Gluck, K. A., Scheutz, M., Gunzelmann, G., Harris, J., Kershner, J. (2007). Combinatorics meets processing power: Large-scale computational resources for BRIMS. In *Proceedings of the Sixteenth Conference on Behavior Representation in Modeling and Simulation* (pp. 73-83). Orlando, FL: Simulation Interoperability Standards Organization.

Combinatorics Meets Processing Power: Large-Scale Computational Resources for BRIMS

Kevin Gluck
Air Force Research Laboratory
Warfighter Readiness Research Division
6030 S. Kent St.
Mesa, AZ 85212
480-988-6561 x-677
kevin.gluck@mesa.afmc.af.mil

Matthias Scheutz
Department of Computer Science and Engineering
351 Fitzpatrick Hall
University of Notre Dame
Notre Dame, IN 46556
574-631-0353
mscheutz@nd.edu

Glenn Gunzelmann
Jack Harris
Air Force Research Laboratory
Warfighter Readiness Research Division
6030 S. Kent St.
Mesa, AZ 85212
480-988-6561 x-674; x-675
glenn.gunzelmann@mesa.afmc.af.mil;
jack.harris@mesa.afmc.af.mil

Jeff Kershner
L3 Communications at
Air Force Research Laboratory
Warfighter Readiness Research Division
6030 S. Kent St.
Mesa, AZ 85212
480-988-6561 x-677
jeffrey.kershner@mesa.afmc.af.mil

Keywords:
combinatorics, computational models, high performance computing, volunteer computing

ABSTRACT: *The sophistication of computational cognitive architectures has opened the door to model development across a range of human activity. However, a continuing challenge for model developers is validating the model both by exploring the range of a model's behavior and by optimizing mechanisms, knowledge, and parameters to best account for human cognition and performance. We have been developing an infrastructure to facilitate this process that takes advantage of high performance computing resources and technologies to allow for more ambitious model development and validation efforts. This work involves a combination of increased computational power and increased sophistication in running models using available resources. Examples of the promise of this work are presented, along with current areas of emphasis and future directions.*

1. Combinatorics in Cognitive Architecture and Model Development

Cognitive architectures (e.g., ACT-R, Soar) are intended to serve an integrating and unifying purpose within the cognitive science community. The goal is to bring together what are typically isolated mathematical and computational theories of the many components of the human perceptual, cognitive, and motor systems into a unified account of the entire complex, integrated system.

This is a fantastically ambitious scientific objective because implementing the computational mechanisms, representations, and processes that replicate and explain human performance across the full range of human experience requires simultaneous theorizing at three levels: overarching architectural design and control mechanisms, internal implementation of architectural sub-components/modules, and knowledge. Gray (in press) refers to these levels as Type 1, Type 2, and Type 3 theories of the human cognitive architecture.

Each of these three levels of theorizing has its own associated quantitative and qualitative parameters, and each of those internal parameter spaces is very large. The implementation of any particular cognitive model involves committing to a location at the intersection of these three very large (perhaps infinite) theoretical state spaces and subsequently evaluating the extent to which that location in the joint state space is an appropriate account of human performance in that situation. If the account is deemed inadequate, then the search is on for a different combination of qualitative and quantitative parameters that will provide a better account.

A thorough search and evaluation of even a modest portion of the total possible theoretical state space will require an unprecedented amount of computing power because of the combinatorics associated with searching a multi-dimensional space consisting of (1) additional architectural components or modifications to existing components, (2) changes to numerical parameters, (3) changes to the initial knowledge state (including new task strategies) of the architecture/model at the beginning of a run, (4) changes to the simulated task environment, or (5) all possible combinations of those things. Seemingly innocuous assumptions and implementation decisions can have dramatic downstream consequences in a complex system like a cognitive architecture that interacts with a simulation environment (Gluck, Ball, & Krusmark, in press).

To make this combinatorics issue more concrete, we will provide an example from an ongoing line of research in the development of a computational account of the effects of sleep-related fatigue on cognitive functioning.

1.1 An Example from Recent Research in Fatigue Modeling

Fundamentally, our research in this area involves identifying parameters that can be associated with fatigue within the ACT-R cognitive architecture (Anderson et al., 2004), and then identifying how those parameters change as fatigue levels increase. We (Gross, Gunzelmann, Gluck, Van Dongen, & Dinges, 2006; Gunzelmann, Gluck, Van Dongen, O'Connor, & Dinges, 2005) have developed an initial account of this relationship for a sustained attention task (the Psychomotor Vigilance Task, or PVT). The PVT involves monitoring a known location for the appearance of a stimulus. Participants respond as quickly as possible to the stimulus by pressing a response button. The stimulus appears at random but relatively frequent intervals (ranging from 2-10 s).

One challenge that is faced in this research is accurately replicating the changes in human performance that occur across a range of psycho-physiological conditions, from fully rested and awake to severely sleep deprived and fatigued. Human performance changes dramatically across this continuum, and so parameter values for the model must be determined and evaluated across a continuum of values as well. In addition, we want to use the model's performance across a range of values to infer a relationship between the parameters in ACT-R and predictions about overall 'alertness' that come from existing biomathematical models in the literature (e.g., Hursh et al., 2004; Jewett & Kronauer, 1999).

For our fatigue modeling efforts using the PVT, we manipulated 3 parameters across a relatively modest range of values to identify best fitting parameter values. One of these was related to a new architectural mechanism that we added to ACT-R, while the other two were existing parameters in the architecture. The full combinatorial parameter space had a total of 12,096 nodes (3 parameters, with 16, 21, and 36 different values respectively). Performing this evaluation on local, independent machines was manageable, but not efficient. It took nearly a month of real time to complete the parameter space exploration using 5 local computers. We manually divided the parameter space into many smaller pieces, to avoid system crashes from memory stack overflows. Thus, the month-long process included significant down-time resulting from situations in which the evaluation of a particular subspace would finish in the middle of the night or over a weekend, when no one was around to manually start the next component. It also required substantial hands-on involvement to run models, monitor progress, and merge and analyze the resulting data.

The results of this modeling effort have been successful, leading to a tentative link between fatigue and two procedural parameters in ACT-R (see Gross et al., 2006). However, the story is more complicated. There are actually substantial individual differences, both in the effects of fatigue on human cognitive performance and in human performance on particular tasks (including the PVT). This, of course, suggests that different individuals may be represented better by models containing different parameters and by different functions mapping alertness to ACT-R parameters. More surprisingly, even in this simple reaction time task, we have identified five ACT-R model variations which differ in the micro-structure of their psychomotor strategies for doing the PVT. These variations produce qualitatively different reaction time profiles, all of which can be observed in different human participants.

Because of the complications associated with modeling the effects of fatigue at this level of detail, exploration of the full parameter space becomes untenable using our local resources. For the PVT, we have access to data from 13 participants, each of whom performed 44 sessions of the task across 88 hours of total sleep deprivation. With five versions of the model, the parameter space expands to 60480 nodes. Not only does this increase the overhead associated with managing the parameter space, but on local machines it also results in an unacceptable lag between conceptualization of the evaluation to be performed and when the results are available for analysis. To overcome these obstacles, we have turned to high performance computing (HPC), which we describe next.

2. High Performance Computing

There are two broad ways in which HPC can be used in the context of cognitive architectures and in human behavior representation more generally: (1) to explore parameters and system configurations to improve the performance of a particular model in a specific context (improving goodness-of-fit or demonstrating sufficient capability), and (2) to evaluate the performance of a particular model in a broad range of conditions (testing robustness, generalizability, and validity). Although (1) is mainly aimed at the formative, design stage of a model, and (2) is mainly aimed at the summative, evaluation stage, it is possible to combine (1) and (2) in either stage. For example, parameters can be determined for a variety of initial conditions, or a model can dynamically change parameters during performance to adapt to changing circumstances. Our use of HPC to date has focused primarily on use type 1 (improving goodness-of-fit and demonstrating sufficiency), but as our computational theory of the effects of fatigue on the cognitive system matures our thoughts and actions are increasingly on use type 2 (robustness, generalizability, and validity).

Moreover, our use of HPC to date has nearly entirely involved full combinatorial search of target parameter spaces. There are three reasons for this. One is that it can be informative to explore combinations of parameter values over wide ranges. Sometimes it confirms your knowledge and intuitions regarding the functioning of the architecture and the influence of the parameters, and other times there are interesting surprises in the interactions. A second reason is that we explicitly wanted to scale up our demands on the HPC resources in order to learn how they held up and what sort of return time we got from our batch run requests. A third reason for our use of full combinatorial search is that we wanted to get some baseline data on search completion time, so that we can use those data as a standard against which to compare the completion times and conclusions regarding optimal parameter combinations that come from more sophisticated and efficient search techniques. These include sampling of a given parameter space according to some distribution, genetic algorithms that determine good values in the space, different gradient-based methods (e.g., gradient descent based on a cost-function), and constraint satisfaction search methods. Moreover, it is possible to use domain-specific heuristics that can guide the search process, possibly including meta-reasoning to evaluate the current state of a parameter search and to determine which heuristics to employ in addition to seeding genetic algorithms. For modelers, the advantage of such methods is that they run relatively autonomously, thereby dramatically reducing the time required to explore the model implementation space. Modelers can then use their expertise to define heuristics and meta-rules and to interpret the results streaming back from the HPC. Moreover, by automating the process of parameter fitting, models can be produced more quickly, alternative models can be generated as part of the process, and different high-quality models for one condition can be automatically compared across a variety of other conditions to determine their strengths and weaknesses.

2.1 Initial Demonstrations

Over the last year we have used internal research funds from AFRL's Human Effectiveness Directorate to establish an account and begin performing ACT-R model parameter searches using AFRL's Major Shared Resource Center for High Performance Computing, which is located at Wright-Patterson Air Force Base, Ohio. That facility has two clusters of 2,048 processors running Linux RedHat 2.4. The less capable of these, the SGI Altix 3700 (Eagle) cluster, has 1.6 GHz processors, with 1 GB of memory per processor and 100 TB of data storage capacity. The more capable cluster, the HP XC (Falcon) cluster, has 2.8 GHz processors, with 2 GB of memory per processor, and a 97 TB workspace. Naturally, we chose to use the more capable cluster for our four demonstration batch runs.

These demonstrations involved full combinatorial searches of increasingly large parameter spaces using an ACT-R cognitive model for the Walter Reed Serial Addition-Subtraction Task. This task involves solving simple, single-digit addition and subtraction problems. For non-negative results, the correct response is the ones digit of the result. For negative results, the correct response is the result, plus 10. Thus, the task requires participants to respond with the result of the operation, mod 10. We also have performance data for this task from human participants in a sleep deprivation study. We use these data to evaluate the validity of the model as a replication of human performance under varying degrees of sleep deprivation.

For those who may be interested in such details, Table 1 provides the breakdown of ACT-R parameters used in each of these four HPC batch runs. The table shows the minimum and maximum values used for each parameter, which defines the range across which we searched. "Step" defines the grain size of the search within that range. "Levels" is determined by the range and the step size. For instance, in our first HPC batch run we searched across values of *G* (interpreted as "arousal" in this particular model) from 1.5 to 2.2, at a step size of .1, which results in 8 levels of *G*. Since this was full combinatorial search, multiplying the levels for all of the parameters within a batch run reveals the total number of parameter value combinations, or number of nodes, across which we searched. These values are presented in Table 2 and in Figure 1. The first two searches were both 9,600 nodes, while the third was 118,482 nodes. The last batch run was 1,777,236 nodes.

Table 1
ACT-R Parameters Used in Four HPC Batch Runs

	HPC Batch Runs			
Parameter	1	2	3	4
G				
Min	1.5	3.0	2.0	3.0
Max	2.2	4.5	5.0	5.0
Step	0.1	0.1	0.1	0.1
Levels	8	16	31	21
T_u				
Min	1.6	1.6	1.5	1.5
Max	2.0	1.8	4.0	4.0
Step	0.1	0.1	0.1	0.1
Levels	5	3	26	26
BLM				
Min	5,000	5,000	5,000	4,000
Max	25,000	25,000	25,000	30,000
Step	5,000	5,000	1,000	250
Levels	5	5	21	105
T_r				
Min	-3.0	-3.0	-2.0	-2.0
Max	0.5	0.5	1.0	1.0
Step	0.5	0.5	0.5	0.1
Levels	8	8	7	31
ANS				
Min	0.20	0.20	0.25	0.25
Max	0.30	0.28		
Step	0.02	0.02		
Levels	6	5	1	1

Note. *G* traditionally refers to the value of the goal in ACT-R models, but for our fatigue theory we have reinterpreted it as arousal; T_u is the utility threshold; *BLM* is a Base-Level Multiplier for chunk activations; T_r is the chunk retrieval threshold; *ANS* is activation noise.

In addition to the number of parameter combination nodes in each batch run, Table 2 provides data on the number of model iterations at each node, the precise form of our requests to the HPC cluster (a request to the cluster is called a “job” and each job specifies a number of processors, each for a specific period of time), how many CPU hours each batch run required, and how long it took to get results back after we submitted the request.

Figure 1 plots CPU Time (in 1000’s of hours) and Total Time to complete (in hours) for each of our four demonstration batch runs. These data are available in Table 2, but they are worth plotting visually as line graphs as well in order to emphasize three important points: improvements in research efficiency, enabling new research options, and the eventual exhaustion of the HPC resources as parameter searches become more aggressive. We discuss each of these points in more detail below.

Table 2
Data on Batch Requests and Results Turnaround Time

	HPC Batch Runs			
	1	2	3	4
Nodes	9,600	9,600	118,482	1,777,230
Iterations	25	25	50	50
Jobs	1	1	31	273
Processors	64	75	26	62
CPU Tm (hrs)	400	515	8,624	96,096
Wait Tm (hrs)	0.01	0.17	12	71.87
Run Tm (hrs)	6.25	6.87	10.7	5.48
Tot Tm (hrs)	7.32	7.37	44.25	205.18

Note. “Wait Tm” is the average time each job waited in the queue before it started running; “Run Tm” is average run time per processor once the job started; “Tot Tm” is the amount of real time that passed between submitting the first job request and completion of the runs on the HPC cluster.

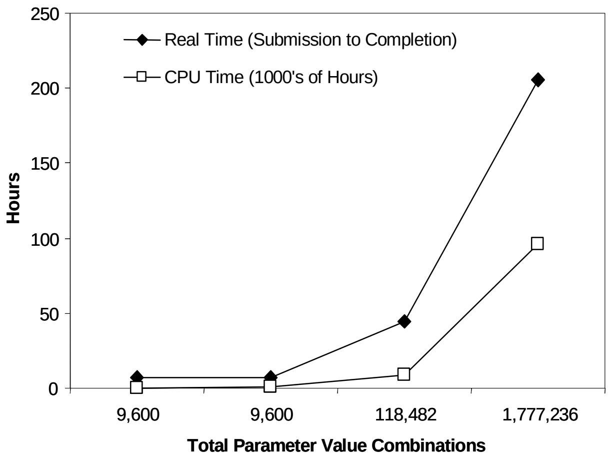

Figure 1. Use Data for Serial Addition-Subtraction Task (SAST) Cognitive Model Batch Runs on the AFRL HPC (Falcon Cluster)

In Section 1.1 we described the month-long process associated with searching a fairly modest parameter space (12,096 nodes) using five or so otherwise-idle research computers in our local laboratory. Now consider demonstration batch run 3, which was 118,482 nodes. Our best guess estimate of the completion time for a run of this size using locally available processors in our laboratory in Mesa, AZ is

1.5 years. The results came back from the HPC cluster at Wright-Patterson AFB in less than 45 hours, for an improvement in research efficiency of more than two orders of magnitude. Two orders of magnitude improvements in efficiency have a big impact on a research program when the baseline completion times are measured in months and years.

When we increase the size of the parameter space another order of magnitude (and then some) to the nearly 1.8 million nodes in batch run 4, any comparison to completing that parameter search with local resources becomes ridiculous. We simply would not undertake a search of that depth and breadth with local resources, because it would take 10 or more years to complete. Thus, access to the HPC resources is enabling us to ask research questions and do things with our models that previously have not even been possible.

Still considering the very large Batch Run 4, when we divide total CPU time required to search that space (96,096 hours) by the real completion time required for the run (205 hours), we find that over the course of that week we averaged 468 computer processing hours per hour. The maximum allowable use of the Falcon cluster per account is 1024 processors simultaneously, so we were averaging slightly over 45% of our maximum allowable usage during those 205 hours.

This level of usage appears to have been acceptable to the automated HPC scheduler and to the human HPC management and maintenance employees for a period of a week, but clearly they are not going to allow a single user to occupy 25-50% of their resource indefinitely. This is relevant because it speaks to the fact that it won't be long before we exhaust the AFRL HPC as a resource if we continue the brute force, full combinatorial search approach we have been using to date.

3. More Intelligent Use of HPC

Since we already envision a time in the not-too-distant future in which our computational requirements will exceed even the impressive resources available through AFRL's HPC, we have begun investing in ways to decrease our total computational load per research question through more intelligent use of the HPC resources.

A key component of our emerging strategy is the use of middleware that provides intelligent interoperability between our cognitive models and whatever processing resources are being used at the time, HPC or otherwise. This system, called the SoftWare Architecture for Grid Experimentation System (SWAGES), is described in this section.

3.1 An Introduction to SWAGES

SWAGES[1] is a grid-based experimentation environment that is targeted at scheduling and supervising possibly distributed runs of cognitive models or agent-based simulations. It consists of several heterogeneous, distributed components that cooperate closely to achieve a high level of resource utilization in a dynamic, possibly uncontrolled computing environment (such as publicly accessible computing clusters at Universities where the uptime of machines cannot be guaranteed). SWAGES is implemented in the distributed ADE system (Scheutz 2006), which provides a location-independent multi-agent system infrastructure, with automatic error detection and recovery, and works in homogeneous fixed clusters (e.g., Beowulf clusters) and heterogeneous ad-hoc clusters (e.g., individual workstations that can only be used if nobody is logged in) alike. It requires no set-up procedures on the host participating in simulation experiments (as is required by other distributed computation environments like CONDOR or SETI@HOME) other than the commonly installed *secure shell tools* for secure, remote login and file transfer) and will run on all operating systems that support the JAVA virtual machine.

Over the last several years, SWAGES has been used in combination with the agent-based SimWorld simulation environment for investigations of large parameter spaces of agent architectures for various kinds of agents, from simple reactive agents (Scheutz & Schermerhorn, 2005, Schermerhorn, 2006) to more complex, deliberative agents (Scheutz & Schermerhorn, 2002). For example, a parameter search involving over 2 million simulations (of 3 min. each) is reported in Schermerhorn (2006). This parameter search required three months of compute time on our AIROLAB cluster, which consists of 22 nodes with two 2.4 GHz Pentium IV CPUs each running Linux 2.6 kernels (on a single 2.4 GHz Pentium CPU the computations would have taken over 11 years). SWAGES has also been used in conjunction with a neural network simulator to conduct parameter searches for neural network-based cognitive models (Scheutz & Gibson, 2006).

Recently, new components have been integrated in the SWAGES system, which will significantly increase its

1 SWAGES can be obtained from the following site: http://www.nd.edu/~airolab/software/. Please contact Matthias Scheutz with SWAGES-related questions.

utility for cognitive modeling by partly automating several critical steps in the modeling process: (1) automatic model discovery, (2) automatic parameter fitting, and (3) automatic model verification. Prerequisite for all three steps is the system's ability to run simulations, analyze the results, determine whether they fit a given criterion, and possibly scheduling more simulations if the criterion is not met. In the following, we will first briefly describe the overall SWAGES architecture, and then show how the various parts work together to allow for the automation of various aspects of the modeling process.

The SWAGES architecture (see Figure 2) can be divided into *server-side* and *client-side* components, where the *server-side* components provide the distributed computation infrastructure, and the *client-side* components provide the communication components and the simulation platform SimWorld. Currently, all server-side components are run on a single host – the *grid server* – while client-side components run on multiple *simulation hosts*.

3.1.1 Server-side Components

The grid server is the central locus of control of a SWAGES system, running various server-side components to schedule, distribute, start, and monitor the execution of distributed simulations and recover from failures.

The *experiment server* is responsible for setting up *experiment sets* (possibly consisting of large numbers of individual experiments (see Fig. 1). Important factors here are generation of initial conditions (unique or identical) across different experiments in a set, priorities of experiments and scheduling parameters, levels of supervision and recovery parameters, format of data collection and location for data storage, statistical analysis of results and output formats, and user notification of progress.

The *scheduler* is responsible for taking experiments from several priority-based queues (in which new experiments are submitted by the experiment server) and starting them on remote hosts. The experiment scheduler will dynamically create experiment data structures for large-scale experiment sets (to avoid memory shortages), and only schedule a new experiment when new hosts are available that are not needed for other experiments to finish.

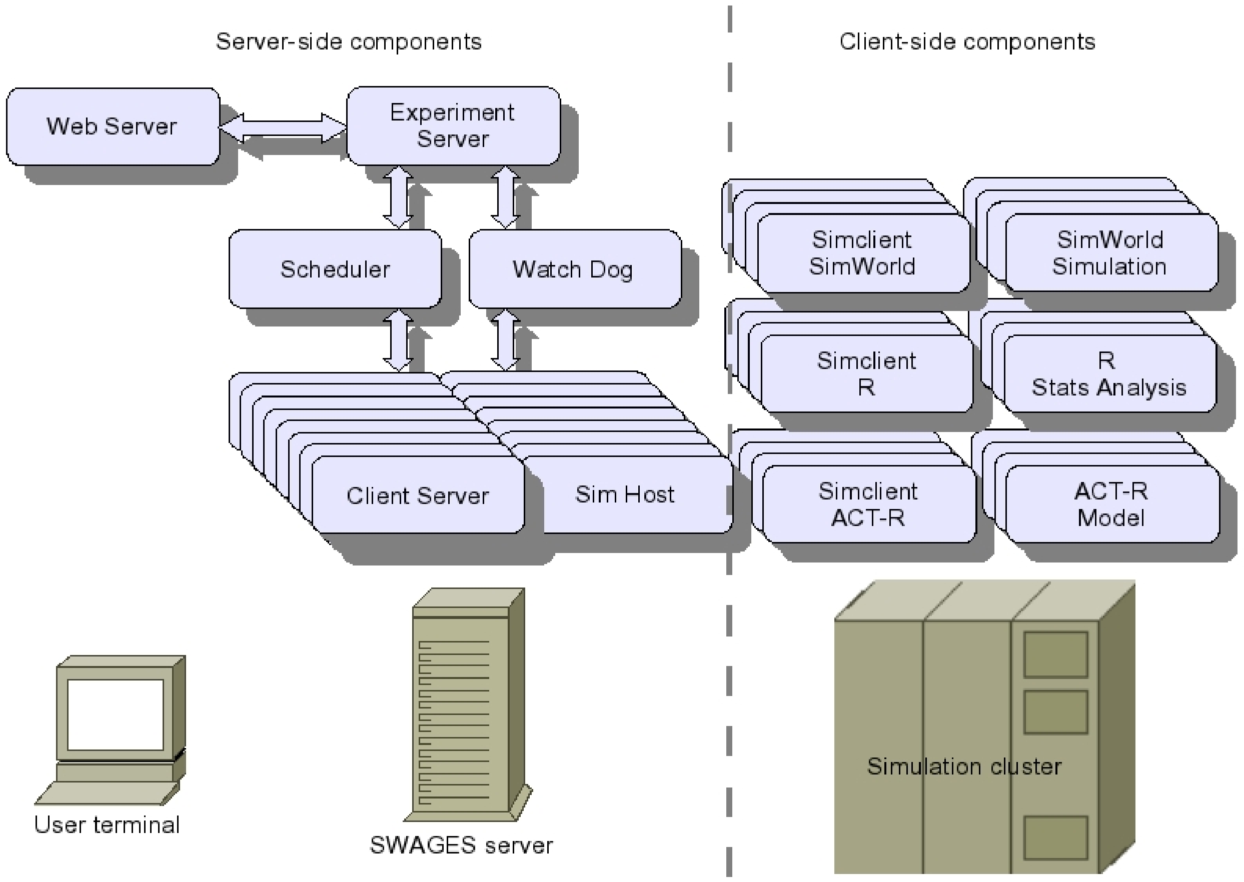

Figure 2. Diagrammatic representation of the SoftWare Architecture for Grid Experimentation System

The *client server* is the server-side representation of a remote simulation. It maintains an open communication channel with the simulation instance, keeping track of the simulation's progress, state, update, and degree of parallelization. The client-side *simclient* (described below) passes this information to the client server periodically, which stores the information for use by the *watchdog* and *web server* components (below). In addition, the simclient can request to be migrated to another host when user-specifiable conditions on the current simulation host are not met. When a new simhost becomes available, the simclient is started there. It is critical for error detection and recovery: when a simulation crashes (e.g., due to OS problems on its host), is not responding (e.g., due to network problems), or cannot be continued (e.g., because its current host no longer meets user-defined criteria for running simulations, CPU load average or number of users), the client server can resume the simulation elsewhere based on saved state.

The *watchdog* implements a second level of supervision which is particularly important for dynamic computing environments where hosts can “disappear” from a pool of usable machines. It regularly checks each client's status (as recorded by the client server) for progress. If the simclient has not checked in with the client server within the specified time frame, the watchdog assumes there has been a failure (e.g., due to OS problems on its host, or to network problems). The client is terminated on the server side (i.e., the data structures associated with that client are cleaned up), and the simulation is rescheduled on a new host. The new simulation is either restarted from the beginning, or, if there is saved state available, continued from the most recently saved state.

The *web server* provides a simple web-based interface to SWAGES that can be used to submit experiments, check their status, perform simple statistics and view the results.

The *simhost* is the server-side representation of a remote simulation host. It keeps track of which simulations are running on each host. The simhost also is responsible for monitoring the availability of the host for simulations. The simhost detects when the user-specified criteria for host availability are met (either initially when SWAGES is started, or after a simclient has requested to be migrated), as well as the physical availability of the host (e.g., whether the host is back up after a reboot caused the watchdog to reschedule a simulation previously running there).

3.1.2 Client-side Components

The client-side components are responsible for running the actual simulation and communicating its state to the server-side components.

The *simclient* represents the simulation to the (server-side) client server, periodically communicating updates about the simulation (e.g., cycles executed). It is also responsible for periodically saving the state of the simulation and checking the host it is running on for availability according to user-defined criteria. For example, the user may specify that simulations may only execute on hosts with no console user, or with a particular maximum load average. When the simclient detects a violation of these constraints, it can notify the simulation to save its state (if this is supported) and send it to the grid server, notifying the client server that the simulation needs to be migrated and restarted. Simclients also interact with simulations to gather statistics and data that can be stored in a common place and used for further analysis. The SimWorld simclient is a specific extension of the default *simclient* which can be used to automatically and adaptively parallelize SimWorld simulation in SWAGES based on host availability using a novel algorithm for automatic parallelization of agent-based simulations (Scheutz & Schermerhorn, 2006). Finally, simclients cannot only be used to control simulations, but also for performing statistical analyses, and for determining whether additional simulations need to be run in order to meet a modeling criterion (e.g., parameter fits).

3.2 SWAGES for Automatic Cognitive Modeling

Here we briefly describe our ongoing project on automating parameter exploration, parameter fitting, and ultimately model building itself using the SWAGES environment. The first step in making SWAGES work with any cognitive modeling environment like ACT-R, SOAR, EPIC, and others is to define a *simclient* in the programming language of the modeling environment (e.g., LISP or C/C++) to allow the client server to exchange message with the simulation. By implementing the simclient in the target language of the environment, the simclient works as a mediator that can directly call functions or methods of the modeling environment (e.g., to start, stop, and resume simulations), while relaying information about the simulation state in a generic way to the client server. Communication between the simclient and the client server is accomplished via sockets, where data structures of the simulation's programming environment are serialized and deserialized on the SWAGES side (currently, serializers for JAVA, C, LISP, and POP11 are available). That way it is possible to let modelers define experiments in the environment's native programming language (e.g., LISP for ACT-R), while allowing SWAGES to extract generic parameters from the experiment description (see below).

We have developed an ACT-R simclient for SWAGES that provides access to the ACT-R modeling environment and has specific support for automatic parameter space exploration. This is best demonstrated with the following simple example of an experiment definition written in LISP syntax for the ACT-R modeling environment (keywords for the ACT-R simclient and for the client server are printed in bold):

```
;;; Model simulation parameters
(((initfile "actr6/load-act-r-6.lisp")  ;;; load the ACT-R 6 virtual
machine
  (initfile "my-favorite-cognitive-model.cl")  ;;; load the cognitive
model
  (quitafter 5000) ;;; quit the model simulation after 5000 model
cycles
  (eval (setf modelparameter (variate V from V_low to V_high step
V_step))) ;;; see text
  (record end modeloutcome)) ;;; record this value at the end of the
simulation
;;; SWAGES experiment set parameters
 ((name "myexperiment" "mydirectory")  ;;; name of experiment and
storage location
  (quitif (< (compute average modeloutcome) threshold)) ;;;
condition for ending experiment
  (user "myself") ;;; the user name used for running the simulations
  (parallel 100))) ;;; how many random initial conditions
```

First note that the experiment description consists of two parts: a specification of parameters necessary to run the cognitive model (used by the simclient) and a specification of information about what kinds of experiments to conduct, where to store the results, and how to analyze them (used by the client server). In the particular example, the ACT-R 6 cognitive architecture together with a particular cognitive model will be loaded by the simclient, run for 5000 update cycles, and the value of the “modeloutcome” variable in LISP will be saved upon the completion of the simulation. Before the model is run, the variable “modelparameter” is set to a specific value as given by the “variate V”. Variates are special variables in experiment definitions that are parsed by SWAGES and set to particular values in the set of values given by the lower (V_low) and upper boundaries (V_upper) and the spacing in between (V_step). For example, the expression “(variate V from 1 to 10 step 1)” will result in 10 different experiment definitions. Thus, variates in SWAGES effectively allow for the easy definition of parameter spaces and subsequent samplings thereof. In the above case, the variate specification is used to schedule and run experiments until the outcome meets a termination criterion. Specifically, for each value of the variate starting from V_low to V_high in steps of V_steps, the model will be run with the “modelparameter” set to that value in 100 different random initial conditions (as specified by the “parallel” keyword), the average value of “modeloutcome” (over the 100 conditions) will be computed as soon as all 100 simulations have finished, and the result will be compared to the specified “threshold” according to the criterion expressed in the “quitif” condition. If the condition is met, the experiment is considered complete and no more conditions for different variate values will be generated. Conversely, if the condition is not met, the next 100 simulations for the next variate value will be generated and run, possibly exhausting all variate values without finding a value that meets the criterion.

While this is one of the simplest experiment definitions possible, we believe that it already shows the potential of SWAGES for automating the modeling process. Clearly, parameter spaces can be explored using the proposed mechanism in a systematic way, with or without termination criteria, possibly leading to new insights about the relation among different parameters (e.g., automatic correlations could be performed among variates to determine dependencies and searches finished if dependencies are found). Moreover, by systematically searching a parameter space until a specific termination criterion is met, SWAGES allows for model fitting – the parameter space to be searched would be the space of free model parameters and the termination criterion some error measure determining the quality of the fit. Finally, if model rules were allowed to be varied – a functionality not currently possible, but planned for future development – then the space of possible models itself could be searched in the above outlined way, thus allowing for automatic model discovery (within the confines of the automatic rule variation and generation).

Recently, we finished the first proof-of-concept demonstration of the above kind of automated parameter space search for a simple ACT-R test model on the AIROLAB cluster at Notre Dame. In a next step, we will demonstrate this process on the AFRL HPC cluster to perform more extensive evaluations of our theory of how fatigue affects the cognitive system, implemented as a computational theory in ACT-R. This demonstration will have a twofold outcome: (1) it will yield data that will help us to extend our account of fatigue to include individual differences in the impact of sleep loss on performance, and (2) it will yield performance data for the SWAGES system that can be used to quantify the utility of using SWAGES to automate the modeling process (e.g., in terms of time savings for obtaining good fits compared to manual explorations of model parameters).

4. Volunteer Computing

In addition to investing in more intelligent use of HPC, we have begun exploring the idea of volunteer computing as a means of acquiring greater processing power. In recent years, scientific communities facing large, computationally complex problems have found ways to leverage volunteer home computing nodes to vastly increase their access to computing resources. According to the University of California, Berkeley website, volunteer computing grids "supply more computing power to science than does any other type of computing." (See the volunteer computing website at Berkeley for more information: http://boinc.berkeley.edu/volunteer.php) A few of the communities already using this vast resource include astronomy, physics, chemistry, earth sciences, biology, medicine, mathematics and strategy games. One of the best known projects is Berkeley's SETI@home (Search for Extra Terrestrial Intelligence) which has 475,000 active computers worldwide processing on average 615 Tera-floating point operations per second. With funding from the National Science Foundation, Berkeley has developed a software platform for distributed computing using volunteered computer resources called the Berkeley Open Infrastructure for Network Computing (BOINC). A single BOINC project with a single server can provide computing power equivalent to a cluster with tens of thousands of CPUs. To the best of our knowledge the HBR community has not yet begun using this resource, and we would like to change that.

Work on our BOINC-based cognitive modeling infrastructure has recently begun, following the arrival of seed funding from the AFRL Human Effectiveness Directorate. The hardware and software used to support a BOINC project have been configured, and designs for the actually distributed cognitive model application will begin soon. Once completed, the project will become operational and cognitive models can be distributed to volunteer clients. We plan to report the results of this initial demonstration in a future paper.

5. Summary and Concluding Comments

In this paper we described our initial forays into the use of HPC resources for computational cognitive modeling. Our interest in HPC arose, at least partially, out of the realization that scaling up the computational rigor and thoroughness of our search for a valid theory of fatigue in the cognitive system was being impeded by the limitations of our modest local computational resources. This situation is far from unique to us. In fact, this is the status quo among researchers in cognitive modeling and human behavior representation.

In the short time we have been investigating the use of HPC for our research purposes, it has become apparent that if we continue the path we are on (exponential growth of computational load) then soon we will exceed even the capacity of the very large HPC resources available through AFRL. One strategy for dealing with this is to search more intelligently and therefore more efficiently. We think at this time that the SWAGES system will serve a helpful role in accomplishing that. Another strategy is to acquire access to even more computational resources than those available through the AFRL HPC. This is what is motivating our interest in volunteer computing, which has the advantage not only that it potentially creates access to tens or hundreds of thousands of computer processors, but it also is available to anyone (unlike the AFRL HPC, which requires a security clearance in order to set up a user account).

We hope this description of our progress motivates other cognitive modelers and human behavior representers to begin using HPC and/or volunteer computing resources in their own research. We will make faster progress as a field if we work together to overcome the combinatorial nightmare inherent in the development of formal models of human beings.

6. References

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y . (2004). An integrated theory of the mind. *Psychological Review 111*, (4). 1036-1060.

Andronache, V., & Scheutz, M. (2004) Integrating Theory and Practice: The Agent Architecture Framework APOC and its Development Environment ADE. In *Proceedings of AAMAS 2004*, ACM Press

Davis, L. W., & Ritter, F. (1987). Schedule optimization with probabilistic search. *Proceedings of the Third Conference on Artificial Intelligence Applications*. IEEE Computer Society. 231-236.

Gluck, K. A., Ball, J. T., & Krusmark, M. A. (in press). Cognitive control in a computational model of the Predator pilot. To appear in W. Gray (Ed.) *Integrated Models of Cognitive Systems*. New York, NY: Oxford University Press.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Gray, W. (Ed.) (in press). *Integrated models of cognitive systems*. New York, NY: Oxford University Press.

Gross, J. B., Gunzelmann, G., Gluck, K. A., Van Dongen, H. P. A., & Dinges, D. F. (2006). Prediction of circadian performance during sleep deprivation. In R.Sun & N Miyake (Eds.), *Proceedings of the Twenty-Eighth Annual Meeting of the Cognitive Science Society* (p. 297-302). Mahwah, NJ: Lawrence Erlbaum Associates.

Gunzelmann, G., Gluck, K. A., Van Dongen, H. P. A., O'Connor, R. M., & Dinges, D. F. (2005). A neurobehaviorally inspired ACT-R model of sleep deprivation: Decreased performance in psychomotor vigilance. In B. G. Bara, L. Barsalou, and M. Bucciarelli (Eds.), *Proceedings of the 27th Annual Meeting of the Cognitive Science Society* (pp. 857-862). Mahwah, NJ: Lawrence Erlbaum Associates.

Hursh, S. R., Redmond, D. P., Johnson, M. L., Thorne, D. R., Belenky, G., Balkin, T. J., Storm, W. F., Miller, J. C., & Eddy, D. R. (2004). Fatigue models for applied research in warfighting. *Aviation, Space, and Environmental Medicine, 75(3),* A44-60.

Jewett, M. E., & Kronauer, R. E. (1999). Interactive mathematical models of subjective alertness and alertness in humans. *Journal of Biological Rhythms, 14,* 588-597.

Kramer, J., Scheutz, M., Brockman, J., & Kogge, P. (2006) "Facing up to the Inevitable: Intelligent Error Recovery in Massively Parallel Processing in Memory Architectures." In *Proceedings of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, NV.

Schermerhorn, P. (2006). The cost of communication: Efficient coordination in multi-agent territory exploration tasks. Thesis. Department of Computer Science and Engineering, University of Notre Dame.

Scheutz, M. (2006). "ADE - Steps Towards a Distributed Development and Runtime Environment for Complex Robotic Agent Architectures". *Applied Artificial Intelligence, 20,* 4-5, 275-304.

Scheutz, M., & Andronache, V. (2004) The APOC Framework for the Comparison and Evaluation of Agent Architectures. In *Proceedings of AAAI Workshop on Intelligent Agent Architecture 2004.* AAAI Press.

Scheutz, M., & Gibson, B. (2006). Visual attention and the semantics of space: Evidence for two forms of symbolic control. *Proceedings of the 28th Annual Meeting of the Cognitive Science Society.* Vancouver, British Columbia, Canada.

Scheutz, M., & Schermerhorn, P. (2002). Steps towards a systematic investigation of possible evolutionary trajectories from reactive to deliberative control systems. *Proceedings of the 8th International Conference on the Simulation and Synthesis of Living Systems.*

Scheutz, M., & Schermerhorn, P. (2005). Predicting population dynamics and evolutionary trajectories based on performance evaluations in ALife simulations. *Proceedings of the 2005 Genetic and Evolutionary Computation Conference*, Washington, D.C.

Scheutz, M., & Schermerhorn, P. (2006). Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. *Journal of Parallel and Distributed Computing, 66* (8), 1037-1051.

Scheutz, M., Schermerhorn, P., Connaughton, R., and Dingler, A. (2006) "SWAGES--An Extendable Parallel Grid Experimentation System for Large-Scale Agent-Based Alife Simulations". In *Proceedings of the 10th International Conference on the Simulation and Synthesis of Living Systems*, Bloomington, IN.

Author Biographies

KEVIN GLUCK is a Senior Research Psychologist at the Air Force Research Laboratory's Warfighter Readiness Research Division, where he is the Senior Scientist for the Integrated Research Operations Branch and the Team Lead for the Performance and Learning Models (PALM) research team. Kevin earned his B.A. in psychology at Trinity University in 1993 and his PhD in cognitive psychology from Carnegie Mellon University in 1999.

MATTHIAS SCHEUTZ is Assistant Professor of Computer Science and Engineering and Director of the Artificial Intelligence and Robotics Laboratory (AIROLAB) at the University of Notre Dame. His research interests include artificial intelligence, cognitive science, and philosophy. Matthias earned a PhD in Philosophy from the University of Vienna in 1995 and a joint PhD in Cognitive Science and Computer Science from Indiana University Bloomington in 1999.

GLENN GUNZELMANN is a Research Psychologist with the Air Force Research Laboratory. His research is oriented around the primary interest of developing psychologically valid computational accounts of human cognition and performance. Primary research projects include using multiple tasks and contexts to identify and validate mechanisms for human spatial competence and for explaining the effects of fatigue on human cognitive performance. Dr. Gunzelmann received a B.A. in psychology from Albright College (1997), a M.S. in psychology from the University of Florida (1999), and a Ph.D. in cognitive psychology from Carnegie Mellon University (2003).

JACK HARRIS is a Computer Scientist at the Air Force Research Laboratory's Warfighter Readiness Research Division. His research interests include artificial intelligence for high performance and volunteer computing platforms and cognitive modeling in dynamic, complex, time-pressured domains, such as Predator reconnaissance missions. Jack earned his B.S. in Computer Science from the Georgia Institute of Technology in 2001.

JEFF KERSHNER is a Software Engineer with L3 Communications, working at the Air Force Research Laboratory's Mesa Research Site. His interests range from web development to computer graphics to database administration to high performance computing. He received his A.A. in Music Theory at Villa Maria (1996) and a B.S. in Computer Science at the State University of New York at Fredonia (2000).