Adaptive Client to Mirrored-Server Assignment for Massively Multiplayer Online Games[1]

Steven Daniel Webb, Sieteng Soh
Department of Computing, Curtin University of Technology
Kent Street, Bentley, Perth, Western Australia, +61 8 9266 7680
steven.webb@postgrad.curtin.edu.au, S.Soh@curtin.edu.au

ABSTRACT

The Mirrored Server (MS) architecture for network games uses multiple mirrored servers across multiple locations to alleviate the bandwidth bottleneck and to reduce the client-to-server delay time. Response time in MS can be reduced by optimally assigning clients to their mirrors. The goal of optimal client-to-mirror-assignment (CMA) is to achieve the minimum average client-to-mirror delay considering player joins (CMA-J) and leaves (CMA-L), and mirrors with limited capacity. The existing heuristic solution considers only CMA-J, and thus the average delay of the remaining players may increase when one or more players leave. Furthermore, the solution ignores mirror capacity, which may overload mirrors. In this paper we present a resource usage model for the MS architecture, and formally state the CMA problem. For both CMA-J and CMA-L we propose a polynomial time optimal solution and a faster heuristic algorithm that obtains near optimal CMA. Our simulations on randomly generated MS topologies show that our algorithms significantly reduce the average delay of the existing solution. We also compare the merits of the solutions in terms of their optimality and running time efficiency.

Keywords: Client/server, distributed servers, game delay, mirrored servers, MMOG, network game, peer-to-peer.

1. INTRODUCTION

Massively Multiplayer Online Games (MMOG) differ from traditional network games as they present a single universe in which thousands or tens of thousands of players participate simultaneously. Furthermore, these worlds are persistent; hence, the state of the world evolves even when the player is offline. Thus, in addition to addressing game consistency, responsiveness, and cheat-free requirements, one must also address game persistency, system scalability and reliability [13].

The vast majority of networked games use a Client/Server (C/S) architecture, in which the server is the game authority. With only one centralized trusted server, keeping the game consistent, persistent, and cheat free in C/S is straightforward [13]. Most MMOG use a distributed C/S architecture in which multiple servers located in a data centre simulate the virtual world [4,8,12]. The mirrored-server (MS) architecture [4] comprises multiple trusted servers (mirrors) deployed at geographically different locations connected via a private well-provisioned network. Each mirror has its own Internet connection, and each client sends every update to its local mirror which, in turn, multicasts it to all other mirrors. All mirrors simulate the game world based on all client updates, and therefore are able to resolve inconsistencies. Finally every mirror periodically sends updates to its clients. MS improves the bandwidth scalability and reliability of C/S. Since the mirrors are geographically distributed, the average delay between each player and his server is reduced. Improving game responsiveness is important since lower game delay increases player satisfaction [3,12].

Some players can be significantly closer to some mirrors than to others; therefore, to minimise the average game delay MS requires an effective client to mirror assignment (CMA) algorithm. References [4,12] propose a greedy solution in which

each joining client is assigned to its closest server. However, as players are not uniformly distributed in the real world, if many players are close to one of the mirrors, it may be overloaded. Furthermore, they [4,12] do not attempt to improve the assignment of connected players when one or more players leave, and thus the average delay of the remaining players may increase. A realistic CMA algorithm must consider mirrors with limited capacity. Furthermore, as the rate of players joining and leaving may be very high, the algorithm should minimize the average client-to-mirror delay in real time.

The remainder of the paper is organized as follows. In Sec. 2 we describe our system model and formulate the CMA problem. Sec. 3 describes our optimal algorithms for CMA and faster heuristic algorithms that obtain near optimal solutions. Then, in Sec. 4 we show the effectiveness and efficiency of our approaches. Sec. 5 concludes our paper.

2. PROBLEM FORMULATION

2.1 System model

We consider an MS architecture [4,12] in which mirrors $M=\{M_1, M_2, \ldots, M_m\}$ are distributed at geographically different locations, and connected to each other via a well provisioned (high bandwidth, low delay, lossless) network. Every mirror simulates the entire virtual world, and runs a synchronization mechanism [4,12] to maintain consistency amongst mirrors. A mirror is a data centre comprising multiple servers. Due to the cost overhead of commissioning a data centre the game publisher provisions a small number of centres [8], each containing hundreds of servers. The tasks to simulate the virtual world are distributed and load balanced between servers within each centre. Since distributing and balancing the virtual world is game specific, they are beyond the scope of our work.

We consider a set of n clients/players $P=\{P_1, P_2, \ldots, P_n\}$ each of which is connected to a mirror. We assume each player equally requires b bandwidth and L_s+L_c processing resources of the mirror. Note that L_s (L_c) is the processing required to simulate the virtual world (to send and receive updates, and to perform AoI filtering). L_c is a significant portion of a server's workload [1]. Clients require equal resources because: (i) many games require clients to produce updates at fixed intervals [2,9,12]; (ii) averaged over many updates the resources required for each update is approximately equal.

Every mirror M_j provides fixed bandwidth α_j and processing power β_j. Therefore, M_j can support a fixed number of players; we call this number the mirror capacity C_j. For bandwidth, since every player requires b units, M_j can support α_j/b clients. From the processing perspective, every mirror requires at least $n*L_S$ units of processing power to simulate the virtual world involving all players in P; thus, M_j can support $(\beta_j-(n*L_S))/L_C$ players. Therefore, the capacity of M_j is calculated as $C_j = min\ \{(\alpha_j/b), ((\beta_j-(n*L_S)/L_C)\}$. Note that M_j is saturated when it has more than C_j connected players, and the game experience users receive rapidly deteriorates. It is obvious that players must not connect to mirrors at full capacity.

We define the delay $d_{i,j}$ ($d_{j,i}$) as the difference between the time a mirror M_j (client P_i) receives an update and the time the update was sent by client P_i (mirror M_j); we assume $d_{i,j} = d_{j,i}$. The delay experienced by a player has considerable impact on his game enjoyment [3]; therefore, it is desirable to minimise the delay. Note that in the MS architecture there is also network delay between mirrors; however, as this is not affected by the player assignment we do not consider it in our delay model.

2.2 CMA Problem statement

Let DM_n denote a delay matrix of size n x m for all clients in P and mirrors in M. An element in row i and column j of DM_n represents delay $d_{i,j}$ for P_i and M_j, where $i \leq n$ and $j \leq m$. Let CMA_n be a set of all client-to-mirror assignments for DM_n. For $i=1,2,\ldots,n$ and $j=1,2,\ldots,m$, we define CMA_n to be optimal if $D = \Sigma\ d_{i,j} * x_{i,j}$ is minimized, subject to constraints (i) $\Sigma\ x_{i,j} = n$, and (ii) $\Sigma\ x_{i,j} \leq C_j$, where $x_{i,j} \in \{0,1\}$ is equal to 1 (0) if P_i is (is not) assigned to M_j. Constraint (i) ensures that every player is assigned to a mirror, and constraint (ii) avoids saturating a mirror. We assume fixed M and dynamic P (due to player joins and leaves). The CMA-J problem is to construct an optimal CMA_{n+1}, from an optimal CMA_n, that includes the assignment of a joining P_i. Similarly, the CMA-L problem is to construct an optimal CMA_{n-1} which excludes the assignment of a leaving P_i.

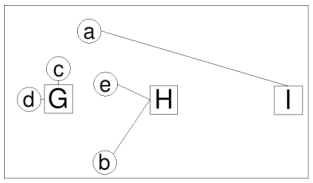
Both CMA-J and CMA-L may require re-assigning players to different mirrors to achieve the optimal delay. As every mirror stores and simulates the entire virtual world, MS architectures are well suited to transfer players between mirrors, as no game state must be passed. We believe the benefits of transferring players to improve the average game responsiveness far out-weight the costs as the bandwidth and processing required to perform a hand off is far below that required to process, simulate, and send updates about the game. Furthermore, our randomised simulations in Sec. 4 show that the average number

of transferring clients is minimal (0.6 to 2 on average for each client join/leave). The responsibility of re-assigning players to mirrors belongs to the authentication server as it stores the delays of every player to mirror.

Fig. 1 shows the CMA for three mirrors: *G*, *H*, and *I*, and five clients: *a*, *b*, *c*, *d*, and *e* with C_G=C_H=C_I=2. Each number in Fig. 1 represents the delay from each client to every mirror. Fig. 1(b) shows the optimal CMA_5 for DM_5; the cost of each client assignment is shown in bold in Fig. 1(a). The total, average, and maximum delays of the solution are 21, 4.2, and 10 respectively. For the joining player f with $d_{f,G}$=2, $d_{f,H}$=4, and $d_{f,I}$=10, the CMA-J problem is to obtain CMA_6 from CMA_5 that minimizes delay *D* subject to constraints (i) and (ii). Fig. 1(c) shows the updated DM_6 and Fig. 1(d) gives the optimal CMA_6. The optimal result requires reassigning client *b* from *H* to *I*, and assigning *f* to *H*. Similarly, considering the optimal CMA_6 in Fig. 1(d), and leaving player *f*, the solution to the CMA-L problem results in the assignment in Fig. 1(b).

	G	*H*	*I*
a	4	5	**10**
b	4	**5**	10
c	**1**	8	10
d	**1**	8	10
e	2	**4**	10

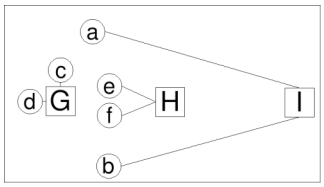
(a) Delay matrix DM_5



(b) Optimal CMA_5 for DM_5

	G	*H*	*I*
a	4	5	**10**
b	4	5	**10**
c	**1**	8	10
d	**1**	8	10
e	2	**4**	10
f	2	**4**	10

(c) Delay matrix DM_6 with joining *f*



(d) Optimal CMA_6 for DM_6

Fig. 1. Example client and mirror configuration for CMA problem

2.3 Related Problems

The clients to servers assignment (CSA) problem for web servers and DNS systems [5,6] is similar to CMA; however, CSA assumes short lived sessions. In contrast, a playing session in a network game may last for tens of minutes to several hours, and thus the solutions to the CSA may not be applicable to the CMA. The solutions [6] assume either the use of anycast (which is not widely available), and/or running as a distributed program (which is inappropriate for our centralized model).

In [9], each mirrored server simulates only a portion of the virtual world, in contrast to ours in which each mirror simulates the entire world. Thus, their model assumes a larger number of mirrors, each serving fewer clients than those in our model. Their goals: to maximise the number of clients that can connect within the delay bound, and to minimise the mirror processing requirements by reducing the number of mirrors simulating each region, are different to ours.

Reference [10] divides the virtual world into zones containing interacting clients, and considers a set of geographically distributed and well-connected servers. Every zone is designated a *target* server, responsible for simulating all events in the zone. Each server may simulate multiple zones but has limited CPU capacity. The *contact* server for a client is the server it is connected to. If a client's contact and target servers are different, the contact server is responsible for forwarding updates between the client and the target server. Let D^S be the maximum round-trip client-to-server delay of all clients in the zone, and *D* be the required game delay requirement. They [10] attempt to maximize the number of clients whose $D^S \leq D$. Their work is to find clients-to-servers assignment subject to delay *D* and CPU capacity constraints, which is different from ours in two areas. First, they assume that each mirror is a single server that simulates the virtual world zones. Due to the cost overhead at each location [8], we believe that assuming each mirror with multiple servers that simulates the entire world is more realistic. Second, they attempt to maximize the number of players whose game delay is less than the required delay, and to minimize the server processing requirements. In contrast, we attempt to minimize the average delay assuming each mirror can support a bounded number of players. We believe that minimizing the average delay implicitly maximizes the number of peers meeting the delay requirement. Note, their solutions do not consider the dynamic nature of the system that must address, in real time, new client joins, existing client leaves, and clients that move from one zone to another.

3. CLIENT TO MIRROR ASSIGNMENT ALGORITHMS

As in [12] we assume the existence of an Authentication Server (AS) to verify client identities. For CMA-J, the AS verifies the joining player for subscription, banning, *etc*., and then transmits the list of mirrors to the client. The client uses *echo*

messages to calculate the delay to each mirror and transmits the delays to the AS. The AS uses a CMA-J algorithm to determine the optimal mirror for the client. The AS notifies the client and may ask other clients to transfer to a different mirror. For each leaving player, the AS uses a CMA-L algorithm to determine the optimal client-to-mirror assignments for the remaining players. The AS may ask some remaining clients to transfer to a different mirror.

3.1 CMA-J Algorithms

We propose two CMA-J algorithms: J-SA, and J-greedy. Our J-SA considers the CMA-J as a special case of the Terminal Assignment (TA) problem [7]. The TA problem determines the assignment of each terminal to a concentrator such that no concentrator exceeds its capacity and the overall system delay is minimised; this problem has been shown NP-complete [10]. Note that the TA problem assumes fixed numbers of concentrators and terminals and they are known in advance, which is different to ours as players join and leave the game without warning. In our model, every client (terminal in TA) has equal weight since we assume that every player consumes the same amount of bandwidth and processing power from its mirror (concentrator in TA). This special case can be optimally solved in polynomial time using either the SA [11] or the AC [7] algorithms. We use SA for our J-SA since the AC algorithm is not well suited for CMA-L.

Fig. 2 shows the J-SA algorithm. There are two cases of player join. In the first case the closest mirror M_j to the joining player P_i has spare capacity; therefore, P_i is assigned to M_j. In the second case M_j is full, and the optimal assignment may require a sequence of player re-assignments (a chain). The minimum chain and its cost is calculated as follows. First J-SA generates two arrays MC and MP, each of size $m \times m$, where MC[j,k] stores the minimum cost of re-assigning a player from M_j to M_k. MP[j,k] stores the ID of the re-assigned player corresponding to MC[j,k]. Second, an array Label of size m is generated, and stores the delay cost of the minimum chain for each mirror. The minimum chain is the sequence of client re-assignments with the lowest total delay. Third, the minimum cost of assigning the new player is calculated from Label. If the cost of connecting P_i to a non-full mirror M_j is equal to the minimum chain then P_i is assigned to M_j; else, the minimum chain of re-assignments is performed using Label to determine the sequence of mirrors and MP to determine which clients to move. Note that our J-SA is the implementation of SA for CMA, and thus its complexity is given as $O(mn)$ [11].

Even though the J-SA algorithm produces an optimal assignment in polynomial time it may still prove excessively slow for MMOG that involve millions of players. Our J-greedy is an extension to the solution in [4,12] which produces near-optimal results, and is much faster than J-SA. However, as observed in [7], if the system is close to full capacity this greedy algorithm will produce poor results, as latter joining players are assigned to mirrors with high delays. Fig. 2 shows the J-greedy algorithm that is faster than J-SA and is able to produce nearly optimal client assignments. The greedy algorithm is an extension to [4,12] that considers mirror capacity. J-greedy sorts all mirrors in increasing delay from the joining player, and greedily searches for the closest mirror with spare capacity. The greedy heuristic is the delay to each mirror. The time complexity of J-greedy is $O(m \log m)$ due to sorting the mirrors in increasing order of the delay to the joining player. Note that if all mirrors have spare capacity J-greedy will produce optimal results; however, the closer the system is to full the further from optimal the assignment will be.

3.2 CMA-L Algorithms

To obtain optimal delays for CMA-L, one may use an extremely slow brute-force approach that runs the J-SA on each of the remaining players. Note, the SA algorithm in [11] cannot be directly used to solve our CMA-L. Therefore, we propose two novel algorithms: L-SA and L-greedy. L-SA produced optimal results in our simulations in Sec. 4, and the L-greedy produces near optimal results, and on average is much faster than L-SA.

L-SA in Fig. 2 is an $O(mn)$ algorithm that utilizes some of the properties of J-SA. It considers two cases for player leave. If the mirror M_j assigned to a leaving algorithm player P_i has spare capacity, no re-assignments of players are required since the CMA remains optimal. If M_j is full, a chain of player re-assignments may be required to reach the optimal configuration. The minimum chain and its cost are calculated as follows. First L-SA generates two arrays MC and MP, each of size $m \times m$, where MC[j,k] stores the minimum cost of re-assigning a player from M_j to M_k. MP[j,k] stores the ID of the re-assigned player corresponding to MC[j,k]. Note that MC[j,k] is negative if re-assigning MP[j,k] to M_k reduces the player's delay. Second, an array Label of size m is generated, and stores the cost of the minimum chain for each mirror. Initially each Label[j] is a pair (*delay*, *mirror*) where *delay* is the delay reduction of the chain and *mirror* is the next mirror in the chain. Each label is initialised to (∞, -) if the mirror is full and (0, -) if the mirror has spare capacity. The merit of the optimal chain

is calculated by passing through all of the labels, updating them when a lower delay chain is found. This is repeated until no more labels are updated. Third, the end of the chain is the label with the minimum delay. If the delay of the chain is less than zero the chain of re-assignments is performed; else the chain is not performed as there is no benefit from re-assigning any players. The sequence of mirrors and players for the chain are determined from the Label and MP arrays, respectively.

```
Algorithm: J-SA
Imports: P_i - The joining player
         cost – 1D array of the d_{i,j} between P_i and each M_j.
begin
  j = min(cost)        //index of the closest mirror
  if (M_j is not full) then assign P_i to M_j
  else  generate MC, MP, and Label
    while (labels were changed) do
      for (every M_j∈M where M_j is at full capacity) do
        for (every M_k∈M where k ≠ j) do
          if (Label[k].delay + MC[j][k] < Label[j].delay) then
            Label[j].delay = Label[k].delay + MC[j][k]
            Label[j].mirror = k
    min = calculate the cost of the minimum chain
    if ( ∃ M_j with spare capacity and cost[j] = min) then
      assign P_i to M_j
    else
      M_f = mirror at the end of the chain    //mirror from
      assign joining P_i to M_f
      repeat
        M_t = Label[f].mirror  //mirror to
        i = MP[f][t]
        re-assign P_i to M_t
        f = t
      until (M_f does not exceed its capacity)
end

Algorithm: J-greedy algorithm
Imports: P_i - The joining player
         cost – 1D array of the d_{i,j} between P_i and each M_j.
begin
  sort cost by delay in ascending order.
  for (each j ∈ cost) do
    if (M_j has spare capacity) then
      assign P_i to M_j
      break
end
```

```
Algorithm: L-SA
Imports: P_i - the leaving player
begin
  M_j = P_i's assigned mirror
  if (M_j is not at full capacity) then remove P_i
  else
    generate MC, MP, and Label
    while (labels where changed) do
      for (every M_j ∈ M) do
        for (every M_k ∈ M where k ≠ j) do
          if (Label[k].delay + MC[j][k] < Label[j].delay) then
            Label[j].delay = Label[k].delay + MC[j][k]
            Label[j].mirror = k
    minIndex = min(Label)    //index of the smallest chain
    if (Label[minIndex].delay < 0) then
      f = minIndex              //get the from mirror
      t = Label[f].mirror       //get the to mirror
      while (t ≠ j) do
        t = Label[f]
        i = MP[f][t]
        re-assign P_i to M_t
        f = t
end

Algorithm: L-greedy
Imports: P_i - the leaving player;    M_j - P_i's assigned mirror
begin
  if (M_j is not full) then remove P_i and update M_j's capacity
  else
    while (continue = true) do // initially continue = true
      find P_m   // player with max benefit from connecting to M_j
      b = d_{m,k} – d_{m,j}  // Let M_k be P_m's assigned mirror
      if (b >= 0) then continue = false
      else
        re-assign P_m to M_j
        j = k
        if (M_j is not full) then continue = false
end
```

Fig. 2. The join and leave algorithms

We demonstrate L-SA with an example. Assume that player *c* from Fig. 1(d) is leaving; the resulting DM_5 is shown in Fig. 3(a). The initial (non-optimal) assignment of players is shown in bold. Fig. 3(b) and 3(c) show MP and MC respectively. For example, MP[*G*,*H*]=7 is obtained from DM_5 by finding the player P_i assigned to *G* (P_d for this example) with the minimum $d_{i,H}$ - $d_{i,G}$; therefore, MC[*G*,*H*] is set to *d*. Initially the labels are: Label[*G*] = (0,-), Label[*H*] = (∞,-), Label[*I*] = (0,-). In the first iteration, Label[*G*] = (0,-) is not updated since there is no lower chain. Label[*H*] is calculated as the cost of the chain at *G* (0 from Label[*G*]) plus the cost of moving a player from *H* to *G* (MP[*H*][*G*]=-2); thus Label[*H*] is updated to (-2,*G*).

Similarly Label[*I*] is updated to (-7,*G*) as the chain at *H* is -2, and player *a* can be moved to *H* with a delay reduction of -5. The second iteration does not change the labels therefore the chains are complete. As Label[*I*] has the minimum delay the chain begins at *I*, and the sequence re-assignments is: *a* to *H*, *e* to *G*.

	G	*H*	*I*
a	4	5	**10**
b	4	5	**10**
d	**1**	8	10
e	2	**4**	10
f	2	**4**	10

(a) DM_5 after removing player *c*

	G	*H*	*I*
G	0	7	9
H	-2	0	6
I	-6	-5	0

(b) MP

	G	*H*	*I*
G	-	*d*	*D*
H	*e*	-	*E*
I	*a*	*a*	-

(c) MC

Fig. 3. Example of CMA-L using L-SA.

The L-greedy heuristic re-assigns the player that will give the maximum reduction to the total system delay. As shown in Fig. 2, assuming a leaving player P_i, L-greedy searches for the client that will have the greatest decrease in delay if re-assigned to P_i's mirror. This is repeated for each re-assignment until a client is re-assigned from a non-full mirror or there is no client that will benefit from being re-assigned. The worst case runtime complexity is $O(mn)$; however, in practice we believe the average runtime is much lower. Note, the CMA solution that dynamically allows players joining and leaving must comprise both a CMA-J and CMA-L. Notice that J-SA (L-SA) requires optimal CMA_n to produce CMA_{n+1} (CMA_{n-1}), and therefore cannot be combined with any of the non-optimal CMA-L (CMA-J) algorithms.

4. PERFORMANCE EVALUATION

We considered three scenarios: player join, player leave, and dynamic (join and leave). For each of these we computed the average and maximum delay from all players in the system, and recorded the time taken to run the simulation. We randomly generated a set of mirrors and clients, and generated links with random delay between each client and every mirror. First, we used the BRITE Internet Topology Generator to obtain a topology of 5000 nodes as in [9] (50 AS domains derived by the Barabasi-Albert model and 100 nodes per AS derived by the Waxman model); every link is uniformly distributed in the delay range [0*ms*, 25*ms*] as in [10]. Then, we used Dijkstra's single source shortest path algorithm for every node to generate delay $d_{i,j}$ for every pair of nodes *i* and *j*. Finally, 1000 clients and 5 mirrors are randomly selected from the 5000 nodes for the join simulation and leave simulation, and 1300 clients and 5 mirrors for the dynamic simulation. For the simulation in Sec. 4.4, we used the same number of randomly selected clients from Sec. 4.1-4.3, but we randomly selected 5, 10, and 20 mirrors.

4.1 Player Join

We compare the average and maximum delays generated by the J-greedy and J-SA algorithms. We consider five mirrors with capacities 100, 150, 200, 250, and 300; thus the system supports 1000 players. We iteratively used each algorithm to assign every player, and for each assignment we compute the average and maximum game delays. As shown in Fig. 4, both algorithms produce optimal results when no mirror has reached its capacity (less than 175 joins). With one or more saturated mirrors, J-greedy does not produce optimal results; the closer the system is to full capacity the worse the performance of J-greedy. The figure shows that for high system load (>80%) the delays are considerably higher for greedy than SA. Fig. 5 shows the Cumulative Distribution Function (CDF) of player delays after 1000 joins. The figure shows that for optimal CMA (*i.e.*, using J-SA) more players have lower delays, and assuming the game delay requirement of 100ms, more players assigned using J-SA can meet the requirement than those using J-greedy (949 vs. 869 players).

4.2 Player Leave

The solution implicitly implied in [4,12] (we call it L-ignore) does not attempt to improve the assignment of the remaining connected clients that may be poorly assigned. We compare the resulting average and maximum delays generated by the L-ignore, L-greedy, and L-SA algorithms. We used a brute-force (BF) approach that obtains optimal delay to gauge their optimality. Note that for BF after each player leave, the J-SA algorithm is used repeatedly to construct the optimal assignment for the remaining clients. For L-greedy and L-ignore (L-SA) we assume the system has been populated with 1000

players using J-greedy (J-SA) as described in Sec. 4.1. Every player leaves in the order in which they joined and the average and maximum delays are calculated. This order follows the assumption that the player with the longest session is the most likely to leave (FIFO). The results are shown in Fig. 6. Note that the delays obtained by the L-SA algorithm match those by BF, and therefore we conjecture that L-SA produces optimal results. From the figure the L-greedy algorithm's average and maximum delays are close to optimal except when the system is near full capacity. However, the J-greedy/L-ignore approach, as implicitly proposed in [4,12], results in significantly worst average and maximum delays.

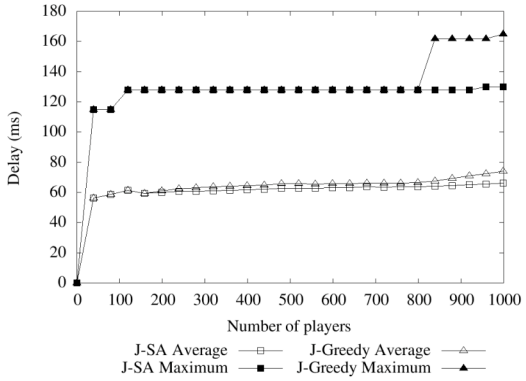


Fig. 4. Average and maximum delays for player-join case

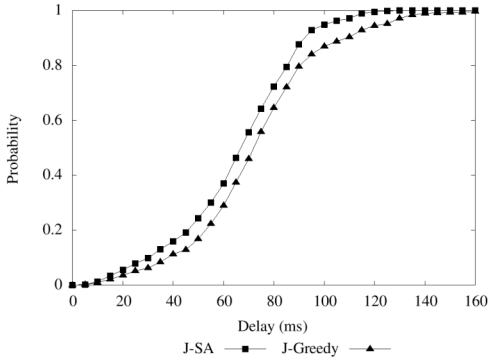


Fig. 5. CDF for player-join case

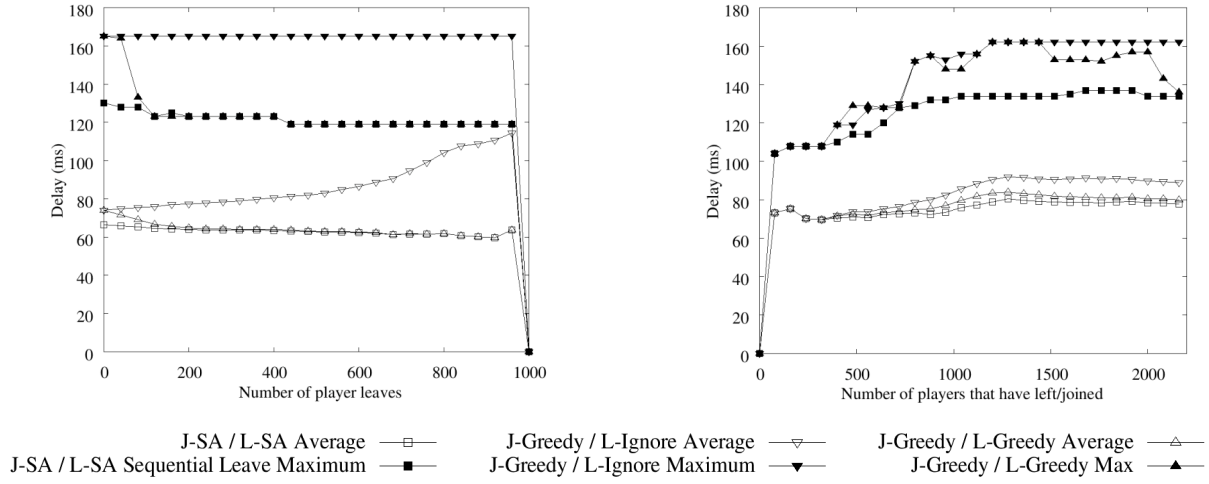


Fig. 6. Average and maximum delays for player-leave case

Fig. 7. Average and maximum delays for dynamic case

4.3 Dynamic Player Join and Leave

This section considers a player may join or leave the game at any time. We consider 5 mirrors with capacities 50, 75, 100, 125, and 150. We first randomly populated the system to 400 players by repeatedly using each of the CMA-J algorithms to insert two players consecutively and each of the CMA-L algorithms to remove a player randomly (0 to 1200 in Fig. 7). Then, we simulated 500 joins and 500 leaves in a random order using each of the join and leave algorithms (1200 to 2200 in Fig. 7). We considered all three possible combinations: J-SA/L-SA, J-greedy/L-greedy, and J-greedy/L-ignore. Note that the SA algorithms assume optimal CMA as input, and therefore they cannot be combined with non-optimal CMA algorithms. Note that the J-greedy/L-ignore scheme is the client assignment method used currently for mirrored architectures [4,12]. In Fig. 7,

the average delay generated by J-greedy/L-greedy are far closer to the optimal delay (J-SA/L-SA) than that obtained by J-greedy/L-ignore. However, the maximum delays generated by the J-SA/L-SA are significantly better than those of the others.

4.4 Speed comparisons

Columns 2 of Fig. 8 compare the speed of the various CMA-J and CMA-L algorithms for generating the results in Sec. 4.1 to 4.3. Consistent with their time complexities, J-greedy is faster than J-SA. Further, although their time complexities are equal, L-greedy runs faster than L-SA, as the worst case complexity for L-greedy rarely occurs. Due to the small time difference between L-ignore and L-greedy, we believe the latter is superior as it produces lower client delays. If the system is expected to be close to capacity, we recommend using J-SA and L-SA. We repeated the simulations in Sec. 4.1 to 4.3, but with 10 and 20 mirrors while maintaining the same number of clients and total system capacity (each mirror's capacity is reduced proportionately). The results show that the number of mirrors had little impact on the algorithms, except for J-SA and L-SA.

	Mirrors		
	5	**10**	**20**
J-greedy	12	16	15
J-SA	688	3438	16519

(a) Player Join

	Mirrors		
	5	**10**	**20**
L-ignore	4	3	3
L-greedy	43	50	60
L-SA	888	4683	20077

(b) Player Leave

	Mirrors		
	5	**10**	**20**
J-greedy/L-ignore	19	21	26
J-greedy/L-greedy	52	55	60
J-SA/L-SA	1542	5053	22769

(c) Dynamic Player join and leave

Fig. 8. Algorithms running time (ms)

We have formally defined the CMA problem and proposed an optimal solution (J-SA/L-SA) and a faster heuristic solution (J-greedy/L-greedy). We have shown that both solutions produce significantly lower average and maximum client delays than the existing scheme (J-greedy/L-ignore). To maintain low average and maximum delays when the system is nearly full (above 85%) we recommend using the slower, but optimal, J-SA/L-SA as J-greedy/L-greedy performs poorly at high capacity. We have shown that our J-greedy/L-greedy runs almost as fast as J-greedy/L-ignore, while producing near-optimal results. We believe that the J-SA and L-SA algorithms can be implemented more efficiently if arrays MP and MC can be dynamically updated, rather than completely rebuilt for each player join/leave. We also plan to modify the AC algorithm [7] to support CMA-L and compare the performance with J-SA/L-SA.

REFERENCES

[1] Abdelkhalek, A., Bilas, A., & Moshovos, A. *Behaviour and performance of interactive multiplayer game servers*. Special Issue of Cluster Computing: the Journal of Networks, Software tools and applications, 2002.

[2] Armitage, G. *An experimental estimation of latency sensitivity in multiplayer Quake 3*. ICON 2003, pp. 137-141.

[3] Beigbeder, T., Coughlan, R., Lusher, C., & Plunkett, J. *The effects of loss and latency on user performance in Unreal Tournament 2003*. In Proc. SIGCOMM '04 Workshops, pp. 144-151.

[4] Cronin, E., Kurn, A., Filstrup, B., & Jamin, S. An *efficient synchronization mechanism for mirrored game architectures*. Multimedia Tools and Applications 23, **1** (2004), pp. 7-30.

[5] Crovella, M., & Carter, R. *Dynamic server selection in the Internet*. HPCS '95, pp. 158-162.

[6] Fei, Z., Bhattacharjee, S., Zegura, E., & Ammar, M. *A novel server selection technique for improving the response time of a replicated service*. INFOCOM '98, pp. 783-791.

[7] Kershenbaum, A., *Telecommunication Network Design Algorithms*, McGraw-Hill, 1993.

[8] Kushner, D. *Engineering EverQuest: online gaming demands heavyweight data centers*. IEEE Spectrum 42, **7** (2005), pp. 34-39.

[9] Lee, K., Ko, B., Calo, S. *Adaptive Server Selection for Large Scale Interactive Online Games*. Computer Networks 49, **1** (2005), pp. 84-102.

[10] Ta, D., Zhou, S., & Shen, H. *Greedy Algorithms for Client Assignment in Large-Scale Distributed Virtual Environments*. IEEE PADS '06, pp. 103-110.

[11] Tang, D.T., Woo, L.S., & Bahl, L.R., *Optimization of teleprocessing networks with concentrators and multiconnected terminals*. IEEE Trans. Computers, vol. C-27, no. 7, July 1978, pp. 594-604.

[12] Webb, S., Soh, S., & Lau, W. *Enhanced Mirrored Servers for Network Games*. ACM Netgames '07, pp. 117-122.

[13] Webb, S., Soh, S., & Lau, W. *RACS: a Referee Anti-Cheat Scheme for P2P gaming*. NOSSDAV '07, pp. 37-42.