# Resource Allocation in a Client/Server Hybrid Network for Virtual World Environments

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

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

## Abstract

*The creation of a virtual world environment (VWE) has significant costs, such as maintenance of server rooms, server administration, and customer service. The initial development cost is not the only factor that needs to be considered; factors such as the popularity of a VWE and unexpected technical problems during and after the launch can affect the final cost and success of a VWE. The capacity of servers in a client/server VWE 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 VWE to support an increase in users. In this study, we develop and implement five static heuristics to implement a secondary server scheme that reduces the time taken to compute the state of the VWE. 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. A lower bound of the performance is derived to evaluate the results of the heuristics.*

## 1 Introduction

Heterogeneous distributed and parallel systems may operate in an environment where certain system performance features degrade due to unpredictable circumstances, such as sudden machine failures, higher than expected load, or inaccuracies in the estimated system parameters. The environment considered in this research is a virtual world environment (VWE) (e.g., a massive multiplayer online gaming (MMOG) environment). In a VWE, each user controls an *avatar* (an image that represents and is manipulated by a user) in a virtual world and interacts with other users. In general, most VWEs use a client/server architecture to control the virtual game world. The client/server 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. In addition to the initial development cost, other factors such as the popularity of a VWE, and unexpected technical problems during and after the launch can also affect the final cost and success of a VWE [24].

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

The performance of the heterogeneous system used to simulate the game world must not degrade beyond acceptable parameters even if the VWE is oversubscribed. The heuristics must convert users to $SS$s and assign the remaining users to an existing $SS$ or the $MS$. This problem is similar to the assignment of tasks to machines (e.g., [3, 6, 25, 27]) with $SS$ and the $MS$ as machines and the remaining users as tasks.

A session in the VWE is assumed to last for an extended period of time, with a small break between ses-

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

sions [20]. During these sessions it is assumed that no users enter or leave the game (i.e., the users in a gaming session do not change). These simplifying assumptions make a static resource allocation heuristic viable [1].

In this study, five heuristics for determining the number of $SS$s, which users are converted to $SS$s, and how users are distributed among the $SS$s and the $MS$ are proposed. The heuristics were used to minimize the time it takes to process the world update requests from all users. A mathematical lower bound is derived to evaluate the performance of the heuristics in this VWE.

This paper is organized as follows. Section 2 provides the problem statement and describes the performance metrics. In Section 3, we focus on the five proposed heuristics. Section 4 discusses the lower bound on our performance metrics. Our results for the five heuristics are shown in Section 5. In Section 6, we provide the related work and in Section 7 we present our conclusions.

## 2 Problem Statement

### 2.1 Problem Description

In this study, we will consider an MMOG environment where the performance of a user is sensitive to latency [2]. The purpose of this research is to maintain an acceptable system performance (despite the $MS$ used to maintain the MMOG environment being oversubscribed) without increasing the processing power of the $MS$. The proposed solution is to convert users to $SS$s that assist the $MS$ in computation. In the client/server solution shown in Figure 1(a), all users connect to the $MS$, therefore the $MS$ is the only machine performing computation. In the $SS$ solution shown in Figure 1(b), the $MS$ and $SS$s perform computation and the $MS$ resolves conflicts among users and $SS$s connected to it.

The allocation of users as $SS$s has similar security requirements as distributed servers and peer-to-peer based MMOG systems. The issue of security in distributed servers and peer-to-peer MMOG environments are studied in [4] and will not be discussed here because we consider it to be a separate research problem.

The following simplifying assumptions are made about the communication model in this system. The communication time from node (user computer, $SS$, or $MS$) $A$ to node $B$ is the same as the communication time from node $B$ to node $A$; however, the communication time between different pairs of nodes will vary. The communication times among the users, $SS$s, and the $MS$ do not change during a session. These times are independent of the number of users connected to the $SS$ or $MS$. These simplifying assumptions are used to reduce the complexity of the simulations.

Without loss of generality, the level of activity in the VWE of all the users is considered identical (i.e., the frequency of interaction with the VWE is the same for all players); therefore, 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 need to consider how the computation grows with the increase in the number of users. Let $\underline{n_\alpha}$ be the number of users connected to secondary server $\underline{\alpha}$ ($\underline{SS_\alpha}$) and $\underline{\mu_\alpha}$ be a constant for $\underline{SS_\alpha}$ (this constant represents the heterogeneity in the computing power of the users' computers, and each user has a different constant). In [15], latency in a MMOG environment shows a "weak exponential" increase with an increase in players. This study uses these observations to create a model for the MMOG environment. The computation time for an $SS$ ($\underline{Comp_\alpha}$) can be calculated as:

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

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

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

### 2.2 Objective Function and Performance Metric

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

$$\begin{aligned} RT_x &= Comm(U_x, MS) + Comp_{MS} \\ &+ Comm(MS, U_x); \\ RT_x &= 2 \cdot Comm(U_x, MS) + Comp_{MS}. \quad (3) \end{aligned}$$

If a user is connected to an $SS_\alpha$ then the equation is:

$$\begin{aligned} RT_x &= Comm(U_x, SS_\alpha) + Comp_\alpha \\ &+ Comm(SS_\alpha, MS) + Comp_{MS} \\ &+ Comm(MS, SS_\alpha) + Comm(SS_\alpha, U_x); \\ RT_x &= 2 \cdot Comm(U_x, SS_\alpha) + Comp_\alpha \\ &+ 2 \cdot Comm(SS_\alpha, MS) + Comp_{MS}. \quad (4) \end{aligned}$$

If $U_x$ is $SS_\alpha$ then Equation 4 is used with $Comm(U_x, SS_\alpha) = Comm(SS_\alpha, U_x) = 0$.

The performance metric of a resource allocation in this environment is:

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

The goal of the heuristics is to minimize $RT_{max}$. The value of $RT_{max}$ will be influenced by three elements:

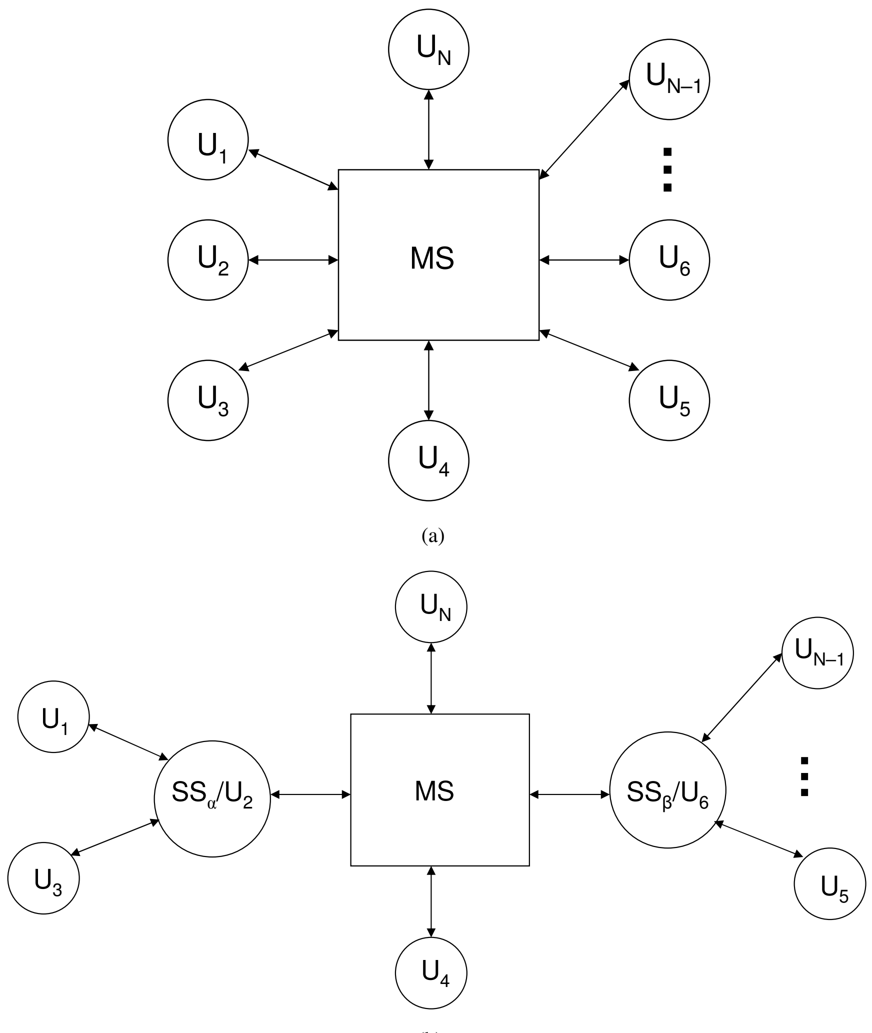

**Figure 1. (a) Client/server architecture versus (b) Secondary Server architecture.**

the number of $SS$s, which users' computers are converted to $SS$s, and the assignment of *non-SS* users to an $SS$ or to the $MS$. If a user is connected to an $SS$ then this user cannot have any users connected to it.

## 3 Resource Allocation Heuristics

### 3.1 Overview

All heuristics were limited to a maximum execution time of 10 minutes. Five heuristics for determining an allocation of resources are presented in this section. 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.

### 3.2 Dual Iterative Minimization

In Dual Iterative Minimization (DIM), a solution is represented in the form of an vector whose $i^{th}$ element indicates the way user $i$ is connected to the $MS$, either directly or the $SS$ to which it is connected.. A potential host ($\underline{PH}$) is a user that is not connected to the $MS$ through an $SS$; i.e., either it is attached directly to the MS (possibly as an SS) or is unassigned at this point. This heuristic considers assigning an unassigned user to all $PH$s or the $MS$ and picks the $PH$ that provides the minimum $RT_x$. If this $PH$ is not already an $SS$ then it is converted to one. The pseudo-code is shown in Figure 2. A post-optimization procedure is run on the solution from the DIM heuristic. This post-optimization is shown in Figure 3.

### 3.3 Tabu Search

Tabu Search [12] enhances the performance of a local search method by storing the previously visited areas in

```
(1) Mark all users as unassigned.
(2) For each unassigned user ($u$) in a fixed arbitrary order.
    (a) Define $minRT$ 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 to the $MS$
        through $PH$ ($RT_{u \to PH \to MS}$) .
        (i) If $RT_{u \to PH \to MS}$ is less than $minRT$ then attach $u$ to $PH$, and convert $PH$ to an $SS$
            if it is not already one.
        (ii) Else, attach $u$ directly to the $MS$.
    (c) Mark user $u$ as assigned.
(3) Output final solution.
```

**Figure 2. DIM pseudo code.**

the search space using a tabu list so they are not revisited. To make the size of the tabu list reasonable, only the last $n$ (set empirically to 20) visited neighborhoods are saved [12]. The solution was represented by vectors as in the DIM heuristic.

Local moves (or short hops) explore the neighborhood of the current solution, searching for the local minimum. All the moves that we use in the Tabu Search are considered greedy in the sense that we accept a neighboring solution 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 solution with a new set of $SS$s that is not in the tabu list. The procedure for Tabu Search is shown in Figure 4 and the procedure for the short hops is shown in Figure 5.

## 3.4 Minimum Return Time (MRT)

The Minimum Return Time (MRT) heuristic is based on the Minimum Completion Time (MCT) heuristic [14, 22]. This heuristic is used by the discrete particle swarm optimization heuristic to generate a solution from a set of users that are connected directly to the $MS$.

Directly connected users (DCUs) are users that are connected directly to the $MS$. In MRT, the users that are not DCUs are assigned in a fixed arbitrary order to the DCU that gives each user its minimum RT. The pseudocode for implementing the MRT is shown in Figure 6.

## 3.5 Discrete Particle Swarm Optimization

Discrete particle swarm optimization (DPSO) is based on the original particle swarm optimization [18]. In [18], the authors developed the algorithm based on the flocking behavior of animals such as birds or fish. Instead of organisms, they called the objects flying through the air "particles." The DPSO implemented in this study is based on the discrete version of particle swarm optimization in [23].

For the DPSO, there are $P$ (determined empirically) particles, and each particle has $N$ dimensions, where each dimension corresponds to a user. For the $i^{th}$ particle, the value in the $j^{th}$ dimension ($X_{ij}$) indicates whether the $j^{th}$ user is a DCU or a non-DCU (0 if the user is not a DCU and 1 if the user is a DCU). A particle has a set of DCUs that will be used by the MRT heuristic to determine the resource allocation. This representation will cause the search space to be a hypercube.

The velocity of a particle is used to explore other areas of the hypercube. $V_{ij}$ is the velocity of the $i^{th}$ particle in the $j^{th}$ dimension ($V_{ij} \in (-\infty, \infty)$). To convert this velocity to a position a sigmoid function ($s(V_{ij}) = 1/\left(1 + e^{-V_{ij}}\right)$) is used (this function returns a value from 0 to 1). At each iteration of the DPSO, the position of particle $i$ may be changed for all $i$; however, the best personal solution ($P_i^*$), is updated only when particle $i$ finds a better solution. Thus, each particle $i$ has a best personal solution ($P_i^*$, an N dimensional vector). The swarm has a best global solution ($G^*$, an N dimensional vector). $G^*$ is equal to the $P_i^*$ with the lowest $RT_{max}$ over all $i$. The particles will be attracted to their best personal solution and to the best global solution. This attraction allows exploration around the best known solutions. The notation $P_{ij}^*$ represents the value (0 or 1) of the $j^{th}$ user for the best personal solution of particle $i$, and $G_j^*$ represents the value of the $j^{th}$ user for the best global solution.

We define $w$ as the inertia coefficient that slows the velocity of the particle over time, $p_w$ as the attraction to the best personal solution, and $g_w$ as the attraction of the particle to the best global solution. These three constants are determined empirically. The particles are allowed to move for $iter_{max}$ iterations. The procedure is shown in Figure 7.

```
(1) $RT_{best}$ is equal to the value of $RT_{max}$ of the solution from the DIM heuristic.
(2) For each user ($U_x$) connected to an $SS$.
    (a) Connect $U_x$ to the $MS$ and connect the user with $RT_{max}$ to $U_x$.
    (b) Find the $RT_{max}$ of this configuration.
    (c) If $RT_{max} < RT_{best}$ then save this solution as the best solution.
(2) For each user ($U_x$) connected to an $SS$.
    (a) Swap $U_x$ with the user with $RT_{max}$.
    (b) Find the $RT_{max}$ of this configuration.
    (c) If $RT_{max} < RT_{best}$ then save this solution as the best solution otherwise undo the swap.
(3) For each $SS$ ($SS_x$).
    (a) Connect the user with $RT_{max}$ to $SS_x$.
    (b) Find the $RT_{max}$ of this configuration.
    (c) If $RT_{max} < RT_{best}$ then save this solution as the best solution otherwise undo the change.
(4) Output best solution.
```

**Figure 3. DIM post-optimization pseudo code.**

```
(1) Start from a complete resource allocation (generated randomly).
(2) Perform short hops (to do local search in a neighborhood) for a maximum number of iterations ($iter_{max}$) experimentally set to 300.
(3) The set of secondary servers that represent the neighborhood of the final solution from (2) is added as an entry to the tabu list of size 20 (oldest entry in the tabu list is replaced).
(4) Perform a long hop by randomly generating a complete resource allocation that has a set of secondary servers that is not identical to a set in the tabu list.
(5) Steps (2) to (4) are executed until the available time for resource allocation expires (the time limit was 10 minutes).
(6) The best known resource allocation is output as the solution (once the available time expires).
```

**Figure 4. Procedure for Tabu Search.**

## 3.6 Min-Min RT

The Min-Min RT heuristic is based on the concept of Min-Min heuristic [14]. The Min-Min heuristic is widely used in the area of resource allocation [7, 10, 11, 14, 16, 17, 19, 27]. It takes a given set of DCUs and assigns remaining users to a DCU or to the $MS$. The procedure to implement the Min-Min RT is shown in Figure 8. This heuristic is used by the RT Iterative Minimization and the Genetic Algorithm.

## 3.7 Genetic Algorithm

The genetic algorithm (GA) is a heuristic based on the process of evolution [13]. It uses a chromosome to represent a solution; this solution can be "mixed" with other solutions and "mutated" to produce new solutions.

In this study, the chromosome represents which players are DCUs. To convert this representation to a resource allocation, we use the Min-Min RT heuristic. The procedure for the chosen variation of a GA is shown in Figure 9. The selection process for this heuristic is done using the linear bias function [26].

## 3.8 RT Iterative Minimization

This greedy heuristic consists of two phases. In phase 1, we iteratively adjust the number of DCUs (by adding or removing) to minimize $RT_{max}$. In phase 2, we swap the DCUs with users to minimize $RT_{max}$. This heuristic also uses the Min-Min RT heuristic to convert a set of DCUs to a resource allocation. The procedure for RT

```
(1) Set the number of executed short hops $sh$ to 0
(2) Do:
    (a) Increase $sh$ by 1 and determine the user with $RT_{max}$ (denoted $\underline{U_{RTmax}}$). If $U_{RTmax}$ is the only one connected to a given $SS$ then go to (3) (to avoid changes in the neighborhood).
    (b) For each server $\underline{S_\alpha}$ ($MS$ and all $SS$s) in a fixed arbitrary order, if assigning $U_{RTmax}$ to $S_\alpha$ decreases the $RT_{max}$, then assign $U_{RTmax}$ to $S_\alpha$.
(3) While $sh < iter_{max}$ ( = 300) and a better assignment is found in (2b) then go to (2).
(4) Do:
    (a) Increase $sh$ by 1 and randomly select a user connected to the $MS$ ($U_{MS}$).
    (b) For each secondary server $\underline{SS_\alpha}$ (only $SS$s) in a fixed arbitrary order, if assigning $U_{MS}$ to $SS_\alpha$ decreases the $RT_{max}$, then assign $U_{MS}$ to $SS_\alpha$.
(5) While $sh < iter_{max}$ ( = 300) and a better assignment is found in (4b) then go to (4).
(6) Do:
    (a) Increase $sh$ by one and determine $U_{RTmax}$.
    (b) For each server $S_\alpha$ ($MS$ and all $SS$s) in a fixed arbitrary order, if assigning $U_{RTmax}$ to $S_\alpha$ decreases the $RT_{max}$ then assign $U_{RTmax}$ to $S_\alpha$. If the assignment of $U_{RTmax}$ to $S_\alpha$ leaves an $SS$ without any users connected it, then it is removed from the set of secondary servers.
(7) While $sh < iter_{max}$ ( = 300) and a better assignment is found in (6b) then go to (6).
```

**Figure 5. Short hop moves for Tabu Search.**

```
(1) A list of users that are not DCUs is generated in a fixed arbitrary order.
(2) The first user in this list is assigned to the DCU that gives it the minimum $RT$.
(3) The user assigned in (2) is removed from the list.
(4) The computation time for the DCU where the user was assigned is updated.
(5) Steps (2) through (4) are repeated until all users have been assigned.
```

**Figure 6. Procedure for MRT.**

Iterative Minimization is shown in Figures 10 (Phase 1) and 11 (Phase 2).

## 4 Lower Bound

The primary purpose of deriving a mathematical lower bound was to evaluate the experimental results of our proposed heuristics. The bound has two components that can be calculated independently. The first component finds the minimum possible *computation time* of the $MS$ and $SS$s (by performing an exhaustive search of all possible *computation times*). This component has three assumptions: (a) communication times are ignored, (b) all users have the same computational constant ($\rightarrow \mu_{min} = \min_{\forall U_x} \mu_x$), and (c) users connected to $SS$s are evenly distributed among $SS$s. Component (a) reduces the complexity because heterogeneity in communication is removed from the problem, (b) removes the heterogeneity in computing power of the $SS$s, and (c) minimizes the maximum computation time among $SS$s.

The second component is the lower bound on the communication time. This bound is calculated by finding the minimum time each user requires to connect to the $MS$ (either connected directly to the $MS$ or through another user), and then finding the maximum of these times. For the calculation of this second component, the computation time is ignored.

Let $\underline{n}$ be the total number of users that are connected to the $MS$, $n_{nss}$ plus $n_{main}$. The lower bound ($\underline{\text{LB}}$) is given as:

```
(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) Determine $RT_{max}$ using the MRT heuristic.
(3) Initialize the global and best personal solutions ($P_i^*$s) using the particles in (1), and set the best global solution to
    the best $P_i^*$ over all $i$.
(4) For ($i = 1$ to $P$) do
    (a) For ($j = 1$ to $N$) do
        (i) Generate 3 different random variables ($R_1$,$R_2$, and $R_3$) each with a uniform distribution $[0, 1] \in \Re$
        (ii) $V_{ij} = w \cdot V_{ij} + p_w \cdot R_1 \cdot (P_{ij}^* - X_{ij}) + g_w \cdot R_2 \cdot (G_j^* - X_{ij})$
        (iii) If $V_{ij}$ is less than the minimum allowed velocity ($V_{min}$) then $V_{ij} = V_{min}$.
        (iv) If $V_{ij}$ is greater than the maximum allowed velocity ($V_{max}$) then $V_{ij} = V_{max}$.
        (v) If ($R_3 < S(V_{ij})$) then $X_{ij} = 1$, otherwise $X_{ij} = 0$.
    (b) Determine $RT_{max}$ using the MRT heuristic.
(5) Set the best personal solutions ($P_i^*$) using the particles in (4),and set the best global solution to the best $P_i^*$ over
    all $i$.
(6) Repeat (4) until the number of iterations is equal to $iter_{max}$.
```

**Figure 7. Procedure for DPSO.**

```
(1) Given a predetermined set of DCUs, all users that are not in the set of DCUs are marked as unassigned.
(2) For each unassigned user, the server (main or DCU) that gives the minimum RT is determined (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.
(5) Steps (2) through (4) are repeated until all tasks are assigned.
```

**Figure 8. Procedure for Min-Min RT.**

```
(1) An initial population of 100 (determined empirically) chromosomes is generated.
(2) For each chromosome, the $RT_{max}$ is calculated based on the full resource allocation
    determined with the Min-Min RT heuristic.
(3) While there are less than 1000 iterations without improvement in the best solution
    or 10 minutes have not elapsed.
    (a) 50 pairs of parents are selected using the linear bias function.
    (b) From the 50 pairs of parents, 100 offspring are generated using 2 point crossover.
    (c) For each offspring there is a 1% probability (determined empirically) of mutating
        each field in the chromosome.
    (d) A new population is created with the offspring. If the elite chromosome (the best chromosome found so far)
        is not in the new population then remove the worst offspring and insert the elite chromosome in its place.
(4) The output is the elite chromosome.
```

**Figure 9. Procedure for the GA.**

**Phase 1:**

(1) Randomly pick $k$ (a random number between 0 to $N$) users to form the initial set of DCUs ($\underline{Set_{SS}}$), with $\rho_{SS} \in Set_{SS}$.

(2) Create a full resource allocation using Min-Min RT heuristic with $Set_{SS}$.

(3) Find the user with $RT_{max}$. If the user is in $Set_{SS}$ with no players connected to it then remove it from $Set_{SS}$ and go to (2).

(4) Set $\underline{RT_{ref}}$ = $RT_{max}$.

(5) For each $\rho_{SS} \in Set_{SS}$. Find the $RT_{max}$ using the Min-Min RT heuristic with $Set_{SS} - \{\rho_{SS}\}$. This will create $|Set_{SS}|$ values of $RT_{max}$ and for each a corresponding set of secondary servers.

(6) Find the smallest $RT_{max}$ value and the set of DCUs that gives the smallest $RT_{max}$ from 5, and store them as $\underline{RT_{remove}}$ and $\underline{Set_{remove}}$, respectively.

(7) For each non-$SS$ user ($\underline{\rho_{nonSS}} \notin Set_{SS}$). Find the $RT_{max}$ using the Min-Min RT heuristic with $Set_{SS} \cup \{\rho_{nonSS}\}$. This will create $N - |Set_{SS}|$ values of $RT_{max}$ and for each a corresponding set of secondary servers.

(8) Find the smallest $RT_{max}$ value and the set of $SS$s that gives the smallest $RT_{max}$ from 7, and store them as $\underline{RT_{add}}$ and $\underline{Set_{add}}$, respectively.

(9) If $RT_{remove} < RT_{add}$ and $RT_{remove} < RT_{ref}$, then $Set_{SS} = Set_{remove}$.

(10) If $RT_{add} \leq RT_{remove}$ and $RT_{add} < RT_{ref}$, then $Set_{SS} = Set_{add}$.

note If $RT_{add} > RT_{ref}$ and $RT_{remove} > RT_{ref}$, then no improvement was found.

(11) Steps (3) through (10) are repeated until 1000 iterations have passed or no improvement is found.

**Figure 10. RT Iterative Minimization: Phase 1.**

**Phase 2:**

(1) Determine the set of DCUs in the resource allocation (denoted $\underline{Set_{SS}}$) from the resource allocation produced in Phase 1.

(2) Create a full resource allocation using the Min-Min RT heuristic with $Set_{SS}$.

(3) Find the user with $RT_{max}$ (denoted $\underline{U_{RTmax}}$). If $U_{RTmax}$ is in $Set_{SS}$ with no players connected to it then remove it from $Set_{SS}$ and go to (2).

(4) Set $\underline{RT_{ref}}$ to $RT_{max}$.

(5) If $U_{RTmax}$ is connected to a secondary server ($\underline{SS_{max}}$), then

   (a) For each $\underline{\rho_{nonSS} \notin Set_{SS}}$, find the $RT_{max}$ using the Min-Min RT heuristic with $Set_{SS} \cup \{\rho_{nonSS}\} - \{SS_{max}\}$.
   This will create $N - |Set_{SS}|$ values of $RT_{max}$ and for each a corresponding set of secondary servers.

   (b) Find the smallest $RT_{max}$ value and the set of DCUs that gives the smallest $RT_{max}$ from 5(a), and store them as $\underline{RT_{swap}}$ and $\underline{Set_{swap}}$, respectively.

   (c) If $RT_{swap} < RT_{ref}$, then $Set_{SS} = Set_{swap}$; else, no improvement is obtained.

(6) If $U_{RTmax}$ is connected to the $MS$, then

   (a) For each $\rho_{nonSS}$, find the $RT_{max}$ if $\rho_{nonSS}$ is converted to an $SS$ and $U_{RTmax}$is attached to it. This will create $N - |Set_{SS}|$ values of $RT_{max}$ and for each a corresponding $\rho_{nonSS}$.

   (b) Find the smallest $RT_{max}$ value and the $\rho_{nonSS}$ that gives the smallest $RT_{max}$ (now called $\underline{\rho_{toswap}}$) from 6(a).

   (c) For each $\rho_{SS}$, find the $RT_{max}$ using the Min-Min RT heuristic with $Set_{SS} \cup \{\rho_{toswap}\} - \{\rho_{SS}\}$.
   This will create $|Set_{SS}|$ values of $RT_{max}$ and for each a corresponding set of secondary servers.

   (d) Find the smallest $RT_{max}$ value and the set of DCUs that gives the smallest $RT_{max}$ from 6(c), and store them as $\underline{RT_{SSswap}}$ and $\underline{Set_{SSswap}}$, respectively.

   (e) If $RT_{SSswap} < RT_{ref}$, then $Set_{SS} = Set_{SSswap}$; else, no improvement is obtained.

(7) Steps (4) through (6) are repeated until 1000 iterations have passed or no improvement is found.

(8) To get the final solution, use Min-Min RT heuristic with $Set_{SS}$.

**Figure 11. RT Iterative Minimization: Phase 2.**

$$f_{comp}(n, n_{nss}) = \mu_{min} \cdot \left\lceil \frac{N-n}{n_{nss}} \right\rceil^2 + c \cdot (N-n) + b \cdot n^2 \; ; \quad (6)$$

$$f_{comm}(U_x, U_y) = 2 \cdot Comm(U_x, U_y) + 2 \cdot Comm(U_y, MS) \; ; \quad (7)$$

The case where $U_x = U_y$ is considered to account for the case when $U_x$ is connected to the $MS$.

$$LB = \min_{1 \le n \le N} \left( \min_{0 \le n_{nss} \le n} (f_{comp}(n, n_{nss})) \right) + \max_{U_x \in all\ users} \left( \min_{U_y \in all\ users} (f_{comm}(U_x, U_y)) \right) . \quad (8)$$

*Proof.* The proof will be divided into two parts. The first part will be to prove that the computational bound is minimum and the second part will be to prove the communicational minimum.

The first part of the bound does an exhaustive evaluation of all possible configurations for $n_{nss}$ and $n$. This will give us all the possible computations times. It will move $n$ 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 $n = N$ (i.e., the only scenario where we do not have $SS$s 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 maximum among these times. This method does an exhaustive search of the possible communication times (through an $SS$ or directly connected to the $MS$). Therefore, the user with the maximum communication time cannot have a smaller communication time independently of the configuration.

□

## 5 Results

The simulation had 200 users interacting in the MMOG environment. The constants for these simulations were $b$ = 0.03 and $c$ = 0.06 (the values for these constants were set to approximate realistic values for latencies in an MMOG environment). For this study, 100 scenarios were created with varying communication times and $\mu_\alpha$ for each user. Because the $RT_{max}$ of the optimal solution can be intractable to compute, a lower bound was used to compare the performance of the results.

Figure 12 shows the results averaged over the 100 scenarios. All the results are shown with a 95% confidence interval. From Figure 12, we can observe that the Tabu Search had the worst performance compared to all of the other heuristics. One problem with the Tabu Search is that the long hop can result in a very poor solution. This was improved by seeding the result of the DIM into the Tabu Search. When the DIM is used in the Tabu Search (Seeded Tabu), it outperforms all other heuristics. The DIM, DSPO (1500 iterations using a population of 600 particles with $p_w$ and $g_w$ equal to 2, and $w$ equal to 1), and GA heuristics performed relatively well compared to all the other heuristics (approximately 20 time units above the LB). The RT for the Seeded Tabu was about 13 time units more than the mathematical lower bound.

If all users were connected to the $MS$ then the $RT_{max}$ would be approximately 1200 time units ($200^2 \cdot b + \max_{\forall U_x} 2 \cdot Comm(U_x, MS) \approx$ 1200). The use of the secondary server based approach in our simulations leads to an improvement of an order of magnitude (i.e., 83 time units versus 1200 time units).

## 6 Related Work

Various MMOG architectures are reported in the literature (e.g. client/server [9], peer-to-peer [5, 15, 20], mirrored server [8]). Each architecture has its own advantages, for example the client/server and mirrored server allows the company that develops the MMOG environment to maintain a 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 assert a consistent VWE. The advantage of using a peer-to-peer architecture is that there is no single point of failure and the MMOG environment can be maintained without a significant monetary cost.

Maintaining a seamless interactive experience for the users is an important factor in MMOG, because an increase in latency within the system can lead to deterioration in the gaming experience [2, 9]. In [15], the authors show that the latency follows a "weak exponential increase" as the number of users grows in system. Our study focuses on latency as a critical performance parameter that must be maintained and uses the results in [15] to model the relationship between latency and the number of users.

This study proposes a hybrid client/server architecture to combine the best elements of both the centralized client/server and peer-to-peer architectures. Our work is most similar to [21]. In [21], a distributed system uses intermediate servers (analogous to our definition of secondary servers) to reduce the communication latency to the central server. The main difference between our

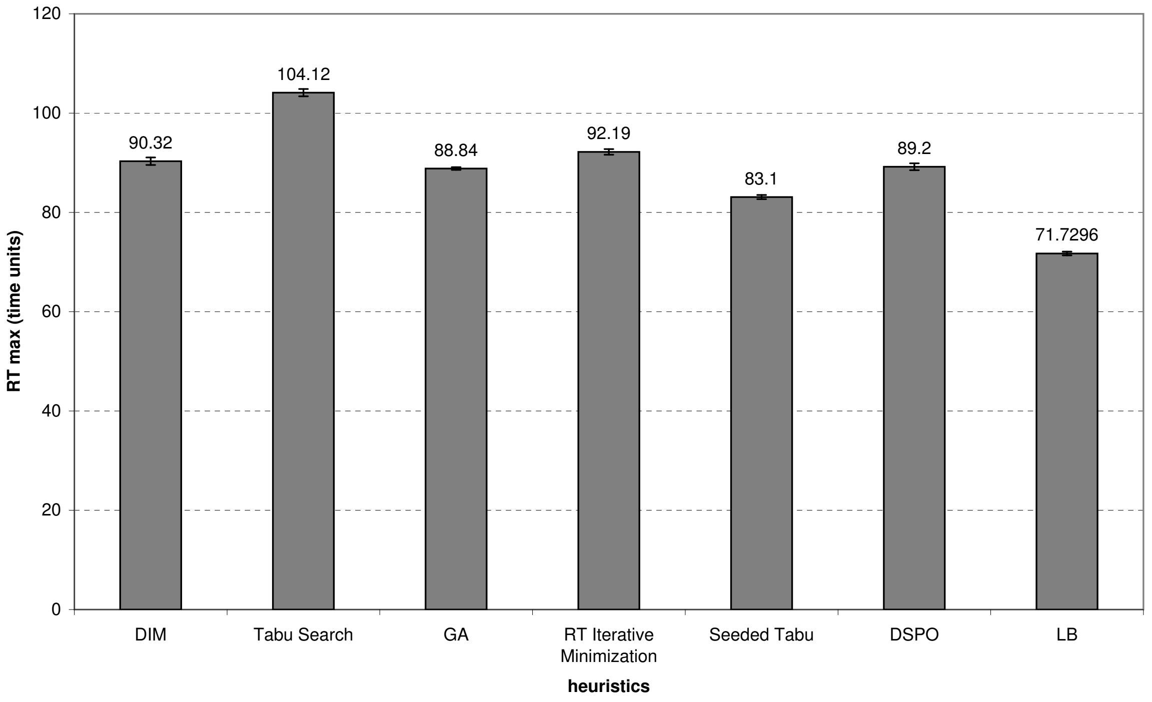

**Figure 12. Results for heuristics with the computational parameters of the $MS$ set to: $b$ = 0.03 and $c$ = 0.06. The values are averaged over 100 scenarios, and the error bars show the 95% confidence intervals**

studies is that in [21] the intermediate servers are predefined and do not participate as users in the VWE. Our work is different from [8, 9] because it considers converting users to secondary servers. This work is also different to [5, 15, 20] because it has a "non-peer" centralized server. 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.

## 7 Conclusions

This study evaluated an oversubscribed VWE that employs a group of users to do parts of the calculations required to operate a VWE. The main objective of this study was to develop heuristics to evaluate the performance of a secondary server based VWE.

In this study, we showed that the secondary servers add capacity to the main server. This extra capacity can be used to maintain an acceptable performance even when the VWE is oversubscribed. The heuristics were able to improve the performance the $MS$ would be able to have if all users were directly connected to the it by more than an order of magnitude. The LB was calculated to compare the performance of the heuristics to a mathematical bound on performance. The LB indicates that the solutions for the GA and Seeded Tabu Search are close to the LB (by about 17 and 12 time units respectively). It would be interesting to seed the GA with the solution from the DIM heuristic to see if the solution of the GA improves.

A possible expansion of this study is to improve the model by removing the simplifying assumptions (e.g., the constant communication times, same time to the a server and back). This study also assumed that users are willing to become an $SS$. This problem could also be reformulated using game theory to consider the behavior of selfish and/or cooperative users.

*Acknowledgments: The authors would like to thank Samee Khan for his valuable contributions.*

## References

[1] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao. Characterizing resource allocation heuristics for heterogeneous computing systems. In *Advances in Computers Volume 63: Parallel, Distributed, and Pervasive Computing*, pages 91–128, 2005.

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

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

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

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

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

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

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

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

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

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

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

[13] J. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, First Edition*. The University of Michigan, 1975.

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

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

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

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

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

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

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

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

[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. *Journal of Parallel and Distributed Computing*, 59(2):107–121, Nov. 1999.

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

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

[25] V. Shestak, E. K. P. Chong, H. J. Siegel, A. A. Maciejewski, L. Benmohamed, I.-J. Wang, and R. Daley. A hybrid branch-and-bound and evolutionary approach for allocating strings of applications to heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*. accepted, to appear.

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

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

## Biographies

**Luis D. 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. He is the Director of the CSU Information Science and Technology Center (ISTeC), a university-wide organization for promoting, facilitating, and enhancing CSU's research, education, and outreach activities pertaining to the design and innovative application of computer, communication, and information systems. 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 a B.S. degree in electrical engineering and a B.S. degree in management from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He has co-authored over 350 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 was Program Chair/Co-Chair of three major international conferences, General Chair/Co-Chair of seven international conferences, and Chair/Co-Chair of five workshops. He is a member of the Eta Kappa Nu electrical engineering honor society, the Sigma Xi science honor society, and the Upsilon Pi Epsilon computing sciences honor society. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. For more information, please see www.engr.colostate.edu/∼hj.

**Anthony A. Maciejewski** received the B.S.E.E, 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. An up-to-date vita is available at www.engr.colostate.edu/∼aam.

**Ye Hong** is a Ph.D. student in Electrical and Computer Engineering at Colorado State University. He received his Masters degree in Computer Science from Tsinghua University in 2006, and his Bachelors degree from Tsinghua University in 2002. His current research interests include parallel and distributed computing, heterogeneous computing, robust computer systems, and performance evaluation.

**Bradley L. Lock** received his B.S. in Computer Engineering from Colorado State University in May 2001. He is pursuing an M.S. degree in Electrical Engineering at Colorado State University. He currently works for Intel Corporation as a VLSI design engineer.

**Mohammad Nayeem** received his B.E. degree in Electronics and Communications Engineering from Regional Engineering College, University of Kashmir in 2000 and his M.S. in Computer Science from Colorado State University in 2007. He worked in Wipro Infotech India from 2000 to 2003 as a Customer Services Engineer and as a software engineer in Hewlett-Packard from 2003 to 2005. He is currently a Computer Science Ph.D. student at Colorado State University. His research is focused on machine learning, scheduling, and optimization using artificial intelligence techinques.

**Fadi Wedyan** received his B.S. in Computer Science from Yarmouk University in 1995 and his M.S. in Computer Science from Al-albayt University in 1999. He worked as a Systems developer from 1996 to 1999 at the Free Zones Corp., Jordan and as an Instructor at the Jordan University of Science and Technology from 2000 to 2005. He is currently a Computer Science Ph.D. student at Colorado State University. His research interests are scheduling and optimization.