Stochastic Query Optimization in Distributed Databases

P. E. DRENICK and E. J. SMITH
Polytechnic University

Many algorithms have been devised for minimizing the costs associated with obtaining the answer to a single, isolated query in a distributed database system. However, if more than one query may be processed by the system at the same time and if the arrival times of the queries are unknown, the determination of optimal query-processing strategies becomes a stochastic optimization problem. In order to cope with such problems, a theoretical state-transition model is presented that treats the system as one operating under a stochastic load. Query-processing strategies may then be distributed over the processors of a network as probability distributions, in a manner which accommodates many queries over time.

It is then shown that the model leads to the determination of optimal query-processing strategies as the solution of mathematical programming problems, and analytical results for several examples are presented. Furthermore, a divide-and-conquer approach is introduced for decomposing stochastic query optimization problems into distinct subproblems for processing queries sequentially and in parallel.

Categories and Subject Descriptors: C.2.4[**Computer-Communication Networks**]. Distributed Systems—*distributed databases*; H.2.4 [**Database Management**]: Systems—*distributed systems*; *query processing*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms. Algorithms, Performance, Theory

Additional Key Words and Phrases: Distributed query processing, state-transition model, stochastic query optimization

1. INTRODUCTION

The problem of optimizing the processing of a single, isolated query in a distributed database system has received a great deal of attention in recent years [4, 8, 27, 41]. Most of the literature devoted to this topic addresses the problem of finding a deterministic strategy for assigning the component joins of a relational query to the processors of a network that can most efficiently execute the joins and that can most economically perform any required interprocessor data transfers. Thus, for each new type of query that arrives

Authors' addresses: P. E. Drenick, EE/CS Department, Polytechnic University, Route 110, Farmingdale, NY 11735; E. J. Smith, EE/CS Department, Polytechnic University, 333 Jay Street, Brooklyn, NY 11201.

at the system, a new optimal strategy is determined that minimizes the overall cost to the network for processing that query.

However, in practice, it is the capacity of distributed systems for concurrent processing that often motivates the distribution of a database in a network in the first place. That is, a different approach to query optimization is obtained if the system is viewed more generally as one which receives different types of queries at different times and if the system is capable of load-sharing the processing of more than one query at the same time. In contrast to the static single-query optimization problem, the multiple-query problem is not deterministic. Since it is usually not known when a particular type of query will arrive at the system, the multiple-query input stream constitutes a stochastic process. Instead of searching for a deterministic or "pure" strategy for processing a single query, the system's multiple-query strategy is distributed over the sites of the network as a probability distribution or "mixed" strategy. Thus, the "decision variables" of stochastic query optimization problems are just the probabilities that a component join is executed at a particular site of the network.

A general approach is proposed in this paper for stochastic query optimization in relational [10, 13], distributed database systems, based on a Markov model of the system. The model is an extension of one described in [15], which is based on the original multiprocessing model of [14] and [16]. The model may also be regarded as a probabilistic extension of the state-transition model due to Lafortune and Wong [27] from a single-query input to a multiple-query stochastic load. That is, following [27], a change of state in the network is associated with the execution of each component join of a query. The model is sufficiently general to encompass both long-haul and local-area networks, semijoin strategies, and full or partial replication of relations over sites of the network. The approach taken in this paper is multidisciplinary, employing techniques and principles of such varied fields as automata theory [3, 19, 25], scheduling theory [11], mathematical programming [22, 31, 37], and performance analysis [17, 23]. Stochastic models are common in database literature [e.g., 1, 5, 36, 37], but this paper differs from these references in that it is concerned primarily with query-processing capacity, and not with the performance of concurrency mechanisms.

The main objective of the model is to provide a means for determining query-processing strategies that are globally optimally, in the sense that the system achieves maximum throughput for the network as a whole. Using the system throughput as a performance measure, optimal query-processing strategies are sought as the solution of the mathematical programming problem defined by maximizing the throughput as objective function, subject to a set of suitable constraints on the system's query-processing strategies. Since it has been shown that optimal strategies for systems under stochastic load, subject to overload constraints (as defined in Section 2), are mixed [14], probabilistic query plans are of primary interest in this introductory paper.

This paper is organized according to the type of input queries to be processed by the model, in order of their increasing complexity. The simplest case of a single-join query is considered first in Section 2, which also

describes many of the basic assumptions and concepts for the sequel. In Section 3, it is shown that single-join models may always be solved as linear programming problems, even when more than one single-join query may be processed at different sites. However, multiple-join queries lead in general to nonlinear programming problems, and both combinatorial and iterative solutions of such problems are discussed in Section 4. The complexity of the general model is also shown in Section 4 to be polynomial in the number of sites in the network and exponential in the number of joins in a set of input queries.

2. THE SINGLE-JOIN, SINGLE-QUERY MODEL

2.1 Basic Assumptions and Concepts

This section is concerned with the stochastic optimization of the most common type of input query: the single-join query type. Although the most significant feature of the model is its ability to accommodate more than one type of query, the single-query case is convenient for introducing some basic assumptions and key concepts that apply to multiple-join, multiple-query models, as well.

The model to be presented is similar in principle to the model of Lafortune and Wong [27], and it may also be regarded as an extension of the data flow model in [15]. As a state-transition model, there is a very natural choice of state space for the model, i.e., the distributed database itself. More formally, the state of the system at time t is defined to be the set of relations stored in local memory at each processor site in the network at time t, so that the granularity of distribution is the relation.

For example, let Q_1 denote the single-query type consisting of the single-join

$$Q_1 = A \bowtie B = A'$$

where at time $t = 0$ relation A is stored in local memory at site 1 and relation B at site 2, and suppose, for the sake of simplicity, that there are only two autonomous processors in the network under consideration (as will be seen, the model is easily extended to networks having more than just two sites). Then, the "initial state" or "materialization" [40] of relations referenced by the query Q_1 in the two-site network is just the two-component column vector

$$x_0 = \begin{pmatrix} A \\ B \end{pmatrix}$$

where the i-th component of the vector x_0 is the set of relations stored at site i ($i = 1, 2$) at time $t = 0$. There may be many such initial states, as discussed further in Section 2.2.

It will be assumed that the input to the system consists of a single stream of type Q_1 queries and that the initial state x_0 is known with given time-invariant probability $p_0 = p(x_0)$. That is, p_0 is the probability that relation A is available at site 1 and relation B at site 2, and that neither relation is locked due to updating or is unavailable for query processing for

any other reason. Thus, the initial state x_0 represents a possible distributed database design in the static sense of [9], [12], and [20] with probability p_0. Moreover, following Lafortune and Wong [27], it will be assumed that query processing cannot be interrupted, that the network is fully connected, and that the identity of the sites of query arrival and answer are irrelevant for global optimization purposes (see [15] for ways in which these restrictions may be relaxed).

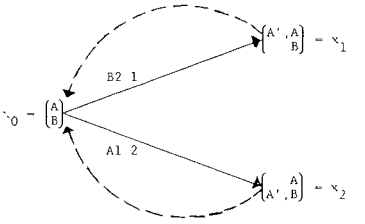
The state-transition rules for the model are also based on those of [27]. Given that the system is in state x_0 when a type Q_1 query arrives, the state x_0 undergoes transition to one of two other final states

$$x_1 = \begin{pmatrix} A', & A \\ & B \end{pmatrix}, \quad \text{or} \quad x_2 = \begin{pmatrix} & A \\ A', & B \end{pmatrix}$$

where the join $A' = A \bowtie B$ is computed at site i in state i, $(i = 1, 2)$. The two transitions from x_0 to x_1 and to x_2 are depicted in the (extended) state-transition graph of Figure 1a, where the transition arcs have been labeled by the data transfers needed to effect the joins shown in the destination states. That is, the label $B2:1$ that appears along the arc from x_0 to x_1 indicates that relation B should be transferred from site 2 to site 1 so that the join $A' = A \bowtie B$ may be computed at site 1. Similarly, $A1:2$ represents the transmission of relation A from site 1 to site 2 so that A' may be executed at site 2. The colon notation (relation source: destination) for message passing is borrowed from the Smalltalk-80 language [18].

It will be assumed in Section 4 that one join of a multiple-join query must be executed before the next join begins, so that final states x_1, x_2 may be considered "join execution states." Thus, the feedback loops shown as dashed arcs in Figure 1a represent the transition from an execution state back to the initial state. As will be seen, the feedback loops are unnecessary for query optimization purposes and are omitted in the sequel. It should be mentioned that Figure 1a assumes that the join A' is not cached and that subsequent queries of Q_1 require recomputation of A'. However, if the system caches intermediate and/or final joins, it will be assumed that such joins must be recomputed nevertheless if component relations of the joins are updated. Thus, for systems with caching, the symbol Q_1 also represents a request that joins be recomputed to reflect updates to their component relations. This topic is discussed further in Section 3.1.

Since the state-transition graph of Figure 1 contains a node with two possible transition arcs on the same input symbol Q_1 (i.e., the state x_0), the corresponding finite-state machine M defined by the graph is nondeterministic [2, p. 319]. For networks having more than two sites, possibly with replication of data over the sites, and for more complex queries requiring more than the single-join of Q_1, there may be many candidate sites for performing a component join, hence, many transitions arcs per node of the state-transition graph. As in [27] and [40], it will be assumed typically that only sites at which one of the two operand relations of a component join resides may be a candidate site for the join. This simplification avoids such



(a) State-transition graph

Machine M Present State (PS)	Next state (NS) Q_1
x_0 = (A,B)	x_1 = ((A',A), (B))
	x_2 = ((A) (A',B))

(b) State-transition table

M_1 PS	NS Q_1	M_2 PS	NS Q_1
x_{10} = (A)	x_{11} = (A',A)	x_{20} = (B)	x_{21} = (B)
	x_{12} = (A)		x_{22} = (A',B)

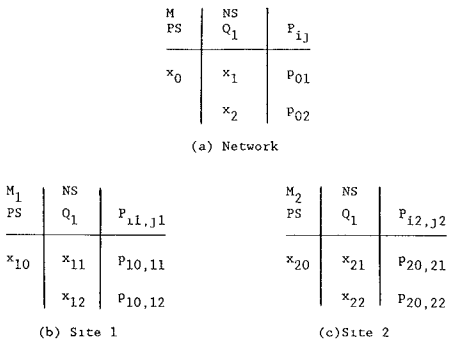
(c) Communicating finite state machines

Fig. 1 Single-join, single-query model Q_1.

complications as the relaying of data through intermediate, nonoperand sites (but this restriction may also be removed, as described in [15]).

A common alternative to the state-transition graph as a representation for finite-state machines is the state-transition table, as shown in Figure 1(b) for Q_1, with the states transposed as row vectors. Projecting over the first and second components, respectively, of the state yields the component state-transition tables of Figure 1(c), which correspond to finite-state machine models for site processors 1 and 2, such that the original network machine M is the composite machine of the site machines M_1 and M_2 [25, p. 386]. Note that the states x_{ij} of the site machines are double subscripted, first by the subscript i of the site, and then by the subscript j of the corresponding network state. Thus, when site machines 1 and 2 are in their respective initial states x_{10} and x_{20}, M_1 may undergo transition to x_{11}, in which case it receives the message $B2:1$ from site 2, or it may transmit the message $A1:2$ to site 2 and undergo transition to state x_{12}. Similarly, M_2 may send and receive messages to and from M_1, and it may change state accordingly. In this way, the model may be regarded as a network of communicating finite-state machines M_1 and M_2 [7].

Suppose now that a transition probability is associated with each transition arc of a state-transition graph, or equivalently, that a third column is added to the state-transition table for M, as shown in Figure 2(a). That is, let p_{ij} denote the conditional, time-invariant probability that the system undergoes



M PS	NS Q_1	p_{ij}
x_0	x_1	p_{01}
	x_2	p_{02}

(a) Network

M_1 PS	NS Q_1	$p_{i1,j1}$
x_{10}	x_{11}	$p_{10,11}$
	x_{12}	$p_{10,12}$

(b) Site 1

M_2 PS	NS Q_1	$p_{i2,j2}$
x_{20}	x_{21}	$p_{20,21}$
	x_{22}	$p_{20,22}$

(c) Site 2

Fig. 2. Sequential stochastic automata.

transition to state x_j, given that it is in state x_i. Thus p_{01} and p_{02} represent the relative frequencies with which sites 1 and 2 are selected for computing the join A', and any value assignment to the probability distribution $\{p_{01}, p_{02}\}$ constitutes a query-processing strategy for the query Q_1, provided that we have:

$$\begin{aligned} p_{01} + p_{02} &= 1 \qquad \text{(normalization)} \\ p_{01}, p_{02} &\geq 0 \qquad \text{(nonnegativity)}. \end{aligned} \tag{2.1}$$

It will be assumed that the transition probabilities of all state-transition models in this paper are also the transition probabilities of a finite Markov chain, for each input query type, which is equivalent to the assumption that the join process is Markovian. (This assumption may be relaxed, and a similar analysis may be carried out [15].) Note that, unlike the imbedded Markov chains of queueing theory [23], no explicit relationship between "transition time" (measured in units of transition steps) and real time is assumed for this paper (i.e., the real time between transitions of the abstract Markov model of this paper is undefined). The network machine M together with the transition probabilities p_{ij} then constitute a nondeterministic, finite-state sequential machine, with Markovian transition probabilities or a "sequential stochastic automaton" (SSA) [3].

The site machines M_1 and M_2 may be similarly regarded as SSAs. Consider a particular tagged arrival of Q_1 in the input stream to the system. If M_1 undergoes transition from x_{10} to x_{11}, then the join A' is computed at M_1, while M_2 contributes no processing towards the execution of the tagged query Q_1 (i.e., $x_{20} = x_{21}$). Although real-time processor 1 may execute A' while processor 2 is busy performing some other task, the abstract automata M_1, M_2, and M of the model execute transition steps in parallel with respect to the processing of tagged query Q_1, as shown in Figure 2, so that we have:

$$\begin{aligned} p_{0j} &= p(x_j|x_0) = p(x_{1j}, x_{2j}|x_{10}, x_{20}) \\ &= p(x_{1j}|x_{10}) = p(x_{2j}|x_{20}), \qquad j = 1,2 \end{aligned}$$

or

$$p_{0j} = p_{10,1j} = p_{20,2j}, \qquad j = 1,2. \tag{2.2}$$

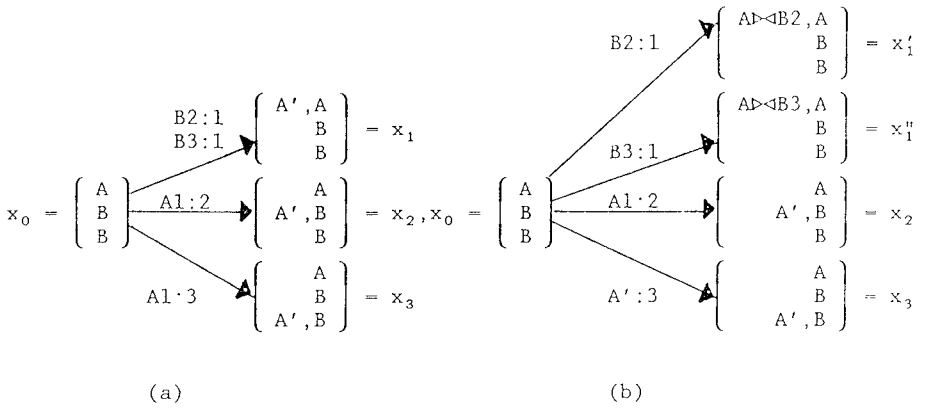


Fig. 3. State splitting for data replication.

Hence, the global strategy p_{0j} for M and the local strategies $p_{10,1j}$, $p_{20,2j}$ for M_1 and M_2 are all equivalent.

The model is also easily extended to networks having more than two sites and to systems with data replication, by means of the following "state-splitting" technique. Suppose, for example, that the network has three sites and that two copies of relation B are located at sites 2 and 3, so that the initial state x_0 equals (A, B, B). Thus, the join A' may be computed at site 1 by transferring relation B from either site 2 or site 3 to site 1, as indicated by the two message labels along the transition arc from x_0 to x_1 in Figure 3(a). In order to distinguish between these two alternatives, state x_1 may be split into two new states x_1', x_1'', as shown in Figure 3(b). Therefore, the query-processing distribution for this example consists of four components $\{p_{01}', p_{01}'', p_{02}, p_{03}\}$.

2.2 The Linear Programming Problem

The query-processing distribution $\{p_{01}, p_{02}\}$ defined in the previous section represents a family of strategies for scheduling the processing of a single query Q_1. In order to determine a best strategy in some sense from among all those satisfying (2.1), a suitable performance criterion must be selected. In this section, the system throughput is proposed as a performance measure. However, it will be convenient to determine first the system's mean processing time as a key performance parameter.

The transition probabilities provide an effective means for defining the expected processing time for Q_1 at each site. More specifically, delays due to query processing can be associated with each node of the state-transition graph corresponding to the computation of a component join. The expected delay due to computing the join is then just the product of the delay and the corresponding transition probability. Accordingly, let $t_m(Ai, Bj)$ denote the delay, or join-processing time due to computing $A \bowtie B$ at site m, where Ai denotes the event that relation A is located at site i and Bj the event

that relation B is at site j. Thus, as a function of random events, t_m is a random variable, and by the transition rule for Q_1 we have:

$$m = \begin{cases} i, & \text{if } Ai \bowtie Bj \text{ takes place at site } i, \\ j, & \text{if } Ai \bowtie Bj \text{ takes place at site } j, \\ i = j, & \text{if both } A, B \text{ are located at site } m. \end{cases}$$

A variety of techniques have been suggested for estimating the local processing cost t_m, usually based on the expected size of the operand relations A, B and measured in terms of number of disk accesses, CPU time, etc. [32, 41]. For this paper, only coarse estimates of join-processing times are necessary.

In addition to associating join-processing times with the nodes of the state-transition graph, it is similarly possible to assign communication costs to the arcs of the graph. This type of delay, denoted $c_{ij}(R)$, is the total time required to transfer relation R from site i to site j. Thus, for example, in order that $A' = A \bowtie B$ be performed at site 1 for query Q_1, the relation B must be routed first from site 2 to site 1, incurring the communication cost $c_{21}(B)$. Typically, communication costs are considered negligible in local-area networks, but may dominate the overall processing time in long-haul networks [13, p. 592].

For the query type Q_1, let T_i, $(i = 1, 2)$, denote the total processing time (join and communication) required to compute A' in state i, so that we have:

$$\begin{aligned} T_1 &= t_1(A1, B2) + c_{21}(B), \\ T_2 &= t_2(A1, B2) + c_{12}(A), \end{aligned} \tag{2.3}$$

which will be assumed constants for query optimization purposes. Thus, by Equation (2), the mean processing time τ_i at site i is given by

$$\tau_i = T_i p_{0i}, \qquad i = 1, 2. \tag{2.4}$$

It should be emphasized that the processing time T_i depends on the initial state x_0 and that τ_i is therefore a conditional mean. Note also that τ_i is regarded as an abstract "cost" to the automaton M_i.

Suppose now that input queries of type Q_1 arrive at the system at average intervals of length δ and that successive inputs are statistically independent, so that the system may be said to operate under stochastic load with mean interarrival time δ.

Since inputs arrive at average intervals of length δ, it seems reasonable to require that none of the processors in the network be allowed to take longer on the average than the period δ to execute its task. If it did, the cumulative delay at each site due to queueing could increase indefinitely, requiring the use of infinite buffer storage at each site. Since, in practice, only finite buffer storage is ever available, the system may be regarded as overloaded if the mean processing time τ_i is permitted to exceed δ at any site. In fact, for either discrete or continuous time input streams, it can be shown under quite general conditions that such overload can be avoided if the inequalities

$$\tau_i \leq \Delta < \delta \tag{2.5}$$

are satisfied [35], where $\triangle$ represents a common upperbound on τ_i, for each processor i in the network. Inequalities (2.5) will accordingly be called overload constraints. Note that input process needs not be stationary for Inequalities (2.5), in contrast to Lindley's analogous G/G/1 stability condition for queueing theory [29].

Thus, in order to maximize the system throughput, or query-processing capacity $\lambda = 1/\delta$, the system's mean interarrival time $\triangle$ may be minimized, where $(\delta - \triangle) > 0$ is chosen to be sufficiently large to provide for adequate, yet reasonable finite buffer storage requirements (see also [35]). The choice of $\triangle$ as objective function leads quite naturally to an optimization model that will be referred to as the *stochastic query optimization model.* By (2.4) and Inequalities (2.5), the stochastic query optimization problem for Q_1 assumes the form

$$\min \triangle$$

subject to

$$\begin{aligned} \tau_1 &= T_1 p_{01} \leq \triangle \\ \tau_2 &= T_2 p_{02} \leq \triangle \\ p_{01} &+ p_{02} = 1 \\ p_{01}, p_{02} &\geq 0 \end{aligned} \tag{2.6}$$

where p_{01}, p_{02} and $\triangle$ are the decision variables (i.e., the unknowns) and where T_1, T_2 are assumed known constants. This is a linear programming problem with a simple analytical solution. By eliminating $p_{01} = 1 - p_{02}$ from (2.6), three constraints remain:

$$\begin{aligned} T_1 - T_1 p_{02} &\leq \triangle \\ T_2 p_{02} &\leq \triangle \\ p_{01}, p_{02} &\geq 0 \end{aligned} \tag{2.7}$$

From the theory of linear programming [34], the minimum value of $\triangle$ occurs at an extreme point (i.e., vertex) of the region of feasible solutions defined by Constraints (2.7). But there are only three such extreme points, and it is easily verified (e.g., graphically) that the optimal solution is given by the following:

$$p_{01}^* = \frac{T_2}{T_1 + T_2}, \quad p_{02}^* = \frac{T_1}{T_1 + T_2}, \quad \triangle^* = \frac{T_1 T_2}{T_1 + T_2} \tag{2.8}$$

for which both site means τ_1^*, τ_2^* attain the minimum $\triangle^*$, i.e., the optimal solution is load-balanced, with both overload constraints "active" [31]. Note that this solution is a mixed strategy for query processing, in the sense that site 1 is selected to perform the join $A' T_2$ out of $(T_1 + T_2)$ times, on the average, while site 2 is chosen T_1 out of $(T_1 + T_2)$ times, and neither site is selected always in favor of the other. That is, Strategy (2.8) is not a "pure," or deterministic, query-processing strategy. In particular, if T_1 is less than T_2,

the pure strategy $p_{01} = 1$, $p_{02} = 0$ yields interarrival time $\triangle_p$ when substituted in Equation (2.6), where we have

$$\triangle^* = \frac{T_2}{T_1 + T_2} \triangle_p < \triangle_p. \tag{2.9}$$

Thus, the use of Strategy (2.8) is clearly preferable from a stochastic point of view, while the use of a pure strategy could result in severe degradation in throughput, particularly if we have $T_1 \gg 0$.

This simple example serves to illustrate the general principle that the presence of overload constraints is incompatible with the optimality of pure strategies [14, 38]. In general, stochastic operation cannot be avoided unless suboptimality is considered acceptable. This principle applies also to constrained optimization with respect to other objective functions (e.g., minimization of the total mean processing time, subject to overload constraints). Although it is usually possible to restrict the solutions of mathematical programming problems to integer values (see, e.g., [34]), mixed real-valued strategies for stochastic query optimization problems will be of primary interest in this paper.

The single-join, single-query model for Q_1 is also useful for describing the implications of stochastic query optimization on both distributed database design and performance analysis. For the design problem, two optimal solutions could be computed, with respect to $x_0 = (A, B)$ and $x_0' = (B, A)$. Presumably, the design x_0 would be chosen if $\triangle^*$ is less than $\triangle'^*$. As the number of sites, replicated relations, queries, etc., increases, the number of initial states that would have to be considered in such a comparative design study would become prohibitively large (as discussed further in Section 4). However, in practice, the number of competitive designs is often limited by application-dependent considerations, such as local autonomy constraints, so that the optimal location of the component relations A, B with respect to the system's capacity for processing Q_1 is given by

$$x_0^* = \arg \min_{x_0} \triangle^*(x_0) p(x_0).$$

In contrast to the distributed database design problem, a performance analysis would typically assume that the total query-processing capacity of the system is given by the total probability mean

$$\sum_{x_0} \triangle^*(x_0) p(x_0)$$

where $\triangle^*(x_0)$ denotes the optimal mean interarrival time conditioned on x_0, and the index x_0 of the sum ranges over a prescribed set of initial states for all input queries under consideration. In practice, operational query scheduling should be monitored and augmented by decision-making procedures based on statistical performance histories [32]. Furthermore, (2.9) indicates that such procedures would be particularly relevant for scheduling high-frequency queries that reference database relations that are expensive to process.

To illustrate the extension of the stochastic query optimization model to more than two sites, consider the second example of Figure 3 in Section 2.1, in which there are three sites and in which the initial state x_0 equals (A, B, B). This is again a linear programming problem given by

$$\begin{aligned} &\min \Delta \\ \tau_1 &= T_1' p_{01}' + T_1'' p_{01}'' \le \Delta \\ \tau_2 &= T_2 p_{02} \le \Delta \\ \tau_3 &= T_3 p_{03} \le \Delta \\ & p_{01}' + p_{01}'' + p_{02} + p_{03} = 1 \\ & p_{01}', p_{01}'', p_{02}, p_{03} \ge 0 \end{aligned} \tag{2.10}$$

where

$$\begin{aligned} T_1' &= t_1(A1, B2) + c_{21}(B), \\ T_1'' &= t_1(A1, B3) + c_{31}(B), \\ T_2 &= t_2(A1, B2) + c_{12}(A), \\ T_3 &= t_3(A1, B3) + c_{13}(A). \end{aligned}$$

This problem also has an analytical solution, but unlike the two-site model, it does not have a unique solution, since there are five unknowns (p_{01}', p_{01}'', p_{02}, p_{03} and Δ) in four active constraints (excluding the nonnegativity constraints). However, by eliminating p_{02} and p_{03}, all points along the line segment,

$$p_{01}' = -\frac{T_1'(T_2 + T_3) + T_2 T_3}{T_1'(T_2 + T_3) + T_2 T_3} p_{01}'' + \frac{T_2 T_3}{T_1'(T_2 + T_3) + T_2 T_3} \tag{2.11}$$

for which $0 \le p_{01}', p_{01}'' \le 1$, are optimal. Additional constraints, such as those discussed in [15], are required if a unique optimum is desired.

3. GENERAL SINGLE-JOIN MODELS

3.1 Sequential Operation

One of the main incentives for implementing a database as a distributed system is the need for the use of the database as a shared resource. Frequently, more than one transaction may be executing several different types of queries against the same database at the same time. Two modes of multiple-query distributed database operation are identified in this paper: sequential and parallel operation. In this section, the sequential mode, in which queries arrive separately, one after the other, with an average interarrival time of length δ, is presented for the general case of N query types and a network of $M \ge N$ processor types. It is assumed that no writes (updates) may be made to the relations referenced by the N query types while they are being processed.

Let

$$Q_j = A_j \bowtie B_j = A'_j, \qquad j = 1, 2, \ldots, N$$

denote N distinct query types (although the relations A_j and B_j need not be distinct).

Algorithm 3.1. For constructing the network state-transition graph (table) for N sequential query types and $M \geq N$ sites, given initial state x_0.

For $j = 1, 2, \ldots, N$
  Set $i := 0$
For $m = 1, 2, \ldots, M$

Step 1. **If** $A_j, B_j \notin x_0(m)$, **do** nothing (i.e., increment m and repeat step 1).

Step 2. **If** $A_j, B_j \in x_0(m)$, **then set** $i := i + 1$ **and add** a transition arc (row) from x_0 to x_{ij}, where,

$$x_{ij} = \begin{cases} \{A'_j\} \cup x_0(m), \\ x_0(\ell), \ell \neq m, \ell = 1, 2, \ldots, M \end{cases}$$

Step 3. **If** $A_j \in x_0(m)$ **and** $B_j \notin x_0(m)$, **then**:
For $n = 1, 2, \ldots, M, (n \neq m)$
If $B_j \in x_0(n)$, **then**:

Step 3a. **Set** $i := i + 1$ **and add** a transition arc (row) from x_0 to x_{ij}, labeled $B_j n : m$, where,

$$x_{ij} = \begin{cases} \{A_j \bowtie B_j n\} \cup x_0(m) \\ x_0(k), k \neq m, \quad k = 1, 2, \ldots, M \end{cases}$$

Step 3b. **Set** $i := i + 1$ **and add** a transition arc (row) from x_0 to x_{ij}, labeled $A_j m : n$, where,

$$x_{ij} = \begin{cases} \{A'_j\} \cup x_0(n) \\ x_0(k), k \neq n, \quad k = 1, 2, \ldots, M \end{cases}$$

Step 4. **If** $B_j \in x_0(m)$ **and** $A_j \notin x_0(m)$, **then** execute Step 3 with A_j, B_j interchanged.

End

This algorithm generates all states that contain a single join in one state component, according to the transition rule that a join may be executed at operand sites, only, as described in Section 2.1. In particular, Step 2 of the algorithm is concerned with the case in which both operands of a join are located at the same site, and Steps 3 and 4 handle the two possible cases in which the operands are located at different sites. In order that this algorithm be consistent with the state-splitting convention for data replication, as described in Section 2.1, Algorithm 3.1 generates a state-transition graph that is actually a tree. That is, there is only one incoming transition arc per noninitial state, and in Step 3a of the algorithm, the join $A_j \bowtie B_j m$ is labeled by the transmitting site m to distinguish joins formed by sending B from different sites.

LEMMA 3.1. *The query-processing strategies for the network and site machines, obtained by projecting over each site component of the network machine, are equal.*

PROOF. Since the site machines are defined so that the projection operator $\pi_m(x) = x_m$ of network state x to site state x_m is both a one-to-one and onto mapping and because there is a site machine transition from x_m to y_m iff there is a network transition from $x = \pi_m^{-1}(x_m)$ to $y = \pi_m^{-1}(y_m)$, the mapping π_m is a state machine isomorphism [19, p. 19] for each site $m = 1, 2, \ldots, M$ (see, e.g., Figure 3). Hence, the transition probability distributions of the site and network machines are defined on isomorphic sample spaces, and the site machine transition probabilities constitute the marginal distributions of the network transition probability distribution. Thus, it follows that

$$p_{0i} = p_{10,1i} = p_{20,2i} = \cdots = p_{M0,Mi} \tag{3.1}$$

where i ranges over the states generated by Algorithm 3.1. □

This lemma is really a consequence of the manner in which the site machines are defined and will hold for all state-transition models of this paper. Note that (3.1) indicates that it suffices to consider just the network transition probabilities as decision variables for stochastic query optimization purposes.

For the following proposition, let q_j denote the probability that a given query type is of type Q_j, where $q_1 + q_2 + \cdots + q_N$ equals 1. It will be assumed that q_j is known and that the input to the system may be regarded as a merged stream of mixed query types. For example, for continuous-time Poisson inputs, such a merged stream may be obtained by superposition and is again Poisson with rate $\lambda = 1/\delta = \sum \lambda_i$ and $q_i = \lambda_i/\lambda$, as described in [17, p. 11]. For discrete-time Bernoulli inputs, merging may be carried out as described in the Appendix. Also, it should be recalled that, for systems that cache joins, Q_j represents the computation of the j-th join due to update of its component relations, as described in Section 2.1. For such systems, q_j denotes the corresponding fraction of the update load due to Q_j, as in [12].

PROPOSITION 3.1. *The general, sequential single-join stochastic query optimization model for N queries and M sites defines a linear programming problem that may be decomposed into N independent linear programming subproblems.*

PROOF. It will be convenient to define first the following sets of state indexes,

$$\begin{aligned} K(j) &= \{i | A'_j \epsilon x_{ij}(m), \quad \text{for some } m, 1 \le m \le M\}, \\ K(j,m) &= \{i | A'_j \epsilon x_{ij}(m)\}, \quad j = 1, 2, \ldots, N \\ &\qquad\qquad m = 1, 2, \ldots, M. \end{aligned} \tag{3.2}$$

That is, $K(j)$ is the set of indexes i of states generated by Algorithm 3.1 for query Q_j, and $K(j,m)$ is the set of indexes i of those states generated by Algorithm 3.1 that contain A'_j in component m, for query Q_j (i.e., with data

replication, there may be more than one such state). Then the overload constraints have the form

$$\tau_{mj} = \sum_{i \in K(j,m)} T_{ij}(A'_j) p_{0,ij}, \qquad \begin{matrix} j = 1, 2, \ldots, N \\ m = 1, 2, \ldots, M \end{matrix} \tag{3.3}$$

where $p_{0,ij}$ is by Lemma 3.1 the network transition probability associated with a change of state from x_0 to x_{ij}, where τ_{mj} is the mean processing time at site m, conditioned on the event that a query is of type Q_j, and where $T_{ij}(A'_j)$ is the total (join and communication) processing time for A'_j, as defined in Section 2.2. The linear programming problem defined by the model is obtained as in Section 2.2 by minimizing the mean interarrival time $\triangle$, subject to the system's overload, normalization, and nonnegativity constraints, i.e.,

$$\min \triangle$$

subject to

$$\begin{aligned} \sum_{j=1}^{N} q_j \cdot \tau_{mj} &\leq \triangle, \quad m = 1, 2, \ldots, M \\ \sum_{j=1}^{N} \sum_{i \in K(j)} p_{0,ij} &= 1 \\ p_{0,ij} &\geq 0, \quad \begin{matrix} j = 1, 2, \ldots, N \\ i \in K(j) \end{matrix} \end{aligned} \tag{3.4}$$

where i again ranges over the states generated by Algorithm 3.1. But since $\triangle$ can always be written as the sum of conditional means,

$$\triangle = \sum_{j=1}^{N} q_j \triangle_j \tag{3.5}$$

where $\triangle_j$ is the mean interarrival time conditioned on query type Q_j, $\triangle$ is separable in its conditional means $\triangle_j$. By the decomposition principle of linear programming [21, p. 520], the original problem (3.4) may be divided into subproblems by query type, i.e., for each $j = 1, \ldots, N$,

$$\min \triangle_j$$

subject to

$$\begin{aligned} \tau_{mj} &\leq \triangle_j, \qquad m = 1, 2, \ldots, M \\ \sum_{i \in K(j)} p_{0,ij} &= 1 \\ p_{0,ij} &\geq 0, \qquad i \in K(j) \end{aligned} \tag{3.6}$$

constitutes a linear programming subproblem in a set of decision variables disjoint from the decision variables of the other subproblems. Thus, each such subproblem may be solved independently of the others, and the total mean can be computed then by deconditioning on query type, i.e., by the probability sum (3.5). □

COROLLARY 3.1. *(a) The optimal solution of Proposition* 3.1 *is the minimax processing-time solution, which is load-balancing*; *(b) The optimal location of component relations with respect to query-processing capacity is given by*

$$\arg \min_{x_0} \triangle^*(x_0)p(x_0).$$

Part (a) of this corollary follows from the theory of linear programming and may also be shown (with some minor modifications) as in [16, pp. 71–72]. The minimax solution is load-balancing since all mean processing times τ_i achieve the same maximum throughput $\triangle^*$. Part (b) simply states that the component relations are best located according to the state with the greatest throughput capacity.

Although the optimization problem of this proposition is based on the assumption of sequential arrivals, it is mixed-mode with respect to query processing, in the sense that no specific assumptions are made concerning the order of arrivals and their execution. Thus, for $N = 2$, the execution of Q_1 may precede the execution of Q_2 (or vice-versa), on the same or on different machines. Moreover, since all of the overload constraints hold for all of the processors, the execution of Q_1 and Q_2 may overlap in real time; i.e., they may be executed concurrently. For the case of discrete-time input, the concurrent execution of several queries can be formulated in a more explicit manner, as described in the next section.

Example 3.1. Let $N = M = 2$ and

$$Q_1 = A \bowtie B = A', \qquad Q_2 = B \bowtie C = B'$$

and suppose we have $x_0 = ((A, C), (B))$. The state-transition graph for this model is shown in Figure 4, and the mean processing times for the joins A', B' at sites 1 and 2 are given by

$$T_{11}(A') = t_1(A1, B2) + c_{21}(B), \qquad T_{12}(B') = t_1(B2, C1) + c_{21}(B)$$
$$T_{21}(A') = t_2(A1, B2) + c_{12}(A), \qquad T_{22}(B') = t_2(B2, C1) + c_{12}(A),$$

as described in Section 2.2. The two linear programming subproblems for this example are

$$\min \triangle_1$$
$$\tau_{11} = T_{11}(A')p_{0,11} \leq \triangle_1$$
$$\tau_{21} = T_{21}(A')p_{0,21} \leq \triangle_1$$
$$p_{0,11} + p_{0,21} = 1$$
$$p_{0,11}, p_{0,21} \geq 0$$
$$\min \triangle_2$$
$$\tau_{12} = T_{12}(B')p_{0,12} \leq \triangle_2$$
$$\tau_{22} = T_{22}(B')p_{0,22} \leq \triangle_2$$
$$p_{0,12} + p_{0,22} = 1$$
$$p_{0,12}, p_{0,22} \geq 0$$

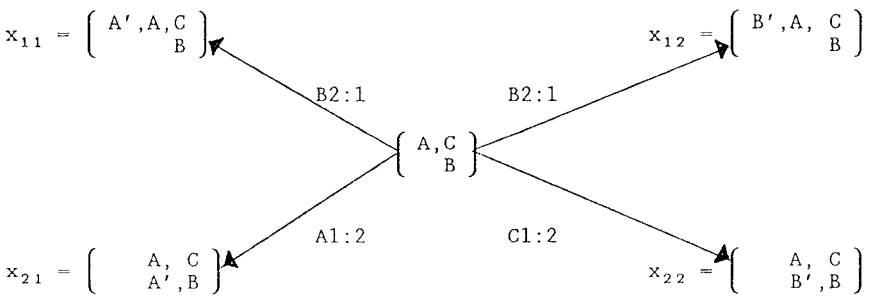


Fig. 4. State-transition graph for Q_1, Q_2.

with optimal solutions

$$p^*_{0,11} = \frac{T_{21}(A')}{T_{11}(A') + T_{21}(A')}, \qquad p^*_{0,12} = \frac{T_{22}(B')}{T_{12}(B') + T_{22}(B')},$$

$$p^*_{0,21} = \frac{T_{11}(A')}{T_{11}(A') + T_{21}(A')}, \qquad p^*_{0,22} = \frac{T_{12}(B')}{T_{12}(B') + T_{22}(B')},$$

$$\triangle^*_1 = \frac{T_{11}(A')T_{21}(A')}{T_{11}(A') + T_{21}(A')}, \qquad \triangle^*_2 = \frac{T_{12}(B')T_{22}(B')}{T_{12}(B') + T_{22}(B')}.$$

Note that the two optimization problems are completely independent of each other, having no common variables. Their separate solutions may be combined by the probability summation in (3.5), yielding

$$\triangle^* = q_1 \frac{T_{11}(A')T_{21}(A')}{T_{11}(A') + T_{21}(A')} + q_2 \frac{T_{12}(B')T_{22}(B')}{T_{12}(B') + T_{22}(B')}. \tag{3.7}$$

This example also provides an opportunity to illustrate the use of a suboptimal approach to developing query plans, which first selects the "best" deterministic plan for each query type, as in R^* [32], and then seeks to control the load on the system by choosing which query should be executed, based on the given state of the system. In terms of the model's parameters, this approach selects first the optimal pure strategy for each conditional subproblem, followed by solution of the overall problem, deconditioned by query type, for the probabilities q_i as unknown load control variables. That is, under this interpretation of the model, q_i represents the probability that the system schedules Q_i for execution, given initial state x_0.

For Example 3.1, suppose that we have

$$T_{21}(A') = \alpha T_{11}(A'), \qquad \alpha > 1$$
$$T_{12}(B') = \beta T_{22}(B'), \qquad \beta > 1$$

so that $T_{11}(A')$ is less than $T_{21}(A')$ and $T_{22}(B')$ is less than $T_{12}(B')$. Substituting the corresponding pure (deterministic) strategy $p_{0,11} = p_{0,22} = 1$,

$p_{0,21} = p_{0,12} = 0$ into the above conditional subproblems for $\triangle_1$ and $\triangle_2$, followed by deconditioning on query type, yields the linear program,

$$\begin{aligned} &\min \triangle_p \\ \tau_1 &= T_{11}(A')q_1 \leq \triangle_p \\ \tau_2 &= T_{22}(B')q_2 \leq \triangle_p \\ q_1 &+ q_2 = 1 \end{aligned}$$

with optimal solution,

$$q_1^* = \frac{T_{22}(B')}{T_{11}(A') + T_{22}(B')}, \quad q_2^* = \frac{T_{11}(A')}{T_{11}(A') + T_{22}(B')},$$

$$\triangle_p = \frac{T_{11}(A')T_{22}(B')}{T_{11}(A') + T_{22}(B')}.$$

On substituting q_1^*, q_2^* in (3.7), the following inequality is obtained:

$$1 < \triangle_p / \triangle^* = \frac{\alpha}{1+\alpha} + \frac{\beta}{1+\beta} < 2.$$

Thus, even this simple example shows that the model provides a theoretical framework for comparing suboptimal query plans with optimal strategies.

3.2 Parallel Operation

The capacity of distributed database systems for load-sharing through parallel processing is of particular significance to the design and performance analysis of such systems. The sequential model of the previous section represents the processing the system undergoes if queries arrive one after the other, but not if they arrive at approximately the same time. Suppose that N queries of types $Q_1, Q_2, \ldots, Q_N$ are to be processed in parallel in the sense that each is processed at a different site (although not necessarily at the same time), as in [27]. In order to provide for explicit parallel query processing in a manner consistent with the previous section, let Q_{N+1} denote another aggregate query type, which occurs with probability q_{N+1}, $(q_1 + q_2 + \cdots + q_{N+1} = 1)$. Thus, the input stream consists of mixed arrivals of types $Q_1, Q_2, \ldots, Q_{N+1}$. For discrete-time Bernoulli inputs, q_{N+1} may be computed as described in the Appendix. Since the probability of simultaneous arrivals in a merged, continuous-time input stream is zero, this section is applicable primarily for discrete-time input streams. The state space for the parallel mode model for Q_{N+1} is based on that of Algorithm 3.1, but changed as follows:

Definition 3.1. The states of the *parallel-network machine* consist of all possible N-way site component-wise unions of single-join states $x_{k_j l}$,

$(j = 1, 2, \ldots, N)$ of the form

$$x_i = \bigcup_{J=1}^{N} x_{k_{jJ}}$$

such that

(1) no component of x_i contains more than one join A'_j, for some j, $(1 \le j \le N)$ and

(2) no particular join A'_j appears in more than one component of x_i.

This state space can be constructed algorithmically using conventional methods (e.g., [2, Chap. 4]). The transitions of the parallel network machine are defined by the transmission of the data transfers associated with each of the corresponding single-join transitions from x_0 to $x_{k_{jJ}}$, $(j = 1, 2, \ldots, N)$.

Example 3.2. Consider again the two-join, two-site model of Example 3.1, with queries

$$Q_1 = A \bowtie B = A', \qquad Q_2 = B \bowtie C = B'.$$

The state-transition graph for the parallel machine for Q_3 is shown in Figure 5, in which we have $x_{13} = x_{11} \cup x_{22}$ and $x_{23} = x_{12} \cup x_{21}$.

PROPOSITION 3.2. *The general, parallel single-join stochastic query optimization model defines a linear programming problem.*

PROOF. Let I denote the total number of parallel network machine states x_i, $(i = 1, 2, \ldots, I)$, and let $K(j, m)$ denote the index set of (3.2), restricted to parallel-network machine states. Then the linear programming problem defined by the parallel-network machine is given by

$$\min \Delta_{N+1}$$

subject to

$$\tau_{m,N+1} = \sum_{J=1}^{N} \sum_{i \in K(j,m)} T_{i(N+1)}(A'_j) p_{0,i(N+1)} \le \Delta_{N+1} \qquad m = 1, 2, \ldots, M$$

$$\sum_{i=1}^{I} p_{0,i(N+1)} = 1$$

$$p_{0,i(N+1)} \ge 0, \qquad i = 1, 2, \ldots, I. \quad \square$$

Example 3.3. For the model of Example 3.2, the stochastic query optimization problem for parallel processing of $Q_1 = A \bowtie B = A'$ and $Q_2 = B \bowtie C = B'$ is given by

$$\min \Delta_3$$

subject to

$$\tau_{13} = T_{13}(A') p_{0,13} + T_{23}(B') p_{0,23} \le \Delta_3$$

$$\tau_{23} = T_{13}(B') p_{0,13} + T_{23}(A') p_{0,23} \le \Delta_3$$

$$p_{0,13} + p_{0,23} = 1$$

$$p_{0,13}, p_{0,23} \ge 0$$

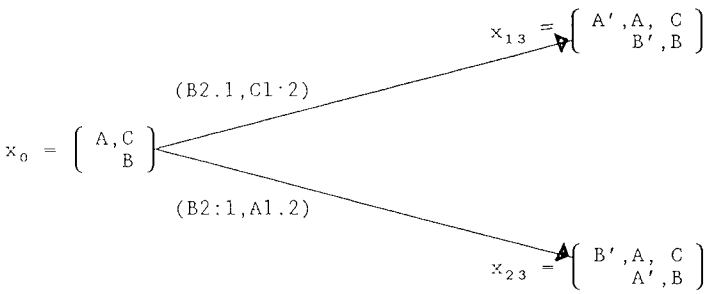


Fig. 5. Parallel-network machine for Q_3.

with optimal solution

$$p^*_{0,13} = [T_{23}(A') - T_{13}(B')]/D$$
$$p^*_{0,23} = [T_{13}(A') - T_{23}(B')]/D$$
$$\triangle^*_3 = [T_{13}(A')T_{23}(A') - T_{13}(B')T_{23}(B')]/D$$

where the denominator

$$D = [T_{13}(A') + T_{23}(A')] - [T_{13}(B') + T_{23}(B')]$$

must be nonzero.

The parallel-mode query type Q_{N+1} may be regarded as a conditioning variable, just like $Q_1, Q_2, \ldots, Q_N$ for the sequential mode, in the sense that the optimum interarrival time $\triangle^*_{N+1}$ may be combined with the results for sequential queries by extending the range of the probability sum (3.5) to include Q_{N+1}, i.e.,

$$\triangle^* = \sum_{J=1}^{N+1} q_J \triangle^*_J$$

and by normalizing the multipliers q_J. In this way, $\triangle^*$ provides a measure of the total query-processing capacity of the system due to both the sequential and parallel processing modes.

4. MULTIPLE-JOIN MODELS

In this section, the state-transition model is extended to queries involving more than one join, and the complexity of the model is discussed. Accordingly, suppose we have

$$Q_4 = A \bowtie B \bowtie C.$$

For query optimization purposes, it is customary to further define the multiple-join queries such as Q_4 by indicating which joins may take place pair-wise (as in, e.g., query graphs [4]). However, it will be convenient for the purpose of stochastic query optimization to simply enumerate all logically valid joins in the order in which they may be executed, as is common practice

for modeling precedence in scheduling theory [11]. For example, suppose that Q_4 has two valid execution sequences

$$QS_1 = (Q_4, S_1) = A \bowtie (B \bowtie C) = A \bowtie B' = A''$$
$$QS_2 = (Q_4, S_2) = (A \bowtie B) \bowtie C = A' \bowtie C = B'' \tag{4.1}$$

where S_1 and S_2 denote the sequences. That is, under sequence S_1, $B' = B \bowtie C$ is computed before $A'' = A \bowtie B'$, and under sequence S_2, $A' = A \bowtie B$ is computed before $B'' = A' \bowtie C$. In this way, the computation of multiple-join queries may be equivalently defined in terms of precedence of the join operator, without reference to the actual join clauses and attributes involved. Also, it will be assumed that any other execution sequence not enumerated in this manner is invalid. Thus, the sequence $(A \bowtie C) \bowtie B$ is invalid for Q_4, presumably due to an improper attempt to join A and C or $A \bowtie C$ and B.

The symbols QS_1, QS_2 will be regarded as subtypes of the query type Q_4, and, using the decomposition principle of separable nonlinear programming [31], the symbols will be used as conditioning variables that divide the original problem into a set of subproblems, in a manner similar in principle to the treatment of mixed query types in Section 3.1. Thus, sequencing comprises another level of decomposition in the divide-and-conquer paradigm proposed in this paper.

The state-transition graph for the sequence S_1 of Q_4 is shown in Figure 6. There are two stages of computation, one stage for each of the two joins of QS_1, with states x_{111}, x_{211} in the first stage (i.e., as in Section 3.3; the first subscript is the state index; the second the query type, or subtype; and the third the stage). The answer states $x_{112}, x_{212}, x_{312}$ may be generated by applying Algorithm 3.1 to the first stage states, with x_{111} and x_{211} as initial states and subject to the stage-2 join $A'' = A \bowtie B'$ as the input to the second stage, in a manner similar to Step 2b of Algorithm 3.2. More particularly, the transition arc from x_{111} to x_{112} is labeled "SS" (Same Site) to indicate that because both operands A and B' of the join $A'' = A \bowtie B'$ are located at site 1, the transition rule of Section 2.1 dictates that the join A'' also take place at site 1, with probability 1, given that the system is in state x_{111}. Similarly, given that the system is in state x_{211}, the transition rule of Section 2.1 applies again, causing a change of state to either x_{212} or x_{312}, with transition probabilities $p_{211,212}$ and $p_{211,312}$, respectively ($p_{211,212} + p_{211,312} = 1$). The state-transition graph for the sequences S_2 of Q_4 may be obtained in the same manner.

The optimal solution for this problem can be computed in a manner similar to that of Section 3. The stochastic query optimization problem for QS_1 is given by

$$\min \Delta_1$$
$$\tau_{11} = T_{111}(B')p_{0,111} + T_{112}(A'')p_{0,111} + T_{212}(A'')p_{0,211}p_{211,212} \leq \Delta_1$$
$$\tau_{21} = T_{211}(B')p_{0,211} + T_{312}(A'')p_{0,211}p_{211,312} \leq \Delta_1 p_{0,111} + p_{0,211}$$
$$= p_{211,212} + p_{211,312} = 1 \tag{4.6}$$

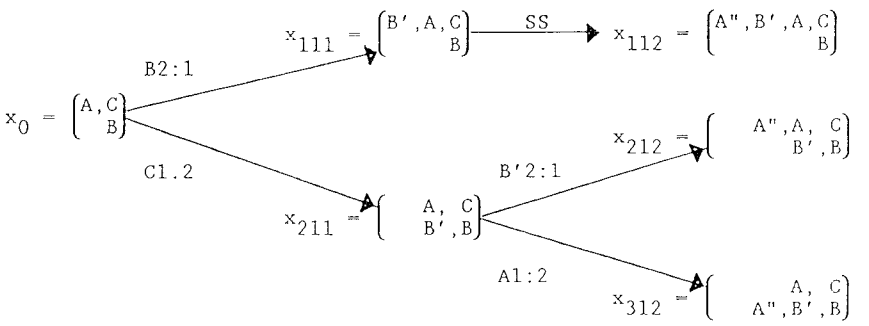


Fig. 6 State-transition graph for QS_1.

where, as usual, the decision variables must also be nonnegative. This problem is a nonlinear programming problem of a special kind, sometimes called *multilinear* (i.e., a function of n variables is said to be multilinear of degree n if it is a polynomial in which the variables are raised to powers of 0 and 1 only [16]). The problem has five unknowns ($p_{0,111}$, $p_{0,211}$, $p_{211,212}$, $p_{211,312}$, and $\triangle_1$) and its overload constraints are multilinear of degree two.

The multilinear optimization problem (4.6) has an analytical solution, which can be obtained directly from its Kuhn–Tucker conditions [22, 31], which are more conveniently expressed in vector notation. Accordingly, let

$$\begin{aligned} y &= (y_1, y_2, \ldots, y_5) \\ &= (p_{0,111}, p_{0,211}, p_{211,212}, p_{211,312}, \triangle_1) \\ h(y) &= (h_1(y), h_2(y)), g(y) = (g_1(y), g_2(y), \ldots, g_7(y)) \end{aligned}$$

where

$$\begin{aligned} f(y) &= y_5, \\ h_1(y) &= p_{0,111} + p_{0,211} - 1 = 0, \\ h_2(y) &= p_{211,212} + p_{211,312} - 1 = 0, g_1(y) = \triangle_1 - \tau_{11} \geq 0, \\ g_2(y) &= \triangle_1 - \tau_{21} \geq 0, \\ g_{2+j}(y) &= y_j \geq 0, j = 1, 2, \ldots, 5. \end{aligned}$$

Then the optimization problem (4.6) may be rewritten in the more compact form

$$\begin{aligned} &\min f(y) \\ &h(y) = 0 \\ &g(y) \geq 0. \end{aligned} \tag{4.7}$$

The Kuhn–Tucker conditions state that if the objective and constraint functions are differentiable and if y is a solution of (4.7), then vectors

$$u = (u_1, u_2, \ldots, u_7), \qquad v = (v_1, v_2)$$

exist such that we have

$$\begin{aligned} -\nabla f(y) + \sum_{j=1}^{7} u_j \nabla g_j(y) + \sum_{j=1}^{2} v_j \nabla h_j(y) &= 0, \\ \sum_{j=1}^{7} u_j g_j(y) &= 0, \\ u &\geq 0, \end{aligned} \tag{4.8}$$

where ∇ denotes the gradient operator (see, e.g., [22, p. 14]). There are two possible approaches for obtaining a solution by means of (4.8). First, the solution space may be divided into subspaces for which various combinations of the Lagrange multipliers v_j are positive or equal to zero (see [15] for the details of this technique). Also, on the other hand, it is possible to solve (4.8) by setting various combinations of constraints active and others inactive (e.g., [31]). In either case, a combinatorial solution is obtained with two possible global optima. In particular, if we have

$$a_1 = [T_{111}(B') + T_{112}(A'')], \qquad b_1 = T_{211}(B')$$
$$a_2 = T_{212}(A''), \qquad b_2 = T_{312}(A'')$$

then the optimal solution for QS_1 is the minimum of the three candidate solutions of Figure 7. Note that all three candidates have a pure (deterministic) component strategy for one stage and a mixed strategy for the other.

Since the stochastic optimization problem is symmetric in the sequences QS_1 and QS_2, the problem for QS_2 may be solved in exactly the same manner as for QS_1. The sequence(s) QS_ℓ corresponding to

$$\ell = \arg \min_{j=1,2} \{\triangle_j^*\}$$

is then the desired optimum. This solution is a stochastic analog for the minimum-cost solutions of many current optimizers (e.g., system R^* [32]).

Also, it is possible to seek approximate solutions to nonlinear programming problems, instead of computing the exact solution by means of the Kuhn–Tucker conditions. Many iterative search techniques have been developed [22], and many general-purpose nonlinear programming codes are commercially available [39]. In practice, the system designer may either choose a realistic query-processing strategy that is not necessarily better than all others, or the designer may conduct a search for a globally optimal one. Iterative techniques offer a means for computing a solution that is unequivocally feasible and locally optimal, while global optimization typically requires highly reliable data, which may be unavailable, or may require excessive computational effort. Thus, in practice, it may be preferable to formulate the design problem as a feasibility problem in mathematical programming, which can then be solved by an appropriate iterative method, or by suboptimization techniques, as in Example 3.2.

It is possible to extend the methods of this section to more than a single, multiple-join query in the manner of Section 3 [15]. However, due to the

	y_1^*	y_2^*	y_3^*	y_4^*	$y_5^*=\Delta_1^*$
	$\frac{b_1+b_2}{a_1+b_1+b_2}$	$\frac{a_1}{a_1+b_1+b_2}$	0	1	$\frac{a_1(b_1+b_2)}{a_1+b_1+b_2}$
$b_1>a_2$	$\frac{b_1-a_2}{a_1+b_1-a_2}$	$\frac{a_1}{a_1+b_1-a_2}$	1	0	$\frac{a_1b_1}{a_1+b_1-a_2}$
$a_2>b_1$	0	1	$\frac{b_1+b_2}{a_2+b_2}$	$\frac{a_2-b_1}{a_2+b_2}$	$\frac{a_2(b_1+b_2)}{a_2+b_2}$

Fig. 7. Candidate global optima for (QS_1).

complexity of the underlying state space of the model, this is computationally feasible for only a relatively small number of joins per query. More particularly, since the transition graph of the model is in general a tree (due to the state-splitting technique of Section 2.1), the (worst-case) complexity of the model for a single query (sub-) type with K stages and M sites with full replication is determined by the number of nodes of an M-ary tree with K levels; and, therefore, this worst-case complexity is of order $O(M^K)$ [33, p. 376]. That is, the complexity of the model is polynomial in the number of sites and exponential in the number of stages, restricting the number of joins in multiple-join queries. Furthermore, for discrete-time input of a query type having $N \leq M$ parallel joins, the total number of ways the N joins may be distributed over the M sites (with no more than one join per site per state) is equal to the number of permutations of M items N at a time, i.e., $M!/(M - N)!$. Thus, the overall worst-case complexity of the state space is of order $O(M!/(M - N)!)^K$, which effectively restricts the full discrete-input model of Section 3.2 to single-join queries or to a few high-frequency multiple-join queries.

5. CONCLUDING REMARKS

A general approach has been described for obtaining optimal stochastic query-processing strategies for distributed relational database systems. Starting from the algorithmic generation of a state-transition graph that defines the ways a query (or queries) may be processed by the system, the graph also defines a set of probability distributions that encompasses the query-processing strategies for transferring operand relations and computing the component joins of the query. Optimal values for each query-processing strategy could then be determined by the solution of standard mathematical programming problems. Also, it was shown that the state-transition approach may be carried out for more complex queries by means of divide-and-conquer techniques that decompose a large problem into a number of smaller subproblems. By conditioning the original problem on suitably chosen system parameters, such as query type and operational sequence, and by making use of the decomposition principles of mathematical programming,

the original stochastic query optimization problem could be factored into conditional subproblems, each of which could be optimized separately. It should be noted that much of the material devoted to optimizing throughput may be applied to maximizing system utilization, as well. The model may be extended also in a number of ways to accommodate application-dependent and network-topological constraints, as well as semijoin strategies [15].

The computational complexity of state-transition models was found to be polynomial in the number of sites in the data network, but to increase exponentially in the number of joins executed sequentially, and for discrete-time input, to increase factorially in the number of joins performed in parallel. In view of the breadth and complexity of distributed database systems, it is hoped that the theory may be developed to the point at which it may be used in conjunction with suitable suboptimal techniques and heuristic algorithms (e.g., [30]). It is anticipated that such an approach would be based on techniques for reducing the size of the state space, such as probability conditioning and state merging.

The main result of this paper is that the state-transition approach provides a general methodology for the design and analysis of distributed relational database systems for query processing. The flexibility of the approach is based on the generality of the probability calculus itself, which acts as a mapping of transition graphs into mathematical programming problems. It is expected that this mapping can be carried out by means of software that would first automatically compile network and database definitions into state-transition tables, from which stochastic query optimization problems could be generated and solved by means of mathematical programming techniques, as required. The single-join models of Section 3 may be particularly appropriate for design purposes. Since the single-join models may be always formulated as linear programming problems, they are readily solved and are amenable to a variety of sensitivity analysis options [21]. Moreover, since the computational complexity of linear state-transition models is polynomial in the number of sites in the network, very large data networks, requiring conceivably up to tens of thousands of variables and constraints [26], may be accommodated.

Before the techniques of the state-transition model can be applied to realistic distributed database systems, further research is still necessary. Since the analysis of this paper was based on the transient (i.e., initial state-dependent) behavior of the system, it would be natural to extend the analysis to its steady-state performance, as well. Such an extension would permit investigation of such topics as throughput degradation due to locking [1, 5, 12, 37], and alternative optimal and suboptimal long-term cache policies, including storage costs, nonhomogeneous input streams, etc..

In view of the potentially high order of computational complexity of state-transition models, further study of solution techniques for stochastic query optimization problems is indicated, as well. As shown in Section 4, such problems may be solved by direct combinatorial optimization techniques, yielding global optima, or by iterative nonlinear programming methods, which typically converge to local optima. Thus, the trade-off between

combinatorial search and large-scale approximation merits further investigation. In particular, it is expected that multiple-join problems of degree considerably higher than the second-degree model of Section 4 will have to be considered. However, it is anticipated that the use of customized nonlinear programming techniques that exploit the linearity of the objective function (if the mean interarrival time is to be minimized) and the multilinearity of the constraints of the stochastic query optimization problem will prove even more effective than the use of general-purpose algorithms (e.g., [39]). Practical suboptimal and heuristic techniques should be compared also with, and possibly used in conjunction with, the optimal strategies of this paper.

APPENDIX. BERNOULLI SEQUENCES

Classical queueing theory [23] is concerned with continuous-time models having nonnegative real-valued interarrival and service times. More recently, there has been some interest in the development of an analogous discrete-time theory [6]. Typical examples of discrete-time queueing systems include synchronous communication channels, interactive terminal systems, etc. [24].

The discrete-time analog for the continuous Poisson input to classical queues is the Bernoulli sequence, in which each input (customer) represents an independent Bernoulli trial, with probability λ of an arrival and probability $1 - \lambda$ of no arrival of a customer on each trial. Since the total number of arrivals by time $n \geq 0$ is a binomial random variable with parameters (n, λ), the interarrival time η between successive inputs has the geometric probability distribution

$$p(\eta = n) = \begin{cases} \lambda(1 - \lambda)^{n-1}, & \eta > 0 \\ 0, & \text{else} \end{cases} \tag{A.1}$$

with mean $1/\lambda$.

Unlike the Poisson processes, which have the desirable "reproductive" property that several Poisson input streams may be "merged" into a single Poisson stream by superposition (see, e.g., [17, p. 10]), the result of merging Bernoulli sequences is not again Bernoulli, a complication which presents a major obstacle in discrete-time queueing networks [24]. Fortunately, for the purposes of the present paper, it is always possible to merge Bernoulli sequences, provided that simultaneous arrivals are interpreted as an "aggregate arrival."

For example, let $N_1(k), N_2(k)$, $(k = 1, 2, \ldots)$ denote two independent Bernoulli sequences with parameters λ_1, λ_2, respectively. The merged stream for N_1, N_2 may then be regarded as a single, four-class multinomial stream [28, p. 172] with parameters

$$\begin{aligned} c_1 &= \lambda_1(1 - \lambda_2), && \text{for arrivals from } N_1, \\ c_2 &= (1 - \lambda_1)\lambda_2, && \text{for arrivals from } N_2, \\ c_3 &= \lambda_1\lambda_2, && \text{for simultaneous arrivals from both } N_1 \text{ and } N_2, \\ c_4 &= (1 - \lambda_1)(1 - \lambda_2), && \text{for no arrival.} \end{aligned}$$

The equivalent Bernoulli sequence with mixed arrivals, including the "aggregate arrivals" corresponding to c_3, has parameter

$$\lambda = 1 - c_4 = c_1 + c_2 + c_3 = 1 - (1 - \lambda_1)(1 - \lambda_2),$$

and the probability that a given arrival is from the sequence N_i is

$$q_i = c_i/\lambda, \quad i = 1, 2, 3,$$
$$q_1 + q_2 + q_3 = 1,$$

where N_3 represents the sequence of simultaneous arrivals. In general, if m Bernoulli sequences are merged, the composite stream is again Bernoulli with parameter $1 - \prod_{i=1}^{m}(1 - \lambda_i)$. The merged stream then contains all possible combinations of simultaneous arrivals, formed by defining each of the $\sum_{i=2}^{m}\binom{m}{i}$ combinations of simultaneous arrivals from the m input streams as an aggregate arrival.

Since parallel query processing is one of the main topics of this paper, it is necessary to account for the presence of simultaneous arrivals in a discrete-time input stream. This is done in Section 3.2 by defining a special query type for each combination of simultaneous arrivals and by conditioning on query type as a natural part of the general solution procedure.

REFERENCES

1. AGRAWAL, R., CAREY, M. J., AND LIVNY, M. Concurrency control performance modeling: Alternatives and implications. *ACM Trans. Database Syst. 12*, 4 (1987), 609–654.
2. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. E. *Data Structures and Algorithms.* Addison-Wesley, Waltham, Mass., 1982.
3. ARBIB, M. A. *Theories of Abstract Automata.* Prentice-Hall, Englewood Cliffs, N.J., 1969. Ch. 9, p. 324.
4. BERNSTEIN, P. A., AND CHU, D.-M., W. Using semi-joins to solve relational queries. *J. ACM 28*, 1 (Jan. 1981), 25–40.
5. BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN, N. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Waltham, Mass., 1987.
6. BHARATH-KUMAR, K. Discrete-time queueing systems and networks. *IEEE Trans. COM-28*, 2 (Feb. 1980), 260–263.
7. BRAND, D., AND ZAFIROPOULO, P. On communicating finite state machines. *J. ACM 30*, 2 (Apr. 1983), 323–342.
8. CERI, S., AND PELAGATTI, G. *Distributed Databases—Principle Systems.* McGraw-Hill, New York, 1984.
9. CHEN, P. P., AND AKOKA, J. Optimal design of distributed information systems. *IEEE Trans. COM-29*, 12 (1980), 1068–1080.
10. CODD, E. F. A relational model for large shared databases. *Commun. ACM 13*, 6 (June 1970), 377–389.
11. COFFMAN, E. G., JR. (ED.). *Computer and Job-Shop Scheduling Theory.* Wiley, New York, 1976.
12. COFFMAN, E. G., GELENBE, E., AND PLATEAU, B. Optimization of the number of copies in a distributed database. *IEEE Trans. SE-7*, 1 (Jan. 1981), 78–84.
13. DATE, C. J. *An Introduction to Database Systems*, Fourth ed., Vol. I. Addison-Wesley, Waltham, Mass., 1986.
14. DRENICK, P. E., AND DRENICK R. F. A design theory for multi-processing computing systems. *Large Scale Syst. 12* (1987), 155–172.
15. DRENICK, P. E. Stochastic query optimization in distributed databases. Ph.D. dissertation, Polytechnic Univ., Brooklyn, N.Y., 1988.

16. DRENICK, R. F. *A Mathematical Organization Theory.* Elsevier, New York, 1986.
17. GELENBE, E., AND MITRANI, I. *Analysis and Synthesis of Computer Systems.* Academic Press, New York, 1980.
18 GOLDBERG, M., AND ROBSON, D. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley, Waltham, Mass., 1983.
19. HARTMANIS, J., AND STEARNS, R. E. *Algebraic Structure Theory of Sequential Machines.* Prentice-Hall, Englewood Cliffs, N J , 1966.
20. HEVNER, A. R., AND YAO, S. B. Query processing on a distributed database. In *Proceedings of the 3rd Workshop on Distributed Databases and Computer Networks.* 1978.
21 HILLIER, F. S., AND LIEBERMAN, G. J. *Introduction to Operations Research.* Holden-Day, San Francisco, 1967.
22. JACOBY, S. L. S., KOWALIK, J. S., AND PIZZO, J T. *Iterative Methods for Nonlinear Optimization Problems.* Prentice-Hall, Englewood Cliffs, N.J., 1972
23 KLEINROCK, L. *Queueing Systems,* Vol. I Wiley, New York, 1975.
24 KOBAYASHI, H. Discrete-time queueing systems. In *Probability Theory and Computer Science,* G Louchard and G. Latouche, Eds. Academic Press, New York, 1983, pp. 58–85.
25. KOHAVI, Z. *Switching and Finite Automata Theory* McGraw-Hill, New York, 1970, p. 386
26. KOZLOV, A. The Karmarkar algorithm· Is it for real?. *SIAM Newsl. 18,* 6 (Nov. 1985), 1.
27 LAFORTUNE, S., AND WONG, E. A state transition model for distributed query processing. *ACM Trans. Database Syst. 11,* 3 (Sept. 1986), 294–322.
28. LARSON, H. J. *Introduction to Probability Theory and Statistical Inference.* Wiley, New York, 1969
29. LINDLEY, D. V The theory of queues with a single server. *Proc. Cambridge Phil. Soc. 48* (1952), 277–289.
30. LU, H , AND CARAY, M. J. Load balancing in a locally distributed database system In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Washington, D.C., May 1986), ACM, New York
31. LUENBERGER, D G. *Introduction to Linear and Nonlinear Programming.* Addison-Wesley, Waltham, Mass., 1973, p. 318.
32. MACKERT, L. F., AND LOHMAN, M. R^* optimizer validation and performance evaluation for distributed queries In *Proceedings of the 12th International VLDB Conference* (Kyoto, Aug. 1988), pp. 149–159.
33. MERRETT, T. H. *Relational Information Systems.* Reston, Va., 1983.
34. SCHRIJVER, A. *Theory of Linear and Integer Programming.* Wiley, New York, 1986.
35. SEN, P. Adaptive channels and human decision-making *IEEE Trans. SMC-14,* 5 (1984), 120–131
36 SUMITA, U , AND LIU SHENG, O. R. Analysis of query processing in distributed database systems with fully replicated files: A hierarchical approach. *Perf. Eval. 8,* 3 (June 1988), 223–238.
37. TAY, Y C. *Locking Performance in Centralized Databases* Academic Press, New York, 1986.
38 WAGNER, H. M. On the optimality of pure strategies. *Manage. Sci. 6,* 3 (Apr 1960).
39. WAREN, A. D., HUNG, M S., AND LASDON, L. S. The status of nonlinear programming software: An update. *Oper. Res. 35,* 4 (July 1987), 489–503.
40. WONG, E. Dynamic rematerialization: Processing distributed queries using redundant data. *IEEE Trans. SE-9,* 3 (May 1983), 228–232.
41 YU, C. T , AND CHANG, C. C. Distributed query processing. *ACM Comput Surv. 16,* 4 (Dec 1984), 399–533.

Received September 1988; revised May 1990; accepted March 1992