DATA ALLOCATION IN MOBILE COMPUTING: A PROJECT DESCRIPTION

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

August 24, 1993

Abstract

Users of mobile computers will soon have access to shared online databases via wireless networks. We are developing a system that dynamically allocates and deallocates copies of the data-items in these databases to mobile computers, for optimizing cost and performance. This dynamic allocation of data-items to mobile computers is guided by input from the users concerning the type of wireless network, overall system reliability and availability requirements, and the objective function to be optimized.

1 Introduction

Mobile computers are becoming as ubiquitous as other personal computers. Laptops, notebook computers, and palmtops are being widely used for computing while commuting (e.g., by foot, cars, or airplanes). Users of mobile computers will soon have online access to a large number databases via wireless cellular-telephone or packet-switch networks. Access may be on demand, or by subscribing to automatically receive updates to the various data-items in these databases.

For example, suppose that most of the time a stock-broker, John, is reachable by the central computer that stores the price of each stock. Then he may subscribe to continuously receive the latest price a particular stock, say XYZ. In other words, the stock-price is allocated to (or cached at) John's computer. Then reads of the XYZ stock-price by John are local, and therefore they do not require communication. However, any update of the XYZ stock-price has to be propagated by the central computer to John.

Often the value of a data-item on a computer is allowed to diverge from the most up-to-date value in order to reduce communication. For example, John may allow the value of the XYZ stock-price at his computer to diverge from the most up-to-date value by at most 3%, or by at most 3 updates, or by at most 3 minutes.

An analogous example exists for a travel agent that checks seat availability on flights and makes reservations, or for a business-person that does both, trades in the stock market and makes flight reservations.

The *data allocation scheme* in distributed applications (or the *allocation scheme* for short), is the allocation of each data-item to users, and the divergence allowed for each (quasi-)copy. In other words, the data allocation scheme contains the 'subscription' of each user to receive updates of the various data-items. In general, the data-items may be as small in size as a number, or as large as a video-clip.

The allocation scheme critically affects the cost, reliability, and performance of distributed applications in general, and mobile computing ones in particular. Consider first the cost. If there are three users, each of which requests its copy of a data-item to be kept up-to-date, then each update will necessitate at least three messages. Experiments have shown that an inappropriate allocation scheme can quadruple the communication-cost of executing a distributed application. Since wireless communication is more expensive than wire-communication, the implications of the allocation scheme are more critical in mobile computing than in stationary computing. Similarly, the allocation scheme affects the reliability and availability of the system. For example, if a data-item is allocated to computers A, B, and C, and all three of them are down or 'out of reach', then the data-item is unavailable. Again, since portable computers are less reliable and they easily move out of reach, the reliability issue is more complex in mobile computing.

The objective of this project is to research, develop and evaluate a system for automatic data allocation in mobile computing environments. By automatic data allocation we mean that the system (rather than the user) will establish and change the allocation scheme. Why shouldn't each user of a mobile computer determine at each point in time which data-items should be allocated to her computer, depending on the expected access pattern ? (For example, if a user expects to heavily use a data-item in the next 10 minutes, then she should allocate a copy of this item to her processor). In other words, why should the software determine the allocation scheme automatically, rather than each user doing so manually?

The reason for automatic allocation is that the optimal scheme depends on the global state of the system, of which individual users may not be aware. For example, the cost benefit of allocating a copy of the item to a user depends on the ratio of the number of local reads (i.e. reads issued by the user) to global updates (i.e. the total number of updates issued by all users). A user usually knows his expected read-activity in the near future, but not the global update activity. As another example, although a user may want to cancel her subscription to receive updates of a particular data-item, she may not be allowed to do so if a reliability constraint requires that there are at least three up-to-date copies of the data-item in the system.

2 Innovative Aspects of the Project

In traditional distributed databases, the allocation scheme is determined at distributed-database design, and it usually remains fixed as long as the computers are functioning properly. However, the optimal data-allocation depends on factors that change dynamically at run-time, such as read/update ratio, network connectivity, and objective function to optimize (e.g. cost, performance, availability). For example, if during a period of time an item is updated frequently and read infrequently, then having a single copy of the item at a central stationary computer optimizes communication cost since it avoids the propagation of updates to the mobile users. In contrast, if for a period of time the item is read frequently and updated infrequently, then copies should be cached at the reading processors in order to optimize communication cost. Changing the allocation scheme in this fashion requires that the system *learns* the access pattern of each user.

Sometimes the access to a data item does not follow a pattern, but is totally chaotic. It turns out that in this case some form of cache-invalidation algorithm optimizes the cost.

We are developing a system that learns the current access-patterns of each user; the system changes the data-allocation accordingly. This dynamic allocation is guided by input from the users concerning the environment (e.g. the type of network), overall system reliability and availability requirements, and the objective function to be optimized.

3 Project Objectives

The project has two main objectives. First, develop, implement, and analyze, a set of algorithms for dynamic allocation and replication of data in mobile computing environments. The purpose of the algorithms is to adapt the allocation to changes in the access pattern, changes in the availability constraints, and changes in the network availability. Second, build a software system and software tools (that can be packaged with an operating system or with a distributed database management system) to provide dynamic allocation and replication. Following are some details on each one of these objectives.

1. *Develop, implement, and analyze algorithms for automatic dynamic allocation.*

   Each algorithm adapts the allocation scheme to the current conditions in the distributed system. Multiple algorithms are necessary since the appropriate one depends on the current requirements of the system, the objective function, and the communication costs. These may change during system's operation, or from data-item to data-item. For example, an algorithm for optimizing the average *time* of a request is different than the algorithm for optimizing the average *cost* of a request. The algorithm for a *regular* read-write pattern, i.e. a pattern in which the recent past is indicative of the near future, may be different than an algorithm for a *chaotic* read-write pattern.

   Other factors that determine the appropriate algorithm are the size of the data-item (which determines the cost of transmitting it). In other words, the algorithm for a large data-item is different than the algorithm for a small data-item. Also, the algorithm for packet switch networks may be different than the algorithm for cellular networks. Divergence-tolerance also influences the choice of the algorithm. Specifically, if the reads issued by users tolerate divergence of the item presented from the latest version, then the algorithm needs to be more intelligent in order to consider the tolerance history for each data item.

2. *Build a dynamic allocation and replication system.*

   The system will provide tools and language primitives for dynamic data allocation. It will run on both, the portable and the stationary computers. Automatic dynamic allocation is provided by an algorithm of the type discussed above, and it is transparent to each user. The dynamic allocation system will enable selection of a dynamic allocation algorithm, and parameterizing it. This selection will be performed manually by the user. An example of a parameter to a dynamic allocation algorithm is a constraint of the following type: 'There must be at least t copies of the data-item D in the network (even at the expense of degrading performance)', or, 'Users i and j must have the latest version of the data-item D at all times', or 'At any time user i or user j must have the latest version of the data-item D', or 'Access patterns that are more than 5 minutes old should be disregarded for learning purposes'.

   The dynamic allocation system will also provide commands for manual intervention and monitoring of the allocation process. Examples of such commands are: 'Create (delete) a copy of the data-item D at computer x' or, 'Retrieve the total number of data-items stored at computer x'. The system will guarantee execution of these primitives in the face of failures.

4 Comparison with Relevant Work

4.1 Relevant Work on Data Allocation in Distributed Systems

Existing research on data allocation in concentrates on stationary (as opposed to mobile) distributed databases. Generally, there are two main purposes in determining the data allocation and replication in distributed systems, namely, performance and availability. The works that concentrate on the availability issue (see for example [1, 15]), assume a fixed allocation scheme in the absence of failures. In a mobile environment these methods may incur prohibitive costs, since they do not adapt to changing conditions.

Performance-oriented works on replicated data consider the <u>static</u> problem of replication, namely establishing a priori an allocation 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 [12, 16] for a survey). These works assume that access patterns are known in advance. We feel that this is not always a realistic assumption, particularly in a mobile computing environment. The works on quasi-copies ([17])

also assume that the allocation scheme, particularly the divergence-threshold, is fixed.

4.2 Relevant Work in Caching and Virtual Memory

In the computer architecture and operating systems literature there are studies of two subjects related to data allocation in distributed systems, namely caching and distributed virtual memory (see for example [2, 4, 5, 8, 9, 10]). In these methods, when a processor issues a read to a shared page that is not in its memory, a read-page-fault is triggered and the page-fault interrupt handler requests the page from another processor; when received, the page is cached locally.

However, there are several important differences between Caching and Distributed Virtual Memory (CDVM) on one hand, and data allocation in distributed systems on the other. First, CDVM methods have been devised for performance enhancement only, and consequently they do not consider the reliability and availability issues.

Second, the size of the cache is often very limited. Thus important issues in CDVM literature are cache utilization, and page replacement strategy (e.g. LRU, MRU, etc.), namely which page to replace in the cache when the cache is full and a new page has to be brought in. In contrast, in distributed systems data may reside in main as well as secondary storage, thus an item is seldom "uncached" for storage limitations.

Third, the appropriate cost and performance measures for distributed data are different from cost and performance measures for CDVM. For example, in CDVM, when a page that is read is found in the cache, no I/O cost is incurred. On the other hand, in distributed systems even when an object is allocated to a user, it may reside in secondary storage, leading to an I/O cost incurred at the time of read.

Fourth, the architecture assumed in CDVM methods is bus-based (e.g. [5, 10]). This architecture supports broadcast at the same cost as a single-cast, and on the other hand incurs contention. In contrast, in distributed systems communication is often point-to-point.

5 Preliminary results

Our preliminary results have appeared in [13, 14, 6, 7, 11]. In [13, 14, 6] we have developed several algorithms that change the data-allocation scheme of an object dynamically, as the read-write pattern of the object changes in the network. They are called Cost-Adaptive-Replication (CAR), Time-Adaptive-Replication (TAR), Dynamic-Data-Allocation (DDA), and Competitive-Dynamic-Data-Replication (CDDR).

The algorithms that we developed are **integrated** with 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 piggy-backing, on the message to y containing the object, an indication that y should keep a replica (and that x will propagate writes to y).

The algorithms that we developed are also **distributed** as opposed to centralized. In a centralized dynamic allocation algorithm, each processor periodically transmits the relevant information (usually statistics) to some predetermined processor, x. In turn, x computes the objective function and orders the change of allocation scheme. In contrast, in a distributed algorithm, each processor makes decisions to locally change the allocation scheme, and it does so based on statistics collected locally. An example of a local change to the allocation scheme is the following: a processor, x, relinquishes a copy of the object (by indicating to x's neighbors that writes of the item should not be propagated to x). This change may result from a comparison of the number of reads to the number of writes, at x (i.e. locally collected statistics).

Distributed algorithms have two advantages over centralized ones. First, they respond to changes in the read-write pattern in a more timely manner, since they avoid the delay involved in the collection of statistics, computation, and decision broadcast. Second, their overhead is lower.

Although in our algorithms the decision-making and scheme-changing are *local*, each algorithm continuously moves the allocation scheme towards an optimal one, where optimality is defined with respect to a *global* objective function. For example, the CAR and CDDR algorithms optimize the communication cost objective function. The communication cost of an allocation scheme is the average number of interprocessor messages required for a read or a write of the object. The TAR algorithm optimizes the communication time objective function. The communication time of a allocation scheme is the average length of a message path required for a read or write of the object. The DDA algorithm optimizes a function that encodes the communication cost as well as the I/O cost of an allocation scheme.

The CAR, TAR, and DDA algorithms are **convergent** in the following sense. Assume that the pattern of access to each object is generally regular. For example, during the first two hours processor x executes three reads and one write per second, processor y executes five reads and two writes per second, etc.; during the next four hour period processor x executes one read and one write per second, processor y executes two reads and two writes, and so on. Then, the CAR, DDA, and TAR algorithms will converge to the optimal allocation scheme for the global read-write pattern during the first two hours, then it will converge to the optimal allocation scheme for the global read-write pattern during the next four hours, etc. The DDA algorithm also accepts an availability constraints of the form: " The size of the allocation scheme of object O must not decrease below some threshold, t ". If so, it will converge to the allocation scheme that is optimal among the ones that satisfy the constraint. In other words, the convergent algorithms adapt to the current read-write pattern.

The CDDR algorithm is **competitive** rather than convergent. This means that the ratio (the cost of CDDR) / (the lower-bound cost) is always smaller than some constant, regardless of the access (i.e. read-write) pattern. The lower bound on the cost of an access pattern is the cost incurred by the optimal centralized serial offline algorithm. A *serial* algorithm does not allow concurrent read-write requests. An *offline* algorithm has complete knowledge of all the future read-write requests. In contrast, the CDDR algorithm is distributed, it can be incorporated into a concurrency control mechanism that preserves 1-copy-serializability (see [3]), and it is online in the sense that it does not have knowledge of future requests.

Competitiveness and convergence are two criteria for evaluating online algorithms. A competitive algorithm may not converge to the optimal allocation scheme when the read-write pattern stabilizes, and a convergent algorithm may unboundedly diverge from the optimum when the read-write pattern is irregular. A competitive online algorithm is more appropriate for chaotic read-write patterns, in which the past access pattern does not provide any indication to the future read-write pattern. In contrast, a convergent online algorithm is more appropriate for regular read-write patterns.

In [7] we compared the communication cost of two existing algorithms, static allocation (SA) and dynamic allocation (DA). DA is a caching algorithm that uses cache-invalidation for updates. In other words, a computer that reads a data-item caches it in its local memory; when the data-item is updated all the cached copies are invalidated. SA follows the traditional approach in distributed databases, namely fixed allocation. We showed that DA is competitive, whereas SA is not. We also compared the two algorithms experimentally, by generating at a set of computers long sequences of read-write requests for a common data-item. The experiments indicate that for a fixed write probability, there exists a static allocation scheme for which SA is optimal. However, if the write probability changes, then the DA algorithm has a lower communication cost.

In [11] we developed and analyzed dynamic allocation schemes for the case a read issued by the user is allowed to return a value of the data-item that diverges from the most up-to-date version. Divergence is specified using amount (e.g. 4%), or time (e.g. 4 minutes), or number of updates (e.g. 4 updates). So a read is parameterized with the allowed divergence. For example, read (data-item, 4 minutes) denotes a read for a version of the item that diverges from the most up-to-date one by at most four minutes.

We developed algorithms that learn the probabilities of the various divergences allowed by reads, and optimize the communication by requesting periodic refreshment of the local copies at the mobile computer. So, for example, the allocation scheme may have a subscription for a computer to receive one in every four updates. The period of refreshment depends on the current read-divergence probabilities.

Thus, when reads allow divergence, then the algorithms change the allocation scheme to contain subscriptions for periodic refreshment. However, if every read requests the most up-to-date version of the item, then our algorithms devise an allocation scheme in which each mobile computer subscribes to receive either every update to the data-item, or none at all.

References

[1] D. Agrawal, A. El Abbadi, "The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data", *Proc. of 16th VLDB*, August, 1990.

[2] J.K. Bennett, J.B. Carter and W. Zwaenepoel, "Adaptive Software Cache Management for Distributed Shared Memory Architectures", *Proceeding of the 17-th Annual Int'l Symposium on Computer Architecture*, Pages 148-159, May 1990

[3] P. Bernstein, V. Hadzilacos, and N. Goodman, "Concurrency control and recovery in database systems", *Addison-Wesley* (1987)

[4] M.J. Carey, M.J. Franklin, M. Livny, E.J. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architectures", *ACM-SIGMOD '91*, Pages 357-366

[5] Susan J. Eggers and Randy H. Katz, "Evaluating the Performance of Four Snooping Cache Coherency Protocols", *Proceeding of the 16-th Annual Int'l Symposium on Computer Architecture*, Pages 2-15, June 1989

[6] Yixiu Huang, Ouri Wolfson, "A Competitive Dynamic Data Replication Algorithm", *IEEE Proc. of 9th Int'l Conf. on Data Engineering* '93, Vienna, Austria

[7] Yixiu Huang, Ouri Wolfson, "Object Allocation in Distributed Databases and Mobile Computers", submitted for publication.

[8] Kim Korner, "Intelligent Caching for Remote File Service", *Proceeding of the 10-th Int'l Conference on Distributed Computing Systems*, Pages 220-226, May 1990

[9] Kai Li and Paul Hudak, "Memory Coherence in Shared Virtual memory systems", *ACM Trans. on Computer Systems*, 7(4):321-359, November 1989

[10] J. Lee and U. Ramachandram, "Synchronization with Multiprocessor Caches", *Proceeding of the 17-th Annual Int'l Symposium on Computer Architecture*, Pages 27-37, May 1990

[11] R. Sloan and O. Wolfson, "Divergence Protocols for Data Allocation in Mobile Computing", in preparation.

[12] O. Wolfson and A. Milo, "The Multicast Policy and Its Relationship to Replicated Data Placement", *ACM TODS*, 16 (1), 1991.

[13] Ouri Wolfson and Sushil Jajodia, "Distributed Algorithms for Dynamic Replication of Data", *Proc. of ACM-PODS*, 1992

[14] Ouri Wolfson and Sushil Jajodia, "An Algorithm for Dynamic Data Distribution", *Proc. of the 2nd Workshop on Management of Replicated Data (WMRD-II)*, 1992, pp. 62-65

[15] S. Davidson, H. Garcia-Molina, D. Skeen, "Consistency in Partitioned Networks", *ACM Computing Surveys*, **17**:3, 1985.

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

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