Computer Science and Information Systems 11(4):1537–1553 DOI:10.2298/CSIS131220078M

Optimization of server performance in the CMX educational MMORPG for Computer Programming

Christos Malliarakis[1], Maya Satratzemi[1], Stelios Xinogalos[1]

[1] Department of Applied Informatics, University of Macedonia, 156 Egnatia Street, 540 06 Thessaloniki, Greece
{malliarakis, maya, stelios}@uom.gr

Abstract. A new generation of computer games has taken over during the last few years, called Massive Multiplayer Online Role Playing Games (MMORPG). In parallel, the usage of games in education has increased, exploiting the fact that young people are familiarized with them and would be more motivated to learn while entertained. However, MMORPG require significant amounts of resources, such as bandwidth, RAM and CPU capacity to support learning. In this paper, we propose a new methodology to achieve monitoring and optimization of the load balancing, so that the essential resources for the proper execution of an educational MMORPG for computer programming can be foreseen and bestowed without overloading the system. Moreover, we present an educational MMORPG called CMX, which aims to teach computer programming through interactive activities and role-playing. We finally apply the proposed model to CMX by conducting an experiment and conclude that the server's performance is indeed increased.

Keywords: MMORPG server optimization, networked game, MMORPG delay, educational game, computer programming.

1. Introduction

Students of the 21st century learn and react accordingly to the digital world they are growing up in. As a result, they are less impressed and motivated to learn in traditional ways. This is due to the fact that computer games with impressive graphics invade their lives from a very early age, making them significantly accustomed to many of the computer's functionalities.

In parallel, students are still confronting severe difficulties in computer programming, which are still a topic of interest for researchers in the last two decades [13]. To hinder these difficulties, several educational programs were developed that also aimed to depict additional competences, such as critical thinking and problem solving. The relevant literature identifies three major categories of environments that have been constructed towards this goal. The first category is educational programming environments, sets of tools supporting students in the process of learning introductory programming [2, 5, 17]. Secondly, programming microworlds are learner-centred worlds explored by directly manipulating objects in the world with a limited set of simple commands coupled with metaphors to aid in problem description and to exploit storytelling [14, 22]. Finally,

educational games are used as educational tools and provide compelling contexts via interactive, engaging and immersive activities [10]. Out of the three categories, educational games are considered to promote the majority of interaction and collaboration towards a final goal by allowing students to engage in activities already familiar to them from computer games. It should be noted that educational games also support engaging learning for learners with disabilities, allowing them to learn through entertaining and motivating activities [24]. These games combine the delivery of domain-specific knowledge with an attractive graphical environment and interesting scenarios to enhance students' motivation. This field is known as Game Based Learning and includes computer games that are structured in a way that they not only entertain the players, but also monitor and guide them through their learning process [9].

A new generation of computer games has started to take over during last few years, called Massive Multiplayer Online Role Playing Games (MMORPG). This field has brought on a vast variety of multiplayer games with attractive virtual environments, interesting and interactive scenarios and technologically enhanced features that have been built on top of advanced infrastructures that can service a large amount of users [18]. A number of MMORPGs are popular with the young generation, which has increased the interest in applying these games in educational settings. However, they require significant amounts of resources in order to function properly, and a number of parameters can play a part in the system's consistent and responsive performance.

Therefore, system delay or failure issues are very important factors for the proper operation of these games in education and they could have a huge impact in the learning process and students' motivation. These issues are even more important when MMORPGs are applied in computer programming courses, where functionalities such as compiling, debugging, and executing complex algorithms in a programming language are required. Also, the packets transmitted between the server and the clients do not include solely score points and players' positions in the world, but they hold lines of code that the server needs to retrieve, compile and send proper feedback to the corresponding client.

Thus, the main problem that presents a significant research interest and remains a challenge is the identification of how we can address the resources consumption in an educational MMORPG in order to ensure its proper operation. In other words, it is important to study how all of the beneficial features of an educational MMORPG can be put to use, while the server hosting the game runs with no problems from beginning to end. This way, teaching and learning is enhanced, students remain immersed in the game's environment, without disorienting their interest or motivation to win, the system does not present delays in its responses and manages to produce the appropriate feedback to the teacher (learning analytics).

For the successful solution of all the different problems that might occur and are crucial to the server system's performance, a number of interesting algorithms have been constructed and are presented in the related work section of the paper.

The next section briefly describes the related work and its limitations, followed by a presentation of CMX, an educational game developed to facilitate teaching and learning of computer programming and will be used for the evaluation of the proposed optimization algorithm. Next, the paper includes a section that presents the system performance analysis and the corresponding creation of a mathematical model that estimates how active players are hosted within an MMORPG game in a server so that

the server administrators can take the necessary measures that will ensure optimized server's performance. Moreover, the section includes a process to ensure server's performance optimization. The last section presents the evaluation of this optimization with an experiment using the CMX game. The paper concludes with conclusions drawn.

2. Related work

The literature review carried out indicated a limited amount of studies that investigate and measure the server system's performance in online multiplayer games, though no studies were identified for focusing on educational games and especially in the field of computer programming. All these studies take into account that multiplayer games can support either TCP/IP or IPX internet protocols and can be played either over LANs or the internet. The goal in each study was to examine the parameters that affect the server's operation and thus should be taken into consideration while optimizing a game's performance. It should be noted that studies on optimizing applications and mainly games has been a research focus for over two decades. However, there is a distinction between optimizing network resources and optimizing the performance of the packets' transmission on the servers, by monitoring CPU usage and RAM efficiency.

A case study carried out by [23] using the "World of Warcraft", identified the specific features of an MMORPG that affect network performance and latency. Their focus was on determining which of the game's actions mobile devices can support; however, they provide useful information that are in accordance with our scope of interest. More specifically, the authors list actions that are usually executed by players in the game across a number of categories, which are deemed to shape the network traffic. A representative overview of these actions' categories is presented in the following Figure.

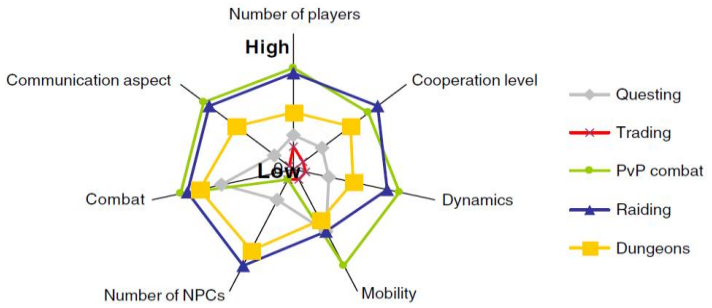

Fig. 1. Categories of actions within "World of Warcraft" 23

Moreover, [23] recorded the specific characteristics that influence network traffic, such as the size of each packet transmitted, the time between each packet's transmissions, the bandwidth that is used and the load of each packet. Other characteristics observed that could affect network performance include the session duration and each player's interest in the actions showed in Fig. 1. Also, [4] and [7] have created their own models on game network traffic. According to their studies, interactive gaming is extremely delay-sensitive, which in many cases can incorrectly

determine who wins or loses a game.

Another research carried out by [7] studies the "ShenZhou Online" MMORPG indicates that the characteristics that constitute the game's traffic include: the average bandwidth, the volume of the traffic in the client and the server, the time interval status update is carried out (e.g. how periodically is the game's status synched), the overall activity of the game and the game sessions per hour.

To examine the network traffic based on the transmit-receive cycle, [4] developed models and tested them on the "Quake 2" game. These models indicated that the performance of the game is highly dependent on the clients' machines as well as on the hosts' CPU speed. The exact characteristics that were assessed regarding the network traffic include the server's CPU, Operating System and Random Access Memory.

Taking into consideration these characteristics and measuring the amount of packets transmitted, the study calculated the mean and standard deviation of the transmission volume as well as the time passed between transmissions. This study does not elaborate on possible techniques of optimizing the traffic and thus the game's performance levels, but suggests such actions as future work.

According to [8], the process followed throughout a game's session includes the transition of packets from the server to the clients logged on regarding the game's state. Each client receives the data and processes the information by synchronizing their local game's state with the server's. This way, all players in different clients can interact with the game while having the exact same visual representation of the game's status at any period interval. In the study, [8] used the "Counter strike" game and developed a model to measure its network traffic. This game is ideal for such examinations, since, like most multiplayer games, it is built under client-server architecture and presents a very high traffic volume available to be examined due to its popularity.

In this model, they studied the server and clients independently, without taking into consideration any possible inter-dependencies between them that could affect the network traffic. This can be also seen in Fig. 2, where individual measures are taken for server and client, and the characteristics that are evaluated are the size of the packets transmitted and received as well as the time spent between each transmission.

	Server (per client)		Client	
	characteristic	**approximation**	**characteristic**	**approximation**
(burst) interarrival time	peak = 55 ms mean = 62 ms coeff. of variation = 0.5	Extreme (a=55,b=6)	mean = 41.7 ms coeff. of variation = 0.24	Deterministic (40 ms)
packet size	mean = 127 Bytes coeff. of variation = 0.74	Extreme (a=120,b=36)	mean = 82 Bytes coeff. of variation = 0.123	Extreme (a=80,b=5.7)

Fig. 2. Traffic evaluation for "Counter Strike" 8

As optimization techniques, the study suggests carrying out predictions on the game's status (e.g. interactions of the players with the game's bots, objects and other players), a method initially proposed by [3] and [15].

An additional study that researched network traffic is [12]. The mathematical model they use to measure traffic delay is shown in the following equation.

$$T = T_{\min} + \frac{S}{R_{eff}} + T_{que,tot} \quad (1)$$

The measures included in the above equation include all features that the authors suggest strongly influence a game's performance, where T_{min} is the minimum packet processing delay in a transmission, S is the packet size measured in bits, R_{eff} is the effective link rate in a packet transmission and $T_{que,tot}$ is the total delay while the packets are queuing.

Additionally, [6] studied the players' behaviour according to network traffic as well as the factors that affect it. They also chose the "Counter Strike" multiplayer game and examined the way the game's state is constantly updated by the server and transmitted into all the different clients as well as how the players interact with the game's objects. This study does not focus on the actual game's performance and the characteristics that could optimize the process.

Another research by [21] carried out a thorough investigation on the different factors that influence network traffic and multiplayer games' performance. More specifically, the affecting factors were categorized into three axes, namely "Resource Management", "Consistency and Responsiveness" and "Logical Platform and Application".

The quantity of resources transmitted from the clients to the servers and vice versa is a feature that was pointed out by all studies as influential in network traffic. [20] move on to elaborate on the individual elements that comprise the resources sent in a multiplayer game. These elements have been identified and form an equation called the "Information Principle Equation", shown in (2).

$$\text{Resources} = \text{M} \times \text{H} \times \text{B} \times \text{T} \times \text{P} \quad (2)$$

According to this equation, the different information types that are part of the resources in a game include the number of messages (*M*), the average number of nodes where a message has to be transmitted (*H*), the average of the network bandwidth that can successfully transmit a message without losses (*B*), the time in which each message is transmitted over the network with the corresponding bandwidth to the corresponding node (*T*) and the actual number of cycles the processor has to complete for the successful message transmission (*P*). The equation shows that all these parameters are strictly interconnected; thus, any attempt to lower one in an attempt to enhance the resource transfer will lead to the necessary increase of one or more of the others.

Moreover, [20] investigated the factors that influence the game's consistency and responsiveness, since both these elements indicate a game's high quality. According to their study, the factors that influence consistency are the bandwidth volume, the latency volume and the number of nodes that are connected to the system. Moreover, the factors that influence responsiveness include fast resources transmission, lack of lost messages and again bandwidth and latency.

Other studies that have been carried out research approximately the same features in assessing multiplayer games' performances and delay factors [16, 11]. The most commonly mentioned factors include bandwidth delay and consumption [1, 19]. However, all aforementioned studies focus mostly on network traffic of strategy or shooting games and less on MMORPGs that require more resources and in different formats, as it is usually the case in the educational domain. All changes in a player's status (e.g. score, new level, new task achieved) are required to be recorded and presented as a feedback not only to the player in question but also to all other players that could see him/her in their view scope. This way, network traffic in MMORPGs involves changing recordings and multiple message documentations and transmissions

to players of the game. This is even more required in multiplayer games that are used in education, since any system delay or failure will not only reduce the quality of the users' entertainment but of the learning process as well.

3. CMX –An educational MMORPG for computer programming

In our effort to tackle the problems students and teachers face during computer programming learning and teaching, we developed an educational game that increases motivation to engage in learning and introduces computer programming elements in a scenario-based, problem-solving setting. In this section we elaborate on the game, which is called CMX, and provide an overall overview of its features and scenario. In doing so, we aim to show why such a system's optimized performance is necessary.

Just as the majority of the multiplayer games, one of the most important features of CMX is the scenario based on which the students are required to learn computer programming. The virtual environment of the game is a toxic waste factory that is equipped with a number of employees and an advanced system for handling all its business activities and ensuring high quality of waste handling. A toxic waste factor with protagonists a series of activists was chosen as the game's scenario in order to send an ecological message through the game. Thus, the MMORPG is not another common war game like most commercial versions; in CMX, players learn how to program in order to stop the waste factory's server from working. Since CMX is not only a multiplayer game but it also supports role-playing, we have configured two opposite teams in the environment, namely "hackers" and "crackers". The hackers are the employees of the factory while the crackers are outsider activists that want to break through the system. Each player, according to the team's role, has to achieve activities in order to win.

Additionally, CMX includes internal characters called Senseis, which aim to tutor the students' characters in the game on how to solve the assigned computer programming tasks. If the students seem to face difficulties in comprehending the materials taught, the Senseis provide explanatory messages and hints to guide them through each activity. Once an activity is successfully executed, students are rewarded with passwords that assist them in proceeding to and interacting with the next levels of Senseis.

We consider the initial tutoring of the students essential for their successful learning process; hence, the training process is divided into three phases based on their level of difficulty, and each phase includes a different Sensei type. The Basic Sensei initiates interactive training and provides a number of multiple-choice questions to its assigned player regarding the learning materials. The player answers these questions and the answers are evaluated automatically by the Sensei. Correct answers are rewarded with passwords and the player moves on to the second level and the Iron Sensei. The player is provided with a drag & drop mechanism and maps the correct answer to a given question. The third training phase includes a Gold Sensei, where students have to write their own lines of code with the guidance and motivation provided by the Sensei. Once all three trainings are completed, the players are equipped with a comprehensive body of essential knowledge and with rewards that will help them win the game.

The successful training of the players requires the transmission of a large amount of messages from the server to the clients and vice versa (e.g. questions, answers, awards,

hints, explanations, level, task number etc). Moreover, it is essential that this transmission occurs with no errors, since the players need to be given correct feedback on their answers. If the server's response is not accurate or if it is delayed significantly, the learning process is strongly and negatively affected.

The second area is called the Arena, where players interact with each other and all players of one team (hackers) confront the players of the other team (crackers). All gained items from the trainings are exploited in this phase, and winning players gather even more weapons they can later on use in their tasks. The game also presents each player's status as bars on the screen, with information such as remaining energy and number and type of remaining weapons. Additionally, students in the Arena can communicate with each other through a specialized chat tool included in the game, shown in Fig. 3.

CMX ensures that students fully understand how to write lines of code in C or Java by including a C or Java compiler in the code editor. The compiler presents all errors made during the code programming and gives the opportunity to students to retry the process.

With this synchronous communication, students are able to help each other, solve problems and communicate ideas in their common effort to execute their assigned tasks.

The game can easily be used by a teacher to facilitate teaching computer programming. Each teacher can redesign the game easily by setting his own multiple choice questions for the Senseis, the drag and drop programs for the iron Senseis and the scenarios of the programs that students are required to write for the gold Senseis.

From students' point of view, the better players they are going to be, the more they will increase their programming skills. This is because during their continuous interactions with all levels of Senseis, they learn the theory more in depth and they practice in different programming code exercises.

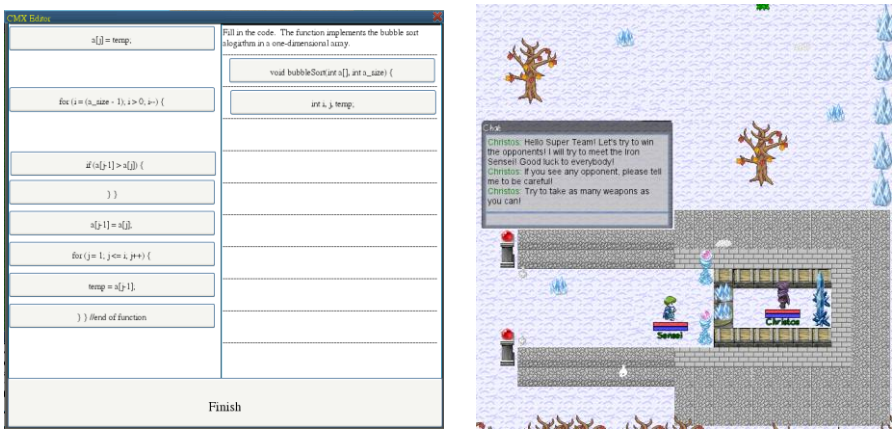

Fig. 3. CMX MMORPG interface

When the game is completed, teachers can access analytical reports documented by the system, where they can easily see which students gathered satisfying scores, which stages they did not manage to pass and where teachers should focus on their teaching.

Players use keyboard buttons in order to navigate through the game, as it has been

established in the majority of computer games. More specifically, they use the arrows in order to move in the four directions, the Alt button in order to chat and collaborate with the other players and the Ctrl button in order to attack their opponents. Finally, facilitating shortcuts have been added, which the players can customize themselves and use in whatever way it may suit them (e.g. use a specific button to activate an element of the game, like restore their health etc).

As it has been mentioned in the descriptions of all game's components, each phase, character, action and graphical environment require proper resource management and correct real-time feedback so that all players will have the same visualization of the game's and their personal status as well as on the messages transmitted.

4. System analysis and performance optimization

The architecture on which most MMORPGs are based comprises of one or more servers and several clients inter-connected through internet access. The server is usually protected by a firewall or a proxy server mainly for stability and safety reasons. The server administrator is responsible for ensuring the stable performance of the system and the optimization of the data stored in the server's database. This data is gathered by the game's system, documented in the database connected with the game and it includes information about each player, such as the player's status, the world's condition, the possible actions that can be executed as well as any instances created by the chat function usually embedded within MMORPGs.

More specifically, the player's status includes all elements that relate to the specific player. Such elements can be the player's statistics which are constantly renewed along with his/her corresponding evolvement during the game (e.g. current difficulty level of programming concepts, health points, strength etc.), the player's position in the virtual world and all the game objects under the player's possession. The world's status comprises of all the characteristics of the virtual world, such as the world map, the virtual players (e.g. intelligent agents) that are programmed by the game creators as well as actions that take place by each player. The actions are all the possible moves and operations that a player can realize or that are occurring at a specific moment (e.g. a spell is being cast; a player's character is walking etc.). These actions involve both the ones carried out by the actual players and agents. Finally, each MMORPG contains groups of players, thus most games include a chat tool that supports the group members' communication through the broadcasting of messages amongst them.

As the number of active players increases, the chat function that transmits simple text messages continues to have a relative small complexity, so it should not be taken into consideration during the search for the system's complexity simplification. Moreover, the internet connection speed will also not be considered as a factor, since it could vary depending on the client and its changes cannot be predicted.

It is important that the system hosting a MMORPG is studied for the identification of all factors that influence its performance and thus could potentially lead to system overload. This way, we can determine which elements to take into consideration during load balancing efforts. To this end, this section will propose a mathematical model for the appropriate estimation of these factors and the subsequent configuration of the

system should their values pose a risk to the game's successful execution by learners.

For example, if we assume that we have one player that is playing in a MMORPG, then the server's performance would depend on the player's status, the world's status and the possible actions that are caused or that can be caused. All these elements are recorded and documented in a database installed within a database server inside the main server; so the more the data produced during a game's operation the heavier the capacity transferred from the individual clients to the main server and stored into the database server. Hence, the rate in which the above elements' values change can significantly affect the server's overall performance and potentially obstruct the game's proper execution and subsequently the learning process's successful realization. To this end, we are proposing a model that will estimate the computational cost and the server administrators can take the appropriate actions for possible server upgrading based on these estimations (i.e. dynamic prediction of resources commitment).

This model includes the usage of an algorithm that measures and grades the players' behaviour, i.e. the value changes of the varying factors that we are examining for each player as they are being stored in the system's database. The indicators can take values between the scale [0.1, 10] and are considered to be versus time, which means that we are recording the activity occurring across time, as transmitted from clients to the server. This activity's value is stored as a vector in the database as shown below:

$$\overrightarrow{S_{CS}(t)} = \left\langle P^{cs}(t) \middle| W^{cs}(t) \middle| A^{cs}(t) \right\rangle = \left\langle s_1^{cs}(t) \middle| s_2^{cs}(t) \middle| s_3^{cs}(t) \right\rangle \quad \textbf{(3)}$$

CS is the indicator of the score that suggests the client-server message transmission, while t is the time parameter, since the values that these variables take, change through time. The S_{cs} (Score) is a vector that corresponds to each player and indicates how active the player is in the game's progression. This score is dependent on the player's status' change rate as opposed to a) the other players' status (variable P^{CS}), b) the virtual world (variable W^{CS}) and c) the possible actions that can be executed (variable A^{CS}). With $P^{CS}(t) \mid W^{CS}(t) \mid A^{CS}(t)$, we indicate that in every time instance the variables' values change and each new value is documented in the database over time. We further map the P^{CS}, W^{CS} and A^{CS} variables to the indices x = {1, 2, 3} respectively in order to alleviate some of the complex representation of the relation and, thus ratings are finally represented as s_x. For example, in a game, when a player moves towards the opponent, the changes that occur are the following: a) his relative position towards the opponent as he is recorded in his screen, b) his absolute position for the virtual world and finally, c) the possible actions that can be executed and the results that can be presented, e.g. when he is near enough he can damage the opponent, an action that could not have happened if he had been far away. Similarly, in a multiplayer educational game oriented to learning programming if a player tries to make moves or other actions by writing lines of code, the changes that occur are relative to the other players' situation and their code should be configured accordingly.

Moreover, we can estimate that the metric of the activities' score for each player at a t_i time by the c client to the s server is the sum of the P^{CS}, W^{CS} and A^{CS} metrics. In order to avoid extreme values, we logarithm them, so that they end up being in a [-1, 1] range, where the negative values mean that the player is less active, while the more the values tend towards 1, the more active the player is. This process is mathematically portrayed by the following relation:

$$s_{cs}(t) = \left|\overrightarrow{S_{cs}(t)}\right| = \sum_{i=1}^{3}\left[\log s_{\mathrm{i}}^{\mathrm{cs}}(\mathrm{t})\right] \quad (4)$$

Each player's final score is configured by his interaction with the other players as well as by the agents. The difference lies in that we can shape the system players' behaviour so that we ensure optimized load balancing, whereas we cannot intervene in the agents' behaviour, as it is automatically determined by the game system.

Finally, we define two indicators, namely the u_s indicator that symbolizes the number of the player's interaction with the game's agents and the u_o indicator that symbolizes the number of the player's interactions with the other players. So for a player a, the final relation is configured as follows:

$$TS_{cs}(t) = \frac{u_s}{u_s + u_o} \cdot \frac{\sum_{\forall t_i < t}\left[s_{cs}(t_i) \cdot t_i\right]}{\sum_{\forall t_i < t} t_i} + \frac{u_o}{u_s + u_o} \cdot \sum_{\forall j \neq a} \frac{\sum_{\forall t_i < t}\left[s_{ja}(t_i) \cdot t_i\right]}{\sum_{\forall t_i < t} t_i} \quad (5)$$

The above relation estimates the Total Score (TS_{cs}) for a player, which depends on values of the elements stored in our game database. In the above relation of the Total Score (TS_{cs}), the first sum refers to the player's (α) interaction with the system (e.g. movements inside the environment, actions towards game's agents), while the second part of the relation refers to the player's (α) interaction with the rest of the players (j where $j \neq a$) for each previous time instance $(\forall t_i < t)$. With the addition of the $\frac{us}{us + uo}$ in the relation we measure the multitude of the user's interaction with the system while with the $\frac{uo}{us + uo}$, we measure the multitude of the user's interaction with the rest of the players. It is also important to mention that since these variables change through time, the last values they take (i.e. the most recent) should be considered more in the final value of the player's total score. To represent this, as we show in the relation, we multiply the rating calculated at a specific time instance t_i, with itself. For example, an instance of the database table that holds this information is depicted in the following Table.

Table 1. MMORPG database instance of a player's recorded activity

PlayerID	P^{CS}	W^{CS}	A^{CS}	t_i	u
2	0.2	0.1	0.6	1	S
2	0.5	0.1	0.4	2	S
2	0.3	0.1	0.3	3	O

Let us assume that we are trying to measure the total score for the player with an ID 2. This player has executed activities during three different time instances within the game as shown in the t_i column. Moreover, the u column shows that two of these activities include interaction with the system, while the last one represents interaction with other players of the game. Using the proposed model, the value of the total score will be estimated taking into consideration the fact that the player's interaction with the system's agents (the first two rows that have the value u=S) has increased in the 2nd time instance, which shows he is becoming more active and thus could need more resources

in the future. According to the above table and the relation (5) of the model, the total activity score for the player with ID=2 and instance t_i=4 is $TS_{CS}(4)$=0.44, which means that his behaviour is very good and according to the goals of the game. We could easily understand that without the algorithm we would have to use the same quantum of resources for all the players, but now that we know their behavior, we could easily customize the game's responses and the corresponding needed resource commitment.

Next, we demonstrate a way to reduce the big computational cost at run-time indicated by the double sums in the last relation of the previous section. The run-time computational complexity of relation (5) is O(*n***y*), where *n* is the number of agents acting in the system and *y* is the total amount of transactions. Both of these numbers increase rapidly in a MMORPG game. More specifically, the $\frac{\sum_{\forall t_i<t}\left[s_{cs}(t_i)\cdot t_i\right]}{\sum_{\forall t_i<t} t_i}$ sum does not need to be estimated at run time, but incrementally, as each rating vector is gathered and stored. We re-symbolize this sum with the $\sum_{i=1}^{3}\left[\sigma_i^{CT}(t)\right]$ variable. Additionally, the $\sum_{\forall j\neq a}\frac{\sum_{\forall t_i<t}\left[s_{ja}(t_i)\cdot t_i\right]}{\sum_{\forall t_i<t} t_i}$ sum can also be estimated incrementally and it will be replaced with the $\sum_{i=1}^{3}\left[\sigma_i^{jb}(t)\right]$ variable. This way, relation (5) is transformed into the following:

$$TS_{CS}(t)=\frac{u_s}{u_s+u_o}\cdot\sum_{i=1}^{3}\left[\sigma_i^{CT}(t)\right]+\frac{u_o}{u_s+u_o}\cdot\sum_{i=1}^{3}\left[\sigma_i^{jb}(t)\right] \quad \textbf{(6)}$$

This relation's complexity is O(*n*), where n is the total number of agents and players inside the system. The sum in the three dimensions does not count because its limit is stable and already known that it is 3. Thus, we are left with only a sum inside the second addendum. The initial relation (5) has a run-time complexity of O(*n***y*), i.e. the number of agents and players (*n*) over the number of transactions (*y*), which is much larger than the complexity of relation (6).

In conclusion, the optimization of the algorithm is possible because the player's active score is calculated and stored in the database as a total number, instead of the system calculating the score's value after each player movement.

5. Evaluation

In order to evaluate our model, we use a specially designed test bed in a 3GHz Core 2 Duo Server with 4GB RAM. We ran our scenario two times, maintaining the exact same conditions in both cases. However, in the first case, we do not exploit the optimization algorithm and in the second case we do. More specifically for the second case, we investigate if and how much the server's performance improves, while at the same time adding virtual players inside the game.

5.1. Test bed overview

We used a special virtual world without any obstacles inside CMX, called "Arena" and we placed many virtual players with behaviours of escalating randomness. This was done in order to simulate the randomness of the players and create a realistic experiment. We moved on to creating additional virtual players with predictable behaviour, so that the appropriate resources can be committed according to their behaviour. More specifically, we created virtual players with predictable behaviour so that we can commit specific resources based on that behaviour, and a group of virtual players with completely random behaviour, which will simulate a possible un-predictable behaviour of an actual player at a specific period.

We continued to load more players in the virtual world and study the server's reaction without and with applying the algorithm. In the first case, as we add the virtual players, equal resources for each player will be committed without any distinctions (e.g. for 100 players, 1% of the entire CPU usage will be used to accommodate the individual traffic). However, this could lead to the engagement of redundant resources because not all players need the same amount of CPU usage, depending on their actions in the game. Thus, in the second case, with using our algorithm, we store in the database features such as how active each player is etc, as mentioned in the previous section. This way, the exact amount of resources needed per player will be calculated and committed.

5.2. Use case scenario

The test bed in the experiment was populated with excellent, good, fair and bad virtual players, depending on their level of computer programming prior knowledge. More specifically, 100 virtual players were used, 10 of which were excellent, 30 good, 20 fair players and 40 bad players. By excellent players we mean players with a completely predictable behaviour, where we know for example, that when such a player at a time t_1 is at a specific location in the virtual world in pursuit of a task, this player will continue walking $\forall t_i > t_1$ times until the goal is achieved. A good player is also navigating through the game in a predictable way; however, there is a small chance that some actions will be not predicted by the server, such as a change of course in the Arena. The probability that a behaviour will be unpredictable increases with fair players, while bad players' actions are completely random. These players constantly make non sequential movements and are not aiming to fulfil the educational game's objectives.

We included several players with unpredictable behaviour in our experiment because we wanted to test our algorithm in extreme situations where the server's overloading with resources, since we already can estimate that in excellent and good players, the game's performance will be adequate. Thus, we endeavour to prove that in the case of an excellent or good player, the minimum possible resources are bound in the CPU, while these increase in the case of fair and bad players. As an overall result, the amount of resources needed with the usage of the optimization algorithm should be significantly less than if the same amount of resources had been bound for all players.

We positioned a player on the "hackers" team and one in the "crackers" team in the game's Arena. Following, we equally populated both teams with virtual players that

exhibited similar behaviour, while monitoring and recording the sent and received messages packets as well as the CPU usage levels. The players of each team tried to win all other players of the opposite team while executing assigned tasks and gathering weapons and spells. It should be noted that we provided unlimited lives to our virtual players to prevent possible elimination from the game.

During a solely entertaining multiplayer game with the same conditions as the ones we have established in our experiment (e.g. 100 players, unpredictable behaviour of movements, constant transmission of multiple resources), a possible overloading of the system would most likely last a few seconds. This is because, in most cases, the majority of the students lose and exit a game rather quickly and thus the resources committed get reduced 6. However, in an educational setting we require the majority of the students to go through all the game's levels in order to obtain the maximum possible knowledge and skills. Thus, any latency on behalf of the server could significantly affect the learning process, the students' motivation as well as the entire game's configuration.

A server's operation is significantly increased when the players interact with each other in the Arena while it is less extreme during the training phase. This is because a large amount of resources are required during the players' confrontations with each other such as each player's absolute and relative position that is occupied in the same screen of the virtual world. This information flow needs to be constant since the data is ever changing, which increases the server's CPU usage.

5.3. The results

The experiment was executed without any issues, and we closely monitored all the information recorded in the server's log file. We initially performed the CMX scenario as-is and thereafter performed the use case scenario that involved the usage of the optimization algorithm. The results gathered are presented in Fig. 4 and they show that when the first 10 players with excellent behaviour are inserted in the Arena, the optimization algorithm seems to reduce the CPU usage by about 5%, since their behaviour has been programmed to be predictable, and thus the algorithm knows what resources they need. The algorithm starts monitoring even more the server's resources when additional players are inserted, even though their behaviour is not completely predictable.

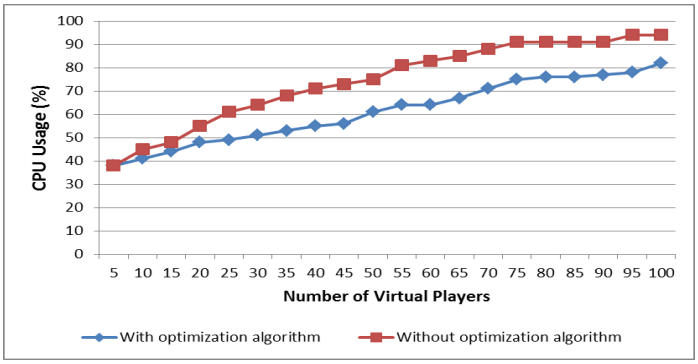

Fig. 4. CPU Usage with and without optimization

Based on the results, the first 60 players have predictable behaviour, since they play according to the game's rules. With these players, we observe that the algorithm is constantly reducing the CPU's computing cost. More specifically, the CPU usage is reduced down to 20% when 60 players in total are a part of the game's environment. However, when we start inserting virtual players with totally unpredictable behaviour, the algorithm's performance is decreased since it cannot predict what resources should be committed. For this reason, we can see three significant drops in Fig. 5, which occur when players with more unpredictable behaviour are inserted. Nonetheless, Fig. 5 also shows that the algorithm manages to maintain an approximately 15% of CPU usage reduction.

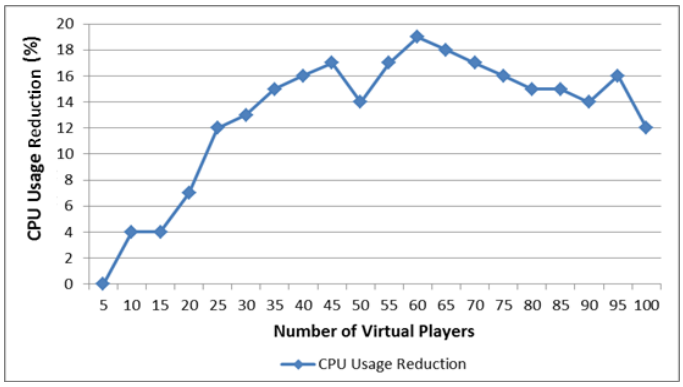

Fig. 5. CPU Usage Reduction

It should be noted that 15% of CPU usage reduction is a significantly satisfying result for the server's performance, as with the corresponding release of resources, the game can operate smoothly and without performance issues and can also host more players. This optimization ratio will remain stable even if the players are dramatically increased. This is because our algorithm depends only on the behaviors of the players in the game and it already takes into consideration different types of behaviours that new players could show (excellent, good, fair and bad). Similarly, the second diagram shows how the CPU's computing cost reduction is increased. It is worth mentioning that the low reduction values shown in the diagram occur due to the addition of new users with constantly extra random behavior.

Furthermore, it should be noted that when we use the CMX's server optimization algorithm, the RAM usage is slightly increased since we store more information; however, in an educational game this increase is very low compared to the CPU usage, and therefore it does not affect negatively on the game's operation.

6. Conclusions

The current paper aimed to propose a new methodology that can achieve optimization of the load balancing in educational MMORPG for computer programming and the bestowal of the proper and only necessary resources without overloading the system. To

this end, we analysed existing published works that identified the factors that influence a computer game's performance. These works, however, focus mostly on network traffic of strategy or shooting games and less on MMORPGs that require more resources, as it is usually the case in the educational domain. Thus, the existing studies could not be used as a comparison for the model presented in our paper.

Our model is based on the notion that not all players are equally active in a learning process and in a game's activities, and thus they do not require the same amount of resources committed by the main server to their client. Therefore, we monitor a player's activity along with other parameters within the game. In the end, we manage to commit only the essential resources for each player, where traditionally we would commit the same amount resulting in abundant resources' commitment and thus server overloading.

The novelty of our approach is the cumulative storage of information in the database in an abstract manner, in order to exploit in the maximum degree the gathered information regarding the players' predictable behaviour and thus ensure the proper commitment of CPU and RAM memory. It should be noted that the reasoning behind the exploitation of the cumulative information can be applied in any MMORPG in combination with algorithms that address the optimization of network resources and packet transmissions for the optimum system's performance.

Moreover, we presented an educational game called CMX, which aims to facilitate learning and teaching of computer programming, we analysed the game's phases, namely training and playing, and elaborated on the elements and functions related to each phase. This way, we indicated the importance of optimizing such a server's performance to ensure consistent and responsive learning processes.

Finally, we applied the proposed algorithm to CMX by creating a use case scenario and inserting virtual players instead of using students. This way, we could control the entire process on the server's point of view, since all virtual players' behaviours were simulated in accordance to the types of behaviours usually presented by students. The results of the test bed indicated that indeed the optimization algorithm managed to decrease the CPU usage by a total amount of 15%, which should also be the case during the implementation of the game in educational settings. Due to the fact that a more accurate and dynamic prediction of resources needed is carried out with the algorithm, in total less resources are committed than the fixed amount of resources that are usually committed by default. Future objectives include the integration of our approach in the CMX's code ran on the client side for the optimization of their performance, as well as the documentation of the results on the client and client-server level.

References

1. Armitage. G.: "An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3", 11th IEEE International Conference on Networks, Sydney, Australia. (2003)
2. Bennane, A.: Adaptive Educational Software by Applying Reinforcement Learning. Informatics in Education, Vol. 12, No. 1, pp. 13-28. (2013)
3. Bernier Y.W.: Latency Compensating Methods in Client/ Server In-game Protocol Design and Optimization, Game Developers Conference, http://www.gdconf.com/ archives/proceedings/2001/prog_papers.html (2001)

4. Borella, M. S.: Source Models of Network Game Traffic. Computer Communications, 23(4):403-410. (2000)
5. Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., & Miller, P.: Mini-languages: a way to learn programming principles. International Journal of Education and Information Technologies, 2, 65–83. (1997)
6. Chang, F. and Feng, W.: Modeling Player Session Times of On-line Games, ACM NetGames 2003. (2003)
7. Chen, K-T., Huang, P., Lei, C-L.: Game traffic analysis: An MMORPG perspective, Computer Networks, Volume 50, Issue 16, pp. 3002-3023, doi: http://dx.doi.org/10.1016/j.comnet.2005.11.005. (2006)
8. Farber, J.: Network Game Traffic Modelling, Proceedings of the first workshop on Network and system support for games. (2002)
9. Gee, J.P.: Why game studies now? Video games: A new art form, Games and Culture, Volume 1 Number 1. (2006)
10. Gunter, G. A., Kenny, R. F., & Vick, E. H.: Taking educational games seriously: using the RETAIN model to design endogenous fantasy into standalone educational games. Educational Technology Research and Development, 56(5/6), 511-537. (2008)
11. Guo, K., Mukherjee, S., Rangarajan, S. and Paul, S. A fair message exchange framework for distributed multi-player games, Proceedings of the 2nd workshop on Network and system support for games, p.29-41, Doi 10.1145/963900.963903. (2003)
12. Jehaes, T., Vleeschauwer, De D., Coppens , T. Van Doorselaer, B., Deckers, E., Naudts, W., Spruyt , K., Smets, R. Access network delay in networked games, Proceedings of the 2nd workshop on Network and system support for games, p.63-71. doi: 10.1145/963900.963906. (2003)
13. Lahtinen, E., Ala-Mutka, K., Jarvinen, H.: A Study of Difficulties of Novice Programmers. In: Innovation and Technology in Computer Science Education, 14–18. (2005)
14. Long, J.: Just For Fun: Using Programming Games in Software Programming Training and Education. Journal of Information Technology Education 6: 279-290. (2007)
15. Ng Yu-Sheng: Designing Fast-Action Games for the Internet, Gamasutra (Online Game Developer Magazine). (1997)
16. Quax , P., Monsieurs, P., Lamotte, W., Vleeschauwer, De D., Degrande, N.: Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game, Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games, Portland, Oregon, USA. (2004)
17. Radosevic, D., Orehovacki, T., Lovrencic, A.: Verificator: Educational Tool for Learning Programming. Informatics in Education, Vol. 8, No. 2, pp. 261-280. (2009)
18. Shanahan, J.: Students Create Game-based Online Learning Environment that Teaches Java Programing, Publisher ACMSE. (2009)
19. Sheldon, N. Girard, E., Borg, S. Claypool, M. Agu. E.: The Effect of Latency on User Performance in Warcraft III, Technical Report WPICS-TR-03-07, Computer Science Department, Worcester Polytechnic Institute. (2003)
20. Singhal S. and Zyda, M.: Networked Virtual Environments: Design and Implementation. Addison Wesley, ISBN 0-201-32557, ACM Press. (1999)
21. Smed, J. Kaukoranta, T. Hakonen, H.: A Review on Networking and Multiplayer Computer Game, Turku Centre for Computer Science. (2002)
22. Stuikys, V., Burbaite, R., Damasevicius, R.: Teaching of Computer Science Topics Using Meta-Programming-Based GLOs and LEGO Robots. Informatics in Education, Vol. 12, No. 1, pp. 125-142. (2013)
23. Suznjevic, M., Dobrijevic, O., and Matijasevic, M.: MMORPG Player actions: Network performance, session patterns and latency requirements analysis. Multimedia Tools Appl., 45, 191-214. (2009)

24. Torrente, J., Blanco, Á. d., Serrano-Laguna, Á., Vallejo-Pinto, J. Á., Moreno-Ger, P., Fernández-Manjón, B.: Towards a Low Cost Adaptation of Educational Games for People with Disabilities. Computer Science and Information Systems, Vol. 11, No. 1, 369–391. (2014)

Christos Malliarakis (e-mail: malliarakis@uom.gr) holds a BSc in Applied Informatics from the University of Macedonia in Thessaloniki and an MSc in Informatics from the Computer Science Department of the Aristotle University of Thessaloniki. He is undergoing his PhD research in Game Based Learning and Game Design on Computer Programming since May 2011, supervised by Professor Maya Satratzemi and Lecturer Stelios Xinogalos. In the context of the undergoing PhD research, he has designed and developed CMX, an educational MMORPG for learning computer programming. He has worked in Primary and Secondary education for 5 years in Mandoulides Schools, a large private school located in Thessaloniki, Greece.

Maya Satratzemi (email: maya@uom.gr) received the BS degree in math from the Aristotle University of Thessaloniki, and the PhD degree in informatics from the University of Macedonia. She is a professor at the Department of Applied Informatics, University of Macedonia. She has published more than 40 papers in journals & book chapters and 80 papers in conferences proceedings. Her current research interests include Programming Environments and Techniques, Adaptive Systems, Serious Games, Game Based Learning, Collaborative Programming – Distributed Collaborative Programming Environments, Educational Technology, Graph Algorithms and Applications.

Stelios Xinogalos (e-mail: stelios@uom.gr) is a lecturer at the Department of Applied Informatics, School of Information Sciences, University of Macedonia, Thessaloniki, Greece. He is author or co-author of 70 research papers on programming environments and techniques, object-oriented design and programming, educational programming environments and educational games for programming, most of which are published in international conferences and journals. He has also designed and implemented two educational programming environments for procedural and object-oriented programming based on 'Karel the Robot'.

Received: December 20, 2013; Accepted: June 14, 2014