RoboBase: An Extensible Framework Supporting Immediate Remote Access to Logfiles

John A. Sear and Rupert W. Ford

Centre for Novel Computing
Department of Computer Science
The University, Manchester
M13 9PL, United Kingdom
{jas,rupert}@cs.man.ac.uk

Abstract. This paper describes RoboBase, a system that provides immediate access to entire libraries of RoboCup logfiles. A centralised database stores the logfiles allowing them to be viewed remotely. Instead of downloading a 2MB uncompressed logfile, the match is transferred and displayed in real-time.
The system has been designed specifically to perform well in low bandwidth situations by using a domain specific compression method. Dynamic frame-rates are also employed, providing uninterrupted viewing in fluctuating network conditions.
The system conforms to an Object Oriented methodology and is implemented in Java allowing extension of the software by the user.

1 Introduction

The RoboCup Soccer Simulation League (SSL) provides an interesting environment for research in domains such as artificial intelligence and multi-agent collaboration. The SSL community evaluate their research through periodically held football competitions, the most prestigious being the annual Robo World Cup. The relative success of teams in these football competitions is typically taken as a measure of the advancement in the associated field of research.

A logfile is produced for every match held in these competitions and made publicly accessible. Logfiles record all the information required for visual replays of games (such as player and ball positions during the game and meta-information such as goals and team names). Logfiles provide a very useful way for team owners and their competitors to evaluate the strengths and weaknesses of teams. As such it is important that these logfiles are easily accessible.

Currently there is no official repository for these logfiles and they are typically spread across the World-Wide-Web (WWW); this makes locating a logfile a potentially laborious process. After a logfile has been located, its download can be relatively time consuming. These files are typically greater than 2MB uncompressed and are required in their entirety before viewing can commence.

A. Birk, S. Coradeschi, and S. Tadokoro (Eds.): RoboCup 2001, LNAI 2377, pp. 92–101, 2002.

This paper describes a central repository of logfiles, a logfile compression technique and a viewing technique, which allow near instant remote viewing of these files on low bandwidth connections, such as modems. These components have been developed using an Open-Architecture (OA) approach, facilitating modification or addition to any component. To illustrate the potential of this approach, a full working implementation has been developed and is freely available from `http://www2.cs.man.ac.uk/~searj6`. When evaluating teams there are many metrics (statistics) that may be used and different users will potentially have very different requirements. An OA approach is expected to be particularly useful here as provides a means to add new metrics.

This paper makes the following contributions: it develops logfile specific compression techniques, describes a new open architecture framework which can be easily extended, and introduces a Dynamic Frame-Rate Algorithm (DFRA) for graceful degradation of viewing in poor network conditions.

2 Related Work

There are many ways of viewing existing logfiles. LogPlayer is included as part of the SoccerServer system and replays the logfiles through the standard viewer (SoccerMonitor). It is also possible to connect other viewers to the SoccerServer, e.g. a 3D viewer: Virtual RoboCup [6]. However, stand-alone players are simpler alternatives and several have been developed.

RoboMon [3] is a java applet which has both a 2D and 3D (VRML) version. This is a commonly used viewer which has graphics that are simple but effective. LogMonitor [8] is a simple 2D Java applet/application viewer with relatively advanced game analysis (statistics) features. The LogMonitor website [10] suggests that new versions of the software may support some form of database, however the intention of this is unknown to the author. Windows SoccerMonitor 1.3[4] combines the SoccerMonitor and LogPlayer into a single program for Windows operating systems. It is currently the easiest option for viewing files within a Windows environment.

The above logplayers are able to play the logfiles at the standard frame-rate of 10fps and provide basic features, such as fast-forward and rewind. However, these logplayers are all limited in that the user must first trawl the internet to locate the file and then wait to download the 2-3MB logfiles[1] before play may begin. Furthermore all facilities are hard coded and the user is therefore limited to features implemented by the programmer at creation time.

Our approach solves the above problems by employing a central database, allowing easy access to logfiles, providing near immediate random access to games, avoiding logfile download times and being developed with an OA approach allowing new features to be added.

[1] These reduce to approximately 500k when compressed with gzip

3 Design Issues

Figure 1 shows the overall structure of the proposed system. Data may be stored in any database which supports the Java DataBase Connectivity (JDBC) protocol. An SQL command file generates a database capable of storing individual matches and information regarding the competition structure. The system enables the inclusion of individual games into the database but is also designed to allow entire competitions to be added at once.

The test data for the database was generated from all of the matches played in the Japan Open 2000 [1], a total of 7 groups and 53 games.

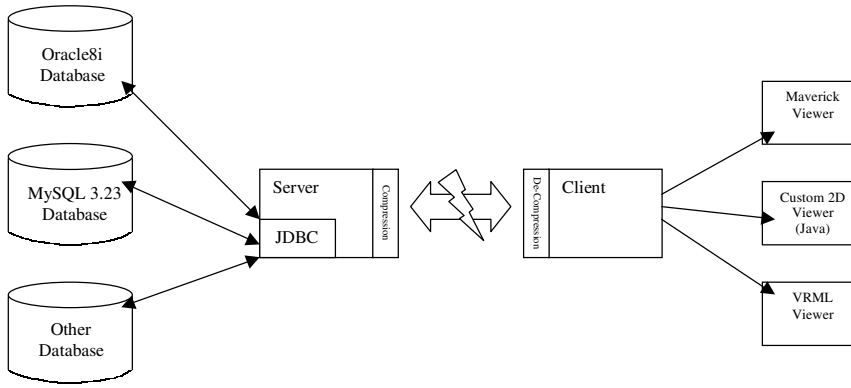

Fig. 1. Overall System Structure

The client application communicates with the RoboBase server via a compressed message format, which then expands these requests into full SQL statements. For example, when the server receives an 'M', this indicates that the next timestep block is required, and since the server has kept a record of the last transmitted block, it will be able to generate an SQL statement such as:

"select * from playertimestep where time>=400 and time<425;"

The data may be compressed before being sent through the communication medium to the client. The data is sent using a send and acknowledge protocol (rather than a constant byte stream) as it allows random access to matches and instant match changes.

At the client side, the first step is to decompress the data (if necessary) and store it in a buffer. Once the data is available, the client software analyses the data and visualises it using the chosen viewer.

The system (Client, Server and StoreFile software) are implemented entirely in Java. Java was chosen for portability as the developers of RoboCup clients use a variety of platforms. Its Object Oriented (OO) nature also means that it is particularly suited to the OA approach, through the extensibility of classes.

4 Implementation Issues

The system is built from a number of components. This section discusses those most relevant to this paper.

4.1 Client/Server

Once the decision was taken to store the data in a database, it was soon clear that the overhead associated with accessing the database directly, meant that real-time playback across low-bandwidth connections would not be possible.

This problem is solved by incorporating a server in between the database and the client. Its function is to remove the JDBC overhead and compress the data into a more concise format. In addition, the JDBC client code (150-500K) can now be excluded resulting in a substantial saving in the program size. Furthermore, this approach solves another problem. There is an applet security issue that needs to be considered, since applets are prohibited from connecting to an IP address other than that from which they were downloaded. Therefore, the server can act as the bridge between the client and database. An additional security benefit is that the the database can restrict access solely to a particular IP address, the server, thus removing the potential problem of unwanted direct connections.

4.2 Compression

In order to achieve the required viewing rate of 10fps through low bandwidth connections some form of efficient data compression is required.

Gzip has already been shown to reduce logfiles from above 3MB to below 1MB and is therefore a potential option. Java includes support to zip and unzip streams of data, hence, implementation of such a system would be relatively simple. However, for low latency of playback, RoboBase requires small blocks of data to be transferred. Under these conditions the compression ability of Gzip is reduced. The processing requirement also places a high burden at the client end, suggesting it might be too slow for real-time decompression.

Another solution is to send every nth frame and simply interpolate for unknown values. Again, this is relatively easy to implement and has the advantage that it requires little computational effort. However, it would introduce inaccuracies into the data at an early stage. This could cause problems in RoboCup as players are not bound by normal physical limits and are able to teleport around the field. Any generated statistics could become inaccurate, e.g. a goal may not be detected if the timestep is absent where the ball has crossed the goal-line.

An alternative technique is to transmit only the differences between frames in the data. Player positional prediction is an example of this. The current direction and heading of a player are used to calculate the next timestep position. Since the time between frames is only 0.1s, the differences are typically small and therefore the prediction will be relatively accurate.

Player prediction was chosen as the final compression routine because it provides the highest compression ratio, allows parameters to alter data size, can provide 100% accurate data and requires little computational effort.

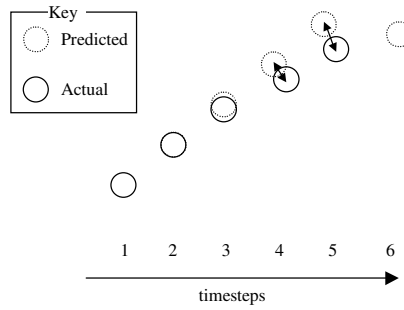

Fig. 2. Prediction Concept

Figure 2 illustrates the prediction technique. Timesteps 1 and 2 are used to calculate a heading and a velocity. The third timestep has only deviated very slightly. Timestep 4 shows a reasonably large deviation. In a real game this would be greater than the allowable tolerance, however, for illustrative purposes, the threshold has been increased. Timestep 5 shows a deviation greater than the threshold, therefore the real position must be stored. The predicted timestep 4 and the actual timestep 5 have been used to generate the predicted location in timestep 6.

It is possible to compress the data further by only transferring relative positions. If a player only moves a small distance between timesteps then this can be represented using a small number of bits. If a player has moved a considerable distance then their absolute location must be conveyed. The compression algorithm is depicted in Figure 3. The first bit defines whether the players are within the tolerance of the bitsize, i.e. if the data sent is relative or absolute. (In the example the bitsize is 4, therefore this tolerance is from -8 to +7). The next 23 bits correspond to which of the ball and players have been updated during this timestep. Each bit set corresponds to a player exceeding the prediction tolerance. After this the player co-ordinate data is listed, ordered by player number. The result is then separated into bytes which are sent across the network as a byte stream.

The compression parameters are the bit size of relative co-ordinates, the maximum player distance tolerance and the number of timesteps per message. These parameters were tuned by experimental examination of file size and accuracy. Future work will examine the possibility of dynamically tuning these parameters depending on network performance.

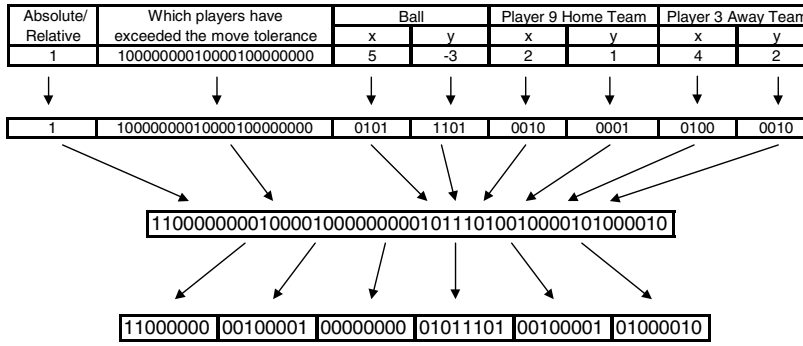

Fig. 3. Compression Algorithm

As the compression class is implemented in the OA framework, new compression algorithms can be added as appropriate.

4.3 Dynamic Frame-Rate Algorithm (DFRA)

If there is a delay in receiving data across the network, then the viewer may not be able to sustain the desired frame-rate of 10fps. A buffer is used to reduce the effect of network performance fluctuations, ensuring that saved data is available when the network is slow and filling up when the network improves. In poor conditions, using a buffer may not be enough to sustain the frame-rate.

Below are three possible approaches to the problem of poor network conditions:

1. Play at a constant frame-rate of 10fps. Once the buffer empties, the viewer pauses until more data arrives.
2. Reduce the frame-rate as the buffer empties. A potential problem is that when the next message arrives, the frame-rate suddenly jumps back to full speed.
3. Use a smoothed variable frame-rate. The frame-rate is reduced as the buffer empties, as above, however once the new data arrives, the frame-rate is gradually increased.

The third option was implemented as it provides the best results in terms of maximising the minimum frame-rate, and provides the smoothest (most pleasing) view to the user. The variables used to configure the DFRA include the buffer threshold, desired, maximum, and minimum frame-rates, which have been set as 50,10,10 and 2fps respectively.

Figure 4 shows how the frame-rate is reduced, once the buffer has less than 50 timesteps left. It is clear that the client has 18 seconds to receive the next message, not the 10 if it were displayed at a constant (full speed) frame-rate.

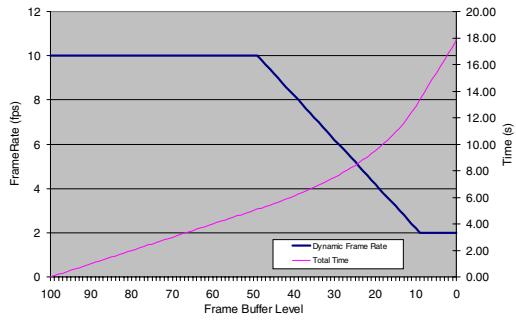

Fig. 4. Dynamic Frame-Rate - Buffer initialised with 100 timesteps

4.4 Statistics

The statistical analysis component is also completely configurable. As the RoboCup simulation community is large and diverse, it is impossible to target analysis for every users requirement. Instead, an extendible class is provided to allow users to implement their own statistics. It is also possible to use this class to interface with other software, such as commentators.

As an illustration the standard statistics given in Sky-Sports English Premiership televised matches are recreated and may be overlaid at anytime throughout the game. A separately windowed statistic, ManMarking, can be seen in the bottom right corner of the screenshot in Figure 5. Alternative metrics can also be graphed using this approach.

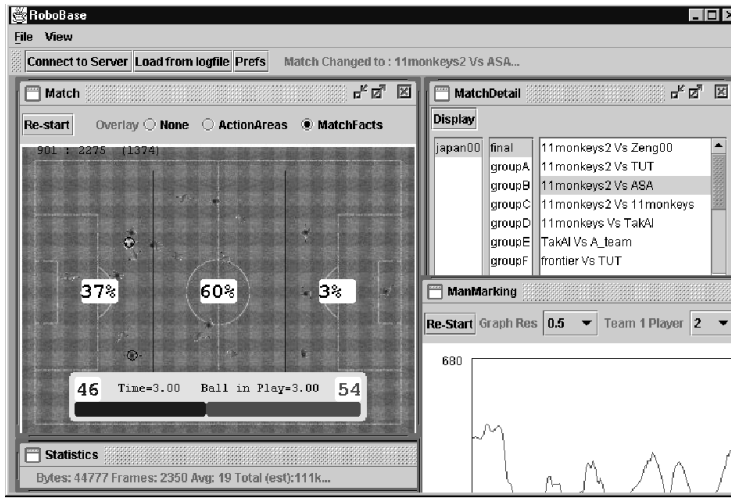

Fig. 5. Screenshot of RoboBase Client

4.5 Visualisation

The visualisation component is another completely configurable unit. At present three versions have been implemented. The first is a basic display using simple polygons to represent the match. The second is depicted in the Match Window of Figure 5 and displays a more realistic 2D view, using images to represent pitch, ball and players. The third employs Java3D to create a 3D view of the stadium and players. It is possible to use this class to interface to other viewing systems, such as VRML [9] using the External Authoring Interface (EAI) [5], or Maverick [7].

5 Experiments and Results

5.1 Dynamic Frame-Rate

In order to test RoboBase's playback performance, three client test conditions were created: a) a 450Mhz Pentium 3 connected to the Internet via a 44000bps modem connection, b) a 450Mhz Pentium 3 connected to the Internet via a 10Mbs ethernet connection and c) local playback of a file. The server is a MySQL database running on a Pentium 3 800Mhz system.

For each test the logfile was played in its entirety using three slightly varying compilations of RoboBase: a) visualisation enabled, b) visualisation disabled and c) visualisation and statistical analysis disabled. All these experiments were performed with the frame-rate limiter disabled. This combination of experiments allowed the frame-rate to be attained for user playback and the effect of the compression and analysis components to be assessed.

The timings for the locally stored logfile are included only as a guideline. This is because when playing back a local logfile, stored in the standard format, the prediction and compression routines are not required.

	Visualisation Enabled	Visualisation Disabled	Visualisation and Analysis Disabled
Modem (Ping time = 250ms)	33.84	34.09	34.20
Ethernet (Ping time<10ms)	57.51	63.42	65.79
Local logfile	59.04	74.06	74.93

Fig. 6. Frame-rate in test conditions

The results of these experiments are illustrated in Figure 6. For all the experiments attempted the minimum frame-rate attained was 34fps; this lower limit was demonstrated when using a modem with the visualisation enabled. This is significantly in excess of the desired 10fps. The limiting factor in these experiments was not the bandwidth of the modem but rather the time it takes to acknowledge a message (ping time). This suggests further experimentation should be performed by increasing the block size (timesteps per message).

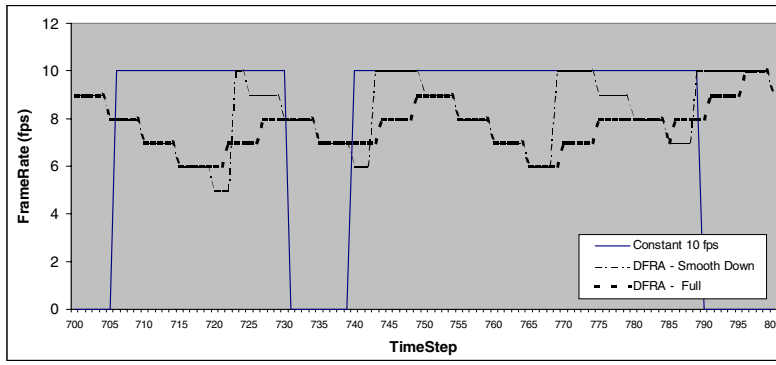

Fig. 7. Frame-rate in poor conditions

Further experiments were performed with numerous clients attached. A group of 10 400Mhz Pentium 3 machines running a mixture of Windows NT and Linux were all connected to the 800Mhz server. Whilst the frame-rate did decrease slightly, from the desired 10fps, with all 10 machines connected, the DFRA ensured a 'pleasing' playback, with no machines dropping below 5fps and average frame-rates above 7fps. This was due to the heavy load at the server end. This may be improved by placing the database and server on different machines.

In our final network test we manufactured poor network conditions, by adding a significant random delay to the response time of the server. Running the viewer at a constant 10fps resulted in many periods of display inactivity, see Figure 7. It can be seen that the DFRA ensures a constant playback and higher minimum frame-rate.

5.2 Compression

The matches from group F of the Japan 2000 RoboCup were played using the different compression ideas discussed earlier and the quantity of data transferred was recorded, see Figure 8. The benefits of the logfile specific techniques described in this paper are clear. In this case the file size is approximately 25 smaller than the original uncompressed version and 5 times smaller than the gzipped version. When downloading the Japan00 competition in its entirety, the logfile specific compression technique reduces the uncompressed 104MB of logfiles to to an estimated 5.5MB compared to 23MB with gzip.

6 Conclusions and Future Work

This paper has described Robobase, a system designed for near instant, remote viewing of RoboCup logfiles. It comprises of a database, storage utility, server and client which allows access to a library of games. Using compression and

Match	File Size (bytes)		
	Uncompressed	*GZ*	*RoboBase*
11monkeys Vs PSI	2,014,534	453,158	93,707
PSI Vs Revolvers	2,118,202	399,728	75,183
Revolvers Vs 11monkeys	2,130,638	415,562	75,837
ThinkingAnts Vs 11monkeys	2,030,326	521,122	103,742
ThinkingAnts Vs PSI	2,060,962	505,609	102,370
ThinkingAnts Vs Revlovers	2,137,918	355,270	69,337
Total	**12,492,580**	**2,650,449**	**520,176**

Fig. 8. Compression Results for Japan00 Group F

DFRA it is able to achieve frame-rates in excess of the desired, even over a low-bandwidth modem connection and in fluctuating network conditions. Examples have also illustrated how the open architecture structure may be extended if required.

The server and database support random access allowing retrieval of any timestep without incurring additional time penalty. This aspect is planned to be used to playback match highlights. Matches would be separated into obvious sections, e.g. when the ball goes out of play. Each section is then rated according to an interest value, such as a high rating for goals. Highlight programmes may then be produced by specifying either a time limit or a minimum interest setting.

The work described in this paper also forms the start of prototyping work for a new application area, 'Interactive Sports Entertainment', which provides both video and data to the home viewer [2]. The results shown in Section 5.1 demonstrate that it is possible to watch a representation of a football match over a low bandwidth connection. 3D rendering can then be used at the client end to provide realistic generated match views, allowing virtual cameras to be placed anywhere in the scene.

References

[1] Japan Open 00. `http://ci.etl.go.jp/~noda/soccer/JapanOpen00/`.
[2] Guy Blair, Rajeeb Hazra, and Richard Qian. *White Paper: Intel's vision of sports entertainment and marketing.* Intel Architecture Labs, 2000.
[3] Antonio Cisternino. Robomon, 2000.
[4] SoccerMonitor Klaus Dorer. `http://www.iig.uni-freiburg.de/cognition/team/members/dorer/robocup/`.
[5] External Authoring Interface. `http://hiwaay.net/~crispen/vrmlworks/`.
[6] B. Jung, M. Oesker, and H. Hecht. Virtual robocup: Real-time 3d visualization of 2d soccer games, 1999.
[7] MAVERICK. `http://aig.cs.man.ac.uk/systems/Maverik/index.html`.
[8] Tomoichi Takahasi. *LogMonitor.* 2000.
[9] VRML. `http://www.vrml.org/technicalinfo/specifications/vrml97/`.
[10] LogMonitor WebSite. `http://157.110.40.100/robocup/LogMonitor/`.