RC22237 (W0111-022) November 7, 2001
Computer Science

IBM Research Report

Scalable and Efficient Update Dissemination for Interactive Distributed Applications

Tianying Chang, George Popescu, Christopher F. Codella
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

Scalable and Efficient Update Dissemination for Interactive Distributed Applications

Tianying Chang
College of Computing
Georgia Institute of Technology
Atlanta,GA 30332
tychang@cc.gatech.edu

George Popescu Chris Codella
IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
{popescu,codella}@us.ibm.com

Abstract— **Interactive distributed applications such as multiplayer games will become increasingly popular in wide area distributed systems. To provide the response time desired by users despite high and unpredictable communication latency in such systems, shared objects will be cached or replicated by clients that participate in the applications. Any updates to the shared objects will have to he disseminated to clients that actually use the objects to maintain consistency. We address the problem of efficient and scalable update dissemination in an environment where client interests can change dynamically and the number of multicast channels available for dissemination is Limited. We present a heuristic based algorithm that can group objects and client in a way that it handles limited bandwidth resources. We show that our algorithm can produce better results than several algorithms that have been developed in the past for update dissemination.**

Keywords— **Distributed Interactive Application, Distributed Object, Object Caching, CORBA, Multicast Grouping, Massive Multiplayer Game**

I. Introduction

Interactive distributed applications like massive multiplayer games, virtual reality video conference, virtual shopping malls, battle field simulations, and collaborative work environments connect remote producers and consumers of information together in real time over the Internet. The *heterogeneity* of the Internet along with real-time requirements and *large* number of users complicate the deployment of such applications.

Distributed object platforms, e.g., CORBA, DCOM and RMI, provide middleware support for distributed applications. However, they do not natively meet the real-time requirements of interactive distributed applications over the Internet. The state of an object is transferred "on demand" to the remote user via an RPC call. The large latency of tbe Internet must be endured twice: first to request the object's state, and second to receive the object's state.

Latency can be reduced by replicating an object at its remote users rather than always transferring its state on demand. When replicating an object, consistency must be considered between the object and its replicas. A replica of an object does not need to be totally consistent with the object. Different object may require different levels of consistency, say, some critical objects require immediate updates on all of its replicas while other objects require only periodical updates. The replica of an object at a remote user may be more or less consistent when compared to replicas of this object at other remote users. The consistency of a replica can depend on the remote user's available network resources. For example, a workstation with a broadband network connection can maintain a more consistent replica of an object than a PDA with a wireless connection. The frameworks and algorithms about what to cache, where to cache, and when to cache are very interesting topics but are out of the scope of this paper.

In this paper, we concentrate on the following problem: how can the consistency of object replicas at a distributed interactive application's remote users be efficiently maintained within each remote user's consistency requirements? This problem is interesting when the number of remote users and replicated objects is very large and network resources are limited. The key to making scalable applications in a heterogeneous environment with limited network resources is in how the state updates of an object are disseminated to remote replicating users.

Two transport technologies can be used to disseminate state updates of an object to remote replicating users. The first option is to use unicast, where the state update of an object is transmitted to only one remote replicating user at a time. However in an interactive application, multiple remote users will replicate overlapping sets of objects. As a result, unicast will use the object's host's outgoing bandwidth inefficiently. Another option is to use multicast. A multicast channel can be allocated for each replicated object, and remote users who replicate this object can listen to this channel. The state update of an object can be transmitted over the appropriate multicast channel and reach all replicating remote users at the same time.

The above approach to using multicast is naive. Multicast channels are not free; they consume resources inside the network, i.e., router memory for storing multicast information and router bandwidth for maintaining multicast information [1][2]. The number of multicast channels available for use by an application is limited. Since an application can have a large number of replicated objects, several objects must be grouped together into the same multicast channel. However a new problem arises: a remote user can receive updates for objects that it does not replicate because these objects are grouped with objects the remote user does replicate. The result is that the remote user's incoming bandwidth

can be used inefficiently.

The key to making large-scale distributed network applications feasible is to group objects into multicast channels in an intelligent manner so that the following requirements are satisfied:

1. The object servers should not send more data than its outgoing bandwidth capacity;
2. Each replicating user should not receive more data than its incoming bandwidth capacity;
3. Only limited number of multicast channels are used.

In this paper, we propose a model for the problem of finding adequate grouping arrangements of replicated objects and replicating users into multicast channels. Our model takes into consideration that remote users have heterogeneous resource constrains and can replicate objects at varying degrees of consistency. We also propose a greedy-heuristic algorithm to solve this problem. Our algorithm is *incremental*: as the set of objects that remote users replicate changes, a new grouping arrangement is derived from the previous grouping arrangement. This allows the number of expensive join and leave operations to be minimized and the running time to be low. Our algorithm is also *adaptive*: if an adequate grouping arrangement cannot be found in a timely manner, the problem can be made easier according to the policy of the application. For example in some applications, objects replicated at low-degree of consistency at a remote user could be dropped in favor of preserving the consistency more important objects.

The remainder of this paper is organized as following. In Section II, we present a matrix-based model that formalizes our problem. In section III, we present the greedy heuristic algorithm to solve the problem. In section IV, we describe related works. In section V, we evaluate our algorithm on a simulated multiplayer game application. We conclude in Section VI.

II. Group Arrangement Problem

A. Example Application: Massive Multiplayer Game

Massive multiplayer games (MMGs) are good representative of interactive distributed applications. MMGs involve a large number players who play over the Internet, and state updates must be disseminated in a timely manner in order for the game to be playable. As MMGs become more sophisticated, they demand more network resources or demand using existing network resources more efficiently. We will use MMGs to demonstrate how our algorithm solves the group arrangement problem.

In an MMG, many players interact in the same virtual world. This virtual world consists of many game entities, for example, the avatars of players, non-player monsters, and props. At any given time, each player is only interested in a relatively small subset of the virtual world's entities, and can be interested in each game entity at varying levels of detail. Each game entity has state associated with it, for example, location, posture, and physical condition. As the state of a game entity changes, state updates must be disseminated to players who would notice the change.

As an example, consider an MMG with thousands of players interacting in a sporting event taking place in a large stadium. Players, or more concretely, the avatars of players, can have different roles in this sporting event: they can be participating directly in the sporting event on the field in the stadium, or they can merely be spectators in the bleacheres. Players of the MMG have interest in various game entities in the stadium according to their role in the game. Direct participants are going to be highly interested in other direct participants on the field, and they might also be lightly interested in the spectators in the bleacheres. Spectators can be interested in the game entities on the field, the scoreboard, the announcer, and other spectators.

When such an application is implemented over a distributed object platform, game entity could be designed as a set of objects, e.g., locations, outlines, colors and textures, which incrementally define more details about the entity. For displaying an entity with different levels of details, different amount of objects are required. Objects could be implemented at a game server and cached at client hosts. Updates to an object, e.g., a component object of an avatar, are first handled on local replica, then immediately unicast to the server and last periodically multicast to other clients that also cache the object. The consistency of objects and their replicas determines whether an jumpy image is displayed.

B. Layered Preferences and Adaptive Group Arrangement

We focus on the periodical update dissemination from the server to the clients in above example. Each client has its set of replicated objects and consistency settings on these objects. Given certain amount of system resources, i.e., number of multicast channels, server's outgoing bandwidth and clients' incoming bandwidth, the possibility of finding a feasible group arrangement depends on both system resources and client preferences. The more updates the clients ask for and the lower resources the system has, the lower the possibility is.

To increase the possibility of finding a feasible group arrangement, the clients can choose to be very conservative. They can ask for minimum amount of data — less cached objects and less frequent updates. This often means lower level of game image quality and meanwhile the system resources are not fully exploited. On the other hand, the clients can choose to ask for a lot of data to achieve higher game image quality and resource utilization but risk the chance of finding a feasible group arrangement. Because each client act independently in such a distributed system, it is difficult for the clients to give "suitable" preferences that neither overload nor waste the system resources. To solve this problem, we propose *layered preference and adaptive group arrangement* strategy as following: When clients submit preferences, they assign different priorities to different objects; The server starts to search a feasible group arrangement that meets all the preferences without concerning the priorities; If it can

not succeed, it drops preference on some or all low priority objects and tries again. This step is repeated until a feasible group arrangement is finally found.

In the remainder of this section and in whole Section III, we only focus on the single step in the above repeating procedure. That is, when we try to find a feasible group arrangement, we concern only the preferences themselves and ignore the associated priorities.

C. Examples of Group Arrangement

We give a couple of simple examples of group arrangement here. In this subsection and hereafter, we start to use a set of more general terms and notations. We use *source* to substitute for *object*, *receiver* for *client*, and *channel* for *multicast channel*. Source, receiver and channel are denoted by s_i, r_i and c_i respectively.

Table I shows the outgoing bandwidth of server and incoming bandwidth capacity of seven receivers and their preferences to seven sources. r_1, r_2, r_3, r_4, r_5 are low end users that require less data and have lower bandwidth capacity; r_6, r_7 are high end users that require more data and have higher bandwidth capacity.

receiver	receiver's preference	B/W
r_1	(s_1, s_2, s_3)	4
r_2	(s_1, s_3, s_4)	4
r_3	(s_1, s_4, s_5)	4
r_4	(s_1, s_5, s_6)	4
r_5	(s_1, s_5, s_7)	4
r_6	$(s_1, s_2, s_3, s_6, s_7)$	8
r_7	$(s_1, s_2, s_3, s_4, s_5)$	8
Server		12

TABLE I
REQUIREMENT CONDITIONS

To provide insights into the difficulty in grouping sources and receivers, we try several possible group arrangements with three channels and show how they may not meet all the requirements.

- Case 1: Table II shows a grouping where all sources are partitioned into three channels, which means no overlaps among sources in different channels. In this case, we can see that the server's outgoing bandwidth is best optimized, it only sends seven sources out, no duplicated sending. However, r_3's incoming bandwidth is overloaded.

channel	receivers	sources
c_1	$r_1, r_2, r_3, r_4, r_5, r_6, r_7$	s_1
c_2	r_1, r_2, r_3, r_6, r_7	s_2, s_3, s_4
c_3	r_3, r_4, r_5, r_6, r_7	s_5, s_6, s_7

TABLE II
GROUP ARRANGEMENT 1

- Case 2: Table III shows a group arrangement where receivers are partitioned into three channels, which means no overlap among receivers in different channels. In this case, server's outgoing bandwidth is overused by four; also r_3, r_4, r_5's incoming bandwidth is overloaded.

channel	receivers	sources
c_1	r_1, r_2	s_1, s_2, s_3, s_4
c_2	r_3, r_4, r_5	s_1, s_4, s_5, s_6, s_7
c_3	r_6, r_7	$s_1, s_2, s_3, s_4, s_5, s_6, s_7$

TABLE III
GROUP ARRANGEMENT 2

- Case 3: Table IV shows a **feasible** group arrangement.

channel	receivers	sources
c_1	r_1, r_2, r_6, r_7	s_1, s_2, s_3, s_4
c_2	r_3, r_7	s_1, s_4, s_5
c_3	r_4, r_4, r_6	s_1, s_5, s_6, s_7

TABLE IV
GROUP ARRANGEMENT 3

D. Mathematical Representations

We represent receivers' diverse preferences with an $m \times n$ matrix, named *preference matrix* and denoted as M^{rs}. n and m are the number of sources and that of receivers, respectively. Each entry in M^{rs} is either **0** or **1**; if receiver r_i prefers source s_j, $M^{\mathrm{rs}}_{i,j}$ equals to 1, otherwise it equals to 0. Put in another way, row i describes the sources that r_i prefers and column j describes the receivers that prefer s_j. Fig. 1(a) shows the preference matrix for example in Table I.

	s1	s2	s3	s4	s5	s6	s7
r1	1	1	1	0	0	0	0
r2	1	0	1	1	0	0	0
r3	1	0	0	1	1	0	0
r4	1	0	0	0	1	1	0
r5	1	0	0	0	1	0	1
r6	1	1	1	0	0	1	1
r7	1	1	1	1	1	0	0

(a) M^{rs}

	c1	c2	c3
r1	1	0	0
r2	1	0	0
r3	0	1	0
r4	0	0	1
r5	0	0	1
r6	1	0	1
r7	1	1	0

(b) M^{rc}

	s1	s2	s3	s4	s5	s6	s7
c1	1	1	1	1	0	0	0
c2	1	0	0	1	1	0	0
c3	1	0	0	0	1	1	1

(c) M^{cs}

	s1	s2	s3	s4	s5	s6	s7
r1	1	1	1	1	0	0	0
r2	1	1	1	1	0	0	0
r3	1	0	0	1	1	0	0
r4	1	0	0	0	1	1	1
r5	1	0	0	0	1	1	1
r6	2	1	1	1	1	1	1
r7	2	1	1	2	1	0	0

(d) $A^{\mathrm{rs}} = M^{\mathrm{rc}} \times M^{\mathrm{cs}}$

Fig. 1. (a) is the preference matrix; (b) is receiver subscription matrix; (c) is source subscription matrix (d) is the product of M^{rc} and M^{cs};

The system resources are defined by server outgoing bandwidth B^{s}, number of channels k and clients' incoming bandwidth vector B^{r}. For example, in Table I, $k = 3, B^{\mathrm{s}} = [12]$, and $B^{\mathrm{r}} = \{4, 4, 4, 4, 4, 8, 8\}$.

When grouping receivers and sources into channels, we use another two 0-1 matrices to represent the group arrangement: a *receiver subscription matrix* and a *source subscription matrix*. The former, denoted by M^{RC}, is an $m \times k$ matrix that describes how receivers are related to channels; $M^{\text{RC}}_{i,k}$ equals to 1 means receiver r_i listens to channel c_k. The latter, denoted by M^{CS}, is a $k \times n$ matrix that describes how sources are related to channels; $M^{\text{CS}}_{k,j}$ equals to 1 means source s_j sends updates to channel c_k. The two subscription matrices corresponding to the example in Table IV are given in Fig. 1(b) and Fig. 1(c).

The product of M^{RC} and M^{CS}, denoted by A^{RS}, represents the actual amount of data received by receivers. Entry $A^{\text{RS}}_{i,j}$ is the actual number of times that r_i receives s_j. Fig. 1(d) shows the value of A^{RS} corresponding to Table IV.

A feasible group arrangement must satisfies the following conditions:

Preference Limitation Receivers' preferences must be met.

$$\forall i, \forall j, A^{\text{RS}}_{i,j} \geq M^{\text{RS}}_{i,j} \tag{1}$$

Bandwidth Limitation Server and receivers' traffic must not exceed their bandwidth.

$$\sum_{t=0}^{k} \sum_{j=0}^{n} M^{\text{CS}}_{k,n} \leq B^{s} \tag{2}$$

$$\forall i, \sum_{j=0}^{n} A^{\text{RS}}_{i,j} \leq B^{r}_{i} \tag{3}$$

III. Finding Feasible Group Arrangement: A Heuristic Searching Algorithm

A. Revised Goal

As previously discussed, a feasible group arrangement must satisfy the following conditions:

- Only given number of channels are used.
- Each receiver receives preferred data.
- Each receiver receives no more data than its incoming bandwidth capacity.
- Server sends no more data than its outgoing bandwidth capacity.

In practice, the server usually has much larger bandwidth than the receivers does. If a group arrangement satisfies the first **three** conditions, it is very unlikely that the arrangement does not satisfies the last condition. Therefore, we do not explicitly attack the satisfaction condition on the server traffic in our algorithm. Instead, we try to reduce the server traffic whenever possible while finding a group arrangement that satisfies the first three conditions.

B. General Idea

Our algorithm starts from an initial group arrangement that uses the given number of channels and meets the receiver preferences. First, we must check if the group arrangement satisfies receiver bandwidth conditions. If it does satisfy receiver bandwidth conditions, it is a feasible group arrangement and the algorithm halts. Otherwise, it is not a feasible group arrangement and it must be adjusted into another group arrangement that also satisfies receivers preferences and use k channels, and then rechecked to see if the receiver bandwidth conditions are met. This check-and-adjust continues until a feasible group arrangement is found.

The pseudo code for the algorithm is shown in Fig. 2. We will go into detail about the single adjustment step in the next subsection, but first we present two practical issues about our algorithm.

First, if network resources are very low relative to receiver preference requirements, a feasible group arrangement either may not exist, or it might take a very long time to find. In such a case, the algorithm would continue the check-and-adjust loop for a long period of time without halting. Given specific receiver preference requirements and network resource constraints, there is no formal way for us to judge how many feasible group arrangements exist, or whether one exists at all. Given the real time nature of our system, the algorithm cannot search for a solution for an unbounded amount of time. Instead, we limit the searching time of our algorithm. If a feasible group arrangement cannot be found within the specified limit, the receiver preference requirements are relaxed in an application specific manner, then the algorithm is run on this new problem. This allows the algorithm to adapt to low resource conditions or aggressive receiver preferences. On the other hand, receiver preference requirements can be tensed in an application specific manner to fully utilize available network resources.

Second, there are several ways to generate an initial group arrangement. A simple way is to randomly generate a group arrangement and *patch* it, which ensures that it satisfies receiver preference requirements. However, in most interactive distributed applications the receiver preferences evolve slowly. It is better to reuse the previously feasible group arrangement, and patch it to ensure that it satisfies the new receiver preference requirements. In this case, the algorithm is more likely to find the new feasible group arrangement more quickly than if it started from a random initial group arrangement. In this sense, our algorithm is **incremental**. By using the previous group arrangement when searching for a new group arrangement, parts of the previous group arrangement that are not effected by new receiver preferences are less likely to be changed in the new group arrangement. This has the added benefit of reducing the amount of costly multicast join and leave operations.

C. Two Phases of A Single Adjustment Step: Splitting and Merging

First, we will explain some terminologies that are going to be used later:

- *waste* is the unwanted data received by a receiver;
- *bad-waste* is the waste that exceeds a receiver's bandwidth capacity. It is the amount of data received minus receiver's

```
Algorithm Greedy_Grouping():
 1.  Generate an initial grouping that uses
     k channels and satisfies all receivers
     preference requirements
 2.  While there are overloaded receivers {
 3.  // Derive a better grouping that
     // still uses k channels
 4.      Split()
 5.      Merge()
     }

Algorithm  Split():
  1. While there are overloaded receivers {
  2.     Select the receiver ri that is
         most overloaded
  3.     Select the channel Ci that sends
         the most waste to ri
  4.     (Ci1, Ci2) = Connection_Partition(Ci)
         // Split Ci into two subchannels Ci1
         // and Ci2.
  5.     Remove Ci from the list of
         channels
  6.     Add Ci1 and Ci2 into the list of
         channels
     }

Algorithm Merge():
 1.  While we are using more than k channels {
 2.      For every possible pair of
         channels (Cx, Cy) {
 3.           Calculate the increased waste
              and bad-waste of merging Cx
              and Cy
          }
 4.      Select the pair (C1, C2) that has
         the least increased bad-waste or
         waste if bad-waste is tied
 5.      Merge C1 and C2 into a new channel
         C3
 6.      Add C3 into the list of channels
 7.      Remove C1 and C2 from the list of
         channels
     }

Algorithm Connection_Partition( Ci ) {
 1.  Identify each valid connection L[ri,sj]
     in channel Ci
 2.  For every pair of connections (L1, L2) {
 3.      Calculate D(L2,L2), the distance
         between the connections L1 and L2
      }
 4.  Create two new empty channels Ci1 and Ci2
 5.  Select the pair of connections (Lx, Ly) that
     have largest distance
 6.  Add  Lx into Ci1
 7.  Add  Ly into Ci2
 8.  Remove Lx, Ly from Ci
 9.  While Ci is not empty{
 10.      For every connection Li in channel Ci {
 11.        Calculate the reduced waste if moving
            Li into Ci1 or Ci2
          }
 12.      Select the connection Li and channel
          Cix that has the most reduced waste
 13.      Remove Li from Ci
 14.      Add Li into Ci1 or Ci2 depending
          on which one reduces more waste
    }
 }
```

Fig. 2. Split/Merge Greedy Algorithm

bandwidth capacity if a receiver receives more data than its bandwidth capacity; otherwise it is 0.

• *distance* between connection L_1 $[r_i, s_j]$ and connection L_2 $[r_x, s_y]$ is called $D(L_1, L_2)$; $[r_i, s_j]$ is a *connection* if r_i is interested in s_j. Distance between two connections/sources/receivers describes the similarity between two connections/sources/receivers.

$$D(L_1, L_2) = w_1 \times D(r_i, r_x) + w_2 \times D(s_j, s_y)$$

w_1 and w_2 are weight of $D(r_i, r_x)$ and $D(s_j, s_y)$ correspondingly

$$D(r_i, r_x) = \frac{|sources(r_i) \bigcup sources(r_x)|}{|sources(r_i) \bigcap sources(r_x)|}$$

$sources(r_i)$ represents the set of sources that are preferred by r_i; similarly, $D(s_j, s_y)$ can be defined.

$$D(s_j, s_y) = \frac{|receivers(s_j) \bigcup sources(s_y)|}{|receivers(s_j) \bigcap receivers(s_y)|}$$

$receivers(s_j)$ represents the set of receivers that are interested in s_j.

When any receiver is overloaded, we use the *split* operation. First, the most overloaded receiver is chosen. Then, the channel that brings the most *waste* to the overloaded receiver is chosen. This channel is split into two sub channels in the way that reduces the **most** *bad-waste*. For a channel with x receivers and y sources, there are 2^{x+y} ways of splitting it into two sub channels. This is computationally too expensive and unnecessary.

In our algorithm, splitting a channel into two subchannels to reduce receivers waste is done by partitioning the *connections* $[r_i, s_j]$ in the channel into two groups. *Partition* here means that an entity will appear in only one group, not both. We can see that the six connections in Fig 3(a) are partitioned into two groups in Fig 3(b). From (a) to (b), the dashed lines are reduced by two, which means receivers are receiving two units less waste. We want to reduce the **most** waste data for receivers when splitting a channel. The connection partition is shown in Fig 2. By splitting the connections in a channel, we are not biased in favor of either receivers or sources. This frees us from doing just receiver partitions where a receiver can only appear in one channel and source partitions where a source can appear only in one channel [3]. In our approach receivers and sources will appear in any number of channels as long as it reduces receivers waste. The splitting operation is repeated until no receiver is overloaded anymore. But, when this condition is reached, it is possible that we may be using more than k channels.

When more than k channels are used, we will use the *merge* operation to combine two channels into one. Merging c_1 and c_2 in figure 3(b), we can get c_1 in Fig 3 (a). After the merge, the server needs to sends out only three source objects, instead of five. However, receivers r_2 and r_3 now receive more waste. In general, merging two channels will

cause receivers to receive more waste and the server to send less data. The increased receiver side bad-waste (or waste) after merging two channels into one is calculated for each pair of channels. We choose the pair of channels that **least** increases receiver side *bad-waste* (or *waste* if *bad-waste* ties) after a merge and merge again greedily. The merge operation is repeated until only k channels are left.

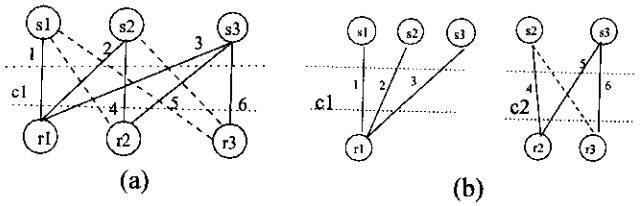


Fig. 3. Receiver r_1 is interested in source s_1, s_2, s_3; receiver r_2 is interested in source s_2, s_3; receiver r_3 is interested in source s_3. There are six connections which are connected by six solid lines. Dashed line connecting a receiver and a source means this receiver is not interested in this source, but is receiving this source since they are in the same group. The two channels in (b) is gotten by splitting the six connections in (a) into group (1, 2, 3) and group (4, 5, 6). From (b) to (a) is merging two channels into one

D. Computation Complexity

Our algorithm consists of repeated splitting and merging steps. The splitting steps are repeated until no receiver is overloaded. Assuming that the number of channels used is x when no receiver is overloaded, then splitting is repeated $x-k$ times when no receiver is overloaded. When we need to merge the x channels back to k channels, merging is repeated $x-k$ times since each merge reduces one channel used.

When splitting a channel with m receivers and n sources, the number of connections c is bounded by $m \times n$. If we move a connection from the original channel into one of the two new channels, we need to calculate the reduced waste of moving each connection in the original channel. The overhead of moving a connection into one of the two new channels is equivalent to the overhead of set merging. Assuming the overhead of set merging is a constant a, the computation complexity of splitting a channel into two is $O(xc^2)$.

While using the merge operation, we need to choose which two channels to merge. To make this choice, we need to calculate the increased waste and bad-waste resulting from merging every possible pair of channels. The overhead of one step in this calculation is the overhead of set merging, assumed to be the constant b. Since this is repeated for all possible pairs of channels, the computation complexity of the merge operation is $O(x^2)$. Since the merge needs to be repeated $x-k$ times to reduce the number of channels to k. So the computation complexity of merging x channels into k is channels is $O(x^3)$.

So the computation complexity of one loop iteration is $O(x^3 + xc^2)$. The number of loop iterations is bounded by an input parameter, which is used to reduce running time to reasonable limits for a server.

IV. Related work

Middleware-level support to interactive distributed applications has been discussed in the literature. To name a few, Krishnaswamy et al. presented a framework for caching distributed objects at client sites and a consistency QoS specification interface that is meaningful at the application level [4]; O'Ryan et al. extended the CORBA Event Service so that it is better suited for distributed interactive simulation applications [5].

Multicast group arrangement problem is not new. Researchers have met this problem in distributed interactive simulation applications and proposed cell-based and entity-based group arrangement algorithms [6] [7] [8] [9] [10] [11] [12] Their efficiency are extensively investigated in [13] [14]. The algorithms are simple yet powerful in simple gaming environment, where the relationships between game entities can be defined by simple vision domain rules. However, they might not be flexible enough for more complex gaming environments(e.g. one as described in Section II.A) or other interactive distributed applications and therefore cannot serve for a general distributed object platform.

Wong et al. were the first to investigate the multicast group arrangement in a general form and proposed an algorithm based on the k-means clustering method [3]. The algorithm does not serve our purpose for two reasons. First, the algorithm targets on optimizing the overall traffic received on all receivers. The heterogeneity of receivers is totally ignored and therefore the algorithm may give a solution that overloads certain individual receivers. Second, the algorithm considers either only similarity among sources or only that among receivers, while the similarity between connections could be a more general and flexible criteria. This limits its scope of finding better group arrangement.

V. Evaluation

To show the effectiveness of our algorithm of finding feasible group arrangement, we conduct experiments in the context of applying our algorithm to a simulated multiplayer game. We compare the performance of our algorithm against that of the cell-based grouping algorithm. We are also interested in finding the effective advantage of the connection similarity that used in our algorithm over the receivers(or sources) similarity that is used in [3].

A. Generating Preferences With Simulated Multiplayer Game

For our experiments, we generate preference matrices with a simulated multiplayer game. We use a model similar to that used in [13]. The virtual world of the multiplayer game is a rectangular battlefield. There are two sets of stationary or moving entities in the battlefield: receivers and sources, and we assume the two sets do not overlap.

To make our algorithm comparable with cell-based algorithm, vision domain circles around receivers are used in the calculation of their preferences to sources. To simulate di-

verse receiver preferences and diverse source popularity, we assign random brightness to each source and assign random vision domain size to each receiver. We define receivers' preferences as follows: if the distance between a source and a receiver is less than the receiver's vision domain size multiplied by the brightness of the source, then this source is in this receiver' preference set.

The brighter a source is, the more receivers that can see it; the larger a receiver's vision domain is, the more sources it can see. There could be *hot receivers*, who have far more preferences than other receivers do. There could also be *hot sources*, which are preferred by far more receivers than other sources are. By configuring different number of hot sources and hot receivers, we can simulate diverse preference, which can in turn affect the quality of grouping.

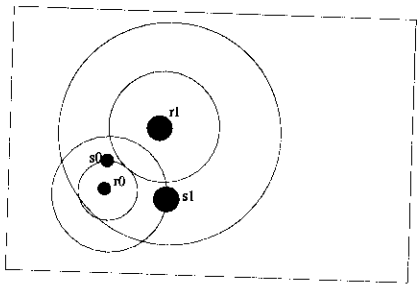


Fig. 4. Receivers `r0` and `r1` and sources `s0` and `s1`; the vision domain of each receiver is drawn as two circles; the area of the inner circle around a receiver represents interest in sources of brightness one, and the area of the outer circle represents interest in sources of brightness two; receivers and sources with larger vision domains and brightness are drawn as larger filled dots.

An example of variable vision domains and brightness is shown in Fig 4. Receiver `r0` has an interest in sources `s0` and `s1`. The source `s0` has a brightness of one, and it is just within receiver `r0`'s vision domain. The source `s1` is twice as bright as the source `s0`, so it can been seen twice as far away by receiver `r0`. The receiver `r1` has a vision domain that is twice as large as receiver `r0`. As a result, it can see source `s1` but cannot see the source `s0`, since it is outside of source `s1`'s brightness-one vision domain.

Receiver and sources move around the virtual world at varying velocities. The preferences of all receivers are periodically sampled to create a preference matrix, taking into account receiver vision domains and source brightness. The result is a sequence of preference matrices, which can be used as the input for various group arrangement algorithms.

We also simulate network resource conditions. We set values for the number of available multicast channels, receiver incoming bandwidth, and server outgoing bandwidth. We vary these values to see performance of group arrangement algorithms under different resource conditions.

Finally, we limit the running time of our algorithm. As we have mentioned previously, given a preference matrix and available network resources, it is possible that our algorithm does not find a feasible group arrangement within a bounded amount of time. In this case, the application adapts to make the problem easier and then our algorithm is reran. When a solution cannot be found, we have chosen to reduce quality by reducing the bandwidth needed to transmit a state update by 10%. When a feasible group arrangement is found without the need to reduce quality, we successively increase quality by 10% until a feasible group arrangement cannot be found. The effect is that the quality of the game experience can vary according to receiver preferences and available network resources, though the game should always be playable.

Cell-based algorithm does not intend to search for a *feasible* group arrangement, so it always halts within certain running time. However, the group arrangement they derive may not be feasible with respect to available network resources. We use the same adaptation policy for these algorithms by increasing and decreasing quality so that quality is as large as it can be while the group arrangement they find is still feasible. Thus, one benchmark for comparing group arrangement algorithms is to compare the quality of the game that can be supported when they are used.

B. Comparison With Cell-based Algorithm

B.1 Game Quality

In cell-based algorithm, the battlefield is evenly divided into k rectangle cells, each of which is associated with a multicast channel. The receivers listen to the multicast channels associated with the cells that overlap with his vision domain.

We compare the game quality supported by our greedy algorithm against that by cell-based algorithm for each preference matrix given a set of resource conditions.Fig 5 shows that our greedy algorithm supports higher game quality than cell-based algorithm does.

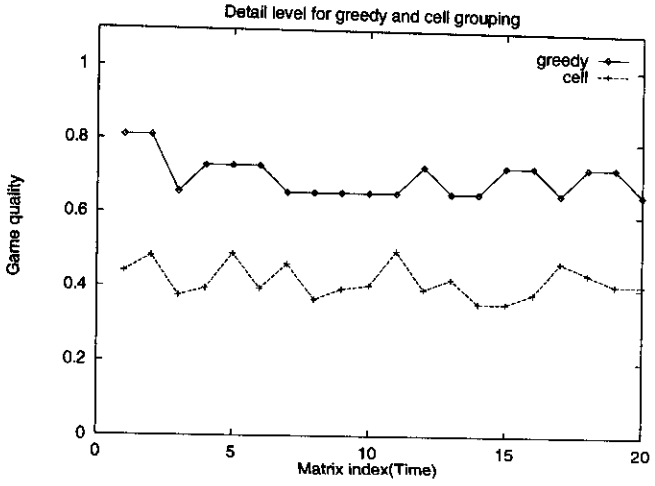


Fig. 5. Game quality of a game having 100 receivers and 100 sources. One third of the receivers have vision circle radius of 40, bandwidth capacity of 15; another third of the receivers have vision circle radius of 60 and bandwidth capacity of 25; the other one third of the receivers have vision circle radius of 80 and bandwidth capacity of 35

Second we show the trend of game quality when resource conditions change. Fig 6 shows the different level of game

quality when different number of multicast channels are used and when receivers have different bandwidth capacity. The figure shows that as more channels used, the quality supported by an algorithm becomes better. Too few channels can provide only very low game quality; but too many channels are not necessary either.

With cell-based algorithm the receivers simply receive data from all the cells that overlap with their vision domains. When it chooses group arrangement, it does not consider individual clients' specific preferences and bandwidth constrains. How much data a receiver can get totally depends on the location of the receiver in the battlefield instead of its actual bandwidth. So receivers can receive large amount of waste data and the overall game quality is degraded. When same number of multicast channels are used, our greedy algorithm is always able to support higher game quality. This suggests our algorithm provides higher game quality under same level of network resource conditions. We see the same trend when we run the experiment under several different resource configurations.

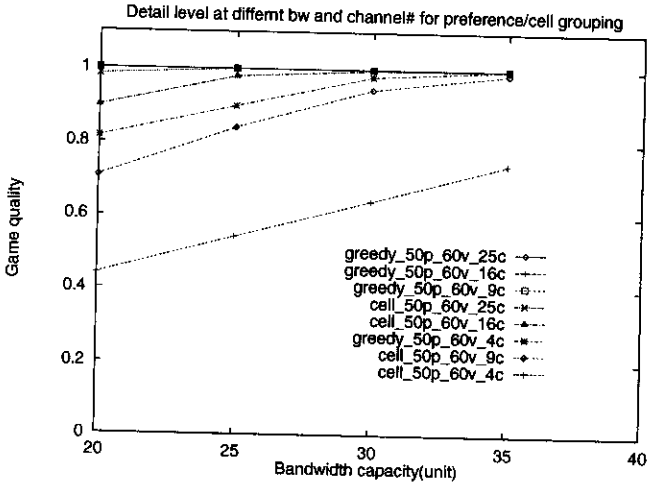


Fig. 6. Game Quality when varying channel number and bandwidth. 50p means there are 50 receivers; 60v means each receivers's vision circle radius is 60; the number before c represents the number of channels used

B.2 Number of Joins and Leaves During Regrouping

When receivers' preferences change, group arrangement changes. Since joins and leaves of members can introduce overhead to the maintenance of a multicast channel, a better group arrangement algorithm should incur less joins and leaves during regrouping.

Fig 7 shows that our group arrangement incurs much less joins during regrouping given same resource conditions and same sequence of preference matrices. The total number of leaves follows.

C. Connection Similarity Versus Receiver(source) Similarity

Receiver(source)-based Clustering is proposed in [3]. Because [3] intends to reduce the **overall** traffic received by

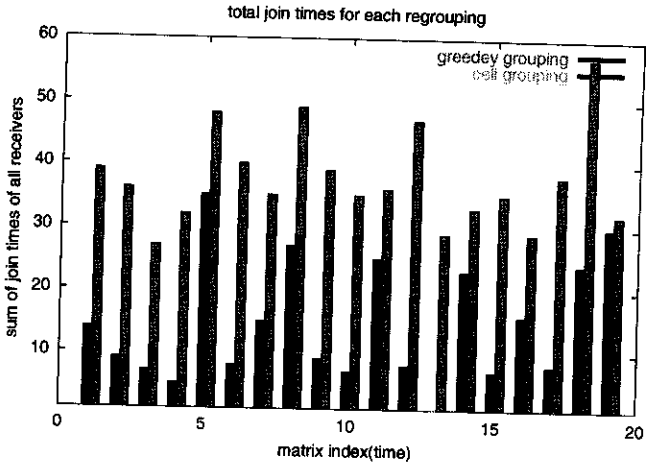
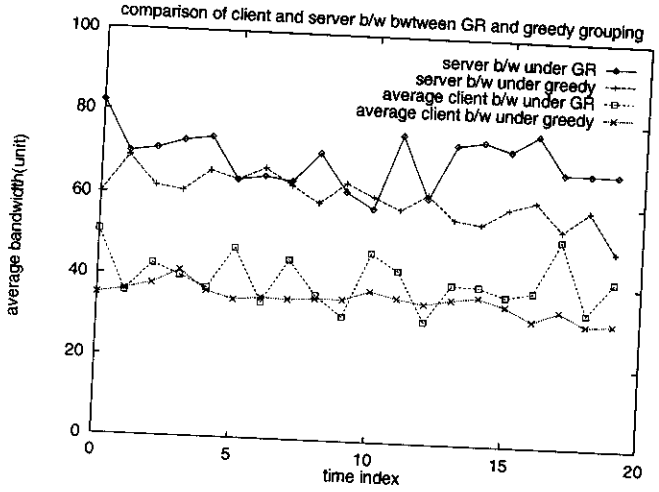


Fig. 7. Total number of joins when regrouping 100 receivers and 100 sources into 16 channels

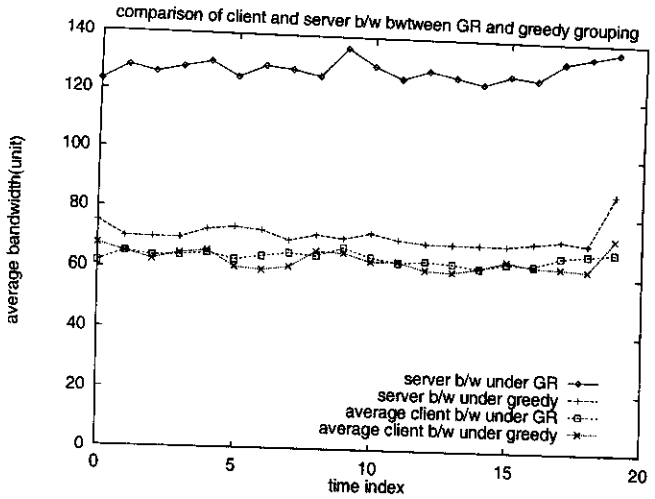
all receivers rather than finding a feasible group arrangement that satisfies given resource conditions, it is hard to compare its performance and that of ours. But it would be interesting to make a comparison between the two by having a close-up look into their underlying methodologies.

We focus on the task of channel splitting, which is common in both algorithm. In Fig. 2, this task is carried by the code segment "Connection partition for channel C_i", which make use of the similarity among connections. In [3], this task is a done by clustering receivers(sources) based on the similarity among receivers(sources).It has been shown in [3] that receiver-based clustering is suitable for multiplayer gaming and source-based clustering is suitable for stock quote streaming. However, both similarity among receivers and that among sources are only approximations of similarity among connections, and therefore they are somewhat less accurate compare to latter, especially when an application does not have constant preference pattern or a pattern that can fit into either multiplayer gaming or stock quote streaming.

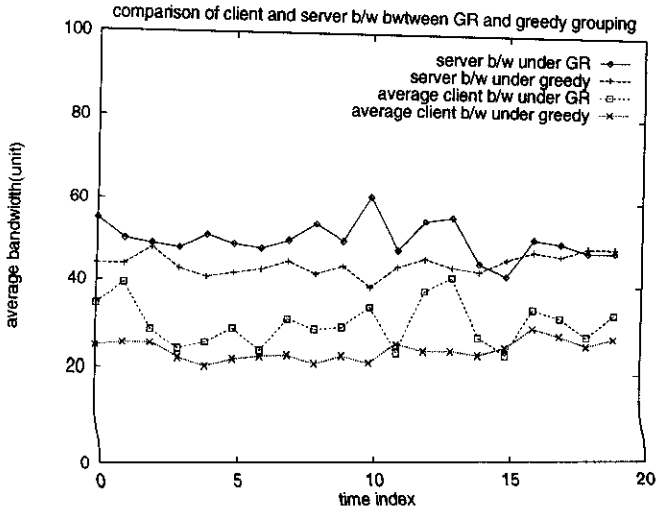
To highlight the difference between connection similarity and receiver(source) similarity, we split a channel with receiver-based clustering using the k-means package in MATLAB and compare the result with that of our algorithm. We examine three cases. In first case, all sources have same brightness and all receivers have same vision domain size and bandwidth capacity. As shown in Fig 8(a), two algorithm bring similar amount of incoming traffic at receivers. In second case, all sources have same brightness but there is large variation in receivers' vision domain size and bandwidth capacity. As shown in Fig 8(b), the two algorithms bring similar amount of incoming traffic at the receivers but less outgoing traffic at the server. In the last case, there are large variance in sources' brightness and receivers' vision domain size and bandwidth capacity. As shown in Fig 8(c), the latter algorithm brings less receiver incoming traffic and less server outgoing traffic.



(a) Case 1



(b) Case 2



(c) Case 3

Fig. 8. Algorithm Using Connection Similarity(greedy) Versus Algorithm Using Receiver Similarity(GR)

VI. Conclusion

We have presented an incremental and adaptive heuristic-based algorithm that can group sources and receivers in a way that it handles limited network resources. We have shown through experiments that our algorithm can produce better results than several algorithms that have been developed in the past for update dissemination.

Since our algorithm considers individual receivers' bandwidth capacity, it can migrate bad-waste from overloaded receivers to the receivers that are not overloaded thereby better preserves the bandwidth of low-end users. Also, since a new feasible group arrangement is incrementally derived from a previous feasible one, number of joins and leaves of multicast channels is reduced. In addition, our algorithm allows both receivers and sources to subscribe to multiple channels and therefore are suitable for a wide range of applcations.

References

[1] BALLARDIE, RANCIS, and CROWCROFT, "Core based trees(cbt):an acritecture for scalable inter-domain multicast routing.," in *SIGCOMM*, 1993.

[2] DEERING, ESTRIN, FRAINACCI, and V. JACOBAON, "An architecure for wide-area multicast routing.," in *SIGCOMM*, 1994.

[3] T. Wong, R. Katz, and S. McCanne, "Evaluation of preference clustering in large-scale multicast applications," in *inforcom*, 2000.

[4] Vijaykumar Krishnaswamy, Ivan B. Ganv, Jaideep M. Dharap, and Mustaque Ahamad, "Distributed object implementations for interactive applications," in *Middleware 2000*, 2000.

[5] Carlos O'Ryan, Douglas C. Schmidt, and J.Rusell Noworthy, "Patterns and performancd of a corba event service for large-scale distributed interactive simulations," in *International Journal of Computer System s Science and engineering, CRL Publishing*, 2001.

[6] S. J. Rak and D. J. Van Hook, "Evaluation of grid-based relevance filtering for multicast group assignment," in *DIS workshop*, 1996.

[7] E. Lety, T. Turletti, and F. Baccelli, "Cell-based multicast grouping in large-scale virtual environments," in *INRIA*, 1996.

[8] Barrus Waters and D.B. anderson, "Locals and beacons: Efficient and precise support for large multi-uer virtual environments," in *IEEE*, mar 1996.

[9] James O. Calvin, Caol J. Chian, and Daniel J. Van Hook, "Data subscription," in *DIS Workshop*, 1995.

[10] K. L. Morse, L. Bic, and M. Dillencourt, "Interest management in large-scale virtual environments," , no. 1, pp. 52–68, feburary 2000.

[11] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barhan, and S. Zesitz, *A Network Software Architecture for Large-Scale Virtual Environments*, doctoral dissertation, Naval Postgraduate school, monterrey, CA, june 1995.

[12] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barhan, and S. Zesitz, "Npsnet: A multi-player 3d virtual environment over the internet.," in *Symposium on Interactive 3D Graphics*, 1995.

[13] L. Zhou, M. H. Ammar, and C. Diot, "An evaluation of grouping techniques for state dissemination in networked multi-user games," in *MASCOTS*, 2001.

[14] R. Macednia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham, "Exploiting reality with multicast group," in *Vurtual Reality Annual Intenational Symposium*, sep 1995.