Incremental Expansion of a Distributed Database System

Amita Goyal Chin
Virginia Commonwealth University, USA

INTRODUCTION

Recent years have witnessed an increasing trend in the implementation of distributed database management systems (DDBMSs) 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 REALLOCATE algorithms in conjunction with the SimDDBMS simulator 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, $s \leq 1$, additional servers at one time.

BACKGROUND

Following the pioneering work in Porcar (1982), many researchers have studied the file or data allocation problem (Daudpota, 1998; Ladjel, Karlapalem, & Li, 1998; So, Ahmad, & Karlapalem, 1998; Tamhankar & Ram, 1998;). 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, & Wiederhold, 1983; Chin, 2001; (Chin) Goyal, 1994; Du & 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 (Brunstrom, Leutenegger, & Simha, 1995; Theel & Pagnia, 1996; Wolfson, Jajodia, & Huang, 1997), which may be adaptive or nonadaptive. Adaptive models (Babcock, Babu, Motwani, & Datar, 2003; Levin, 1982; Levin & Morgan, 1978; Son, 1988) 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. Nonadaptive models (Levin; Porcar; 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 & Lu, 1986; Chu, 1969; Laning & Leonard, 1983; Lee & Liu Sheng, 1992; Rivera-Vega, Varadarajan, & 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 & Lu; Ciciani, Dias, & 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 GROWTH FRAMEWORK

The incremental growth framework 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, midsize, 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.

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; (Chin) 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; (Chin) Goyal, 1994; Hevner & 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,
β_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 & 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; (Chin) Goyal, 1994; Hevner & Yao, 1979):

- for each relation R_j, j = 1, 2, ..., r: n_j, a_j, β_j, and S_j (server to which relation R_j is allocated)
- for each attribute d_k, k = 1, 2, ..., a_j of relation R_j: p_{jk}, w_{jk}, and b_{jk} (projected size, in bytes, of attribute dj_k 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 & 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 & Yao, 1979) considers serial strategies to minimize total transmission time in the simple query environment. For each query q accessing ψ relations, there are $\psi!$ 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 $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_\sigma$, where σ is the number of relations in the query (Hevner & Yao, 1979).

Consider, for example, a query which accesses relations A, B, and C. Then, the $\psi! = 6$ 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 B $\rightarrow$ A. Therefore, given four queries—two of which access two relations, one of which accesses three relations, and one of which accesses four 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 $\beta_1 \leq \beta_2 \leq \ldots \leq \beta_s$, where β_j is the size of relation R_j, j = 1, ..., r (Hevner & Yao, 1978).