FLEXIBLE DATA STORAGE FOR MOBILE COMPUTING

Nuno Preguiça, J. Legatheaux Martins, Henrique J. Domingos, Jorge Simão
Department of Computer Science
Faculty of Sciences and Technology - New University of Lisbon
Quinta da Torre, 2825 Monte da Caparica, Portugal
{nmp,jalm,hj,jsimao}@di.fct.unl.pt

KEY WORDS

Mobile computing; asynchronous collaborative applications; distributed data storage; object framework.

ABSTRACT

In this paper we describe a flexible object storage system aimed at supporting collaborative applications in large-scale environments that include mobile computers. We present an integrated solution to two major problems that arise in such environments: data availability and concurrency control. The first is tackled by the flexible combination of weakly consistent server replication and client caching. The second is tackled through an open object framework that enables easy object development using type specific conflict detection and resolution. This object storage serves as a supporting platform to build distributed and mobile collaborative applications.

INTRODUCTION

Mobile computing is characterized by some intrinsic constraints related with available connectivity, power and hardware resources [11]. Despite the impressive progress in hardware and communication technology [3], mobile hosts have to face lower and highly variable bandwidth capabilities when compared with those of stationary computers. Moreover, these reduced capabilities impose periods of complete disconnection.

As users must be able to access data to perform useful work, the utility of any computer depends largely on the efficiency of the underlying storage system. In mobile environments, where periods of complete disconnection are frequent, data availability must be provided relying on local replicas of data. To support collaborative applications effectively, users must be allowed to perform their contributions in any mobile host without any restrictions, even when disconnected. These concurrent updates must be subsequently merged, and their intended effects taken into account, to produce the final state of the shared data.

In this paper we present an overview of the DAgora storage system, which has been designed to support asynchronous collaborative applications in large-scale settings that include mobile computers.

SYSTEM OVERVIEW

The DAgora distributed storage system manages objects, known as coobjects – from collaborative objects. These coobjects may be rather complex (such as documents or scheduler calendars) and be implemented as an arbitrary composition of regular objects. Sets of related coobjects are grouped in volumes that represent collaborative workspaces and store the data associated with a given workgroup and/or cooperative project.

To provide high availability of data and support for workgroups that are distributed across several physically disjoint locations, volumes of coobjects are replicated by groups of servers. The location of servers must be selected to decrease users' connectivity requirements and nothing prevents a powerful mobile computer from hosting a DAgora server.

Since traditional replication schemes providing one copy serializability and strict consistency yield unacceptably low write availability in partitioned networks or in the presence of disconnected computers [1], weak consistency of replicated data is desirable. Consequently, DAgora has adopted a model in which clients can read and write to any replica independently – read any / write any model.

Updates are propagated among servers during occasional, pair-wise communications known as anti-entropy sessions [9], thus taking into consideration the connectivity characteristics of mobile environments. This epidemic scheme guarantees that each server eventually receives all updates from every other, either directly or indirectly. Therefore, consistency among data replicas may be eventually achieved in a quiescent state when all updates have been propagated to all replicas.

To increase data availability and system usefulness for mobile users, DAgora implements a caching mechanism in clients. Applications employ a *get / modify locally / put changes* model of data access. Private copies of coobjects are obtained through the client component and they are modified by usual method invocations. Updates are exported to a server using a *store-and-forward* model: the client component stores the updates until incremental propagation is possible. The outlined architecture is depicted in figure 1.

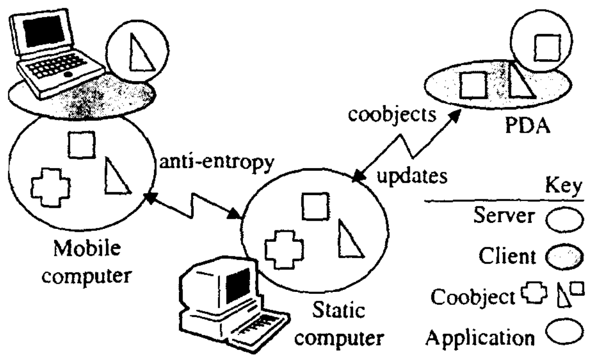

Figure 1 – DAgora architecture composed by three computers with different configurations. Coobjects are replicated by servers, cached by clients and manipulated by users' applications.

Our goal in designing DAgora was to provide system support to ease the development of asynchronous collaborative applications for mobile environments. To this end, providing transparent data availability is just one of the problems involved. Another one, perhaps more difficult to solve, it is to handle concurrent updates in a weakly connected system based on weakly consistent replication.

Distributed file systems, such as Coda [6] and Ficus [8], use system and user defined conflict resolution programs to merge divergent replicas. These systems work very well in environments with few conflicts and their strategy is quite effective for objects with simple semantics – e.g. file directories. They have proven the value of semantic conflict detection and resolution. However, experimental results (30% unsolved update/update file conflicts [8]) suggest that the resolution of conflicts based on simple state propagation may be very difficult for complex objects.

We believe that the observed shortcomings can be overcome executing conflict resolution at the granularity of individual operations and further exploiting domain-specific knowledge (thus extending the principles applied in Coda and Ficus). Several systems, such as Bayou [12], Rover [4] and Sync [7], use different approaches based on the above principles. In [10] we discuss the reasons why we believe our system is more suitable for the target environment.

In the DAgora storage system, updates performed by users are propagated to a server and among all the servers as method invocations. The effect of each update in the *official* state of a coobject is finally determined by the execution of the associated method in each server. Besides the complexity of log management, this approach presents several problems that must be taken into consideration: interpretation of results in user applications; consistency among servers; and respect by the user's intentions when applying one update (to a different coobject state). To help programmers solving these problems in their specific applications, DAgora presents an open object framework that eases coobject development reusing several predefined solutions.

OBJECT FRAMEWORK

The DAgora open object framework structures each coobject in the following five disjoint components: attributes, log, log-ordering, data and capsule. In this section we will briefly present each component, their available predefined implementations and outline how they are used to solve the problems mentioned earlier.

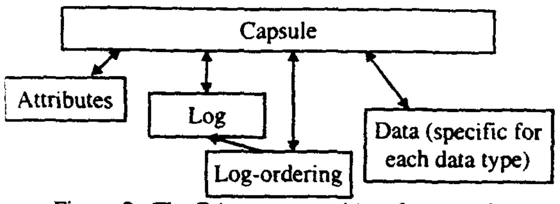

Figure 2 –The DAgora open object framework.

The component "attributes" is used to store general-purpose information relative to the coobject and meta-information relative to the replication process.

The component "log" stores the updates performed by users. For each sequence of updates exported by an application, the log adds a timevector representing the initial version of the coobject. For each sequence of updates received by a server from a client, the log adds a timestamp that allows the establishment of a total order among all updates.

The component "log-ordering" is used to order the execution of updates relying on the information added by the component "log". Currently, several implementations are available, namely: no order, causal order, total order based on a sequencer replica, total causal order based on stability tests, and optimistic total order using undo/redo [5]. This component should be selected according to the semantics of operations to guarantee the desired consistency among servers. In [10] we further discuss available implementations and their usage.

The component "data" represents the real data type being created, with its associated state and operations. Finally, the component "capsule" aggregates the components of a coobject and determines its composition. Common coobjects are composed by one instance of each component. Next, we present some base implementations available for these components and their usage in real applications.

Previous research has concluded that the definition of two states for an update, committed and tentative, is very useful in mobile environments [4,12]. For instance, in a scheduler application, reservations executed by users must be considered tentative until they are committed. Users should be allowed to see tentative data to avoid possible conflicts (tentative data represent a foresight of the coobject's state). In the DAgora system, a programmer may easily create a coobject that stores a tentative and a committed version of the data, relying on simple data objects and using an extended capsule. This capsule is composed by two instances of the components "log-ordering" and "data" and transparently maintains both states – committed data results from the execution of stored updates using a pessimistic total order, while tentative data results from the execution of unstable updates to the committed state using causal order.

For some applications, it is impossible to solve conflicts automatically. For instance, if a base element (e.g. section) of a structured document is modified concurrently by two users, the system usually can neither decide which modification is the best, nor merge both modifications. In such cases, two versions of the conflicting element must be created and resolution must be left to users. In DAgora, we have created a component "data" that implements a set of generic objects with multiple versions. Concurrent modifications of the same object are detected and solved automatically creating multiple versions. Programmers may extend this component and define automatic merging procedures or let this work to users. Another component implements a generic tree-structured organization on top of the above set of objects and can also be extended by programmers. These base components have been used for implementing several structured documents manipulated by a collaborative editor [10].

When the *multi-version* "data" components are not used, the programmer must take into consideration the DAgora model of operation, when implementing the component "data". However, to guarantee that users' intentions are respected when updates are applied in each server and eventual conflicts are detected and solved, the DAgora object framework provides the necessary support to the usage of some simple techniques. First, the existence of concurrent updates may be tested using the timevector associated with each operation. Second, the defined preconditions for the execution of an update may be checked. Third, the definition of alternative actions to be executed dependent on the coobject's state is possible. Fourth, the definition of state-independent operations is also possible. More complex techniques, such as updates' transformations [2], may also be implemented using the updates stored in the component "log".

Our experience with some implemented applications [10] suggests that most applications will use one or two simple techniques (e.g., a careful operation definition associated with a regular precondition checking has been used in a scheduler application, while our *multi-version* components rely uniquely on the timevectors associated with each update). The creation of a new coobject is reduced to the implementation of the component "data" and to the selection of the desired available semantics for the other components.

FINAL REMARKS

The DAgora data storage presents an architecture that allows adaptation to specific environments using different system configurations. It provides high availability of data taking into consideration the constraints of mobile computing.

The associated DAgora open object framework allows programmers to develop specific solutions for their problems. As there is no single solution that solves all problems, the flexibility that is provided by the open object framework is fundamental to support different types of applications. Moreover, the object framework eases the task of programmers allowing them to reuse several predefined components that handle most of the complexity associated with data distribution.

REFERENCES

[1] Coan B., Oki B., Kolodner E. Limitations on database availability when networks partition. In *Proceedings 5th ACM Symposium on Principles of Distributed Computing*, August 1986.

[2] Ellis C., Gibbs S. Concurrency Control in Groupware Systems. In *Proceedings of the ACM SIGMOD Conference on the Management of Data*, June 1989.

[3] Imielinski T., Korth H. Introduction to Mobile Computing. *Mobile Computing*, ed. T. Imielinski and H. Korth, Kluwer Academic Publisher, 1996.

[4] Joseph A., DeLespinasse A., Tauber J., Gifford D., Kaashoek M. Rover: A Toolkit for Mobile Information Access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

[5] Karsenty A., Beaudouin-Lafon M. An algorithm for distributed groupware applications. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, May 1993.

[6] Mummert L., Ebling M., Satyanarayanan M. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

[7] Munson J., Dewan P. Sync: A Java Framework for Mobile Collaborative Applications. *IEEE Computer*, June 1997.

[8] Page Jr. T., Guy R., Heidemann J., Ratner D., Reiher P., Goel A., Kuenning G., Popek G. Perspectives on Optimistically Replicated, Peer-to-Peer Filing. *Software-Practice and Experience*, vol. 28(2), February 1998.

[9] Petersen K., Spreitzer M., Terry D., Theimer M., Demers A. Flexible Update Propagation for Weakly Consistent Replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, 1997.

[10] Preguiça N., Martins J., Domingos H., Simão J. System Support for Large-Scale Collaborative Applications. Technical Report, TR-01-98 DI-FCT-UNL, Dep. Computer Science, New University of Lisbon, 1998.

[11] Satyanarayanan M. Fundamental Challenges in Mobile Computing. In *Proceedings of the 15th ACM Symposia on Principles of Distributed Computing*, 1996.

[12] Terry D., Theimer M., Petersen K., Demers A., Spreitzer M., Hauser C. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.