Is it Practical to Offload AI over the Network?

Jiaqiang Bai*, Daryl Seah†, and Ben Leong‡
Department of Computer Science
National University of Singapore

Existing games typically adopt a monolithic architecture, where the AI agents that control the non-player characters (NPCs) are run on a central server. While simple, scaling up to meet user-demands on such an architecture can be costly, especially for small-time game developers. One possible approach to improve scalability would be to offload the AI agents from the server to clients. This was first proposed by Douceur et al. [1], who suggested offloading AI in order to support increasingly complex agents in the context of massively multiplayer online games (MMOGs). However, their technique only partially offloads the AI to clients by splitting the AI into two components and offloading the more computationally expensive component to clients.

From our experience with game AI, we have found that structuring an AI agent to allow partial offloading is neither natural nor trivial for developers. In our work, we study the feasibility of *completely* offloading an AI agent from a server to a client. Ideally, we would like to be able to offload an AI agent that was written to run on a server to a client completely, with no or minimal modification. With a good understanding of the effects of network latencies, jitter and packet losses on the performance of offloaded AI and the development of practical techniques to mitigate these effects, we believe that significantly better AI agents can be developed for future networked games.

There are two key challenges in fully offloading AI to clients: (i) the state of the game world as perceived by the client is slightly outdated compared to the actual state of the simulation at the server; and (ii) there is a delay between the time when a command is issued by the AI agent running on the client and the time when the command is executed on the server. The key insight of our proposed solution is to employ a modified version of *dead reckoning* to predict the state of the simulation at the time of the expected command execution and to present this state of the game world to the AI agent. The agent can then decide on its next command based on this augmented state instead of the local state perceived by the client.

While dead reckoning is commonly employed in existing games to improve gameplay experience for human players, to the best of our knowledge, it has not been applied to AI agents, and little is known about how well the technique works and its associated limitations. Another limitation of traditional dead reckoning techniques is that only network latency is taken into account. Other factors like packet losses and commands issued by other players which are in flight are not considered. Human players can easily adapt to such factors, for example, by aiming some distance ahead of a moving target or repeating an action which the player thinks has not been performed. An AI agent, on the other hand, will not prima facie be able to do so. As such, our work will extend existing research on dead reckoning in a significant way.

Our solution is evaluated in the context of a tank game that advances the state of the simulation in increments of discrete time intervals called *ticks*. These discrete intervals are necessary because of the need to synchronize the commands from several clients at the server. The AI agent we used was developed as a term project for an undergraduate AI module at our university and is relatively sophisticated. In fact, average human players are typically unable to beat it.

The following is a brief description of our methodology: we start by estimating the *command response time* t_s, which is the time elapsed from the moment a command is issued at the client to the moment the client receives a response from the server. This quantity can change over time due to time-varying network conditions so we use an exponentially-weighted moving average (EWMA). From the estimate of t_s, we can then determine the number of game steps s that we need to predict in advance by dividing the estimated command response time t_s by the tick period.

Suppose a client observes the state W_c of the game world. We note that W_c is slightly outdated since there is

*Jiaqiang Bai, *student*, jiaqiang@nus.edu.sg
†Daryl Seah, *student*, darylseah@comp.nus.edu.sg
‡Ben Leong, *faculty*, benleong@comp.nus.edu.sg

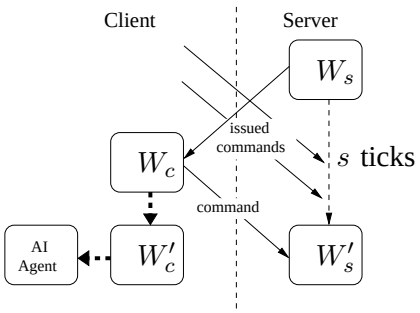

Fig. 1. Implementation of dead reckoning.

a delay before the state of the world object at the server W_s is received at the client as W_c. If a command is issued by the client when the game world is in state W_c, the command will reach the server and be executed when the world state is W'_s. Our goal is therefore to compute an estimate of W'_s from W_c, which we refer to as W'_c, and pass this object to the AI agent for it to make its decision on the next move. This is illustrated in Fig. 1.

To estimate W'_s from W_c, we observe that in addition to the evolution of the game world, there are also commands that would have arrived at the server and executed between W_s and W'_s. The client does not know what commands were issued by other players or agents, but it does know exactly what commands were issued in the past by the AI agent residing on it. Therefore, to derive a more accurate estimate, it will record the commands that were issued by the AI agent for the past s ticks and assume that a command issued by the client would take $\frac{s}{2}$ ticks to reach the server. W'_s is then estimated by simulating W_c for s ticks and applying the appropriate commands issued by the AI agent at their estimated arrival times between W_s and W'_s.

Our work has two key contributions: (i) to the best of our knowledge, ours is the first study of how network latencies can affect AI agents offloaded from a server; and (ii) our preliminary experiments showed that a simple implementation of dead reckoning is effective at delaying the onset of performance degradation for round-trip latencies up to 150 ms. These results are significant as they imply that the offloading of AI agents is practical given that observed latencies on the Internet are often within the range of 100 to 150 ms [6].

We believe that our solution is attractive not only because it is effective, but because it is relatively simple and therefore practical. It should be relatively easy to implement in existing network games as most of them already have dead reckoning implemented to improve the players' experience. Our implementation of dead reckoning involved less than 200 lines of Java code, demonstrating that an AI agent originally developed to be run on a server can be offloaded with minimal modifications.

One drawback of offloading AI to clients is that it might potentially expose the AI to abuse by hackers since the AI code is now accessible on the client machines. Douceur et al. suggested running a deterministic AI agent and have multiple clients executing the same AI code [1]. The results returned by each agent are then compared and the decision taken by quorum. We note also that there are commercially-deployed anti-cheating solutions [4,5] in current multiplayer games and we plan to study if some of them can also help mitigate the problems of cheating.

Our work is currently still work-in-progress. In addition to network latencies, practical deployments of offloaded AI will have to deal with other complications like packet loss and jitter. We plan to explore these issues and develop techniques to deal with these problems in a comprehensive way. We can also improve our implementation of dead reckoning by taking into account the likely commands issued by the opposing AI agents. More sophisticated dead reckoning algorithms [2,3] will likely yield better results as well.

Finally, we will evaluate the scalability of our offloading techniques given the limited bandwidth of typical client machines and determine if they would yield the same advantages on games of different genres or those with more complex physical models. We hope to use these results to develop new network protocols that can effectively address the challenges of offloading AI over the network. More information on our work is available at http://www.comp.nus.edu.sg/~baijia85/offloadingAI.html.

REFERENCES

[1] J. R. Douceur, J. R. Lorch, F. Uyeda, and R. C. Wood. Enhancing game-server AI with distributed client computation. In *Proceedings of NOSSDAV '07*, 2007.

[2] D. Hanawa and T. Yonekura. A proposal of dead reckoning protocol in distributed virtual environment based on the taylor expansion. In *Proceedings of CW '06*, 2006.

[3] A. Krumm-Heller and S. Taylor. Using determinism to improve the accuracy of dead-reckoning algorithms. In *Proceedings of SimTecT '00*, 2000.

[4] nProtect GameGuard. http://eng.nprotect.com/nprotect_gameguard.htm.

[5] PunkBuster. http://www.evenbalance.com/.

[6] Stanford Linear Accelerator Center. The PingER project. http://www-wanmon.slac.stanford.edu/cgi-wrap/pingtable.pl.