Demonstrating Hundreds of AIs in One Scene

Kjetil Raaen, Andreas Petlund, and Håkon Kvale Stensland

Abstract. This demo shows how a server can process heavy, game-enhancing workloads without sacrificing response time by implementing a highly parallelisable game server architecture. The architecture "LEARS" allows for the utilisation of more processing resources for tasks that will enhance the game experience. Using the A* path-finding algorithm, we demonstrate how the server distributes the resources under different levels of load, and how this impacts the response time for the game-clients. The demo allow the viewers to adjust game parameters such as the number of pathfinding entities, the number of player characters and the number of available threads and observe the impact on response time and CPU utilisation.

Introduction

Online massively multiplayer games have experienced a massive growth. Providers must deliver a stable service with short deadlines to please ever more demanding players. Technical limitations are however restricting the creativity of the designers. Increasing numbers of both clients and computer controlled opponents (often referred to as Non-Player characters, NPCs) reduce the available computing power. To alleviate this problem, the traditional approach is partitioning the game world and limiting the number of player and NPCs present (collectively referred to as entities) in each partition. This approach restricts the social part of the game, which is an important aspect in many modern games.

Another important part of experienced Quality of Service (QoS) by the players is response latency from the server. Chen et al.(2006) [2] present some numbers as to what is acceptable latency, by investigating average session length as a function of response time measurements. They examine an online RPG, *Shen-Zhou Online*. Between 45 and 75 ms RTT, they find a linear correlation between increased latency and decreased game session length. For the standard deviation of latency which would represent "jitter", the correlation is even stronger. To allow for some network latency, we suggest that servers should keep processing time below 20 ms.

A proposed solution, termed LEARS, that allow both a high number of interacting entities and heavy computation for each is demonstrated by Raaen et al. [4] and analysed in [5]. However, that work focuses primarily on extreme numbers of computationally cheap entities. The main costs in this work are collision between entities as well distributing information to clients. Potential uses of expanded access to computational resources on the server are numerous. The most computationally expensive tasks in games are usually 3D rendering of the world, and physics simulation. These, however are at the moment handled

more or less exclusively on the client, with the server only running an extremely simplified physics model. Moving physics, and basic awareness of how the world looks, to the server can allow players much more detailed interaction with the environment at the cost of extreme computational demands. By running physics simulations on the server, we also enable much more detailed checking of the rules, eliminating cheating. Conversely, have chosen to focus on Artificial Intelligence (AI) as one of the most demanding subsystems typically run server-side. Describing the most commonly used AI techniques, Narayek [3] lists the two categories decision engines such as Finite State Machines, as well as movement including pathfinding and steering. For the purposes of this demo, the latter approach has the great advantage of being easy to visualize. Other advantages include the ability to easily scale the cost of the problem, either by varying the number of entities running the pathfinding, the frequency of updates or the complexity of the maze. Among the movement algorithms discussed, we chose to use the pathfinding algorithm known as A*, which is the most common as well as most efficient approach for navigating in games.

This work demonstrates how the LEARS approach can provide near-perfect resource utilisation on a multicore server, allowing for game-improving heavy-duty computations like A* in a complicated maze to be deployed to enhance the game without sacrificing server response time.

Design and Implementation

Traditionally, game servers are based around a main loop, which updates every active element in the game sequentially. These elements include for example player characters, non-player characters and projectiles. Since all updates are sequential operations all actions can be performed directly. The character reads input from the network, performs updates on itself according to the input, and updates other elements with the results of its actions. All this makes parallel execution tricky and modifications to this approach such as [1] show only limited success, implying a need for a system designed from the ground up with parallelism in mind. The LEARS architecture is described in detail in papers [4] and [5].

The system demonstrated here uses a thread-pool executor as the core of the execution engine. When an active element is created, it is scheduled for execution. When it executes, the active element updates its state exactly as in the single threaded case, with the simple limitation that elements are not allowed to write to the state of other elements. When the task is finished for one time slot it can reschedule itself for the next slot, allowing active elements to have any lifetime from one-shot executions to the duration of the program. It also allows different elements to be updated at different rates, depending on the requirements from game designers.

To make elements in the game able to affect each other without the ability to write directly, inter-element communication must be implemented. Our solution is "blocking queues". These are queue implementations that are synchronised

Fig. 1. Setup of the demonstration

separately at each end. Operations on a blocking queue are extremely quick, so the probability of blocking is low. Each active object that can be influenced by others has a blocking queue of messages. During its update, it will read and process the pending messages from its queue. Other active elements put messages in the queue to be processed when they need to change the state of other elements in the game.

Additions to [5] are in the form of a large, scalable maze, and computer controlled entities that will path-find through that maze. Implementation of the map and pathfinder is based on Keith Woodward's "StraightEdge" [6]. Each entity has a separate copy of the maze, to avoid resource contention. Scripted players characters are still included. These do not perform pathfinding, as they simply follow a scripted route through the maze, shooting at predefined times. Their purpose is to be targets for the pathfinding NPC to track and follow. Each of these also need to keep track of the position of all other entities, continuously reporting this information to their controlling clients, keeping the focus on thread-to-thread communications. Each pathfinding entity needs to keep track of all target entities to choose which to follow.

Demonstration

The demo setup consists of three machines and is illustrated in figure 1. One machine is the game server. For this purpose, we are using a quad-core CPU,

which gives us enough parallel processing power to illustrate improvements over single threaded implementations. Another machine plays back a trace of the actual gameplay. Combined with the pathfinding entities, these will load the server to the desired level. The client simulation machine is not a bottleneck, since simulating clients requires very little calculation. The last machine is for running the real game client, so the users can see what is going on on the server and play the actual game. The setup also allows others to download the client and play the game during the demo session using a wireless LAN. As shown in figure 2 the game is a simple maze-game. The objective is to shoot opponents, computer controlled and player controlled, without getting hit yourself. NPCs will choose among the players, follow them and shoot if they get within range. If the NPC is shot three times, it will disappear for a while to reappear somewhere else. Hits on a player will simply be counted. Simplicity makes effects of the technology presented easier to visualise than more complicated games, and allows for a controlled environment to validate server architecture decisions.

The server control panel in figure 3 lets the experimenter control the conditions for running the game. It allows manipulation of the number of computer controlled opponents which will be the main factor loading the server in the demonstration. Next, it controls the number of scripted, emulated players. Another important control is the ability to dynamically regulate the number of available threads. Controlling the number of threads help experimenters evaluate how performance scales with the available parallel processing power. During the running of the game, the conditions on the server are displayed. Importantly the response time to clients is graphed, to show potential performance problems.

Fig. 2. Screenshot of the game

Fig. 3. Server control panel screenshot

Conclusions and Future Work

The setup demonstrated here show how our Lockless, Relaxed Atomicity State Parallel Game Server (LEARS) handles different conditions by adding and removing both computer controlled opponents, emulated players and real players. The demo allows us to thoroughly investigate how the system reacts when the described parameters are changed. Utilising this setup, we will be able to

study how different classes of hardware, such as cloud and clusters will handle games requiring complex computation. Highly parallel server environments such as LEARS allow for hundreds of game clients to operate within the same virtual instance. Such environments pose challenges to inter-player network communications since a large number of players need updates on all events in their immediate vicinity. How to effectively handle the network challenges will be a focus for our upcoming investigations in game-server technology.

References

1. Abdelkhalek, A., Bilas, A.: Parallelization and performance of interactive multi-player game servers. In: Proceedings of IPDP Symposium, pp. 72–81 (2004)
2. Chen, K.-T., Huang, P., Wang, G.-S., Huang, C.-Y., Lei, C.-L.: On the Sensitivity of Online Game Playing Time to Network QoS. In: Proceedings of IEEE INFOCOM 2006 (2006)
3. Nareyek, A.: AI in Computer Games. *Queue* 1(10), 58–65 (2004)
4. Raaen, K., Espeland, H., Stensland, H.K., Petlund, A., Halvorsen, P., Griwodz, C.: A demonstration of a lockless, relaxed atomicity state parallel game server (LEARS). In: Proceeding of NETGAMES 2011 (2011)
5. Raaen, K., Espeland, H., Stensland, H.K., Petlund, A., Halvorsen, P., Griwodz, C.: LEARS: A Lockless, Relaxed-Atomicity State Model for Parallel Execution of a Game Server Partition. In: Proceedings of ICPP 2012 Workshops (2012)
6. Woodward, K.: StraightEdge, <http://www.keithwoodward.com/straightedge/>