Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

CONTROLLING SCALABILITY OF DISTRIBUTED VIRTUAL ENVIRONMENT SYSTEMS

H. Lally Singh

Google, Inc.
111 8th Ave
New York, NY 10011, USA

Denis Gračanin

Department of Computer Science
Virginia Tech
2202 Kraft Drive
Blacksburg, VA 24060, USA

Krešimir Matković

VRVis Research Center
Donau-City Str. 1
A-1220 Vienna, AUSTRIA

ABSTRACT

A Distributed Virtual Environment (DVE) system provides a shared virtual environment where physically separated users can interact and collaborate over a computer network. There are three major challenges to improve DVE scalability: effective DVE system performance measurement, understanding the controlling factors of system performance/quality and determining the consequences of DVE system changes. We describe a DVE Scalability Engineering (DSE) process that addresses these three major challenges for DVE design. The DSE process allows us to identify, evaluate, and leverage trade-offs among DVE resources, the DVE software, and the virtual environment. We integrate our load simulation and modeling method into a single process to explore the effects of changes in DVE resources.

1 INTRODUCTION

Distributed Virtual Environment (DVE) systems provide a shared virtual environment (virtual world) where physically separated users can interact and collaborate over a computer network. A DVE system consists of a physics engine, a virtual environment, a network synchronization, and the user interface components.

Current performance engineering techniques are not well suited for DVE systems. The primary performance factors in DVE systems are not well understood and vary from system to system. The user load is unusually complex since it is determined by the user behavior inside the virtual environment and the overall demands the users' behaviors put on the DVE system and resources.

The DVE system performance requirements depend on the characteristics of the virtual environment component. We need to understand interactions between user behavior, the virtual environment, and system performance in order to determine the performance requirements and performance consequences of DVE design decisions for the virtual environment and the supporting DVE system components.

The current scalability limits in DVE systems reduce their utility and quality of user experience. DVE systems need to have partitions to scale up the number of users. The attempts to hide the existence of these partitions make DVE systems harder to use due to their requisite user interface complexity and artificial in-environment limitations. For a DVE system to be successful, the users must choose to participate in it.

Traditionally, performance management was done on an ad-hoc basis. As DVE systems continue to become more sophisticated, a methodology becomes more useful. A DVE Scalability Engineering (DSE)

process is a performance engineering process specifically designed for the interactions DVE systems have among user behavior, system performance, and the underlying software and computer resources.

We have demonstrated (Singh, Gračanin, and Matković 2012a, Singh, Gračanin, and Matković 2012b) how to apply the DSE process to find significant performance (and consequently, scalability) factors and how to adjust the DVE system's virtual environment to perform (and scale) better. This paper presents an expanded version of the DSE process that addresses the three major challenges for DVE design, i.e., effective DVE system performance measurement, understanding the controlling factors of system performance/quality and determining the consequences of DVE system changes.

2 RELATED WORK

Singhal and Zyda (1999) introduced the Networked Virtual Environment (Net-VE) Information Principle:

> The resource utilization of a Net-VE is directly related to the amount of information that must be sent and received by each host and how quickly that information must be delivered by the network.

The resource use is described using Equation 1:

$$\text{Resources} = M \times H \times B \times T \times P \tag{1}$$

- M = number of messages transmitted in the Net-VE
- H = average number of destination hosts for each message
- B = average amount of network bandwidth required for a message to each destination
- T = timeliness with which the network must deliver packets to each destination (that is, larger values of T imply that packets must be delivered with minimal delay, while small values of T may be delivered with longer delays)
- P = number of processor cycles required to receive and process each message

Net-VE Information Principle, or NVE-IP for short, gives us a rough understanding of how design decisions for the communications and simulation components of a DVE system influence resource use.

2.1 Distributed Virtual Environments

DVE systems are used for therapy, training, collaboration, and play. For example, Baños et al. (Baños, Guillen, Quero, García-Palacios, Alcaniz, and Botella 2011) discuss using virtual environment to recreate the traumatic experience to help people recover from Post-Traumatic Stress Disorder (PTSD). Another example is World of Warcraft, a Massively Multiplayer Online Role-Playing Game (MMORPG) with ten million users. Pittman's five-week study (Pittman and GauthierDickey 2010) found a few players online for over eight hours, but most user sessions lasted 200 minutes or less.

Many DVE systems, especially First Person Shooters (FPSs), follow the pioneering work in Distributed Interactive Simulation (DIS) and SIMNET (Fullford 1996): a shared virtual environment with system updates at essentially frame rate. While some systems may require that each user awaits their turn to act, the experience in these DVE systems is fully interactive, with time moving continuously. The list of exemplar FPSs is large, including Doom, Quake (Bylund and Espinoza 2002), and America's Army.

DVEs such as Second Life, where users can build the contents of virtual worlds, have potentials as a platform for serious applications e.g., group meetings and Computer Supported Cooperative Work (CSCW) (Lindeman, Reiners, and Steed 2009, Wadley and Ducheneau 2009, Friedman, Karniel, and Dinur 2009). It is important to understand the relationship between changes in DVE system characteristics/behavior and group dynamics (Lazem, Gračanin, and Harrison 2012). DVE development can benefit from component/module based approach to reuse and compose the existing content and behaviors components rather than building DVE systems from scratch. For example, OpenSimulator (Fishwick 2009), an open source

alternative to the Second Life "back end" server, provides a basic set of simulation functionality that allows DVE developers to have more control over DVE system characteristics in response to the user's actions and changes in group dynamics. More modular approaches, with a rich set general purpose, reusable simulation and content modules (Liu, Bowman, Hunt, and Duffy 2012), could empower non-expert users to develop DVE systems.

The most widely used but still expensive and time-consuming method to accurately user load test a DVE system is conducting a user study with many real users. Weilbacher noted that the Gears of War 3 project needed over 100 participants to perform a large user study test (Weilbacher 2012).

2.2 Quality of Experience

Quality of Experience (QoE) focuses on the subjective measurement of overall end-user experience. The quality of the user's experience in a DVE system is an important factor that can affect how well users can achieve their goals and whether they choose to participate. Several classifications exist, each separating qualitative elements (QoE) and quantitative elements (Quality of Service — QoS).

One particularly comprehensive attempt at defining QoE is presented by Laghari and Connelly (Laghari and Connelly 2012). They identify four domains: human, technological, contextual, and business. The human domain includes roles, demographics, objective and subjective QoE factors. Human QoE factors include psychological and physiological factors, such as intentions and behavior (Connelly), memory, attention, task performance, and human response time.

Wu et al. (Wu, Arefin, Rivas, Nahrstedt, Sheppard, and Yang 2009) present a QoE Quality Framework with a focus on quantitative analysis and prediction. Their methodology includes measurable system performance values (latency, jitter, etc.), measurable subjective user values (sense of presence, perceived system usefulness, etc.), and measurable objective user behavioral consequences. The latter two are gathered via surveys. Wu et al. correlate survey results with measured QoS variables as users perform tasks. In their study, significant correlations between factors were found. They use a "causal chain" starting with the environment, then focusing on cognition, and lastly, user behavior. Technology acceptance is a cognitive factor, with adoption as a behavioral consequence. Additional behaviors include task performance and exploration within the DVE system.

2.3 Performance Engineering Processes

Software Performance Engineering (SPE) (Smith and Williams 2002) is an iterative process for evaluating design alternatives and performance objectives. SPE is UML based, and it uses UML notation and terminology for the process, models and analysis. To model the system, SPE uses *execution graphs:* flow-charts of major steps of the program's execution, each annotated with best-case resource usage. An execution graph can be generated by tracing over the vertical axis of a UML sequence diagram. Then, some hierarchization is applied to provide the appropriate level of abstraction.

The Rapid Object-oriented Process for Embedded Systems (ROPES) process (Douglass 1999) uses standard UML meta-model notation for its semantic framework and notation, and has four (serious) phases. ROPES is a general, iterative process that can be used for primary software development. It includes performance and safety as requirement domains alongside traditional functional ones. ROPES is intended for use in embedded and real-time contexts.

3 PROBLEM DESCRIPTION

DVE systems are resource-intensive systems with many complex performance requirements, complex user load, and many ways to change performance and resource use. The overall dependency, a loop, between qualitative (user experience) and quantitative aspects of DVE systems, called the "Causal Chain," (Wu, Arefin, Rivas, Nahrstedt, Sheppard, and Yang 2009) links Quality of Service (QoS) attributes in an environment

(user interface usability, responsiveness, and navigation options), the user's perceptions of the system, and the user's behavior when using the system. Figure 1 shows a DVE system in the Causal Chain.

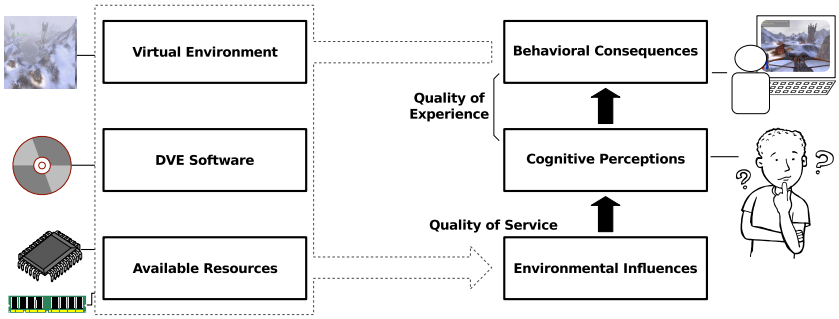


Figure 1: A DVE system in the Causal Chain.

Figure 2 provides a more detailed description of the Causal Chain that includes and map QoS values to the DVE system components. We have filled in connections between the components of the Causal Chain, and the components of the DVE system. Unfortunately, we don't have any information on what or how the DVE system components interact with one-another.

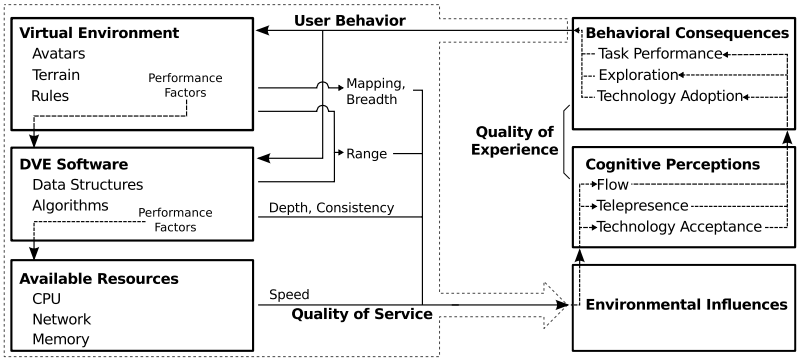


Figure 2: Detailed relationships between the DVE and the Causal Chain.

The left side of Figure 2 includes all the elements of a DVE system. At the top is the virtual environment: the rules governing what users can do in the environment, the virtual space, and the media assets. Some combination of these elements, coupled with the way users act because of them, may be major factors in the performance and quality of the system (abbreviated to "Performance Factors" in Figure 2). The specific combinations of the virtual environment's elements and the users' behaviors that become major performance factors are unknown. The second block is the DVE's software implementation. This includes all the data structures, algorithms, utility and runtime components, and the system design and architecture. In this block may also be major factors that determine the performance and quality of a DVE system. The final block on the left side of Figure 2 is the set of hardware resources available to the DVE system. The available CPU cycles, memory, and network bandwidth are given as examples, but any available resource may constrain the DVE system's scalability.

Figure 2 shows that the DVE system's Quality of Service is determined by a set of measurable attributes that include user interface choices, input devices, output media, and performance characteristics of the DVE system. The user experiences them all as a combination, and changes in one attribute's metrics can

affect the desired values of other attributes' metrics. The user's response to the experience, in terms of actions within the virtual environment and specific inputs given to the DVE software, affects the amount of work the system must execute to maintain the virtual environment. The cycle complicates the three major parts of performance engineering normally used to scale systems: determining the current level of performance, understanding the controlling factors of its performance, and changing performance.

4 PROPOSED APPROACH

We researched the requirements for all three stages of scaling DVEs: user load testing, data interpretation, and modification. The three major challenges are effective DVE system performance measurement, understanding the controlling factors that determine system performance/quality and determining the consequences of DVE system changes. We refined the DVE Scalability Engineering (DSE) process that addresses these three major challenges for DVE design.

Our strategy was to develop an easy-to-run simulated user load experiment with deep instrumentation. We developed a customized performance-engineering process around the experiment. We also took an existing model basis for DVE system performance and extended it for use within the process. The DSE process allows us to identify, evaluate, and leverage trade-offs among the available DVE system resources, the DVE system components, DVE system software, and the virtual environment. The DSE process substantially extends Software Performance Engineering (SPE) (Smith and Williams 2002) by integrating new user load testing and modeling methodologies. As a consequence, it is possible to explore and analyze the effects of changes in the DVE system design on the DVE system performance and quality of experience. The findings help us understand the primary scalability factors and help us make informed DVE system design decisions.

It is important to keep all the DVE system components in our study as realistic as possible. The environment, users, software, and runtime environment should be realistic. Consequently, we start with a real DVE system and attempt to build a realistic load simulation system for it. To develop the experiment, we start with the most complex part: human behavior. We run user studies, characterize the observed behavior and then modify a copy of the DVE software to emulate the observed behaviors.

With a mechanism for accurate user load testing, we next move into instrumentation for the DVE system to complete our platform for DVE system analysis. Using the data from the instrumentation, our modeling basis, and modeling method, we build a calibrated model of how the DVE system uses its resources. By doing this, the part of the DVE system software using the most resources is identified, and additional instrumentation and simulated-load experiments help to identify the controlling factors.

Singhal and Zyda (Singhal and Zyda 1999) defined the Networked Virtual Environment (NVE-IP) Information Principle relating key characteristics of a DVE system's structure and workload to the DVE system resource needs. The model, however, is insufficient for scaling DVE systems because it does not help in interpreting a DVE's use of any given resource. Also, the model does not provide guidance on the elements unique to a DVE that could be changed to improve resource usage, such as a data structure, space in the virtual environment, or tools available to users. It also fails to provide adequate information on the performance structure of the DVE.

We extend NVE-IP and apply it to each of three primary resources: CPU time, memory, and network bandwidth. The equations developed for each primary resource have variables/components that are directly measurable in a DVE system. These equations provide the modeling basis.

Using this modeling basis, we have defined an iterative modeling process that traces resource use from top-level DVE system software structures down to individual software components. The process traces the resource it finds most constrained. For example, the process follows CPU usage down the function call stack of each thread. Similarly, memory usage follows the instance-graph of allocations. The tracing process follows resource use up the relevant hierarchy until it identifies primary factors.

The process works thanks to a new instrumentation tool `ppt`. The `ppt` tool is designed for high-rate, low-overhead instrumentation of systems with a focus on enabling outlier and correlation analysis in and

between developer-selected groups of variables. Combined with offline analysis, the process and tool can quickly find dominating factors in resource use.

4.1 Process for DVE Scalability Engineering

Once we identify performance factors, we can experiment with changing the virtual environment or DVE software. The factors may directly reduce resource use or change user behavior to indirectly reduce use.

We started with an existing engineering process, Software Performance Engineering (SPE) (Smith and Williams 2002) and adjusted it as needed. SPE is an iterative process based on determining the performance feasibility of a design. We replaced the initial analysis with an accurate simulated user load experiment, and specialized its iterations into several "cycles" designed to more accurately build the model and experiment with changes in both the DVE software and virtual environment. The DSE process (Singh, Gračanin, and Matković 2012b) is shown in Figure 3.

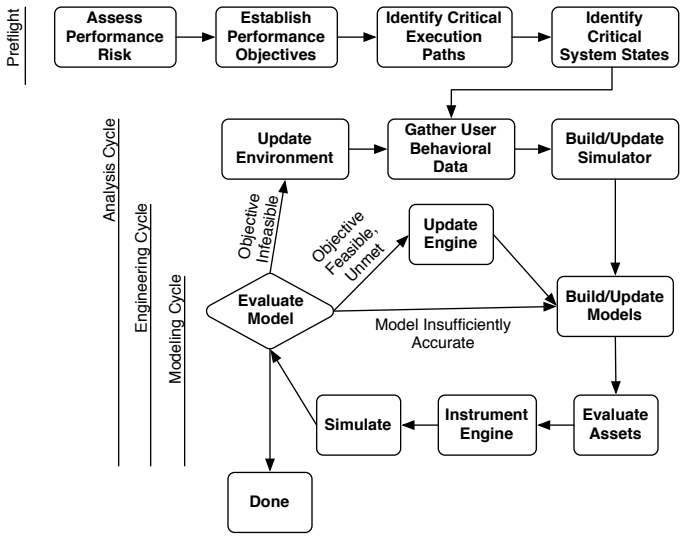


Figure 3: DVE Scalability Engineering (DSE) process.

We have developed a custom low-overhead performance instrumentation tool called `ppt`. Traditional instrumentation tools are not intended to help the explorative correlation analysis used by DSE process' modeling cycle. In that cycle, almost arbitrary parameters of the DVE system are measured or recorded together for the sake of correlation or causal discovery.

We also added a new instrumented user load test experiment, modeling methodology, and engineering process for DVE systems. The user load testing simulates users logging into an instrumented version of the DVE system. The modeling methodology identifies the primary factors that control the DVE system's resource usage. The process incorporates the user load testing and modeling methodology and uses them as a basis for experimenting with changes in the DVE system.

The user load testing experiments simulate the users (i.e., real users' behavior in the virtual environment), and are based on user study observations. The user load testing experiment can run autonomously at arbitrary scale, limited only by available computer resources. The experiments can also be relatively quick to run, with a batch of tests runnable in a few hours. The instrumentation gives data for the modeling methodology, and runs with minimal overhead.

Over multiple user load testing experiments, the modeling methodology measures the parts of the DVE system that use the most resources. The instrumentation tool collects ad-hoc data at high rates (multiple kHz) and lets us run sophisticated analysis after the user load test. Over each cycle, the modeling methodology

uses the instrumentation to follow resource usage from the top-most of the DVE system software to the individual software components that use the most resources. The correlation analysis is used to identify factors that control those software components' resource usage. The factors can be in software, or in interactions between user behavior and the DVE system software.

4.2 User Load Testing Experiment

The DSE process user load testing experiment has two parts: the user load simulator and the DVE system software instrumentation. The user load simulator acts as a dynamic model of the DVE system users's behavior, the experiment acts as the test of the model, and the instrumentation acts as a view of the results.

For an accurate simulation, we should include a sufficient number of users and their individual and group tactics and strategies. This approach is unusual because user load simulators for DVE systems often have randomized user activity or pre-recorded sessions played back (at user counts different from the recorded session). Our behavioral-simulation approach lets us directly experiment with behaviors when running user load testing experiments. Figure 4 shows the user load testing experiment. The "Fix Problems" stage may change how users behave in the system, so we have two options: one that re-acquires user behavioral data, and one that doesn't.

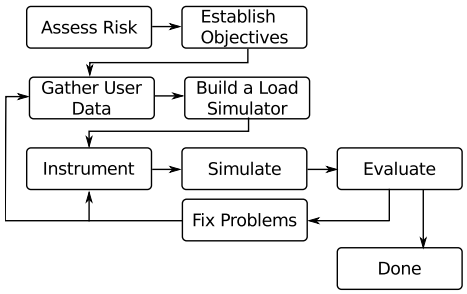


Figure 4: A DVE scaling process with user load simulation experiments.

We partially replicate human behavior for a realistic DVE system user load. User behavior is purposeful, imperfect, tactical, strategic, and social. Our approach attempts match it.

Simulator construction starts with a user study ("Gather User Behavioral Data"). The user study has users acting as normal as possible within the DVE system. Their actions are recorded by the DVE system's "demo recording" mode, normally used for demonstration and debugging purposes. This mode records all data in and out of a user's session in the DVE system, and can play them back later. Next, a DVE system designer/developer plays each demo back. The developer categorizes the observed user activity into behaviors and records the time breakdown of each behavior in the recorded session (for all the recorded sessions). The resulting data forms a behavioral distribution usable for constructing the user load simulator. The user interface, virtual environment, interactions, and active virtual objects need testing inside the DVE system for validation. Once the data is collected, we move to the "Build/Update Simulator" stage, take the data and turn it into a user load simulation component.

Our simulation approach is to reapply the existing work. Substantial work in realistic Non Player Characters (NPCs, also referred to as AI characters) already exists. That work includes techniques for chasing other users, path-finding, swarm movement, and goal selection. We implement each observed behavior in a modified version of the DVE system software that serves as a user load simulator. The user load simulator emulates user input and reads the DVE system's local view of the virtual environment to execute the user behaviors. It selects these behaviors according to the tabulated distribution. To reduce the computing resource costs of the user load simulation, the user load simulator can also have its graphical rendering and user input and sound systems disabled.

We run one instance of the simulator for each user that we want to simulate. With availability of commodity virtual machines (cloud computing), we can run simultaneous user load simulations with simulated users at relatively little cost. The user load simulator needs only be accurate enough to match the instrumentation data we would get from the equivalent (in terms of the number of users) user study. For small numbers of users, the user load simulator is less likely to significantly represent the actions of users. For simulations with more simulated users, we expect that the underlying population sample used for the behavior model to become more representative of the expected behavior. Alternatively, the user load simulators can be artificially tuned to cause some excessive user load in a different mode, to test the DVE system's ability to handle exceptional user situations.

5 CASE STUDIES

There are several commercial open source DVE systems that are real-time (versus turn-based) such as include Torque (Maurina III 2008), Quake III, and Quake IV. Torque was used as the basis of the Tribes II game on desktop computers, and is now used as the basis of several downloadable XBox 360 games. Quakes III and IV were both released directly as games. We used Torque and a maintained version of Quake III called ioquake (Nussel, Schulz, Angus, White, and Slater 2014), two of the most popular DVE systems with publicly available source code and significant real-world use. Our DSE process was used for the case studies to establish a simulated user load testing experiment, iteratively model the DVE system to identify primary performance factors, and experiment with changes to the DVE system.

5.1 Torque

Our hypothetical scenario is a deployment of many instances ("shards") of a Torque-based (Maurina III 2008) DVE system. We use the example "starter" virtual environment definition that comes with Torque's development kit. We initially aim for supporting instances of sixty users each. The "starter" virtual environment is a tiny virtual village, which is crowded with sixty people.

We want to fit as many instances we can onto our hypothetical deployment hardware. After we fit all of our desired users, we can try to alter the DVE system to fit more users in each shard. We want to keep the total round-trip-time (RTT) between a client and server below 60ms, and that includes the time spent between the server's reception of client data, processing, and transmission of data back. Torque uses a client-server architecture, and we focus on the server. The clients have less work, except for rendering.

We ran two user studies to get data for a user load simulator. In each, we put study participants in a computer lab and asked them to play the game for an hour. Torque's built-in mechanisms recorded every message sent or received. We played back each recording individually, and noted the user's behavior in sub-minute intervals. We recorded each behavior/tactic used with its frequency. We implement each behavior using a state machine. The "starter" world is very simple: one weapon, a small village, and only "kills" counted for points. There were small health packs scattered across the area. We modified the "starter" to ensure that every player had unlimited ammunition.

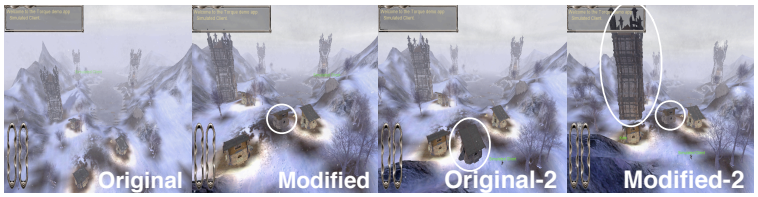
5.1.1 Modeling

Our initial model is empty and we did not know which resource is a bottleneck. It could be either CPU, memory, or network bandwidth. In order to identify the bottleneck resource, we ran a user load simulation.

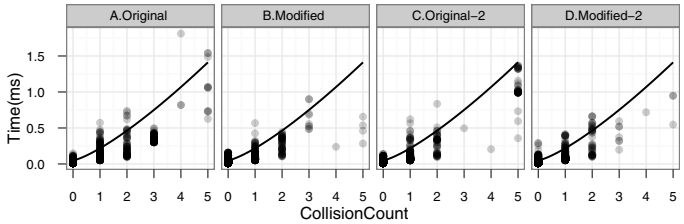
We time each run of Torque's primary software loop's four parts: the core loop, input processing, physical simulation of the virtual environment, and output.

We instrument the two types of virtual objects we think may take up a lot of simulation time: user's avatars and the projectiles they fire. The former has complex movement and collision detection. There are many instances of the latter that may add up. Another simulation gives us clear results. 72% of all CPU time in the system is spent simulating users' avatars, and 1.2% is spent on the projectiles they fire. Between these two types of virtual objects, and the simple synchronization measurement, we have

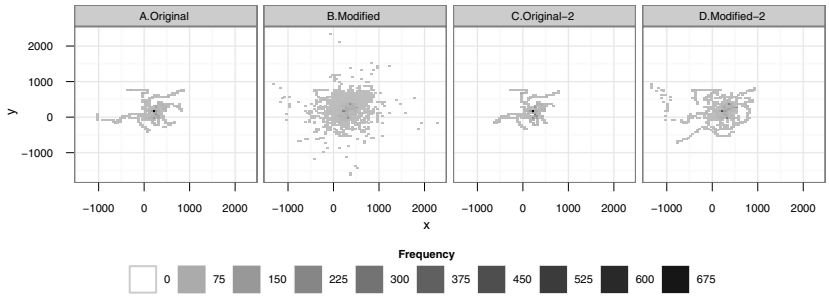
almost ninety-percent of Torque's CPU usage covered. We also know the biggest part of our work: avatar simulation. However, that doesn't give us anything actionable. We conducted an analysis of terrain and how the changes in the terrain affect the distribution of players and collision count (Figure 5).



(a) Level Screenshots.



(b) Collision Detection vs Collisions (N=30), with Model Overlay.



(c) Original (A), Intentionally Modified (B), and Randomly-Modified (C,D) Level Densities, 5000 Randomly–Sampled Points Each.

Figure 5: Level Variants, Collisions, and Densities.

There are four density maps. The grid size used for all four density maps is large enough to hold exactly one avatar at a time. Each density map is taken from a random sample of 5,000 simulate frames for avatars. The first ("A. Original") is the "starter" terrain we have now.

In the density map for our current ("Original") terrain, we can see a single cell substantially darker than the rest, at approximately (250; 200). That is near the center of the virtual village. As an experiment, we put a small building there. We call the resulting new virtual environment "Modified."

As controls, we created two additional variants. The first one is "Original-2," with a new building at a different part of the village. The second variant is a second variation upon "Modified," denoted "Modified-2." It keeps the new building from "Modified" and adds a tower to the periphery of the village.

In terms of reducing areas of high density, the intentional modification alone seems the most effective. In terms of overall computational load, the original average time through the main software loop was 10:66ms, and the modified variant 10:36ms — a tiny three-percent improvement.

The variance in simulation time was reduced to 20:65ms^2. The ninetieth percentile is substantially lower at 23:99ms, a 21.73% improvement. The other two variants had worse mean runtimes, 10:91ms for "Modified-2" and 10:99ms for "Original-2." The variances were 86:05ms^2 and 87:32ms^2, respectively. The additional buildings, placed poorly, seemed only to increase collision pressure in the region.

5.2 Quake III

Quake III has two hard-set limits on how many users it will hold. Globally, a Quake III server can have a configurable maximum that defaults to twelve. Each level also has their own immutable maximum set for in-game balance and available virtual environment. With the existing body of work on the Quake series, we looked for modern concerns on an older engine. Mobile devices are an interesting new deployment option for multi user DVE systems. We aim to understand Quake's networking characteristics, for applicability to the different wireless technologies available on a modern mobile phone.

We ran user studies as a regular weekly game with Google employees using their own computers. Users were constantly running and shooting at one another on the ground, mid-jump, or mid-fall. Most of them had played the game before, a few of them were experts.

We found a fast-moving, tactics-driven playing style across all users. Each one varied greatly in aim, speed, and coverage of the map. When not tracking one user and observing the entire sequence executing in parallel, a simpler structure emerged. A fast cycle between four primary behaviors was found: chasing another user, moving to a new place, getting a health kit, or a new weapon or ammunition. We found that the relative percentages for these were 40%, 30%, 10% and 20%. Through observation, we estimate the mean behavior duration as roughly a second and a half.

5.2.1 Load Simulator

A state machine implements each behavior. Each has three sub-states: INIT, MOVE, and POST. The first (INIT) sub-state selects behavior-specific state, the second (MOVE) executes any necessary movements, and the final (POST) completes any final activity after having moved to the destination. The simulator selects a behavior and runs it for one and a half seconds. Next, it increments a behavior-specific *count*, and a global *behavior counter*. After the increment, it determines the *relative representation* of each behavior as the ratio of its count versus the global behavior counter. Finally, the simulator selects a new behavior.

5.2.2 Instrumentation

We added instrumentation to Quake III to record information about every packet sent or received from the running DVE system. This includes a high-resolution time stamp, whether the data was being sent or received, and the amount of data transferred. The instrumentation was inserted at each `send(2)` and `recv(2)` call. We applied Equation 1 singularly to the bandwidth dimension (Equation 2):

$$\text{Bandwidth} = X_M \times B_X + R_M \times B_R \tag{2}$$

X_M and R_M are the message transmission and receive rates, respectively. We break B term of Equation 1 into B_X for the average amount of bandwidth needed for transmission, and B_R for received data. The instrumentation measured the total bandwidth used.

5.2.3 Modeling

We ran a 15-user simulation and collected a little over 2.5 million values over a short (9.3 minute) interval. We instrumented the server only. Two percent (51,120 I/O records) was lost in the process. The basic statistics of the collected data show that while constantly sending with only an average $54\mu s$ between packet transmissions, there was a fairly bursty receive interval in clusters centered over 8.043ms.

We calculated the variance, standard deviation, and one-sided 90^{th} percentile value for both the intervals and byte sizes. The percentile was calculated using a variation of Cantelli's inequality: $P(X-\mu \geq k\sigma) \leq \frac{1}{1+k^2}$, with $k = 3$. We were able to determine that a 15-user Quake III session would require a mean 2.60 $kBps$ send transfer rate, but a 619.09 $kBps$ receive rate. Balani (Balani 2007) gives transmission power formulas for three networks: Bluetooth 1.1, 802.11, and GSM/EDGE. While not listed in the report, we presume from the relevant time period that it discusses 802.11b. We found two options: one case of using 802.11b

for its low network latency, at the cost of a substantial power consumption requirement, or GSM/EDGE at ten to twenty percent of the power consumption, with a cost of over ten times the network latency.

6 CONCLUSION

We tried to identify and quantify the relation between user behavior and system performance using the Casual Chain described in Section 3 to provide a basis for an iterative process for scaling DVE systems. We built on top of the existing techniques for user load testing and developed a way to accurately simulate specific user populations. We combined the user load simulation system with a custom performance measurement tool to develop a rapid, iterative modeling methodology, a DVE Scalability Engineering (DSE) process. The DSE process allowed us to find the major performance factors in a DVE system while minimizing the effort related to testing, modeling, and changing the DVE system to enhance its performance.

We demonstrated the utility of the DSE process for scale-related analysis in two real-word cases studies: Torque and Quake III DVE systems. The Torque case study illustrated the use of DSE process for scalability issues (in terms of the number of users). The Quake case study demonstrated scaling a DVE system for use on a mobile phone platform. We analyzed Quake III to model the network requirements that were used as a basis for network latency/network bandwidth/power consumption trade-off analysis. The DSE process has also been shown useful in a very small DVE system, Asteroids. We believe the DVE process can be effectively used for very large DVE systems with large number of users.

The DSE process provides a foundation for exploring new research challenges and opportunities to better understand the dependency between performance (QoS) and scale (QoE). This, in turn, could lead to better understanding of a global optimal balance in QoE, QoS, user tasks/goals in the virtual environment, and required DVE system resources. Applying the DSE process to various DVE systems will help us identify the common trade-offs and DVE system design patterns for scalability.

REFERENCES

Balani, R. 2007. "Energy Consumption Analysis for Bluetooth, WiFi and Cellular Networks". Technical report, University of California at Los Angeles.

Baños, R. M., V. Guillen, S. Quero, A. García-Palacios, M. Alcaniz, and C. Botella. 2011. "A virtual reality system for the treatment of stress-related disorders: A preliminary analysis of efficacy compared to a standard cognitive behavioral program". *Int. Journal of Human-Computer Studies* 69 (9): 602–613.

Bylund, M., and F. Espinoza. 2002, January. "Testing and Demonstrating Context-Aware Services with Quake III Arena". *Communications of the ACM* 45 (1): 46–48.

Connelly, K. 2007. "On Developing a Technology Acceptance Model for Pervasive Computing". In *Proceedings of the Ubiquitous System Evaluation (USE) — A Workshop at the Ninth International Conference on Ubiquitous Computing (UBICOMP)*, 177–183.

Douglass, B. P. 1999. *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Reading, Massachusetts: Addison-Wesley Professional.

Fishwick, P. A. 2009. "An introduction to OpenSimulator and virtual environment agent-based M&S applications". In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, 177–183. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Friedman, D., Y. Karniel, and A. L. Dinur. 2009, August. "Comparing Group Discussion in Virtual and Physical Environments". *Presence: Teleoperators & Virtual Environments* 18 (4): 286–293.

Fullford, D. 1996. "Distributed interactive simulation: it's past, present, and future". In *Proceedings of the 1996 Winter Simulation Conference*, edited by J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 179–185. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Laghari, K., and K. Connelly. 2012, April. "Toward total quality of experience: A QoE model in a communication ecosystem". *IEEE Communications Magazine* 50 (4): 58–65.

Lazem, S., D. Gračanin, and S. Harrison. 2012, 21–25 May. "On the Relationship between Changes in Distributed System Behavior and Group Dynamics". In *Proceedings of the 2012 International Conference on Collaboration Technologies and Systems (CTS 2012)*, 345–353.

Lindeman, R. W., D. Reiners, and A. Steed. 2009, March-April. "Practicing What We Preach: IEEE VR 2009 Virtual Program Committee Meeting". *IEEE Computer Graphics and Applications* 29 (2): 80–83.

Liu, H., M. Bowman, W. A. Hunt, and A. M. Duffy. 2012. "Enabling behavior reuse in development of virtual environment applications". In *Proceedings of the 2012 Winter Simulation Conference*, edited by C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, 1–12. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Maurina III, E. F. 2008. *Multiplayer Gaming and Engine Coding for the Torque Game Engine*. Wellesley, Massachusetts: A K Peters, Ltd.

Nussel, Ludwig and Schulz, Thilo and Angus, Tim and White, Tony J. and Slater, Zachary J. 2014. "ioquake3". Accessed July 1, 2014. http://ioquake3.org.

Pittman, D., and C. GauthierDickey. 2010. "Characterizing Virtual Populations in Massively Multiplayer Online Role-playing Games". In *Proceedings of the 16th International Conference on Advances in Multimedia Modeling*, 87–97. Berlin, Heidelberg: Springer-Verlag.

Singh, H. L., D. Gračanin, and K. Matković. 2012a. "An Approach to Tuning Distributed Virtual Environment Performance by Modifying Terrain". In *Proceedings of the 11th International Working Conference on Advanced Visual Interfaces (AVI 2012)*, 628–631. New York: ACM.

Singh, H. L., D. Gračanin, and K. Matković. 2012b. "Software Scalability Engineering for Distributed Virtual Environments". In *Proceedings of the 5th Workshop on Software Engineering and Architectures for Realtime Interactive Systems — IEEE VR 2012 Workshop (SEARIS@VR2012)*, 52–59.

Singhal, S., and M. Zyda. 1999. *Networked Virtual Environments: Design and Implementation*. ACM Press SIGGRAPH Series. Reading, Massachusetts: Addison Wesley.

Smith, C. U., and L. G. Williams. 2002. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison Wesley.

Wadley, G., and N. Ducheneau. 2009, 7–11 September. "The 'out-of-avatar experience': object- focused collaboration in Second Life". In *Proceedings of the 11th European Conference on Computer Supported Cooperative Work (ECSCW'09)*, edited by I. Wagner, H. Tellioğlu, E. Balka, C. Simone, and L. Ciolfi, 323–342. London: Springer.

Weilbacher, M. 2012. "Dedicated Servers in Gears of War 3: Scaling to Millions of Players". In *Game Developers Conference*.

Wu, W., A. Arefin, R. Rivas, K. Nahrstedt, R. Sheppard, and Z. Yang. 2009. "Quality of experience in distributed interactive multimedia environments: toward a theoretical framework". In *Proceedings of the 17th ACM international conference on Multimedia*, MM '09, 481–490. New York: ACM.

AUTHOR BIOGRAPHIES

H. LALLY SINGH is a Software Engineer at Google. His research focuses on the design of distributed virtual environment systems and related performance issues. His email address is lally.singh@gmail.com.

DENIS GRAČANIN is an Associate Professor in the Department of Computer Science at Virginia Tech. His research interests include virtual reality and distributed simulation. He is a senior member of ACM and IEEE and a member of AAAI, APS, ASEE and SIAM. His email address is gracanin@vt.edu.

KREŠIMIR MATKOVIĆ is a Senior Researcher in VRVis Research Center, Vienna, Austria and an Adjunct Associate Professor at Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. His research interests include virtual reality, human computer interaction, tangible user interfaces and human perception. He is a member of IEEE Computer Society, ACM, and Eurographics. His email address is matkovic@vrvis.at.