Rezvan 3D Development Team Description

Ali Nadalizadeh, Hessameddin Akhlaghpour
[nadalizadeh@gmail.com, hessam.akhlaghpour@gmail.com]

Computer and Robocup Science Department,
Allame-Helli high school, Tehran, Iran

Abstract. The new SimSpark server was released in order to guide the efforts of Robocup3D teams toward developing more realistic tactics and techniques in soccer. Our main activity has been divided into two distinct parts. The first part is to develop a system to speed up the server and make it possible to conduct a game of two teams with eleven agents each, without noticing a decrease in the servers performance. The second part of our activity is to develop a web interface to display log-files generated by 3D simulation server. This would make it easy for ordinary users to view previously held matches online. You can examine the details of the proposed ideas and methods by reading this document.

1 Speeding Up the Server

The new released SimSpark server can only handle a few agents simultaneously. By increasing the number of concurrent agents, you can observe a rapid increase of time required to process each frame. Running 6 or 7 agents overwhelmingly slows down the server, let alone 22 agents. As long as this server can only handle a small number of agents, team tactics can't be developed. This problem calls for the need of an improvement that allows a large number of agents to connect to the server with no or minimum impact on the server's performance. Our goal is to develop efficient methods to make it possible to run 22 simultaneous agents, without witnessing a noticeable decrease in the server's speed.

1.1 Observations

We have conducted an experiment with the server where we run a bunch of dancer agents and record the average time elapsed between two subsequent cycles. (A dancer agent is a humanoid agent that continuously moves its limbs regardless of what the servers sends to it; Therefore it consumes practically no time for CPU processing.) We expected to see a polynomial trend in the order of n where n is the number of agents. Surprisingly we observed an exponential trend(2 to the power of n) order of magnitude impact.(See figure 1)

The number of times that the collision handling function is called for two objects (in each frame) is linearly related to the number of agents. This result was as anticipated and has an intuitive explanation that not so many collisions occur between robots. Obviously, at most two or three robots are colliding with

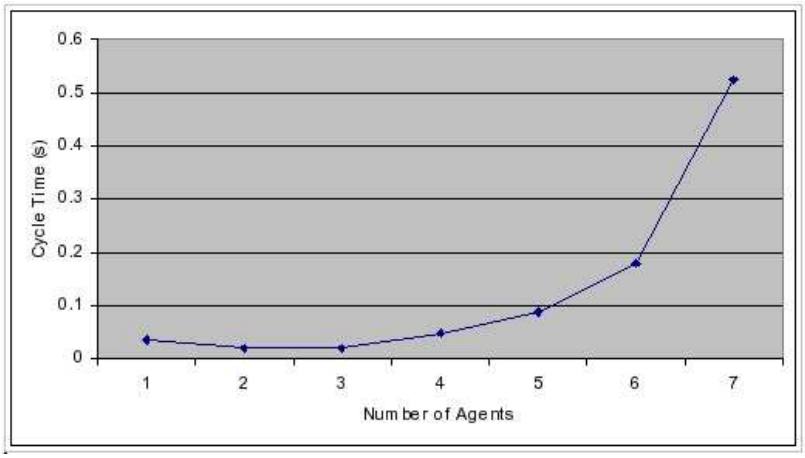

Fig. 1.

each other at the same area. This shows that the reason that the server slows down is not because of physical simulations. Graphics rendering is also not the cause, as the computer we used had graphics rendering hardware acceleration. The memory the server consumed was also a constant value, in spite of adding new agents. This means that there is no memory mismanagement in the server code.

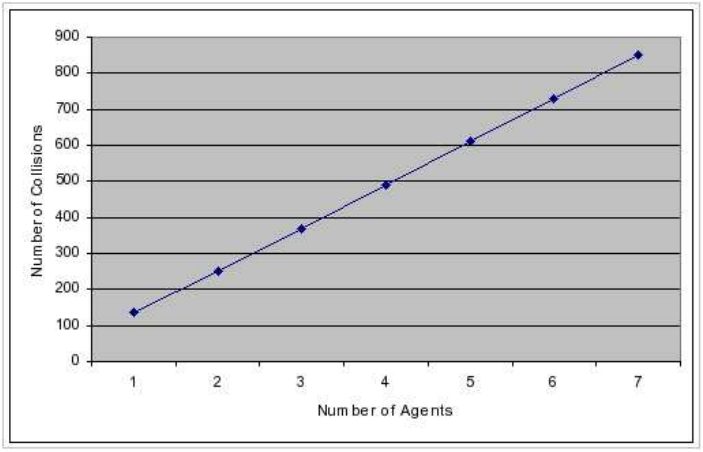

Fig. 2.

1.2 Parallel Processing Overview

As the observations had shown us, the robots are usually far apart from each other. This becomes more evident when the soccer field grows larger. This gives us an idea where we can divide the soccer field into independent areas and simulate them independently. Using parallel processing will significantly speed up the server. As we had observed, there was no problem running a small number of agents on a single computer. All we have to do is to balance the load between several computers on a network.

More specifically the server could consist of a number of simulation servers (slaves) and a main server (master). The master server distributes the physical objects between the slaves. Notice that these objects are independent and do not touch each other. This means that the slave servers simulate their objects independently. After each server finishes its job, it sends the new states of its objects to the master server, which merges the objects and creates the complete soccer field. As you can see in figure 3, the agents contact the server by directly sending and receiving messages to and from the master server. Since the only interface between the agents and the server is by connecting to the master server, the agents wont feel the structure change. Depending on the network congestion we will provide the option to compress the data to avoid network traffic.

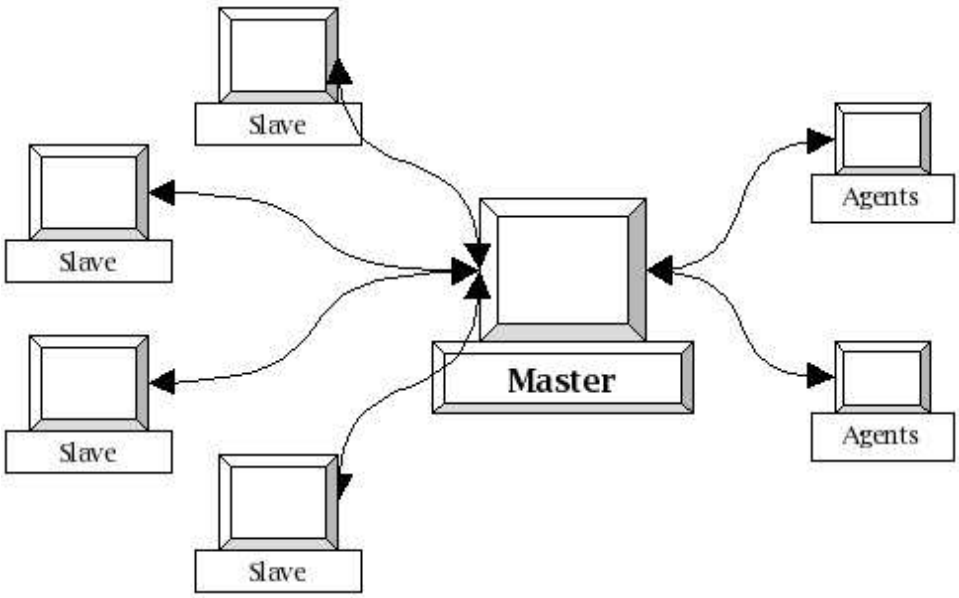

Fig. 3.

1.2.1 Distribution Algorithm The main server must distribute the objects between the simulation servers and merge the results of their simulations. It is important that the main server does this quickly and uses fast algorithms. Merging the objects is obviously O(n) . Distributing them is a bit harder. We will present an O(nlogn) algorithm to divide the agents into separate groups.

Consider each agent by its x and y positions. We can draw a square in the 2D field that contains the agent inside. This square is specified by a simple heuristic estimation. The problem is reduced to finding independent sets of squares; sets that have no intersections with each other. The solution to this problem is a quad-tree. A quad-tree data structure is capable of solving this problem in an order of nlogn.

1.3 Considering Other Physical Simulators

ODE has been a good choice but there might be better alternatives for the servers physics engine. Some other good candidates are Bullet, PAL, and OPAL which are all open-source. We may also want to consider some closed-source (but free) software such as Newton Game Dynamics.

2 Browser-Embedded Log-Player

It has been a long time that users have been looking for an easy-to-use log-player to see the previously held matches, without installing the whole simulation server. Developing such software will be a major leap in developing monitor applications, as the Robocup society will be able to present their matches to ordinary users via Internet. As last years 3D-Development competitions had shown, Macromedia Flash Player isnt capable of simulating such complex and heavy 3D simulations. This problem will grow as we see more complex servers being developed (such as SimSpark). Conclusively Flash isnt a good option for displaying the 3D games to ordinary users. This year, the Rezvan 3D-Dev team is proposing a good replacement: A light browser plug-in. Our goal is to develop a plug-in that is capable of displaying Robocup games, whether sphere players or humanoid players.

2.1 The Plug-in Structure

As you may know, a plug-in for a certain browser is a program that handles certain types of files. A small plug-in for Robocup log-files can use the underlying OpenGL native 3D libraries to let the browser draw every frame of the game much faster than a flash player. The source code of the rendering part can be very similar to the current 3D-moniter.

2.2 Advantages and Disadvantages

The size of our plug-in is relatively small (approximately 350KB) and it is very easy to install. The only dependency that it has is OpenGL, and most computers

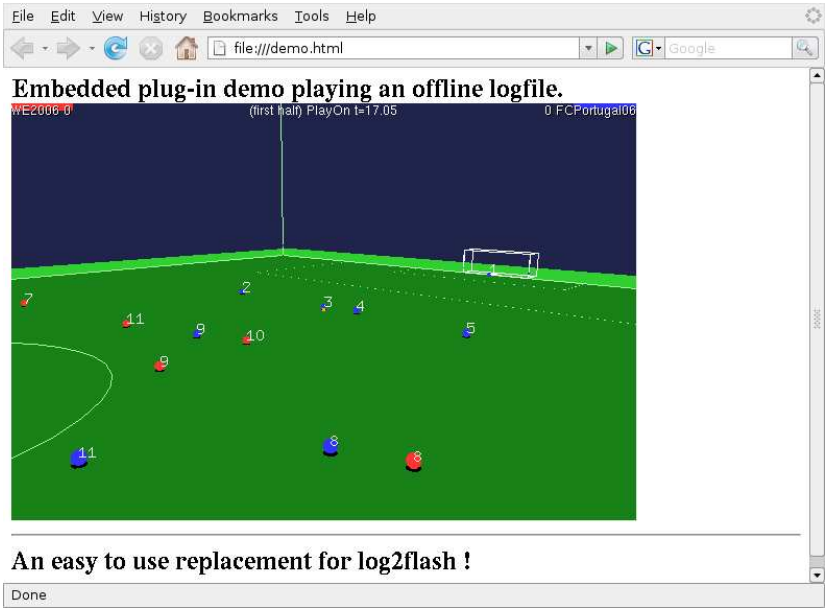

Fig. 4.

have it installed already. The plug-in development is synchronized with the server and monitor development, because the source code of the rendering part is very similar to that of the Robocup server rendering. In addition, our plug-in takes advantages of graphics rendering hardware acceleration to render whereas Flash Player may not be able to use. The only disadvantage that our plug-in has is that ordinary users trust Flash Player more than a plug-in, and might not prefer to download our software.