RECURSIVE INTEREST MANAGEMENT FOR ONLINE GAMES

Pawan Kumar and Qasim Mehdi

School of Computing and Information Technology
University of Wolverhampton
Wolverhampton, UK WV1 1SB
{pawan.kumar, q.h.mehdi}@wlv.ac.uk

KEYWORDS

Multi-player online games, interest management, HLA, data distribution management, multi-dimensional routing spaces

ABSTRACT

Performance and scalability in multi-player online games and distributed simulators mainly depends on the effectiveness of the deployed interest management schemes. These schemes aim at providing message-filtering mechanisms that reduces the communication overheads. However, in order to do so, they incur computational costs that are quite significant and are not suitable for scalable real time systems. In this paper, a recursive algorithm for interest management is presented that can be applied for systems that use multi-dimensional routing spaces for interest management. The algorithm's simulation shows that it is more efficient and scalable than existing approaches.

1. INTRODUCTION

Networked games allow multiple players at geographically dispersed location to share and play in a common virtual world. Typically, a player's node will contain some subset of the shared virtual world whose state is influenced and maintained by the player. In order to have a mutual consistent view of the virtual world, events or messages are exchanged between player nodes (either directly or indirectly through a server) [1]. However, an update occurring at one node is likely to have an immediate significance for only a subset of other nodes in the system. The techniques that exploit this *interest* of each node to minimise the number of messages sent are referred as *interest management* (IM) schemes.

Interest management systems have been incorporated in several large-scale distributed simulators [2,3], collaborative virtual environments [4,5,6] and multiplayer online games [7,8,9]. These have been incorporated mainly to allow systems to scale seamlessly and efficiently. The scalability in these systems is primarily related to the number of nodes that can participate and the computational complexity of the model that is being simulated (e.g. in a game it could be the number of entities the game has, etc). Without the IM system, it would entail every update or state changes at one node to be communicated to all the other nodes. This could significantly increase the bandwidth usage, message sent per second and computational requirements at processing these messages. However, incorporating IM system would try to minimise the above at the expense of computational costs for it's processing and thus affecting the real-time requirements of these systems and degrading performance. Thus, performance and scalability of these systems mainly depend on the effectiveness of the deployed IM scheme in these systems.

Several IM schemes have been adopted in which a node expresses its interest to some subset of the world. Only information that is pertinent to the node gets forwarded to the node. The filtering techniques used to achieve this can be *grid-based* [4,5,10], where the world is partitioned into an n-dimensional grid cell. Each node subscribes to some set of cells and updates are sent only between nodes whose subscriptions fall into the same grid cell. The advantage of this scheme is in its simplicity by statically partitioning the world in advance and each grid cell can be associated with a multi-cast address. However, issues of cell granularity can either result in imprecise filtering (for coarse grained) or significant overheads of leaving and joining multicast groups (for fine grained). Another filtering mechanism is based on *class based* filtering [3,11], where nodes express interest through subscription to some set of classes and send/receive updates to only those classes. This scheme is quite powerful and is used along with other filtering schemes [3]. However, in its entirety it is not sufficient for scalable systems. Further, there is *region-based* scheme [3,6,9], where simulation entities or nodes specify interest areas in the form of publisher and subscriber regions where intersection between them represents a potential communication between the entities. The regions can be specified as homogenous multi-dimensional routing space [3,9] or as auras [6]. The region-based scheme is more powerful and general as it allows each node to specify, create and modify the regions at run time. However, this expressivity does come with a significant overhead that could lead to a time complexity of $O(n^2)$. In this paper, a recursive algorithm (order O(nlogn)) for region matching is proposed where regions are defined using multi-dimensional routing space (Figure 1). Several other techniques for IM includes use of N trees [12] for massively multiplayer online games, sphere of influence [13] for distributed agent simulation and use

of message oriented middleware technologies [14] for networked games. However, their discussion is beyond the scope of the paper.

The remainder of this paper is organised as follows. Section 2 provides a background of existing approaches to region matching for multi-dimensional routing spaces. In section 3, the recursive algorithm for region matching is presented along with its complexity analysis. In section 4, simulation results and performance evaluation is provided. Finally, section 5 details conclusion and future work.

2. BACKGROUND AND RELATED WORKS

Much of the works of interest management using multi-dimensional routing spaces has been undertaken in the context of High Level Architecture (HLA) [3,16] and similar concepts have been applied in multiplayer online games middleware [9]. HLA is a standard interface specification that provides a general infrastructure and services for distributed simulation. Two of its services namely, *Declaration management* (DM) and *Data distribution management* (DDM) offers IM facilities. While the DM service provides IM using *class-based* approaches, the DDM service provides IM based on *region matching* where publisher and subscriber regions are specified using routing space information. A *routing space* is defined as a collection of dimensions that are used to define regions. Typically, a region comprises of set of *extents* where each extent has a *bounded range* defined along each dimension of the routing space as shown in (Figure 1). For IM, node's specifies object attributes or interactions that it wishes to publish or subscribe (using DM) and associate regions (extents) with those subscriptions (using DDM). When publisher regions overlap with subscriber regions, connectivity is established between the two nodes and finally the information is transmitted. Thus for successful implementation of IM, it is required that region matching and connectivity be established as efficiently as possible. Depending on the context of usage, several algorithms have been established for DDM. These are discussed next.

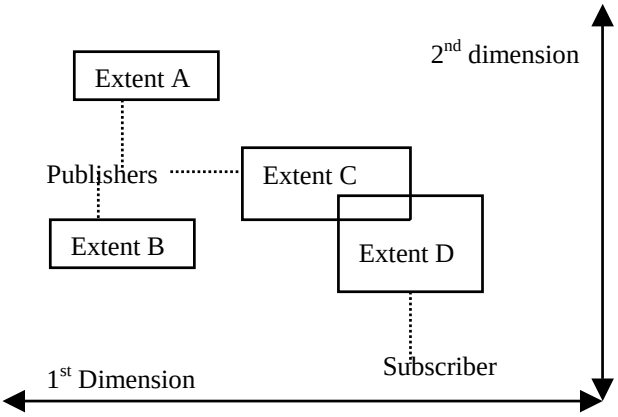

Figure 1: Extents in 2-dimensional routing space

2.1. Brute Force Approach

Brute force approach for region matching simply checks each of the publisher extents with each of the subscriber extents for an overlap. This results in a complexity of $O(n^2)$. The advantage of this approach is in its simplicity and performs well in situations when most of the publisher extents are overlapping with the subscriber extents, as probability of getting a matching pair early is high. However, this algorithm does not scale well except in situations where high number of regions overlap.

2.2. Grid Based Approaches

Grid based approaches can be utilised for DDM implementation as it completely eliminates the complexity associated with brute force approach. In this, routing spaces are partitioned into grid of cells and extents are mapped to some subset of the cells. Whenever, publisher extents and subscriber extents overlap with the same grid cell, they are assumed to be overlapping with each other. A prior application specific knowledge of the world being simulated is required for efficient realization of this approach. One such implementation partition the world into grid cells and statically assigns multicast address for each grid cell [10]. This approach is simple and scalable than the brute force approach. However, issues of cell granularity, as discussed in [15], can significantly affect the performance and resource usage of the IM system. A large cell size could lead to imprecise filtering, as extents may not be overlapping even though they are mapped to same grid cell. This would not only deliver additional data across the wire but also requires additional processing at the receiver to filter irrelevant data. On the other hand, a finer grid cell would require additional resources for maintaining data structures for each cell. Further, if multi-casting is used, then this could lead to thrashing in network layer where extents are modified frequently and are joining and leaving several multicast groups. In addition, there can be practical limitation of available multi-cast addresses [15] that could significantly affect the scalability of the systems. Thus, application specific knowledge, network bandwidth, resource availability are the key metrics for deciding an optimal cell size. In [17], multi resolution grids were used that were statically defined and were allocated multicast addresses. One section of the world uses coarse resolution while another section uses a fine resolution.

In addition to static approaches, more recent works uses dynamic grids [18] and hybrid approaches [19] as an extension to grid based approaches. In dynamic grids, multicast addresses are dynamically allocated to the cells that have at least one publishing extent and one subscribing extent. This results in efficient utilization of multi-cast address. Hybrid approach, on the other hand, combines brute force with grids. In this, world is partitioned into grid cells and then for all extents in each grid cell, brute force algorithm is used for region matching. This approach is more scalable

than a simple brute force approach and produces less irrelevant messages than grid based approaches. However, both still suffer from optimal grid size and application specific knowledge for their efficient implementation and have similar drawbacks as with static grids.

2.3. Spatial Partitioning Based Approaches

Like grid-based approaches, [16] suggested the use of spatial partitioning or hierarchical approaches for decomposing the world. In these, world can be partitioned using persistent spatial data structures such as quad-trees or oct-trees. These data structures provide efficient queries for searching. Depending upon usage, a node retrieved from a tree can have a multi-cast address associated per node or can have multiple extents that uses brute force for region matching. Further, extents can be stored at leaf nodes or at middle nodes and tree's depth can be statically fixed a priori or be allowed to grow dynamically. In addition to these issues, there are significant costs associated for storage, maintenance and balancing of these complex trees. Further, proper partitioning requires application specific knowledge.

2.4. Sort Based Approach

Recent work of [20] proposed a sort-based approach for region matching algorithm. In this, for each dimension in a routing space, a list of endpoints (representing coordinates of extents in that dimension) is created, sorted and examined in ascending order to obtain the overlap information in that dimension. Extents overlap if an overlap is found in all the dimensions. However, despite making improvements to their original algorithm and incorporating additional data structure for optimisation and intermediate storage of overlap information, the overall complexity still remained quadratic [20] and is therefore not scalable.

3. RECURSIVE ALGORTIHM FOR REGION MATCHING

All above discussed approaches have weaknesses and are not suitable for interest management in systems that demand performance and scalability. Our research aims at finding new approaches for interest management that uses multi-dimensional routing spaces and demand performance and scalability. Here we propose an efficient and scalable recursive algorithm for region matching that can be applied in these systems.

3.1. Problem Definition

Given

$$Sp = Space,$$
$$D(Sp) = Set\ of\ Dimensions\ for\ S$$

A region ***R*** comprises of set of extents and can be associated as a publisher or a subscriber. Therefore problem of region matching requires finding overlapping publisher and subscriber extents. Let

$$P = Set\ of\ Publisher\ Extents$$
$$S = Set\ of\ Subscriber\ Extents$$
$$R(x) = Set\ of\ Ranges\ for\ Extent\ x$$

Then we have

$$\forall P_i \in P : \exists R(P_i) \ \wedge \ \left|R(P_i)\right| = \left|D(Sp)\right|$$
$$\forall S_i \in S : \exists R(S_i) \ \wedge \ \left|R(S_i)\right| = \left|D(Sp)\right|$$

The problem here is to identify overlapping publisher and subscriber extents to establish connectivity between publishers and subscribers. The extents overlap if

$$\exists P_i \in P, \exists S_i \in S : Overlap(R(P_i), R(S_i))$$

Where $Overlap(R(x), R(y))$ is a predicate that is true if and only if

$$\forall i, j \ R_i \in R(x), R_j \in R(y) : R_i overlaps R_j \wedge i == j$$

i.e. each range of one extent overlaps with the corresponding range of the other extent.

For scalability, it is required that the above overlapping information of extents is obtained using minimal use of computational resources.

3.2. Algorithm Description

The algorithm for region matching takes the best of the existing approaches for region matching. In particular, this work extends the recent sort based approach to recursively do region matching and combines brute force approach when clusters of overlapping extents have been found. In order to understand recursive algorithm, a simple scenario is examined. Figure 1 shows four extents in a 2D routing space. Extents A, B and C belong to region that acts as publisher whereas extent D belongs to region that acts as subscriber. The first step in the algorithm identifies grouping of overlapping extents in one of the dimensions of the routing space. Figure 2 depicts this scenario. Here all the extents are projected to one of the dimensions. As seen, two groups can be obtained, one containing extents A and B and another containing extents C and D.

To algorithmically achieve this grouping information, a mechanism is required for evaluating the lower bound and upper bound points of each of the extents in that dimension. For this, a list of all endpoints in that dimension is *created* and *sorted* from lowest to highest

value. Figure 3 shows the list before and after sorting of Figure 1 extents in 1st dimension.

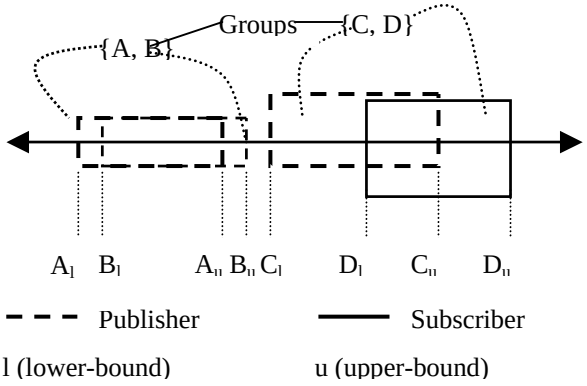

Figure 2: Extents projected to one of the dimensions

List = { A_l, A_u, B_l, B_u, C_l, C_u, D_l, D_u }

(a) List before sorting

List = { A_l, B_l, A_u, B_u, C_l, D_l, C_u, D_u }

(b) List after sorting

Figure 3: List of endpoints along one of the dimensions

Once sorting has completed, the algorithm proceeds to find the group of overlapping extents in that dimension. A container *group* comprising of two sets: one for holding references of publisher extents and another for holding references of subscriber extents, is used for storing the group of extents that are currently overlapping in that dimension. The algorithm scans for groups by sweeping along the dimension and examining endpoints as depicted in pseudo code below.

```
Input: Gp (group)
Gp.P = Publisher Extent Set
Gp.S = Subscriber Extent Set
L = sorted endpoints list along a dimension
count =0;
---
For each endpoint e in the list L
{
   if (e == lower bound){
      ++count;
       extent = getExtentRef(e);
       if (extent == publisher)
             Gp.addPublisher(extent);
       else // subscriber
             Gp.addSubscriber(extent);
   }
   else {// e is upper bound
        --count;
        if (count == 0){
             groupCompleted(Gp);
        }
        useGroup(Gp);
        Clear(Gp);
   }
}
```

Figure 4: Pseudo code for finding groups of overlapping extents along one dimension

For the above scenario, the pseudo code in Figure 4 will form two groups as shown in Table 1. These would be used further for recursive matching.

Step	End point	count	Group information (Gp)	
			Gp.P	Gp.S
1	A_l	1	{A}	{}
2	B_l	2	{A, B}	{}
3	A_u	1	{A, B}	{}
4	B_u	0	Completed	
5	C_l	1	{C}	{}
6	D_l	2	{C}	{D}
7	C_u	1	{C}	{D}
8	D_u	0	Completed	

Table 1: Iterations of grouping algorithm for the scenario of Figure 1

A special consideration to sort function is required while sorting the endpoints to consistently group the extents. This is depicted in Figure 5. Here extent A's upper bound coincides with the extent B's lower bound. Clearly the order in which they get sorted is important for proper grouping. If A_u occurs before B_l then extent A and B would incorrectly be placed in separate groups whereas if B_l occurs before A_u then both A and B occupies the same group. Therefore for proper grouping, the sort function is implemented as a binary predicate as depicted in Figure 6.

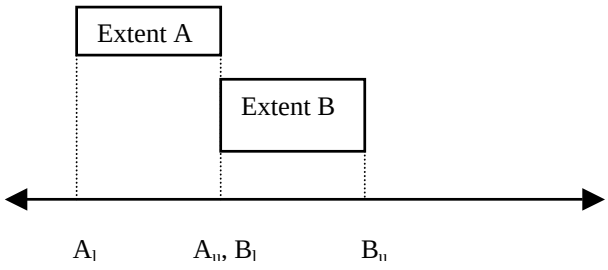

Figure 5: Extents projected along one dimension where sorting affects grouping

Input: $x \in l, y \in l$ $(l = list\ of\ endpoints)$

Output: Boolean

$$x < y\ if \begin{Bmatrix} x < y \\ (x == y) \wedge (x\ is\ lower) \wedge (y\ is\ upper) \end{Bmatrix}$$

Figure 6: Binary predicate used for sorting

The group formed in the previous step contains a reduced set of publisher extents and subscriber extents that may be overlapping along one of the dimensions. Thus a mechanism is needed to determine whether this reduced set of extents overlap or not. This is determined through recursion. Here, the groups found

in one dimension are sent along other dimensions to be examined in those dimensions. The same process is repeated for this group. A list of endpoints along that dimension is created; sorted and examined whether further sub-groups can be created or not. This process continues until there is no subdivision of the group or else the group size reaches a *threshold*, both of these used as a criterion for stopping recursion.

For our example scenario, the first group comprising of extents A and B will be sent along 2nd dimension that will result in two subgroups one containing A and the other containing B (Figure 7a). On the other hand, the second group comprising of extent C and D do not subdivide any further (Figure 7b).

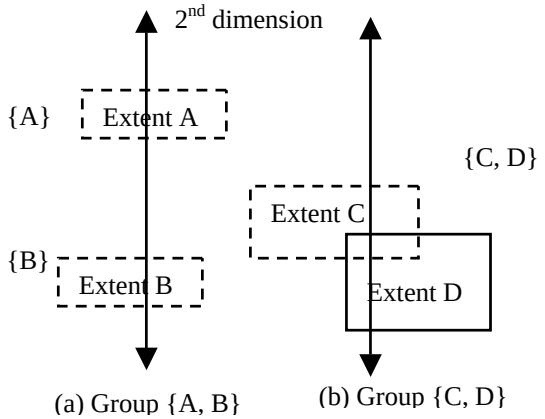

Figure 7: Groups of Figure 2 sent along second dimension

Here only three groups were created at the end of algorithm processing i.e. {A}, {B} and {C, D}. Their sizes are quite small and it is easy to identify overlapping extents. With large number of extents, the sizes of these groups can become significant and thus every recursive call will attempt to reduce the size. However, recursion has its own overheads of function calls and could lead to stack overflow. Thus for efficiency and to overcome recursion overheads, a *threshold* specifying the maximum group size is used to stop recursion and extents within the group are finally examined using brute force approach. The brute force algorithm efficiently considers each of the extents ranges for a *non-overlap* using separating axis along range's dimension (Figure 8). The steps of the complete recursive algorithm are depicted in Figure 9. Steps 1-7 of the algorithm are quite straightforward and are based on the above discussion. When a sub group is formed and completed (step 8), a recursive call is made so that it can be examined along the next dimension. All groups with size greater than *threshold* are examined in all the dimensions before recursion can be stopped. Since dimension is an abstract concept, this algorithm can be extended to any n-dimensional routing space and thus unlike other approaches, it is not limited to 2D or 3D routing spaces.

$Input: P \in Gp.P, S \in Gp.S$

$Output: boolean$

$for\ each\ \ d\ \ in\ D(Sp)$

$\quad if\ ((P_u^d < S_l^d) \vee (S_u^d < P_l^d))$

$\quad\quad retrun\ false;$

$return\ \ true;$

X_y^d : Extent ***X***'s endpoint in dimension ***d***

Figure 8: checking for non-overlap using separating axis

Input: Gp (Group of all publisher and subscriber extents)
Output: list (list of Overlapping pairs)

1. If (Gp.size < *threshold* || invalid dimension ***d***)
2. Do brute force
3. Else
4. Examine current dimension ***d*** ∈ ***D(Sp)***
5. Create list ***l*** of endpoints in ***d***
6. Sort ***l*** using binary predicate of Figure 6
7. Find sub-groups of Gp using grouping algorithm of Figure 4
8. If Gp is sub-divided, then do steps 1 to 8 for each sub-group in other dimensions until
   a) The sub-group does not get subdivided and
   b) All dimensions have been examined for the sub-group

Figure 9: Steps of recursive algorithm for region matching

3.3. Algorithm's Complexity Analysis

The algorithm discussed above uses combination of sorting, brute force approach and recursion to achieve region matching. The overall aim of this algorithm is to reduce the inherent O(n^2) complexity of matching problem for scalability and performance. In order to provide complexity analysis, we assume there are ***n*** publisher and ***n*** subscriber extents and thus the group size is ***2n***. The first step in the algorithm checks the group size with the threshold ***t*** and uses brute force approach when the group size is less than or equal to ***t***. Thus, maximum complexity of brute force approach is of order ***O(t*t)***. Since sub groups comprises of extents that are possibly overlapping, the brute force algorithm yields best performance in this case.

Further, if the group is to be sub divided, it is examined along one of the dimensions. This requires creating a list of endpoints and sorting them. The list creation is a linear time operation of order ***O(m)*** for average sized group (***m<2n***) and sorting can be achieved in order ***O(mlogm)*** using a quick sort.

For an average case where most of the extents are non overlapping, we believe that every recursive call reduces the average group size ***m*** significantly and thus the algorithm's overall time complexity is of order ***O(nlogn)***. This is evident from the experimental simulation of the algorithm as discussed in section 4.

The worst-case performance of the algorithm arises mainly in two cases. Firstly, when all the extents are overlapping and secondly, when recursion goes very deep. In the first case, the algorithm examines all dimensions and finds only one group in the end that is sent to the brute force. This process is of order ***O(3nlogn + n*n)*** or equivalently ***O(n*n)***. In the second case, the recursion gets very deep mainly because of asymmetrical sub-division of the group such that in worst case there is only one extent in one of the sub-groups and the remaining n-1 extents in the other. If a similar sub-division occurs on every recursive functional call for the larger sub-group, then there could be n function calls in total having a time complexity of order ***O(n)***. Further, each function call creates and sorts the list of endpoints that has a time complexity of order ***O(nlogn)***. Thus worst case complexity is of order ***O(n*n*logn)***. However, in practice such cases are very rare and based on the average case performance, this algorithm can be applied for region matching where there are large number of extents.

4. Algorithm Simulation and Performance Evaluation

In order to see the effectiveness of the algorithm, we simulated the algorithm in C++ using visual studio .net on windows XP running on Pentium 4 3.2 GHz processor. We implemented DDM service interface of the HLA standard as defined in the draft specification [3]. This involved creation and defining classes for a routing space, dimension, region, extent, range, etc. Further, singleton *object factory* was created that acts as one-stop shop for creation of all the regions, extents and ranges. The factory maintains set of pooled buffers for regions, extents and ranges where each of these buffers maintains *free-list* and *used-list* for efficient creation and deletion of the associated types. In addition, spaces were used as indices to retrieve the pooled buffer for regions and extents whereas spaces and spaces dimension were used as indices to retrieve range buffers. All these optimisations have been incorporated so that the region-matching algorithm can have access to the buffers and associated objects in constant time.

For evaluation, a comparison of the algorithm's performance with the brute force algorithm is made. In addition, the following performance criteria were considered for evaluation: the computation time, scalability factor i.e. total number of extents supported, variability in threshold i.e. the affects of varying threshold size on the performance of the algorithm and the size of the routing space i.e. how algorithm behaves from average case scenarios to worst case scenarios. Further, random distribution of publisher extents and subscriber extents were used throughout the experiment and average of five runs of the algorithm for each data set is used to obtain the timing information. The routing space of 3 dimensions is used throughout the experimental simulation and the sizes of extents ranges were generated randomly using the same random number generator for all the scenarios. Three scenarios were considered for performance evaluation. For each of these scenarios, the extents were varied in numbers from 50 to 5000 whereas threshold is varied from 5 to 30. Charts 1-4 below details the simulation results.

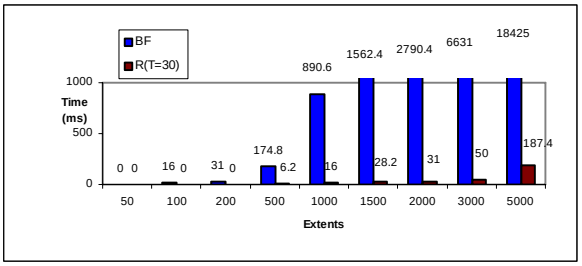

Chart 1: Performance comparison of a recursive algorithm (Threshold t=30) with brute force for 3 dimensional routing space of size 100000*100000*100000 (Average case scenario)

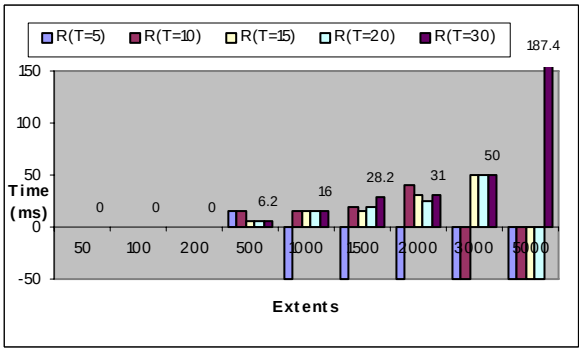

Chart 2: Recursive algorithm's simulation results for variable thresholds (T=5-30) for 3D routing space size: 100000*100000*100000 (Average case scenario)

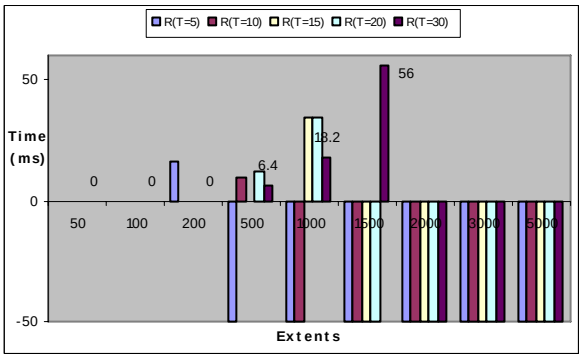

Chart 3: Recursive algorithm's simulation results for variable thresholds (T=5-30) for 3D routing space size: 10000*10000*10000 (Average to worst case scenario)

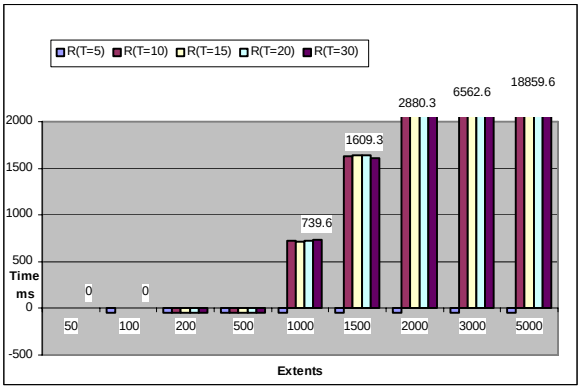

Chart 4: Recursive algorithm's simulation results for variable thresholds (T=5-30) for 3D routing space size: 1000*1000*1000 (Worst case scenario)

The first scenario for evaluation of the algorithm was based on the average case. For this, an expansive routing space of maximum 100000 units for each of its dimension was considered (chart 1 and 2). Here, extents were randomly distributed with ranges having sizes between 1 to 20 units. This forms an average case scenario as not only the extents are randomly distributed but also the size of extents is very small as compared to the world. This would form an ideal scenario for a massively multiplayer online game. Clearly from chart 1, it is evident that recursive algorithm outperforms the brute force approach. As hypothesized in the pervious section, the recursive algorithm does the region matching in logarithmic time as compared to the brute force approach that rises exponentially as the number of extents increases. Further, chart 2 details the recursive algorithm's behaviour with different choice of threshold. The ***negative*** bars in the chart reflect occurrence of stack overflows. Clearly, some interesting observation can be made form the chart. The algorithm performs smoothly up to 500 extents for any choice of threshold. However, as the extents were increased in numbers, the overheads of recursion dominate and result in stack overflows for small threshold values. This is true as more groups are being found with more number of extents and thus smaller threshold values do not break the recursion. Further two important observation made are: *when the number of extents is raised to 5000, the algorithm only succeeds with a threshold of 30 and whenever the recursion succeeds, the choice of threshold hardly have impact on the algorithms performance*. These two observations reflect key points with respect to scalability and performance of the algorithm. As the number of extents increases, the value of threshold should be increased appropriately and if the recursion succeeds, then this increase in threshold would not affect the algorithm's performance. A metric or a heuristic will be required that can achieve this seamlessly. A simple mechanism would be to catch the exception and then increase the threshold. This is yet to be incorporated in the current implementation of the algorithm.

In addition to the above scenario, we simulated the algorithm's performance for two other cases. One that reflects an average to worst-case scenario i.e. a routing space of maximum of 10000 units for each of its dimension (chart 3) and the other that reflects a worst-case scenario i.e. a routing space of maximum of 1000 units for each of its dimension (chart 4). In both these scenarios, extents were randomly distributed and the size of the their ranges varied between 1 to 20 units, as was the case with average case scenario. Because of this, the sizes of extents are quite large as compared to the size of the world and therefore there is a high probability that most of the extents would be overlapping with each other. However, such large extent sizes do not exist in reality for most of the games. Charts 3 and 4 details the simulation results for these scenarios. In chart 3, the algorithm performs smoothly for up to 1500 extents with higher end threshold values and thereafter recursion overhead dominates and result in stack overflows (*negative bars*) when extents are in between 1500 to 5000 (near worst-case). This happens mainly because of asymmetrical sub division of groups and because of possibly large numbers of extent's ranges are overlapping in each of the dimension. Further, it is observed (not in the chart) that at threshold values of 60 and 200, the algorithm succeeds and performs exceptionally well when the number of extents is 2000 and 3000 respectively. This information details some interesting facts. Comparing these results with average case scenario (chart 1), it can be seen that algorithm succeeds in both the cases when proper threshold is used. Further, with increase in the number of extents, the threshold value required increases in small steps (linearly) for scenario in chart 1 whereas for scenario in chart 2, it makes larger jumps (increases exponentially). Therefore, for proper running of the algorithm either a heuristic need to be developed that take this information into account for setting the value of the threshold or lookups could be used if the number of extents and world size are known a priori and remains fixed.

Lastly, chart 4 details the algorithm's performance for the worst-case scenario. Here it can be seen that algorithm performs well for only up to 100 extents. After that, recursion overheads dominate for extents between 200 and 500. However, some interesting results are obtained in the cases when extents are in numbers between 1000 and 5000. Here, the recursion succeeds (unlike the previous two cases) to actually do the region matching and the performance of the algorithm degrades to the order of brute force (see chart 1 and chart 4). This happens mainly because most of the extents are overlapping in this small world size and very large groups are being formed that cannot be sub divided and thus after going through all the recursive calls, the algorithm reverts back to brute force approach to complete the region matching. However such a scenario is impossible and has be created only for evaluation purposes.

5. Conclusion and Future Work

In this paper we presented a recursive algorithm for region matching that can be applied for interest management systems that uses multi-dimensional routing spaces. The concept of routing spaces for interest management has been adopted in standard distributed simulators and online games platforms. For scalability and performance, these systems require efficient solutions to interest management and the algorithm presented in this paper has potential to be used in these systems. The simulation results show promising results for region matching as compared to more traditional brute force approach. The results were even more impressive when large numbers of extents were involved. However, proper mechanism is required for setting the values of the threshold and we are researching techniques that can be used for that purpose. Further, for complete scalable interest management system, connectivity is to be established between nodes whose extents overlaps in order to do information exchange and therefore real evaluation of the algorithm would entail consideration of the network bandwidth, connectivity mechanism, allocation of multicast address, etc to do information exchange as done in real time online networked games. For this, we plan to integrate the algorithm in federated simulations development kit (FDK) [21]. FDK is an open source implementation of HLA standard developed at Georgia Institute of Technology. In our previous work [22], we discussed approaches for integrating FDK with game engines like Ogre3D so that it can be used for distributed agent simulation in online games. The algorithms integration in the FDK will not only improve the current HLA architecture but also provides middleware services for scalable online games.

References

[1] Singhal S, and Zyda M. 1999. *Netwroked Virtual Environments: Design and Implementation*. Addision Wesley

[2] Morse K. 2000. “An Adaptive, Distributed Algorithm for Interest Management”; *PhD Thesis*, University of California, Irvine

[3] US Defence Modelling and Simulation Office. 1998. High Level Architecture (HLA)- Interface Specification, version 1.3

[4] Macedonia M, Zyda M, Pratt D, Brutzmann D and Barham P. 1995. “Exploiting Reality with Multicast Groups: A Network Architecture for Large-Scale Virtual Environments”; *IEEE Computer Graphics and Applications*, 15(3): 38-45

[5] Miller D and Thorpe J A. 1995. “SIMNET: The Advent of Simulator Networking”, *Proc. of IEEE*, 83(8): 1114-1123

[6] Greenhalgh C and Bendford S. 1995. “MASSIVE: A Distributed Virtual Reality System Incorporating Spatial Trading”, *Proc. of 15th International conference on distributed computing systems (DCS 95)*, IEEE Computer Society, 27-35

[7] Epic Games 1999. *The Unreal Networking Architecture*. World Wide Web, http://unreal.epicgames.com/Network.htm

[8] Yu A and Vuong S T. 2005. “MOPAR: A Mobile Peer-to-Peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games”, *in proc. of International Workshop on Network and Operating systems Support for Digital Audio and Video*, pp: 99-104

[9] Liu E, Yip M and Yu G. 2005. “Scalable Interest Management for Multidimensional Routing Space”, *in proc. of the ACM symposium on Virtual Reality Software and Technology*, pp: 82-85

[10] Tan G, Ayani R, Zhang Y S and Moradi F. 2000a. “Grid-based data management in distributed simulation”, *In Proc. of 33rd Annual Simulation Symposium*, pp: 7-13, 16-20 April, 2000

[11] DIS. 1995. IEEE 1278 Standard for Distributed Interactive Simulation

[12] GauthierDickey C, Lo V and Zappala D. 2005. “Using n-trees for scalable event ordering in peer-to-peer games”, *In proc. o of International Workshop on Network and Operating systems Support for Digital Audio and Video*, pp: 87-92

[13] Logan B and Theodoropoulos G. 2000. “The Distributed Simulation of Multi-Agent Systems”, *in proc. of the IEEE-Special Issue on Agent Oriented Software Approaches in Distributed Modelling and Simulation*

[14] Morgan G, Lu F and Storey K. 2005. “Interest management middleware for networked games”, *In proc. of the ACM symposium on Interactive 3D graphics and game*, pp: 57-64

[15] Abrams H, Watson K and Zyda M. 1998. “Three-Tiered Interest Management for Large-Scale Virtual Environments”, *in proc. of the ACM symposium on Virtual Reality Software and Technology*, pp: 125-129

[16] Van Hook D, Rak S and Calvin J. 1996. “Approaches to RTI Implementation of HLA Data Distribution Management Services”, *in 15th Workshop on Standards for the Interoperability of Distributed Simulations*

[17] Rak S, Salisbury M and MacDonald R. 1997. “HLA/RTI Data Distribution Management in the Synthetic Theatre of War”, *in proc. of Fall Simulation Interoperability Workshop*, Orlando, Florida, USA

[18] Roy A. 2000. “Dynamic grid-based data distribution management in large scale distributed simulations”, *MS Thesis*, Department of Computer Science, University of North Texas

[19] Tan G, Ayani R, Zhang Y S and Moradi F. 2000b. “A Hybrid approach to Data Distribution Management”, *In Proc. of 4th IEEE International Workshop on Distributed Simulation and Real-Time Applications*, pp: 55-61, San Francisco, CA

[20] Raczy C, Tan G and Yu J. 2005. “A Sort-Based DDM Matching Algorithm for HLA”, *in ACM Transactions on Modeling and Computer Simulation*, pp: 14-38

[21] FDK- Federated Simulations development kit. Available: http://www.cc.gatech.edu/computing/pads/software.html

[22] Kumar P. 2005. “Towards Integrating Ogre3D with FDK”, *in proc. of 7th International Conference on Computer Games (CGAMES)*, Angouleme, France