Incremental data allocation and reallocation in distributed database systems
Chin, A G
Journal of Database Management; January 2001; 12, 1; Library & Information Science Abstracts (LISA)
pg. 35

Vol. 12, No. 1

Incremental Data Allocation and ReAllocation in Distributed Database Systems

AMITA GOYAL CHIN, Virginia Commonwealth University

In a distributed database system, an increase in workload typically necessitates the installation of additional database servers followed by the implementation of expensive data reorganization strategies. We present the Partial REALLOCATE and Full REALLOCATE heuristics for efficient data reallocation. Complexity is controlled and cost minimized by allowing only incremental introduction of servers into the distributed database system. Using first simple examples and then, a simulator, our framework for incremental growth and data reallocation in distributed database systems is shown to produce near optimal solutions when compared with exhaustive methods.

INTRODUCTION

Recent years have witnessed an increasing trend of the implementation of Distributed Database Management System (DDBMS) for more effective access to information. An important quality of these systems, consisting of *n* servers loosely connected via a communication network, is to adjust to changes in workloads. To service increases in demand, for example, additional servers may be added to the existing distributed system and new data allocations computed. Conventionally, this requires a system shutdown and an exhaustive data reallocation. Such static methods are not practical for most organizations for these methods result in high costs and in periods of data unavailability.

We present the incremental growth framework to address incremental expansion of distributed database systems. Data is reallocated using one of two data reallocation heuristics - Partial REALLOCATE or Full REALLOCATE. Both heuristics are greedy, hill-climbing algorithms that compute new data allocation based on the specified optimization parameter of the objective cost function. Due to their linear complexity, both heuristics can be used to solve both small and large, complex problems, based on organizational needs. The robustness of the heuristics is demonstrated first by simple, illustrative examples and then by parametric studies performed using the SimDDBMS simulator.

The REALLOCATE algorithms in conjunction with SimDDBMS can be used to answer many practical questions in distributed database systems. For example, in order to improve system response time, a database administrator (DBA) may use SimDDBMS for parametric evaluation. For example, the DBA may analyze the effect of upgrading CPU processing capability, increasing network transfer speed, or adding additional servers into the distributed database system. Furthermore, SimDDBMS may easily be modified to evaluate heterogeneous servers, with different CPU processing capabilities. A DBA may also use SimDDBMS to determine the impact and cost-benefit analysis of adding some number, $s \geq 1$, additional servers at one time.

RELATED WORK

Following the pioneering work in (Porcar, 1982) many researchers have studied the data allocation problem (Daudpota, 1998; So, Ahmad, and Karlapalem, 1998; Tamhankar and Ram, 1998; Ladjel, Karlapalem, and Li, 1998). The single data allocation problem has been shown to be intractable (Eswaran, 1974), which means that as the problem size increases, problem search space increases exponentially (Garey and Johnson, 1979). Due to the complex nature of the problem, some researchers (Cornell and Yu, 1990; Rivera-Vega, Varadarajan, and Navathe, 1990; Lee and Liu Sheng, 1992; Ghosh and Murthy, 1991; Ghosh, Murthy and Moffett, 1992) have resorted to integer programming methods in search for good solutions. Since optimal search methods can only be used for small problems, heuristic methods are often used for solving large data allocation problems (Apers, 1988; Blankinship, 1991; Ceri, Navathe, and Wiederhold, 1983; Du and Maryanski, 1988).

Researchers have studied both the static data allocation problem, in which data allocations do not change over time, and the dynamic data allocation problem (Theel and Pagnia,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com

1996; Wolfson, Jajodia, and Huang, 1997; Brunstrom, Leutenegger, and Simha, 1995), which may be adaptive or non-adaptive. Adaptive models (Levin, 1982; Son, 1988; Levin and Morgan, 1978) are implemented when the system senses a substantial deviation in access activities; these models determine a one-time reallocation (for a relatively short period of time) in response to surges in demand. For example, the volume of reservations for a particular airline route may increase during a specific season. Therefore, an airline reservation system may temporarily store additional copies of the files associated with the route at a local server. However, this is a short-term situation which is resolved by introducing replicated file copies. Non-adaptive models (Levin, 1982; Porcar, 1982, Segall, 1976) are employed at the initial system design stage or upon system reorganization; these models do not adjust to variations in system activities.

Most previous research on data allocation assumes a fixed number of servers in the distributed database system (Carey and Lu, 1986; Chu, 1969, Laning and Leonard, 1983; Lee and Liu Sheng, 1992; Rivera-Vega, Varadarajan, and Navathe, 1990). Experiments and simulations are designed to test DDBMS factors such as the degree of data replication, workloads per server, and different levels and classes of queries and transactions (Carey and Lu, 1986; Ciciani, Dias, and Yu, 1990). Simulation runs vary the number of servers to arbitrary values. However, these values are fixed per run and vary only between runs. Incremental system growth and subsequent data reallocation has not previously been addressed.

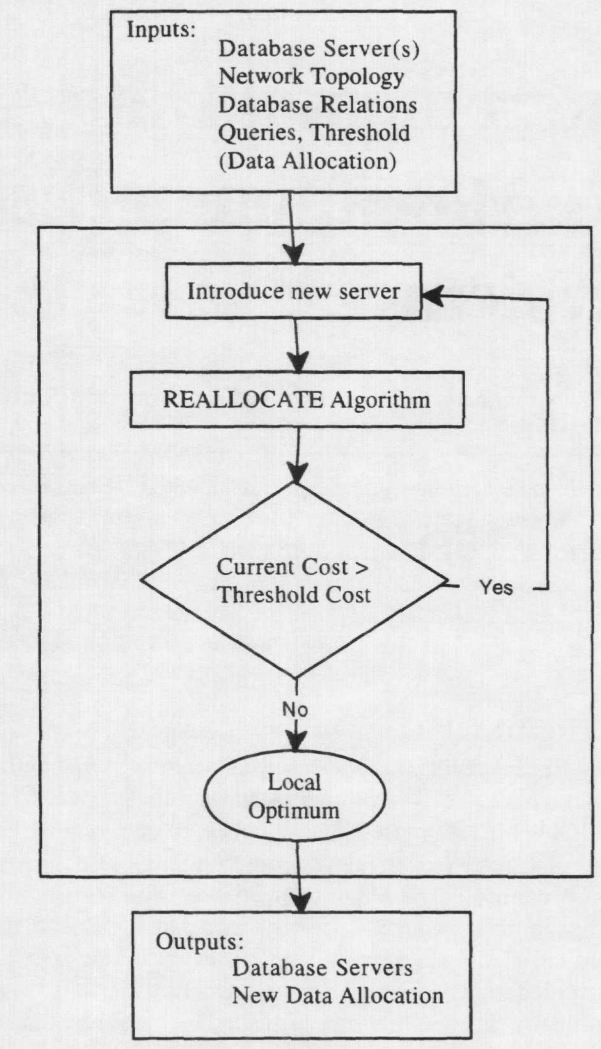
INCREMENTAL GROWTH FRAMEWORK

The incremental growth framework (see Figure 1) is invoked when system performance, as computed using the objective cost function, is below the acceptable threshold (specified by the DBA). To return to an acceptable state, new servers are introduced incrementally, one at a time, into the distributed database system. With the introduction of each new server, a new data reallocation for the system is computed. This process is iteratively executed until acceptable performance is achieved or the number of servers equals the number of relations in the distributed database system (the latter constraint can easily be relaxed in a distributed database system housing partitioned data).

The incremental growth framework, which can easily be adapted for one-server or multiple-server systems, can be used by small, mid-size, and large organizations, each having distributed database systems of varying size. In one-server systems, the initial data allocation locates all relations at the server. In multiple-server systems, the current data allocation is required as input into the framework.

Additional input information required for the incremental growth framework includes: the database server or servers, including the local processing capacity; the network topology, including transmission capacity; the database relations, including relation sizes and selectivities; the query set, the optimization parameter, and the acceptable threshold for the optimization parameter.

Figure 1



DEFINITIONS

The relational data model is used to describe the data and query processing on the data. Only simple queries are considered. Queries are assumed to be independent and are solved independently. Queries are processed in parallel in the distributed database system. To simplify the estimation of query result sizes, the concept of selectivity (Blankinship, 1991; Chin, 1999; Date, 1991; Goyal, 1994) is utilized. Attribute values are assumed to be uniformly distributed and each attribute in a relation is assumed to be independent of all other attributes in the database. The simple query environment has been chosen because it has a manageable complexity while remaining realistic and interesting.

The parameters describing the simple query environment are (Blankinship, 1991; Chin, 1999; Goyal, 1994;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com

Hevner and Yao, 1979):

S_i: Network Servers, $i = 1, 2, ..., s, s+1$
(S_{s+1} = the new server joining the system)

R_j: Relations, $j = 1, 2, ..., r$

For each relation R_j, $j = 1, 2, ..., r$:

n_j: number of tuples,

a_j: number of attributes,

$ß_j$: size (in bytes)

For each attribute d_{jk}, $k = 1, 2, ..., a_j$ of relation R_j:

p_{jk}: attribute density, the number of different values in the current state of the attribute divided by the number of possible attribute values. So, $0 <= p_{jk} <= 1$ (Hevner and Yao, 1979). During join operations the density is used as a selectivity coefficient.

w_{jk}: size (in bytes) of the data item in attribute d_{jk}

For local transaction processing, each server in the distributed database system maintains a queue of incoming requests. Queries are maintained in queue until they are processed using a First In, First Out (FIFO) order.

Finally, the distributed database system maintains a centralized data dictionary housing the following information (Blankinship, 1991; Goyal, 1994; Hevner and Yao, 1979):

- for each relation R_j, $j = 1, 2, ..., r$: n_j, a_j, $ß_j$, and S_j (server to which relation R_j is allocated)
- for each attribute d_{jk}, $k = 1, 2, ..., a_j$ of relation R_j: p_{jk}, w_{jk}, and b_{jk} (projected size, in bytes, of attribute d_{jk} with no duplicate values)

Optimizing query strategies is not within the scope of this research. However, since the optimal data allocation is dependent on the implemented query strategy, when computing new data allocations, Algorithm Serial (Hevner and Yao, 1979) for query processing is implemented. Any query optimization algorithm from the research literature, however, can be used in place of Algorithm Serial.

Algorithm Serial (Hevner and Yao, 1979) considers serial strategies to minimize total transmission time in the simple query environment. For each query q accessing $\hat{A}$ relations, there are $\hat{A}!$ possible combinations for processing q. The serial strategy consists of transmitting each relation, starting with R_1, to the next relation in a serial order. The strategy is represented by $R_1 \rightarrow R_2 \rightarrow ... \rightarrow R_\sigma$, where σ is the number of relations in the query (Hevner and Yao, 1979). Consider for example a query which accesses relations A, B, and C. Then, the $\hat{A}! = 6$ processing combinations for the query are: $A \rightarrow B \rightarrow C$, $A \rightarrow C \rightarrow B$, $B \rightarrow A \rightarrow C$, $B \rightarrow C \rightarrow A$, $C \rightarrow A \rightarrow B$, $C \rightarrow B \rightarrow A$. Therefore, given 4 queries, two of which access 2 relations, one of which accesses 3 relations, and one of which access 4 relations, the number of possible serial strategy combinations is $(2!)(2!)(3!)(4!) = (2)(2)(6)(24) = 576$. The serial order is computed so that $ß_1 \leq ß_2 \leq ... \leq ß_\sigma$, where $ß_j$ is the size of relation R_j, $j = 1, ..., r$ (Hevner and Yao, 1978).

SYSTEM COST EQUATIONS

A fully connected, reliable communication network (Liu Sheng, 1992) with all servers having equal local storage and processing capacity is assumed. A single instance of each relation is allocated to the distributed database system (Blankinship, 1991; Cornell, 1989; Goyal, 1994). (Data partitioning and data replication are not considered in this research.)

When measuring system response time, the objective cost function consists of three cost components: transmission costs, local processing costs, and queuing costs. Costs are measured in terms of the number of CPU cycles or time ticks (Goyal, 1994) needed to complete a task. The system response time is equal to the number of CPU cycles needed to process $Q = \{Q_1, ..., Q_q\}$ queries.

Transmission cost equations are identical between any two servers (Blankinship, 1991; Goyal, 1994), and costs are based on the amount of data transmitted. Local processing cost equations are also identical at any two servers, and costs are based on the amount of data processed. Queuing costs are based on the number of CPU cycles a query spends in queue at each server. Storage costs are considered negligible and are not considered in the cost analysis.

Using the additional notation:

CS_n: cumulative selectivity for Q_n	$n = 1, ..., q$
QR_n: query result for Q_n	$n = 1, ..., q$
QWT_{ni}: wait time in queue for Q_n at S_i	$n = 1, ..., q; i = 1, ..., s+1$
LPT_{ni}: local processing time for processing Q_n at S_i	$n = 1, ..., q; i = 1, ..., s+1$
$NTT_{ni\hat{i}}$: network transfer time for transferring Q_n from S_i to $S_{\hat{i}}$	$n = 1, ..., q; i, \hat{i} = 1, ..., S+1; i \neq \hat{i}$
Θ_{ni}: transmission of Q_n to S_i	$n = 1, ..., q; i = 1, ..., s+1$
ρ: CPU rate per CPU cycle	
μ: network transfer rate per CPU cycle	

we state our cost equations as follows (Goyal, 1994):

$$QR_n = (CS_n)(ß_j) \qquad n = 1, ..., q; j = 1, ..., r$$

$$LPT_{ni} = \frac{\beta_j + QR_n}{\rho} \text{ at } S_i \qquad n = 1, ..., q; j = 1, ..., r; i = 1, ..., s+1$$

$$NTT_{ni\hat{i}} = \frac{QR_n}{\mu} \text{ from } S_i \text{ to } S_{\hat{i}} \qquad n = 1, ..., q; i, \hat{i} = 1, ..., s+1; i \neq \hat{i}$$

$$QWT_{ni} = T_{P_{ni}} - T_{Q_{ni}} \qquad n = 1, ..., q; \; i = 1, ..., s+1$$

where,

$T_{Q_{ni}}$ = the CPU cycle S_i places Q_n into its queue

$T_{P_{ni}}$ = the CPU cycle S_i begins processing Q_n

The objective cost function when computing data allocations to minimize response time is to minimize:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com

$$RT = \sum_{t=1}^{T} (t) + \varsigma$$

where,

$$C(t) = \begin{cases} 1 \; \textit{if} (Q_n \in Q \textit{ in queue at } S_i \\ \vee (Q_n \in Q \textit{ in process at } S_i \\ \vee (Q_n \in Q \text{ in transmision } S_i \to S_i) \\ 0 \; \textit{otherwise} \end{cases}$$

and ς = system idle time while processing $Q = \{Q_1, ..., Q_q\}$.

The following example demonstrates cost computations:

Example

Given the relations $R=\{R_A, R_B, R_C\}$, with selectivities, size β, and server allocations on $S = \{S_1, S_2\}$ as follows:

Table 1: Example - Relations and Allocations

Relation	Selectivity	β	Allocation
R_A	0.2	200	S_1
R_B	0.4	400	S_1
R_C	0.6	600	S_2

and the following queries and query strategies (computed so that $\text{ß}_1 \leq \text{ß}_2 \leq ... \leq \text{ß}_o$):

Table 2: Example - Queries and Query Strategies

Query ID	Query	Query Strategy
Q_1	Join A, B, & C	$R_A \to R_B \to R_C$
Q_2	Join B & C	$R_B \to R_C$

Based on the example in Table 3 (see next page), the system response time is 370 CPU cycles. The response time for query Q_1 is 221-10=211 CPU cycles and the response time for Q_2 is 370-15=355 CPU cycles.

DATA REALLOCATION HEURISTICS

We present two heuristics for data reallocation: Partial REALLOCATE and Full REALLOCATE. Both algorithms are greedy, iterative, "hill-climbing" heuristics that will not traverse the same path twice. With each iteration, they will find a lower cost solution, or they will terminate. Both algorithms require as input: the current data allocation, the relations, and the queries in the distributed database system.

We define the notation:

$S = \{S_1, ..., S_s, S_{s+1}\}$: set of servers ($S_{s+1}$ = the new server)
$R = \{R_1, ..., R_r\}$: set of relations allocated to S_i, i = 1, ..., s
$R' = \{R'_1, ..., R'_r\}$: set of relations allocated to S_{s+1}
$R \cap R' = \emptyset$
$R_j \Rightarrow S_i$: permanent allocation of R_j to S_i, j= 1, ..., r; i = 1, ..., s
$R_j \to S_i$: temporary allocation of R_j to S_i, j= 1, ..., r; i = 1, ..., s
$O_o = \delta(\forall R_j \in R, R_j \Rightarrow S_i)$, i = 1, ..., s
$O_{(s+1)j} = \delta(R_j \to S_{s+1})$ j= 1, ..., r

where $\delta(R_j \to S_i)$ and $\delta(R_j \Rightarrow S_i)$ is the objective cost function evaluated for $R_j \to S_i$ and $R_j \Rightarrow S_i$, respectively.

Each relation R_j must be allocated to a server and can be allocated to only one server at any given time. Therefore,

For each $R_j \in (R \cup R')$, $\sum_{i=1}^{S+1} x_{ij} = 1$ where

$$X_{ij} = \begin{cases} 1, R_j \to S_i \vee R_j \Rightarrow S_{ji} \\ 0, \text{otherwise} \end{cases}$$

Partial REALLOCATE

The steps of the Partial REALLOCATE algorithm are:

Step 1: Compute O_o.

Step 2: For each $R_j \in R$, Compute $O_{(s+1)j} = \delta(R_j \to S_{s+1})$, where for $R'' = R - R_j$, $\forall R_k$'s $\in R''$, $R_k \not\Rightarrow S_{s+1}$, $1 \leq k \leq (r-1)$.

Step 3: Compare $O_\Delta = \underset{j}{\text{MIN}}\; O_{(s+1)j}$ to O_o. If $O_\Delta \geq O_o$, O_o is the local optimum. If $O_\Delta < O_o$, update O_o to O_Δ, $R' = R' + R_j$, $R = R - R_j$, $R_j \Rightarrow S_{s+1}$.

Consider for example the relations and server allocations in Table 1. Assume $S_{s+1} = S_3$ is the new server joining the distributed system. Then, in Step 1, Partial REALLOCATE computes the value of the objective function given the current allocation. Assume O_o = 115. In Step 2, Partial REALLOCATE computes the value of the objective function when independently moving each relation to S_{s+1}:

* Move only R_A to S_3: Compute $O_{(s+1)A} = \delta(R_A \to S_3)$; $R''= \{R_B, R_C\}$
* Move only R_B to S_3: Compute $O_{(s+1)B} = \delta(R_B \to S_3)$; $R''= \{R_A, R_C\}$
* Move only R_C to S_3: Compute $O_{(s+1)C} = \delta(R_C \to S_3)$; $R''= \{R_A, R_B\}$

Assume $O_{(s+1)A}$ =100, $O_{(s+1)B}$ =75, and $O_{(s+1)C}$ = 125. Then, in Step 3, Partial REALLOCATE selects the move resulting in the minimum cost, $O_\Delta = O_{(s+1)B} = 75$. Since $O_\Delta < O_o$, a lower cost solution has been found; R_B is relocated from S_1 to S_3.

Partial REALLOCATE minimizes the number of $\delta(R_j \to S_{s+1})$ evaluations while searching for a better solution. The number of Partial REALLOCATE tests per additional server is bounded by R, the number of relations in the distributed database system. Partial REALLOCATE is not guaranteed to find the optimum solution (as determined using Exhaustive Search). However, if evaluating $\delta(R_j \to S_{s+1})$ is expensive and/or if Partial REALLOCATE's percentage deviation from the optimal solution is acceptable, Partial REALLOCATE is more cost-effective than either Full REALLOCATE or Exhaustive Search.

Full REALLOCATE

The steps of the Full REALLOCATE algorithm are:

Step 1 & Step 2: Same as in Partial REALLOCATE.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com

Table 3: Example - Cost Computations

Assuming ρ=10 bytes per unit time and μ=20 bytes per unit time,

CPU cycle	Processing and Computations (unit = CPU cycles)
10	Θ_{11}; S_1 places Q_1 in its queue; $QR_1=0$; $CS_1=1.0$
15	Θ_{21}; S_1 places Q_2 in its queue; $QR_2=0$; $CS_2=1.0$
30	S_1 begins processing Q_1; $QWT_{11}=30-10=20$; $LPT_{11}=\frac{200+0}{10}=20$; $QR_1=(1.0)(200)=200$; $CS_1=(1.0)(0.2)=0.2$
50	Since R_B is also at S_1, $NTT_{111}=0$; S_1 begins processing Q_1 with R_B; $QWT_{11}=20$; $LPT_{11}=20+\frac{400+200}{10}=80$; $QR_1=(0.2)(400)=80$; $CS_1=(0.2)(0.4)=0.08$
110	Θ_{12}; $NTT_{112}=\frac{80}{20}=4$; S_1 begins processing Q_2;$QWT_{21}=95$; $LPT_{21}=\frac{400+0}{10}=40$; $QR_2=(1.0)(400)=400$; $CS_2=(1.0)(0.4)=0.4$
114	S_2 places Q_1 in its queue
150	Θ_{22}; $NTT_{212}=\frac{400}{20}=20$; S_2 begins processing Q_1;$QWT_{12}=36$; $LPT_{12}=$; $QR_1=(0.8)(600)=48$; $CS_1=(0.08)(0.6)=0.048$
170	S_2 places Q_2 in its queue
190	S_2 begins processing Q_2; $QWT_{22}=20$; $LPT_{22}=\frac{600+400}{10}=100$; $QR_2=(0.4)(400)=160$; $CS_2=(0.4)(0.6)=0.24$
218	$\Theta_{1(client)}$; $NTT_{12(client)}=\frac{48}{20}=2.4$
221	Client receives result of Q_1
290	$\Theta_{2(client)}$; $NTT_{22(client)}=\frac{160}{20}=80$
370	client receives result of Q_2

Step 3: Holding the R_j yielding $O_{1\Delta} = \underset{j}{\mathrm{MIN}}\ O_{(s+1)j}$ at S_{s+1}, $R_j \Rightarrow S_{s+1}$, $R' = R' + R_j$, $R = R - R_j$, Full REALLOCATE reiterates with a new $R_j \in R$ until either $O_{x\Delta} \geq O_{(x-1)\Delta}$ or $R = \emptyset$.

Step 4: Compare $O_{y\Delta} = \underset{j}{\mathrm{MIN}}\ O_{(s+1)j}$ from the yth iteration, y = x yielding $\mathrm{MIN}(O_{x\Delta}, O_{(x-1)\Delta})$, to O_o. If $O_{y\Delta} \geq O_o$, O_o is the local optimum. If $O_{y\Delta} < O_o$, update O_o to $O_{y\Delta}$, $R' = R' + R_j$, $R = R - R_j$, $R_j \Rightarrow S_{s+1}$.

As in Partial REALLOCATE, Full REALLOCATE begins by computing $\delta(R_j \rightarrow S_{s+1})$ for each $R_j \in R$. Rather than outputting the MIN $\delta(R_j \rightarrow S_{s+1})$, Full REALLOCATE holds the R_j yielding $O_{1\Delta}$ at S_{s+1}, so $R_j \Rightarrow S_{s+1}$, $R' = R' + R_j$, $R = R - R_j$. Full REALLOCATE then reiterates with a new $R_j \in R$ until either $O_{x\Delta} \geq O_{(x-1)\Delta}$ or $R = \emptyset$. The number of Full REALLOCATE tests per additional server is bounded by $\sum_{r=1}^{R} r$.

Full REALLOCATE iterates a greater number of times than Partial REALLOCATE before choosing from the yth iteration of the algorithm.

EXAMPLE - COMPARISON OF PARTIAL, FULL REALLOCATE, EXHAUSTIVE SEARCH

We provide an illustrative example to demonstrate the benefits of implementing the REALLOCATE algorithms over Exhaustive Search. We solve the same problem using each of the three data reallocation algorithm — first using Exhaustive Search, then using Partial REALLOCATE, and finally using Full REALLOCATE. Response times have been computed by the SimDDBMS simulator. The serial query strategy (Hevner and Yao, 1979) has been implemented and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com

Table 4: System Parameters

Parameter	Value
CPU Rate	50
Network Transfer Rate	100
Threshold	200
Optimization Parameter	Response Time

Table 5: Relations

Relation	Selectivity	Size
R_A	1.0	1000
R_B	0.9	900
R_C	0.8	800
R_D	0.7	700

Table 6: Queries

Queries
Join A, B & C
Join B & D
Join A & C
Join A, B, C & D

query results are computed as described in (Blankinship, 1991; Goyal, 1994, Hevner and Yao, 1979). Assume the system parameters in Table 4, the relations in Table 5, and the queries in Table 6.

Base Case

The base case assumes there is only one server, $S = \{S_1\}$, in the system and $\forall R_j$'s $\in R$, $R_j \Rightarrow S_1$. Therefore, the initial allocation is $R_A \Rightarrow S_1$, $R_B \Rightarrow S_1$, $R_C \Rightarrow S_1$, and $R_D \Rightarrow S_1$. Using a poisson arrival rate for queries, SimDDBMS has computed a total response time of 307 CPU cycles. Since 307 is greater than the specified threshold of 200, a new server is added to the distributed database system.

Exhaustive Search

$S = \{S_1, S_2\}$: (incremental growth from s=1 to s+1=2 servers)

There are 16 possible allocations of 4 relations to 2 servers (see Table 7). The Exhaustive Search method finds the lowest response time of 216, with the allocations ($R_A \rightarrow S_1$), ($R_B \rightarrow S_2$), ($R_C \rightarrow S_2$), ($R_D \rightarrow S_1$) and ($R_A \rightarrow S_2$), ($R_B \rightarrow S_1$), ($R_C \rightarrow S_1$), ($R_D \rightarrow S_2$). The first allocation found is arbitrarily chosen as the new data allocation. So, $R_B \Rightarrow S_2$ and $R_C \Rightarrow S_2$. Since the minimum total response time found with two servers is greater than the specified threshold (216 > 200), an additional server is added to the distributed database system.

Table 7: Exhaustive Search with 4 Relations, 2 Servers

	R_A	R_B	R_C	R_D	R_T
1)	1	1	1	1	307
2)	1	1	1	2	279
3)	1	1	2	1	273
4)	1	1	2	2	242
5)	1	2	1	1	224
6)	1	2	1	2	267
7)	1	2	2	1	216
8)	1	2	2	2	244
9)	2	1	1	1	244
10)	2	1	1	2	216
11)	2	1	2	1	267
12)	2	1	2	2	224
13)	2	2	1	1	242
14)	2	2	1	2	273
15)	2	2	2	1	279
16)	2	2	2	2	307

$S=\{S_1, S_2, S_3\}$: (incremental growth from s=2 to s+1=3 servers)

There are 81 possible allocations of 4 relations to 3 servers. The Exhaustive Search method finds the minimum response time of 182, with the allocations ($R_A \rightarrow S_1$), ($R_B \rightarrow S_2$), ($R_C \rightarrow S_3$), ($R_D \rightarrow S_1$) and ($R_A \rightarrow S_1$), ($R_B \rightarrow S_3$), ($R_C \rightarrow S_2$), ($R_D \rightarrow S_1$). Since the response time resulting from adding the third server meets the specified threshold ($182 \leq 200$), no additional servers are added. The final data allocation found by Exhaustive Search is $R_A \Rightarrow S_1$, $R_B \Rightarrow S_2$, $R_C \Rightarrow S_3$, $R_D \Rightarrow S_1$ or $R_A \Rightarrow S_1$, $R_B \Rightarrow S_3$, $R_C \Rightarrow S_2$, $R_D \Rightarrow S_1$ with the total response time of 182 ticks.

Partial REALLOCATE

$S = \{S_1, S_2\}$: (incremental growth from s=1 to s+1=2 servers)

There are R = 4 reallocation tests required. The response time is computed for each independent allocation (see Table 8). The Partial REALLOCATE algorithm finds the lowest response time of 224, with the allocation ($R_A \rightarrow S_1$), ($R_B \rightarrow S_2$), and ($R_C \rightarrow S_1$),($R_D \rightarrow S_1$), so $R_B \Rightarrow S_2$. Since 224 > 200, an additional server is added to the distributed database system.

$S = \{S_1, S_2, S_3\}$: (incremental growth from s=2 to s+1=3 servers)

Again, R = 4 reallocation tests are required. The response time is computed for each independent allocation (see Table 9). The Partial REALLOCATE algorithm finds the lowest response time of 182, with the allocation ($R_A \rightarrow S_1$), ($R_B \rightarrow S_2$), and ($R_C \rightarrow S_3$), ($R_D \rightarrow S_1$), so $R_C \Rightarrow S_2$. Since $182 \leq 200$, Partial REALLOCATE terminates. The final data allocation found by Partial REALLOCATE is $R_A \Rightarrow S_1$, $R_B \Rightarrow S_2$, $R_C \Rightarrow S_3$, $R_D \Rightarrow S_1$ with the total response time of 182 CPU cycles.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com

Table 8: Partial REALLOCATE: 2 Servers, 4 Relations

$(R_i \rightarrow S_2)$	Allocation	RT
$(R_A \rightarrow S_2)$	2111	244
$(R_B \rightarrow S_2)$	1211	224
$(R_C \rightarrow S_2)$	1121	273
$(R_D \rightarrow S_2)$	1112	279

Table 9: Partial REALLOCATE: 3 Servers, 4 Relations

$(R_i \rightarrow S_3)$	Allocation	RT
$(R_A \rightarrow S_3)$	3211	188
$(R_B \rightarrow S_3)$	1311	224
$(R_C \rightarrow S_3)$	1231	182
$(R_D \rightarrow S_3)$	1213	206

Full REALLOCATE

$S = \{S_1, S_2\}$: (incremental growth from s=1 to s+1=2 servers)

There are a maximum of $\sum_{r=1}^{R} r$ =10 reallocation tests.

In the first iteration, the response time is computed for R = 4 independent allocations (see Table 10). The Full REALLOCATE algorithm finds the lowest response time of 224, with the allocation $(R_A \rightarrow S_1), (R_B \rightarrow S_2)$, and $(R_C \rightarrow S_1), (R_D \rightarrow S_1)$, so $R_B \Rightarrow S_2$. Full REALLOCATE reiterates with R_A, R_C, and R_D (see Table 11). The Full REALLOCATE algorithm finds the lowest response time of 216 with the allocation $(R_A \rightarrow S_1)$, $(R_B \rightarrow S_2), (R_C \rightarrow S_2), (R_D \rightarrow S_1)$, so $R_C \Rightarrow S_2$. Full REALLOCATE reiterates with R_A and R_D (see Table 12).

Since the minimum response time found in this iteration is greater than the response time in the previous iteration (244 > 216), Full REALLOCATE does not reiterate with 2 servers. Therefore, the allocation is $(R_A \rightarrow S_1), (R_B \rightarrow S_2), (R_C \rightarrow S_2), (R_D \rightarrow S_1)$ with a response time of 216 CPU cycles. Since 216 > 200, an additional server is added to the distributed database system.

$S = \{S_1, S_2, S_3\}$: (incremental growth from s=2 to s+1=3 servers)

Even with $S = \{S_1, S_2, S_3\}$, there are still only 10 maximum possible allocations with the Full REALLOCATE algorithm. We start with the best allocation (1221) from the result of Full REALLOCATE at 2 servers.

Again, we independently reallocate each R_j to S_{s+1} and evaluate the resulting response time (see Table 13). The Full REALLOCATE algorithm finds the minimum response time of 182 CPU cycles with the allocations $(R_A \rightarrow S_1), (R_B \rightarrow S_3), (R_C \rightarrow S_2), (R_D \rightarrow S_1)$ and $(R_A \rightarrow S_1), (R_B \rightarrow S_2), (R_C \rightarrow S_3), (R_D \rightarrow S_1)$. Arbitrarily choosing the first allocation, $R_B \Rightarrow S_3$, we now test reallocating an additional R_j to S_3 (see Table 14).

Since 216 > 182 and 182 < 200, Full REALLOCATE terminates. The final data allocation found by Full REALLOCATE is $R_A \Rightarrow S_1, R_B \Rightarrow S_3, R_C \Rightarrow S_2, R_D \Rightarrow S_1$ or $R_A \Rightarrow S_1$, $R_B \Rightarrow S_2, R_C \Rightarrow S_3, R_D \Rightarrow S_1$ with the total response time of 182 CPU cycles.

Table 10: First Iteration

$(R_i \rightarrow S_2)$	Allocation	RT
$(R_A \rightarrow S_2)$	2111	244
$(R_B \rightarrow S_2)$	1211	224
$(R_C \rightarrow S_2)$	1121	273
$(R_D \rightarrow S_2)$	1112	279

Table 11: Second Iteration

$(R_i \rightarrow S_2)$	Allocation	RT
$(R_A \rightarrow S_2)$	2211	242
$(R_C \rightarrow S_2)$	1221	216
$(R_D \rightarrow S_2)$	1212	267

Table 12: Third Iteration

$(R_i \rightarrow S_2)$	Allocation	RT
$(R_A \rightarrow S_2)$	2221	279
$(R_D \rightarrow S_2)$	1222	244

Summary

In this example, we have demonstrated that for at least some problems, the Full and Partial REALLOCATE algorithms find the same optimal data allocation solution that is found using the Exhaustive Search algorithm. Additionally, we have shown that the Full and Partial REALLOCATE algorithms greatly reduce the problem search space, and hence, the cost of determining the new data allocation. In this particular example, the search space when incrementing from $S = \{S_1\}$ to $S = \{S_1, S_2\}$ was reduced by 75% using Partial REALLOCATE and 37.5% using Full REALLOCATE. Incrementing from $S = \{S_1, S_2\}$ to $S = \{S_1, S_2, S_3\}$, the search space was reduced by 95% using Partial REALLOCATE and 87.6% using Full REALLOCATE. While Exhaustive Search and Partial REALLOCATE test 100% of their search space,

Table 13: Full REALLOCATE: First Iteration

$(R_i \rightarrow S_3)$	Allocation	RT
$(R_A \rightarrow S_3)$	3221	216
$(R_B \rightarrow S_3)$	1321	182
$(R_C \rightarrow S_3)$	1231	182
$(R_D \rightarrow S_3)$	1223	216

Table 14: Full REALLOCATE: Second Iteration

$(R_i \rightarrow S_3)$	Allocation	RT
$(R_A \rightarrow S_3)$	3321	242
$(R_C \rightarrow S_3)$	1331	216
$(R_D \rightarrow S_3)$	1323	246

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com

Full REALLOCATE tested only 90% of its search space when incrementing from $S = \{S_1\}$ to $S = \{S_1, S_2\}$ and only 70% when incrementing from $S = \{S_1, S_2\}$ to $S = \{S_1, S_2, S_3\}$.

SIMULATION RESULTS

To simulate incremental growth and reallocation, including the implementation of Partial and Full REALLOCATE, we have developed SimDDBMS. We have run over 5,800 simulation experiments. 1,100 smaller, tractable problems have been run using the Exhaustive Search optimal and 4,700 larger, more realistic problems have been run using only Full and Partial REALLOCATE. The experiments have been run to minimize system response time in the simple query environment.

A "base case" simulation is run for each problem. The "base case" assumes there is only one server in the distributed database system, with all of the relations allocated to this one server. (This parameter can be relaxed for distributed database systems consisting of more than one server.)

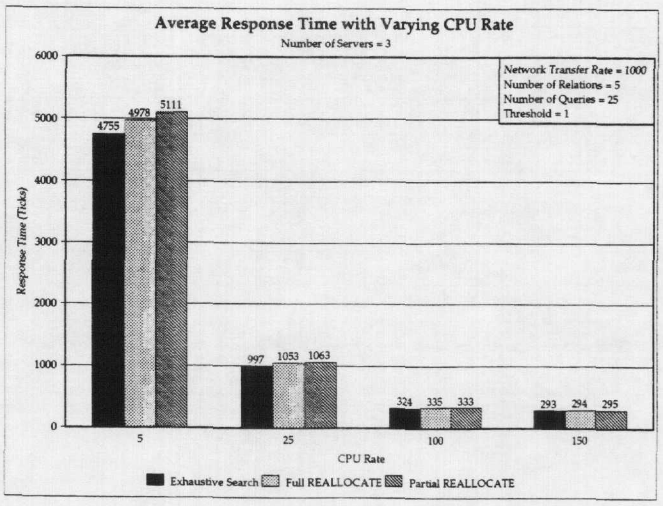
The Exhaustive Search algorithm is used for benchmark comparisons. However, due to the exponential growth in the search space of the Exhaustive Search algorithm, Exhaustive Search computations are restricted to a maximum of 4 servers and 5 relations (a combined search space of 1+32+243+1,024 = 1,300 possible allocations). The effect on the system response time of each of the following parameters (as independent variables) is studied: CPU processing rate, network transfer rate, number of queries, number of relations. In all graphs, each data point represents the average cost for 100 randomly generated problems. Fixed parameter values are displayed in a box at the top of each graph.

Effect of CPU Rate

The REALLOCATE algorithms performed very close to optimal across a broad range of CPU rates (see Figure 2). We observed that at the lowest CPU rate, Full REALLOCATE was on average only 4.92% from the Exhaustive Search optimal with a worst case of 28.4% from optimal. Partial REALLOCATE on average deviated 6.66% from the Exhaustive Search optimal with a worst case of 32.53%. As the CPU rate increased, both REALLOCATE algorithms achieved results even closer to optimal. Additionally, Full REALLOCATE on average reduced execution time by at least 94.5% when compared to Exhaustive Search and Partial REALLOCATE on average reduced execution time by at least 96.5% compared to Exhaustive Search.

When comparing the results of Full and Partial REALLOCATE with a larger number of relations, we found that Partial REALLOCATE actually performed better than Full REALLOCATE at the lower CPU rates and matched Full REALLOCATE at the higher CPU rates (see Figure 3). Partial REALLOCATE required significantly more servers to achieve these results, often close to double the number the number of servers that Full REALLOCATE specified.

Figure 2



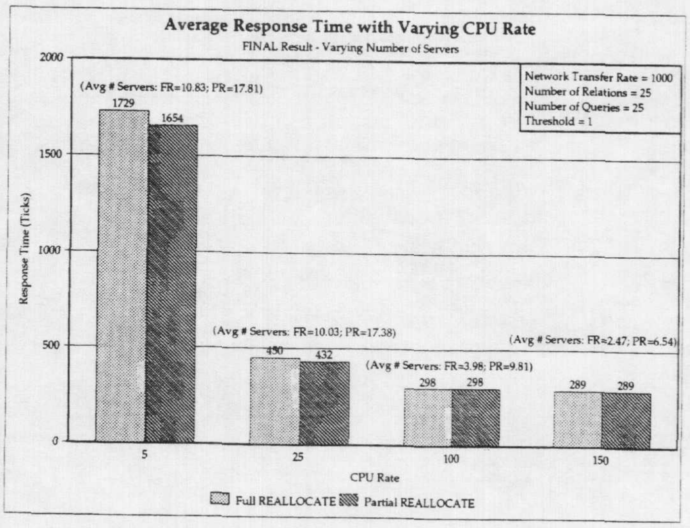
Effect of Network Transfer Rate

As with the effect of varying CPU rates, both Partial and Full REALLOCATE performed well against Exhaustive Search when varying network transfer rate (see Figure 4). At worst, Full REALLOCATE performed within 3% of optimal and Partial REALLOCATE within 2.7%. As network transfer rate increased beyond 100, the effect of network transfer rate became minimal.

Effect of Number of Queries

When increasing the number of queries (see Figure 5), Full REALLOCATE achieved solutions on average of only 3.3% worse than the Exhaustive Search optimal, while Partial REALLOCATE achieved solutions averaging 4% worse than optimal. With a larger number of servers and relations, Partial and Full REALLOCATE obtained identical final cost results. Partial REALLOCATE, however, required a larger number of servers than Full REALLOCATE.

Figure 3



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com

Figure 4

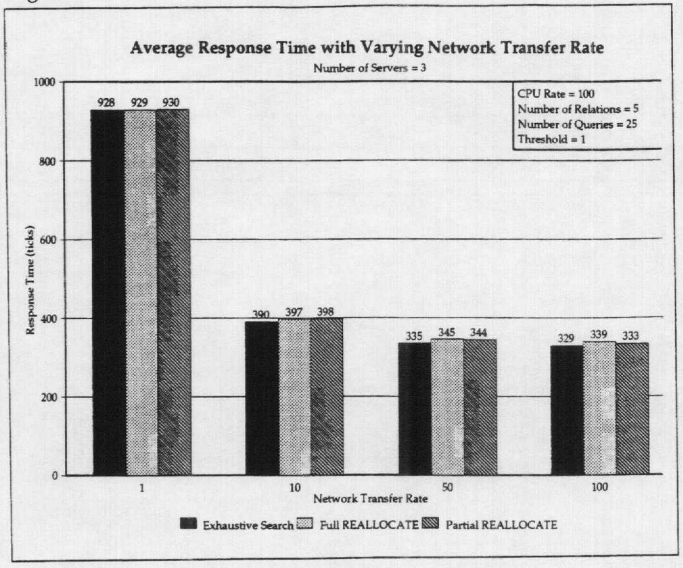


Figure 6

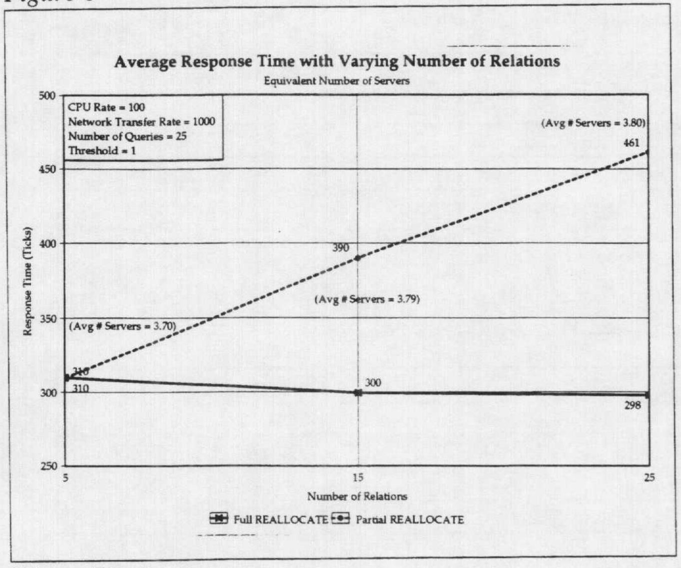


Figure 5

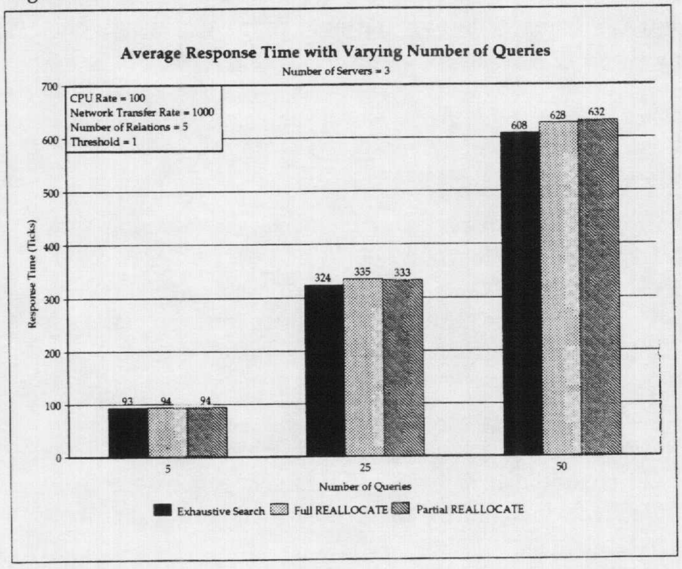


Figure 7

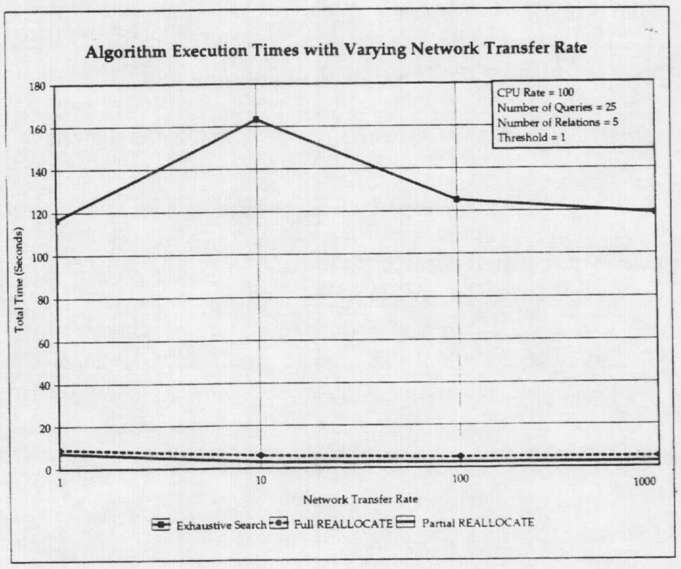
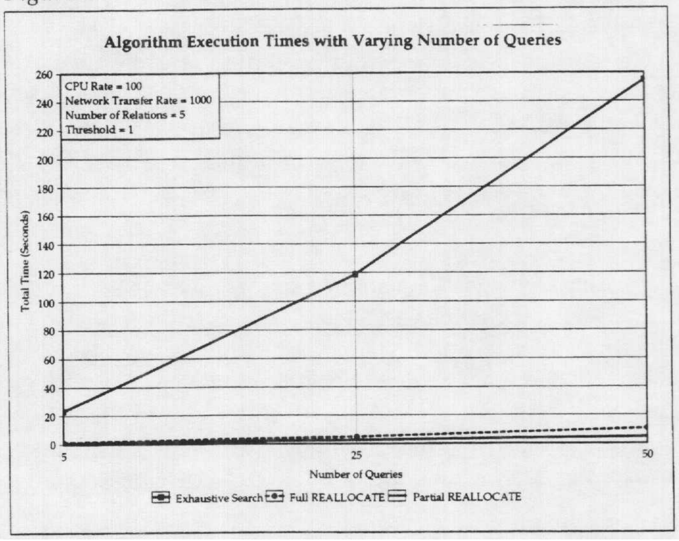


Figure 8



Effect of Number of Relations

Varying the number of relations did not effect the comparative results of Partial REALLOCATE and Full REALLOCATE. Once again, Partial REALLOCATE required a greater number of servers than Full REALLOCATE in order to achieve the same cost results. Examining the response time with an equivalent number of servers (Figure 6) showed that Partial REALLOCATE's response time grew linearly as the number of relations increased. Thus, the efficiency of Partial REALLOCATE dropped per server as compared to Full REALLOCATE.

Figures 7 and 8 compare the execution times of the three data reallocation algorithms. The network transfer rate is varied in Figure 7 and the number of queries is varied in Figure 8 (varying the other system parameters showed similar

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com

Table 15: Comparison of Average Number of Algorithm Tests with Varying Network Transfer Rate

	Network Transfer Rate			
	1	10	100	1000
Partial REALLOCATE	7.02	5.90	5.72	5.90
Full REALLOCATE	11.37	15.15	15.20	15.25
Exhaustive Search	365.34	1,125.92	1,136.16	1,146.40

Table 16: Comparison of Average Number of Algorithm Tests with Varying Number of Queries

	Number of Queries		
	5	25	50
Partial REALLOCATE	6.29	5.90	5.60
Full REALLOCATE	13.85	15.25	15.50
Exhaustive Search	862.63	1,146.40	1,177.12

results). Figure 7 shows that Full REALLOCATE in the best case, only required 3.58% of the time required by Exhaustive Search, and in the worst case, only required 7.73% of Exhaustive Search. Partial REALLOCATE, in the best case, only required 1.74% of the time required by Exhaustive Search, and in the worst case, only required 6.16% Exhaustive Search's time. Figure 8 shows similar results with varying number of queries. In the best case, Full REALLOCATE only required 4.03% of Exhaustive Search's time and only 5.75% in the worst case. Partial REALLOCATE varied from a low of 1.78% to a high of 2.81% of Exhaustive Search's time.

Tables 15 and 16, which correspond to the parameters in Figures 7 and 8, compare the average number of tests required for each of the three reallocate algorithms. Each line $\sum_{r=1}^{R} r$ in the table represents the average of 100 randomly generated problems.

Summary

In summary, the Partial and Full REALLOCATE algorithms have been shown to considerably reduce problem search space, and hence, the cost of testing relation-server combinations. If the cost of each test is one unit, implementing Partial REALLOCATE over exhaustive search results in a cost savings of S^R - R units; implementing Full REALLOCATE over exhaustive search results in a cost savings of S^R - units.

Using SimDDBMS, parametric studies across a range of parameters, including CPU Rate, Network Transfer Rate, Number $\sum^{R} r$ of Queries, and Number of Relations have been performed. The parametric studies have demonstrated the consistency of the REALLOCATE algorithms across a broad range of parametric values. Additionally, the simulation experiments have shown that Partial REALLOCATE and Full REALLOCATE provide good solutions as compared to exhaustive search optimums.

Table 17: Algorithm of Choice

Cost	Level	Algorithm of Choice
Relation-server testing	High	Partial
	Low	Partial, Full
Additional Server	High	Full
	Low	Partial, Full

Partial and Full REALLOCATE have different strengths. As shown in Table 17, if the cost of testing relation-server combinations is high, Partial REALLOCATE is the algorithm of choice. This is because Partial REALLOCATE has a much smaller search space than Full REALLOCATE (R vs , where R is the number of relations in the distributed database system). If the cost of adding additional servers is high, Full REALLOCATE is the algorithm of choice. As demonstrated in the simulation experiments, Partial REALLOCATE is a server-hungry algorithm. It generally requires two to three times as many servers as Full REALLOCATE in order to find a comparable solution.

CONCLUSIONS

We have presented an incremental growth framework and two data reallocation heuristics — Partial REALLOCATE and Full REALLOCATE — for incremental growth and reallocation in distributed database systems. Through simple examples and then through parametric studies performed using the SimDDBMS simulator, we have demonstrated the robustness of both data reallocation heuristics. Due to their linear complexity, the Partial and Full REALLOCATE algorithms can be used for large, complex problems while achieving good results as compared to the exhaustive search optimal.

REFERENCES

Apers, Peter M. G. (1988). "Data Allocation in Distributed Database Systems," *ACM Transactions on Database Systems*, 13(3), September, 263-304.

Blankinship, Rex (1991). "An Iterative Method for Distributed Database Design," Ph.D. Dissertation, University of Maryland at College Park.

Brunstrom, Anna, Scott T. Leutenegger, and Rahul Simha (1995). "Experimental Evaluation of Dynamic Data Allocation Strategies in a Distributed Database with Changing Workloads," CIKM, 395-402.

Carey, Michael J. and Hongjun Lu (1986). "Load Balancing in a Locally Distributed Database System," *Proceedings of the ACM-SIGMOD International Conference on Management of Data, Washington, D.C.*, 108-119.

Ceri, Stefano, Shamkant Navathe, and Gio Wiederhold (1983)."Distribution Design of Logical Database Schemas," *IEEE Transactions on Computers*, SE-9(4),487-504.

المنارة للاستشارات
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com

Chin, Amita Goyal (1999). "An Optimization Algorithm for Dynamic Data Migration in Distributed Database Systems," *Journal of Computer Information Systems*, 39(4).

Chu, Wesley W. (1969). "Optimal File Allocation in a Multiple Computer System," IEEE Transactions on Computers, C-18(10), 885-890.

Ciciani, Bruno, Daniel M. Dias, and Philip S. Yu (1990). "Analysis of Replication in Distributed Database Systems," *IEEE Transactions on Knowledge and Data Engineering*, 2(2), 247-261.

Cornell, Douglas W. (1989). "On Optimal Site Assignment for Relations in the Distributed Database Environment," *IEEE Transaction on Software Engineering*, 15(8), 1004-1009.

Cornell, Douglas W. and Philip S. Yu (1990). "Relation Assignment in Distributed Transaction Processing Environment," *Proceedings of the Sixth International Conference on Data Engineering*, 50-55.

Date, C. J. (1991). *An Introduction to Database Systems*, Addison Wesley.

Daudpota, Nadeem (1998). "Five Steps to Construct a Model of Data Allocation for Distributed Database System," *Journal of Intelligent Information Systems*, 11(2), 153-168.

Du, Xiaolin and Fred J. Maryanski (1988). "Data Allocation in a Dynamically Reconfigurable Environment," Proceedings of the Fourth International Conference on Data Engineering, 74-81.

Eswaran, K., (1974)."Placement of Records in a File and File Allocation in a Computer Network," *Proceedings IFIPS Conference*, 304-307.

Garey, Michael R. and David S. Johnson (1979).*Compilers and Intractability — A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company.

Ghosh, Deb and Ishwar Murthy (1991). "A Solution Procedure for the File Allocation Problem with File Availability and Response Time," *Computers and Operations Research*, 18(6), 557-568.

Ghosh, Deb, Ishwar Murthy, and Allen Moffett (1992). "File Allocation Problem: Comparison of Models with Worst Case and Average Communication Delays," *Operations Research*, 40(6), 1074-1085.

Goyal, Amita (1994). "Incremental Growth and Reallocation in Distributed Database Systems," Ph.D. Dissertation, The University of Maryland at College Park.

Hevner, Alan R. and S.B. Yao (1978). "Optimization of Data Access in Distributed Systems," Computer Science Department, Purdue University, Technical Report TR281.

Hevner, Alan R. and S.B. Yao (1979). "Query Processing in Distributed Database Systems," *IEEE Transactions on Software Engineering*, SE-5(3).

Ladjel, Bellatreche, Kamalakar Karlapalem, and Qing Li(1998)."An Iterative Approach for Rules and Data Allocation in Distributed Deductive Database Systems," *Proceedings of the 1998 ACM 7th International Conference on Information and Knowledge Management*, 356-363.

Laning, Lawrence J. and Michael S. Leonard, "File Allocation in a Distributed Computer Communication Network," IEEE Transactions on Software Engineering, Vol. C-32, No. 3, March 1983, pp. 232-244.

Lee, Heesok, and Olivia R. Liu Sheng (1992). "A Multiple Criteria Model for the Allocation of Data Files in a Distributed Information System," Computers and Operations Research, 19(1), 21-33.

Levin, K.D. (1982). "Adaptive Structuring of Distributed Databases," *Proceedings of the National Computer Conference*, 691-696.

Levin, K. D., and H. L. Morgan (1978). "A Dynamic Optimization Model for Distributed Databases," Operations Research, 26(5), 824-835.

Liu Sheng, Olivia R. (1992)."Optimization of File Migration Policies in Distributed Computer Systems," *Computers and Operations Research*, 19(5), 335-351.

Porcar, H.(1982). "File Migration in Distributed Computing Systems," Ph.D. Thesis, University of California at Berkeley.

Rivera-Vega, Pedro I., Ravi Varadarajan, and Shamkant B. Navathe (1990)."Scheduling Data Redistribution in Distributed Databases," *Proceedings of the Symposium on Reliability in Distributed Software and Database Systems*, 166-173.

Segall, Adrian (1976)."Dynamic File Assignment in a Computer Network," *IEEE Trans. Automat. Control*, AC-21, April, 161-173.

So, Siu-Kai, Ishfaq Ahmad, and Kamalakar Karlapalem (1998)."Data Allocation Algorithm for Distributed Hypermedia Documents," *The 1998 IEEE 17th Symposium on Reliable Distributed Systems*, , 473-478.

Son, Sang H. (1988). "Replicated Data Management in Distributed Database Systems," *SIGMOD Record*, 17(4),

Tamhankar, Ajit and Sudha Ram (1998). " Database Fragmentation and Allocation: An Integrated Methodology and Case Study," IEEE Transactions on Systems, Man, and Cybernetics, 28(3), 288-305.

Theel, Oliver E and Henning Pagnia (1996). "Bounded Dynamic Data Allocation in Distributed Systems," The 1996 3rd International Conference on High Performance Computing, 126-131.

Wolfson, Ouri and Sushil Jajodia, and Yixiu Huang (1997). "An Adaptive Data Replication Algorithm," *ACM Transactions on Database Systems*, 22(2), 255-314.

Dr. Amita Goyal Chin is an Associate Professor in the Department of Information Systems at Virginia Commonwealth University. She received her B.S. in computer science and M.S. and Ph.D. in information systems, all from The University of Maryland at College Park. Her research interests include distributed database systems, text databases and document storage and handling, and collaborative technologies.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission. www.manaraa.com