Optimistic Replication in Pharos, a Collaborative Application on the Web

Esther Pacitti

Atlas Group, Inria and IRIN,
Nantes - France
Esther.Pacitti@irin.univ-nantes.fr

Olivier Dedieu

GIE Dyade (Bull-INRIA)
Le Chesnay, France
Olivier.Dedieu@inria.fr

Abstract

Pharos is a collaborative application which enables users to share document annotations. Annotations of a same subject are stored altogether in channels, and channels are replicated to improve performance. What characterises data replication in a collaborative application like Pharos is the way users see data. If, for instance, mutual consistency is required, then collaboration should be synchronous. On the other hand, if remote users can work in disconnected mode, collaboration can be asynchronous with weaker constraints on data freshness. In this paper, we focus on asynchronous replication which is typically required on the Web. We propose an optimistic replication model based on lazy group replication and a protocol to detect and resolve potential conflicts to refresh the replicas. This protocol is based on the ordering of write operations at each site using its timestamps values. Careful log management is the key to its implementation. We describe the implementation of our model in the Pharos application.

Keywords: *Web, Collaborative Applications, Asynchronous Replication, Optimistic Replication, Log Management*

1 Introduction

The Web enables and encourages collaboration between remote users. There are various ways to collaborate but they all involve groups of people sharing a common interest, which may be professional (e.g. Java programming) or personal (e.g. painting). These groups of people form communities that share information in various ways. The most basic ways (forums, mailing lists, IRC channels) are restricted to the exchange of text messages. In advanced collaborative applications, such as groupware, information is more structured and *data replication* is used for improving information sharing. With the emergence of communities on the Internet, we can expect the majority of Web applications to integrate the collaborative dimension through data replication.

Pharos [http://webtools.dyade.fr/pharos/] is a collaborative application on the Web developed in GIE Dyade, a joint venture between Bull and INRIA. It allows users to share their knowledge on a specific topic through *annotations* that are stored in a specific database called *channel*. If users are located world-wide, the latency of annotation fetching can become unacceptable. To improve performance, channels can be replicated close to users locations. If users requires mutual consistency, then data replication should be *synchronous*. On the other hand, if mutual consistency can be relaxed, for instance users may want to work in disconnected mode, then data replication must be *asynchronous* with weaker constraints on data freshness [11]. In this case, pessimistic replication protocols which lock write operations [1] to get strong consistency are not appropriate. Our solution is to trade consistency for efficiency using optimistic replication. An optimistic replication protocol accepts all writes as they come, and thus may introduce conflicts which should be resolved at some point, guaranteeing *eventual consistency*.

Another important characteristic is the nature of replicated data (replicas). Replicas may be strongly or weakly structured (e.g. objects, tables or text) as well as modifiable or destructible. In addition, if replicas are interrelated among themselves, special care is needed to update them. For instance, a department *d* is deleted from replica R while there is an insertion of a new employee in replica R'. The problem then is how to preserve consistency. The nature of the replicated data combined with the interaction mode (synchronous or asynchronous) determine the replication model for a collaborative application.

In this paper, we focus on asynchronous collaborative applications with interrelated replicas, which are typically required on the Web. Existing replication solutions [10,13], developed in the context of relational DBMS, are limited to unrelated replicas, i.e. single tables, and thus do not address the more general needs of collaborative applications. In Section 2, we present Pharos, the application which motivated our research. In Section 3, we propose an optimistic replication model based on lazy group replication (all replicated data are updateable). We also present the asynchronous protocol used to detect and resolve the conflicts, and refresh the replicas. The principle is to provide a total order on write operations at each site. Section 4 presents the underlying replication architecture used by Pharos. Section 5 presents the main issues of the Pharos implementation. Finally, we present our conclusions in Section 6.

2 Pharos: A of Collaborative Application on the Web

The Web is the easiest way to disseminate information to the global community. Nevertheless, users usually find it hard to find the information they are looking for. The Web infrastructure relies on hypertext and does not provide a mechanism to quickly find a valuable document on a precise topic. As a result, people search information in dedicated sites that index the Web, or consult alternative communication channels such as news groups or mailing lists. Search engines index the content of the Web and provide a query interface of a Web site. Their robots travel through hyperlinks to discover new documents. Their success is due to two main reasons: (i) they are the easiest place to find information; (ii) they have a relatively good coverage of the Web [6]. However, one major drawback of search engines is that they only give links. The user must read and determine whether if the proposed documents are interesting or not. This work is time-consuming and must be done by all users for a given search. Factorizing this repetitive effort would help people to find more quickly the most relevant documents. There is clearly a need to provide a system that integrates functionalities similar to search engines but which references valuable information such as that exchanges within groups of experts.

Pharos is a collaborative infrastructure which allows people to share their knowledge on a specific topic. People finding a valuable document put *annotations* in the appropriate *channel.* An annotation is a structured datum referencing a URL. It is composed of values such as a title, a rating (a subjective note), a free comment and a list of keywords. A channel is a database of annotations dedicated to a specific topic (e.g. games, music, Java language, Web technologies, etc). People interact with Pharos by using a *browser assistant.* It observes the URLs the user accesses, requests channel servers for annotations associated with each URL and displays them. As a result, the user can quickly evaluate the interest of the document. Pharos also helps to find rated information. Queries can be performed to extract annotated documents matching some criteria (e.g. all documents rated as "good" by Bob and categorized with the keyword "documentation:book"). Futhermore, new keywords may be *added*, *modified*, and old ones may be *deleted*.

As the amount of available annotations increases, it becomes more difficult for users to exploit them. If users are located world-wide the latency of annotation fetching can become unacceptable. To improve performance, channels can be replicated close to user's locations. This creates consistency issues which Pharos addresses with an optimistic protocol.

3 Optimistic Replication Model

Our replication model is based on a lazy replication scheme [9]. We focus on *lazy group* replication which is well-suited for collaborative applications on the Web. We characterize a lazy group replication scheme using: *ownership*, *propagation*, *refreshment* and *configuration*. We present the principles of our optimistic protocol based on the previous parameters. To easy comprehension, we use a simple electronic address book application as underlying example.

The *ownership* parameter defines the permissions for updating replica copies. If a replica copy is *updateable*, it is called a primary copy, otherwise it is called a secondary copy. In lazy group replication, primary copies are written asynchronously, i.e. the property of *mutual consistency* is relaxed [8]. Therefore, a site may work in connected mode or in disconnected mode. The *configuration* parameter defines the component sites of a replication scheme. In our case all nodes are master sites (stores primary copies).

Object-oriented languages (e.g. Java) are widely used to write applications on the Web. Therefore, the replicated data may structured in complex ways. We consider that the schema of each primary copy is defined using a Java class. Thus, each replica R consists of a set of objects. For instance, let us define R using the address book schema (name, email, homePhone, etc). Thus, a replica R is a set of objects , each providing address and telephone information of a person. Each site that stores R is called a *master* for this copy.

In lazy group replication, a conflict arises whenever two masters of a same replica R write the same attribute of the same object. For instance, a conflict arises when two master sites change Olivier Dedieu's e-mail address to a different one, which yields an inconsistent state since users may see two different e-mail addresses. Such write operations are not commutative and must be executed in the same order at all sites. Such property is called total oreder. Another case of conflict may also happen with interrelated replicas. Suppose that object *O* in Replica *R* is deleted, however it is referenced by a new created object *O'* at replica *R'*. In this case, write operations execution order must reflect the dependency between replicas. Such property is called causal order. To enforce consistency respecting causal and total order restrictions we use a pseudo Lamport *like* clock (*logical timestamps with temporal increment* [4]*)* to order and produce timestamps values during synchronization.

We now present the principles of the optimistic protocol using the propagation and refreshment parameters. The *propagation* parameter defines "when" the writes done on R at master must be propagated towards other masters of R . In our replication model, we consider that update propagation occurs exclusively between a *Sender* and a *Receiver* in a client-server fashion. The sender performs propagation and the receiver performs refreshment. Therefore, the receiver can participate at most at one refreshment at a time. On the other hand, the sender may propagate to several receivers.

The *refreshment* parameter defines the *management* of write operations to refresh primary copies. In fact, it implements conflict detection and resolution using the log history H of each site, which is used to store the sequence of write operations locally. Therefore, refreshment is performed in three steps:

1) Integration of non-conflicting write operations using the history of each site involved
2) Conflict detection and resolution. For write integration and update/update conflicts, we choose to execute them in timestamp order. On the other hand, for update/delete conflicts, the delete operation is favoured.
3) Finally, updates are propagated *epidemically* [3]: changes made to one copy eventually migrate to all, which guarantees that all replicas will finally reach a consistent state.

4 System Architecture

Figure 1 shows the replication architecture for our model that is similar to the one presented in [12]. Each site has 3 main components: *Object Manager, Log Manager, Synchronizer*.

Object Manager: it implements all procedures and data structures necessary to represent the replicas (Java objects) and manage them (update, delete, create, insert). Replicas are stored in the local database system. After performing local write operations on R, the object manager invokes the Log Manager to log the sequence of operations on *R*. Conversely, whenever R is refreshed, the Log Manager requests the Object Manager to update *R*, because updates are first registerd in the Log. Whenever it runs out of memory, the Object Manager uses a DBMS to swap objects to disk.

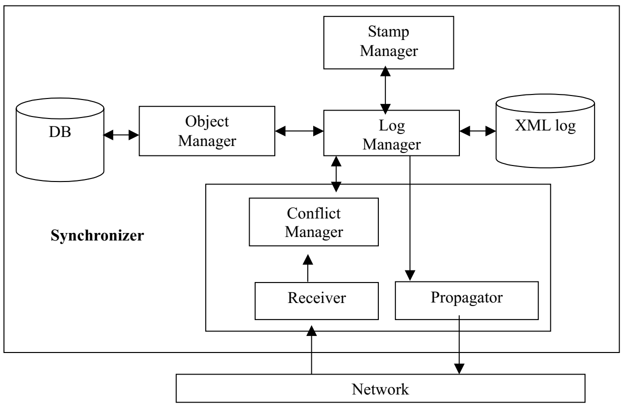

Figure 1: Replication Architecture

Log Manager: To identify conflicts, we need to know the sequence of write operations that produced the conflict. For instance, if a data is missing on a replica, additional information is necessary to check whether the replica never received the data or the data was received but destroyed. Thus, replica storage plays an important role for conflict detection and resolution. It must provide mechanisms to store and retrieve all write operations that may generate conflicts so that conflicts may be analysed to produce a *conciliated* sequence of *write operations*. The log manager is used for replica storage. It records in the log the sequence of all write operations performed on R (create, update, delete) and provides support for querying the log for conflict detection and resolution purposes. Each logged operation is represented in XML which is useful here to describe and query structured data. For each operation on an object, the log manager adds *timestamp, identifier* and *operation* information. *Timestamp* is the clock value when the operation started and is given by the *Stamp Manager*. *Identifier* is the object identifier and *operation* is the type of operation. Figure 2 shows an example of a sequence of 2 log records, as two XML elements with all information represented as XML attributes.

Synchronizer: implements propagation using the network interface when the site acts as a sender. It also performs refreshment on R, using the *Conflict Manager,* whenever the site is a receiver. The Conflict Manager queries the Log Manager for a sequence of write operations since a specific timestamp value. Then, it compares the incoming sequence of write operations sent by the Sender to detect and resolve conflicts. Finally, the Conflict Manager transmits to the log manager the *conciliated* sequence of write operations on R. The network interface is used to propagate and receive messages.

```
<card stamp="ab1:273441", id = "ab1:27341", op = "create",
homeTel =  "01 23 45 67 89", workTel= "01 39 63 51 47", email = dedieu@tif.inria.fr, name = "Olivier Dedieu">
</card>
<card stamp="ab1:27357", id="ab1:27341", op="update",
     email="Olivier.Dedieu@inria.fr">
     </card>
```

Figure 2: A sequence of log records

5 Pharos Implementation

Our replication model and architecture is used as an underlying component of the Pharos system. This section presents Pharos in more details, in particular the way annotations and channels are implemented.

5.1 Annotations

An annotation is a structured datum, published by someone on a channel, to describe a Web document. The user publishes annotations explicitly. An impotant characteristic of an annotation is that it is structured. This data structure depends on the channel class. A basic class, named *BasicClass*, is provided for general purpose. A *BasicChannel* annotation contains a title which is by default the document's title, a *rating* which is the user's appreciation of the document, some *keywords* which are chosen in an extensible hierarchical list, and a *comment* which is a free textual note.

An annotation may contain multiple keywords. The keyword hierarchy may be viewed as a simple thesaurus. This thesaurus helps the community to share a common, tree-structured vocabulary when annotating documents. The thesaurus simplifies searching by keywords and gives a global view of the channel topic organisation. The keyword hierarchy may be modified only by authorized users.

a. Channels

A channel represents a community of users sharing a same interest. The number of users registered in a channel varies according to the community. A channel can be composed of a few users annotationg documents on a very in-depth topic or, at the opposite extreme, it can concern a more general topic intended for mass consumption on the world-wide scale. The channel stores member's annotations in a database. It can extract annotations associated with a specific document and annotated documents matching some annotation criteria. Channels are autonomous entities. That is, instead of having one huge database for all annotations of all topics, they are as many channels as there are topics.

b. Implementation

Pharos has been developped in Java which provides useful features for building dynamic and portable architectures. Channels are mainly composed of two parts: the backend and the frontend. The backend is the server part of a channel. It manages member subscription, the annotation database and additional data such as the thesaurus of keywords. The frontend is the client part of a channel. It contains GUI hosted in the browser assistant. It allows the user to add, display and query annotations, to edit the thesaurus and to customize his profile. The backend processes queries from frontend. The data replication architecture presented in Section 4 is implemented as an autonomous component placed together with the backend part of the channel, queries are submitted towards the annotation database, using the object manager.

Frontend/backend communication relies on RMI (Remote Method Invocation). Each backend exports a remote object which is invoked by frontends. To factorize certains resource consumption (e.g. network port, Web Access, etc), backends may me aggregated in a same machine (*Pharos Server*). This helps users to discover and choose the channels they want to subscribe.

Pharos handles a variety of channels thanks to a composition architecture. This architecture relies on the JP framework [5] which is intended for building modular and extensible Java applications. Channel frontends and backends are Jplug components which are respectively plugged into the browser assistant and the PharosServer. When a user subscribes to a new channel, the corresponding frontend is downloaded, installed and loaded.

Pharos's Web site [http://webtools.dyade.fr/pharos/] features more than twenty different channels. Pharos is also used by FTPress, a French news agency, and Wanadoo, the first Internet Service Provider in France, to allow for collaborative annotation and ranking of documents.

6 Conclusion

Pharos is a collaborative infrastructure which allows people to share their knowledge on a specific topic. People finding a valuable document put annotations in the appropriate channel. Channels are replicated to improve performance. This paper has several contributions. First, we proposed an optimistic replication model for asynchronous collaborative applications on the Web with interrelated replicas. Our model is based on lazy group replication. Second, we proposed a protocol to detect and resolve potential conflicts, and refresh the replicas. Log management plays an important role in our protocol since it is used to identify conflicts. In addition, timestamp ordering is used as the criteria to solve conflicts.

Third, we proposed a replication architecture that contains the most relevant components to implement lazy group replication. In our architecture we proposed Java as the underlying language for object management and we use XML for log management description. Java makes our components highly portable while XML makes our log records easy to exchange between sites.

Our model and architecture has been used in Pharos, and validated through existing Pharos applications. Our solution is novel in that it deals with interrelated replicas and provides a complete solution to conflict resolution. It is also lightweight in the sense that it does not need to be integrated within a DBMS.

Several works address collaborative systems . Marais and Bharat [7] uses both content and collaborative indexing. However, they use a centralised database which represents the *CommonKnowledge* of the community. As a result, they do not address replication issues as proposed in this paper. From a replication point of view, the closest work to ours is [10]. The authors propose an optimistic replication model with session guarantees as coherency criteria. In our case, we do not impose any consistency criteria because the underlying *timestamp* management is sufficient for consistency enforcement.

References

[1] P.A. Bernstein, E. Newcomer: *Principles of Transaction Processing*. Morgan Kaufmann, 1997.

[2] V. Bouthors, O. Dedieu: Pharos, a Collaborative Infrastructure for Web Knowledge Sharing. *European Conference on Digital Libraries (ECDL'99)*, Paris, Lecture Notes in Computer Science, Springer-Verlag, 1999.

[3] A. D. Birrel, R. Levin., R.M. Needham, M. D.Schroeder : "*Grapevine: an exercise in distributing computing*", Communications of the ACM (25) 4, April 1982.

[4] O. Dedieu: "*Réplication Optimiste pour les Applications Collaboratives Asynchrones*", Thèse de Doctorat, Systèmes Informatiques, Université de Marne la Vallée, 2000.

[5] O. Dedieu : "J*Plug, a framework to build modular applications*", http://webtools.dyade.fr/jplug.

[6] S. Lawrence , C. L. Giles: "*Searching the World Wide Web*", Science 280, 5360, 1998

[7] H. Marais, K. Bharat : Supporting cooperative and personal surfing with a desktop assistant. ACM Symposium on User Interface Software and Technology (UIST-97), ACM Press, pp. 129-138, NY, 1997.

[8] M.T. Özsu, P.Valduriez: *Principles of Distributed Database Systems"*, 2nd Edition, Prentice Hall, 1999.

[9] E. Pacitti, P. Minet, E. Simon: "Replica Consistency in Lazy Master Replicated Databases". *Distributed and Parallel Databases*, Kluwer Academic, accepted for publication, to appear.

[10] E. Pacitti, P. Minet, E. Simon: Fast Algorithms for Maintaining Replica Consistency in Lazy Master Replicated Databases. *Int. Conf. on Very Large Databases (VLDB'99)*, Edinburgh, 1999.

[11] E. Pacitti, E. Simon: Update Propagation Strategies to Improve Freshness in Lazy Master Replicated Databases. *The VLDB Journal*, 8(3-4) :305-318, 2000.

[12] E.Pacitti, E. Simon, R. Melo: Improving Data Freshness in Lazy Master Schemes. *IEEE Int. Conf. on Distributed Computing Systems (ICDCS'98)*, Amsterdam, 1998.

[13] E. Pacitti, P. Valduriez: Replicated Databases: concepts, architectures and techniques. *Networking and Information Systems Journal*, 1(4-5): 519-546, 1998.

[14] D. B. Terry, A. J. et. al. : Session Guarantees for Weakly Consistent Replicated Data. *Int. Conf. On Parallel and Distributed Information Systems (PDIS' 94*), Austin, Texas, 1994.