Resource Allocation in a Client/Server System for Massive Multi-Player Online Games

Luis Diego Briceño¹, Howard Jay Siegel¹,², Anthony A. Maciejewski¹, Ye Hong¹, Brad Lock¹, Charles Panaccione², Fadi Wedyan², Mohammad Nayeem Teli², and Chen Zhang²

Abstract

The creation of a Massive Multi-Player On-line Game (MMOG) has significant costs, such as maintenance of server rooms, server administration, and customer service. The capacity of servers in a client/server MMOG is hard to scale and cannot adjust quickly to peaks in demand while maintaining the required response time. To handle these peaks in demand, we propose to employ users’ computers as secondary servers. The introduction of users’ computers as secondary servers allows the performance of the MMOG to support an increase in users. Here, we consider two cases. First, for the minimization of the response times from the server, we develop and implement five static heuristics to implement a secondary server scheme that reduces the time taken to compute the state of the MMOG. Second, for our study on fairness, the goal of the heuristics is to provide a “fair” environment for all the users (in terms of similar response times), and to be “robust” against the uncertainty of the number of new players that may join a given system configuration. The number of heterogeneous secondary servers, conversion of a player to a secondary server, and assignment of players to secondary servers are determined by the heuristics implemented in this study.

I. INTRODUCTION

The environment considered in this research is a massive multiplayer online gaming (MMOG) environment. In an MMOG environment, each user (also referred to in this paper as a player) controls an avatar (an image that represents and is manipulated by a user) in a virtual world and interacts with other users. The experience (positive or negative) the user has with the MMOG environment is dependent on how quickly the game world responds to the user’s actions, and the “fairness” of the game world.

There are various criteria that can be used to quantify the responsiveness of the game world to a user’s action, and the “fairness” of the system. In this study, we consider two. The first performance metric is the maximum response time (time needed to send an action to the server and receive the result of that action from the server). When the performance metric is response
time, the goal of the resource allocation is to minimize the maximum response time among all users. The second performance metric is “fairness.” When the performance metric is “fairness,” the goal of the resource allocation is to maximize the number of new users that can join while obeying a constraint on the maximum response time and the differences between the response times allowed so that the system is both “fair” and responsive.

In general, most MMOG environments use a client/server architecture to control the virtual game world. This architecture 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 [1]. Also, the architecture of MMOGs is non-standard [2].

The environment we are interested in is massive on-line first-person shooters. These types of games usually involve a large group of people (e.g., 256 people in Massive Action Game (MAG) developed by Zipper Interactive [3]) competing in a virtual world attempting to complete specific goals. 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 this can be difficult. It is possible for the game to process these users’ actions in the incorrect order.

This study focuses on simulating an MMOG where secondary servers (SSs) can be used to modify the system based on demand. Consider an environment where there is a main server (MS) that controls the state of the virtual world, and each user (N is the total number of users) produces a data packet that needs to be processed by the MS. If the performance falls below acceptable standards, the MS can off-load calculations to SSs. An SS is a user’s computer that is converted into a server to avoid degradation in the performance of the MMOG environment (see Fig. 1). The purpose of using the users’ computers as SSs is to create a distributed ad-hoc system that will keep the response times low and fair.

In our simulation environment, we derived a model for the computation of the MS and SS. We modeled the users as being in a fully connected network. The model considers alignment of computation on the SSs and MS, the communication times between users as well as delay times (time a user action has to wait at the SS before being processed). We created a mathematical model based on real world-data to study the potential advantages of using a hybrid client/server architecture in an MMOG. Therefore, the results should be representative of how the studied resource allocation heuristics would perform in a real system.
Fig. 1. (a) Client/server architecture, using a single server to do processing; and (b) secondary server architecture, using users’ computers to assist the Main Server in processing.

The allocation of users as SSs has similar security requirements as distributed servers and peer-to-peer based MMOG systems. These issues are studied in [4]–[6] and will not be discussed here because we consider it to be a separate research problem.

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 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 [7]. These assumptions make a static resource allocation heuristic viable [8].

This study addresses two problems in the operation of the MMOG environment with different optimization criteria. The first problem is the minimization of the maximum response time of any user, while the second is the maximization of the number of players that can be added while maintaining fairness. For the first, we develop resource allocation heuristics that decide which users to make secondary servers and assign users to these secondary servers or the main server. For the second, we develop resource allocation heuristics that again decide which users to make secondary servers and assign users to these secondary servers or the main server, but in this case with the goal of allowing the greatest number of new users to be added without violating constraints on the minimum and maximum allowed response time for any user (fairness).

To solve these problems, we derive mathematical models to capture the dynamic MMOG en-
vironment, and designed heuristics that determine 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., [9]–[13], [15], [16]) with the SSs and the MS as machines and the users as tasks.

The contributions of this paper are: (a) mathematically modeling an MMOG environment, (b) designing heuristics to minimize the response time, (c) studying and simulating an MMOG environment where an unpredictable number of players may want to join an ongoing session, (d) 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 (e) 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 II provides the common environment. In Section III, we analyze the MMOG environment when the optimization criterion is to minimize the maximum response time. This section includes the heuristics used to do resource allocation, a bound on performance, and the results for this simulation setup. Section IV focuses on the proposed heuristics for maximizing the robustness for fairness, an upper bound on the number of new players that can join the game, and results for this section. We provide the related work in Section V, and in Section VI we present our conclusions.

II. ENVIRONMENT

A. Overview

For both problem domains considered in this study, the elements of the MMOG environment that we can control are (a) which users are converted into secondary servers, and (b) which users are assigned to the MS or to an SS. In the client/server solution shown in Fig. 1(a), all users connect to the MS, therefore the MS is the only machine performing computation. In the SS solution shown in Fig. 1(b), the MS and SSs perform computation and the MS resolves conflicts among the users and SSs connected to it.

The time it takes the system to respond to a user’s request (latency) is very important [17]. The communication time between different pairs of nodes (user computer, SS, or MS) will vary. To simplify the calculation of a server’s response time to a user, the following assumptions are made about the communication model in this system. 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 \textit{SS} or the \textit{MS}. These assumptions are used to reduce the complexity of the simulations. A glossary of terms used is in Fig. 2.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_\alpha )</td>
<td>number of users connected to ( SS_\alpha )</td>
</tr>
<tr>
<td>( \mu_\alpha )</td>
<td>computational constant of ( SS_\alpha )</td>
</tr>
<tr>
<td>( \text{Comp}_{\alpha} )</td>
<td>computation time for ( SS_\alpha )</td>
</tr>
<tr>
<td>( n_{\text{secondary}} )</td>
<td>total number of users connected to all the ( SS )s</td>
</tr>
<tr>
<td>( n_{\text{main}} )</td>
<td>the number of users connected directly to the ( MS )</td>
</tr>
<tr>
<td>( n_{\text{nss}} )</td>
<td>number of ( SS )s</td>
</tr>
<tr>
<td>( \text{Comp}_{\text{MS}} )</td>
<td>computation time for ( MS )</td>
</tr>
<tr>
<td>( b ) and ( c )</td>
<td>computational constants for ( \text{Comp}_{\text{MS}} )</td>
</tr>
<tr>
<td>( U_x )</td>
<td>user ( x )</td>
</tr>
<tr>
<td>( RT_x )</td>
<td>response time for ( U_x )</td>
</tr>
<tr>
<td>( \text{Comm}(A,B) )</td>
<td>communication time between node A and node B</td>
</tr>
<tr>
<td>( \Delta_{\text{MS}} ) and ( \Delta_{SS} )</td>
<td>time a packet has to wait at the ( MS ) or ( SS ) respectively</td>
</tr>
<tr>
<td>( RT_{\text{new}} )</td>
<td>maximum ( \text{RT} ) of the system with ( n_{\text{new}} ) players</td>
</tr>
<tr>
<td>( \text{Comp}_{\text{MS}}' )</td>
<td>the computation at the ( MS ) with ( n_{\text{new}} ) players added</td>
</tr>
</tbody>
</table>

Fig. 2. Glossary of terms.

B. Computational Model for Main Server and Secondary Servers

To simplify the study, the level of activity in the MMOG environment of the users 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 \( SS \)s, we consider how the computation time increases with an increase in the number of users. In [18], 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 \( n_\alpha \) be the number of users connected to secondary server \( \alpha (SS_\alpha) \), and \( \mu_\alpha \) be a computational constant for \( SS_\alpha \) that represents the computing power heterogeneity across different users’ computers (each user has a different constant). The computation time for an \( SS_\alpha (\text{Comp}_\alpha) \) can be modeled as:

\[
\text{Comp}_\alpha = \mu_\alpha \cdot (n_\alpha)^2.
\]
Let \( n_{\text{secondary}} \) be the total number of users connected to all the \( SSs \), \( n_{\text{nss}} \) be the number of \( SSs \), \( n_{\text{main}} \) be the number of users connected to the \( MS \), and \( b \) and \( c \) be computational constants of the \( MS \), \( 0 \leq b, c \leq 1 \). The computation time of the \( MS (\text{Comp}_{\text{MS}}) \) is:

\[
\text{Comp}_{\text{MS}} = c \cdot n_{\text{secondary}} + b \cdot (n_{\text{main}} + n_{\text{nss}})^2.
\] (2)

We assume the game world state is updated and synchronized every \( \text{Comp}_{\text{MS}} \) time units.

C. Objective Functions \( RT_{\text{max}} \) and \( RT_{\text{min}} \)

Let \( RT_x \) represent the Response Time (RT) of a packet (representing an action in the game world) sent by the computer of user \( x (U_x) \) to the \( MS \) (possibly through an \( SS \)) and returning to \( U_x \) with the corresponding consequence of that action in the game world. Let \( \text{Comm}(A,B) \) be the communication time between node \( A \) and node \( B \).

Let \( \Delta_{\text{MS}} \) be the time a packet has to wait before being processed. Consider the case where \( U_x \) is connected directly to the \( MS \) (Fig. 3). To calculate \( RT_x \) for this case:

\[
RT_x = \text{Comm}(U_x, MS) + \Delta_{\text{MS}} + \text{Comp}_{\text{MS}} + \text{Comm}(MS, U_x).
\] (3)

Consider the case where \( U_x \) is connected to \( SS_\alpha \) as illustrated in Fig. 1(b). The \( SS_\alpha \) knows its computation time (based on number of users assigned to it), and its communication time to the \( MS \). We assume that \( SS_\alpha \) will send out an update immediately before the game world is synchronized across users. For example in Fig. 3, \( SS_\alpha \) begins \( \text{Comp}_\alpha \) so that \( \text{Comp}_\alpha \) and \( \text{Comm}(SS_\alpha, MS) \) completes at time \( t(k+1) \). We also assume that \( \text{Comp}_\alpha + \text{Comm}(SS_\alpha, MS) \leq \text{Comp}_{\text{MS}} \). Let \( \Delta_{\text{SS}} \) be the wait time between a \( U_x \) action and when \( SS_\alpha \)’s computation starts. If a user is connected to an \( SS_\alpha \) then the equation is:

\[
RT_x = \text{Comm}(U_x, SS_\alpha) + \Delta_{\text{SS}} + \text{Comp}_\alpha + \text{Comm}(SS_\alpha, MS) + \text{Comp}_{\text{MS}}
\]

\[
+ \text{Comm}(MS, SS_\alpha) + \text{Comm}(SS_\alpha, U_x).
\] (4)

A graphical representation of this equation is shown in Fig. 4. If \( U_x \) is \( SS_\alpha \), then Eq. 4 is used with \( \text{Comm}(U_x, SS_\alpha) = \text{Comm}(SS_\alpha, U_x) = 0 \). In the greedy resource allocation heuristics, the \( RT \) calculation uses partial information about the current state of the mapping. Each time a user is added, the values of \( \text{Comp}_{\text{MS}} \) and \( \text{Comp}_\alpha \) used to calculate \( RT_x \) are updated. Due to the heterogeneous communication times, it is possible that for a user it is better to communicate indirectly to the \( MS \) through an \( SS \).
If a user’s action misses the start of the computation at $SS_\alpha$ then the maximum time it will be waiting for computation is $\Delta_{MS}$ or $\Delta_{SS}$ equal to $Comp_{MS}$. Thus, at the next mapping time interval, to calculate the maximum response time (i.e., $RT_{max}$) we use:

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

with $\Delta_{MS} = Comp_{MS}$ or $\Delta_{SS} = Comp_{MS}$. Fig. 5 shows $RT_{max}$ for the case when $U_x$ is connected to $SS_\alpha$, and $\Delta_{SS} = Comp_{MS}$, and this time represents the maximum time any user will have to wait for a response from the $MS$ for this case.

Let $RT_{min}$ represent the fastest any user can interact with the $MMOG$ environment, then:

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

with $\Delta_{MS} = 0$ or $\Delta_{SS} = 0$. The case when $U_x$ is connected to $SS_\alpha$, and $\Delta_{SS} = 0$, is in Fig. 6.

### III. RESPONSE TIME MINIMIZATION

#### A. Problem Statement

To deal with the situation when the $MS$ is oversubscribed, we consider in this section the conversion of users to $SS$s that assist the $MS$ in computation. In the client/server solution shown in Fig. 1(a), all users connect to the $MS$, therefore the $MS$ is the only machine performing computation. In the $SS$ solution shown in Fig. 1(b), the $MS$ and $SS$s perform computation and the $MS$ resolves conflicts among users and $SS$s connected to it. The objective, of this section, is to minimize $RT_{max}$ (i.e., the resource allocation that produces the smallest $RT_{max}$ is the best).
Fig. 5. $RT_{\text{max}}$ when the user just misses the deadline for sending computation to the SS$_a$, i.e., $\Delta_{SS} = \text{Comp}_{\text{MS}}$.

Fig. 6. $RT_{\text{min}}$ when the user with just makes the deadline for sending computation at the SS, i.e., $\Delta = 0$.

In Subsection III-B, we describe six heuristics for minimizing $RT_{\text{max}}$ by converting selected users to SSs. A lower bound on $RT_{\text{max}}$ is derived in Subsection III-C. Our evaluation of the heuristics through simulations is given in subsection III-D.

B. Heuristics for Response Time Minimization

1) Heuristic Requirements: All heuristics were limited to a maximum execution time of ten minutes on a single computer (i.e., one core) with no optimization. This execution time of the heuristics could be reduced to less than five minutes after optimizing the code to run on multiple cores. We assume that the players wait in a game lobby while the game fills up, and that five minutes before the game starts no users will be allowed to enter the game. In this context, resource allocation implies assigning a user in one of three ways: (1) attaching it directly to the MS without making it an SS (although it can become one), (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. If a user is connected to a DCU, that DCU is also an SS; any SS is a DCU, but a DCU is not necessarily an SS. Because of the equations for calculating MS and SS computations and
the heterogeneity for communication between pairs of devices, $RT$ might be smaller through a $DCU$ instead of the $MS$.

For this study, we used heuristics from three categories. In our study, the greedy search heuristics find a mapping based on a greedy heuristic. The global search heuristics are those that search the entire search space, and the “directed” global search heuristics are those that use a greedy heuristic to find a local minima based on a set of secondary servers.

2) Min-Min $RT$: The Min-Min heuristic concept [19] is a greedy heuristic that is widely used in the area of resource allocation (e.g., [19]–[22]), and has been shown in the literature to be very effective in various environments (e.g., Max-Max Robust [23]). This heuristic selects the best pairing among unmapped users and servers ($SS$ or $MS$). The Min-Min $RT$ subroutine is used by the other heuristics to complete a resource allocation. Min-Min $RT$ requires the “main heuristic” to determine which users will be connected directly to the $MS$ (i.e., the $DCUs$). The Min-Min $RT$ then determines how to connect unassigned users to $DCUs$ (which then become $SS$s). The procedure to implement the Min-Min $RT$ is shown in Fig. 7.

(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, find the $DCU$ that gives the minimum $RT$ (first minimum).
(3) The best paired user/server (i.e., with smallest $RT$) among all the pairs generated in (2) is selected (second minimum).
(4) The user in the best pair selected in (3) is then assigned to its paired server, and marked as assigned.
(5) Steps (2) through (4) are repeated until all users are assigned.

Fig. 7. Procedure for using Min-Min $RT$ to generate a resource allocation.

(1) All users that are marked as unassigned.
(2) For each unassigned user, find the $DCU$ or $MS$ that gives the minimum $RT$ (first minimum).
(3) The best paired user/server (i.e., with smallest $RT$) among all the pairs generated in (2) is selected (second minimum).
(4) The user in the best pair selected in (3) is then assigned to its paired server, and marked as assigned. Users that are connected directly to the $MS$ are marked as $DCUs$.
(5) Steps (2) through (4) are repeated until all users are assigned.

Fig. 8. Procedure for using Min-Min $SS$ to generate a resource allocation.

(1) Mark all users as unassigned.
(2) For each unassigned user ($u$) in a fixed arbitrary order.
   (a) Define $minRT_u$ as the $RT$ if $u$ is connected directly to the $MS$.
   (b) Among all $PH$s find the $PH$ that minimizes $RT$ of $u$ connected indirectly to the $MS$ through $PH$ ($RT_{u\rightarrow PH\rightarrow MS}$).
      (i) If $RT_{u\rightarrow PH\rightarrow MS}$ is less than $minRT_u$ then
         (α) attach $u$ to $PH$.
         (β) convert $PH$ to an $SS$ if it is not already one.
         (γ) if $PH$ was an unassigned user, assign it to the $MS$.
      (ii) Else, attach $u$ directly to the $MS$.
   (c) Mark user $u$ as assigned.
(3) Output final resource allocation.

Fig. 9. Procedure for Stage 1 of the Iterative Minimization heuristic.
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 DCUs. The heuristic will determine the set of SSs by allowing users to consider connecting to the MS in addition to DCUs; i.e., a user can be assigned to the MS or a DCU established by a earlier iteration of the heuristic. The procedure for Min-Min SS is shown in Fig. 8. Recall from Subsection III-B1, the minimum $RT$ may be through an SS.

\begin{itemize}
  \item[(1)] $RT_{best}$ is equal to the $RT_{max}$ value of the resource allocation generated by Stage 1.
  \item[(2)] While total moves is less than $MOVE_{LIMIT}$ (empirically set to 300) or no improvement is achieved
    \begin{itemize}
      \item[(a)] For each user ($U_x$) connected to an SS
        \begin{itemize}
          \item[(α)] Connect $U_x$ to the MS and connect the user with $RT_{max}$ to $U_x$.
          \item[(β)] Find the $RT_{max}$ of this configuration.
          \item[(γ)] If $RT_{max} < RT_{best}$ then save this resource allocation as the best, otherwise undo the change.
        \end{itemize}
      \item[(b)] For each user ($U_x$) connected to an SS
        \begin{itemize}
          \item[(α)] Swap $U_x$ with the user with $RT_{max}$.
          \item[(β)] Find the $RT_{max}$ of this configuration.
          \item[(γ)] If $RT_{max} < RT_{best}$ then save this resource allocation as the best, otherwise undo the swap.
        \end{itemize}
      \item[(c)] For each SS ($SS_x$)
        \begin{itemize}
          \item[(α)] Connect the user with $RT_{max}$ to $SS_x$.
          \item[(β)] Find the $RT_{max}$ of this configuration.
          \item[(γ)] If $RT_{max} < RT_{best}$ then save this resource allocation as the best, otherwise undo the change.
        \end{itemize}
    \end{itemize}
  \item[(3)] Output best resource allocation.
\end{itemize}

Fig. 10. Procedure for Stage 2 of the Iterative Minimization heuristic.

4) Iterative Minimization: The Iterative Minimization (IM) is a greedy search heuristic that we developed for this environment. A potential host ($PH$) is a user that is either a DCU or is unassigned at this point. The intuition behind the IM heuristic is to find a good solution using a greedy heuristic, and then refine it with an iterative minimization procedure. This heuristic considers connecting an unassigned user to all $PH$s or the MS and picks the $PH$ or $MS$ that provides the minimum $RT_x$. If a $PH$ provides the minimum $RT_x$ and is not already an SS then it is converted to one. If the $PH$ selected was an unassigned user, when it is made to be an SS, it is assigned to be directly connected to the MS. The procedure is in Fig. 9.

In Stage 2, an iterative minimization procedure considers moving the user with $RT_{max}$ to a different secondary server or to the MS to find a better resource allocation. Three different local search operator are attempted at each “iteration.” The goal is that after this stage the solution has reached a local optima. By re-assigning the user with the current $RT_{max}$, we hope to reduce the $RT_{max}$ of the system. The procedure for this iterative minimization is shown in Fig. 10.

5) Tabu Search: The Tabu Search concept [24] is a directed global search heuristic that stores the previously visited areas in the search space using a tabu list so they are not revisited. Local moves (or short hops) explore the neighborhood of the current resource allocation, searching
for the local minimum. The short hops try to find better resource allocations within the same neighborhood (same set of DCUs) by moving the user with $RT_{max}$ to other SSs, or by reducing the computation of the SS where the user with $RT_{max}$ is connected. All the local moves that we use in the Tabu Search are considered greedy in the sense that we accept a resource allocation if it has a smaller $RT_{max}$ (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 resource allocation with a new set of SSs that is at least 50% different than previous solutions in the tabu list. The size of the tabu list was not limited in this study (on average our simulation had less than 4000 successful long hops). The sum of both long hops and short hops was limited to a ten minute execution time. The number of short hops allowed per long hop was limited to a maximum $\text{max}_{\text{shorthops}}$ hops (determined empirically to be 100), or ten short hops without improvement. The procedure for Tabu Search is shown in Fig. 11 and the procedure for the short hops is shown in Fig. 12.

![Tabu Search Procedure](image)

In our experiments, the Tabu Search was seeded with the results from other heuristics. This was accomplished by replacing the first long hop Fig. 11 with the solution from the seed heuristic.

6) **Discrete Particle Swarm Optimization:** Discrete Particle Swarm Optimization (DPSO) is a “directed” global search heuristic based on the particle swarm optimization concept in [25]. The authors in [26] implemented a discrete version of the particle swarm optimization in [25]. We built upon this discrete version to design a DSPO for our problem domain. Intuitively, this algorithm samples the search space of possible SS configurations, and then uses the Min-Min RT algorithm to generate a complete mapping from a set of SSs.

We only generated sets of SSs because it allows the DPSO to quickly find a very good combination of SSs. The reason for this implementation is that if we have a good set of SSs the greedy heuristics can generate good solutions for the remaining users. This methodology, however, comes with the disadvantage that we are only searching the search space of potential SSs and not the search space of complete mappings.
In DPSO, the position of a particle represents a solution (resource allocation). 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 \((X_{ij} = 1)\), or a non-DCU \((X_{ij} = 0)\) 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 \( V_{\text{min}} \) represent the minimum and \( V_{\text{max}} \) represent the maximum allowed velocity for a particle. A particle \( i \) will have a velocity in each direction \( j \) \((V_{ij} \in [V_{\text{min}}, V_{\text{max}}])\). A sigmoid function is used to probabilistically convert the real value of \( V_{ij} \) into a position of either 0 or 1 for \( X_{ij} \). A coefficient \((w \leq 1)\) is used to slow the current velocity of the particle over time. A particle is allowed to “move” for a pre-determined number of iterations \((\text{iter}_{\text{max}})\), determined based on the maximum allowed execution time.

1. Set \( \text{short hops} \) to 0 and set \( \text{max short hops} \) to the maximum allowed short hops.
2. Given the resource allocation found in the long hop, we determine \( RT_{\text{max}} \) of this resource allocation.
3. While \( \text{short hops} < \text{MAX SHORT HOPS} \)
   
   (a) Find the user with \( RT_{\text{max}}(U_{\text{max}}) \).
   (b) For each server \( s \) (first DCUs then attempt MS), do
      (i) if moving \( U_{\text{max}} \) to connect to \( s \) decreases \( RT_{\text{max}} \) then
         (\( \alpha \)) implement the move,
         (\( \beta \)) update \( RT_{\text{max}} \),
         (\( \gamma \)) break out of loop and go to step (c).
   (c) Increase \( \text{short hops} \) by one.
   (d) Find the SS that has the user with \( RT_{\text{max}} \).
   (e) Select a random user that is connected to this SS \((U_{\text{random}})\).
   (f) For each DCU \( s \) (in a fixed arbitrary order), connect \( U_{\text{random}} \) to \( s \), and if the move decreases \( RT_{\text{max}} \) then
      (i) implement the move,
      (ii) update \( RT_{\text{max}} \),
      (iii) break out of loop and go to step (g).
   (g) Increase \( \text{short hops} \) by 1.

Fig. 12. Procedure for using the short hops to improve a resource allocation.

Each particle \( i \) will keep a record of its best solution over time \((P^i)\), where each \( P^i \) has an entry for each user \( j \) \((P^i_j \in \{0,1\})\), where \( P^i_j = 0 \) means user \( i \) is not a DCU, and \( P^i_j = 1 \) means user \( i \) is a DCU. The particle \( i \) will be attracted back to \( P^i \) with a given personal weighting coefficient \((\text{weight}_p, \text{for all } i)\). This coefficient will attract this particle to explore areas of the search space close to \( P^i \).

The system as a whole will keep a best global solution \((G)\), when \( G_j = 0 \) means user \( i \) is not a DCU, and \( G_j = 1 \) means user \( i \) is a DCU. 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 (of all particles to \( G)\) is determined by a global weighting coefficient \((\text{weight}_g)\). The coefficient promotes the exploration around the best known solution.
The values of the coefficients $w$, $weight_g$, and $weight_p$ were selected by experimentation to optimize the performance of this heuristic. Our procedure for DPSO is in Fig. 13.

The DPSO heuristic was seeded with the results from other heuristics. This was accomplished by creating a particle that has the same set of $SS$s as the resource allocation generated by the seed heuristic; Only the set of $SS$s is taken from the seeds not all user assignments.

(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) Apply the Min-Min RT heuristic to each particle to generate a complete mapping.
(3) Determine $RT_{max}$ using the Min-Min RT heuristic for the mapping found for each particle.
(4) Initialize the global and each particles best positions.
(5) For $i = 1$ to number of particles do
   (a) For $j = 1$ to number of dimensions do
      (i) $R_1 = U(0,1); R_2 = U(0,1); R_3 = U(0,1)$
      (ii) $V_{ij} = w \cdot V_{ij} + weight_p \cdot R_1 \cdot (P_j^i - X_{ij}) + weight_g \cdot R_2 \cdot (G_j - X_{ij})$
      (iii) If ($V_{ij} < V_{min}$) then $V_{ij} \leftarrow V_{min}$.
      (iv) If ($V_{ij} > V_{max}$) then $V_{ij} \leftarrow V_{max}$.
      (v) If ($R_3 < Sigmoid(V_{ij})$) then $X_{ij} = 1$, else $X_{ij} = 0$.
   (b) Apply the Min-Min RT heuristic to each particle to generate a complete mapping.
   (c) Determine $RT_{max}$ using the Min-Min RT heuristic for the mapping found for each particle.
(6) Set each particle’s best position ($P^i$) using the particles in (4), and set the best global position to the best $P^i$ over all $i$.
(7) Repeat (4) and (5) until the number of iterations is equal to $iter_{max}$.

Fig. 13. Procedure for using the DPSO heuristic to generate a resource allocation.

7) Genitor RT: The Genitor RT heuristic is based on the Genitor heuristic [27], which is a type of Genetic Algorithm. Genetic algorithms are global search heuristics that have proven to be useful in HC environments (e.g., [28]). Genitor is a steady state genetic algorithm that only does one crossover and mutation operation per iteration. In our version of Genitor, the fitness function we use for our chromosomes is $RT_{max}$, therefore a solution with a small $RT_{max}$ has a better rank than one with a higher $RT_{max}$. We chose this fitness function because it represents the metric we want to minimize, and the computation time to need to compute $RT_{max}$ is not that significant. The results of the crossover and mutation are evaluated and inserted in the ordered population based on their rank. The heuristic uses the ranked population to keep the best chromosomes in the population (of size 200 determined empirically).

This heuristic uses a chromosome that represents a full mapping. A chromosome is a vector of size $N$ where the $i^{th}$ entry is where the $i^{th}$ user is connected. The value $j$ in this entry indicates that user $i$ is connected to the $MS$ through user $j$, if $1 \leq j \leq N$, and directly to the $MS$, if $j = 0$. While our chromosome representation is intuitive, it has disadvantage that the crossover and mutation operations can cause invalid resource allocations that need to be fixed as described below. For example, a mutation on an $SS_i$ can cause $SS_i$ to be connected to another $SS$ (e.g., $SS_j$). Another example of an invalid mapping is when a user is connected to another user that is not an $SS$. 
1) An initial population of 200 chromosomes is generated randomly and evaluated.
2) While there are less than 1000 iterations without improvement or ten minutes have not elapsed.
   a) A pair of parents is selected using linear bias [27].
   b) Two offspring are generated using two-point crossover (see Fig. 15).
   c) For each offspring there is a 0.2% probability of mutating each field in the chromosome.
   d) The offspring are evaluated and inserted into the ordered population displacing the worst chromosomes.
3) The output is the best resource allocation.

Fig. 14. Procedure for using the Genitor RT to generate a resource allocation.

The Genitor RT procedure is shown in Fig. 14. The first operator is crossover. For the crossover, we randomly select two points (from 1 to \( N \)) in the two parent strings and exchange the entries of the parents between these two points. If the crossover causes a user \( x \) to be mapped to another user that is no longer an \( SS \), then user \( x \) is assigned to an existing \( SS \) that gives it the smallest \( RT_x \) time. The procedure for the crossover is shown in Fig. 15.

(1) Select two parents for crossover (parent 1 and parent 2) using a linear bias function.
(2) Generate two random numbers between 1 and \( N \) (\( R_1 < R_2 \))
(3) The entries between \( R_1 \) and \( R_2 \) in parent 1 are exchanged with the value the entries have in parent 2 generating a child.
(4) For each entry (i.e., user assignment) in the child:
   a) Check if the entry has a valid assignment.
   b) If the entry has an invalid assignment (e.g., assigned to a user that is not an \( SS \)) then assign it to the server (\( MS \) or an \( SS \)) that gives the user the minimum \( RT_x \).
(5) The entries between \( R_1 \) and \( R_2 \) in parent 2 are exchanged with the value the entries have in parent 1 generating a child, and repeat step (4).

Fig. 15. Procedure for using crossover to generate new resource allocations.

The second operator is mutation. For the mutation, with a fixed probability (determined empirically), the assignment of a user is potentially mutated. A user \( x \) (\( u_x \)) chosen for mutation is randomly assigned to the \( MS \), an \( SS \), or user \( i \) (\( u_i \))—that is not an \( SS \). The procedure for mutation is shown in Fig. 16. If \( u_x \) was an \( SS \) and is assigned to the \( MS \) then we do not have to fix the resource allocation (\( u_x \) was already assigned to the \( MS \)). If we assign \( u_x \) to an existing \( SS \), to have a valid resource allocation, we need to reassign the players connected to \( u_x \) to other existing \( SS \)s (selected randomly). Likewise, if we assign \( u_x \) to \( u_i \) then we need to make \( u_i \) an \( SS \), and reassign the players connected to \( u_x \) to other existing \( SS \)s.

If \( u_x \) was not an \( SS \), and we assign \( u_x \) to the \( MS \) or an existing \( SS \) then the mapping does not need to be fixed. If we assign \( u_x \) to \( u_i \) then we need to make \( u_i \) an \( SS \).

(1) Set \( k \) to 1.
(2) Based on a fixed probability, determine if the \( k^{th} \) entry in the chromosome is mutated.
(3) If the entry is mutated, then:
   a) Generate a random assignment (connected to the \( MS, SS \), or user \( i \)).
   b) If the entry being modified is an \( SS \), then reassign the players assigned to this \( SS \) to existing \( SS \)s (selected randomly).
   c) 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).

Fig. 16. Procedure for using mutation to change a resource allocation.

The Genitor RT was seeded with the results from other heuristics by using the resource
8) Heuristic Design: While the underlying concepts of some of the heuristics are known, the actual heuristics described in this section are new and unique for this problem domain. For example, while the Tabu search concept is known and cited, the design of the long hops and short hops is new and unique for this study. As another example, while the discrete particle swarm optimization concept is known and cited, the interpretation of the binary values generated by the discrete particle swarm and how to convert them to a resource allocation in our environment is new and unique.

C. Lower Bound

The mathematical lower bound is to evaluate the experimental results of our heuristics for the minimization of $RT_{\text{max}}$. The bound has two components that can be calculated independently. The first component finds the minimum possible computation time of the MS and SSs (by performing an exhaustive search of all possible computation times). This component has two simplifying assumptions that are consistent with generating a lower bounds: (a) all users have the same computational constant ($\mu_{\text{min}} = \min_{U \in \mu} \mu_x$), and (b) users connected to SSs are evenly distributed among SSs. Component (a) removes the heterogeneity in computing power of the SSs, and (b) minimizes the maximum computation time among SSs. Let $n_{\text{base}} = n_{\text{nss}} + n_{\text{main}}$ be the total number of users that are connected to the MS. Given the assumptions above, we set $\Delta = \text{Comp}_{\text{MS}}$, and using Eqs. 1, 2, and 4 we can calculate the computation $f_{\text{comp}}$:

$$f_{\text{comp}}(n_{\text{base}}, n_{\text{nss}}) = \text{Comp}_a + 2 \cdot \text{Comp}_{\text{MS}}$$

$$= \mu_{\text{min}} \cdot \left[ \frac{N - n_{\text{base}}}{n_{\text{nss}}} \right]^2 + 2 \cdot \left[ c \cdot (N - n_{\text{base}}) + b \cdot n^2_{\text{base}} \right].$$

(8)

The second component is the lower bound on the communication time. We find the minimum time each user requires to connect to the MS (either connected directly to the MS or through another user), and then find the minimum among these times. Assuming user $x$ connects to the MS through user $y$ gives user $x$’s communication time $f_{\text{comm}}$:

$$f_{\text{comm}}(U_x, U_y) = \text{Comm}(U_x, U_y) + \text{Comm}(U_y, U_x) + \text{Comm}(U_y, \text{MS}) + \text{Comm}(\text{MS}, U_y).$$

(9)

The case where $U_x = U_y$ is considered to account for the case when $U_x$ is connected to the MS, i.e., $\text{Comm}(U_x, U_x) = 0$. The lower bound (LB) on $RT_{\text{max}}$ is given as:

$$\text{LB} = \min_{1 \leq n_{\text{base}} \leq N} \left( \min_{0 \leq n_{\text{nss}} \leq n} \left( f_{\text{comp}}(n_{\text{base}}, n_{\text{nss}}) \right) \right) + \min_{U_x \in \text{all users}} \left( \min_{U_y \in \text{all users}} \left( f_{\text{comm}}(U_x, U_y) \right) \right).$$

(10)
Proof: The proof has two parts. The first part will be to prove that the computational bound is minimum and the second part will be to prove the communication is minimum.

The first part of the bound does an exhaustive evaluation of all possible configurations for $n_{nss}$ and $n_{base}$. This will give us all the possible computations times. It will move $n_{base}$ from 1 (only one user connected to the MS) to $N$ (all users connected to the MS). For each of these values of $n$ it will attempt all possible configurations of $n_{nss} \leq n$. It is important to note that $n_{nss} = 0$ is an invalid configuration unless $n_{base} = N$ (i.e., the only scenario where we do not have SSs is when all users are connected directly to the MS), and in this case we consider $(N - N)/(0) = 0$. Because we are considering all the possible configurations it is not possible to get a smaller computation time.

The second part of the bound finds the smallest communication time for each user, then it finds the minimum among these times. This method does an exhaustive search of the possible communication times (through an SS or directly connected to the MS). Therefore, there is a user with this minimum communication time. To this user’s communication time we add the smallest possible computation time to get a lower bound on $RT_{\max}$.

D. Simulation Results

1) Simulation Setup: 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 communication times between nodes were allowed to vary from 0 to 40 ms with a uniform distribution. The computational constant ($\mu_{\alpha}$) 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 $\mu_{\alpha}$ for each user. For the purpose of comparison, each heuristic was limited to a maximum execution time of ten minutes per scenario.

The values of the various parameters for DPSO were selected by experimentation to optimize the performance. In our simulations, we used 30 particles. The $weight_g$ and $weight_p$ parameters were both set to 1500. The velocity was not attenuated ($w$ is set to 1), and the minimum and maximum velocity are −10,000 and 10,000, respectively. The DPSO was allowed to run for the full ten minutes.

For the Tabu Search, we limited the number of short hops to 100. The size of the tabu list
was not bounded. The Tabu Search heuristic was allowed to run for the full ten minutes.

For the Genitor heuristic, we used a population of 200 chromosomes. The probability of mutation per entry within the chromosome was set to 0.2% by experimentation similar in structure to DPSO. The Genitor heuristic was allowed to run for the full ten minutes.

2) Results for Minimization of Response Times: Fig. 17 shows the results averaged over the 100 scenarios. We can observe that the DPSO had the best performance in all cases (unseeded, seeded with Min-Min SS, and seeded with IM) and DPSO with the Min-Min SS seed was the best overall performing variation. Additionally, all heuristics were able to improve upon the allocation obtained from the Min-Min SS seed and from the IM.

There are three important observations about the performance of the heuristics in Fig. 17. First, we can see that in these results global search heuristics have a difficult time quickly finding local minima in the complete search space. The unseeded Genitor RT does not give a very good solution and takes ten minutes. This might be due to the fact that the complete search space is very large, and it has solutions that are not valid. Second, greedy heuristics can find good solutions in a small amount of time (less than 30 seconds). The last observation is that, in general, “directed” global search heuristics produce the best solutions.

The global search and “directed” search heuristics were able to performance better when they were seeded with a greedy heuristic. Between the DPSO and the Tabu Search, the DPSO searches better around the best solution than Tabu Search (DPSO algorithm favors exploration around the best solution). This means that the search space around the initial greedy solution is explored better in the DPSO heuristic and therefore it gets a slightly better solution.

The LB compares the heuristics to a mathematical bound on performance. The lower bound is about 44.7 time units less than the best performing heuristic (DPSO with Min-Min seed).

If all 200 users were connected to the MS (i.e., there are no secondary servers so $n_{secondary} = N_{nss} = 0$) then $RT_{max}$ would be about 2400 time units. This is based on Eqs. 2, 3, and 5 where we can assume zero communication time, $\Delta_{MS} = Comp_{MS}$, and $b = 0.03$:

$$RT_{max} = 2 \cdot Comp_{MS} = 2 \cdot b \cdot (n_{nss} + n_{main})^2$$

$$= 2 \cdot 0.03 \cdot 200^2 = 2,400.$$  \hspace{1cm} (11)

The use of the secondary server based approach in our simulations leads to an improvement of more than an order of magnitude (i.e., 112 time units versus 2400 time units).
IV. ROBUSTNESS TO ADDITIONAL PLAYERS JOINING THE GAME

A. Problem Statement

The purpose of this section is to determine an allocation that will allow the maximum number of new players to join an on-going game, i.e., be robust to additional players. The concept of robustness [29]–[31] is described in detail in Section IV-B.

The heuristics presented in this section are required 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 new players are not aware of the initial configuration of SSs, and therefore we assume that they can only connect to the MS. The latency for original users may increase above the QoS bound as new players join the game. The goal of the heuristics is to 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.

B. Robustness Metric
1) Overview: Using the FePIA (Performance Features, Perturbation Parameters, Impact, and Analysis) procedure described in [30], we define the characteristics that make the system robust. The FePIA procedure should respond to three fundamental robustness questions [30], [31]. First, what behavior of the system makes it robust? Second, what uncertainties is the system robust against? And third, how is robustness quantified?

2) Performance Feature: The first step of the FePIA procedure is to define the QoS requirement that makes the system robust. Here, this is that all the RTs are within a pre-determined range. The maximum RT time the system can allow is $\beta_{max}$:

$$RT_{max} \leq \beta_{max}.$$  \hspace{1cm} (13)

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

$$RT_{max} - RT_{min} \leq \Delta_{max}.$$ \hspace{1cm} (14)

For the system to be robust the constraints shown in Eqs. 13 and 14 need to be satisfied.

3) Perturbation Parameter: The second step of the FePIA procedure is to determine the perturbation parameter that is the uncertainty in the system. Here, the perturbation parameter is the number of new players joining the game after the initial resource allocation is done.

4) Impact of Perturbation Parameter 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 (as discussed in Section II-B). This increase in time will make the RT of all the users that are already in the game increase by the same amount, and hence $RT_{max}$ and $RT_{min}$ will increase by the same amount. Thus, if the initial resource allocation satisfies Eq. 14, then it will remain satisfied.

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., $RT_{max} - RT_{new} \leq \Delta_{max}$). Thus, new players have comparable time to other connected users.

5) Analysis: The number of new players that can be added to the system before $RT_{max}$ exceeds $\beta_{max}$ for existing players can be calculated exactly. We define $\Gamma$ 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, $\Gamma$ is given by:

$$\Gamma = Comm(U_x, MS) + Comm(MS, U_x),$$ \hspace{1cm} (15)
and if $U_x$ is connected to $SS_\alpha$ then $\Gamma$ is:

$$\Gamma = \text{Comm}(U_x, SS_\alpha) + \text{Comp}_\alpha + \text{Comm}(SS_\alpha, MS)$$
$$+ \text{Comm}(MS, SS_\alpha) + \text{Comm}(SS_\alpha, U_x).$$

(16)

Therefore, based on Eqs. (3) or (4) when $\Delta_{MS}$ or $\Delta_{SS}$ is equal to $\text{Comp}_{MS}$

$$RT_{max} = \Gamma + 2 \cdot \text{Comp}_{MS}.$$  

(17)

Let $\text{Comp}_{MS}'$ be the computation with $n_{\text{new}}$ new players added to the $MS$ such that

$$\beta_{\text{max}} = 2 \cdot \text{Comp}_{MS}' + \Gamma.$$  

(18)

That is, the system will be at the boundary of robustness when $2 \cdot \text{Comp}_{MS}' + \Gamma$ is equal to $\beta_{\text{max}}$

Let $n_{\text{base}}$ be equal to $n_{\text{main}} + n_{\text{nss}}$ from Eq. (2). This implies that

$$\beta_{\text{max}} = \Gamma + 2 \cdot (c \cdot n_{\text{secondary}} + b \cdot (n_{\text{base}} + n_{\text{new}})^2).$$  

(19)

The quadratic term can be expanded so that

$$\beta_{\text{max}} = \Gamma + 2 \cdot (c \cdot n_{\text{secondary}} + b \cdot (n_{\text{base}}^2 + 2 \cdot n_{\text{base}} \cdot n_{\text{new}} + n_{\text{new}}^2)).$$  

(20)

Combining Eq. 2 and 17,

$$RT_{max} = \Gamma + 2 \cdot \text{Comp}_{MS} = \Gamma + 2 \cdot (n_{\text{secondary}} + b \cdot n_{\text{base}}^2).$$  

(21)

Then Eq. 19 can be simplified to

$$\beta_{\text{max}} = RT_{max} + 2 \cdot b \cdot (2 \cdot n_{\text{base}} \cdot n_{\text{new}} + n_{\text{new}}^2).$$  

(22)

This can be re-written in standard quadratic form:

$$2 \cdot b \cdot n_{\text{new}}^2 + 4 \cdot b \cdot n_{\text{base}} \cdot n_{\text{new}} + (RT_{max} - \beta_{\text{max}}) = 0.$$  

(23)

With the roots given by the quadratic formula, the robustness metric, i.e., the maximum number of new players that can be added, is quantified as:

$$n_{\text{new}} = -n_{\text{base}} \pm \sqrt{n_{\text{base}}^2 - \frac{RT_{max} - \beta_{\text{max}}}{2 \cdot b}}.$$  

(24)

This result requires some interpretation, because it has two roots. If Eq. 24 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 Eq. 24 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 $RT_{\text{max}}$, which is determined by the given resource allocation; hence, better resource allocation (smaller $RT_{\text{max}}$) will result in larger values for the robustness metrics.

For some heuristics, it is necessary to give a “robustness” value to all possible resource allocations (even those that are not practical solutions). If the resource allocation cannot achieve robustness (i.e., Eq. 24 has two complex roots), then we approximate the robustness as:

$$n_{\text{new}} = \frac{-\sqrt{RT_{\text{max}} - \beta_{\text{max}}} \cdot b}{\sqrt{2}}. \quad (25)$$

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

C. Heuristics for Maximizing Robustness

1) Recursive Optimization Algorithm for Robustness (ROAR): The Recursive Optimization Algorithm for Robustness (ROAR) iteratively adds SSs and uses the Min-Min RT algorithm to assign non-DCUs to DCUs. Initially, the ROAR heuristic creates a sorted list ($\text{COMM \ list}$) of users in ascending order of communication time to the MS. The first element of this list is added as a DCU, and the Min-Min RT heuristic is used to assign the non-DCUs. If the constraints are met by this resource allocation, then the robustness is calculated, and compared against the best known robustness. If the constraints are not met, then the next element in the list is also added as a DCU. This procedure continues until a stopping criteria is met, which is the number of iterations without improvement, or we have added all the users as SSs. The procedure for the ROAR heuristic is shown in Fig. 18.

![Fig. 18. Procedure for using the ROAR heuristic to generate a resource allocation.](image)

To understand why this heuristic is a reasonable approach, consider the equations used to describe the system (i.e., Eqs. 2, 3, and 4). If all users are connected to the MS (no SSs) then
there is a quadratic increase in \( \text{Comp}_{\text{MS}} \) and therefore the \( RT_x \) of all users; likewise, if we try to make all users \( SS \)s then we encounter the same problem. We increase the number of \( SS \)s one by one (so that those users with the smallest latency to the \( MS \) are chosen first) and find the inflection point of \( RT_{\text{max}} \) versus number of \( SS \)s. This is a very fast greedy approach.

2) **Robust Tabu Search**: The Robust Tabu Search is very similar to the Tabu Search in Section III-B5. The procedure used to generate a resource allocation with Robust Tabu Search is shown in Fig. 19. This uses the short hop procedure. In Fig. 12, the differences are that we switch from minimizing \( RT_{\text{max}} \) to maximizing the robustness in steps 2, 3(b), 3(b).ii, 3(f), and 3(f).ii.

| (1) Create a tabu list of the visited neighborhoods. |
| (2) While the execution time is less than 10 minutes |
| (a) Performs a long hop by generating a random set of \( DCUs \) (\( \text{rand}_{\text{DCUs}} \)). If there is at least 50% difference between the long hop and previous solutions in the tabu list, and the solution meets the QoS constraints, then continue to step (b), otherwise repeat step (a). |
| (b) Use the Min-Min RT heuristic with \( \text{rand}_{\text{DCUs}} \) to generate a full mapping. |
| (c) Use the short hop procedure in Fig. 12 modifying steps 2, 3(b), 3(b).ii, 3(f), 3(f).ii to maximize robustness. |
| (d) If the solution meets the QoS constraints, then update the tabu list by adding the set of \( DCUs \) from step (c). |

Fig. 19. Procedure for using the Robust Tabu to generate a resource allocation.

3) **Robust Discrete Particle Swarm Optimization**: The Robust Discrete Particle Swarm Optimization (Robust DPSO) is very similar to the Discrete Particle Swarm Optimization (DPSO) in Section III-B6. The Robust DPSO and DPSO heuristics have two main differences. First, the objective function of Robust DPSO is to maximize robustness instead of minimizing \( RT_{\text{max}} \). The second difference is that Robust DPSO checks the resource allocation to ensure that the fairness constraints are met in steps 3 and 5(c) in Fig. 13.

4) **Robust Genitor**: The Robust Genitor heuristic is very similar to the Genitor RT (Section III-B7). The differences are that the chromosomes are ranked based on decreasing robustness instead of increasing \( RT_{\text{max}} \) and chromosomes are only allowed entry into the population if both fairness constraints are met in step 2(d) in Fig. 14.

D. **Upper Bound**

The primary purpose of deriving a mathematical upper bound was to evaluate the experimental results of our proposed heuristics for the maximization of robustness. This bound is based on the lower bound in Section III-C, and uses the same simplifying assumptions. The basic idea of the upper bound is (1) to find a lower bound on \( RT_{\text{max}} (RT_{\text{bound}}) \) for each specific configuration (i.e., values of \( n_{\text{base}}, n_{\text{nss}}, \) and \( n_{\text{main}} \)), and (2) using \( RT_{\text{bound}} \) with the number of users connected to the \( MS \) (\( n_{\text{base}} = n_{\text{nss}} + n_{\text{main}} \)) to calculate a true upper bound using Eq. 24. Let \( \text{Comm}_{\text{min}} \)
denote the communication part of Eq. 10 given by

\[ \text{Comm}_{\text{min}} = \min_{U_x \in \text{all users}} \left( \min_{U_y \in \text{all users}} \left( f_{\text{comm}}(U_x, U_y) \right) \right). \]  

(26)

Then, \( f_{\text{comp}} \) from Eq. 8 is used to calculate the computation required due to \( n_{\text{base}} \) and \( n_{\text{nss}} \):

\[ RT_{\text{bound}}(n_{\text{base}}, n_{\text{nss}}) = f_{\text{comp}}(n_{\text{base}}, n_{\text{nss}}) + \text{Comm}_{\text{min}}. \]  

(27)

The robustness (number of new players that can be added) associated with a particular \( RT_{\text{bound}} \) can be calculated using Eq. 24 by substituting \( RT_{\text{bound}} \) for \( RT_{\text{max}} \). If the robustness of a configuration can be calculated with the solution to the quadratic equation shown in Eq. 24, then the specific \( f_{\text{quad}} \) will be a positive value, i.e.,

\[ f_{\text{quad}}(n_{\text{base}}, n_{\text{nss}}) = n_{\text{base}}^2 - \frac{RT_{\text{bound}}(n_{\text{base}}, n_{\text{nss}}) - \beta_{\text{max}}}{2 \cdot b}. \]  

(28)

If \( f_{\text{quad}} \) is positive, then

\[ \text{rob}_{\text{max}}(n_{\text{base}}, n_{\text{nss}}) = -n_{\text{base}} + \sqrt{f_{\text{quad}}(n_{\text{base}}, n_{\text{nss}})}. \]  

(29)

If \( f_{\text{quad}} \) is negative, we have complex roots and the game cannot be played in this configuration so that these roots can be ignored.

The upper bound (\( UB \)) will be the maximum \( \text{rob}_{\text{max}} \) over all possible configurations, i.e.,

\[ UB = \max_{1 \leq n_{\text{base}} \leq N} \left( \max_{0 \leq n_{\text{nss}} \leq n_{\text{base}}} \left( \text{rob}_{\text{max}}(n_{\text{base}}, n_{\text{nss}}) \right) \right). \]  

(30)

To show that the bound is true, we must first prove that \( RT_{\text{bound}} \) is a lower bound on \( RT_{\text{max}} \) for a specific \( n_{\text{base}} \) and \( n_{\text{nss}} \). The value of \( RT_{\text{bound}} \) is composed of the communication lower bound and the computation lower bound with \( \Delta = \text{Comp}_{\text{MS}} \).

The lower bound on communication does an exhaustive search of the possible communication times (through an \( SS \) or directly connected to the \( MS \)). Therefore, no user can have a smaller communication time than \( \text{Comm}_{\text{min}} \) independent of the configuration.

The lower bound on computation (\( \Delta = \text{Comp}_{\text{MS}} \)) will calculate the minimum computation given a specific configuration. It consider values of \( n_{\text{base}} \) from 1 (only one user connected to the \( MS \)) to \( N \) (all users connected to the \( MS \)). For each of these values of \( n_{\text{base}} \), all possible values of \( n_{\text{nss}} \leq n_{\text{base}} \) are considered. For each combination of \( n_{\text{base}} \) and \( n_{\text{nss}} \) we generate a value of \( f_{\text{comp}} \), and the sum of \( f_{\text{comp}} \) and the bound on communication will give us \( RT_{\text{bound}} \). For each \( RT_{\text{bound}} \), the robustness can be calculated using Eq. 29, and the maximum is the \( UB \).
Fig. 20. Results for maximizing the robustness of the system against additional players joining the game. The computational parameters of the $MS$ were set to $b = 0.03$ and $c = 0.01$, values are averaged over 100 scenarios, and the error bars show the 95% confidence intervals. The values for $\beta_{\text{max}}$ and $\Delta_{\text{max}}$ are 200 and 150 milliseconds, respectively.

E. Simulation Results

1) Simulation Setup: The simulation setup is the same as in Section III-D1. The values for $\beta_{\text{max}}$ and $\Delta_{\text{max}}$ are 200 and 150 milliseconds, respectively. These values for $\beta_{\text{max}}$ and $\Delta_{\text{max}}$ were selected because they are realistic parameters for the QoS constraints.

2) Results for Maximization of Robustness: The Robust Tabu Search, Robust DPSO, and Robust Genitor were run with and without a seed as shown in Fig. 20. The results shown were the best results for each heuristic found after doing parameter sweeps on controlled parameters, e.g., probability of mutation in the Robust Genitor and velocity weighting parameters in DPSO. The Robust Tabu Search, Robust DPSO, and Robust Genitor had an execution time of ten minutes, while the ROAR and Min-Min SS had an execution time of less than one minute.

The Min-Min SS heuristic was used as a comparison to see how heuristics that optimize $RT_{\text{max}}$ perform when considering robustness as the optimization criterion. It had a performance that was not able to add as many users as the ROAR heuristic (on average it could add approximately 8 fewer players). The unseeded Robust Genitor did not perform well. This could be due to the method used for generating random resource allocations, i.e., resource allocation with a negative robustness were not screened out of the initial population. For the Robust Tabu, the average result from the long hop (over all long hops in all scenarios) 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 resource allocation, with an average of 24.5 short hops. This shows that the short hops are able to improve the resource allocation by exploring the neighborhood.

The Robust Tabu, Robust Genitor, and Robust DPSO significantly improved with the introduction of the seed. However, this improvement in performance was mostly due to the high robustness generated by the ROAR heuristic. The DPSO heuristic had a 0.21 (≈ 1%) improvement, Tabu Search had a 0.15 (≈ 1%) improvement, and Robust Genitor had a 0.41 (≈ 2%) over the ROAR heuristic. The results of the ROAR seeded heuristics were (on average) 7.4 time units less than the $UB$ (about 76% of the $UB$). Thus, for this simulation study, the implemented heuristics allow the addition of about 22 uses to the original 200, for a total of 222 users, all who meet the QoS constraints.

In comparison, if no secondary servers were used, an estimate of the maximum number of users that can be connected directly to the $MS$ (i.e., there are no secondary servers, so $n_{secondary} = n_{nss} = 0$), without violating the QoS constraint ($\beta_{max} = 200$) can be calculated for the parameter used in the simulation. For this calculation, we do not consider the communication time. We assume that $\Delta_{MS} = Comp_{MS}$, and $\beta_{max} = RT_{max}$. Using Eqs. 2, 3, and 5

$$\beta_{max} = 2 \cdot Comp_{MS} = 2 \cdot b \cdot n_{main}^2$$

so that

$$2 \cdot 0.03 \cdot n_{main}^2$$

$$n_{main} = \sqrt{100/0.03} \approx 58$$

Thus only 58 users can be connected directly to the $MS$ while still maintaining the QoS constraints. Therefore, the use of $SS$s and heuristics was able to more than triple the performance of an approach where all users are connected directly to the $MS$.

The results from the heuristics for robustness maximization show that with the constraints set for this environment, a large number of users can be added while maintaining the fairness conditions (greater than 10% more users). Thus, the ROAR heuristic with its one minute execution time maybe preferred in practice to the Robust Genitor (which is only 2% better with an execution time of ten minutes).

V. RELATED WORK

Various MMOG architectures are reported in the literature (e.g., client/server [32], [33], peer-to-peer [7], [18], [34]–[36], mirrored server [37]). Each architecture has its own advantages. For
example, the client/server and mirrored server allows 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 maintain 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. The use of the centralized server in the hybrid approach may have a single point of failure, however it allows the game developer to control the MMOG environment and uses peers to reduce the computation of the main server. Our work is different from [32], [37] because it considers converting users to secondary servers. Our work is also different from [7], [18], [34] because it has a “non-peer” centralized server, and fairness is not directly addressed.

In [33], a hybrid approach is presented where peers are clustered together and they update movement information independent of the main server. Because movement information can make up a significant amount of the traffic generated by an MMOG [38], this off-loading can reduce the main server’s computational load. The work in [33] shares similarities with our work, however the focus of our study is resource allocation considering latency and heterogeneity in a hybrid MMOG environment. Our algorithm uses heterogeneity of computational capabilities as part of the information used to make resource allocation decisions. Also, the study in [33] does not use a robustness metric to evaluate resource allocation.

The authors in [39] present a comparison between a client/server based architecture and a hybrid client/server architecture. The are several differences between our study and their study. For our study, we define a robustness metric to guarantee fairness in the MMOG environment. The authors in [39] use a different metric to evaluate performance (the ratio of packets that miss a 300 ms deadline to serve a request).

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 [40], 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 study and theirs is that in [40] the intermediate servers are predefined and do not participate as users in the MMOG, and
we have a robustness criterion to guarantee fairness.

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 [17], [32]. In [18], the authors show that the latency follows a “... weak exponential increase ...” as the number of users in the system increases. Our study focuses on latency as a critical performance parameter that must be maintained and uses the results in [18] to model the relationship between latency and the number of users.

VI. CONCLUSIONS

In this study, we created a detailed mathematical model of a hybrid MMOG environment, and derived metrics to analyze the performance of the system. For the first part of the study, we designed heuristics that minimize the maximum response time ($RT_{max}$) among all players in this environment. Heuristics for this environment need to determine (a) how many users are converted into secondary servers, (b) which users are converted into secondary servers, and (c) how the remaining users are connected among the secondary servers and the main server. For this environment, we derived a mathematical lower bound on $RT_{max}$, and showed it to be a true lower bound. We used a simulation study to compare heuristics against each other and to the lower bound. In this part of the study, we saw that we could decrease the response time from approximately 2.4s (with all users connected to the main server) to about 112ms using the proposed resource allocation heuristics.

The mathematical model of the MMOG environment was additionally used to address the problem of adding players to an on-going game session. The problem of adding players was modeled in terms of fairness and robustness. We designed heuristics to maximize a robustness metric, i.e., number of player that can join an on-going game, that guarantees (using QoS constraints) the configuration of the system is fair. We derived a mathematical upper bound on the number of players that can be added, and used this bound to evaluate the performance of the heuristics. In this part of the study, using the proposed resource allocation heuristics, we were able to add approximately 10% more players ($\approx 22$) while maintaining a system that is fair for a total of 222 players. If all users were connected directly to the main server, then we would able to support a maximum of 58 users within the fairness constraints. This shows that, in our environment, the hybrid client/server configuration found by the resource allocation heuristics can more than triple the number of players that can interact the system.
There are multiple extensions to this research that could be done to improve the accuracy of the model. For example, we could include more realistic behaviors like players randomly leaving the match (requiring a mechanism to ensure the game world does not lose its state when users acting as secondary servers leave or fail) or different computational requirements for each player. The true culmination of this research will be when it can be included in the infrastructure of a real commercial MMOG.

ACKNOWLEDGMENTS

This research was supported by the NSF under grants CNS-0615170 and CNS-0905399, and by the Colorado State University George T. Abell Endowment. A preliminary version of portions of this material appeared in the 17th Heterogeneity in Computing Workshop, and in the 4th International Conference on Foundations of Digital Games. The authors thank Richard Wallace, Paul Maxwell, and Samee Khan for their valuable contributions.

REFERENCES


30


BIOGRAPHIES

Luis Diego Briceño is currently pursuing his Ph.D. degree in Electrical and Computer Engineering at Colorado State University. He obtained his B.S. degree in electrical and electronic engineering from the University of Costa Rica. His research interests include heterogeneous parallel and distributed computing.

Howard Jay Siegel was appointed the Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU) in 2001, where he is also a Professor of Computer Science. From 1976 to 2001, he was a professor at Purdue University. Prof. Siegel is a Fellow of the IEEE and a Fellow of the ACM. He received two B.S. degrees from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from Princeton University. He has co-authored over 400 technical papers. His research interests include robust computing systems, resource allocation in computing systems, heterogeneous parallel and distributed computing and communications, parallel algorithms, and parallel machine interconnection networks. He was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing, and was on the Editorial Boards of both the IEEE Transactions on Parallel and Distributed Systems and the IEEE Transactions on Computers. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. Homepage: www.engr.colostate.edu/~hj.

Anthony A. Maciejewski received the B.S., M.S., and Ph.D. degrees in Electrical Engineering in 1982, 1984, and 1987, respectively, all from The Ohio State University. From 1988 to 2001, he was a Professor of Electrical and Computer Engineering at Purdue University. In 2001, he joined Colorado State University where he is currently the Head of the Department of Electrical and Computer Engineering. He is a Fellow of IEEE. A complete vita is available at www.engr.colostate.edu/~aam.
Mohammad Nayeem Teli: I am a Ph.D. Candidate in Computer Science Department at Colorado State University. I am interested in Computer Vision, machine learning and big data analysis in general. In particular I have interest in face detection, face recognition, biometrics, correlation filters and EEG analysis. Currently I am pursuing research in face detection in videos and still images. In my spare time I like to spend time hiking, camping and climbing in wilderness and play sports.

Brad Lock: Brad received his BS in Electrical and Computer engineering from Colorado State University in 2001 and his MS in Computer Engineering from Colorado State University in 2011. He currently works for Intel.

Fadi Ibrahim Ali Wedyan: Fadi received his Ph.D. in Computer Science from Colorado State University in 2011. He is currently an assistant professor at The Hashemite University in Jordan.

Ye Hong received his Master degree in Computer Science from Tsinghua University in 2006, and his Bachelor degree from Tsinghua University in 2002. His current research interests include parallel and distributed computing, heterogeneous computing, robust computer systems, and performance evaluation.

Charles Panaccione: Charles is pursuing his Ph.D. at Colorado State University in Computer Science. He has worked for the National Center for Atmospheric Research and for Pearson.

Chen Zhang: Chen is pursuing her master’s degree in Computer Science at Colorado State University.