Robust Resource Allocation in a Massive Multiplayer Online Gaming Environment

Luis Diego Briceño[1], Howard Jay Siegel[1,2], Anthony A. Maciejewski[1], Ye Hong[1], Brad Lock[1], Mohammad Nayeem Teli[2], Fadi Wedyan[2], Charles Panaccione[2], Chris Klumph[1], Kody Willman[1], and Chen Zhang[2]

Colorado State University
[1]Department of Electrical & Computer Engineering
[2]Department of Computer Science
Fort Collins, CO 80523
{LDBricen, HJ, AAM, YHong, Bradley.Lock, Mohammad.Teli, Fadi.Wedyan, Charles.Panaccione, Christopher.Klumph, Kody.Willman, Chen.Zhang}@colostate.edu

ABSTRACT

The environment considered in this research is a massive multiplayer online gaming (MMOG) environment. Each user controls an avatar (an image that represents and is manipulated by a user) in a virtual world and interacts with other users. An important aspect of MMOG is maintaining a fair environment among users (i.e., not give an unfair advantage to users with faster connections or more powerful computers). The experience (either positive or negative) the user has with the MMOG environment is dependent on how quickly the game world responds to the user's actions. This study focuses on scaling the system based on demand, while maintaining an environment that guarantees fairness. Consider an environment where there is a main server (MS) that controls the state of the virtual world. If the performance falls below acceptable standards, the MS can off-load calculations to secondary servers (SSs). An SS is a user's computer that is converted into a server. Four heuristics are proposed for determining the number of SSs, which users are converted to SSs, and how users are assigned to the SSs and the MS. The goal of the heuristics is to provide a "fair" environment for all the users, and to be "robust" against the uncertainty of the number of new players that may join a given system configuration. The heuristics are evaluated and compared by simulation.

Keywords

robustness, MMOG, client/server, peer-to-peer, heterogeneous, FePIA, resource allocation

1. INTRODUCTION

The environment considered in this research is a massive multiplayer online gaming (MMOG) environment. Each user controls an avatar (an image that represents and is manipulated by a user) in a virtual world and interacts with other users. An important aspect of MMOG is maintaining a fair environment among users (i.e., not giving an unfair advantage to users with faster connections or more powerful computers). The experience (either positive or negative) the user has with the MMOG environment is dependent on how quickly the game world responds to the user's actions.

A problem may occur when considering interaction with other users. For example, consider a war game where two users are shooting at each other. One way of determining the winner of this contest is to determine who shot first. However, determining who shot first in the game world can be difficult. It is possible for the game to process these users' actions in the incorrect order.

In general, most MMOG environments use a client/server architecture to control the virtual game world. This has some disadvantages: the initial procurement of servers is expensive, server administration is required, customer service is necessary, and the architecture is hard to scale based on demand. Other factors such as the popularity of a game, and unexpected technical problems during and after the launch, can also affect the final cost and success of an MMOG environment [26].

The performance of the heterogeneous system used to simulate the game world must not degrade beyond acceptable parameters even if the MMOG environment is oversubscribed. In this system, the number of players that may join a given system configuration (after the game session has started) is uncertain. The goal of the heuristics is to provide a "fair" environment for all the users, and to be "robust" against this uncertainty. A fair environment will ensure a high quality gaming experience for all connected users.

This study focuses on scaling the system based on demand, while maintaining an environment that guarantees fairness. Consider an environment where each of N users produces a data packet that needs to be processed. There is a main server (MS) that controls the state of the virtual world. If the performance falls below acceptable standards, the MS can off-load calculations to secondary servers (SSs). An SS is a user's computer that is converted into a server to avoid performance degradation. These SSs will be employed if necessary by the MS to guarantee fairness.

The introduction of SSs causes the game-state to be handled differently than with a single MS. Each SS handles conflicts among the players attached to it, and sends conflict-free information to the MS. However, this information may conflict with information from another SS. If there is a conflict between SSs then it will be resolved by the MS.

This study assumes all players are willing to become SSs. Our approach could easily be adapted to account for having a subset of players who are not willing to be an SS, i.e., we can have a list of

This research was supported by the NSF under grant CNS-0615170 and by the Colorado State University George T. Abell Endowment.

ICFDG 2009, April 26-30, 2009, Orlando, FL, USA.

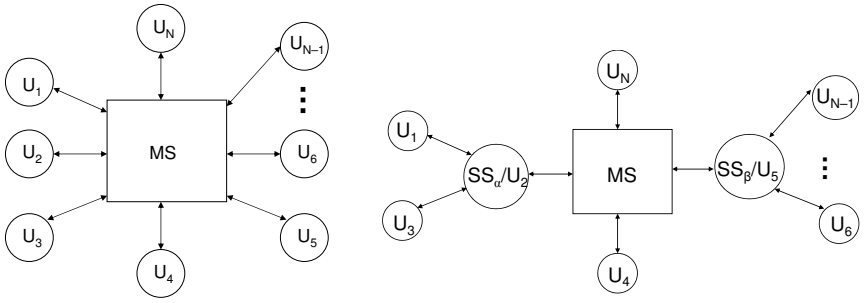

Figure 1: (a) Client/Server Architecture versus (b) Secondary Server Architecture

players eligible to become SSs.

A session in the MMOG environment is assumed to last for an extended period of time, with a small break between sessions [23]. Recall that players may join during a session and it is assumed, for the study, that players do not leave during a session because they will be rewarded at the end of the session. The small break, prior to the beginning of the session, can be used to determine which users are SSs. These assumptions make a static resource allocation heuristic viable [1].

In this study, four static resource allocation heuristics are proposed for determining the number of SSs, which users are converted to SSs, and how users are distributed among the SSs and the MS. The assignment of users to SSs and the MS is related to the assignment of tasks to machines (e.g., [4, 7, 27, 29]) with the SSs and the MS as machines and the remaining users as tasks. A mathematical upper bound is used to evaluate the performance of the heuristics. The contributions of this paper are: (a) studying and simulating an MMOG environment where an unpredictable number of players may want to join an ongoing session, (b) creating parameters to quantify the robustness of a system against the uncertainty of the number of players that will try to join an ongoing session, and (c) deriving resource allocation heuristics that maximize the number of players that can join an existing game session while still maintaining a fair system.

This paper is organized as follows. Section 2 provides the problem statement and describes the performance metrics. In Section 3, we focus on the four proposed heuristics. We provide the related work in Section 4. Our results for the four heuristics are shown in Section 5 and in Section 6 we present our conclusions.

2. PROBLEM STATEMENT

2.1 Environment

In this study, we will consider an MMOG environment where the performance of a user is sensitive to latency [3]. The purpose of this research is to maintain "robust" system performance (despite the MS used to maintain the MMOG environment being oversubscribed) without increasing the processing power of the MS. The robustness is described in detail in Section 2.4.

The goal of the heuristics is to provide an environment where the differences in latency among all users are bounded by a quality of service (QoS) constraint. This QoS constraint is based on human perception (i.e., the difference in response times between players is imperceptible). If the QoS is met then the environment provides a high-quality interactive experience. New players are users that join the game after the initial resource allocation and are connected to the MS. The latency for some users may increase above the QoS bound as new players join the game. The heuristics will provide a resource allocation that maximizes the number of new players that can be connected to the MS, while still maintaining the QoS for all users.

The proposed solution is to convert users to SSs that assist the MS in computation while maintaining performance constraints. In the client/server solution shown in Figure 1(a), all users connect to the MS, therefore the MS is the only machine performing computation. In the SS solution shown in Figure 1(b), the MS and SSs perform computation and the MS resolves conflicts among users and SSs connected to it.

The allocation of users as SSs has similar security, trust, and cheating issues as distributed servers and peer-to-peer based MMOG systems. There are various publications that study these issues in MMOGs, e.g., [5]. These issues will not be discussed here because we consider them to be separate research problems.

The following simplifying assumptions are made about the communication model in this system. The users are assumed to be on a fully connected network; however, the simulation could easily be adapted to consider different topologies. The communication time between different pairs of nodes (user computer, SS, or MS) will vary. The communication times among the users, SSs, and the MS do not change during a session. These times are independent of the number of users connected to an SS or the MS. These simplifying assumptions are used to reduce the complexity of the simulations.

2.2 Computational Model

To simplify the simulation study, the level of activity of all the users in the MMOG environment is considered identical (i.e., the frequency of interaction with the MMOG environment is the same for all players). Thus, the computational load is based on the number of users (i.e., they have the same computational needs). To model the computation times of the MS and SSs, we need to consider how the computation time increases with an increase in the number of users.

In [17], latency in an MMOG environment shows a "weak exponential" increase with an increase in players; we approximate this by using a constant communication time and a quadratic factor for the computation. Let n_α be the number of users connected to secondary server α (SS_α), and μ_α be a computational constant for SS_α that represents the heterogeneity in the computing power of the users' computers (each user has a different constant). The factor μ_α can represent the capabilities of a user's computer, and what portion of those capabilities the user is willing to devote to operating as an SS. The computation time for an SS_α ($Comp_\alpha$) can be modeled as:

$$Comp_\alpha = \mu_\alpha \cdot (n_\alpha)^2. \quad (1)$$

Let $n_{secondary}$ be the total number of users connected to all the SSs, n_{nss} be the number of SSs, n_{main} be the number of users connected to MS, and b and c be computational constants of the MS. The computation time of the MS ($Comp_{MS}$) is:

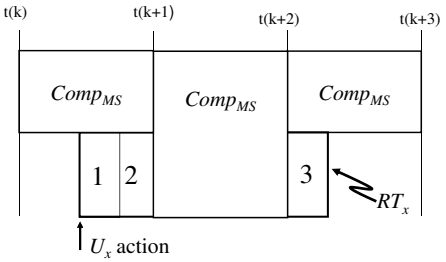

Figure 2: Visual representation of RT_x when U_x is connected directly to the MS: 1) Comm(U_x,MS), 2) δ, and 3) Comm(MS,U_x)

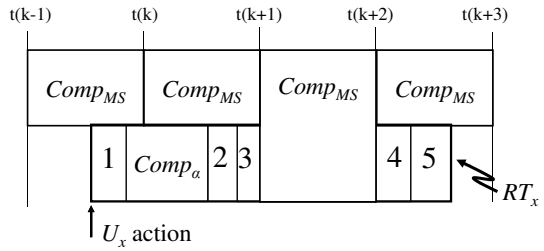

Figure 3: Visual representation of RT_x when U_x is connected to the MS through an SS_α: 1) Comm(U_x,SS_α), 2) Comm(SS_α,MS), 3) δ, 4) Comm(MS,SS_α), and 5) Comm(SS_α,U_x)

$$Comp_{MS} = c \cdot n_{secondary} + b \cdot (n_{main} + n_{nss})^2. \quad (2)$$

2.3 Objective Functions RT_{max} and RT_{min}

Let $\underline{RT_x}$ represent the Response Time (RT) of a packet (representing an action in the game world) sent by the computer of user x ($\underline{U_x}$) to the MS (possibly through an SS) and returning to U_x with the corresponding consequence of that action in the game world. Let $\underline{Comm(A,B)}$ be the communication time between node A and node B, and $\underline{\delta}$ is the time a packet has to wait before being processed. The equation used to calculate RT_x if U_x is connected directly to the MS is:

$$\begin{aligned} RT_x &= Comm(U_x, MS) + \delta \\ &+ Comp_{MS} + Comm(MS, U_x). \end{aligned} \quad (3)$$

A graphical representation of this equation is shown in Figure 2. A computation cycle of the MS starts at time t(i) and finishes at time t(i+1). This cycle processes all the unprocessed actions received before time t(i). If a user is connected to an SS_α then the equation is:

$$\begin{aligned} RT_x &= Comm(U_x, SS_\alpha) + Comp_\alpha + \delta \\ &+ Comm(SS_\alpha, MS) + Comp_{MS} \\ &+ Comm(MS, SS_\alpha) + Comm(SS_\alpha, U_x). \end{aligned} \quad (4)$$

A graphical representation of this equation is shown in Figure 3. If U_x is SS_α then Equation 4 is used with $Comm(U_x, SS_\alpha) = Comm(SS_\alpha, U_x) = 0$.

For this work, we will use the difference between RT_{max} and RT_{min} to quantify the fairness in the MMOG environment. The RT_{min} and RT_{max} features determine the robustness and depend on the resource allocation. To calculate RT_{max} we use:

$$RT_{max} = \max_{\forall U_x} (RT_x), \quad (5)$$

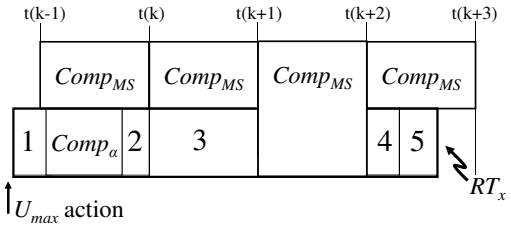

Figure 4: Visual representation of RT_{max}: 1) Comm(U_{max},SS_α), 2) Comm(SS_α,MS), 3) δ, 4) Comm(MS,SS_α), and 5) Comm(SS_α,U_{max})

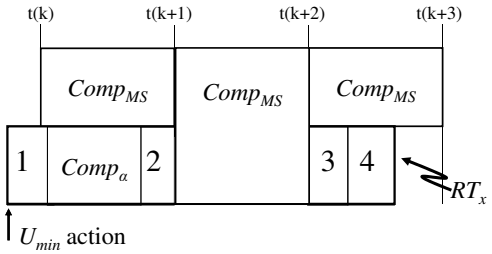

Figure 5: Visual representation of RT_{min}: 1) Comm(U_{min},SS_α), 2) Comm(SS_α,MS), 3) Comm(MS,SS_α), and 4) Comm(SS_α,U_{min})

with $\delta = Comp_{MS}$. As shown in Figure 4, this time represents the maximum time any user will have to wait for a response from the MS (worst case RT). Let $\underline{U_{max}}$ represent the user with RT_{max}, and $\underline{U_{min}}$ the user with RT_{min}. Figures 4 and 5, show examples where U_{max} and U_{min} are connected an SS_α. A visual representation of RT_{min} is shown in Figure 5 and it is calculated as follows:

$$RT_{min} = \min_{\forall U_x} (RT_x), \quad (6)$$

with $\delta = 0$. This time represents the fastest any user can interact with the MMOG environment.

2.4 Robustness Metric

2.4.1 Overview

Using the FePIA (Performance Features, Perturbation Parameters, Impact, and Analysis) procedure described in [2], we define the characteristics that make the system robust. The FePIA procedure should respond to three fundamental questions. First, what behavior of the system makes it robust? Second, what uncertainties is the system robust against? Quantitatively, exactly how robust is the system?

2.4.2 Performance Features

The first step of the FePIA procedure is to describe quantitatively the QoS requirement that makes the system robust. The requirement that makes the system robust is that all the RTs are within a pre-determined range. The maximum RT time the system can allow is $\underline{\beta_{max}}$:

$$RT_{max} \leq \beta_{max}. \quad (7)$$

However, to maintain fairness RT_{min} also has a constraint. A time window ($\underline{\Delta_{max}}$) is used to specify the allowable range of RT_x for all users. The constraint that RT_{min} must meet is:

$$RT_{max} - RT_{min} \leq \Delta_{max}. \quad (8)$$

For the system to be robust the constraints shown in Equations 7 and 8 need to be satisfied.

2.4.3 Perturbation Parameters

The second step of the FePIA procedure is to determine the perturbation parameters that represent the uncertainties in the system. For this study, the perturbation parameter is the number of new players joining the game after the initial resource allocation is done.

2.4.4 Impact of Perturbation Parameters on the QoS Performance Features

In this study, it is assumed that new players joining a game in progress connect to the MS. When new players join, the computation at the MS will increase quadratically. This increase in time will make the RT of users that are already in the game increase, and hence RT_{max} will increase. When new players are added to the MS, the RT_x for all players increases equally. Thus, if the initial resource allocation satisfies Equation 8, then it will remain satisfied. Also, U_{max} will be the first player to violate the robustness constraint shown in Equation 7.

Let RT_{new} be the RT for a new player. We assume the system does not allow new players whose response time exceeds RT_{max} (i.e., $RT_{new} < RT_{max}$); or violates the fairness criteria (i.e., $RT_{max} - RT_{new} \leq \Delta_{max}$).

2.4.5 Analysis

The increase in the number of new players that can be added to the system before RT_{max} violates the QoS constraint can be calculated. Using equations 3, 4, and 5, we can calculate how many new players can be added before the robustness constraint is violated. We define Γ as the components of the RT equation that do not depend on the number of players connected to the MS. When U_x is connected to the MS, Γ is given by:

$$\Gamma = Comm(U_x, MS) + Comm(MS, U_x), \tag{9}$$

and if U_x is connected to SS_α then Γ is:

$$\begin{aligned} \Gamma &= Comm(U_x, SS_\alpha) + Comp_\alpha + Comm(SS_\alpha, MS) \\ &+ Comm(MS, SS_\alpha) + Comm(SS_\alpha, U_x). \end{aligned} \tag{10}$$

Therefore,

$$RT_{max} = \Gamma + 2 \cdot Comp_{MS}. \tag{11}$$

The system will be at the boundary of robustness when RT_{max} is equal to β_{max} with $\delta = Comp_{MS}$, that is

$$\beta_{max} = \Gamma + Comp_{MS} + \delta, \text{ or} \tag{12}$$

$$\beta_{max} = \Gamma + 2 \cdot Comp_{MS}\ . \tag{13}$$

Let $\underline{y} = n_{main} + n_{nss}$ and $\underline{x}$ represent the number of new players that can be added. This implies that

$$\beta_{max} = \Gamma + 2 \cdot (c \cdot n_{secondary} + b \cdot (y + x)^2)\ . \tag{14}$$

The quadratic term can be expanded so that

$$\beta_{max} = \Gamma + 2 \cdot (c \cdot n_{secondary} + b \cdot (y^2 + 2 \cdot y \cdot x + x^2)). \tag{15}$$

Using Equation 11, this can be simplified to

$$\beta_{max} = RT_{max} + 2 \cdot b \cdot (2 \cdot y \cdot x + x^2). \tag{16}$$

This can be re-written in standard quadratic form:

$$2 \cdot b \cdot x^2 + 4 \cdot b \cdot y \cdot x + (RT_{max} - \beta_{max}) = 0. \tag{17}$$

With the roots given by the following equation, the robustness metric, the maximum number of new players that can be added, is quantified as:

$$x = \frac{-(4 \cdot b \cdot y) \pm \sqrt{16 \cdot b^2 \cdot y^2 - 8 \cdot b \cdot (RT_{max} - \beta_{max})}}{4 \cdot b}. \tag{18}$$

This result requires some interpretation, because it has two roots. If Equation 18 has two real roots, then the largest value is selected. If the largest value is positive then this is the number of players the current resource allocation can add without violating the QoS constraints. If the largest value is negative then this is the number of players that need to be removed for the system to become robust. If the roots generated by Equation 18 are complex then the robustness cannot be achieved due to excessive communication or computation at an SS. The value of the robustness metric is based on RT_{max} which is determined by the given resource allocation; hence, better resource allocation will result in larger values for the robustness metrics.

3. RESOURCE ALLOCATION HEURISTICS

3.1 Overview

Heuristics for determining an allocation of resources are presented in this section. Recall that resource allocation implies assigning a user in one of three ways: (1) attaching it directly to the MS without making it an SS, (2) attaching it to the MS and making it an SS, or (3) attaching it to an existing SS. An unassigned user is one that has not been assigned yet. Directly connected users (DCUs) are users that are connected directly to the MS (cases (1) and (2) above).

For some heuristics, it is necessary to give a "robustness" value to all solutions. If the solution cannot achieve robustness (i.e., Equation 18 has two complex roots), then we approximate the robustness. In this case, the robustness is calculated as:

$$x = \frac{-\sqrt{RT_{max} - \beta_{max}}}{\sqrt{2 \cdot b}}. \tag{19}$$

This gives a negative bias to all the solutions that cannot reach robustness.

3.2 Min-Min RT

The Min-Min RT sub-heuristic is based on the concept of the Min-Min heuristic [16]. The Min-Min heuristic is widely used in the area of resource allocation (e.g., [8, 12, 13, 16, 19, 20, 22, 29]). This sub-heuristic requires a set of DCUs to generate a resource allocation, therefore it is not an independent heuristic. It is used by other resource allocation algorithms to obtain a full mapping. The procedure to implement the Min-Min RT is shown in Figure 6.

3.3 Min-Min SS

The Min-Min SS heuristic is similar to the Min-Min RT heuristic. The difference is that the Min-Min SS does not require an initial set of SSs. The heuristic will determine the set of SSs by allowing users to connect to the MS in step (2) of Figure 6 (if this assignment has the minimum RT).

3.4 Discrete Particle Swarm Optimization

Discrete Particle Swarm Optimization (DPSO) is based on the particle swarm optimization in [21]. The authors in [25] implemented a discrete version of the particle swarm optimization in [21], upon which we base our implementation. Intuitively, this algorithm samples the search space of possible SS configurations, and then uses the Min-Min RT algorithm to generate a complete

(1) Given a predetermined set of DCUs, all users that are not in the set of DCUs are marked as unassigned.
(2) For each unassigned user, the DCU that gives the minimum RT is determined (first minimum).
(3) The user/server with smallest RT among all the pairs generated in (2) is selected (second minimum).
(4) The user in the pair selected in (3) is then assigned to its paired server.
(5) Steps (2) through (4) are repeated until all tasks are assigned.

Figure 6: Procedure for Min-Min RT

mapping from a set of SSs. The algorithm used to implement DPSO is shown in Figure 7.

In DPSO, particles represent solutions. Each particle is composed of N entries (each entry represents a user). Let $X_{ij} \in \{0, 1\}$ represent whether user j is a DCU ($X_{ij} = 0$), or a non-DCU ($X_{ij} = 1$) in particle i. Particles move around through different possible solutions based on how their velocity is composed. The direction of the velocity will determine whether user j changes to a DCU or a non-DCU. Let V_{min} represent the minimum, and V_{max} represent the maximum allowed velocity for a particle. A particle i will have a velocity in each direction j ($V_{ij} \in [V_{min}, V_{max}]$). A coefficient ($w$) is used to slow the current velocity of the particle over time.

Each particle i will keep a record of its best personal solution (P^i), where each P^i has an entry for each user j ($P^i_j \in \{0, 1\}$). The particle i will be attracted back to P^i with a given personal weighting coefficient (p_w). This coefficient will allow the solution to explore areas of the search space close to P^i.

The system as a whole will keep a best global solution (G). This best global solution has an entry for each user j ($G_j \in \{0, 1\}$). All the particles in the system are attracted to the best global solution. The force of the attraction is determined by a global weighting coefficient (g_w). The coefficient promotes the exploration around the best known solution. The values of the coefficients w, g_w, and p_w were selected by experimentation to optimize the performance.

3.5 Tabu Search

The Tabu Search heuristic uses a "tabu list" to keep a record of the visited areas of the search space. These areas were explored using a local search procedure. The purpose of this tabu list is to provide a short term memory of explored areas [14]. To make the size of the tabu list reasonable, only the last $size_{TS}$ visited neighborhoods are saved [15].

Tabu Search combines global search and local search. The global search is done by generating random solutions. The number of SSs and which users are SSs is determined randomly, and the assignment of the remaining users to SSs or to the MS is done using the Min-Min RT algorithm.

Local moves (or short hops) explore the neighborhood of the current solution, searching for the local minimum. The short hop procedure is shown in Figure 8. All the moves that we use in the Tabu Search are considered greedy in the sense that we accept a neighboring solution if it is more robust (better objective function value); however, applying greedy moves may cause the Tabu Search to reach a local minimum that it cannot escape. The global move (or long hop) is used to escape local minima by producing a random solution with a new set of SSs that is not in the tabu list. The procedure for Tabu Search is shown in Figure 9.

(1) Initialize an array of P particles by N dimensions randomly with 0 or 1 (a value of 0 indicates a user is not a DCU and 1 indicates the user is a DCU).
(2) Evaluate the robustness of each particle using the Min-Min RT algorithm.
(3) Initialize the global and individual best positions.
(4) For ($i = 1$ to number of particles) do
  (a) For ($i = 1$ to number of dimensions) do
    (i) $R_1 = U(0, 1)$
    (ii) $R_2 = U(0, 1)$
    (iii) $R_3 = U(0, 1)$
    (iv) $V_{ij} = w \cdot V_{ij} + p_w \cdot R_1 \cdot (P^i_j - X_{ij}) + g_w \cdot R_2 \cdot (G_j - X_{ij})$
    (v) If ($V_{ij} < V_{min}$) then $V_{ij} \leftarrow V_{min}$.
    (vi) If ($V_{ij} > V_{max}$) then $V_{ij} \leftarrow V_{max}$.
    (vii) If ($R_3 < Sigmoid(V_{ij})$) then $X_{ij} = 1$, else $X_{ij} = 0$.
  (b) Evaluate the robustness (use Min-Min RT algorithm to generate the complete mapping).
  (c) Update global and personal best positions.

Figure 7: Procedure for DPSO

3.6 Genitor Robustness

The Genitor Robustness heuristic is based on the Genitor heuristic [28]. The Genitor is a steady state genetic algorithm that only does one crossover and mutation operation per iteration. The results of the crossover and mutation are evaluated and inserted in the population based on their rank. The heuristic uses the ranked population to keep the best chromosomes in the population.

This heuristic uses a chromosome that represents a full mapping. A chromosome is a vector of length N. The ith entry indicates whether user i is connected to the MS or an SS (represented by its user number). With this representation, the crossover and mutation operations can cause invalid solutions that need to be fixed.

The first operator is crossover; for the crossover we randomly select two points (from 0 to $N - 1$) in the two parent strings and exchange the entries (between these two points) to generate two new offspring. If the crossover causes a user to be mapped to another user that is no longer an SS, then the first user is assigned to the existing SS that minimizes the user's RT_x. The procedure for the crossover is shown in Figure 10.

The second operator is mutation; this operation is done to the new offspring. For the mutation, we determine with a fixed probability (determined empirically) if the assignment of a user is mutated. The mutation is done by selecting if a user should be connected directly to the MS, an SS, or assigned to user i. If as a result of the mutation the user is connected directly to the MS or is an SS then no further repairs need to be made to the assignment; however, if the user is connected to user i then i needs to be converted into an SS (if it was not one already). The procedure for mutation is shown in Figure 11. The complete procedure for the Genitor Robustness heuristic is shown in Figure 12.

4. RELATED WORK

Various MMOG architectures are reported in the literature (e.g. client/server [11], peer-to-peer [6, 17, 23], mirrored server [10]). Each architecture has its own advantages. For example, the

(1) Set $short_{hops}$ to 0 and set $MAX_{SHORT\ HOPS}$ to the maximum allowed short hops.
(2) Given the solution found in the long hop, the best known robustness (rob_{best}) is the robustness of this solution.
(3) While ($short_{hops} < MAX_{SHORT\ HOPS}$).
  (a) Given a full mapping find U_{max}.
  (b) For each server s_{move1} (DCU or MS), reconnect U_{max} to s_{move1}.
    * If the move increases the robustness then accept the move, update rob_{best}, and go to step (c).
  (c) Increase $short_{hops}$ by one.
  (d) Find the SS that has the user with RT_{max} denoted SS_{max}.
  (e) Select a random user that is connected to the SS_{max} denoted U_{random}.
  (f) For each DCU s_{move2} , reconnect U_{random} to s_{move2}.
    * If the move increases the robustness then accept the move, update rob_{best}, and go to step (g).
  (g) Increase $short_{hops}$ by one.

Figure 8: Procedure for local search

client/server and mirrored server allow the company that develops the MMOG environment to maintain tight control of the game state; however, there is a significant monetary cost associated with maintaining a large-scale MMOG environment. In a peer-to-peer architecture, because of the absence of a centralized game state controller, no peer has full control over the game state making it difficult to guarantee a consistent MMOG environment. The advantage of using a peer-to-peer architecture is that there is no single point of failure and the MMOG environment can be maintained without a significant monetary cost.

Maintaining a seamless interactive experience for the users is an important factor in MMOG, because an increase in latency within the system can lead to deterioration in the gaming experience [3, 11]. In [17], the authors show that the latency follows a "weak exponential increase" as the number of users grows in the system. Our study focuses on latency as a critical performance parameter that must be maintained and uses the results in [17] to model the relationship between latency and the number of users.

This study proposes a hybrid client/server architecture to combine the best elements of both the centralized client/server and peer-to-peer architectures, and guarantee a robustness criteria that creates a fair environment.

Our work is similar to [24] where a distributed system uses intermediate servers (analogous to our definition of secondary servers) to reduce the communication latency to the central server. The main differences between our studies and [24] is that: (a) in [24] the intermediate servers are predefined and do not participate as users in the MMOG, and (b) we have a robustness criteria created using the FePIA procedure to guarantee fairness.

Our work is different from [10, 11] because it considers converting users to secondary servers. This work is also different from [6, 17, 23] because it has a "non-peer" centralized server, and we directly address the issue of fairness.

In [9], it was shown that by employing users' computers to off-

(1) Create the tabu list of size $size_{tabu}$.
(2) While (execution time is less than the maximum allowed execution time)
  (a) Generate a random set of DCUs ($rand_{DCUs}$). If this random set is not in the tabu list then continue to step (b), otherwise repeat step (a).
  (b) Use the Min-Min RT heuristic with $rand_{DCUs}$ to generate a full mapping. If this mapping does not meet the robustness requirements then go to step (a).
  (c) Use the local search (short hop procedure).
  (d) Update the tabu list by adding the set of DCUs from step (c) and removing the oldest set of DCUs.

Figure 9: Procedure for Tabu Search

(1) Select two parents for crossover (parent 1 and parent 2) using the linear bias function.
(2) Generate two random numbers between 0 and $N-1$ (R_1 and R_2 with $R_1 < R_2$).
(3) Copy parent 1 to offspring 1 and parent 2 to offspring 2.
(4) The entries between R_1 and R_2 in offspring 1 are exchanged with the value the entries have in offspring 2.
(5) Determine the set of SSs for offspring 1.
(6) For each entry (i.e., user assignment) in offspring 1:
  (a) Check if the entry i has a valid assignment.
  (b) If entry i has an invalid assignment (e.g., user i is connected to a user j that is not connected to the MS) then assign it to the server (MS or an SS) that gives the user the minimum RT_x.
(7) Repeat steps (5) and (6) for offspring 2.

Figure 10: Procedure for crossover

load computation the RT_{max} of the system was decreased by an order of magnitude. This study is different from the research in [9] because this study evaluates the performance of the heuristics based on a robustness metric. Both this study and [9] use a very similar system model. In [9], the optimization criteria was to minimize RT_{max}. The goal of this research is to maximize the number of users allowed to be added during a game session while maintaining fairness. Subsequent to our work in [9], a related concept was presented in [18]. This work presented a hybrid server similar to the one we considered. However, because we focus on determining which SSs can give the best improvement, while [18] does not focus on this issue.

5. RESULTS

The simulation had 200 users interacting in the MMOG environment. The constants for these simulations were b = 0.03 and c = 0.01 (the values for these constants were set to approximate realistic values for latencies in an MMOG environment). The values for β_{max} and Δ_{max} are 200 and 150, respectively. The communication times between nodes were allowed to vary from 0 to 40 time units with a uniform distribution. The computational constant

(1) Set k to
(2) Based on a fixed probability, determine if the kth entry in the chromosome is mutated.
(3) If the entry is mutated, then:
  (a) Generate a random assignment (connected to the MS, SS, or connected to user i).
  (b) If the entry being modified was an SS then reassign the players assigned to this SS to existing SSs (selected randomly) and change the value of the entry to the random assignment.
  (c) Otherwise, change the value of the entry to the random assignment and if this is an assignment to a user that is not an SS convert that user to an SS.
(4) Increase k by 1.
(5) If $k \leq N$ then go to (2).

Figure 11: Procedure for the mutation

(1) An initial population of 200 chromosomes (determined empirically) is generated and evaluated.
(2) While (there are less than 1000 iterations without improvement or 10 minutes have not elapsed).
  (a) A pair of parents is selected for crossover and mutation using roulette wheel selection.
  (b) two offspring are generated using two-point crossover.
  (c) For each offspring there is a 2% probability of mutating each field in the chromosome.
  (d) The offspring are evaluated and ranked into the population replacing the worst chromosome.
(3) The output is the best solution.

Figure 12: Procedure for the Genitor

(μ_α) at each user node was allowed to vary between 0.5 and 1 with a uniform distribution. For this study, 100 scenarios were created with varying communication times and μ_α for each user. For the purpose of comparing heuristics, they were limited to a maximum execution time of 10 minutes (Min-Min SS executes in less than 1 second). Because the maximum robustness of the optimal solution can be intractable to compute, an upper bound was used to compare the performance of the results.

A parameter sweep was done on Tabu Search, DPSO, Genitor Robustness and Genitor Robustness seeded with the Min-Min SS heuristic. Figure 13 shows the best results for each heuristic found after doing parameter sweeps and the 95% confidence intervals.

The performance of the seeded Genitor Robustness and DPSO had similar performance (about 20 players could be added). The unseeded Genitor Robustness did not perform well; this could be caused by the method used for generating random solutions (solutions with a negative robustness are not screened out of the initial population).

For the Tabu Search, the average result from a long hop was a robustness of 9.06 users (a total of 3458 long hops were executed). The average improvement obtained by the local search was 24.45% upon the initial solution with an average of 24.5 short hops. This shows that the short hops are able to improve the solution by exploring the neighborhood.

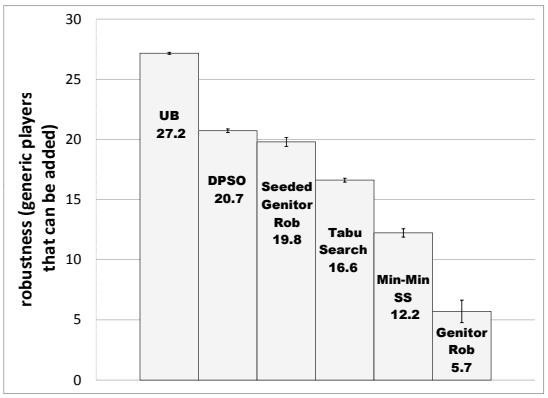

Figure 13: Simulation results; UB (upper bound) based on [9]

The performance of the Genitor Robustness heuristic was improved significantly with the introduction of the Min-Min SS seed. The performance of the Min-Min SS heuristic was 45% of the UB, however it was 60% of the best value obtained among the studied heuristics. The improvement of the Genitor over the Min-Min SS was on average 7.559 time units (a 61.8% improvement).

The results of the heuristics were (on average) more than 6.44 time units less than the UB (about 76% of the UB). It is possible that the simplifying assumptions used to calculate the bounds could be making it very loose.

6. CONCLUSIONS

This study evaluated an oversubscribed MMOG environment that employs a group of users to do portions of the required game-state calculations. The main objective of this study was to develop heuristics to create a fair environment in a secondary server based MMOG environment. The allocation of users' computers as SSs allows a reduction in the RT_{max} time [9], and increases the number of users that can join the game after it starts while maintaining a QoS constraint. This QoS constraint avoids the users from feeling they are at an unfair disadvantage during their interaction with the MMOG environment. The results from the heuristics show that with the constraints set for this environment, a large number of users can be added while maintaining the fairness conditions (approximately 10% more users).

A possible extension of this study is to improve the model by removing simplifying assumptions (e.g., the constant communication times). This study also assumed that users are willing to become an SS. This problem could also be reformulated using game theory to consider the behavior of selfish and/or cooperative users. This problem assumes a fully connected network, however, any network configuration can be user. To model another network configuration, stochastic communication values (represented by a probability mass function or probability density function) could be used.

7. ACKNOWLEDGMENTS

The authors would like to thank Richard Wallace, Paul Maxwell, Jay Smith, Vladimir Shestak, Jerry Potter, Ricky Kwok, and Samee U. Khan for their valuable contributions.

8. REFERENCES

[1] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao. Characterizing resource allocation heuristics for heterogeneous computing

systems. In *Advances in Computers Volume 63: Parallel, Distributed, and Pervasive Computing*, pages 91–128, 2005.

[2] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim. Measuring the robustness of a resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):630–641, July 2004.

[3] G. J. Armitage. An experimental estimation of latency sensitivity in multiplayer Quake 3. In *11th IEEE International Conference on Networks (ICON '03)*, pages 137–141, Sept. 2003.

[4] L. Barbulescu, L. D. Whitley, and A. E. Howe. Leap before you look: An effective strategy in an oversubscribed scheduling problem. In *19th National Conference on Artificial Intelligence*, pages 143–148, July 2004.

[5] N. E. Baughman and B. N. Levine. Cheat-proof playout for centralized and distributed online games. In *IEEE Conference on Computer Communications (INFOCOM '01)*, pages 104–113, Mar. 2001.

[6] A. Bharambe, J. Pang, and S. Seshan. Colyseus: A distributed architecture for online multiplayer games. In *3rd Symposium on Networked Systems Design and Implementation*, pages 155–168, 2006.

[7] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, June 2001.

[8] L. D. Briceño, M. Oltikar, H. J. Siegel, and A. A. Maciejewski. Study of an iterative technique to minimize completion times on non-makespan machines. In *International Heterogeneity in Computing Workshop (HCW '07)*, page 138, Mar. 2007.

[9] L. D. Briceño, H. J. Siegel, A. A. Maciejewski, Y. Hong, B. Lock, M. N. Teli, F. Wedyan, C. Panaccione, and C. Zhang. Resource allocation in a client/server hybrid network for virtual world environments. In *International Heterogeneity in Computing Workshop (HCW '08)*, 2008.

[10] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. *Multimedia Tools Applications*, 23(1):7–30, 2004.

[11] G. Deen, M. Hammer, J. Bethencourt, I. Eiron, J. Thomas, and J. H. Kaufman. Running Quake II on a grid. *IBM Systems Journal*, 45(1):21–44, 2006.

[12] Q. Ding and G. Chen. A benefit function mapping heuristic for a class of meta-tasks in grid environments. In *1st International Symposium on Cluster Computing and the Grid (CCGRID '01)*, page 654, May 2001.

[13] S. Ghanbari and M. R. Meybodi. On-line mapping algorithms in highly heterogeneous computational grids: A learning automata approach. In *International Conference on Information and Knowledge Technology (IKT '05)*, May 2005.

[14] F. Glover. Tabu search — Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[15] A. Hertz and D. de Werra. The tabu search metaheurstic: How we used it. *Annals of Mathematics and Artificial Intelligence*, 1(1–4):111–121, Sept. 1990.

[16] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on non-identical processors. *Journal of the ACM*, 24(2):280–289, Apr. 1977.

[17] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In *3rd ACM SIGCOMM workshop on Network and System Support for Games*, pages 116–120, Aug. 2004.

[18] J. Jardine and D. Zappala. A hybrid architecture for massively multiplayer online games. In *Seventh Workshop on Network and Systems Support for Games (NetGames '08)*, 2008.

[19] Z. Jinquan, N. Lina, and J. Changjun. A heuristic scheduling strategy for independent tasks on grid. In *Eighth International Conference on High-Performance Computing in Asia-Pacific Region '05*, page 6, Nov. 2005.

[20] K. Kaya, B. Ucar, and C. Aykanat. Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories. *Journal of Parallel and Distributed Computing*, 67(3):271–285, Mar. 2007.

[21] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, Nov. 1995.

[22] J.-K. Kim, H. J. Siegel, A. A. Maciejewski, and R. Eigenmann. Dynamic mapping in energy constrained heterogeneous computing systems. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS '05)*, Apr. 2005.

[23] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *IEEE Conference on Computer Communications (INFOCOM '04)*, pages 96–107, Mar. 2004.

[24] K.-W. Lee, B.-J. Ko, and S. Calo. Adaptive server selection for large scale interactive online games. *Computer Networks*, 49(1):84–102, Sept. 2005.

[25] J. Pugh and A. Martinoli. Discrete multi-valued particle swarm optimization. In *IEEE Swarm Intelligence Symposium '06*, pages 103–110, May 2006.

[26] A. Shaikh, S. Sahu, M.-C. Rosu, M. Shea, and D. Saha. On demand platform for online games. *IBM Systems Journal*, 45(1):7–20, 2006.

[27] V. Shestak, E. K. P. Chong, H. J. Siegel, A. A. Maciejewski, L. Benmohamed, I.-J. Wang, and R. Daley. A hybrid branch-and-bound and evolutionary approach for allocating strings of applications to heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 28(3):1157–1173, 2008.

[28] D. Whitley. The GENITOR algorithm and selective pressure: Why rank based allocation of reproductive trials is best. In *3rd International Conference on Genetic Algorithms*, pages 116–121, June 1989.

[29] M. Wu and W. Shu. Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In *9th IEEE Heterogeneous Computing Workshop (HCW '00)*, pages 375–385, Mar. 2000.