An Algorithm for Dynamic Data Distribution

Priliminary Version

Ouri Wolfson
Department of Electrical Engineering and Computer Science
University of Illinois
Chicago, IL 60680
wolfson@dbis.eecs.uic.edu

Sushil Jajodia
Information and Software Systems Engr. Department
George Mason University
Fairfax, VA 22030-4444

1 Introduction

The replication scheme of a distributed database determines how many replicas of each object are created, and to which processors these replicas are allocated. This scheme critically affects the performance of a distributed system, since reading an object locally is less costly than reading it from a remote processor. Therefore in a read-intensive network a widely distributed replication is mandated. On the other hand, an update of an object is usually written to all, or a majority of the replicas, and therefore in a write-intensive network a narrowly distributed replication is mandated. In other words, the optimal replication scheme depends on the read-write pattern for each object.

Currently, the replication scheme is established in a static fashion, when the distributed-database is designed, and it remai ns fixed until the designer manually intervenes to change the number of replicas or their location. If the read-write patterns are fixed and are known a priori, this is a reasonable solution. However, if the read-write patterns change dynamically, in unpredictable ways, a static replication scheme may lead to severe performance problems.

In this paper we propose a practical algorithm, called Dynamic-Data-Allocation (DDA), that changes the replication scheme of an object (i.e. the processors which store a replica of the object) dynamically as the read-write pattern of the object changes in the network. We assume that the changes in the read-write pattern are not known a priori.

The algorithm DDA is based on the primary copy method, it preserves 1-copy-serializability, and it is distributed in the sense that the decision on whether or not to store a copy of the object is made by each individual processor based on locally collected statistics. The algorithm changes the replication scheme of an object to optimize a global cost function for the current read-write pattern. As the read-write pattern changes, so does the replication scheme. Specifically, if during a period of time it is cost effective for some processor to store a copy of the object (due to the dominant cost of reads initiated locally), then it will do so. When the processor determines that, due to the dominant cost of writes initiated at other processors, it is not cost effective to retain the copy, then it will give it up.

The DDA algorithm is also integrated. In such an algorithm, redistribution of the replicas is integrated into the processing of reads and writes, instead of executing independently of these operations. So, for example, a processor x creates a replica of an object at some neighbor, y, in response to y's request to read the object from x. x creates the replica by piggybacking, on the message to y containing the object, an indication that y should keep a replica (and that x will propagate writes to y). Additionally, we show that the DDA algorithm can cope with failures.

2 The Cost Function

Consider a data object O in a database that is replicated over a set of processors interconnected by a communication network. The set of processors at which O is allocated is called the *O-scheme*. Transactions executing in the distributed system issue logical reads and logical writes of O. Each logical read is translated into a read of a physical replica of O, and each logical write is translated into the write of all physical replicas of O. A logical-read (write) will be called a read (write) for short.

Suppose that the cost for a read operation of O, executed at a processor that has a copy of O is one (the I/O cost). The cost for a read operation executed at a processor that does not have a copy of the item is $1 + d$, where d is a constant representing the communication cost of transmitting the item from one processor to another. The cost of a write is proportional to the number of copies, c. Specifically, for each copy there is a cost of transmission (of the object to the processor, and an I/O cost (of writing the object). Therefore, the cost of a write is[1] $c \cdot (1 + d)$.

Consider a distributed database system in which a processor i performs $\#W_i$ writes of O and $\#R_i$ reads of O, during each time unit. (For example, the time unit may be three minutes.). Then, the total cost of reading and writing O during a time-unit is a function of the O-scheme. It is quite straight-forward to show that this cost function is minimum when the O-scheme consists of the set of processors $\{\ k \mid (1+d) \cdot \sum_j \#W_j \leq d \cdot \#R_k\ \}$. In other words, a copy is allocated to a processor k if and only if: $1+d$ times the total number of writes in the network during a time unit, is not higher than d times the number of reads issued by k during a time unit. Intuitively, the reason for this being the optimal allocation scheme is the following. Allocating the copy to a processor k increases the cost of each write of O (issued by any processor in the network) by $1 + d$, and it decreases the cost of each read issued by k by d (since k saves the communication cost of remotely reading O). Therefore, a copy of O should be placed at each processor which causes the increase in cost to be lower than the decrease in cost.

3 The DDA Algorithm

The optimal replication scheme defined at the end of the last section optimizes the cost of accessing an object O under the assumption that the read-write pattern, i.e., $\#W_j$ and $\#R_j$ for each j, are fixed; e.g., for each time unit, a processor j performs $\#W_j$ writes and $\#R_j$ reads. In practice, the read-write pattern may change over time. The DDA algorithm presented in this section, adapts the O-scheme to optimize the cost function for the current time-unit.

The DDA algorithm consists of two parts. First, the execution of reads and writes of O, and second, changing of the O-scheme. We first describe the read and write algorithm; it is based on the primary-copy method. At any point in time there is a processor, called the $primary - copy$ processor and denoted $P(O)$, that stores the primary copy of O. $P(O)$ services the reads of processors that do not have a copy of O, as well as the writes of O. Processor $P(O)$ knows which other processors have copies of O. A processor that does not have a copy of O forwards read and write requests for O to $P(O)$. Write requests are forwarded by all the processors to $P(O)$, which in turn propagates them to all the processors of the O-scheme (except for the processor that issued the write, obviously).

Changing the O-scheme (the second part of DDA) consists of two comparisons called $exit$ (the O-scheme) and $enter$. The exit comparison is executed by every processor of the O-scheme, and the enter comparison is executed by $P(O)$.

Now we elaborate upon these two comparisons. For each write received by a processor j of the O-scheme, j compares two values that were determined by the fact that j kept a copy of O during the last time unit. One value is the increase in the cost of writes, and the other is the decrease in the cost of reads. If the increase is higher than the decrease, then j relinquishes its copy. Specifically, when a write is received by j, it performs the following comparison.

(Exit) If during the time unit that ends at the present time $(1+d) \cdot \sum_i \#W_i > d \cdot \#R_i$, then j relinquishes its copy.

The procedure for relinquishing the copy depends on the processor. Specifically, $P(O)$ relinquishes the copy differently than another processor j of the O-scheme. j relinquishes the copy by simply requesting $P(O)$ not to propagate any more writes to j. Further read requests of j will be sent to $P(O)$. In contrast, $P(O)$ relinquishes the copy by selecting another processor of the O-scheme, say i, and designating i as the new primary-copy processor. If $P(O)$ is the only processor in the O-scheme, then the new primary-copy processor i is the one that issued the maximum number of reads during the time unit. Designating a new primary-copy processor involves performing the following three functions: 1. $P(O)$ informs i of the

[1] Strictly speaking, the write from a processor that stores a copy of O costs only $c \cdot (1 + d) - d$, since the communication cost, d, is not incurred for the local copy. We ignore this point in the present paper, and we'll just mention that all the statements made in the rest of the paper can be easily adapted for the more complicated cost function.

designation, and sends to i the identification of the processors comprising the current O-scheme. 2. $P(O)$ broadcasts to all the processors in the network the fact that i in the new primary-copy processor. 3. If subsequently (the old) $P(O)$ receives read and write requests (because, for example, these may already have been on their way at the time the primary-copy processor switch was announced), then it simply propagates them to the new $P(O)$ (i).

The enter comparison is performed by $P(O)$ upon the receipt of a read request from a processor j that does not have a copy of O. $P(O)$ compares two values that were determined by the fact that j did not keep a copy of O during the last time unit. One value is the saving in the cost of writes, and the other is the penalty in the in the form of increased cost of reads from j. If the increased cost is higher, then $P(O)$ allocates a copy of O at j. Specifically, when $P(O)$ receives a read request from j, it performs the following comparison.

(Enter) If during the time unit that ends at the present time, $(1+d)\cdot\sum_i \#W_i \leq d \cdot \#R_i$, then $P(O)$ tells j to join the O-scheme (by saving a copy of O; $P(O)$ commits to propagate to j further writes of O).

4 Practical Considerations for the DDA Algorithm

In the DDA algorithm, concurrency control can be performed as follows. Each write exclusively locks all the copies of O, and each read share-locks the primary copy (at $P(O)$) or the local copy, depending on the processor which performs the read. It is easy to see that two-phase locking combined with this scheme ensures 1-copy-serializability.

Since the number of copies of O varies in time, the dynamic allocation algorithm is vulnerable to failures that may render O inaccessible. To address this problem, the user may impose reliability constraints of the following form: "The number of copies cannot decrease below a threshold, say t." If such constraint is present, then $P(O)$ refuses to accept the exit of an O-scheme processor, if such exit will downsize the O-scheme below the threshold. In other words, $P(O)$ informs a processor j that requests to relinquish the copy that the request is denied; subsequently, writes continue to be propagated to j. j continues to reissue the request whenever the exit comparison dictates to do so. The request may be granted later on, if the O-scheme expanded in the meantime.

Failure of any processor, except $P(O)$, does not affect the execution of reads and writes at the other processors. If $P(O)$ fails and there are no other processors in the O-scheme, then O will be unavailable until $P(O)$ recovers. Otherwise, the remaining processors will execute an election protocol to select the new $P(O)$. In case of network partition, the partition containing the primary-copy processor will continue to process reads and writes, and all the others will not do so.

Notice that in the DDA algorithm, the user may change the sensitivity of the algorithm to variations in the read/write pattern, by changing the le ngth of the time-unit. The shorter the time unit, the more sensitive the algorithm. On the other hand, shorter time-units induce more frequent changes of the O-scheme.

5 Relevant Work

The two main purposes for replicating data are increased availability and increased performance. This paper addressed the performance issue. Many performance-oriented works on replicated data consider the *static* problem of replication, namely establishing a priori a replication scheme that will optimize performance, but will remain fixed at runtime. This is called the file-allocation problem, and it has been studied extensively in the literature (see [WM], and [DF] for a survey). In contrast, in this paper we discussed dynamic replication.

Another approach to improve the performance in a replicated distributed database is to relax the serializability requirement. Works on quasi-copies ([ABG1, ABG2, BG]), lazy replication ([LLS]), and bounded ignorance ([KB]) fall in this category. These works also assume a static replication scheme. In contrast, our approach preserves one-copy-serializability, since the DDA algorithm is read-one-write-all (although the meaning of "all" changes dynamically).

In the theoretical computer science community there has been work on online algorithms (e.g. [BLS]), particularly for paging (e.g. [BS, CL, ST]), searching (e.g. [ST]) and caching (e.g. [KMRS]). These works are similar in spirit to the DDA algorithm; however, the models in such works is inappropriate for managing replication in distributed databases. For example, the cost of I/O, that is considered by the DDA algorithm, is not a factor in caching; and the replacement issue (i.e. which page to replace), that is important in paging, usually is not a factor in replicated data management.

Finally, in [WJ] we proposed algorithms for dynamic replication. However, these algorithms were dependent on the network having the tree topology, a limitation removed by the DDA algorithm; also, the [WJ] algorithms ignored the I/O cost of replication, and they were not amenable to changing the frequency of O-schema adaptation.

Acknowledgement

This research was partially supported by grants from AFOSR # 90-0135 and ONR # N00014-90-J-1601. The work of Ouri Wolfson was also supported by NSF grant IRI-90-03341. The authors would like to thank Dr. Ralph Wachter for a stimulating discussion on this topic.

References

[ABG1] R. Alonso, D. Barbara, H. Garcia Molina, "Quasi-copies: Efficient data sharing for information retrieval systems" *Proc. of EDBT'88, LNCS 303*, Springer Verlag.

[ABG2] R. Alonso, D. Barbara, H. Garcia Molina, "Data caching issues in an information retrieval system," *ACM Transactions on Database Systems,* **15**:3, 1990

[BG] D. Barbara, H. Garcia Molina, "The case for controlled inconsistency in replicated data," *Proc. of the IEEE workshop on replicated data,* 1990.

[BLS] A. Borodin, N. Linial, M. Saks "An Optimal Online Algorithm for Metrical Task Systems", *Proc. STOC*, 1987.

[BS] D. L. Black and D. D. Sleator "Competitive Algorithms for Replication and Migration Problems", manuscript, 1989.

[CL] M. Chrobak and L. L. Larmore "An Optimal Online Algorithm for k Servers on Trees", manuscript, 1990.

[DF] L. W. Dowdy and D. V. Foster, "Comparative Models of the File Assignment Problem", *ACM Computing Surveys,* **14**:2 1982.

[KMRS] A. Karlin, M. Manasse, L. Rudolph, D. Sleator, "Competitive Snoopy Caching", *Algorithmica,* **3**:1, 1988.

[KB] N. Krishnakumar and A. Bernstein, "Bounded ignorance in replicated systems," *Proc. of ACM-PODS* '91.

[LLS] R. Ladin, B. Liskov, L. Shrira, "A technique for constructing highly available distributed services," *Algorithmica* , 3, 1988.

[ST] D. Sleator and R. Tarjan, "Amortized Efficiency of List Update and Paging Rules," *CACM* 28(2), 1985.

[WJ] O. Wolfson and S. Jajodia, "Distributed Algorithm for Dynamic Replication of Data," *Proc. of ACM-PODS* '92.

[WM] O. Wolfson and A. Milo, "The Multicast Policy and Its Relationship to Replicated Data Placement", *ACM Transactions on Database Systems,* **16**:1, 1991.