CVRetrieval: Separating Consistency Retrieval from Consistency Maintenance

Yijun Lu, Hong Jiang, and Ying Lu
Department of Computer Science and Engineering
University of Nebraska-Lincoln
{yijlu, jiang, ylu}@cse.unl.edu

Abstract

In distributed online collaboration applications, such as digital white board and online gaming, it is important to guarantee the consistency among participants' views to make collaboration meaningful. However, maintaining even a relaxed consistency in a distributed environment with a large number of geographically dispersed participants still involves formidable communication and management cost among them.

In this paper, we propose CVRetrieval (Consistency View Retrieval) to solve this scalability problem. Based on the observation that not all participants are equally active or engaged in distributed online collaboration applications, CVRetrieval differentiates the notions of consistency maintenance and consistency retrieval. Here, consistency maintenance implies a protocol that periodically communicates with all participants to maintain a certain consistency level; and consistency retrieval means that passive participants (those with little updating activity) explicitly request a consistent view from the system when the need arises in stead of joining the expensive consistency maintenance protocol all the time. The rationale is that, if a participant does not have updating activities, it is much more cost-effective to satisfy his or her needs on-demand.

The evaluation of CVRetrieval is done in two parts. First, we theoretically analyze the scalability of CVRetrieval and compare it to other consistency maintenance protocols. The analytical result shows that CVRetrieval can greatly reduce communication cost and hence make consistency control more scalable. Second, a prototype of CVRetrieval is developed and deployed on the Planet-Lab test-bed to evaluate its performance. The results show that the active participants experience a short response time at some expense of the passive participants that may encounter a longer response time depends on the system setting. Overall, the retrieval performance is still reasonably high.

1. Introduction

Consistency control among participants in distributed online collaboration applications has been an active research area [4, 8, 9, 12, 15, 17, 19, 20]. Recently, applying relaxed, but not overly loose, consistency control to achieve a more scalable system has become a mainstream method [20]. For example, in a distributed digital white board in which multiple participants draw on the same virtual white board, a perfect consistency (everyone sees exactly the same picture) is too costly to maintain and too slow to respond, and an unbounded optimistic consistency (its inconsistency level is unbounded) is too loose to facilitate the collaboration as it causes confusion (think about two users draw different figures on the same place at the white board). Thus, providing relaxed consistency with certain consistency level guarantee (for example, two users can have overlap of several lines but not the overlap of a whole figure) is commonly used in this scenario to strike a balance between scalability and accuracy.

While we agree with this kind of tradeoff, we believe that the current mode of consistency maintenance is still not efficient enough because most consistency control schemes in use today still rely on applying the same protocol on all participants, which could induce high communication overhead [4].

To address this limitation, we propose a new low-overhead, hence more scalable, consistency control architecture called *consistency retrieval* and differentiate it from the notion of *consistency maintenance*.

In this paper, *consistency maintenance* refers to the enforcement of consistency through communication

among all the participants. The maintenance cost grows with the number of participants and, in a truly large system, such as online gaming, the consistency maintenance cost can be formidable. There are a number of systems available to support consistency maintenance, such as our own related work IDEA [17] and others [4, 19].

A straightforward way to reduce the maintenance cost is to reduce the number of participants that a consistency maintenance module needs to include. We believe that this is both doable and preferable. First, this is doable because not all participants in a collaboration application are equally active or engaged. In one digital white board scenario where students listen to a lecture, for example, the lecturers are more likely to issue updates while a majority of the students are observers—they monitor the white board and rarely issue updates. From a consistency maintenance point of view, the lecturers are more important than passive students. So there is really no real need to consider the passive students group as far as consistency maintenance is concerned at most of the time. The rationale is that, if a participant does not have updating activities, it is far more cost-effective to satisfy his or her needs on-demand. Second, this is preferable because it does not change the way most current consistency control protocol work, and hence is easier to be adopted. In this paper, we refer to this on-demand-based mechanism as *consistency retrieval*.

We present *Consistent View Retrieval* (CVRetrieval) that supports the functions of consistency retrieval. To support the retrieval functions, CVRetrieval deploys publishers and subscribers in the system to serve as rendezvous points, similar to the publish-subscribe schemes [1]. CVRetrieval chooses publishers and subscribers based on applications' semantics to capture the common interest (with the consistent view of a particular application) among participants.

To evaluate CVRetrieval, we deploy IDEA, our previous developed adaptive consistency maintenance protocol, as the consistency maintenance component for CVRetrieval (for active participants) and the evaluation is done in two parts. First, we theoretically analyze the scalability of CVRetrieval and compare it to other consistency maintenance protocols. The analytical result shows that CVRetrieval can greatly reduce communication cost and hence make consistency control more scalable. Second, a prototype of CVRetrieval is developed and is deployed on the Planet-Lab test-bed for performance evaluattion. The results show that active participants in CVRetrieval have faster response times than pure consistency maintenance protocols at the slight expense of passive participants that can experience longer response times depends on the system setting, although the retrieval performance is still reasonably efficient for the latter. Thus, based on the evaluation results, we believe that CVRetrieval makes a sensible tradeoff.

The rest of the paper is organized as follows. Section 2 discusses the background of and challenges faced by CVRetrieval. Section 3 discribes the targeted applications. Section 4 outlines the IDEA infrastructure and detailed design of CVRetrieval is presented in Section 5. The scalability of CVRetrieval is evaluated analytically in Section 6 and its performance is evaluated through prototyping in Section 7. Then, Section 8 presents related work. Finally, Section 9 concludes this paper and discusses future work.

2. Background of and Challenges Faced by CVRetrieval

In the following discussion, we briefly discuss the definition of consistency we adopt, the IDEA infrastructure we use as a consistency maintenance protocol for active participants, and the challenge faced by CVRetrieval.

2.1. What is consistency and how to measure it?

Depending on the contexts, the meaning of consistency has many interpretations. In this paper, we deal with file or object consistency in the context of a replica-based distributed system. More specifically, we deal with the file consistency in a wide-area online sharing system in which the file/object (for simplicity and without the loss of generality, we talk about one file only) is replicated for both availability and fault tolerance.

A good example to think about this is a distributed digital white board. In this scenario, a group of participants communicate with one another via a white board, which is replicated on each participant's site. If we treat the white board as one single file (it is only a matter of granularity if the white board contains multiple files), then this file is replicated in each participant's site and we need to maintain its consistency by making sure all the participants have the same view, *i.e.* if an update appears in one site, it should appear in other sites too. Also, multiple updates should appear in the same order in different sites.

According the TACT [20], a pioneer research on defining metrics to quantify consistency degrees, there

are three metrics that can be used to evaluate and quantify consistency in this context:

- **Delay**. A remote update should appear on a participant's site with very little delay.
- **Order preservation**. Multiple updates should appear on all the sites in the same order.
- **Correctness**. The content of an update should be the same on all sites, including the site where it originated.

These three metrics, as shown in previous research, are able to capture a wide range of applications, including the distributed online collaboration applications that CVRetrieval targets.

2.2. IDEA

Because CVRetrieval uses IDEA [17] as the consistency maintenance module, we briefly discuss the IDEA infrastructure, our previously developed adaptive consistency maintenance protocol, as follows.

IDEA is a detection-based two-layer infrastructure that checks the consistency level of a file and resolves the inconsistency among all potential writers to that file. Further, IDEA divides the writers into two groups: active writers and passive writers. This differentiation is based on each writer's updating frequency: if its frequency is above a pre-defined threshold, it is classified as an active writer and is put into the top layer. Other writers are classified as passive writers and are put into the bottom layer. The rationale behind this distinction is that, by focusing primarily on active writers in the top layer where most inconsistencies arise, IDEA can detect and resolve inconsistencies more efficiently. Our previous study has shown that, under a variety of conditions, the two-layer infrastructure can capture most inconsistencies in the top layer with minimal delays [18].

Based on this efficient inconsistency detection mechanism, IDEA maintains consistency of a system in the following manner: (1) IDEA resolves inconsistency in the background periodically to improve the consistency level continuously; and/or (2) upon any participant's request, IDEA can actively resolve the inconsistency on demand. CVRetrieval uses IDEA to guarantee the consistency level of the retrieved view of a distributed online collaboration application.

2.3. Challenges faced by CVRetrieval

The main function of CVRetrieval is to retrieve a consistent view for passive participants on demand. Because the consistency level of a view is ultimately determined by the active writers who issue updates (we focus on active writers here because, comparing with casual writers, active ones cause much more inconsistencies), CVRetrieval needs to reach active writers in order to retrieve a consistent view. However, because we use IDEA, CVRetrieval only needs to reach any one of the active writers for the retrieval purpose due to its ability of resolving inconsistencies from any active writer, as explained previously. While this greatly simplifies the design of CVRetrieval, the locations of the retrieved objects (where the relevant active writers are) are not fixed and their number of the to-be-retrieved objects could be fairly large, as explained below.

- **The member of active writers is not fixed**. This is because the participants' updating patterns change from time to time. Current active writers may not be active writers a moment later. While history data can be collected, it is still not clear how they can help predict future active writers.
- **The number of active writers is potentially large when multiple files are considered**. While the number of active writers for a particular file is usually small, the number can quickly add up when we consider hundreds or even thousands of files in a truly large system. The challenge is to handle this large number of active writers without incurring high communication cost.

Only after solving these two challenges, can CVRetrieval build a mechanism to efficiently retrieve the updates for the passive participants.

3. Applications and Their Characterizations

We consider four representative distributed online collaboration applications, as summarized in Table 1 that lists some key application characteristics.

White board. In this application, participants appear online at the same time to collaborate. For example, a group of people can draw on the same white board to communicate with one another.

Online gaming. Online gaming has become more and more popular. Popular games, such as World of Warcraft [16] and SecondLife [13], have hundreds of

	White Board	Online Game	Bulletin Board	E-Business
Collaboration type	Synchronous	Synchronous	Asynchronous	Asynchronous
# of active writers	Small	Large	Small	Small
# of passive writers	Medium	Large	Large	Medium
# of observers	Large	Large	Large	Medium
# of shared objects	Small	Large	Small	Medium
Order preservation	Low	Low	High	High
Low latency	High	High	Low	Medium
Correctness	Low	Medium	High	Very high

Table 1: Four representative distributed online collaboration applications

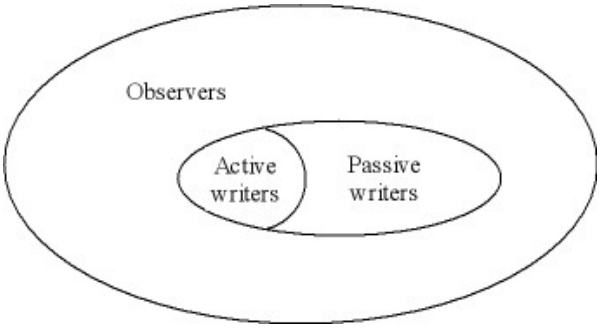

Figure 1: Three classes of participants

thousands of paid subscribers. While people usually think games as leisure activities, it has in fact become a common platform for people to interact. A recent *Business Week* cover story reported that people earn big real money from online gaming, such as SecondLife, and even traditional companies are entering the gaming arena for both advertisement and direct sale [7].

Bulletin board. People read the post in a bulletin board and post messages when they want to communicate with one another.

E-business. Think about an airline ticket booking system that is handled by multiple servers. The content of an update should be the same on all sites. In e-business applications, the writers are indeed the servers that handle transactions because they issue updates on behalf of the customers.

In Table 1, we characterize qualitatively the four applications based on eight metrics. First of all, white board and online gaming are synchronous collaborations because the participants usually appear online at the same time; bulletin board and e-business are asynchronous collaborations because the participants come and go: people post on the bulletin board and check back later; in e-business, each business center has different request pattern, thus issuing updates asynchronously [6].

The number of active writers for a given object is large in online gaming because, due to the usually large number of participants, even a small portion of them acting as active writers can result in a large number comparing with that of other three applications. The white board application has a small number of active writers because there are usually a small number of participants who lead the collaboration (think the lecturers in the remote education scenario). The other two applications have small numbers of active writers because, due to the nature of an asynchronous collaboration, not all writers are active at the same time. The numbers of passive writers in white board and e-business are medium comparing with that of online game and bulletin board for the following reasons. For white board, the total number of participants is usually small because it is intended for a not-so-large group of people to communicate, hence the number of passive writers. For e-business applications, as mentioned before, the servers that handle transactions are the writers. Thus, at least currently, the number of these servers is not large, so we conclude that the number of passive writers will be small too.

In terms of the number of observers, all, except for e-business, the other three have a large number of observers. In e-business, unlike the other three applications in which participants communicate with one another and a lot of people care about the perceived qualify of consistency, only stakeholders who can earn serious money from the e-business are interested in the consistency of the e-business application and that number is relatively small. Finally, the numbers of shared objects are large in online game as modern games become more and more sophisticated. White board and bulletin board have smaller numbers of shared objects than that of e-business because these two applications serve special features for sharing (*i.e.* dedicated white board and bulletin board) while it is

possible for an e-business application to run multiple types of transactions at the same time.

In terms of the consistency requirement, white board and online gaming have relatively low requirement of order preservation because people prefer fast responses and are usually willing to figure out the errors by themselves. For example, a recent study of online gaming shows that in chat room people prefer to receiving conversation word-by-word (faster speed) than receiving the finished sentence at once (slower speed) [3]. For the same reason, white board and online gaming require low latency than bulletin board and e-business. With the increased emphasis on accuracy—for example, an error in e-business can cost a lot of money for the company—the requirement of correctness increase from the left to the right side in Table 1.

Before we end this discussion, we need to mention that CVRetrieval can potentially benefit all these four applications for the following reasons. First, the white board, online gaming, and bulletin board applications all have a large number of observers, which makes reducing the consistency control overhead for these observers a significant reduction in consistency control overhead for the system as a whole. Second, while the absolute number of observers is small in e-business comparing to those of other three applications, it is still larger than, or at least comparable to the number of writers (both active and passive ones), which are essentially the e-business servers. Thus, CVRetrieval can also benefit an e-business application in that it can significantly reduce the consistency control overhead.

4. Architecture

CVRetrieval is designed to improve the efficiency of consistency control by providing a consistent view retrieval service for the observers of an application while letting an existing consistency maintenance protocol to maintain the consistency for writers. Theoretically, CVRetrieval can work with any consistency maintenance protocol. This is because, to use CVRetrieval, a consistency maintenance protocol only needs to differentiate observers and other active participants: the observers use CVRetrieval and the consistency maintenance protocol enforces consistency among the rest of the participants. Further, the consistency maintenance protocol has to define the entry point for CVRetrieval to retrieval the consistent view. In the case of IDEA, the active writers are the

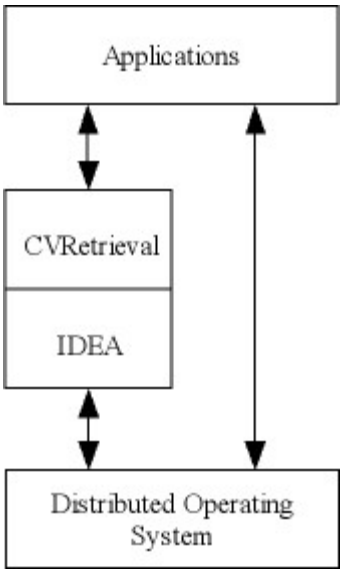

Figure 2. The Architecture of CVRetrieval

entry points. Once the entry point is defined, the information about it will be published by CVRetrieval.

In this paper, we choose IDEA, a detection-based consistency maintenance protocol that was developed by the authors previously and briefly described in Section 2.2, as the consistency maintenance protocol for CVRetrieval. Using IDEA as the consistency maintenance module, the architecture of CVRetrieval is depicted in Figure 2.

As shown in the figure, CVRetrieval is between the application layer and a general distributed operating system. When the application needs to guarantee consistency, it interacts with CVRetrieval. CVRetrieval depends on a consistency maintenance module—in this case, IDEA—to maintain consistency among writers and to guarantee the consistency level of the retrieved view. Finally, applications interact with the distributed operating system directly when no consistency issue is involved.

5. System Design

We try to address several design issues in this design section:

- How do participants join the system and how to map the participants to the IDEA infrastructure?
- How does IDEA communicate with the publishers so that the publishers have the updated information of the top layer nodes (that includes all active writers) for different object?
- How do ISPs subscribe on behalf of their clients?

- How does the publish-subscribe scheme work?

Throughout this section, we use a virtual white board application to make the discussion concrete.

5.1. A virtual white board scenario

We consider a distance education scenario in which several lecturers give lectures and a group of students join the discussions by manipulating a virtual white board (logically centralized and physically distributed on each participant's site). Other students who are not part of the discussion group will passively observe the discussion by watching the virtual white board.

In this scenario, the lecturers and the students in the discussion group conduct active discussions by issuing updates on the white board. Due to the nature of discussion, not all the members in the discussion group will speak up at the same time. During the discussion, membership of the active white-board-based speaker goup will change constantly, and such change is usually unpredictable because the spontaneity of an active discussion.

5.2. Participants join the system

We assume that there is a mechanism for participants to know the ID of the white board session and the time when the session starts. In practice, this can be done by some offline method, such as through an email list.

After all the participants log in, they form a group. Each participant modifies his or her own white board and those updates will show on others' white boards.

5.3. Mapping between participants and the IDEA infrastructure

As illustrated in Figure 1, we differentiate three types of participants: active writers, passive writers, and observers. They are mapped to IDEA as follows.

First of all, CVRetrieval differentiate observers from writers. When participants log in the white board application, they are required to indicate whether they are members of the discussion group. If yes, they are characterized as writers; if no, they are classified as observers.

Second, IDEA differentiates active writers from inactive writers after the system starts to run. IDEA tracks active writers (by its top layer) and passive writers (by the bottom layer) based on their updating frequency as explained in Section 2.2.

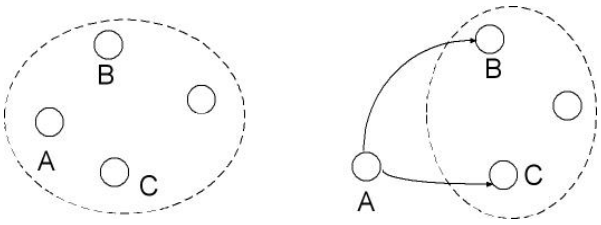

Figure 3. Use pointers to handle stale information

5.4. Communication between IDEA and publishers

In CVRetrieval, each object has a designated publisher, which is responsible for publishing the top layer nodes' information on behalf of the objects. There are two issues here: (1) how to map an object to a publisher? (2) how do publishers learn the top layer nodes' information from IDEA?

There are two ways to map an object to a publisher based on the total number of shared objects. If the number of shared objects is small in an application, such as in the white board application, the shared objects can be mapped to a single publisher. If the number of shared objects is large, such as in online gaming, certain mechanism is needed to balance multiple publishers' load. Hashing table based scheme (choose publishers based on the hashed value of the object IDs), such as DHT [10, 11, 14], is desirable for both its load balancing and its easy lookup (subscribers can find the right publishers by hashing the object IDs themselves).

The publishers learn the top layer nodes through communication with them. From the mapping procedure, the top layer nodes of an object know where their corresponding publisher is. The top layer nodes will communicate with their publisher whenever a node joins or leaves the top layer. The publisher will publish these updates to its subscribers subsequently.

However, this published information may become obsolete due to the propagation delay. For example, a subscriber could have old information (it states that *A* is in the top layer of object *f* but *A* is in fact no longer in the top layer anymore). We use pointers to solve this problem. In an example illustrated in Figure 3, we let *A* keep two pointers of its fellow members when it is in the top layer of object *f* (left half of Figure 3) and, when *A* is no longer in the top layer, it can at least forward the request to the other top layer nodes (*B* or *C* in this case, see the right half of Figure 3). Because it is very unlikely that all three nodes are leaving the top layer during the time of the propagation delay, this kind of old information will be transparent to users. In the case that this mechanism does not work, the

request can always be returned back to the subscriber, who can then pull updated information from the publisher (see Section 5.6).

5.5. ISPs subscribe on behalf of their clients

We use ISPs (Internet Service Providers), instead of the clients themselves, as the subscribers for two reasons. First, the ISPs, as the Internet entry point for its clients, are much more stable than its clients. Hence, using ISPs as the subscribers makes the publish-subscribe structure (*i.e.* the positions of publishers and subscribers) much more stable too. Second, while clients change their interests rather frequently, which—if we use clients as subscribers—causes frequent membership change for a publisher and the publisher that in turn needs to adjust its publishing scheme to reflect that change, ISPs' interests are relatively stable because their interests do not change with respect to how many and which clients are interested in an object, as long as some client is interested in that object.

When a client becomes interested in an object, it informs its ISP, which will subscribe the object's information if it hasn't done so. If the ISP has already subscribed for that object, it will just add the client into its client list and inform the client about all the future updates about that object's top layer nodes. When a client becomes uninterested in an object, it informs its ISP too. If, after this client's exit, the ISP has no client for that object, it will unsubscribe this object; otherwise, it simply deletes the client from its client list quietly.

In CVRetrieval, subscribers have two responsibilities. First, it informs a writer to periodically push new updates to it at a predefined rate and, when a new update arrives, immediately forwards the update to its clients. Second, when a client is in need of a consistent view immediately, the client can explicitly ask the subscriber to retrieve the view on its behalf. When a subscriber receives the retrieval request, it either returns a view from its cache (if it has one because other clients have just retrieved it before) or retrieves the view directly from the writer.

5.6. The publish-subscribe scheme

We use a multicast tree and filters to sent information from publishers to their subscribers. In this scheme, each publisher builds a multicast tree and an interior node forwards the packets further down the tree only if there are some nodes in its subtree that have subscribed it. While there are other publish-subscribe schemes available, such as shared tree or structure-less schemes, we choose the multicast tree structure because it provides a stable infrastructure that is suitable for our targeted applications that usually last a long period of time.

In the naïve form, the publisher sends the whole top layer information down the tree structure and all the subscribers will receive that information. To improve the system's scalability and efficiency, CVRetrieval incorporates the following optimizations.

First, a publisher in CVRetrieval only sends a subset of the list of the top layer nodes to each subscriber to preserve the network bandwidth. This raises two questions: how to choose a subset for a given subscriber and how to disseminate different subset of top-layer node information through a multicast tree (by definition, a multicast tree disseminates the *same* information to all the nodes)?

When choosing the subset, the publisher has several factors to consider. First, the active writers in the subset should be physically close to the subscribers so that the retrieval can be done efficiently. Second, one or two remote active writers can be included in each subset to provide redundancy because physically close machines tend to go down at the same time (for example, a power outrage). Third, the publisher needs to consider load balance so that no active writer is overwhelmed by retrieval requests.

Now we illustrate how to disseminate the different subsets via a multicast tree. First of all, the subscribers report their physical locations to the root in a bottom-up fashion and the messages are aggregated at each interior node. Second, the publisher chooses different subsets for its *immediate* children in the multicast tree based on these children's subtree's interests (i.e. the collective interest of the nodes in its children's subtree) and disseminate the subsets. For each interior node, it further divides the subset for its own *immediate* children. This process continues until the leave nodes are reached.

When a client explicitly retrieves a consistent view, as explained in Section 5.5, its subscriber will either return one view from its cache (if one exists) or pull a request from a writer. While rare, there is a possibility that the writer is no longer an active one and it has no way of reaching another one. In this case, the subscriber will need to contact its publisher for the most up-to-date information about the active writer list.

One key challenge is that the shared objects, unlike mp3 music files, are perishable—as time goes by, a perfectly consistent view can potentially become very inconsistent. Thus, directly sharing a previously retrieved view without the consideration of its

timeliness is pointless. In CVRetrieval, the sharing can be done with timeliness-conscious caching: when a subscriber receives a new consistent view, it caches the view after it has forwarded the view to the interested clients; later, when another client is asking for this view, the subscriber decides whether the cached view it received previously is still satisfactory for this new request by considering the time gap. In this way, participants can share a retrieved consistent view through local subscribers without putting much burden on the top layer nodes.

6. Scalability of CVRetrieval

In this section, we compare the communication cost of the CVRetrieval approach with other consistency maintenance approach. This analysis is crucial because the main hypothesis of CVRetrieval is that it can save communication cost, thus make the consistency control as a whole more scalable.

However, due to the long history of research on consistency maintenance, there exist a large number of consistency maintenance protocols that are suitable for various scenarios. This is a challenge to this analysis because the dynamism makes it hard, if not impossible, to compare CVRetrieval with all of them under one unified evaluation framework. To cope with this challenge, we simplify the analysis as follows without the loss of generality.

First, we classify the consistency maintenance protocols into four major categories by extending previous work in this area (categorizing consistency maintenance protocols for distributed collaboration systems) and compare CVRetrieval with each category.

Second, we realize that, to accurately compare communication cost of the protocols, we need to consider a group of factors, including the average communication message size, the traveling distance of each message, and the total number of messages. However, considering such details will make the analysis intractable because of dynamism in a large scale distributed system. Thus, in this analysis, we assume that all the protocols incur the same average message size and, on average, each message travels the same distance. Hence, the differentiator of the protocols is reduced to the total number of messages incurred by each protocol.

6.1. Categorization of consistency maintenance protocols

Our categorization follows the research work by Yang and Li [19], but extending their work in two aspects. First, it covers more recent research work, including our previously developed detection-based IDEA protocol. Second, this categorization focuses primarily on consistency maintenance protocols in the online distributed collaboration systems, which make it more focused than theirs. In this analysis, we consider four categories: locking, serialization, operational transformation, and detection-based consistency maintenance.

Locking. Locking mechanism controls consistency by locking a data object and only allowing one user to modify the data at a time. Depending on whether the mechanism allows users to continue their work while a lock is requested and released, locking can be further divided into three types: pessimistic (work is blocked in both lock requesting and releasing), semi-pessimistic (work is blocked only in lock releasing), and optimistic (work in not blocked in either case). While locking is widely used in small networks, we believe that it is not suitable for distributed online collaborations because users usually do not tolerate long delay caused by the locking operation.

Serialization. In this mechanism, all the users are allowed to modify their replicas, but their updates need to be serialized at a single point to maintain a consistency state. There are two flavors of serialization: pessimistic serialization and optimistic serialization. While users are not allowed to continue their work until their previous updates are serialized in the former, the latter allows users to continue their work and will rollback their inconsistent updates when needed. Because we consider a replica-based distributed system, we assume that this is a distributed serialization. An example of this model is Deno [4], a peer-to-peer voting protocol in which each writer's update travels across the whole replica group to detect and resolve any inconsistency. During Deno's serialization process, further updates are allowed, so this is an optimistic serialization. While the enforcement mechanism of a bounded inconsistency level by TACT [20] can be generally considered as an optimistic serialization because they let each server loose its consistency control to the degree that the total inconsistency across the server group is still within a predefined inconsistency bound, its enforcement mechanism is not a unified one. Rather, TACT developed a set of schemes to enforce the bound for different aspects of consistency. For this reason, the mechanism of TACT is not directly comparable to that of CVRetrieval because the latter targets at a unified consistency maintenance protocol.

Operational Transformation. This mechanism differs from optimistic serialization in how it reacts to inconsistencies. While optimistic serialization repairs

inconsistency when it arises by rolling back inconsistent updates, operational transformation does not undo the effects to reduce overhead. This operation is useful when the inconsistency is either not repairable or it is insignificant. Essentially, this is an extreme optimistic operation and, because it does not guarantee any level of consistency, we believe that it is not suitable for distributed online collaborations in which unbounded inconsistency can cause confusion and make meaningful collaboration impossible.

Detection-based scheme. We previously presented IDEA as the first detection-based consistency maintenance protocol for large-scale distributed systems. Instead of enforcing a fix consistency protocol beforehand, IDEA detects inconsistencies when they arise and resolve them based on the applications' ongoing need for consistency. IDEA achieves adaptability for the applications and supports flexibility for the end users to adjust their consistency level on the fly. IDEA is suitable for distributed online collaboration because it allows the users control to adjust the perceived consistency level.

Among the above four categories, locking and operational transformation are not comparable to CVRetrieval since the former causes long delay and the latter does not guarantee consistency level at all. Thus a meaningful comparison will be among the optimistic serialization (we use Deno as a representative protocol), IDEA, and CVRetrieval with the goal of determining whether the added communication overhead of CVRetrieval is much smaller than the communication cost it saves by decreasing the communication cost of consistency maintenance.

6.2. Assumptions

To analytically evaluate the communication cost savings by CVRetrieval, we make the following assumptions and definitions.

[1] c: the average number of simultaneous writers.
[2] n: the total number of nodes in the system that join the consistency control process.
[3] n_1: number of writers.
[4] n_{hot}: number of hot writers among the n_1 writers.
[5] f_1: number of updates of hot writers during a given period of time t.
[6] n_{pass}: number of passive writers among the n_1 writers, where $n_{hot} + n_{pass} = n_1$.
[7] f_2: number of updates of passive writers during a given period of time t.
[8] n_2: number of observers, where $n_2 = n - n_1$.
[9] p: total number of publishers in CVRetrieval.
[10] s: total number of subscribers in CVRetrieval.
[11] q_1: number of publishings during a given period of time t.
[12] q_2: number of retrievals during a given period of time t.
[13] C_deno: total number of messages exchanged in Deno.
[14] C_idea: total number of messages exchanged in IDEA.
[15] C_r: number of messages exchanged in CVRetrieval.

As shown above, we use the number of messages exchanged as a metric to analyze the communication cost saving by CVRetrieval.

6.3. The Analysis

We conduct the analysis in three steps. First, we derive the communication cost associated with the consistency maintenance protocol Deno, followed by the derivation of communication cost of IDEA. Second, we derive the communication cost associated with CVRetrieval. Finally, we compare the three mechanisms. In this analysis, we consider the consistency control for one single object because this simplifies the analysis and, based on its result, it is easy to extend the analysis to multiple objects.

6.3.1. Communication cost of Deno

In Deno, each update travels the whole group and, when it meets another conflicting update, the update will be resolved at that time. In this analysis, each time an update reaches a node, we consider it as a new message because the node that is reached essentially regenerates the original message by relaying it. Thus, given an update, it only stops traversal when it meets another conflicting update. From the assumption 1, we know that there are c conflicting updates in the system at one time on average. For simplicity, we further assume that the updates propagate along a linear structure (without this assumption, the updating process becomes intractable). Then, on average, an update travels $1/c$ of the network to meet a conflicting update and stops.

While observers do not issue updates, they are certainly able to keep the updates when they pass by. There is no promise that those kept updates are a complete set of all the updates for a particular object. Thus it is reasonable to assume that there must be some mechanism for an observer to retrieve a

consistent view of this object. Unfortunately, this mechanism is not defined in Deno, not least because it is intended to be used in a server cluster environment where the observer group is not a concern. To make its comparison with IDEA and CVRetrieval both fair and complete, we make two assumptions. First, we consider that each observer has to connect to some other node to request for a consistent view and the requested node has to send the view back, which results in two messages for each request. Second, when a node receives a request, it is able to use some mechanism to find a consistent view if there is no such one available. However, we choose to omit this overhead in our analysis because: (1) Deno has not defined such a mechanism and it is unfair to assume a random one for it; and (2) this overhead is rather a one time overhead as it can be cached for a while for later use, which make it an insignificant part of the total communication cost. Since each observer will issue q_2 requests in time t, the total communication overhead is $2*n_2*q_2$.

Now we calculate the communication cost as follows. Because there are n nodes in the system, each update needs to travel n/c hops, which equals to n/c messages in total. In a given period of time t, there are $n_{hot}*f_1 + n_{pass}*f_2$ updates, so the total number of messages generated in a given period of time t is:

$$C\_deno = \frac{n}{c} \times (n_{hot} \times f_1 + n_{pass} \times f_2) + 2 \times n_2 \times q_2 \quad (1)$$

6.3.2. Communication cost of IDEA

In IDEA, the updates from hot writers will be detected among the hot writers and those from the passive writers will need to go through the whole network to be detected.

Similarly to the analysis in Deno, we assume the existence of c concurrent conflicting updates at one time. However, in the case of IDEA, the updates from hot writers stay at the top layer, implying that the hot writers actually see less than c concurrent updates because the updates from passive writers won't show up in the top layer at the same time. So, while passive writers still see c concurrent updates, we assume that the active writers sees only c_{hot} concurrent updates, where $c_{hot} < c$. Then an update from a hot writer will generate n_{hot}/c_{hot} messages, and that from a passive writer will generate n/c messages. There are $n_{hot}*f_1$ updates from hot writers and $n_{pass}*f_2$ updates from passive writers in a given period of time t.

For the communication cost associated with observers, we follow the calculation used in the Deno case and conclude that the overhead is two messages (one for request, one for reply) for each retrieval-type request. Then, because we have assumed that, on average, each observer will issue q_2 requests in time t, the total communication overhead is $2*n_2*q_2$.

Putting the communication cost of writers and observers together, the communication cost of IDEA is:

$$C\_idea = \frac{n_{hot}}{c_{hot}} \times n_{hot} \times f_1 + \frac{n}{c} \times n_{pass} \times f_2 + 2 \times n_2 \times q_2 \quad (2)$$

6.3.3. Communication cost of CVRetrieval

The communication cost of CVRetrieval involves three parts: (1) the detection of inconsistency among hot and passive writers; (2) the cost associated with the publish-subscribe scheme, which includes the communication cost between writers and publishers, between publisher and subscriber, and between subscribers and their clients; and (3) the retrieval operation for observers.

First, CVRetrieval detects inconsistency among hot writers in the same manner with that of IDEA because it depends on IDEA to maintain consistency. Thus the communication cost incurred by hot writers is $(n_{hot}/c)*n_{hot}*f_1$. For passive writers, however, they need not to go through the whole network; instead, they only need to detect among the writers' group (with n_1 writers) that excludes the observers. Thus, the communication cost associated with the updates from passive writers is $(n_1/c)*n_{pass}*f_2$.

Second, for the communication cost associated with publish-subscribe scheme, we first derive the cost for one round of publish and then multiply it by the publish rate q_1 to get the total communication cost in a given period of time t. Because a hot writer only notifies its publisher when it becomes a hot writer and when it becomes a passive writer. Here we conservatively assume that, in one round of publish, half of the hot writers are new ones (this is indeed a very extreme scenario because we essentially assume 50% of the hot writers leave the group and the same number of new hot writers join the group). Thus, in one round of publish, there are n_{hot} messages exchanged between writers and publishers because each old hot writer or new hot writer needs to inform exactly one publisher.

Sets	n	n_1	n_{hot}	c	c_{hot}	f_1	f_2	q_1	q_2	s	Deno	IDEA	CVRetrieval
1	1000	50	10	4	3	5	3	2	5	19	52000	39667	13125
2	1000	100	20	4	3	5	3	2	5	18	94000	69667	17543
3	1000	200	50	4	3	5	3	2	5	16	183000	124667	36399

Table 2: Analytical Results

Then, there are *s* messages exchanged between publisher and subscribers because there are *s* subscribers in total and each needs to be informed exactly once. Finally, let's conservatively assume that all the n_2 observers will need to be informed about its subscription. Then we know that n_2 messages are exchanged in one round. Adding the three parts of cost together and then multiplying the publishing frequency, we get the total communication cost associated with the publish-subscribe scheme in time *t* is $q_1*(n_{hot}+s+n_2)$.

Third, each observer will retrieve a consistent view for the object he or she is interested in, which results in n_2 retrievals. Because each retrieval consists of two messages (one request, one reply), there are $2*n_2$ messages exchanged in one retrieval operation. Finally, because we assume that each observer retrieve q_2 consistent views in time *t*, the total number of message exchanged in *t* is $2*q_2*n_2$.

So the total communication cost in a given period of time *t*, incorporating all three parts, is:

$$C\_r = \frac{n_{hot}}{c_{hot}} \times n_{hot} \times f_1 + \frac{n_1}{c} \times n_{pass} \times f_2 + q_1 \times (n_{hot} + s + n_2) + 2 \times q_2 \times n_2 \quad (3)$$

Note that parameter *s* is related to n_2 because there are s subscribers serving the n_2 clients (recall that each observer subscribes *k* objects). Although there is no ground rule about how many clients a subscriber should have, it is intuitive that the number of clients should not overwhelm the subscribers. Considering that the information that is being published is rather small in quantity (it is only a list of active writers and the message is maybe only a few KBs), we believe that each subscriber can support at least up to 50 clients, which incurs less than 1MB data traffic and should not be a burden for a subscriber. Thus, in the following analysis, we use $n_2/50$ as the value for *s*.

Further, the value of q_1 is associated with how frequent the active writer group changes and q_2 is associated with the observers' interests. Because CVRetrieval deals with loosely coupled distributed online collaboration applications, we believe that, in a short period time of *t*, it is sufficient to assign a small numerical value for q_1. For q_2, we believe that it should be reasonably large so that it can satisfy observers' need of consistent view. However, q_2 cannot be too large, which implies smaller inter-retrieval time, because there is no point of issuing the second retrieval before response of the first request has arrived. Thus, we believe that it should be reasonable to make q_2 two to three times as large as q_1.

6.3.4. The comparison

We now can compare *C_deno*, *C_idea*, and *C_r* by assigning real numbers to the parameters in their respective expressions. In particular, we set $s = n_2/50$ and assign 2 and 5 to q_1 and q_2, respectively. We also set c_{hot} as $3*c/4$, which is actually quite conservative and put IDEA and CVRetrieval in disadvantage considering that most updates should come from hot writers. The analytical results are summarized in Table 2.

As shown in Table 2, CVRetrieval incurs much lower communication cost than pure consistency maintenance protocols in all three sets of data. This observation indicates that the majority overhead of CVRetrieval comes from the consistency maintenance of writers, which validates our hypothesis that, by separating observers from writers, the consistency control overhead can be substantially reduced.

Additionally, the overhead of CVRetrieval increases in a slightly slower speed than those of Deno and IDEA when the number of updates increases (reflected by the number of active writers). Comparing the results of set 1 and set 3 and we can see that the overhead of CVRetrieval in set 3 is 2.8 times as large as that in set 1, while that ratio is 3.5 for Deno and 3.1 for IDEA. We believe that this is an indication that CVRetrieval scales better than the other methods.

7. Experimental Results

We implement a prototype of CVRetrieval on top of the Planet-lab and use it to evaluate the performance of CVRetrieval. More specifically, we want to evaluate the response time of CVRetrieval in comparison with other consistency maintenance protocols. This measurement is important because it determines how fast an end user can perceive a certain level of consistency and naturally the faster the response time, the higher the user's satisfaction.

The response time is defined as follows. For a consistency maintenance protocol, it is defined as the time difference between the point when an update of an object is first committed and that when a participant receives that update (with a certain level of consistency guarantee). In the case of CVRetrieval, however, the response time has different definition for writers and observers. For writers, the definition of response time is the same as that in a consistency maintenance protocol. For observers in CVRetreivals, however, the response time is between the point of time when an observer issues a retrieval request for a consistent view of an object and that when it receives the view.

Our hypothesis is that, on the one hand, the response time for writers will be smaller in CVRetrieval than that in other consistency maintenance protocols because CVRetrieval has a smaller writers group. On the other hand, the response time for observers could be longer than that for writers in other consistency protocols because of the added publisher-subscribe scheme and that it only receives updates periodically or on demand.

7.1. Experiment setup

We emulate a white board application for evaluation purposes. The application is emulated by following its operational sequences. More specifically, we abstract the distributed white board as a set of objects that are replicated on each participating node. Then, we treat each update on the white board (from the writers group) as a write operation on its local replica. After updates are issued, IDEA works to maintain the overall consistency level of the virtual white board above a certain degree. Because our purpose of the experiments is to evaluate the consistency control, we assume that these updates are all conflicting with one another (otherwise, users need not to care about them).

Type	Max (seconds)	Min (seconds)	Average (seconds)
active writer	1.73	1.41	1.59
passive writer	11.8	10.2	10.98

Table 3: Response time for writers

In the current setting, each writer informs its publisher when it becomes or ceases to be a hot writer. The publisher then informs its subscribers (the ISPs who subscribe on behalf of their clients) periodically. Through the publish/subscribe infrastructure, subscribers get the list of the hot writer group, which, with a very high probability, have the most consistent view of the shared application. For observers, they specify their interest and inform their subscribers about that. The subscribers, based on the information received from publishers (the hot writer list), choose a nearby hot writer as the source for retrieval purposes. When an observer is not satisfied with the retrieved view, it can issue a "retrieval" request directly to the subscriber and the subscriber will then retrieve the most recent consistent view from the hot writers on behalf of the observer.

We conduct the experiment on the Planet-Lab test-bed. In the current setting, there are ten writers among which four are active writers and the other six are passive ones. There are one publisher and four subscribers. Each subscriber serves three observers. In other words, this is a 22-nodes system, excluding publisher and subscribers.

At the beginning of the experiment, each hot writer issues one update every 5 seconds until the experiment ends. These updates got disseminated among active writers immediately and, once it starts to propagate to passive writers, each hop will only disseminate the updates once every 5 seconds (to save bandwidth by combining multiple updates). We let each observers retrieve the consistent view every 20 seconds during the experiment. The experiment runs 300 seconds.

We also implemented a Deno-like protocol for comparison. In the Deno-like protocol, we organize the 22 participants (here, we don't consider the publisher and subscribers as participants because they are only facilitating CVRetrieval) in a linear fashion in which the updates are propagated from one to the other. To make the results comparable, we assume the same updating patterns for the ten writers.

7.2. Response time for writers

We measure response times for active writers and that for passive writers. The experiment was run ten times and the average response time, as well as maximum and minimum values, are measured and shown in Table 3.

From the result, we can see that the response time of active writers is very small. This is because the dissemination of updates is instant among active writers. While it is usually very costly to disseminate update instantly among participants, CVRetrieval can afford to do so because, via classification, there are only a relatively small number of active writers in existence.

As shown here, the average delay for passive writers is over 10 seconds, which looks rather high. However, this is because we set a five-second delay between the dissemination of updates among passive writers. In practice, system administrators can choose a shorter delay to improve the response time for passive writers at the expense of increased bandwidth overhead.

7.3. Response time for observers

There are two aspects of response time for observers. First, the time that it takes for them to receive the periodically published updates. Because this part of delay primarily depends on the publishing rate, we do not measure it here. Second, the response time for an explicit retrieval operation, *i.e.* when the observers actively retrieve the most updated view from the subscribers, the time it takes to get the view.

The delay of explicit retrieval depends on whether the observer can find the view in its subscriber's local cache (because another observer retrieved the same view a moment ago). Intuitively, the more retrievals can be satisfied with the subscriber's cache (a higher cache hit rate), the smaller the response time is. In this experiment, we give three settings of the cache hit ratio: 50%, 66.7%, and 75%. For each setting, we run ten experiments and the results are summarized in Table 4.

The result shows that the retrieval process is indeed very efficient and this efficiency increases with cache hit rate in subscribers.

7.4. Comparison to consistency maintenance protocols

We now compare the performance of CVRetrieval with a pure consistency maintenance protocol. For a pure consistency maintenance protocol, we assume that all participants are treated equal. In terms of updates dissemination, there are two types: active ones that disseminate a received update to other participants as soon as it arrives and passive ones that only periodically disseminate all the updates it received so far. Because the passive ones work similarly to the way CVRetrieva/IDEA treats passive writers, but with more participants, it is doubtless that CVRetrieval/IDEA will have a better performance. For this reason, we only experimentally compare CVRetreival to the active ones.

The consistency maintenance protocol we considered here has all the 22 participants (not including the publisher and subscribers because they are add-on features of CVRetrieval) we used in the CVRetrieval evaluation. Because this protocol actively disseminates updates, each participant relays a received update as soon as it is received. Finally, the writers have the same updating patterns as in previous experiments. We run this experiment ten times and the results are shown in Table 5.

Cache hit rate	Max (seconds)	Min (seconds)	Average (seconds)
50%	0.48	0.33	0.37
66.7%	0.3	0.24	0.28
75%	0.16	0.12	0.14

Table 4: Response time for observers

Max (seconds)	Min (seconds)	Average (seconds)
2.45	1.77	2.07

Table 5: Response time of a pure consistency maintenance protocol with active update dissemination

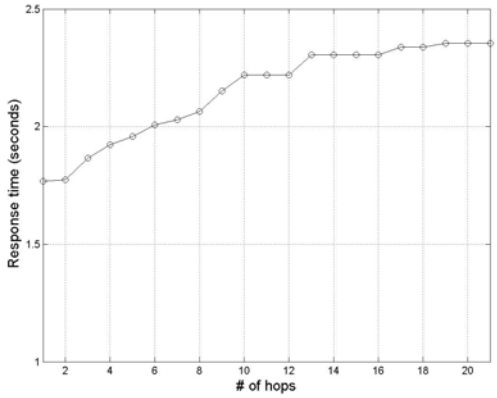

Figure 4: Response time for different hops

From this table, we can see that the response time of the pure maintenance protocol is larger than that of CVRetrieval's active writers (comparing to the data in Table 3). However, the absolute value of the response time is not that large. We suspect that is because, due to the heavy load of planet-lab nodes, the write operation alone needs too much time to be committed. To validate our hypothesis, we profile one run of the experiment and record the response time for all 21 participants (this does not include the writer who committed this update) and the result is depicted in Figure 4.

From this figure, we can clearly see that the first hop delay dominates the system's response time. With greater computing power that can minimize the cost of committing updating operations, we expect the advantage of the CVRetrieval approach to be much more obvious.

It is worth noting that most current protocols uses passive update dissemination method, with which the advantage of CVRetrieval will become more pronounced. Furthermore, the most important advantage of CVRetrieval is its saving of communication cost, especially in a system with a large number of participants, as analyzed in Section 6. We believe that the two features—efficiency and scalability—together make CVRetrieval a viable alternative to pure consistency maintenance protocols.

8. Related Work

Most collaboration applications nowadays originate from single-user applications. For example, MS Word was previously used by a single user to edit his or her file and then is modified to incorporate collaboration capabilities. A straightforward way to share these applications is to place a central control for consistency maintenance. In MS NetMeeting, for example, only one participant can operate on the shared object; all other participants will be blocked [2].

To prevent blocking, which causes access delay, the granularity of sharing is always adjusted to make the sharing unit small enough to prevent blocking to some extent. However, this approach is inherently not scalable for two reasons. First, for any given system, the granularity cannot be spited indefinitely. Second, it is still a centralized system and, in the presence of a hot unit, the blocking cannot be prevented and that makes it not suitable for a large-scale system with a large number of participants.

Newly developed distributed online collaboration applications use replication-based scheme to improve scalability and availability. However, as all the replicas have a copy of the collaboration application, inconsistency level among them hence is relaxed [9, 12]. While this scheme works well in many applications and helps distributed collaboration applications scale to large-scale distributed networks, relaxed consistency does not guarantee the QoS.

Recently, researchers have been trying to achieve relaxed inconsistency for distributed online collaboration applications. Yu and Vahdat defined metrics to evaluat consistency level for a wide range of applications [20]. Chang et. al. derived an algorithm to support different consistency level for different users in an online conference application [5]. Also, Local-lag and Timewarp were developed by Vogel and Mauve to eliminate short term inconsistencies and repair inconsistency, thus prevent unbounded inconsistencies [15]. A more recent work extended Vogel and Mauve's work by considering the same problem in a larger network [8]. However, these works are still use consistency maintenance for all participants, which cause high overhead for a system with a large number of participants.

CVRetrieval differs from previous work because it considers the consistency *retrieval* aspect, not just consistency maintenance, in distributed online collaboration applications. To the best of our knowledge, CVRetrieval is the first work to explicitly consider the retrieval aspect of consistency control in distributed online collaboration applications.

9. Conclusions and Future Work

In this paper, we presented the design, analysis, implementation, and evaluation of CVRetrieval, a system that improves the scalability of consistency control in large-scale distributed online collaboration applications by separating consistency retrieval from consistency maintenance.

CVRetrieval is fully evaluated by both analytical modeling and prototyping measurement. The analysis result showed that, comparing to pure consistency maintenance protocols, CVRetrieval incurs significantly less communication overhead and hence improves the scalability of consistency control in general. Through prototyping on the Planet-Lab test-bed, we evaluated the response time of CVRetrieval and the results showed that CVRetrieval achieves a sensible tradeoff: it achieves shorter response times for writers at the expense of a longer response time

for observers and, more importantly, improves the system's scalability as a whole.

In the future, we plan to improve the scalability and performance of CVRetrieval further by exploring more ways to optimize it. For example, we can drive active writer information towards the most needed subscribers by controlling the publish rate along different paths. For example, the subscribers (the ISPs) report their interests (in terms of frequency of issued requests) to the publisher, which in turn adjusts the publishing rate by publishing at a higher rate to a path that can reach subscribers that reveals higher interest than others.

References

[1] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, and D. C. Sturman, An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems, In International Conference on Distributed Computing Systems 1999

[2] J. Begole, M. B. Rosson, and C. A. Shaffer, Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application-Sharing Systems, ACM Trans. On Computer-Human Interaction, Vol, 6, No. 2, June 1999. pp. 95-132.

[3] B. Brown and M. Bell, 'There' as a collaborative virtual environment, in ACM conf. on Computer Supported Cooperative Work (CSCW 2004). Chicago, Illinois, Nov. 2004.

[4] U. Cetintemel, P. J. Keleher, B. Bhattacharjee, and M. J. Franklin, Deno: A Decentralized, Peer-to-Peer Object-Replication System for Weakly-Connected Environments, IEEE Transactions on Computers, 52(7), 2003

[5] T. Chang, G. Popsecu, and C. Codella, Scalable and Efficient Update Dissemination for Interactive Distributed Applications, in Proc. ICDCS 2002, Viena, Austria, July, 2002.

[6] H. Chandler, The Complexity of Online Groups: A Case Study of Asynchronous Distributed Collaborations, ACM Journal of Computer Documentation, 2001, 25, (1): 17-24.

[7] Robert D. Hof, My Virtual Life, Business Week, May 1st, 2006.

[8] F. Li, L. Li, and R. Lau, Supporting Continuous Consistency in Multiplayer Online Games, ACM Multimedia 2004, Oct. 2004, New York, New York, USA. pp. 388-391.

[9] A. Prakash and H. S. Shim, DistView: Support for Building Efficient Collaborative Applications using Replicated Objects, ACM CSCW 94, Chapel Hill, NC, USA. pp. 153-164.

[10] Sylvia Ratnasamy, Paul Francis Mark Handley, Richard Karp, and Scott Shenker. A Content Addressable Network. In Proceedings of SIGCOMM 2001.

[11] A. Rowstron and P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.

[12] C. Schuckmann, L. Kirchner, J. Schummer, and J. M. Haake, Designing Object-oriented Synchronous Groupware with COAST, ACM CSCW 96, Cambridge, MA, USA. pp. 30-38.

[13] Second Life, http://secondlife.com/

[14] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM 2001, San Deigo, CA, August 2001, pp. 149-160.

[15] J. Vogel and M. Mauve, Consistency Control for Distributed Interactive Media, ACM Multimedia 2001, 2001, Ottawa, Canada. Pp. 221-230.

[16] World of Warcraft, http://www.worldofwarcraft.com/

[17] Y. Lu, Y. Lu, and H. Jiang, IDEA: An Infrastructure of Detection-based Adaptive Consistency Control, Technical report, *TR-UNL-CSE-2007-0001*, University of Nebraska-Lincoln, January, 2007.

[18] Y. Lu, X. Li, and H. Jiang, IDF: an Inconsistency Detection Framework – Performance Modeling and Guide to Its Design, Technical report, *TR-UNL-CSE-2006-0003*, University of Nebraska-Lincoln, March, 2006.

[19] Y. Yang and D. Li, Separating Data and Control: Support for Adaptable Consistency Protocols in Collaborative Systems, ACM CSCW 2004, Chicago, Illinois, Nov. 2004, pp. 11-20.

[20] H. Yu and A. Vahdat, Design and Evaluation of a Continuous Consistency Model for Replicated Services, In. Proc. OSDI 2000.