

Sun™ Game Server Technology

An Executive Overview
June 2004



© 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, CA 95054 USA

All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California.

Sun, Sun Microsystems, the Sun logo, and Java are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a). DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS HELD TO BE LEGALLY INVALID.



Please
Recycle



Adobe PostScript

Table of Contents

Executive Summary	1
The Target Developer	1
The Target Application	1
Design Points	2
Game Server Architecture	2
Traditional Massively Multiplayer Game Architecture	2
EverQuest and the State of the Industry	2
Limitations of Existing MMP Game Solutions	2
Sun's Game Server Technology	3
The Sun Game Server Architecture	3
Benefits of Sun Game Server Technology	4
Conclusion	5
For the Game Developer	5
For the Game Operator	5
For More Information	5

Executive Summary

Sun is developing new technologies to address the needs for low-latency, high-bandwidth, fault-tolerant, highly scalable simulation services. Although designed with the massively multiplayer, networked game developer in mind, it could be reasonably deployed anywhere where such requirements are paramount.

The Target Developer

The primary target for a simulation services system is the game developer. The game developer has a very different and unique set of needs, skills, and understandings compared to Sun's traditional corporate developer.

- *No Threading Knowledge.* Game developers live in a world of mono-threaded, discrete, time-based simulations. For this reason, Sun™ Game Server technology does not require any knowledge of threading or threading issues to successfully program and deploy simulation content.
- *No Database Knowledge.* Game developers typically load all the data they need for a given “level” at the start of that level. (A level is one unit of continuous player interaction.) They have no experience with transactions, deadlocks, query optimization, or any of the other myriad database complexities that enterprise developers regularly deal with. For this reason, it is paramount that the system make any database interaction 100% transparent to the game developer.
- *Little Experience With Architecting Scalable Systems.* Scalability solutions in the game world today are still very primitive. In general, players are divided up into groups that only interact with each other. These groups still have scalability limits that result in hard and arbitrary limits on the number of players that can play together. This limits game design in ways that strongly limit the type of game that can be built and the manner in which it can be represented.

A key value add of Sun's Game Server technology is that it frees the game developer from these limits and allows much more flexibility and creativity in game design. In doing so, it must off-load all the scaling issues, making them transparent to the user.¹

The Target Application

Games fall into the realm of near-realtime simulations and thus, have some very stringent performance requirements. Sun's goal of supporting massively multiplayer games adds extreme scalability and latency requirements. To be successful, a massively multiplayer simulation system must meet these challenges:

- *Very Low Latency.* Total latency overhead for all activity in the back end other than developer-provided game code must measure in the tens of milliseconds per interaction.
- *Scalable From Hundreds to Tens of Thousands of Simultaneous Players.* The system not only has to handle a high end of 10,000+ of users, but must scale economically from that case all the way down to a system supporting a few hundred online users at a time. As different games typically go through different speeds of growth curves, there must be an economical way to grow from the low end to the high end either slowly or very quickly as user needs demand.

1. It is nearly impossible to build a system that will run bad code with maximum efficiency. Although virtually any code will run correctly in the Sun Game Server, there are a very few, very simple best-practices the developer should follow in his or her data-structure design to get the best scalability performance from the system.

Design Points

These goals for developer features lead directly to the following design points in Sun's Game Server technology.

- *Simple Application Model.* An application consists of a “world” of serializable simulation objects. These objects have their methods invoked to perform actions upon them.
- *Automated Execution Model.* Simulation objects are invoked in an event-driven, race-proof, deadlock-proof, fault-tolerant manner. All the issues of object contention and failure handling are dealt with by the execution environment and are transparent to the SO code.
- *Transparent Continuous Persistence.* All details of acquiring simulation objects for execution, completing the execution, and saving the results are handled by the execution environment. Every game action is durable within moments of its completion.
- *Transparent Massive Scalability.* Execution of simulation objects is transparently spread across a horizontally scalable array of independent processors with independent memory spaces — ranging from a very small number of processors to a very large number.

Game Server Architecture

Traditional Massively Multiplayer Game Architecture

In order to understand Sun's Game Server technology, it helps to have a high-level familiarity with what has come before. Those very familiar with current massively multiplayer (MMP) game architectures can skip this section.

EverQuest and the State of the Industry

The MMP games of today are designed to support thousands of simultaneous players, and require a centralized server environment in which game computation is distributed across multiple processes on multiple host machines. These games, for the most part, are based on a software architecture pioneered by EverQuest. The games are typically divided into “Areas” or “Regions” (sometimes referred to as “Zones”) in which each Region runs its own process on its own server, while managing state in its own memory space.

A user logging onto such a game starts by connecting to a login server. Once authenticated, the client is instructed to disconnect from the login server and connect to a starting Region server (if this is a new player) or to the last Region server to which the player was connected (if this is a returning player). When a user moves to a different Region, their client is again instructed to drop the current connection and connect to the server for their new Region.

Limitations of Existing MMP Game Solutions

Limited Scalability

Regions still have a limit on how many users they will allow to connect at once. When that limit is reached the Region is “full” and will not allow new players in until a current player leaves. This puts an absolute cap on the number of users this model can support.

Sony gets around this limit by duplicating the entire system many times. (Sony calls each duplicate a “Shard.” Shard has become familiar industry terminology for a duplicated massively multiplayer game session.) However, the problem with this scaling solution is that it splits the user base, first by Region and then by Shard. It multiplies customer service costs and makes it harder to maintain critical online mass of users in any given Shard. Furthermore, players who play on a full Shard have no incentive to encourage friends to play because they may not be able to play together.

Minimal Fault Tolerance

The Shard model is not very fault tolerant. If the login server of a Shard goes down, players cannot connect until it comes back up. If the current Region server goes down, players in that region are stuck and cannot play until it comes back up.

Inefficient Use of Processors

The Region model would divide load perfectly if people naturally spread themselves out in a Gaussian distribution. Unfortunately, they don't. We are social animals and social animals "clump" together.

The end result is that there are often processes just sitting idle maintaining their state while few to no players are actually using the attached processor. Furthermore, load on a given Region process can shift from nothing to high load very quickly, so you cannot just deploy multiple server processors to a single host CPU. Each process has to be capable of handling its full load at any time. The result is large-scale processor waste.

Limited Persistence

Generally, what is persisted to disk from the memory image is just player state. Regions, therefore, are highly static; they cannot change their geometry because they must be able to be reloaded from original geometry for failure recovery at any time. A Region's inventory (those objects dropped by players or monsters and lying "on the ground") generally does not persist through a process restart. Monsters are "spawned" as part of the runtime behavior of the region process and thus also do not persist through a restart.

The granularity for disaster recovery of player state at best tends to be about 15 minutes because the system has to more or less stop what it was doing to do a backup. In worst cases, a failure can wipe out 24 hours of character updates.

Sun's Game Server Technology

Sun's Game Server technology radically changes the landscape for programming massively multiplayer games. It off-loads all the issues of scaling, fault tolerance, and persistence from the game developer, giving them what appears to be a single, event-driven, multithreaded programming environment. Programs that run within this environment, however, are automatically scaled across multiple processors and are automatically persistent and fault tolerant.

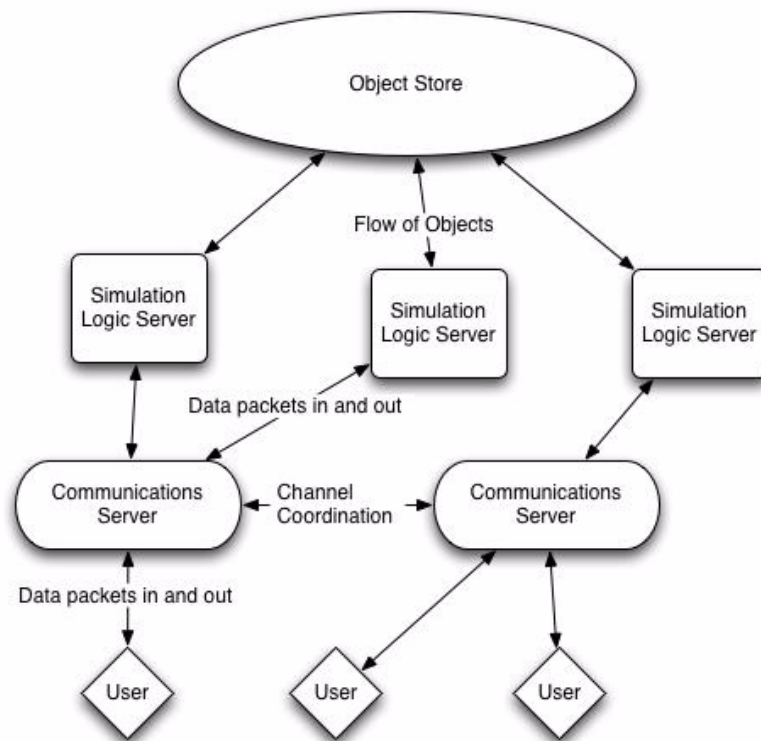
The Sun Game Server Architecture

Sun's Game Server technology logically is divided vertically into 3 layers: Communications, Simulation Logic, and Object Store.

The **Object Store** layer contains the game states for all games running in the Game Server. It is a highly efficient (tenths of a millisecond per operation), scalable, and fault-tolerant transactional database layer that provides deadlock proof access to the simulation objects, which can either be locked (a write-lock) or peeked (a nonrepeatable read).

The **Simulation Logic** layer is responsible for executing the actual game code. Here, tasks are created based on incoming events which, in turn, check objects out of the Object Store as needed. When a task is completed, the object is updated and returned to the Object Store.

The **Communications** layer organizes player communication into channels of grouped communicators. It manages routing of data packets between the players and the Simulation Logic servers, and between the players themselves. It also is responsible for translation to and from other forms of networking (e.g., HTTP communications to and from cell phones).



Benefits of Sun Game Server Technology

Extreme Scalability

By utilizing a highly-efficient and scalable common object repository for game data, Sun's Game Server technology eliminates the need to divide the world along regional boundaries, making Regions and Shards obsolete. Games may be created as a single, unified, seamless environment that can support tens of thousands of concurrent players. Developers can continually modify and expand the contents of the world while enabling the game operator to quickly deploy new servers as needed to meet increased demand — all without disrupting existing game play for connected users.

Highly Fault Tolerant

When a Simulation Logic server or Communications server goes down, game events and player connections are automatically redistributed to other servers, eliminating the downtime commonly associated with login server or Region server failure. This downtime is one of the commonly cited sources of player dissatisfaction with existing game services.

Maximum Processor Efficiency

By distributing the workload based on player activity rather than regional divisions, Sun's Game Server technology provides automatic load balancing, not just within a single game, but among all games running within the same data center. This drastically reduces the cost of redundant, backup, and idle compute server resources.

Total Persistence

With Sun's Game Server technology, all objects — not just player state — are reliably persisted. This provides the developer with greater flexibility to create highly dynamic environments while eliminating the players' frustration that comes from seeing their hard work wiped out by momentary server failure or operator error. Both player satisfaction and customer loyalty are increased.

Conclusion**For the Game Developer**

Sun's Game Server technology offers a radically simplified programming model for massively multiplayer game developers. At the same time, it offers a degree of fault tolerance and scalability akin to Sun's enterprise products. By using the Game Server, a game programmer is freed from all the issues surrounding server-side programming and left to do what he or she does best — write great game logic.

For the Game Operator

Sun's Game Server technology is capable of securely running the logic for many different games simultaneously. It will automatically load balance not just within a single game, but among all games running within its structure. This provides maximum CPU efficiency for game epicenters, where many games are all run by the same service provider.

All games running within a simulation back end have shared administration needs, thus allowing one administration team across all running games. A Java™ technology-based coding standard ensures that game logic is safely contained and cannot effect the rest of the network environment. It also protects investments by enabling operators to deploy future game products into the existing back end.

For More Information

If you are interested in learning more about how this technology may fit into your network game development plans, please contact us at gametech@sun.com.

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA Phone 1-650-960-1300 or 1-800-555-9SUN Web sun.com



Sun Worldwide Sales Offices: Argentina +5411-4317-5600, Australia +61-2-9844-5000, Austria +43-1-60563-0, Belgium +32-2-704-8000, Brazil +55-11-5187-2100, Canada +905-477-6745, Chile +56-2-3724500, Colombia +571-629-2323, Commonwealth of Independent States +7-502-935-8411, Czech Republic +420-2-3300-9311, Denmark +45 4556 5000, Egypt +202-570-9442, Estonia +372-6-308-900, Finland +358-9-525-561, France +33-134-03-00-00, Germany +49-89-46008-0, Greece +30-1-618-8111, Hungary +36-1-489-8900, Iceland +354-563-3010, India-Bangalore +91-80-2298989/2295454; New Delhi +91-11-6106000; Mumbai +91-22-697-8111, Ireland +353-1-8055-666, Israel +972-9-9710500, Italy +39-02-641511, Japan +81-3-5717-5000, Kazakhstan +7-3272-466774, Korea +822-2193-5114, Latvia +371-750-3700, Lithuania +370-729-8468, Luxembourg +352-49 11 33 1, Malaysia +603-21161888, Mexico +52-5-258-6100, The Netherlands +00-31-33-45-15-000, New Zealand-Auckland +64-9-976-6800; Wellington +64-4-462-0780, Norway +47 23 36 96 00, People's Republic of China-Beijing +86-10-6803-5588; Chengdu +86-28-619-9333, Guangzhou +86-20-8755-5900; Shanghai +86-21-6466-1228; Hong Kong +852-2202-6688, Poland +48-22-8747800, Portugal +351-21-4134000, Russia +7-502-935-8411, Saudi Arabia +9661 273 4567, Singapore +65-6438-1888, Slovak Republic +421-2-4342-94-85, South Africa +27 11 256-6300, Spain +34-91-596-9900, Sweden +46-8-631-10-00, Switzerland-German 41-1-908-90-00; French 41-22-999-0444, Taiwan +886-2-8732-9933, Thailand +662-344-6888, Turkey +90-212-335-22-00, United Arab Emirates +9714-3366333, United Kingdom +44-1-276-20444, United States +1-800-555-9SUN or +1-650-960-1300, Venezuela +58-2-905-3800, or online at sun.com/store

SUN™ © 2004 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Information subject to change without notice. v3.0 06/04 R1.0