Providing Quantitative Scalability Improvement of Consistency Control for Large-Scale, Replication-Based Grid Systems

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

Abstract

Consistency control is important in replication-based Grid systems because it provides QoS guarantee. However, conventional consistency control mechanisms incur high communication overhead and are ill suited for large-scale dynamic Grid systems. In this paper, we propose CVRetrieval (Consistency View Retrieval) to provide quantitative scalability improvement of consistency control for large-scale, replication-based Grid systems.

Based on the observation that not all participants are equally active or engaged in distributed online collaboration, 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 explicitly request consistent views from the system when the need arises in stead of joining the expensive consistency maintenance protocol all the time. The rationale is that it is much more cost-effective to satisfy a passive participant's need on-demand.

The evaluation of CVRetrieval is done in two parts. First, we analyze its scalability and the result shows that CVRetrieval can greatly reduce communication cost and hence make consistency control more scalable. Second, a prototype of CVRetrieval is deployed on the Planet-Lab test-bed and the results show that the active participants experience a short response time at expense of the passive participants that may encounter a longer response time.

1. Introduction

A popular strategy to improve the availability of shared data in large-scale Grid systems is to replicate data on geographically dispersed nodes. In this way, participants can fetch the data from a nearby copy with improved availability and response time. After retrieving a copy to the local node, the local copy becomes a new replica of the data and can be used to serve other nodes' need. In this type of replication-based systems, it is important to guarantee the consistency among participants' copies of the same data to make collaboration meaningful. In that sense, improved consistency among participants can improve participants' perceived Quality of Service (QoS) of the application.

There are two obstacles facing the design of a highly scalable consistency control mechanism for large-scale, replication-based Grid systems. First, large-scale Grid systems have a large number of nodes that are often geographically dispersed globally. Due to the network congestions and the inability to control remote nodes, maintaining even a relaxed consistency in such systems involves formidable communication and management cost.

Second, large-scale Grid systems are often dynamic. *I.e.*, nodes could join or leave the system at their will. With such dynamism, both the group of replicas and that of the nodes are interested in getting a replica keeps changing. Thus, any static—in the sense that the protocol fixed with certain replicas—is not suitable.

Current consistency maintenance mechanisms rely either on applying the same protocol on all participants or is based on the assumption that the replica group does not change. The former scheme is not scalable in large-scale Grid systems because it induces high communication overhead in the presence of a large number of participants (Cetintemel 2003). The dynamic nature of large-scale Grid systems means the latter scheme is not suitable as well.

In this paper, we propose a new low-overhead, hence more scalable, consistency control architecture to address this limitation, thus improving the QoS from the consistency control's point of view. This architecture is *consistency retrieval*. We also present the design, implementation, and evaluation of *Consistent View Retrieval* (CVRetrieval), a system that supports the consistency retrieval functionalities.

1.1. Consistency retrieval *vs.* consistency maintenance

Consistency retrieval is in contrary to 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. In a truly large system, the consistency maintenance cost can be formidable.

Consistency retrieval reduces maintenance cost by reducing the number of participants that a consistency maintenance module needs to include. This approach is both doable and preferable in practice. This is doable because not all participants in a collaboration application are equally active or engaged. In a 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 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 intensive updating activities, it is far more cost-effective to satisfy his or her needs on-demand. This approach is also preferable because it does not change the way most current consistency control protocol work. Thus it is easier to be adopted. In this paper, we refer to this on-demand-based consistency control mechanism as *consistency retrieval*.

It is noteworthy that, while it is easy to statically separate passive participants from active participants and only maintain consistency for active participants, CVRetrieval is significantly different from such an approach in two aspects. First, CVRetrieval is not merely differentiating active and passive participants once and staying with a fixed differentiation permanently. Instead, differentiation in CVRetrieval is a dynamic one, meaning that the active and passive participants are relative concepts and can change from time to time. The ability to capture this dynamics is a salient feature that sets CVRetrieval apart from any static approaches. Second, CVRetrieval assumes that passive participants do occasionally care about consistency, instead of assuming that they are not interested in the shared data[1] at all.

[1] We use the term "share data" and "shared object" interchangeably depending on the scenarios.

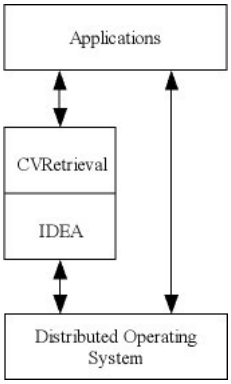


Figure 1. The Relationship between CVRetrieval and IDEA

1.2. CVRetrieval

CVRetrieval is a system that supports the consistency retrieval functionalities. CVRetrieval is built on top of IDEA (Lu 2007b; Lu 2008), an efficient consistency maintenance protocol proposed by the authors, as the consistency maintenance module. The relationship between CVRetrieval and IDEA is illustrated in Figure 1.

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 results show 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 (Peterson 2003) for performance evaluation. The results show that active participants in CVRetrieval have faster response times than in pure consistency maintenance protocols at the slight expense of passive participants that can experience longer response times depending on the system setting, although the retrieval performance is still reasonably efficient for the latter.

The rest of the paper is organized as follows. Section 2 discusses background and related work. Section 3 discussed the design issues of CVRetrieval. The design of CVRetrieval is then presented in Section 4. Section 5 analyzes the scalability improvement of CVRetrieval. Section 6 experimentally evaluates CVRetrieval based on a prototype deployed on Planet-Lab. Section 7 discusses future trends in consistency control research for large-scale Grid systems. Finally, Section 8 concludes this paper and discusses future work.

2. Background

Improving the scalability of consistency control has been a major research topic in distributed collaboration applications.

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 is then 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 (Begole 1999).

To prevent blocking, which causes access delay, the granularity of sharing is often 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 an active 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. As all the replicas have a copy of the collaboration application, inconsistency level among them hence is relaxed (Prakash 1994; Schuckmann 1996). While this scheme works well in many applications and helps distributed collaboration applications scale to large-scale distributed networks, relaxed consistency does not provide QoS guarantee.

Recently, researchers have been trying to achieve relaxed yet bounded inconsistency for distributed online collaboration applications. Yu and Vahdat defined metrics to evaluate consistency level for a wide range of applications (Yu 2000). Chang *et. al.* derived an algorithm to support different consistency level for different users in an online conference application (Chang 2002). Also, Local-lag and Timewarp were developed by Vogel and Mauve to eliminate short term inconsistencies and repair inconsistency, thus prevent unbounded inconsistencies (Vogel 2001). A more recent work extended Vogel and Mauve's work by considering the same problem in a larger network (Li 2004). 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 in the sense that it considers the consistency *retrieval* aspect, not just consistency maintenance. 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.

3. Design Issues

CVRetrieval has two design issues. First, we need to differentiate different roles of CVRetrieval and the conventional consistency maintenance protocol. Second, we need to define a procedure for CVRetrieval to satisfy passive participants' consistency needs on demand.

3.1. The roles of IDEA and CVRetrieval

IDEA achieves efficient consistency maintenance by detecting and resolving inconsistencies among active writers more frequently than passive participants, in which active writers are dynamically tracked by IDEA. To reduce the number of nodes maintained by IDEA, CVRetrieval only lets IDEA handle active participants who are actively updating their replicas.

3.2. Satisfying passive participants' consistency needs on demand

Since CVRetrieval does not actively maintain consistency for passive participants who may need to access their replicas occasionally, CVRetrieval provides a way for these passive participants to access consistent objects when the need arises. From the passive participants' point of view, the only thing that they need to know is where to find a consistency object. *In IDEA, any active writer can provide a consistent object. So CVRetrieval just has to inform passive writers about this active writers' information.*

CVRetrieval deploys a publish-subscribe infrastructure (Banavar 1999) to publish the active writers information to the passive participants. In this way, CVRetrieval satisfies passive participants' consistency needs with an on-demand fashion. Moreover, CVRetrieval chooses publishers and subscribers in a way to capture the common interest among participants. In this way, passive participants associated with the same subscriber can help each other without fetching data from publisher all the time. As we will see in Section 6.3, exploiting common interest greatly improves the scalability of CVRetrieval.

4. CVRetrieval Design

We try to address several design issues here:

- How do participants join the system and how to map the participants to the IDEA infrastructure?
- What is the workflow of CVRetrieval?
- 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 to choose subscribers for observers?
- How does the publish-subscribe scheme work?

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

4.1. 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 group will change constantly, and such change is usually unpredictable because the spontaneity of an active discussion.

4.2. 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.

4.3. Mapping between participants and the IDEA infrastructure

As illustrated in Figure 2, we differentiate three types of participants: active writers, passive writers, and observers. They are mapped to IDEA as follows.

First, 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 that are handled by IDEA; if no, they are classified as observers that are handled by CVRetrieval.

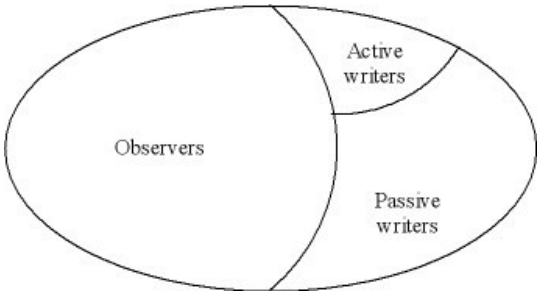


Figure 2: Three classes of participants

Second, IDEA differentiates active writers from inactive writers after the system starts to run using a two-layer structure. IDEA tracks active writers (by its top layer) and passive writers (by the bottom layer) based on frequency of their updating activities.

4.4. The Workflow

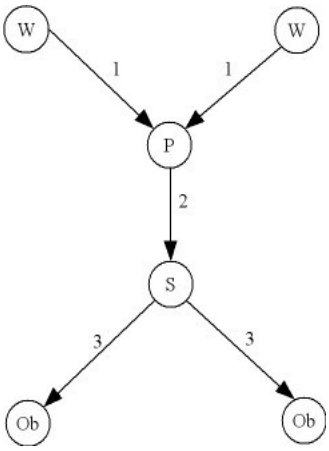
Figure 3 shows the workflow of the publish-subscribe mechanism as well as the retrieval process.

The basic publish mechanism is shown in Figure 3.1. In step 1, the active writers notify publisher about their presence; in step 2, a publisher notifies its subscriber about the up-to-date active-writer group; finally, in step 3, a subscriber notifies its clients (the observers) about the active-writer group.

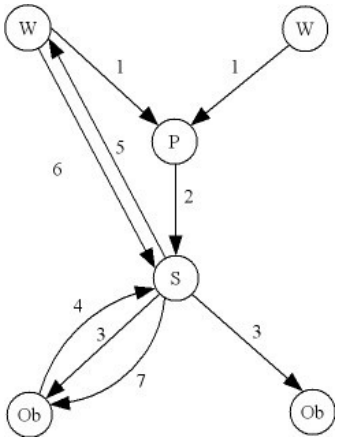
A client can issue an on-demand retrieval request, as shown in Figure 3.2. In step 4, an observer issues a retrieve request to its subscriber. If the subscriber has a valid cache, it will return the local copy to the observer (step 7); otherwise, it requests a consistent view from one of the active writers (step 5) and, after receiving the view (step 6), it returns the copy to the observer (step 7) and caches the view locally.

An observer can also indicate his or her preference to retrieve a consistent view periodically. In this case, the observer does not need to explicitly issue a retrieval request on-demand. As shown in Figure 3.3, this process is similar to that in Figure 3.2 except that there is no step 4, and steps 5 through 7 are executed periodically.

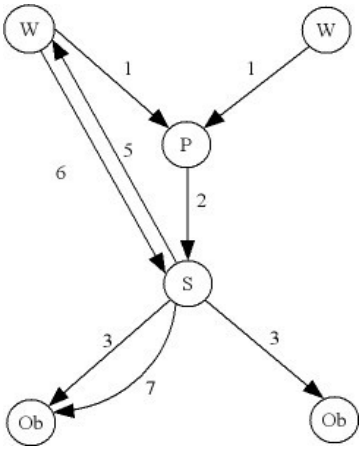
If the subscriber is already overwhelmed by the retrieval requests or publishing, there is no point of sending more retrieval request to it, and that is where the active-writer group information received by observers in step 3 comes into play. As shown in Figure 3.4, an observer can use its knowledge of the active-writer group to contact a nearby active writer



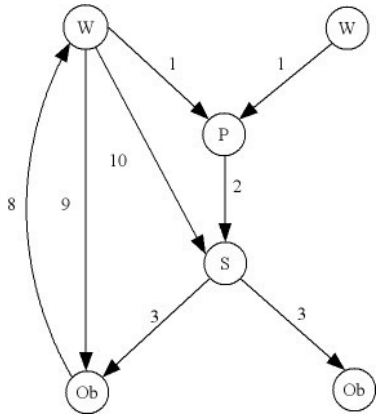
3.1: Basic publish mechanism



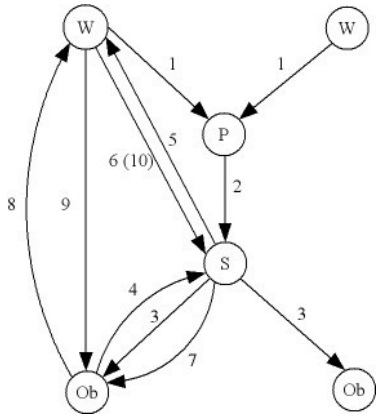
3.2: Active retrieval for observers



3.3: Automatic/Periodic retrieval from observers



3.4: When subscribers are already overwhelmed



3.5: The complete process

Figure 3: Workflow of CVRetrieval

directly (step 8 and 9). As an optional step, the active writer can forward a copy to the subscriber so that the subscriber will have a fresh copy as long as it is able to handle more requests again (step 10).

Finally, the complete process is illustrated in Figure 3.5. We will discuss the key components of the process in more details in the rest of this section.

4.5. 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. Hash-table-based scheme (choose publishers based on the hashed value of the object IDs), such as Distributed Hash Tables (DHT) (Ratnasamy 2001; Rowstron 2001; Stoica 2001), is desirable for both its load balancing and its easy lookup (subscribers can find the right publishers by simply hashing the object IDs).

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 4, we let *A* keep two pointers of its fellow members when it is in the top layer of object *f* (left half of Figure 4) 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 4). 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 4.7).

4.6. Choose subscribers for observers

While there are many ways to choose subscribers, we use ISPs (Internet Service Providers) of the observers, rather than some observers themselves, as the subscribers for two reasons. First, the ISPs are much more stable than their clients (*i.e.,* observers) because of their status as Internet entry point. Hence, using ISPs as the subscribers makes the publish-subscribe structure (*i.e.* the positions of publishers and subscribers) much more stable. 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 is not interested in an object anymore, 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 deletes the client from its client list.

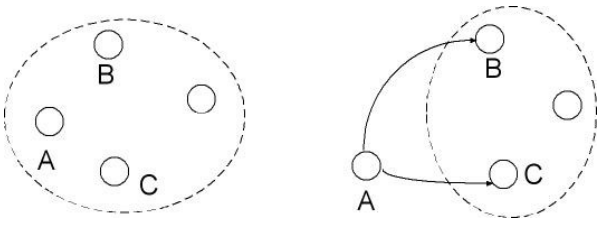


Figure 4. Use pointers to handle stale information

A subscriber has two responsibilities. First, it informs a publisher 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.

4.7. The publish-subscribe scheme

As shown in Figure 3, 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.

In the naïve form, the publisher sends all the active writers' 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?

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.

5. Scalability of CVRetrieval

In this section, we compare the communication cost of the CVRetrieval with two consistency maintenance protocols—Deno and IDEA—because these two are the most similar approaches to CVRetrieval. Due to page limit, interested readers are referred to Chapter 5.3 of (Lu 2007a) for a full discussion about the rationale behind this comparison.

5.1. Deno and IDEA

Deno (Cetintemel 2003) is 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 but their updates need to be serialized at a single point to maintain a consistency state.

IDEA is a detection-based consistency maintenance protocol for large-scale distributed systems proposed by the authors. Instead of enforcing a fixed consistency protocol beforehand, IDEA detects inconsistencies when they arise and resolve them based on the applications' ongoing need for consistency.

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 the total number of messages incurred by each protocol*.

5.2. Assumptions

In this analysis, 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 active writers among the n_1 writers.
[5] f_1: number of updates of active 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.

5.3. The Analysis

In this analysis, we consider the consistency control for one single object because this simplifies the analysis and, based on the result, it is easy to extend the analysis to multiple objects.

5.3.1. 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.

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:

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

5.3.2. Communication cost of IDEA

In IDEA, the updates from active writers will be detected among the active 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 active writers stay at the top layer, implying that the active 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 an active 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 active 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:

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

5.3.3. Communication cost of CVRetrieval

The communication cost of CVRetrieval involves three parts: (1) the detection of inconsistency among active 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 active writers in the same manner with that of IDEA because it depends on IDEA to maintain consistency. Thus the communication cost incurred by active 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 an active writer only notifies its publisher when it becomes an active writer and when it becomes a passive writer. Here we conservatively assume that, in one round of publish, half of the active writers are new ones (this is indeed a very extreme scenario because we essentially assume 50% of the active writers leave the group and the same number of new active writers join the group). Thus, in one round of publish, there are n_{hot} messages exchanged between writers and publishers because each old active writer or new active writer needs to inform exactly one publisher.

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

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	42500	39667	13125
2	1000	100	20	4	3	5	3	2	5	18	85000	69667	17543
3	1000	200	50	4	3	5	3	2	5	16	175000	124667	36399

Table 1: Analytical Results

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 should 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.

5.3.4. The comparison

In this comparison, we first do an asymptotic analysis to compare the overall growth rate of Deno, IDEA, and CVRetrieval. Since the asymptotic analysis is approximate in nature, we then use a sensible setting of the parameters to calculate and compare the three protocols.

We conduct the asymptotic analysis as follows. In the equation 1 for the communication cost of Deno, n_1 and n_2 are fractions of n, so n_1 and n_2 grows as fast as n. Then, f_1 and f_2 are updates in a period of time and is not supposed to be a large number and won't grow with n, so we can safely treat them as rough constants. Hence, the cost of Deno would be $O(n^2)$.

For the analysis of the communication cost of IDEA, we follow the analysis the same way as that of Deno—n_2 have similar growth as n, f_1 and f_2 are more like constant. Then, from equation 2, the cost of IDEA is $O(n^2 + n)$, which is also $O(n^2)$. Similarly, the cost of CVRetrieval, derived from equation 3, is also $O(n^2)$.

The main message here is this, while there are differences in the communication cost among all the three protocols, the difference is not an exponential one. This makes sense because all three protocols, to some extent, depend on intercommunication of a group of nodes, which results the $O(n^2)$ result. The real difference is how large the group is—the larger the group, the more communication cost will be incurred. From this aspect, Deno has the largest group (the whole system), IDEA has a smaller number (only for the group of active writers). CVRetrieval has the same group size as that of IDEA but has a much smaller size of passive writers, hence achieving the smallest communication cost.

We now proceed to the second step of this comparison by comparing C_deno, C_idea, and C_r by assigning real numbers to the parameters in their respective expressions. Based on the logic presented earlier, 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 active writers. The analytical results are summarized in Table 1.

As shown in Table 1, 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 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 4.1 for Deno and 3.1 for IDEA. We believe that this is an indication that CVRetrieval scales better than the other two methods.

6. Experimental Results

We have implemented a prototype of CVRetrieval on top of the Planet-Lab (Peterson 2003). We use this

prototype to evaluate the performance of CVRetrieval. The metric we use is response time.

For a consistency maintenance protocol, the response time 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, 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.

6.1. Experiment setup

We emulate a white board application for evaluation purposes. The application is emulated by following its operational sequences. Further, we assume that these updates are all conflicting with one another. A writer informs its publisher when it becomes or ceases to be an active writer. The publisher then informs its subscribers (the ISPs who subscribe on behalf of their clients) periodically. Observers specify their interest and inform their subscribers about that.

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 active 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. Each observer retrieves the consistent view every 20 seconds. 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.

6.2. 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 2.

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

Table 2: Response time for writers

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.

6.3. 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 3.

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

6.4. 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 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 4.

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 2). 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 with the pure consistency maintenance protocol 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 5.

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 5. We believe that the two features—efficiency and scalability—together make CVRetrieval a viable alternative to pure consistency maintenance protocols.

7. Future Trends

As Grid computing becomes a key enabling technology for large-scale collaboration application, quantitatively guaranteeing its QoS will become more

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 3: Response time for observers

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

Table 4: Response time of a pure consistency maintenance protocol with active update dissemination

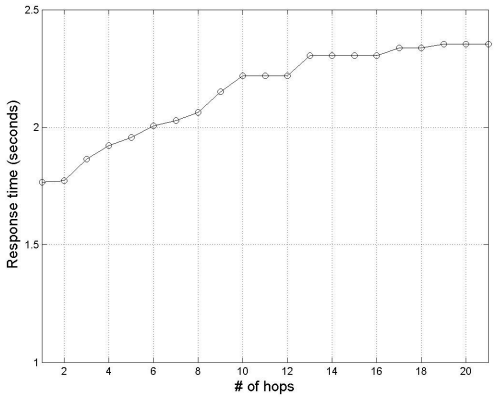


Figure 5: Response time for different hops

and more important. In terms of consistency control, advances in QoS will, based on the authors' opinion, be on two fronts: scalability and reliability.

Scalability refers to a protocol's ability to scale to a large number of geographically dispersed nodes. With the size of Internet keeping increasing, it becomes necessary for the Grid systems to maintain a meaningful consistency among different nodes while at the same time imposing very low communication overhead. Without this ability, practitioners will face a dilemma: either accept poor QoS in order to run the system in a large scale, or achieve high QoS at the expense of not cooperating with remote nodes. CVRetrieval presented in this paper is one way to improve scalability. Other alternatives are certainly possible.

Reliability refers to the robustness of a protocol. As Grid becomes an essential computing platform upon which numerous applications run, it is essentially that any key component of the Grid itself, including consistency control, is resilient to frequent packet delay/ loss and node failure. In this respect, any new

consistency control protocols will need to explicitly consider packet delay/loss and node failure in the design phase. Reliability in CVRetrieval depends on the robustness of the publish/subscribe infrastructure. It is interesting to see other alternatives that can provide even stronger and low overhead reliability guarantee.

8. 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, replication-based Grid systems by separating consistency retrieval from consistency maintenance.

CVRetrieval is fully evaluated by both analysis and prototyping. 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 through optimization. For example, we can drive active writer information towards the most needed subscribers by controlling the publishing rates along different paths. In such a scenario, the subscribers (the ISPs) will report their interests (in terms of frequency of issued requests) to the publisher, which in turn adjusts the publishing rates by publishing at a higher rate to a path that can reach subscribers that reveals higher interest than others.

References

Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R.E., & Sturman, D. C. (1999). An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems, *International Conference on Distributed Computing Systems* (pp. 262-272). Washington D.C. USA.

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

Cetintemel, U., Keleher, P. J., Bhattacharjee, B., & Franklin M. J. (2003). Deno: A Decentralized, Peer-to-Peer Object-Replication System for Weakly-Connected Environments. *IEEE Transactions on Computers, 52(7)*, 943-959.

Chang, T., Popsecu, G., & Codella, C. (2002). Scalable and Efficient Update Dissemination for Interactive Distributed Applications, *International Conference on Distributed Computing Systems* (pp. 143-152). Viena, Austria.

Li, F., Li, L., & Lau, R. (2004). Supporting Continuous Consistency in Multiplayer Online Games. *ACM Multimedia* (pp. 388-391). New York, New York, USA.

Lu, Y. (2007a). Improving Data Consistency Management and Overlay Multicast in Internet-scale Distributed Systems. Ph.D. Dissertation, University of Nebraska-Lincoln.

Lu, Y., Lu, Y. & Jiang, H. (2007b). IDEA: An Infrastructure of Detection-based Adaptive Consistency Control. *16th International Symposium on High Performance Distributed Computing* (pp. 223-224). Monterey, CA.

Lu, Y., Lu, Y. & Jiang, H. (2008). Adaptive Consistency Guarantees for Large-Scale Replicated Services. *2008 IEEE International Conference on Networking, Architecture and Storage,* Chongqing, China.

Peterson, L. L., Anderson, T. E., Culler, D. E., & Roscoe, T. (2003). A Blueprint for Introducing Disruptive Technology into the Internet. *Computer Communication Review. Vol. 33, No.* 1. 59-64.

Prakash, A. & Shim, H. S. (1994). DistView: Support for Building Efficient Collaborative Applications using Replicated Objects. *ACM conference on computer supported cooperative work.* (pp. 153-164). Chapel Hill, NC.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., & Shenker, S. (2001). A Content Addressable Network. ACM SIGCOMM (161-172). San Diego, CA.

Rowstron, A. & Druschel, P. (2001). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, *IFIP/ACM International conference on distributed systems platforms (Middleware)* (pp. 329-350). Heidelberg, Germany.

Schuckmann, C., Kirchner, L., Schummer, J., & Haake, J. M. (1996). Designing Object-oriented Synchronous Groupware with COAST. *ACM conference on computer supported cooperative work* .(pp. 30-38). Cambridge, MA.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *ACM SIGCOMM.* (pp.149-160). San Deigo, CA.

Vogel, J. & Mauve, M. (2001). Consistency Control for Distributed Interactive Media. *ACM Multimedia.* (pp. 221-230). Ottawa, Canada.

Yu, H. & Vahdat, A. (2000). Design and Evaluation of a Continuous Consistency Model for Replicated Services, *4th conference on Symposium on Operating System Design & Implementation*. San Diego, California.