Is it Practical to Offload AI over the Network?

Jiaqiang Bai* and Ben Leong†
Department of Computer Science,
School of Computing,
National University of Singapore

ABSTRACT
In this paper, we study the degradation of performance for an offloaded AI agent with increasing network latencies and evaluate the effectiveness of dead reckoning in mitigating the observed degradation. Dead reckoning refers to a class of algorithms typically employed to predict the state of objects in existing games to mitigate the effects of game lags and improve player experience. For a realtime tank game, we found that increasing latencies will cause gradual degradation to the performance an AI agent and the agent is rendered completely ineffective when latencies reach about 300 ms. We show that a simple implementation of dead reckoning is effective at mitigating the effects of network latencies. Our solution is able to delay the onset of performance degradation for round-trip latencies up to 150 ms. Since the average latency within the continental North America is approximately 55 ms and inter-continental latencies are in the vicinity of 200 ms, our results demonstrate that it is feasible to offload AI to client machines. Most importantly, our method is practical because it allows offloaded AI agents to be developed in a network-oblivious manner similar to what is presently done for server-based AI.

1. INTRODUCTION
There have been many proposals for peer-to-peer (p2p) games (Knutsson, Lu, Xu, & Hopkins, 2004; Bharambe, Agrawal, & Seshan, 2004; Chan, Yong, Bai, Leong, & Tan, 2007) and we are fast approaching the practical deployment of such games (Yong, Razeen, Koh, Liew, & Leong, 2008). While the traditional server-client architecture is expected to continue to be the dominant player for “heavy-weight” games like the World of Warcraft (Blizzard Inc, 2008), multiplayer versions of the genre typically referred to as “casual games” are becoming more common. We believe that a p2p architecture is an attractive option because it allows game developers to save on infrastructural costs by utilizing the clients’ available computing capabilities and bandwidth.

Multiplayer networked games can be viewed abstractly as a synchronization operation across multiple nodes. We believe that eventual p2p deployments will favor an approach that elects pseudo-server nodes to manage the synchronization and recover game state from backup nodes whenever these server nodes fail (Chan et al., 2007). Existing games typically adopt a monolithic architecture, where the AI agents that control the non-player characters (NPCs) are run on the servers. In the context of our pseudo-server-based p2p architecture, the offloading of AI from the server nodes to clients running on peer nodes will make the pseudo-server processes more light-weight.

Because these light-weight pseudo-servers maintain less game state, it would be easier to recover when they fail. The overall architecture becomes more resilient and player experience...
is improved. Also, with the availability of additional computational resources from peers, it is conceivable that we can support more complex AI behaviours (Douceur, Lorch, Uyeda, & Wood, 2007).

Unfortunately, the offloading of an AI agent to peer nodes is not without complications. Because there is network latency between the server and the AI agent, game state cannot be instantaneously updated and commands issued are only executed after a delay. If the person writing the AI agent has to take into account the possibilities for varying network latencies, his job will be significantly more complex, perhaps even intractable. Ideally, we would like to be able to offload an AI agent that was written to run on the server to a client with no or minimal modification.

In this paper, we study the degradation of performance for an AI agent as network latencies increase and evaluate the effectiveness of dead reckoning in mitigating the observed degradation. Our goal is to develop a method that allows AI agents to be developed in a network-oblivious manner similar to what is presently done for server-based AI and also to understand the limits of performance that can be achieved for offloaded AI.

As mentioned, there are two key challenges in offloading AI to clients: (i) the state of the game world as perceived by the client is slightly behind 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. Our key insight is to employ dead reckoning to predict the expected state of the simulation when a command is expected to be executed and to present this state of the game world to the AI agent. The AI agent will then make its decisions based on this augmented state instead of the local state perceived by the client.

1.1. Paper Organization
The remainder of this paper is organized as follows: in Section 2, we provide a review of existing and related work. In Section 3, we describe our experimental methodology. Our evaluation results are presented in Section 4. Finally, we discuss our work and conclude in Section 5.

2. RELATED WORK
To the best of our knowledge, the offloading of AI to clients was first proposed by Douceur et al. (Douceur et al., 2007). The context of their problem was however quite different. Their goal was to offload AI in order to support increasingly complex AI in the context of massively multiplayer online games (MMOGs). Douceur et al. proposed the splitting of AI into two components and offloading the more computationally expensive component to clients. They also showed that their approach is able to tolerate latencies of up to 1 second. The goal of our work is to offload AI completely from the server so that the pseudo-server node can be simpler and therefore simplify the recovery process when the pseudo-server node fails. Given the context of our work, Douceur et al.’s solution is not feasible as the splitting of an AI into two parts can make the recovery process for a p2p setup even more complicated. Also, from our experience with game AI, we believe that it might not always be feasible to split AI into two components in a straightforward way in the general case.
3. PROBLEM AND ALGORITHM

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.

![Figure 1. Implementation of dead reckoning.](image)

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 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 $W'_s$. 
4. EVALUATION

We ran our experiments on two desktop machines, two Intel Core2 Duo desktop running Linux kernel 2.6.24.4. One machine has Netem (Netem, 2008) installed to control upstream and downstream network latency for connected machines. Netem provides network emulation functionality for testing protocols by emulating network variables like delay, loss, duplication and re-ordering.

In each experiment, two tanks controlled by identical AI agents were deployed against each other. One agent was run on the same machine as the server with no network latency while the other player was run remotely on another machine connected through Netem configured with a specified network latency. The setup is shown in Figure 2. The AI agent attempts to encircle the enemy tank and fire at it. Because both tanks are in constant motion, they have to constantly adjust their turrets in order to score hits. We adopted this strategy because network latencies will have little or no effect on the stationary objects since updates from the server would not be required.

![Figure 2. Experiment setup](image)

In Figure 3, we observe that when the remote AI agent was ran with dead reckoning, the onset of performance degradation was delayed. The performance was delayed by about 150 ms. Thereafter, the performance of the remote AI agent degrades relatively quickly.

![Figure 3. Plot of proportion of wins by remote AI agent against network latency.](image)
One important observation is that dead reckoning can in fact improve the performance of the remote agent slightly even when the network latency is zero. This is because even without latency, a command is executed only at the next tick and if the AI agent makes a decision based on the current tick, the commands will be issued slightly late as an artifact of the simulation. It was interesting to see that dead reckoning naturally corrected for this artifact.

We see that the hit rate of the local AI agent is approximately 65 to 75% and relatively constant, as expected. The degradation of performance for the remote agent in terms of its firing accuracy is somewhat more gradual than what we might infer from the proportion of wins. Overall, our modified dead reckoning is to be able to improve the accuracy of the remote agent by about 10 to 20%.

However, we note that at 275ms and beyond, our implementation starts to perform worse than the agent without dead reckoning. This is because we used a simple step forward mechanism for the world. In other words, tanks are simulated to move forward in a straight line if they are moving. We can definitely do better by observing the past states of the tanks and deduced a better estimate of their next move. For example, if we see that a tank is moving in a circular path in the past states we simulate its future movement to be circular.

It was noted that the accuracy of the local AI agent degrades at high network latencies. We found that this was because at such high latencies, the tank controlled by the remote AI agent tends to drift away leading to more misses by the local-AI-agent-controlled tank. This is a minor artifact of the experiment.

5. CONCLUSION
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 (Douceur et al., 2007). 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 (nProtect GameGuard, 2008; PunkBuster, 2008) 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 (Krumm-Heller & Taylor, 2000; Hanawa & Yonekura, 2006) 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.

6. REFERENCES


