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

Luis Diego Briceño, Howard Jay Siegel, *Fellow*, *IEEE*, Anthony A. Maciejewski, *Fellow*, *IEEE*, Ye Hong, Brad Lock, Charles Panaccione, Fadi Wedyan, Mohammad Nayeem Teli, and Chen Zhang

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.

Index Terms—Heterogeneous computing, resource allocation, massive multiplayer online games

1 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 *MMOG*s 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 (*SS*s) 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

- *L.D. Briceño, H.J. Siegel, A.A. Maciejewski, Y. Hong, and B. Lock are with the Department of Electrical & Computer Engineering, Colorado State University, Fort Collins, CO 80523. E-mail: {LDBricen, HJ, AAM, YHong, Bradley.Lock}@colostate.edu.*
- *H.J. Siegel, C. Panaccione, F. Wedyan, M.N. Teli, and C. Zhang are with the Department of Computer Science, Colorado State University, Fort Collins, CO 80523. E-mail: {Charles.Panaccione, Fadi.Wedyan, Mohammad.Teli, Chen.Zhang}@colostate.edu.*

Manuscript received 18 Mar. 2013; revised 21 July 2013; accepted 29 July 2013. Date of publication 05 Sep. 2013; date of current version 12 Nov. 2014. Recommended for acceptance by D. Bader.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2013.178

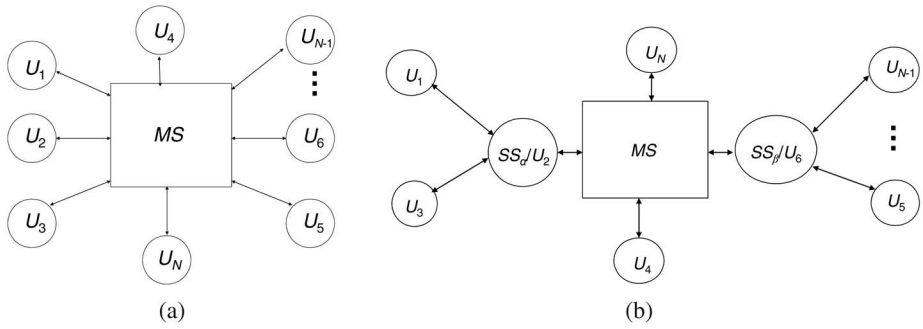


Fig. 1. (a) Client/server architecture, using a single server to do processing; and (b) secondary server architecture, using users' computers to assist the Main Server in processing.

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.

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 environment, 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 2 provides the common environment. In Section 3, 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 4 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 5, and in Section 6 we present our conclusions.

2 Environment

2.1 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

$n_\alpha \rightarrow$ number of users connected to SS_α
$\mu_\alpha \rightarrow$ computational constant of SS_α
$Comp_\alpha \rightarrow$ computation time for SS_α
$n_{secondary} \rightarrow$ total number of users connected to all the SSs
$n_{main} \rightarrow$ the number of users connected directly to the MS
$n_{nss} \rightarrow$ number of SSs
$Comp_{MS} \rightarrow$ computation time for MS
b and $c \rightarrow$ computational constants for $Comp_{MS}$
$U_x \rightarrow$ user x
$RT_x \rightarrow$ response time for U_x
$Comm(A, B) \rightarrow$ communication time between node A and node B
Δ_{MS} and $\Delta_{SS} \rightarrow$ time a packet has to wait at the MS or SS respectively
$RT_{max} \rightarrow max_{\forall U_x}(RT_x)$ with Δ_{MS} or $\Delta_{SS} = Comp_{MS}$
$RT_{min} \rightarrow min_{\forall U_x}(RT_x)$ with Δ_{MS} or $\Delta_{SS} = 0$
$X_{ij} \rightarrow$ represents whether user j is a DCU in particle i
V_{min} and $V_{max} \rightarrow$ represents the minimum and maximum allowed velocity for DPSO
$V_{ij} \rightarrow$ represents the velocity of particle i in the direction j
$w \rightarrow$ coefficient used to slow down particle
$iter_{max} \rightarrow$ limit on number of iterations for a DPSO heuristic
$P^i \rightarrow$ record of best solution for particle i
$weight_p$ and $weight_g \rightarrow$ coefficients of explorations around the best personal and global solution (respectively)
$G \rightarrow$ record of best global solution
$\beta_{max} \rightarrow$ the maximum RT time the system can allow while still being robust
$\Delta_{max} \rightarrow$ a time window used to specify the allowable range of RT_x for all users
$\Gamma \rightarrow$ component of RT equation that does not depend on number of players connected to MS
$n_{new} \rightarrow$ number of new players added to the MS
$n_{base} \rightarrow n_{main} + n_{nss}$
$RT_{new} \rightarrow RT_{max}$ of the system with n_{new} players
$Comp'_{MS} \rightarrow$ the computation at the MS with n_{new} players added

Fig. 2. Glossary of terms.

number of users connected to an SS or the MS. These assumptions are used to reduce the complexity of the simulations. A glossary of terms used is in Fig. 2.

2.2 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 SSs, 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_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 computing power heterogeneity across different users' computers (each user has a different constant). 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 the MS, and b and c be computational constants of the MS, $0 \leq b, c \leq 1$. The computation time of the MS ($Comp_{MS}$) is:

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

We assume the game world state is updated and synchronized every $Comp_{MS}$ time units.

2.3 Objective Functions RT_{max} and RT_{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 $Comm(A, B)$ be the communication time between node A and node B.

Let Δ_{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:

$$RT_x = Comm(U_x, MS) + \Delta_{MS} + Comp_{MS} + Comm(MS, U_x). \quad (3)$$

Consider the case where U_x is connected to SS_α as illustrated in Fig. 1(b). The SS_α knows its computation time (based on number of users assigned to it), and its communication time to the MS. We assume that SS_α will send out an update immediately before the game world is synchronized across users. For example in Fig. 3, SS_α begins $Comp_\alpha$ so that $Comp_\alpha$ and $Comm(SS_\alpha, MS)$ completes at time $t(k+1)$. We also

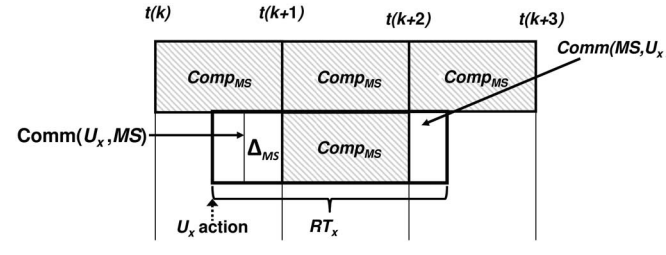


Fig. 3. Return time for a user x (U_x) connected directly to the MS.

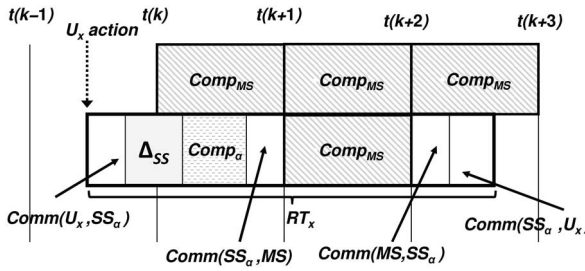


Fig. 4. Return time for a user x (U_x) connected to the MS through an SS.

assume that $Comp_\alpha + Comm(SS_\alpha, MS) \leq Comp_{MS}$. Let $\underline{\Delta_{SS}}$ be the wait time between a U_x action and when SS_α's computation starts. If a user is connected to an SS_α then the equation is:

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

A graphical representation of this equation is shown in Fig. 4. If U_x is SS_α, then Eq. 4 is used with $Comm(U_x, SS_\alpha) = Comm(SS_\alpha, U_x) = 0$. 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 $Comp_{MS}$ and $Comp_\alpha$ 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_α then the maximum time it will be waiting for computation is Δ_{MS} or Δ_{SS} equal to $Comp_{MS}$. Thus, at the next mapping time interval, to calculate the maximum response time (i.e., $\underline{RT_{max}}$) we use:

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

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

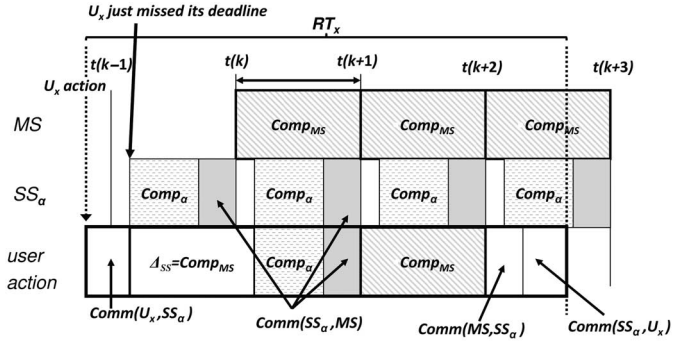


Fig. 5. RT_{max} when the user just *misses* the deadline for sending computation to the SS_α, i.e., $\Delta_{SS} = Comp_{MS}$.

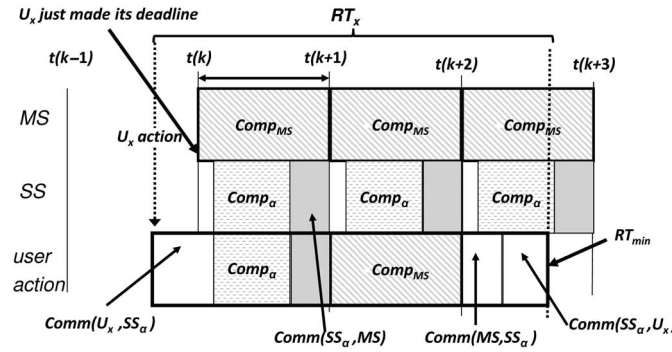


Fig. 6. RT_{min} when the user with just *makes* the deadline for sending computation at the SS, i.e., $\Delta = 0$.

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

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

3 Response Time Minimization

3.1 Problem Statement

To deal with the situation when the MS is oversubscribed, we consider in this section the conversion of users to SSs 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 SSs perform computation and the MS resolves conflicts among users and SSs 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).

In Section 3.2, we describe six heuristics for minimizing RT_{max} by converting selected users to SSs. A lower bound on RT_{max} is derived in Section 3.3. Our evaluation of the heuristics through simulations is given in Section 3.4.

3.2 Heuristics for Response Time Minimization

3.2.1 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 ($\underline{DCU}$s) 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

(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 PHs, find the PH that minimizes RT of u connected indirectly to the MS through PH ($RT_{u \to PH \to MS}$).
    (i) If $RT_{u \to PH \to MS}$ 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.

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.

3.2.2 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 SSs). The procedure to implement the Min-Min RT is shown in Fig. 7.

3.2.3 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 Section 3.2.1, the minimum RT may be through an SS.

3.2.4 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 PHs 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.

3.2.5 Tabu Search

The Tabu Search concept [24] is a directed global search heuristic that stores the previously visited areas in the search

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

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

(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 ($rand_{DCUs}$). If there is at least 50% difference between the long hop and previous solutions 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.
  (c) Use the short hop procedure.
  (d) Update the tabu list by adding the set of DCUs from step (c).

Fig. 11. Procedure for using the Tabu Search to generate a resource allocation.

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

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

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 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 $max_{shorthops}$ 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.

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.

3.2.6 Discrete Particle Swarm Optimization

Discrete Particle Swarm Optimization (DPSO) is a "directed" global search heuristic based on the particle swarm

(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) $V_{ij} = w \cdot V_{ij} + weight_p \cdot R_1 \cdot (P^i_j - X_{ij}) + weight_g \cdot R_2 \cdot (G_j - X_{ij})$
    (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.

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_{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 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 ($iter_{max}$), determined based on the maximum allowed execution time.

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 ($weight_p$, 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 ($weight_g$). 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 SSs as the resource allocation generated by the seed heuristic; Only the set of SSs is taken from the seeds not all user assignments.

3.2.7 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 ith entry is where the ith 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.

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) 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.

(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 and R_2 with $R_1 < R_2$)
(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.

(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 SSs (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 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 SSs (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 SSs.

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.

The Genitor RT was seeded with the results from other heuristics by using the resource allocation generated by the seed heuristic as a chromosome in the population.

3.2.8 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.

3.3 Lower Bound

The mathematical lower bound is to evaluate the experimental results of our heuristics for the minimization of RT_{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 ($\mu_{min} = \min_{\forall U_x} \mu_x$), 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 $n_{base} = n_{nss} + n_{main}$ be the total number of users that are connected to the MS. Given the assumptions above, we set $\Delta = Comp_{MS}$, and using Eqs. 1, 2, and 4 we can calculate the computation f_{comp}:

$$f_{comp}(n_{base}, n_{nss}) = Comp_{\alpha} + 2 \cdot Comp_{MS} \quad (7)$$

$$= \mu_{min} \cdot \left\lceil \frac{N - n_{base}}{n_{nss}} \right\rceil^2 + 2 \cdot \left[c \cdot (N - n_{base}) + b \cdot n_{base}^2 \right]. \quad (8)$$

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_{comm}:

$$f_{comm}(U_x, U_y) = Comm(U_x, U_y) + Comm(U_y, U_x) + Comm(U_y, MS) + Comm(MS, U_y). \quad (9)$$

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

$$LB = \min_{1 \le n_{base} \le N} \left(\min_{0 \le n_{nss} \le n} \left(f_{comp}(n_{base}, n_{nss}) \right) \right) + \min_{U_x \in all\ users} \left(\min_{U_y \in all\ users} \left(f_{comm}(U_x, U_y) \right) \right). \quad (10)$$

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 $n_{nss} \le n$. 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}. □

3.4 Simulation Results

3.4.1 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 (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 comparison, each heuristic was limited to a maximum execution time of ten minutes per scenario.

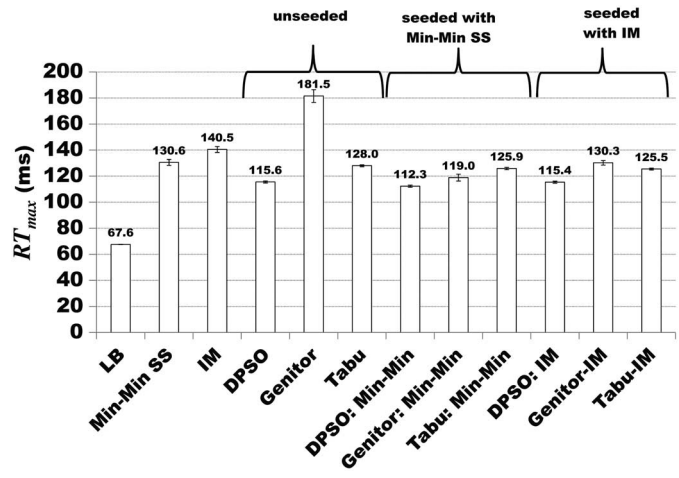


Fig. 17. Results for response time minimization. The computational parameters of the MS were set to: $b = 0.03$ and $c = 0.01$, values are averaged over 100 scenarios, and the error bars show the 95% confidence intervals.

The values of the various parameters for DPSO were selected by experimentation to optimize the performnace. In our simulations, we used 30 particles. The $weight_g$ and $weight_p$ parameters were both set to 1500. The velocity was not attentuated (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.

3.4.2 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 $b = 0.03$:

$$RT_{max} = 2 \cdot Comp_{MS} = 2 \cdot b \cdot (n_{nss} + n_{main})^2 \quad (11)$$

$$= 2 \cdot 0.03 \cdot 200^2 = 2{,}400. \quad (12)$$

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).

4 Robustness to Additional Players Joining the Game

4.1 Problem Statement

The purpose of this section is to determine an allocation that will allow the maximum number of new players to join an ongoing game, i.e., be robust to additional players. The concept of robustness [29]–[31] is described in detail in Section 4.2.

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.

4.2 Robustness Metric

4.2.1 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?

4.2.2 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 β_{max}:

$$RT_{max} \leq \beta_{max}. \quad (13)$$

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

$$RT_{max} - RT_{min} \leq \Delta_{max}. \quad (14)$$

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

4.2.3 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.

4.2.4 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 2.2). 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.

4.2.5 Analysis

The number of new players that can be added to the system before RT_{max} exceeds β_{max} for existing players can be calculated exactly. 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:

$$\Gamma = Comm(U_x, MS) + Comm(MS, U_x), \quad (15)$$

and if U_x is connected to SS_α then Γ is:

$$\Gamma = Comm(U_x, SS_\alpha) + Comp_\alpha + Comm(SS_\alpha, MS) + Comm(MS, SS_\alpha) + Comm(SS_\alpha, U_x). \quad (16)$$

Therefore, based on Eqs. 3 or 4 when Δ_{MS} or Δ_{SS} is equal to $Comp_{MS}$

$$RT_{max} = \Gamma + 2 \cdot Comp_{MS}. \quad (17)$$

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

$$\beta_{max} = 2 \cdot Comp'_{MS} + \Gamma. \quad (18)$$

(1) Create a list with all users sorted in ascending order based on communication time to the MS ($COMM_{list}$).
(2) Set $best\ robustness$, i, and j to 0.
(3) While $j <$ stopping criterion and $i < N$ do:
  (a) Add the i^{th} entry in the list to the set of DCUs (DCU_{set}). Note that when a user is added to DCU_{set}, it will remain in DCU_{set} until the heuristic finishes executing.
  (b) Use the Min-Min RT heuristic with DCU_{set} to generate a full mapping.
  (c) If the QoS constraints are not met then go to step (f), otherwise continue to step (d).
  (d) Calculate the robustness of the current mapping (R_x).
  (e) If $R_x > best\ robustness$ then
    (i) $best\ robustness \leftarrow R_x$ and $j \leftarrow 0$.
    (ii) The current mapping is stored as the best known resource allocation.
    Otherwise, increment j by 1.
  (f) Increment i by 1.
(4) The best known resource allocation is output.

Fig. 18. Procedure for using the ROAR heuristic to generate a resource allocation.

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

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

$$\beta_{max} = \Gamma + 2 \cdot (c \cdot n_{secondary} + b \cdot (n_{base} + n_{new})^2). \quad (19)$$

The quadratic term can be expanded so that

$$\beta_{max} = \Gamma + 2 \cdot (c \cdot n_{secondary} + b \cdot (n_{base}^2 + 2 \cdot n_{base} \cdot n_{new} + n_{new}^2)). \quad (20)$$

Combining Eq. 2 and 17,

$$RT_{max} = \Gamma + 2 \cdot Comp_{MS} = \Gamma + 2 \cdot \left(n_{secondary} + b \cdot n_{base}^2\right). \quad (21)$$

Then Eq. 19 can be simplified to

$$\beta_{max} = RT_{max} + 2 \cdot b \cdot (2 \cdot n_{base} \cdot n_{new} + n_{new}^2). \quad (22)$$

This can be re-written in standard quadratic form:

$$2 \cdot b \cdot n_{new}^2 + 4 \cdot b \cdot n_{base} \cdot n_{new} + (RT_{max} - \beta_{max}) = 0. \quad (23)$$

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:

$$n_{new} = -n_{base} \pm \sqrt{n_{base}^2 - \frac{RT_{max} - \beta_{max}}{2 \cdot b}}. \quad (24)$$

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_{max}, which is determined by the given resource allocation; hence, better resource allocation (smaller RT_{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:

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

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

4.3 Heuristics for Maximizing Robustness

4.3.1 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 ($COMM_{list}$) 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 $COMM_{list}$ 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.

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 $Comp_{MS}$ and therefore the RT_x of all users; likewise, if we try to make all users SSs then we encounter the same problem. We increase the number of SSs one by one (so that those users with the smallest latency to the MS are chosen first) and find the inflection point of RT_{max} versus number of SSs. This is a very fast greedy approach.

4.3.2 Robust Tabu Search

The Robust Tabu Search is very similar to the Tabu Search in Section 3.2.5. 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

(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 ($rand_{DCUs}$). 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 $rand_{DCUs}$ 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.

that we switch from minimizing RT_{max} to maximizing the robustness in steps 2, 3(b), 3(b).ii, 3(f), and 3(f).ii.

4.3.3 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 3.2.6. 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_{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.

4.3.4 Robust Genitor

The Robust Genitor heuristic is very similar to the Genitor RT (Section 3.2.7). The differences are that the chromosomes are ranked based on decreasing robustness instead of increasing RT_{max} and chromosomes are only allowed entry into the population if both fairness constraints are met in step 2(d) in Fig. 14.

4.4 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 3.3, and uses the same simplifying assumptions. The basic idea of the upper bound is (1) to find a lower bound on RT_{max} (RT_{bound}) for each specific configuration (i.e., values of n_{base}, n_{nss}, and n_{main} and (2) using RT_{bound} with the number of users connected to the $MS(n_{base} = n_{nss} + n_{main})$ to calculate a true upper bound using Eq. 24. Let $Comm_{min}$ denote the communication part of Eq. 10 given by

$$Comm_{min} = \min_{U_x \in all\ users} \left(\min_{U_x \in all\ users} (f_{comm}(U_x, U_y)) \right). \quad (26)$$

Then, f_{comm} from Eq. 8 is used to calculate the computation required due to n_{base} and n_{nss}

$$RT_{bound}(n_{base}, n_{nss}) = f_{comp}(n_{base}, n_{nss}) + Comm_{min}. \quad (27)$$

The robustness (number of new players that can be added) associated with a particular RT_{bound} can be calculated using Eq. 24 by substituting RT_{bound} for RT_{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_{quad} will be a positive value, i.e.,

$$f_{quad}(n_{base}, n_{nss}) = n_{base}^2 - \frac{RT_{bound}(n_{base}, n_{nss}) - \beta_{max}}{2 \cdot b} \quad (28)$$

If f_{quad} is positive, then

$$rob_{max}(n_{base}, n_{nss}) = -n_{base} + \left| \sqrt{f_{quad}(n_{base}, n_{nss})} \right|. \quad (29)$$

If f_{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 rob_{max} over all possible configurations, i.e.,

$$UB = \max_{1 \le n_{base} \le N} \left(\max_{0 \le n_{nss} \le n_{base}} (rob_{max}(n_{base}, n_{nss})) \right). \quad (30)$$

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

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 $Comm_{min}$ independent of the configuration.

The lower bound on the computation ($\Delta = Comp_{MS}$) will calculate the minimum computation given a specific configuration. It consider values of 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_{base}, all possible values of $n_{nss} \le n_{base}$ are considered. For each combination of n_{base} and n_{nss} we generate a value of f_{comp}, and the sum of f_{comp} and the bound on communication will give us RT_{bound}. For each RT_{bound}, the robustness can be calculated using Eq. 29, and the maximum is the UB.

4.5 Simulation Results

4.5.1 Simulation Setup

The simulation setup is the same as in Section 3.4.1. The values for β_{max} and Δ_{max} are 200 and 150 milliseconds, respectively. These values for β_{max} and Δ_{max} were selected because they are realistic parameters for the QoS constraints.

4.5.2 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

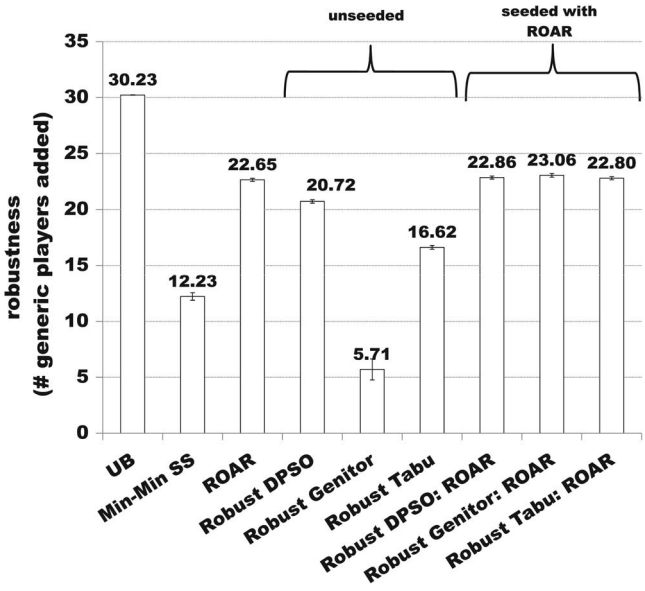


Fig. 20. Results for maximizing the robustness of the system against additional players joining the game. The computational parameters of the MS were set to $b = 0.03$ and $c = 0.01$, values are averaged over 100 scenarios, and the error bars show the 95% confidence intervals. The values for β_{max} and Δ_{max} are 200 and 150 milliseconds, respectively.

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_{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 ($\approx 1\%$) improvement, Tabu Search had a 0.15 ($\approx 1\%$) improvement, and Robust Genitor had a 0.41 ($\approx 2\%$) 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 $n_{seconday} = n_{nss} = 0$), 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

$$\beta_{max} = 2 \cdot Comp_{MS} = 2 \cdot b \cdot n_{main}^2. \quad (31)$$

$$so\,that = 2 \cdot 0.03 \cdot n_{main}^2. \quad (32)$$

$$n_{main} = \sqrt{100/0.03} \approx 58. \quad (33)$$

Thus only 58 users can be connected directly to the MS while still maintaining the QoS constraints. Therefore, the use of SSs 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).

5 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.

6 Conclusion

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 112 ms 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 (≈ 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.

Acknowledgment

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 17^{th} Heterogeneity in Computing Workshop, and in the 4^{th} International Conference on Foundations of Digital Games. The authors thank Richard Wallace, Paul Maxwell, and Samee Khan for their valuable contributions.

References

[1] A. Shaikh, S. Sahu, M.-C. Rosu, M. Shea, and D. Saha, "On demand platform for online games," *IBM Syst. J.*, vol. 45, no. 1, pp. 7–20, 2006.

[2] Y.-T. Lee and K.-T. Chen, "Is server consolidation beneficial to MMORPG? A case study of World of Warcraft," in *Proc. 2010 IEEE 3rd Int. Conf. Cloud Comput.*, 2010, pp. 435–442.

[3] MAG. *Zipper Interative* [Online]. Available: http://www.mag.com/gate.html, accessed on Mar. 2011.

[4] N. E. Baughman and B. N. Levine, "Cheat-proof playout for centralized and distributed online games," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM'01)*, Mar. 2001, pp. 104–113.

[5] P. Kabus, W. W. Terpstra, M. Cilia, and A. P. Buchmann, "Addressing cheating in distributed MMOGs," in *Proc. 4th ACM SIGCOMM Workshop Netw. Syst. Support Games*, 2005, pp. 1–6.

[6] M. Picone, S. Sebastio, S. Cagnoni, and M. Amoretti, "Peer-to-peer architecture for real-time strategy MMOGs with intelligent cheater detection," in *Proc. 3rd Int. ICST Conf. Simul. Tools Techn.*, 2010, pp. 8:1–8:8.

[7] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM'04)*, Mar. 2004, pp. 96–107.

[8] 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 *Proc. Adv. Comput. Vol. 63: Parallel Distrib. Pervasive Comput.*, 2005, pp. 91–128.

[9] L. Barbulescu, L. D. Whitley, and A. E. Howe, "Leap before you look: An effective strategy in an oversubscribed scheduling problem," in *Proc. 19th Nat. Conf. Artif. Intell.*, Jul. 2004, pp. 143–148.

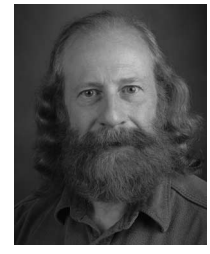
[10] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, Jun 2001.

[11] Y. C. Lee and A. Y. Zomaya, "Rescheduling for reliable job completion with the support of clouds," *Future Gener. Comput. Syst.*, vol. 26, no. 8, pp. 1192–1199, Oct. 2012.

[12] 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," *J. Parallel Distrib. Comput.*, vol. 68, no. 4, pp. 410–426, Apr. 2008.

[13] M. Snir and D. A. Bader, "A framework for measuring supercomputer productivity," *Int. J. High Perform. Comput. Appl.*, vol. 18, no. 4, pp. 417–432, 2004.
[14] M. Wu and W. Shu, "Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems," in *Proc. 9th IEEE Heterogeneous Comput. Workshop (HCW'00)*, Mar. 2000, pp. 375–385.
[15] Q. Zheng, B. Veeravalli, and C. K. Tham, "On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs," *IEEE Trans. Comput.*, vol. 58, no. 3, pp. 380–393, Mar. 2009.
[16] A. Dogan and F. Özguner, "Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems," *Cluster Comput.*, vol. 7, no. 2, pp. 177–190, Apr. 2004.
[17] G. J. Armitage, "An experimental estimation of latency sensitivity in multiplayer Quake 3," in *Proc. 11th IEEE Int. Conf. Netw. (ICON'03)*, Sep. 2003, pp. 137–141.
[18] T. Iimura, H. Hazeyama, and Y. Kadobayashi, "Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games," in *Proc. 3rd ACM SIGCOMM Workshop Netw. Syst. Support Games*, Aug. 2004, pp. 116–120.
[19] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *J. ACM*, vol. 24, no. 2, pp. 280–289, Apr. 1977.
[20] L. D. Briceno, H. J. Siegel, A. A. Maciejewski, and M. Oltikar, "Characterization of the iterative application of makespan heuristics on non-makespan machines in a heterogeneous parallel and distributed environment," *J. Supercomput.*, vol. 62, no. 1, pp. 461–485, Oct. 2012.
[21] J.-K. Kim, H. J. Siegel, A. A. Maciejewski, and R. Eigenmann, "Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1445–1457, Nov. 2008.
[22] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 59, no. 2, pp. 107–131, Nov. 1999.
[23] L. D. Briceno, B. Khemka, H. J. Siegel, A. A. Maciejewski, C. Groer, G. Koenig, G. Okonski, and S. Poole, "Time utility functions for modeling and evaluating resource allocations in a heterogeneous computing system," in *Proc. 20th Int. Heterogeneity Comput. Workshop (HCW'11)*, 2011, pp. 7–19.
[24] A. Hertz and D. de Werra, "The tabu search metaheurstic: How we used it," *Ann. Math. Artif. Intell.*, vol. 1, no. 1–4, pp. 111–121, Sep. 1990.
[25] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, Nov. 1995, pp. 1942–1948.
[26] J. Pugh and A. Martinoli, "Discrete multi-valued particle swarm optimization," in *Proc. IEEE Swarm Intell. Symp.*, May 2006, pp. 103–110.
[27] D. Whitley, "The GENITOR algorithm and selective pressure: Why rank based allocation of reproductive trials is best," in *Proc. 3rd Int. Conf. Genetic Algorithms*, Jun. 1989, pp. 116–121.
[28] L. D. Briceno, H. J. Siegel, A. A. Maciejewski, M. Oltikar, J. Brateman, J. White, J. Martin, and K. Knapp, "Heuristics for robust resource allocation of satellite weather data processing on a heterogeneous parallel system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 11, pp. 1780–1787, Nov. 2011.
[29] L. Bölöni and D. C. Marinescu, "Robust scheduling of metaprograms," *J. Schedul.*, vol. 5, no. 5, pp. 395–412, Sep. 2002.
[30] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 7, pp. 630–641, Jul. 2004.
[31] S. Ali, A. A. Maciejewski, and H. J. Siegel, "Perspectives on robust resource allocation for heterogeneous parallel systems," in *Handbook of Parallel Computing: Models, Algorithms, and Applications*, S. Rajasekaran and J. Reif, Eds. Boca Raton, FL, USA: Chapman & Hall/CRC Press, 2008, pp. 41-1–41-30.
[32] G. Deen, M. Hammer, J. Bethencourt, I. Eiron, J. Thomas, and J. H. Kaufman, "Running quake II on a grid," *IBM Syst. J.*, vol. 45, no. 1, pp. 21–44, 2006.
[33] A. Chen and R. Muntz, "Peer clustering: A hybrid approach to distributed virtual environments," in *Proc. 5th ACM SIGCOMM Workshop Netw. Syst. Support Games*, 2006, pp. 1–9.
[34] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: A distributed architecture for online multiplayer games," in *Proc. 3rd Symp. Netw. Syst. Des. Implementation*, 2006, pp. 155–168.
[35] J. Gilmore and H. Engelbrecht, "A survey of state persistency in peer-to-peer massively multiplayer online games," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 5, pp. 818–834, May 2012.
[36] J. Luo and H. Chang, "A scalable architecture for massive multiplayer online games using peer-to-peer overlay," in *Proc. 12th Int. Conf. Adv. Commun. Technol. (ICACT)*, 2010, pp. 604–608.
[37] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin, "An efficient synchronization mechanism for mirrored game architectures," *Multimedia Tools Appl.*, vol. 23, no. 1, pp. 7–30, 2004.
[38] J. Lee, "Considerations for movement and physics in MMP games," in *Massively Multiplayer Game Development*. Hingham, MA, USA: Charles River Media, 2003, pp. 275–289.
[39] G. Wang and K. Wang, "An efficient hybrid P2P MMOG cloud architecture for dynamic load management," in *Proc. 2012 Int. Conf. Inf. Netw. (ICOIN)*, 2012, pp. 199–204.
[40] K.-W. Lee, B.-J. Ko, and S. Calo, "Adaptive server selection for large scale interactive online games," *Comput. Netw.*, vol. 49, no. 1, pp. 84–102, Sep. 2005.

Luis Diego Briceño received the BS degree in electrical and electronic engineering from the university of Costa Rica, San jose, and the PhD degree in electrical and computer engineering from Colorado State University, Fort Collins, USA. His research interests include heterogeneous parallel and distributed computing.



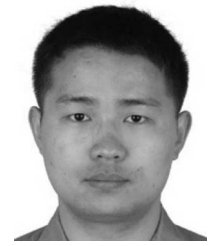
Howard Jay Siegel received two BS degrees from the Massachusetts Institute of Technology (MIT), Cambridge, and the MA, MSE, and PhD degrees from Princeton University, New Jersey. He was appointed the Abell Endowed chair distinguished professor of electrical and computer engineering at Colorado State University (CSU), Fort Collins, CO, in 2001, where he is also a professor of computer science. From 1976 to 2001, he was a professor with Purdue University. 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 is a Fellow of the ACM. He has co-authored over 400 technical papers. 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 BS, MS, and PhD degrees in electrical engineering, from Ohio State University, Columbus, in 1982, 1984, and 1987, respectively. From 1988 to 2001, he was a professor of electrical and computer engineering at Purdue University, West Lafayette, IN. In 2001, he joined Colorado State University, Fort Collins, where he is currently the head of the Department of Electrical and Computer Engineering.

Mohammad Nayeem Teli received the PhD degree from Computer Science Department, Colorado State University. His research interests include computer vision, machine learning, big data analysis, face detection, face recognition, biometrics, correlation filters, and EEG analysis.

Brad Lock received the BS degree in electrical and computer engineering and the MS degree in computer engineering from Colorado State University in 2001 and 2011, respectively. He currently works for Intel, Santa Clara, CA.

Fadi Ibrahim Ali Wedyan received the PhD degree in computer science from Colorado State University in 2011. He is currently an assistant professor at The Hashemite University, Zarqa, Jordan.



Ye Hong received the bachelor and master degrees in computer science from Tsinghua University, Beijing, China, in 2002 and 2006, respectively. His research interests include parallel and distributed computing, heterogeneous computing, robust computer systems, and performance evaluation.

Charles Panaccione is pursuing the PhD degree in computer science at Colorado State University. He has worked for the National Center for Atmospheric Research and for Pearson.

Chen Zhang is pursuing the master's degree in computer science at Colorado State University.