OpenMosix for the creation of low-latency, high-efficiency game clusters

Carlo Daffara,
Conecta srl cdaffara@conecta.it

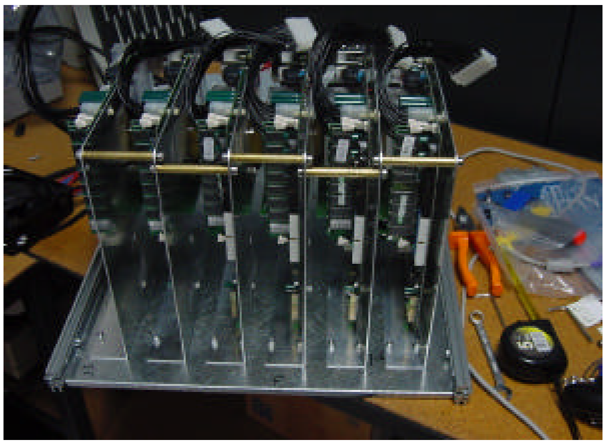
In the variegated world of software there are many niche markets of significant value. One of this niches is online gaming, exemplified by blockbuster games like ID software's Quake III or Sony's EverQuest; popular online gaming sites report more than 10 million players, on 600000 servers, while EverQuest has 433000 paying gamers (for a total of 5 Million dollars/month in revenues).

These games are structured in a strictly client-server style, where the client is the game installed on the player's PC or console, and the server maintains the status, the interaction of players with the environment and the unraveling of the game story. Traditionally, the client/server interaction is maintained using many small UDP packets; most server systems use differential prediction techniques and latency-hiding systems to preserve a smooth playing even in presence of network congestion, packet loss and uneven traffic loads. For a high quality experience, the maximum total ping (round trip) time must be below 150 msec, and the game server must gracefully degrade performance under high load.

The extreme, real-time like requirements and the wide variation in the number of players force the carriers that want to provide this service to host a large number of distinct machines, each hosting a few players per tournament. All the major providers host from 200 to 500 servers, with a players/server ratio of 17. This is a considerable burden, as most of the servers are chronically underutilized, and the large number of servers poses significant maintenance and hardware costs, that makes most of these ventures unfeasible from an economical point of view.

As a system integrator, we got the contract to build a prototype of a high-efficiency (in terms of player/server density) and high-quality (in terms of perceived user experience) game server, designed to host around 1000 concurrent players. We already have experience of Mosix and OpenMosix in the realization of a medium scale video broadcasting system, and so we started development of a custom hardware/software combination that can easily be deployed to telecom carriers.

Our final hardware design consists of 6 Athlon XP 1.8 Ghz systems, built from standard off-the-shelf motherboards. Each board has 768Mb of ram, a dual 100Mbit ethernet connection for performing channel bonding (in a newer version of the cluster we substituted them with single channel copper gigabit boards) and boots through the PXE network boot protocol; the system will be sold in the beginning of 2003 as a universal “tile” for assembling OpenMosix clusters.


The packing system

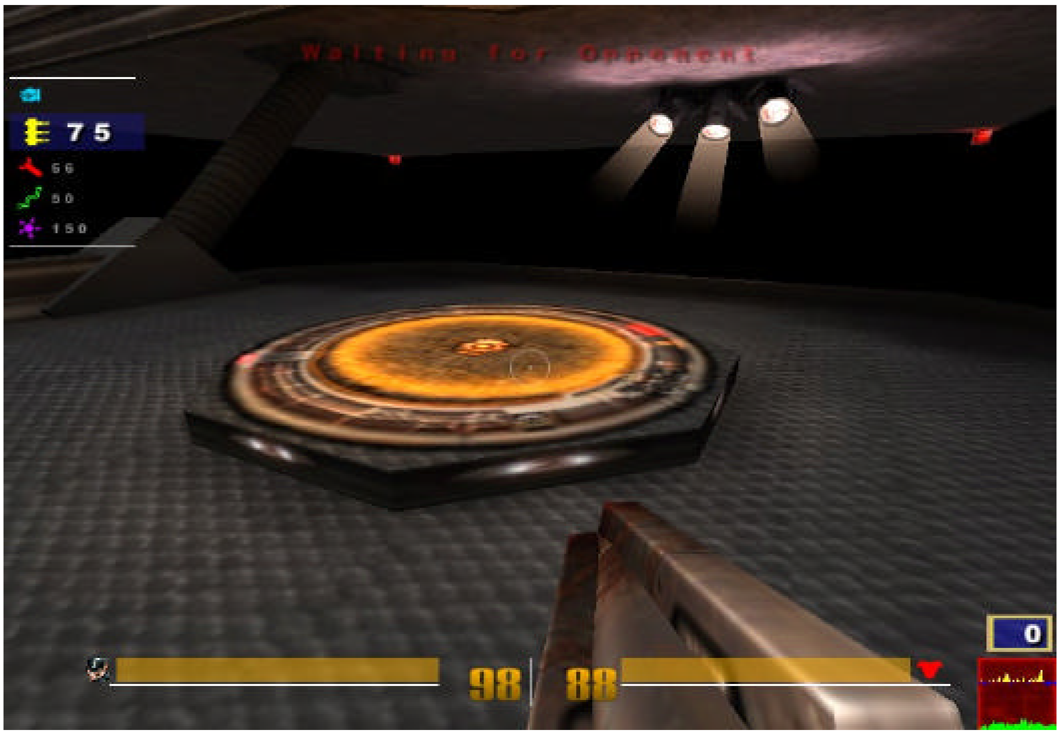
..and partially assembled cluster

The master node has twin IDE disks in software mirroring mode, under a custom linux distribution originally created for high availability servers. Since some tasks use shared memory and thus cannot be transparently distributed, we replicated most of the distribution to every node's NFS-mounted root. These tasks are then manually started on a user-assigned remote node, and an iptables-based proxy forwards the packets in and out of the master server (the only one with a real IP address). Large files are not copied, but replicated through hard symlinks to avoid unnecessary disk waste, especially considering that large game maps can easily exceed a gigabyte in size.

Since the small spacing between the boards, and the high temperature load of the Athlon cpus, we had to implement a simple temperature load monitor for all the boards. To this end, we used the linux lmsensors package, that monitors the onboard temperature and voltage parameters. Each machine has a very simple cron script that execute the sensor check every 5 minutes, and writes the results to a file in the /tmp; thus, on the master node, all temperatures can be checked from the master simply by looking into the /tftpboot/<client_ip>/tmp/temp-out.

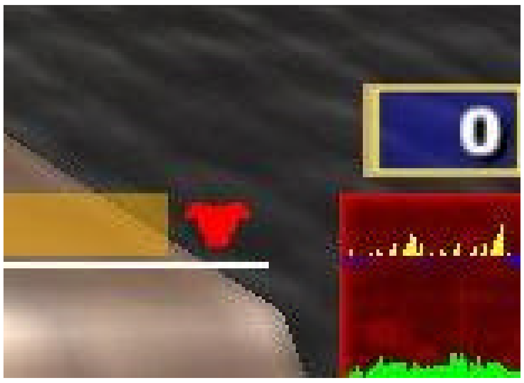
Another problem appeared during testing: since the game server memory footprint is large (around 80 Mbytes each), we discovered that the migration of processes slowed down the remaining network activity, introducing significant packet latency (especially perceptible, since packets are very small). So, we used the linux iproute2 queue controls to establish a stochastic fair queuing discipline to the ethernet channels used for internode communications; this works by creating a set of network “bins” that host the individual network flows, marked using hashes generated from the originating and destination IP addresses and the other part of the traffic header. The individual bins are then emptied in round robin, thus prioritizing small packets over large transfer and not penalizing large transfers (like process migration).

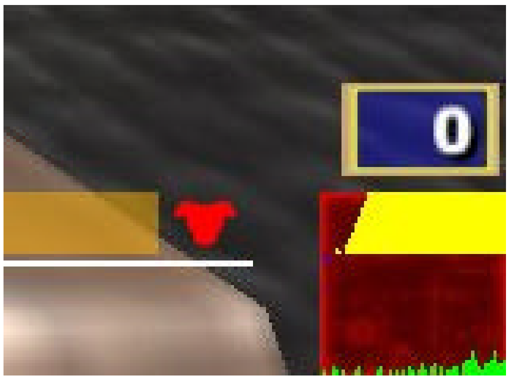
The effect of this prioritization process can be seen in the following screenshots of in-game play (with the included network latency display):



Ingame screenshots: actual game (top), with stochastic fairness (bottom left), without SFQ (bottom right)

The top indicator is the server delay in answering to client packets; the bottom indicator is the client ping time (round trip time).
It is quite clear that the packet discipline is considerably improving responsiveness even in presence of substantial internode traffic; this is useful for other realtime-like applications, like video on demand, streaming and large scale instant messaging applications.





References

[PW2001] LaPointe D., Winslow J. "Analyzing and Simulating Network Game Traffic", Worchester Polytechnic Institute, Computer science department, MQP MLC-NG01, available at http://www.cs.wpi.edu/~claypool/mqp/net-game/

[Feng2002] Feng W., Chang F., Walpole J., "Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server", In Proceedings of the Internet Measurement Workshop, November 2002.