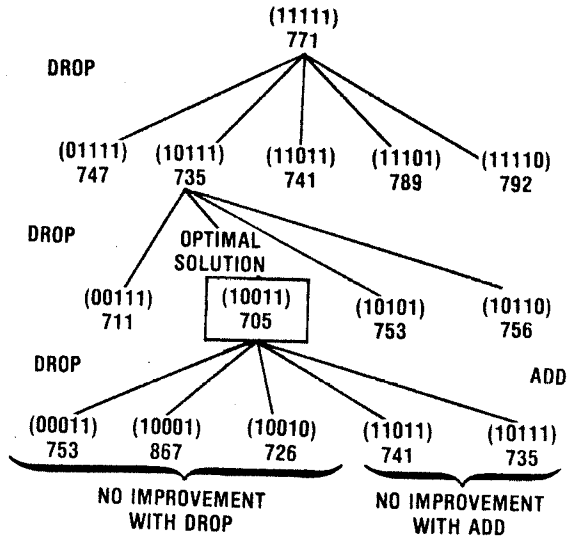

Figure 4. Application of add-drop heuristic on Casey's five-node example.

Availability constraint. Multiple copies of a file must be kept to insure that at least one copy is readily accessible if the network or a computer fails. The availability of a given file is defined as the probability that it is accessible through the storage location of the file. The reliability of a file is the probability that at least one intact copy is maintained. High reliability can be achieved by maintaining a large number of copies on the system, but high availability cannot because it depends on the network configuration.

The availability requirement is a constraint to establish the minimum acceptable availability for file f (α^f). It should be less than or equal to the average availability as evaluated from the file distribution and network configuration ($\bar{\alpha}^f$).

Mahmoud and Riordon evaluated $\bar{\alpha}^f$ as the average availability of file f accessed from node j, $\bar{\alpha}^f_j$, weighted over the total traffic demand with adjustments, for the total traffic demand for file f at node j.[15] Since static routing was assumed in the model, α^f_j was approximated in terms of the availability of a fixed set of network paths leading to copies weighted by the network connectivity. Similar availability constraints were used by Laning and Leonard.[17]

The general problem allocates files with design requirements while controlling program accesses and specialized hardware.

Irani and Khabbaz expressed availability as the probability that the network is connected and that at least one computer can make a copy available.[13] Network reliability is a function of the network diameter, degree of connectivity, and site and link reliabilities. Copies can be correctly updated only if the network is connected, and when the network is partitioned, updates within a partition cannot be communicated to other partitions.

Other constraints. Other constraints can be added for capacity, privacy/security and parallelism. The capacity constraint has been used by Chu,[2] but other researchers excluded it, since storage cost is very low and virtually unlimited storage capacity can be provided at any node. Morgan and Levin felt that the network management should provide storage transparent to network users.[2] The capacity constraint is perhaps important only for some small microcomputer-based systems, and since the size of files may grow, it is difficult to anticipate future capacity needs at design time. Consequently, the designer need distinguish only between expandable nodes and unexpandable nodes.

For file security, it may be necessary to preclude the storage of copies at some nodes. The constraint $Y^f_k = 3$ prevents the storage of file f at node k. And to take advantage of special processing capabilities of nodes and to allow maximum parallelism, a constraint may be needed for copies in a subset of nodes or for copies of two files on different computers.[10]

Heuristic solution techniques. Constraints increase the complexity of the problem. The conditions of Table 2 are no longer applicable because they reflect no restrictions. In addition, delay and availability constraints prevent the decomposition of optimization for multiple files, so the most reliable method for solving placement problems is exhaustive enumeration, such as branch-and-bound algorithms.[18]

Heuristics including the add-drop heuristic,[15] clustering algorithm,[13] steepest ascent and subgradient method, and greedy algorithm[13] have been used effectively in sample cases. Simulations have also been used iteratively to search for a feasible solution.[17] A linear-programming, lower bound without the integrality constraints can be developed, but it cannot be put into a closed form as in SFAP,[6] and the linear program must be evaluated each time. Other integer-programming algorithms can also be applied,[2,12] but the size of solvable problems is very restricted.

General file allocation with network design

Optimal file allocation parallels network design. Besides placing programs and files at appropriate locations, the capacities of communication lines are assigned,[15,16] computer processing power is chosen,[18] and a network with maximal connection and minimal diameter is designed.[16] The general form of the optimization problem is

minimized cost = storage cost + link cost

subject to network reliability constraints, database availability constraints, and access delay constraints, and others.

Among objective functions, Mahmoud and Riordon considered costs of file storage and leasing communication links.[15] To these, Irani added the termination cost of a link.[16] Both formulations sought optimal link capacities, and query and update effects were included in the delay constraint.

A complicated objective function was developed by Loomis and Popek,[13] who included the specialized needs of a query, the extent of specialization of a node, local memory accessibility, transmission delay time, and the possibility of accessing two objects in parallel. An equally elaborate objective function was also used by Chen and Akoka,[18] who considered the costs for database software, computer equipment, communication line installation, database storage, and communication of query/update from users to programs and from programs to databases. Their goal here was to simultaneously optimize the distribution of processing power, the allocation of programs and databases, and the assignment of communication link capacities on a system in which queries and updates were sent along static routes.

In a heuristic approach, a feasible file allocation is first generated, and the communication network is designed with the given file allocation and the delay and availability constraints. The files are reallocated, and design steps are iterated until no further improvement in cost is possible.[15,16] The cost function includes storage and network. Queries and updates affect only the delay in the network. This approach is an improvement over problems discussed earlier, which model the effects of queries and updates as system costs. It is not appropriate, however, when file migration is allowed because link capacities cannot be changed dynamically. An optimization suitable for investigating file migration should have the following form:

minimized cost = storage cost

subject to access delay contraints, database availability constraints and others.

Here, link capacities are assumed to be statically chosen.

General file allocation with query processing

In general, a query may access multiple files. A query processing strategy schedules the order that the files are accessed. To solve the query processing problem, file locations must be known, while solutions to the file allocation problem require a query processing strategy. All the studies described so far assume that a query or update is directed toward a single file and that queries and updates originate from a known location. Morgan and Levin have assumed a more general query or update originating from a given node, which must be processed by a program before it is sent to a file.[12]

Solving combined query and allocation problems is extremely complex, especially when delay and availability constraints are also included. Apers proposed a solution based on the assumption that the overhead of communication depends on the amount of data sent and that the cost of communication per unit data is constant in the network.[19] The two problems can then be solved independently. First, the query processing order is optimized independently for each query or update providing that distinct copies of files are used. Distinct copies are located at virtual sites and the file allocation becomes the mapping of the virtual sites to physical sites so that the total communication costs are minimal. It is assumed that the cost of communication within a physical site is zero. The optimization can be formulated as a nonlinear integer program. Branch-and-bound algorithms and heuristics have been proposed to solve the combined query and allocation problem, and network configuration would not enter into the optimization.

Optimization is further complicated when the communication cost is not uniform in the network because the query processing order and file allocation must be optimized together. Future research will have to establish conditions that permit the two problems to be studied independently.

File migration

The problem of file migration arises from the dynamic nature of accesses. Although there may be a locality of access for a file, there are occasionally very few activities inside the locality, and the file is accessed outside its locality. Moreover, the locality of access may be time-varying, as in global networks distributed over different time zones. Then it is more efficient to allow the file to migrate to major access points.

A typical migration method involves the application of a static algorithm. Access rates are set initially, and migrations occur at fixed intervals. At the end of each interval, either migrations take place or they do not. Therefore, over a period of t intervals, there are a minimum of 2^t possible alternatives.

To simplify evaluation of the file allocation algorithm in each iteration, additional conditions are set to eliminate unnecessary enumerations.[12] If I_t is the index set of allocation in period t and $F(I_t)$ is the query and storage costs with allocations I_t, then

$$\bar{F}(I_t) = F(I_t) + M_{t+1}(I_t, I_{t+1}),$$

where $M_{t+1}(I_t, I_{t+1})$ is the migration cost from I_t to I_{t+1} at the beginning of period $t+1$. Only alternatives with $F(I_t) \le F(I_t - \{i\})$ and $\bar{F}(I_{t-1}) \le \bar{F}(I_{t-1} - \{i\})$ have to be considered in the file allocation algorithm.

Dynamic programming, both static and stochastic, has also been used in solving optimal migration sequences. In static dynamic programming, it is assumed that time is divided into periods during which the access rates remain constant. In the stochastic version, the access rate is modeled as a continuous random variable with a fixed number of different values that vary according to a Markov model.[20] These assumptions permit solutions, but they do not correspond to conditions in real systems.

Optimal allocation places programs and files at appropriate locations and assigns communications capacities.

Porcar has studied user behavior of file systems collected on traces and proposed synthetic generation of user accesses.[21] He observed that users access files at independently variable rates but in a rather precise order. Basically, they tend to access files that have been recently accessed in the working set. Heuristics for file migration were proposed and were verified by simulations.

Current models are based on the user-file relationship and on the assumption that access rates can be predicted or collected accurately in real time. Future studies should concentrate on the characterization of database behavior and modeling of systems with long access delays, in which the collection of accurate access rates is difficult. A model has been investigated to determine the complexity of migration detection. The NP-hardness of the problem makes enumerative algorithms impractical for real time applications. Furthermore, the access characteristics are dynamic and unpredictable and, therefore, cannot be used by the same dynamic programming that precomputes the migration sequence. A polynomial time heuristic proves to be very efficient for coping with the transient nature of accesses.

Assuming that the access rates are set beforehand, determining when to migrate multiple copies of a single file in the time interval $[0,t]$ to minimize the cost is NP-hard. This fact can be proved by showing that the corresponding decision problem—whether or not there is a migration sequence such that the total operating cost equals B— is reducible from the knapsack decision problem.* A special form of the problem in which migrations can be initiated at discrete times is used to determine the NP-hardness of the general problem.

This knapsack decision problem is reducible to the above decision problem. Given an instance of the knapsack problem with inputs $a_1, a_2, \ldots, a_n, A$, we can construct in polonimial time an instance of the migration problem by selecting the parameters as follows:

Let

$$x_i = \begin{matrix} 0 \text{ (no migration is initiated at } t_i) \\ 1 \text{ (migration is initiated at } t_i) \end{matrix}$$

$a_i =$ the number of file movements at t_i (assuming that each file movement incurs a unit cost).

$A =$ B (the total migration cost).

There are no other costs associated with the operation of the system.

The knapsack decision problem is, therefore, reducible to selecting the migration points. Since the knapsack problem is NP-complete, the file migration problem is also NP-hard, as are more general forms of the file migration problems. It is clear that, even when the access rates are known, exhaustive enumeration, such as in Levin's approach,[12] is necessary to determine the optimal migration sequence.

In general, specific access rates are rarely constant, although the average access rate over a relatively long interval is usually predictable. When there is a transient increase in the access rate at a node without a copy, it may be desirable to allow a copy of the file to migrate there, then eliminate it when the accesses cease. Experience shows

- Access rates are not predictable in a short time interval, and there may be transient variations. It is difficult to extrapolate trends from past behavior.
- Communication overhead and the transient nature of accesses make it difficult to collect changes in access rates to a central computer in real time and to determine the optimal migration sequence. It is best for each node to determine in a distributed fashion whether or not to permit migration.
- The file allocation algorithm (optimal or heuristic) is time-consuming when executed locally each time the access rate changes. A simpler criterion should be used by each node to determine whether migration is needed.

*The knapsack decision problem has been shown to be NP-complete.[5] In this problem, a set of integers $a_1, a_2, \ldots, a_n$, and A are given. It is necessary to find a set $\{x_i : x_i \in \{0,1\}, i = 1, \ldots, n\}$ such that $\sum a_i x_i = A$.

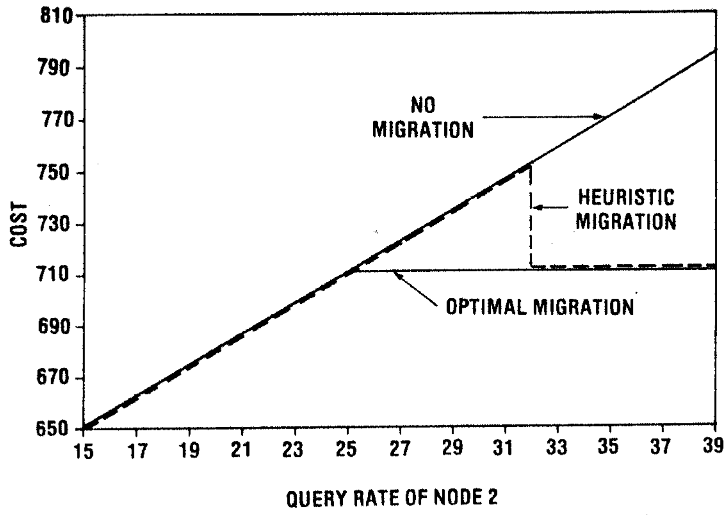

Figure 5. An illustration of heuristic migration detection using Casey's five-node example (only query rate of node 2 is changing)

These observations suggest a simple heuristic that permits limited migration in the locality of node j and determines when migration should be carried out. Let $d_{j,k}$ be the cost per unit size of query from j to k, I^f be the index set of nodes with a copy of file f, and λ^f_j be the query load originating at node j for file f per unit time. Suppose k is a node in the locality of j containing a copy of file f, such that

$$d_{j,k} = \min_{l \in I^f} d_{j,l} \text{ and}$$

$$J^f_k = \{x \mid d_{x,k} = \min_{l \in I^f} d_{x,l}\} \text{ with ties broken accordingly.}$$

J^f_k is the set of nodes in the locality of j which also accesses the copy of file f at node k. Note that

$$j, k \in J^f_k.$$

Suppose each node initiates migrations for the neighboring nodes based on the local change in access rates. When λ^f_j is changed to $\lambda^f_j{}'$ and the access rates of other nodes remain constant, the copy at node k is allowed to migrate to any of the remaining $|J^f_k| - 1$ nodes; or the copy at node k is eliminated, and all the nodes in J^f_k will access other copies in the system. Whenever migration is necessary, the static file allocation algorithm is called to find the optimal allocations. This detection procedure requires $|J^f_k| + 1$ evaluations of the cost function and is relatively simple even with delay and availability constraints.

As an illustration, the detection procedure can be applied on Casey's five-node example, where there is one file and where index f is dropped. As evaluated before, the optimal cost is 705 with allocations at nodes 1, 4 and 5. The copy at node 1 is shared between nodes 1 and 2, that is

$$J_1 = \{1,2\}$$

Figure 5 shows a plot of the cost function when λ_2 is increasing, and the cost for migrating the copy from node 1 to node 2 is negligible. The algorithm detects that migration should be initiated at $\lambda_2 = 32.05$. At this point, the static file allocation algorithm is called to find that the optimal allocations are at nodes 3, 4, and 5, but for the system to operate optimally, migrations should be initiated when λ_2 reaches 25.05.

The advantage of the above scheme is that migration is not initiated until after it is first needed to avoid transient increase in the access rate. Moreover, the detection procedure is relatively simple to execute in real time. But if the access rate does not reach a threshold, migration will not be initiated and the system will be operating suboptimally. A more accurate estimation algorithm may have to be used.

File migration requires that access rates be assessed in real time at each node, which may be difficult for queries that have not been executed. The assessment can be made by estimating the access rate from the average access rate in a previous time window. The size of the window is application-dependent, but should not be so large that transients are filtered out. The estimated access rate should be transmitted regularly to other nodes.

Special file migration detection

Although the previous section shows that the detection problem for file migration is NP-hard, some special cases of the problem can be solved in polynomial time. The Engineering Computer Network at Purdue University, provides the capability of virtual terminal access. It allows a user to connect directly to a remote host while the terminal acts as though it were connected physically to the remote host. Basically, the local host relays information between the user and the remote host without interpreting the information. Many users can share the link and access remote files through the Unix high-level operating system.

In a virtual terminal access approach, the processes that access remote files are executed on the remote host. A user editing a remote file actually communicates with an alternative is to edit on the local host and communicate over the network with the remote file. This approach is rarely effective, however, because a large part ciate over the network with the remote file. This approach is rarely effective, however, because a large part of the file may have to be searched in executing an editor command. The last alternative migrates the file to the local host, edits locally, and writes the file back at the end of the session. These three techniques are illustrated in Figure 6.

Using the first or the third approach, we can determine precisely when migration should be initiated. Let x be the amount of information transferred between a user and the editor on a remote file. It includes requests sent and the results returned. $G(x)$ is the distribution function of x and

$$\bar{x} = \int_0^\infty x\,[dG(x)]$$

is the mean of x.

Let S be the size of the remote file. We want to find a threshold T such that, whenever x exceeds T, the file should be migrated to the local host for editing.

The information transfer is

$$\int_0^T x\,[dG(x)] + \int_T^\infty 2S\,[dG(x)]$$

It assumes that the file size is unchanged at the end of the editing session. To find the value of T such that the amount of information transfer is minimal, we can rewrite the amount as

$$\bar{x} + \int_T^\infty (2S - x)\ [dG(x)]$$

We note that the integral is minimum when we start the integration at $T = 2S$. In this case, the integrand is always negative and the minimum of the integral is obtained.

For this example, the detection of migration is simple and unique. At the beginning of the editing session, if $x > 2S$, the file should be migrated immediately to the local host; otherwise, remote editing should be done. Of course, the value of x is unknown until the editing session is completed. In this case, an estimated x, such as $\bar{x}$, will be used.

Besides the editor process, this migration detection is applicable to other processes in the system. For instance, if the overhead of remote execution is greater than the overhead of migration and rewriting, migration is beneficial. Still the workload of the local and remote computers cannot be overlooked, and if the local computer is seriously overloaded, it may be advisable to execute the processes remotely.

File migration is more difficult to solve than the file allocation problem mainly because of the dynamic

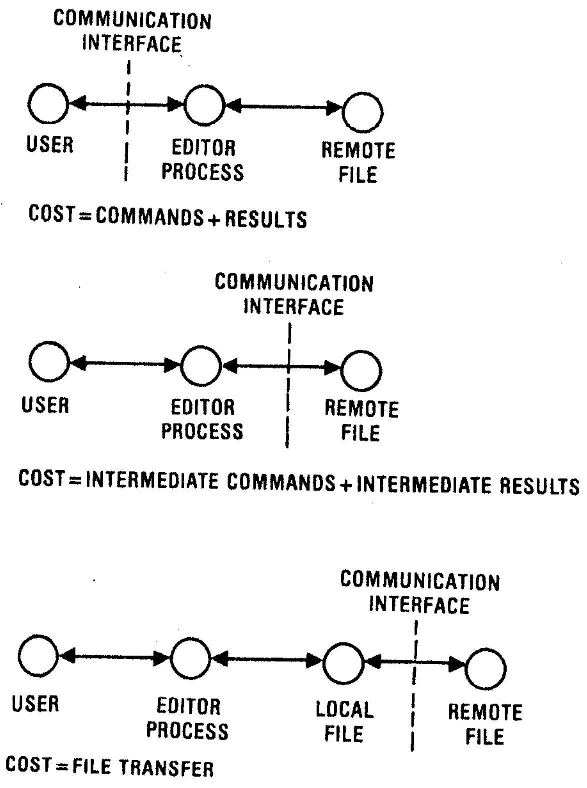

Figure 6. Three techniques of editing a remote file.

nature of accesses and the infeasibility of collecting accurate information in real time. Heuristics seem to be appropriate for most applications, but adjusting parameters for these heuristics is application-dependent and must be verified by experiments. The important feature of a real-time file migration algorithm is to detect when migrations should be carried out. The detection problem is NP-hard, and a heuristic should determine the time for migration. Long-term prediction of the access rates may be feasible, but will not contribute to the optimal system performance. Current research is in a combination of file placement, query processing, file partitioning, and network design problems. Most studies make very simple assumptions about the other problems in order to make the file placement problem solvable. The future trend lies in the relaxation of these assumptions while keeping the file placement problem mathematically tractable. ■

Acknowledgments

This research was partially supported by National Science Foundation grant ECS81-05968 and by CIDMAC, a research unit sponsored by Purdue University, Cincinnati Milicron Corporation, Control Data Corporation, Cummins Engine Company, Ransburg Corporation, and TRW.

References

1. S. Ceri, and S. B. Navathe, " A Methodology for the Distribution Design of Databases," *Proc. Compcon 83,* Feb. 1983, pp. 426-431.
2. W. W. Chu, "Multiple File Allocation in a Multiple Computer System," *IEEE Trans. Computers*, Vol. C-18, No. 10, Oct. 1969, pp. 885-889.
3. K. P. Eswaran, "Placement of Records in a File and File Allocation in a Computer Network," *Information Processing 74,* IFIPS, 1974, pp. 304-307.
4. C. V. Ramamoorthy and B. W. Wah, "The Isomorphism of Simple File Allocation," *IEEE Trans. Computers*, Vol. C-32, No. 3, Mar. 1983, pp. 221-232.
5. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman and Company, 1979.
6. M. A. Efroymson and T. C. Ray, "A Branch and Bound Algorithm for Plant Location," *Operations Research*, May-June 1966, pp. 361-368.
7. A. A. Keuhn and M. J. Hamburger, "A Heuristic Program for Locating Warehouses," *Management Science*, Vol. 9, No. 4, July 1963, pp. 643-666.
8. R. G. Casey, "Allocation of Copies of a File in an Information Network," *SJCC*, 1972, pp. 617-625.
9. E. G. Coffman, Jr., et al, "Optimization of the Number of Copies in a Distributed Database," *IEEE Trans. Software Engineering*, Vol. SE-7, No. 1, Jan. 1981, pp. 78-84.
10. V. Whitney, "A Study of Optimal File-Site Assignment and Communication Network Configuration in Remote-Access Computer-Message-Processing and Communication Systems," PhD thesis, Univ. of Michigan, 1970.
11. S. P. Ghosh, "Distributing a Database with Logical Associations on a Computer Network for Parallel Searching," *IEEE Trans. Software Engineering*, Vol. SE-2, No. 2, June 1976, pp. 106-113.
12. H. L. Morgan and K. D. Levin, "Optimal Program and Data Locations in Computer Networks," *Comm. ACM*, Vol. 20, No. 5, May 1977, pp. 315-322.
13. M. E. S. Loomis and G. J. Popek, "A Model for Data Base Distribution," *Computer Networks: Trends and Applications*, 1976, IEEE Press, New York, pp. 162-169.
14. H. S. Stone, "Multi-processor Scheduling with the Aid of Network Flows," *IEEE Trans. Software Engineering*, Vol. SE-3, No. 1, Jan. 1977, pp. 85-93.
15. S. Mahmoud and J. S. Riordon, "Optimal Allocation of Resources in Distributed Information Networks," *ACM Trans. DataBase Systems*, Vol. 1, No. 1, Mar. 1976, pp. 66-78.
16. K. B. Irani, and N. G. Khabbaz, "A Methodology for the Design of Communication Networks and the Distribution of Data in Distributed Supercomputer Systems," *IEEE Trans. Computers*, Vol. C-31, No. 5, May 1982, pp. 419-434.
17. L. J. Laning and M. S. Leonard, "File Allocation in a Distributed Computer Communication Network," *IEEE Trans. Computers*, Vol. C-32, No. 3, Mar. 1983, pp. 232-244.
18. P. P. S. Chen and J. Akoka, "Optimal Design of Distributed Information Systems," *IEEE Trans. Computers*, Vol. C-29, No. 12, Dec. 1980, pp. 1068-1080.
19. P. M. G., Apers, "Centralized or Decentralized Data Allocation," *Distributed Data Sharing Systems,* R. P. van de Riet and W. Litwin, eds., North-Holland, New York, 1982.
20. A. Segall, "Dynamic File Assignment in the Computer Network," part 1, *IEEE Trans. Automatic Control*, Vol. AC-24, no. 5, Oct. 1979.
21. J. M. Porcar, "File Migration in Distributed Computer Systems," PhD thesis, Univ. of California, Berkeley, 1982.

Benjamin W. Wah is an assistant professor of electrical engineering at Purdue University. He received his BS and MS degrees in electrical engineering and computer science from Columbia University in 1974 and 1975, and MS degree in computer science and the PhD degree in engineering from the University of California at Berkeley in 1976 and 1979, respectively. His current research interests include parallel computer architecture, distributed computer systems, and theory of computing.

Wah's address is, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.