Load Balancing and Quality of Service Constrained Framework for Distributed Virtual Environments

Noah Dietrich
College Of Charleston
66 George Street
Charleston, SC 29424
Telephone: 001 843 805 5507
Email: ndietric@edisto.cofc.edu

Shankar M. Banik
The Citadel
171 Moultrie Street
Charleston, SC 29409
Telephone: 001 843 953 5039
Email: shankar.banik@citadel.edu

***Abstract*—Distributed Virtual Environments (DVE) have become increasingly popular over the last few years. Examples of DVEs are Massively Multiplayer Online Games (MMOGs), distributed interactive simulation, and shared virtual worlds. The service providers of DVEs need to ensure that certain Quality of Service (QoS) (message delivered within a threshold delay) is guaranteed for the users participating in the DVE. At the same time the service providers want to balance the load on the servers that maintain the DVE. In this paper, we propose a framework for DVE which provides QoS to the users and balances the load among the servers. Our framework uses the concept of virtual server which is a piece of software that does the processing for the DVE. Each region in the DVE is maintained by an overlay of virtual servers. We provide a heuristic that maps the virtual servers to physical servers, balances the load among the servers and ensures that the servers are not overloaded with objects. We also present a heuristic for creating a Degree and Diameter Bounded Multicast Tree of virtual servers for each region which guarantees QoS for users in the DVE. We have conducted simulation experiments to evaluate the performance of our proposed framework.**

***Index Terms*—Distributed Virtual Environment, Quality Of Service, Load Balancing.**

I. Introduction

Distributed Virtual Environments (DVEs), wherein users represented as digital entities collaborate in a networked virtual environment, are fast becoming popular nowadays. Examples of DVEs are massively multi-player online games, distributed interactive simulations, shared virtual worlds, as well as other virtual collaborative environments. The main facet of DVEs is that its users are geographically distributed all over the world. In this type of environment, response time for user interaction is critical as delays in the action can lead to a degraded performance. As long as events and actions within the DVE are communicated to the user within a threshold delay [1], the state of DVE will appear to the users to be realistic. It is the responsibility of the DVE providers to ensure that certain *Quality of Service* (QoS) is guaranteed to the users of DVE. At the same time the DVE providers want to balance the load on the servers and ensure that the load placed on each server in the DVE does not exceed the capabilities of that server. Here the load is quantified by the number of objects of the DVE that the server manages. In this paper, we propose a framework for DVE which balances the load among the servers and at the same time ensures QoS for the users of DVE.

In our framework we have used *Virtual Servers* (VS) which are applications running on *Physical Servers* (PS) that manages a subset of objects in the DVE. We have also used Application Layer Multicasting [2] [3] [4] [5] [6] where the virtual servers in a DVE region will form an overlay multicast tree to send messages to the users. To ensure QoS for the users, we have restricted the diameter of the multicast tree to a threshold value so that when a user changes the state of an object of the DVE, this event is delivered to all the users in the DVE within the threshold value. At the same time, we have restricted the degree of the multicast tree since we want to evenly distribute the communication load across the servers in the DVE.

There has been a significant amount of research done in designing frameworks for DVE. In [7] the authors have used an interest management scheme which involves breaking up DVE regions into smaller regions with each region being handled by a single server. As more clients enter a region, the region can be sub-divided again and spread across more servers. In [8], the authors have referred the regions as microcells and divided them across multiple servers. Crowding is looked at by authors in [9] where regions are repeatedly subdivided into quad-trees, and then assigned to servers. Communication architecture management is considered by authors in [10] where they use Zoom-In Zoom-Out algorithm to select the specific servers from the set of all possible servers in a cluster to handle the region of a DVE while maintaining an equal synchronization delay model. The drawback of their approach is that the client select the servers and instead of load balancing it uses only minimum number of servers. In [11] the authors have used *Network Address Translation* (NAT) to hide the implementation of server infrastructure from clients. None of these proposed frameworks have taken both load balancing and QoS issues into considerations. Much research has been done in peer-to-peer load balancing for DVE in [12] [13] [14], but along with other researchers and gaming industry we believe that the issues raised by trusting clients to run essential functions of DVE has too many downsides including security, cheating, failure, and synchronization issues.

In this paper we propose a framework for DVEs which balances the load by creating virtual servers on the physical servers to manage objects in the DVE, and then constructs a degree and diameter bound multicast tree with the virtual servers to ensures that users of DVE will receive messages within a specified threshold delay. Towards designing our framework, we have proposed two heuristics - one for mapping virtual servers to physical servers which considers load balancing, and the other for creating a diameter and degree bound multicast tree of virtual servers which considers the QoS constraints of the DVE. We have also performed experimental evaluations of our proposed framework. The paper is organized as follows. In section 2 we describe the system model of our framework. In section 3 we describe the problem formulation along with examples. Section 4 describes our proposed solutions. Section 5 describes our experimental setup and results from performance evaluations. Section 6 concludes the paper.

II. System Model

The servers that run the DVE reside in *datacenters*, which are facilities with clusters of servers connected in a high speed Ethernet Local Area Network (LAN) with very high interconnect speeds between the servers (1 to 10 G-Bit links) and extremely low latency. The datacenter provides power, cooling, network, and server redundancy, and is connected to the Internet through at least one Wide Area Connection (WAN) link that can be as slow as a T1 (1.5 M-Bit) or up to a OC-3 Link (45 M-Bit) and greater, and which have greater latency than LANs. Different datacenters are connected through the public WAN infrastructure using a technology such as MPLS, Frame Relay, ATM or a similar technology.

Every datacenter is capable of reaching every other datacenter, and every server is capable of reaching every other server. Each datacenter has at least one server that clients connect through, called a *connection server*. Clients are geographically distributed all over the world, and are pieces of software on user's home computers which is responsible for sending status messages about the user's movements and actions in the DVE, and updating the client's view of the DVE. The servers perform all positional calculations and processing affecting the DVE, and the clients only display the results provided by the servers. Clients connect to the closest connection server through region-based DNS (Domain Name Service), which looks at the client's IP (Internet Protocol) address and identifies the connection server that is geographically closest to the client, as shown in Fig. 1.

The infrastructure supports the addition and removal of individual servers, clients, as well as entire datacenters. Servers can be brought on- and off-line at any time, for failure or maintenance, or to assist with heavy server load. As new servers are added or removed, the servers that are still active automatically compensate for the join or loss of servers, and redistribute the load automatically.

The DVE is sectioned into regions based on the position within the virtual environment, and each region is represented

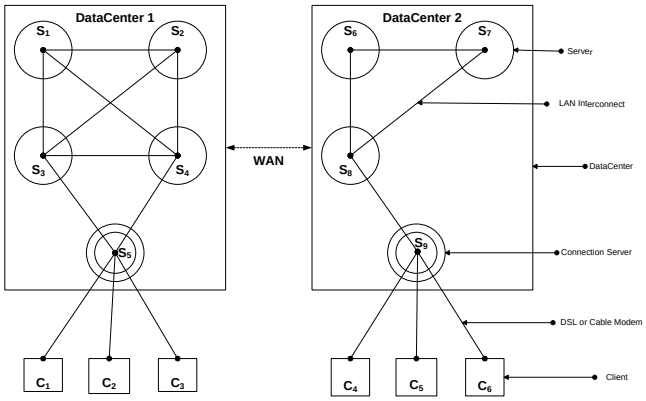
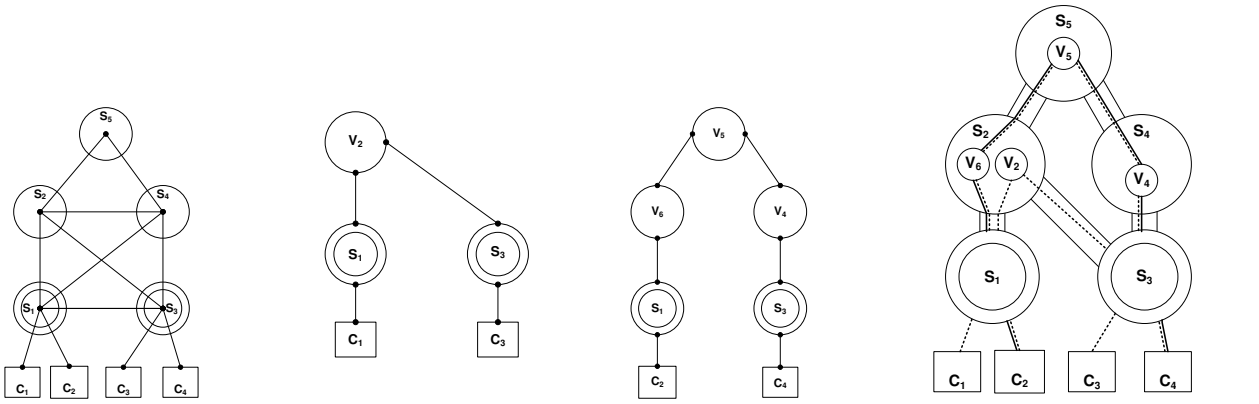


Fig. 1: Physical Connections Between Clients and Servers

by a multicast overlay. Users, objects, and all other items in the DVE are represented by Objects, which contain the objects state and other relevant information. All servers that service the region are a member of the multicast overlay for that region, and use a virtual server to handle the overlay. A virtual server is a piece of software running on a physical server that does processing for a region, and which communicates with other virtual servers participating in the same region to maintain the state of the region. A physical server can run many virtual servers at one time, and virtual servers can be moved between physical servers, split onto multiple virtual servers, or combined into fewer virtual servers (see Fig. 2). In Fig. 2a we see the physical infrastructure, with four clients connected to the DVE, and 2 connection servers (which only pass information to the servers handling the DVE) and three physical servers that host the DVE. In Fig. 2b, we show one overlay and the virtual servers managing it: clients C_1 and C_3 connect to their connection server, which passes all messages to virtual server V_2. At the same time, we also have clients C_2 and C_4 connecting to their own overlay (see Fig. 2c). This overlay is made up of the virtual servers V_2, V_4, and V_5. In Fig. 2d we show how the virtual servers are mapped to the physical servers. In this example, we can see that physical sever S_2 hosts two virtual servers, V_2 and V_6, which are participating in different overlays.

Everything in the DVE can be represented as objects, including clients, environmental objects (trees, rocks), artificial characters (AI opponents), weapons and tools which the client can use. Each virtual server is responsible for a subset of objects from all objects in the region represented by the DVE. The objects are distributed across the virtual servers representing the DVE, and as objects interact, messages are passed between the virtual servers hosting the interacting objects. In Fig. 3a, we see two clients (objects) interacting through the overlay. When client C_1 wants to interact (send a message) to client C_2, it sends the message through its connection server S_1, which forwards the message to the virtual server V_1. V_1 notes that the message is for client C_2, and sends the message up the hierarchy to the root node



(a) Physical Servers

(b) Overlay One of Virtual Servers

(c) Overlay Two of Virtual Servers

(d) Virtual Servers Mapped to Physical Servers

Fig. 2: Individual Overlays and the Physical Mapping

of the overlay tree, which sends the message down to the virtual servers hosting the client, which sends the message to the connection server for client C_2, which forwards the message to C_2. In Fig. 3b, we see a client in the same overlay interacting with an object in the DVE (striking a rock in the DVE, for example). The object the client is trying to interact with is hosted on virtual server V_3, so the message is sent by client C_1 to the connection server S_1, which forwards the message to the virtual server V_1, which forwards the message up the overlay tree to virtual server V_2, where the message is forwarded down to virtual server V_3, which hosts the object the client is interacting with. Any messages back to the client from the interaction with the object would reverse the path that the original message took.

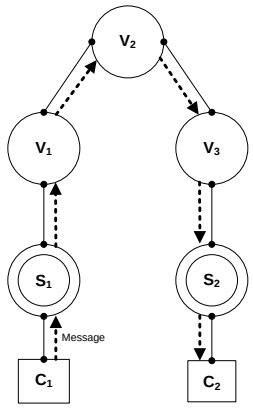
When messages need to be passed to all clients or objects in a region, a multicast tree for the overlay is used. We use a shared multicast tree (made up of virtual servers), with the root node of the tree being responsible for forwarding messages to all nodes in the tree. In Fig. 4a, we show that if an event happens on virtual server V_2, the message passes up to the root node of the multicast tree: V_1 (step 1), which multicasts the message out to all its child nodes as shown in Fig. 4b, (step 2). As each node receives the message, it forwards it on to its children, (step 3). We use a shared multicast tree to ensure that messages destined for all nodes in the tree receive the message within a specified delay, and also it allows us to maintain a single tree with a single root node, rather than a multicast tree for each node in the graph.

Object-to-Object (including clients) QoS must be maintained in order for clients to receive relevant messages without noticing a delay, which would impact the clients' experience in the DVE. The primary requirement is for all clients to have a Total Message Trip Time (TMTT) for messages to remain under some constant, known as Maximum TMTT (often 150 ms in real life) [1]. TMTT is defined as the time it takes for one object (the sending object) to send a message to the virtual server hosting the recipient object, plus the time it takes for the server to process the message, plus the time it takes for the server to distribute the result of the message back to all objects affected by the original message. Clients in the DVE will feel as if the interactions they are having with the DVE is real-time as long as all interactions they have with the DVE occur within the Maximum TMTT. In Fig. 5, we show how the TMTT is calculated as a client sends a message to another client using the multicast overlay. In our example, client C_1 wants to send a message to client C_2. The numbers on each link in the virtual server overlay show the delay each link adds to the TMTT calculation. As the message passes from client C_1 to its connection server S_1, to the virtual server, down to the connection server S_2, and to client C_2, we note that each path adds to the TMTT calculation, giving us a total of 85 ms for the message to complete the path from client C_1 to client C_2.

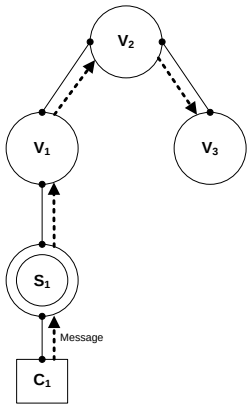
If client C_1 was attacking client C_2, the message path would be slightly different, because the processing for client C_1's attack would be done at the server, and messages would have to be passed back to clients C_1 and C_2, as shown in Fig. 5b. When client C_1 decides to attack client C_2, a message is sent by C_1 to its connection server S_1 (step 1). S_1 sends the message to the virtual server hosting the client (step 2). When virtual server V_1 receives the message, it will process the request, wait for any other events that happen within that time slice, and then send the message back to the clients through their respective connection server (steps 3 and then 4).

III. Problem Formulation With Example

We break the problem of how to construct a DVE given a set of available resources while maintaining an acceptable TMTT into two parts, first we map the virtual servers to the physical servers, and then from the fully-connected graph of virtual servers, we construct a Degree and Diameter bound Multicast Tree for each region in the DVE.

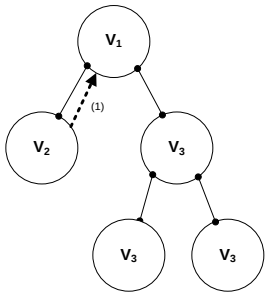


(a) Client To Client Message

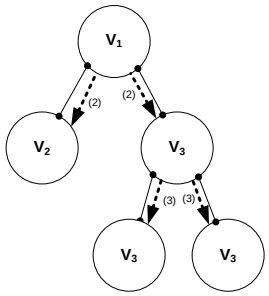


(b) Client To Server Message

Fig. 3: Clients Interacting With Clients and Objects

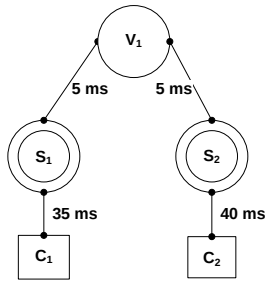


(a) Message Passing From V_2 to the Root

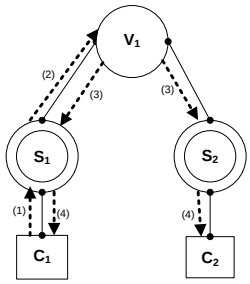


(b) Root to V_2 Multicast's the Message in the Overlay

Fig. 4: Message Passing in the Multicast Tree



(a) Link Delays Used for Calculating TMTT



(b) Steps Used in Calculating TMTT

Fig. 5: Calculating TMTT

A. Mapping Virtual Servers to Physical Servers

The virtual servers participating in the overlay are free to remove virtual servers from the overlay or have other virtual servers join the overlay as processing needs increase or decrease. Virtual servers are mapped to physical servers, and multiple virtual servers can reside on one physical server. The physical servers are mapped in a network that represents the physical layout of all physical servers. We identify the load for each physical server as the total number of objects that it can be responsible for before response times are impacted. We make this assumption because despite there being many types of objects, some requiring larger memory and other requiring more CPU cycles, in aggregate the objects will appear to put equal load on a server. Each server will know the total number of objects it can be responsible for, and the number of objects

held by the different virtual servers running on that physical server. If a physical server can host 1000 objects, and has 4 virtual servers running, with 200, 250, 400, and 100 objects for each respective virtual server, the total server load will be 950 objects, which will work. If one of the virtual servers gets an additional 100 objects, the physical server will not be able to meet its load requirements (leading to delays in processing messages), and will ask the virtual servers to reduce the load, either by moving the virtual server to another physical server, or splitting one of the virtual servers into two virtual servers, and moving the newly created virtual server to another physical server.

In the example shown in Fig. 6 virtual server V_1 has 75 objects. If the DVE represented by V_1 grows in size, and V_1 receives 50 objects, the physical server S_1 will know that it cannot handle the current load, and asks V_1 to reduce load. V_1 decides to split into two virtual servers, V_1 and V_2, and moves V_2 to physical server S_2 (Fig. 6b). If messages need to be passed between regions (between overlays), such as when a client is near the border of a region and can interact with objects in both regions at the same time, messages are passed to the root node of the clients overlay, which are passed to the root node of the overlay for the other region for processing.

B. Creating Degree and Diameter Bounded Multicast Tree with Virtual Servers in a DVE Region

As long as the time for all messages that are sent and received by clients are less than the maximum TMTT, we know that QoS requirement for clients has been met. We consider the set of virtual servers that are managing the region of the DVE that the clients are participating in, and the overlay connecting the virtual servers. This set of virtual servers form a fully connected graph, since the underlying physical servers that the virtual servers are on are fully-connected, and the virtual servers use the same links as the underlying physical servers. These virtual servers can send messages directly to other virtual servers in the overlay, which works well when an object on one virtual server needs to send a message to a single object on another virtual server in the overlay. However, when one virtual server needs to send a message to all virtual servers in the overlay, it would cause a bottleneck on the sending virtual server to queue up that many messages for all other virtual servers. To solve this issue, we construct a *Degree and Diameter Bound Multicast Tree* (DDBMT) from the overlay of virtual servers, and perform application-layer multicast to send overlay-wide messages to all servers in the DVE region. We choose to perform application-layer multicast (at ISO OSI Layer 7) because in a network that spans network links that we do not have control over traditional network-layer (layer 2) where multicast might be blocked. We construct the multicast tree as both a degree and diameter bounded spanning tree. We are first concerned about maintaining the maximum TMTT for all clients in the overlay, which is why the tree must be diameter bound, with the diameter of the tree being half of the client's maximum TMTT (since messages have to go from the client to the server, and then back again). We bound the tree's degree (a maximum degree for each single node), since if we did not, we would wind up with a tree with a single central node connected to each and every other node, and the minimum diameter tree of a tree from a fully-connected graph would produce just such a tree, and this would put a heavy burden on the central node, having to pass each and every message to all the other nodes in the tree. By bounding the degree of each node in the tree, we ensure that a tree is formed that evenly distributes the load of multicast messaging more equally across the nodes in the tree, with each node in the degree and diameter bound tree having to pass a message at most to a small subset of all nodes (based on the maximum permitted degree of the tree).

C. Extended Example

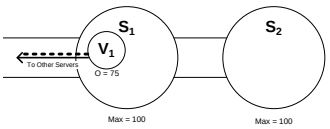
In our extended example, we assume that we have two regions in our DVE, with clients C_1 and C_3 involved in region A, and clients C_2 and C_4 in the region B. We show the overlay network of virtual servers for the clients in Fig. 7a, with *Region A* shown on the left, and *Region B* shown on the right. The physical network of servers and clients is shown in Fig. 7b, with link delay in ms. The physical mapping of virtual servers to physical servers is shown (Fig. 7c), with the maximum load of each server (100 objects) and number of objects for each virtual server shown. If the number of objects in region B (represented by the overlay held by virtual server V_{B1} grows by 20 objects, the physical server S_3 will be overloaded, and will ask the virtual servers to reconfigure themselves to use less resources. Virtual server V_{B1} could split, and move the new virtual serve V_{B2} to physical server S_5, after ensuring that the TMMT was less than the maximum TMMT for all clients and servers in the overlay (Fig. 7d).

IV. Proposed Solutions

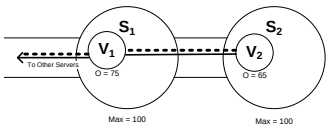
We present heuristics here which are all required to construct the framework of DVE, which comprises of mapping virtual servers to physical servers, and then constructing a degree and diameter constrained multicast tree. The first heuristic maps a DVE to a series of virtual severs running on physical servers, based on the free space available on the physical servers. The second heuristic constructs the degree and diameter constrained multicast tree of virtual servers for a region of DVE. Due to layout constraints, we have presented the heuristics at the end of the paper.

A. Mapping Virtual Servers to Physical Servers

For mapping virtual servers to physical servers, our first step when given a physical environment made up of datacenters, physical servers, link costs, and a DVE with a set number of objects, is to map the DVE to the physical servers using virtual servers on each physical server, to hold a subset of the objects from the DVE. Our solution (as described in Heuristic 1 and 2) iterates through each datacenter, trying to choose a physical server in that datacenter that has enough resources to hold the total number of objects in the DVE divided by the the total number of datacenters. If we can do this for each

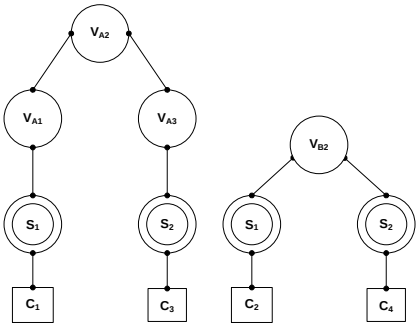


(a) Single Virtual Server

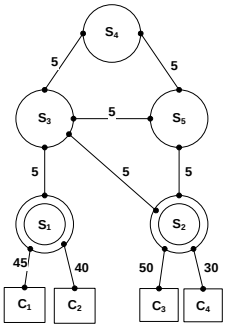


(b) DVE Split Across Second Virtual Server

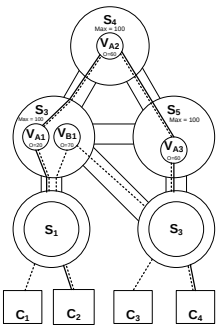
Fig. 6: Load Balancing Objects



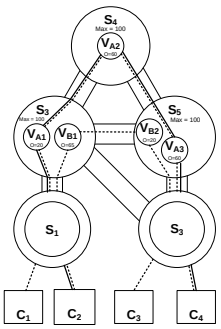
(a) Overlay Networks(Region A and B)



(b) Link Delays Across Physical Network



(c) Virtual Servers for 2 DVE's Mapped to Physical Servers



(d) New Virtual Server Spawned

Fig. 7: Individual Overlays And the Physical Mapping

datacenter, we have successfully mapped the environment. If we cannot find a physical server in the datacenter that has enough free resources, we take the physical server with the best resources, and map that many resources. After iterating through each datacenter, if we still have objects left to map, we take and map as many as possible to the physical server in each datacenter that has the most free resources, for each datacenter. If once done with this, we still have objects to map, we loop through each datacenter, continuing to map as many objects to the physical server with the most free space until we either map all objects, and our heuristic is complete, or run out of physical servers to map objects to, at which point we return an error. The problem of mapping virtual servers to physical servers is similar to the Graph Embedding Problem [15], which is NP-Hard. Our proposed heuristic for mapping virtual servers to physical servers has a complexity of $O(n^2)$.

The heuristic listed in Heuristic 1 requires D, a list of datacenters, each datacenter containing a list P of physical servers, which is broken into two separate sets, P_U, which are the servers powered on, and P_D, the set of servers that are powered off. We use the $POWER\_ON$ function to move a server from P_D to P_U. Every server has L_M, which is the maximum number of objects the server can manage, and L_C, the current number of objects that the server is managing. We have a DVE, consisting of a specific number of objects, which we want to map to a series of virtual servers, which is mapped to physical servers, using the MAP function.

B. Creating Degree and Diameter Bound Multicast Tree

To create the Degree and Diameter Bound Multicast Tree *(DDBMT)*, presented in Heuristic 3, we start with the fully-connected, weighted graph of virtual servers given to us from the earlier mapping heuristic. We start by selecting a starting node, using a heuristic described later, and put that node in our tree. From this starting node, we begin constructing our tree using a modified version of Prim's greedy algorithm [16]. From all nodes in our tree (to start, only the starting node is in our tree), we look at all the nodes that have a degree of less than our maximum degree, Deg_{MAX}, and we find the link with the lowest cost from a node which is not in the tree to a node which has already been included in tree and its degree has not exceeded Deg_{MAX}. We add the node that has that lowest link cost to the tree. We continue this process until we have added all nodes to the tree. As we add each node to the tree, we calculate the diameter of the new tree, and verify that we have not exceeded our maximum TMTT using a heuristic described later, otherwise we fail to construct a DDBMT, and would need to either select a different starting node, or increase our maximum degree for the tree. The problem of finding a diameter and degree bound spanning tree is NP-Hard [15] and our proposed heuristic has a complexity of $O(n^4)$. The heuristic Creating DDBMT uses three sub-routines, which are described below.

1) SubRoutine: SelectStartingNode(Graph G): When using a greedy heuristic with a degree constraint to construct a spanning tree from a fully-connected graph, the node that

we start with will affect the diameter and links selected for the spanning tree. To select a node more optimally than a random selection, we take the node with the best $Diam_{MAX}$ links, based on the assumption that this node has the best connections to its neighbors for all of it's links to neighbors. We do not guarantee that this node is the best starting node, but we assume that it is a good choice for a starting node.

2) SubRoutine: LongestPathLength(Tree T, Node s): When constructing a diameter-constrained spanning tree from a fully-connected graph, it is critical to check the diameter of the tree as each node is added into the spanning tree. This subroutine takes a tree: T, as well as a leaf node: s, and determines the longest path in the tree containing (starting at) node s. Node s is specified since we are constructing a tree by adding new (leaf) nodes to the tree. We know that the tree diameter is less than the $Diam_{MAX}$ prior to adding node s, so we just need to know if adding node s creates a diameter for T longer than $Diam_{MAX}$.

3) SubRoutine: FindTreeCenter(Tree T, Node s): This subroutine is very similar to the *LongestPathLength* subroutine. It begins by walking from the starting leaf node outward, computing the distance from the starting leaf node for each node. Once the longest path length is found, we know that the center node is just past half that distance from the leaf node along the path for the longest path. We remove leaf nodes that are not the starting or finishing node, until we are left with a linked list from the starting leaf node to the final node. We travel down that list until we have passed half the total distance, and that node is the center node for this tree, which we return.

V. Performance Evaluation

For our experimental setup we constructed a representation of the physical environment using object instances in C++. We have a class representation for each of the following: datacenter, physical server, virtual server, link, and DVE. We then create instances of all these objects and map them to each other, using the mapping heuristic from above, then use our second heuristic to construct a DDBMT. For our results, we compiled and linked the application with profiling enabled, and then we ran our program across a large number of simulations with different input parameters to get graphs showing how increasing the size of the environment affected the execution time of our heuristics. We also compared how adjusting the degree of the multicast tree (from DDBMT) changed the diameter of the multicast tree as the diameter of the multicast tree is closely related to the QoS provided by the multicast tree. For our data gathering, we used 10-40 datacenters, with 5-20 servers per datacenter. We chose link costs 2-5 ms for the links between physical servers in the same datacenter, and link costs 20-100ms for physical servers in different datacenters. When constructing our DDBMT, we used degree values beetween 3 and 12.

A. Mapping Virtual Servers to Physical Servers

We first compare the execution time of the mapping heuristic against the number of datacenters in the environment. From Fig. 8 we observe that as the number of datacenters increases, the execution time also increases.

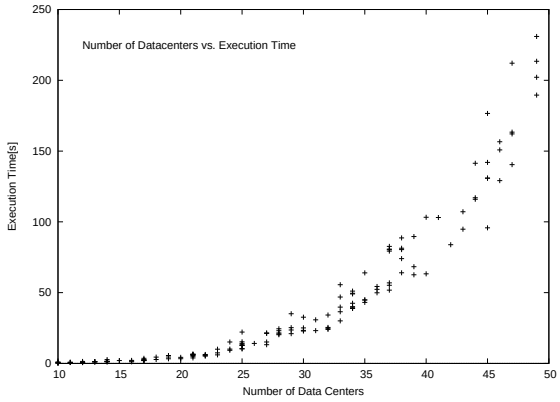


Fig. 8: Number of Physical Servers vs. Execution Time

B. Creating Degree and Diameter Bound Multicast Tree

For the DDBMT heuristic, we first compare the execution time for constructing the DDBMT for various number of virtual servers, using a Deg_{MAX} of 7. In graphing execution time versus the number of virtual servers, as shown in Fig. 9, we observe that the execution time increases when the number of virtual servers increases.

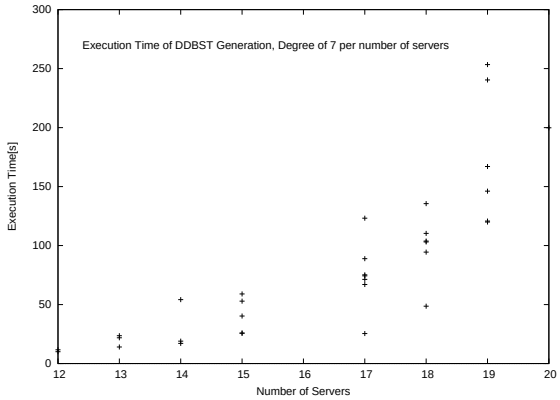


Fig. 9: Number of Nodes vs. Execution Time

For the DDBMT heuristic, we also plot the degree versus the diameter of the multicast tree. From Fig. 10 we observe that as we increase the degree (Deg_{MAX}) in our heuristic, the diameter of the multicast tree decreases as long as the degree is below 6 which implies that if we increase the degree of the multicast tree, we can achieve a better multicast tree in terms of QoS. When the degree is greater than 6, there are few changes in the diameter of the tree.

VI. Conclusions and Future Work

In this paper we have proposed a framework for a DVE which balances the load among the servers that maintain the DVE and ensures QoS for the clients of the DVE. We have used the concept of virtual servers which are mapped

to physical servers to ensure even mapping of resources for the DVE, and then constructed a degree and diameter bound multicast tree to ensure QoS for all clients participating in the DVE. Our solutions are targeted at creating an even mapping of resources for a DVE to physical severs while maintaining QoS for clients and servers communications.

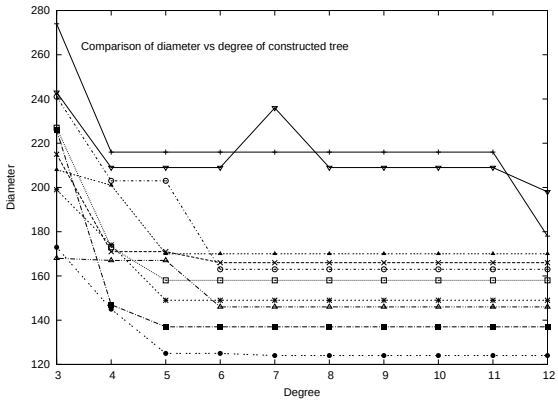


Fig. 10: Number of Nodes vs. Execution Time

We have run extensive simulations to analyze the heuristics of our proposed framework. Using simulated objects and environment, we have varied input parameters across a range that represents both small, medium, and large DVE implementations. From the simulation results we observe that the execution time for Heuristic 1 increases with an increase on the number of datacenters in the DVE. We also observe that the execution time for Heuristic 3 increases when the number of virtual servers increases. We are able to show that virtual servers could be mapped to physical servers using our heuristic as long as there are enough resources in the environment for the mapping, and we could construct trees of varying degree and diameter from the graph of virtual servers.

We also observe that the parameter (Deg_{MAX}) has an impact on the diameter of the multicast tree constructed by our heuristic DDBMT. If we increase (Deg_{MAX}) we can have a better multicast tree in terms of QoS which implies that we will have a multicast tree with less diameter.

Future work should focus on using the feedback from the DDBMT heuristic to our virtual server to physical server mapping heuristic instead of constructing the tree from the completed mapping of the virtual servers to physical servers. Other areas of future research could focus on decreasing the construction time of the tree by mapping nodes that are close to each other together prior to constructing the entire tree.

REFERENCES

[1] M. Claypool, K. Claypool, "Latency and Player Actions in Online Games," in *Communications of the ACM*, Vol 49, No 11, November 2006, pp. 40-45.

[2] Sherlia Y. Shi, Jonathan. S. Turner,"Multicast Routing and Bandwidth Dimensioning in Overlay Networks," in *IEEE Journal on Selected Areas in Communication*, Vol 20, Issue 8, October 2002, pp. 1444-1455.

[3] Eli Brosh, Yuval Shavitt,"Approximation and Heuristic Algorithms for Minimum Delay Application-Layer Multicast Trees," in *IEEE INFOCOM 2004*,March 2004.

[4] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, S. Khuller,"Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," in *IEEE INFOCOM 2003*,March 2003.

[5] Anton Riabov, Zhen Liu, Li Zhang, "Overlay Multicast Trees of Minimal Delay," in *24th IEEE International Conference on Distributed Computing Systems ICDCS 2004*, 2004.

[6] S. Banerjee and B. Bhattacharjee, "A comparative study of application layer multicast protocols," 2001.

[7] D. Lee, M. Lim, and S. Han, "Atlas - a scalable network framework for distributed virtual environments," in *4th International Conference on Collaborative Virtual Environments*, 2002.

[8] M. Assiotis and V. Tzanov, "A distributed architecture for mmorpg," in *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA: ACM, 2006, p. 4.

[9] J. Lim, J. Chung, J. Kim, and K. Shim, "A dynamic load balancing for massive multiplayer online game server," in *ICEC*, ser. Lecture Notes in Computer Science, R. H. R. Harper, M. Rauterberg, and M. Combetto, Eds., vol. 4161. Springer, 2006, pp. 239–249.

[10] K.-W. Lee, B.-J. Ko, and S. Calo, "Adaptive server selection for large scale interactive online games," *Comput. Netw.*, vol. 49, no. 1, pp. 84–102, 2005.

[11] F. Lu, S. Parkin, and G. Morgan, "Load balancing for massively multiplayer online games," in *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA: ACM, 2006, p. 1.

[12] S. Yamamoto, Y. Murata, K. Yasumoto, and M. Ito, "A distributed event delivery method with load balancing for mmorpg," in *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA: ACM, 2005, pp. 1–8.

[13] P. Morillo, W. Moncho, J. M. Ordua, and J. Duato, "Providing full awareness to distributed virtual environments based on peer-to-peer architectures," in *Lecture Notes on Computer Science*, 2006, p. 2006.

[14] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, T. Teifel, and G. Carle, "Clustering players for load balancing in virtual worlds," *Int. J. Adv. Media Commun.*, vol. 2, no. 4, pp. 351–363, 2008.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman & Co Ltd, January 1979.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2001.

Heuristic 1 Mapping VS to PS, Part 1

Require: D, a list of Datacenters $\{D_1, D_2, ..., D_i\}$, each datacenter holding between 1 and $|P|$ servers from P.
Require: P, a list of physical servers $\{P_1, P_2, ..., P_j\}$, with each $P \in D_i$.
Require: P_U, a list of all servers $P_U \in P$ which are powered on.
Require: P_D, a list of all servers $P_D \in P$ which are powered off.
Require: Every Member of $P_U \notin P_D$.
Require: $L_M(P_i)$: The maximum load for server P_i, in terms of number-of-objects.
Require: $L_C(P_i)$: The current load for server P_i, in terms of number-of-objects.
Require: DVE: A Region in a Distributed Virtual Environment, made up of N objects, to be mapped to a set of virtual servers, V, which are mapped to a subset of P_U.
Require: A function $MAP(V(t), P(t), n)$ which maps a virtual server to a physical server with N objects. $L_C(t)$ will be decreased by n, and N will be decreased by n.
Require: A function $Power\_On(p)$, which powers on a server (removes it from P_D and adds it to P_U).

1: **for all** $s \in D$ **do**
2: **for all** $t \in P_U$ WHERE $P_U \in s$ AND $L_C(t) < L_M(t)$ **do**
3: **if** $N/|D| < L_M(t) - L_C(t)$ **then**
4: $MAP(V(t), P(t), N/|D|)$
5: GOTO next s
6: **end if**
7: **end for**
8: **for all** $t \in P_D$ WHERE $P_D \in s$ **do**
9: **if** $N/|D| < L_M(t)$ **then**
10: $Power\_On(t)$
11: $MAP(V(t), P(t), N/|D|)$
12: GOTO next s
13: **end if**
14: **end for**
15: $a = 0$
16: $b = NULL$
17: **for all** $t \in P$ WHERE $P \in s$ **do**
18: **if** $a < L_M(t) - L_C(t)$ **then**
19: $b = t$
20: $a = L_M(t) - L_C(t)$
21: **end if**
22: **end for**
23: $Power\_On(b)$
24: $MAP(V(b), P(b), a)$
25: **end for**

Heuristic 2 Mapping VS to PS, Part 2 (continuation of Heuristic 1

26: **if** $N > 0$ **then**
27: **for all** Servers $r \in$ DVE **do**
28: $MAP(V(r), P(r), L_M(r) - L_C(r))$
29: **end for**
30: **end if**
31: **if** $N > 0$ **then**
32: **for all** $s \in D$ **do**
33: **for all** $t \in P_U$ WHERE $P_U \in s$ AND $L_C(t) < L_M(t)$ **do**
34: $MAP(V(t), P(t), L_C(t) < L_M(t))$
35: GOTO Next s
36: **end for**
37: **end for**
38: **end if**
39: **while** $N \neq 0$ **do**
40: **for all** $s \in D$ **do**
41: **for all** $t \in P_D$ WHERE $P_D \in s$ **do**
42: $Power\_On(s)$
43: $MAP(V(t), P(t), L_M(t))$
44: **end for**
45: **end for**
46: **if** $P_D = \{\}$ **then**
47: HALT, Not enough free space on all servers for this DVE to start.
48: **end if**
49: **end while**
50: **return** A mapping between V and P which satisfies load and response constraints.

Heuristic 3 Creating DDBMT

Require: $G = (V_G, E_G)$.
Require: $Diam_{MAX}$.
Require: Deg_{MAX}.
Require: A function $CalculateStartingVertex(GraphG)$ which selects the starting node to construct the multicast tree.
Require: A function c defined as $c(u, v)$ for $u, v \in V_G$ that gives the cost for the direct link between u and v. $c(u, v)$ will return ∞ if there is no direct link between u and v, and will return 0 if $u = v$.
Require: A function $LongestPathLength(TreeT, Nodes)$, which returns the length of the longest path containing a leaf node s from all other nodes in the tree.
Require: A function $FindTreeCenter(TreeT, Nodes)$, which returns the center node of a weighted tree.

$V_T = \{\}$
$E_T = \{\}$
$g = 0$
$h = NULL$
$s = CalculateStartingVertex(GraphG)$
$V_T = V_T \cup \{s\}$
while $V_T \neq V_G$ **do**
  $W = \{\}$
  for all $u \in V_T$ **do**
    if $Deg(u) < Deg_{MAX}$ **then**
      $W = W \cup \{u\}$
    end if
  end for
  for all $u \in W$ **do**
    $q = \infty$
    $m = NULL$
    $X = V_G - V_T$
    for all $y \in W$ **do**
      for all $z \in X$ **do**
        if $c(y, z) < q$ **then**
          $q = c(y, z)$
          $m = (y, z)$
        end if
      end for
    end for
    $V_T = V_T \cup \{z\}$
    $E_T = E_T \cup \{m\}$
    $e = LongestPathLength(V_T, z)$
    if $e > Diam_{MAX}/2$ **then**
      need a new starting node
    end if
    if $e > g$ **then**
      $h = FindTreeCenter(V_T, z)$
      $g = e$
      $c = \{h\}$
    end if
  end for
end while
return A Multicast Tree $T = (V_T, V_E)$ where for each node $v \in V_T, Degree(v) \leq Deg_{MAX}$, and the diameter of tree $T \leq Diam_{MAX}$.