Dynamic Resource Allocation for MMOGs in Cloud Computing Environments

Chen-Fang Weng and Kuo-Chen Wang.
Institutes of Network Engineering,
National Chiao Tung University

Abstract—A massively multiplayer online game (MMOG) has hundreds of thousands of players who play in the game concurrently. The players consume a great deal of CPU, memory and network bandwidth resources in MMOGs. We combine MMOGs with cloud computing environments, use virtual machine servers (VMSs) in cloud computing environments instead of traditional physical game servers. By using a multi-server architecture, we divide a game world into several zones, and each zone consists of at least one VMS to execute game processes and exchange game information among players in the zone. In addition, we design an artificial neural network (ANN) and also an adaptive neural fuzzy inference system (ANFIS) to predict the load of each zone and decide a resource allocation policy to be performed by the VMS. These policies include (1) this VMS is sufficient to support adjacent VMSs; (2) this VMS will release the resources which has been supported by adjacent VMSs or a secondary VMS which supports this VMS; (3) this VMS will remain in the current state; (4) this VMS requires adjacent VMSs to support it; (5) a secondary VMS will be created between this VMS and an adjacent VMS. Experimental results show that the mean square error of the ANFIS-based load prediction is lower than that of the ANN-based load prediction. Therefore, we incorporate the ANFIS prediction method along with the five resource allocation policies to the MMOG cloud. In terms of average access time, the proposed ANFIS-based resource allocation method is 16.7% better than the deep-level partitioning (DLP) method.

Index Terms—ANFIS, ANN, cloud computing, load prediction, resource allocation.

I. INTRODUCTION

Resource allocation discusses how to make available resources be allocated more efficiently. Resource allocation includes two topics [5]: (1) basic allocation decision — it determines which resources can be used by which objects; (2) contingency mechanism — it chooses which tasks will be sacrificed when the system is overloaded, and it chooses which tasks will enter when the system is idle. There are two methods for resource allocation [5]: (1) real-time resource allocation — it can allocate resources dynamically when needed; (2) pre-allocation — it predicates the loads by historical data then assigns the available resources. In addition, there are two types of resource allocation [5]: (1) centralized approach — all the available resources are controlled by one central device. It is easier to control and deploy all the resources in a system, but the central device might be the bottleneck in the system; (2) distributed approach — all the devices can make some decisions for resources by themselves, but it is hard to implement.

A massively multiplayer online game (MMOG) is a multiplayer video game which is capable of supporting hundreds of thousands of players simultaneously. There are two kinds of actions for each player: (1) independent action — players only concern about the items, equipments or status in the game; (2) interaction — players interact to the game server or communicate with other players [3]. Most MMOGs are MMORPG (massively multiplayer online role, such as Lineage [21] and WOW [22]. There are three architectures for MMOGs: (1) Client-server architecture — a game server governs all the players in this game. However, the client-server architecture has poor scalability. (2) Peer-to-peer architecture — a player is regarded as a peer who shares their game information by itself. However, peer-to-peer architecture is hard to implement for the security problem. (3) Multi-server architecture — it enhances the client-server architecture. The game world is divided into several zones and each zone is governed by a server. However, load unbalancing might still occur in game servers when some zones with crowded players exist in the multi-server architecture. Therefore, we propose a cloud-based dynamic resource allocation method to resolve the multi-server load unbalancing problem by flexible allocating of resources.

There are four resource components in MMOGs [4]: (1) authentication component, (2) storage component, (3) computation component, and (4) communication component. Since communication and computation components consume most of resources, we only consider these two components in this paper.

The rest of this paper is organized as follows. In Section II, we discuss an MMOG load balancing mechanism, deep-level partitioning (DLP) and its multiple hotspots problem [19]. In Section III, we design an artificial neural network (ANN) and also an adaptive neural fuzzy inference system (ANFIS) to predict MMOG load and decide a resource allocation policy which can be performed by a VMS. In Section IV, we describe a simulation environment and discuss simulation results. Finally, concluding remarks and future work are given in Section V.
II. RELATED WORK

In zonal MMOGs, a virtual game world is divided into several zones (or microcells). Each zone consists of a game server to execute game processes and to exchange game information among players in the zone. Ahmed et al. [24] proposed a server pool concept in the virtual game world. A server pool contains several game servers. Each game server serves a zone. The server pool operates load distribution of the game servers which are in it. Moreover, the authors proposed a buffer region between two zones. The buffer region can share game messages between two adjacent zones which are in different server pools. In this approach, since it will transfer zones between game servers, the cost of inter-server communication may increase.

Wang et al. [25] proposed a method for MMOGs to find less loaded game servers. They defined a threshold \( W \). If the server loading \( S \) is larger than \( W \), the game server is regarded as overloading. They used two kinds of lists in their approach, select list and candidate list. In the first list, they select an overloading game server \( A \) in the select list, and they chose the servers which are adjacent to \( A \) in the candidate list. Then if there is an overloaded game server \( B \) in the candidate list, \( B \) will be inserted in the candidate list. And the game servers which are adjacent to \( B \) will be inserted in the candidate list. If there are less loaded game servers in the select list, the least loaded game server \( C \) will be inserted in the select list. And the game servers which are adjacent to \( C \) will be inserted in the candidate list. In this method, the more the number of game zones is, the higher the complexity of the algorithm is.

Carlos Eduardo et al. [26] proposed to use a KD-tree to divide a virtual game world into several zones. Each node of the KD-tree represents a game zone. In this approach, a game zone \( A \) (we mark node \( A \) in the KD-tree) contains two subzones \( B \) and \( C \) (we mark nodes \( B \) and \( C \) in the KD-tree). And nodes \( B \) and \( C \) are the children of node \( A \) in the KD-tree. Each node contains two kinds of values. One value is the load of its children, and the other is the capacity of its children. When a game server is overloaded, it will readjust the load of the server using the KD-tree method. The limitation of this approach is that the game zones should keep a rectangle shape.

Vlad Nae et al [28] proposed a load model for MMOGs. The load model includes CPU load model, memory load model and network load model. In addition, they used the neural network, average, moving average, last value and exponential smoothing to predict the load of CPU, memory and network based on real game traces from RuneScape. They showed that the prediction error of the neural network based method is lower than that of the other prediction methods. And the neural network based method is faster than the other methods. They also found that the dynamic resource provisioning is more efficient than static resource provisioning.

In the deep-level partitioning (DLP) method designed for zonal MMOGs, there is a load threshold \( T_{sw} \), and it defines \( S_i \) as the load of game server \( i \). When \( S_i \geq T_{sw} \), game server \( j \), which is an adjacent game server \( i \) and \( S_j < T_{sw} \), will support game server \( i \). In this way, the overloading problem can be resolved. However, the DLP method becomes inefficient when there are some contiguous zones with crowded players (multiple hotspots problem).

III. DYNAMIC RESOURCE ALLOCATION FOR MMOG CLOUDS

Most MMOGs mainly employ a multi-server architecture. In the multi-server architecture, the game world is divided into several zones, and each zone consists of a game server. In our design, we combine MMOGs with cloud computing environments. We use virtual machine servers (VMSs) instead of physical game servers. There are two advantages for using VMSs: (1) We can easily allocate appropriate VMSs resources from physical game servers; (2) It is easy to migrate players from a VMS to other VMSs. In addition, we will create a SVMS between two adjacent overloaded VMSs to resolve the multiple hotspots problem, as shown in Fig 1. Using the SVMS method can avoid the high cost of supporting the entire game zone by using an extra VMS.

We define five resource allocation policies based on the load level of each VMS:

1. **This VMS can support adjacent VMSs.** When the load of this VMS is light, this VMS will inform adjacent VMSs that it has redundant resource to support those adjacent VMSs with high or heavy load.

2. **This VMS releases the area which is supported by adjacent VMSs or SVMSs.** If there are some areas of this VMS which has high load, have been supported by adjacent VMSs or the SVMSs, this VMS will take back these areas and support them by itself. If it is a SVMS, the SVMS will be released when the VMS does not support any area between this VMS and the adjacent VMSs.

3. **This VMS will remain in the current state.** This VMS will not change its resource allocation policy even if this VMS has supported adjacent VMSs, or has been supported by adjacent VMSs or SVMS.

4. **This VMS will require adjacent VMSs to support it.** Since this VMS becomes overloaded, it informs adjacent VMSs that it needs to be supported. And an adjacent VMS with lightest load can support this VMS.

5. **An SVMS will be created between this VMS and an adjacent VMS with heaviest load.**
The proposed dynamic resource allocation scheme for MMOGs in cloud computing environment is described in Figure 5. First, we collect MMOG game traffic which includes CPU, memory and network bandwidth load from Lineage [21] for each VMS and obtain a historical game dataset. Then we analyze the historical game dataset to predict the load level of each VMS using the proposed artificial neural network (ANN) or the proposed adaptive neural fuzzy inference system (ANFIS). According to the predicted load level, the VMS can execute a selected resource allocation policy. Finally, we measure the VMSs’ CPU, memory and network loads and then add these data to the historical game dataset.

A. Proposed artificial neural network based load prediction

We define the loads for CPU, memory and network, as follows:

\[
\begin{align*}
CPU_{Load} &= \frac{CPU_{usage}}{CPU_{VMS}} \\
MEM_{Load} &= \frac{MEM_{usage}}{MEM_{VMS}} \\
BW_{Load} &= \frac{BW_{usage}}{BW_{VMS}}
\end{align*}
\]

There are three layers, input layer, hidden layer and output layer in an artificial neural network (ANN), the input layer is composed of CPU_{load}, MEM_{load} and BW_{load}, which are from the historical game dataset (we use \(x_i, x_2\) and \(x_3\) to represent CPU_{load}, MEM_{load} and BW_{load}). The hidden layer contains ten neurons and the output layer contains one neuron.

Each neuron in the hidden layer must perform the sum up of the weighted inputs \((x_i, x_2\) and \(x_3\)) and compute them by a log-sigmoid function \(f^{(1)}\). For each neuron \(j (1 \leq j \leq 10)\) in the hidden layer, we have

\[
\begin{align*}
\eta_j &= \sum_{i=1}^{3} w_{ij} x_i - b_j \\
a_j &= f^{(1)}(\eta_j) = \frac{1}{1 + e^{-\eta_j}}
\end{align*}
\]

A neuron of the output layer sums up the weighted \(a_j\), for \(j = 1, \ldots, 10\), which are generated by the neurons in the hidden layer and compute them by a linear function \(f^{(2)}\). By this procedure, we can get an output value \(O\) which is the load level.

In the learning process for an ANN, we define \(W\) as a set of all weights at first. Then we define \(B\) as a set of biases for an ANN. We use partial differential mean square error (MSE) equations for weights and biases to adjust weights and biases in the ANN.

B. Adaptive neural fuzzy inference system load prediction

An adaptive neural fuzzy inference system (ANFIS) combines a neural network with fuzzy inference. The ANFIS contains five layers which include inputs, input membership functions, fuzzy rules, output membership functions and defuzzification output.

The inputs for ANFIS contain CPU_{load}, MEM_{load} and BW_{load} which are defined in A. We use \(x_1, x_2\) and \(x_3\) to represent CPU_{load}, MEM_{load} and BW_{load}. For each input, we define five generated bell-shaped input membership functions which are based on an MMOG.

We define \(S_1 = \{a_{ij}, b_{ij}, c_{ij} \mid 1 \leq i \leq 3, 1 \leq j \leq 5\}\) as a set of premise parameters of \(f^i\) input membership function of \(x_i\) in ANFIS. Premise parameters will be adjusted by the training process of ANFIS. Next, we set up twenty five fuzzy rules based on MMOG data, as shown in TABLE I. Fuzzy rules are in if-then forms. The conditional statements are expressions of the input membership functions. And we choose the product to be the AND method. And we normalize the products from all the fuzzy rules.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Input</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>light</td>
<td>light</td>
</tr>
<tr>
<td>2</td>
<td>light</td>
<td>light</td>
</tr>
<tr>
<td>3</td>
<td>light</td>
<td>low</td>
</tr>
<tr>
<td>4</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>5</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>6</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>7</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>8</td>
<td>low</td>
<td>medium</td>
</tr>
<tr>
<td>9</td>
<td>low</td>
<td>medium</td>
</tr>
<tr>
<td>10</td>
<td>medium</td>
<td>low</td>
</tr>
<tr>
<td>11</td>
<td>medium</td>
<td>medium</td>
</tr>
<tr>
<td>12</td>
<td>medium</td>
<td>medium</td>
</tr>
<tr>
<td>13</td>
<td>low</td>
<td>medium</td>
</tr>
<tr>
<td>14</td>
<td>medium</td>
<td>low</td>
</tr>
<tr>
<td>15</td>
<td>medium</td>
<td>medium</td>
</tr>
<tr>
<td>16</td>
<td>medium</td>
<td>medium</td>
</tr>
<tr>
<td>17</td>
<td>high</td>
<td>medium</td>
</tr>
<tr>
<td>18</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>19</td>
<td>medium</td>
<td>high</td>
</tr>
<tr>
<td>20</td>
<td>high</td>
<td>medium</td>
</tr>
<tr>
<td>21</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>22</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>23</td>
<td>heavy</td>
<td>-</td>
</tr>
<tr>
<td>24</td>
<td>-</td>
<td>heavy</td>
</tr>
<tr>
<td>25</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Each rule corresponds to an output membership function. The output membership functions are linear combination functions with inputs \(x_1, x_2\) and \(x_3\). An output membership function for a rule \(k\) as is follow:

\[
f_k(x_1, x_2, x_3) = p_k x_1 + q_k x_2 + r_k x_3 + s_k\]

where \(1 \leq k \leq 25\) since we set up twenty five fuzzy rules. We define \(S_2 = \{p_k, q_k, r_k, s_k \mid 1 \leq k \leq 25\}\) as a set of consequent parameters. The consequent parameters will be adjusted by the training process of ANFIS.

Finally, the output value \(O\) in ANFIS is the load level of each VMS. \(O\) is the summation of all the output membership functions \(f_k\), \(k = 1\) to \(25\), with normalized weight \(\bar{w}_k\), which is generated by the \(k^{th}\) fuzzy rule.

There are two steps in the learning process of ANFIS. In the first step, we use a least square estimator (LSE) to adjust the
consequent parameters in $S_2$. In the second step, we use partial differential input membership functions for premise parameters in $S_1$, such as the learning process in the ANN.

IV. SIMULATION RESULTS

We collected game traffic which includes CPU, memory and network loads from Lineage. We used MATLAB as our prediction tool. Both of ANN and ANFIS are APIs (application programming interfaces) in MATLAB [29] to predict the load level of game traffic. We used CloudSim [30] as our simulation tool. We used 16 VMSs to deploy in the game world. The number of players in each VMS is 40 ~ 100.

We installed Lineage in our game server. The CPU utilization, memory consumption and network bandwidth usages will be recorded at every minute in our gaming experiment. CPU and network bandwidth usages are proportional to players in Lineage [21]. Also, players’ actions affect CPU utilization and network bandwidth usage. The memory usage increased slowly as the players increased.

We compared the mean square errors for the ANN-based load prediction method and the ANFIS-based load prediction method. Since the ANFIS-based load prediction method has fuzzy rules which were based on the game features from Lineage [21], the mean square error is lower than that of the ANN-basedload prediction method. Moreover, the prediction time of the ANFIS load prediction method is much smaller than that of the ANN-based load prediction method.

We implemented multi-server, ANFIS-based DLP and DLP+SVMS with ANFIS methods. Experimental results show that the average access time (queueing time + CPU time) of the proposed ANFIS-based DLP+SVMS resource allocation method is 16.7% shorter than that of the ANFIS-based DLP method, as shown in Fig 2. In Fig 3, we show the VMS usages of the three resource allocation methods. The proposed ANFIS-based DLP+SVMS method has the smallest number of VMSs used among the three methods.

V. CONCLUSION

There are two phases in the proposed dynamic resource allocation method: load prediction phase and resource allocation phase. In the load prediction phase, we collected historical game data which includes CPU, memory and network loads from a popular MMOG, Lineage. We have designed and simulated an artificial neural network (ANN) and an adaptive neural fuzzy inference system (ANFIS) to predict an appropriate resource allocation policy to be executed in each game zone. Experimental results show that in the load prediction phase, the mean square error and prediction time of the ANFIS-based load prediction scheme are lower than those of the ANN-based load prediction scheme. In the resource allocation phase, the average access time (execution time plus queueing time) of the proposed ANFIS-based deep-level partitioning (DLP) with secondary virtual machine servers (SVMSs) method is 16.7% shorter than that of the ANFIS-based DLP method. In addition, the proposed method has the smallest number of VMSs used among the three methods.

In our current design, we focused only on CPU, memory and network loads in a VMS. In the future, we will include the access time of storage devices in our load prediction. In addition, we will implement and evaluate our proposed load prediction methods and the proposed resource allocation policies in a real cloud computing environment.

REFERENCES


