Enhancing Game-Server AI with Distributed Client Computation

John R. Douceur
Microsoft Research
Redmond, WA
johndo@microsoft.com

Jacob R. Lorch
Microsoft Research
Redmond, WA
lorch@microsoft.com

Frank Uyeda
Univ. of California
San Diego, CA
fuyeda@ucsd.edu

Randall C. Wood
Microsoft Game Studios
Redmond, WA
randallw@microsoft.com

Abstract

In the context of online role-playing games, we evaluate offloading AI computation from game servers to game clients. In this way, the aggregate resources of thousands of participating client machines can enhance game realism in a way that would be prohibitively expensive on a central server. Because offloading can add significant latency to a computation normally executing within a game server's main loop, we introduce the mechanism of AI partitioning: splitting an AI into a high-frequency but computationally simple component on the server, and a low-frequency but computationally intensive component offloaded to a client. By designing the client-side component to be stateless and deterministic, this approach also facilitates rapid handoff, preemptive migration, and replication, which can address the problems of client failure and exploitation. To explore this approach, we develop an improved AI for tactical navigation, a challenging task to offload because it is highly sensitive to latency. Our improvement is based on calculating influence fields, partitioned into server-side and client-side components by means of a Taylor series approximation. Experiments on a Quake-based prototype demonstrate that this approach can substantially improve the AI's abilities, even with server-client-server latencies up to one second.

1 Introduction

In Massively Multiplayer Online role-playing Games (MMOGs), the artificial intelligence (AI) that controls monsters and non-player characters is exceedingly poor. It is common for gamers to complain of monsters that are so stupid as to make the game unchallenging [22]. Despite popular belief, the fundamental problem is not that game developers cannot write better AI algorithms. Rather, the problem is that the servers that host MMOGs have insufficient computing power to support the computational demands of thousands of even moderately sophisticated, concurrently running AIs [19]. Adding more back-end server resources could solve the problem, but at a cost that would be prohibitive given MMOG operations economics.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
NOSSDAV'07 Urbana, Illinois USA
Copyright 2007 ACM 978-1-59593-746-9/06/2007...$5.00.

MMOGs usually run on servers under control of a game operator, who bears the direct financial burden of computation and operations. Switching to a client-based infrastructure [9,13,17] would provide computational power that scales with the count of concurrently active players. However, this approach sacrifices the security and control that is generally considered necessary in persistent-world games. This is especially true in light of exploits that have extracted real money from MMOGs [4,15].

In this paper, we consider an approach to harnessing the aggregate computational power of client machines for improving game AI, for the purpose of making monsters behave in more interesting ways and thus improving the MMOG playing experience. Rather than suggesting a radical restructure of MMOG architecture [6], we propose supplementing server-based computation by offloading components of AI onto client machines.

This approach is not without challenges. It adds a substantial communication delay to code that normally executes within the game server's main loop. It relocates critical functionality to clients that may fail or become disconnected. And, it makes sensitive computations more readily exploitable by unscrupulous players who hack their client software.

We focus specifically on the problem of latency, which we address with the mechanism of *AI partitioning*: splitting an AI into a server-side component that retains critical tight-loop control and a client-side component that computes tuning parameters for the server-side component. We do not specifically address the issues of client failure or exploitation; however, these problems are addressable by migration and replication. AI partitioning can enable migration and replication by making the client-side component stateless and deterministic.

As a demonstration of feasibility, we apply AI partitioning to the latency-sensitive problem of tactical navigation. We develop an improved navigation AI based on summed influence fields, and we offload the bulk of the computational effort as a 2D Taylor series approximation. Experiments demonstrate that the technique is effective even with latencies in excess of one second.

The next section presents our vision for improving the gameplay of MMOGs with sophisticated AI. Section 3 surveys issues that client offloading must deal with, including the issue of latency. Section 4 introduces AI partitioning, a mechanism for addressing the latency problem. Section 5 applies the AI partitioning technique to the problem of enhanced tactical navigation, which we evaluate in Section 6. We briefly survey related work in Section 7 and conclude in Section 8.

2 A vision of sophisticated AI

We envision MMOGs in which monsters and non-player characters display behavior that is complex and interesting, though well within the reaches of present AI technology. We imagine monsters that travel in packs across a wide range, that engage in useful or distracting activities rather than just milling around, that become aware of players by sight, sound, or smell, and that stalk their victims and pounce when unexpected. Sophisticated monsters will work together, attacking the same target and coordinating their efforts. They will assess a group of players collectively, deciding whether to attack based on an assessment of comparative strength.

By contrast, current AIs exhibit astonishingly simple behavior. When unaware of nearby players, a typical monster either waits in a delineated region or roams along a predetermined path. When a player comes within a defined distance, the monster launches a direct attack. When severely wounded, some monsters will fight to the death, whereas others will try to retreat via a simple path.

The problem of retreating is illustrative. A severely wounded monster should not run directly into its attackers, who may be roughly surrounding it. If some of the attackers are wielding melee weapons, it should try to stay out of their weapons' range. It should try to get quickly away from attackers with range weapons, such as archers, who will likely be standing still. These influences collectively suggest a particular instantaneous direction the monster should head. However, the influences will change as players move and prepare attacks, so the optimal direction will rapidly vary as the monster retreats. The calculations involved are not highly complex, but they are computationally demanding, particularly for a server that is determining the appropriate next step for thousands of monsters concurrently.

3 Issues in client offloading

Sophisticated AI calculations can be offloaded to clients only if several issues are addressed, including the availability of client CPU capacity, communication latency between clients and the serve, the possible failure of client machines, and the risk of client exploitation.

For clients to contribute substantially to computation of game AI, there must be enough spare CPU capacity on the clients that the local users' gameplay is not disrupted by the additional computation load. The popular World of Warcraft MMOG requires a minimum CPU speed of 800 MHz and recommends a CPU speed of 1.5 GHz [3], yet 2/3 of desktop machines have CPUs with speeds in excess of 2 GHz [18,20]. This data suggests that there is ample spare CPU on game clients' machines.

Offloading computation to a client may induce a substantial communication delay, as work that was previously performed in the server's main loop is now distributed to clients, processed on those clients, and sent back to the server. Round-trip latency between access networks can reach 400 ms [23], and a 56K-dialup access network can add as much as 500 ms more [10]. Although some aspects of AI, such as high-level strategic planning, may tolerate latencies that approach one second, it is not *a priori* clear whether tactical-level AI can satisfactorily cope with the lag of such a network delay.

Client machines can also fail in various ways: They may crash or spontaneously reboot; network problems can cause intermittent disconnection; players may abruptly quit the game; or a competing client application might become active and leave little available CPU. The server can thus not afford to rely on any particular client to perform any given computation.

Furthermore, in the absence of a secured execution platform, AI code that runs on a client machine can be modified by the machine's owner. The owner might weaken the AI to make monsters stupid and easy to kill, or strengthen the AI to make monsters smarter and readily able to kill competing players. The server cannot safely assume that clients will calculate results honestly.

4 AI partitioning

To deal with the problem of latency, we introduce the mechanism of *AI partitioning*. This technique can be made amenable to redundant computation, to address the problems of client failure and exploitation.

The key idea is to split the AI that controls each subject into two components: a server-side AI and a client-side AI. The server-side AI is fairly simple, highly tunable, and high-frequency in that it runs at the same rate as the server's game loop. It performs any job that is intolerant of lag, such as targeting. The client-side AI is smart, complex, and potentially quite slow. Its function is to compute tuning parameters for the server-side AI, to support tasks such as long-term planning. The split between the two AIs isolates the server not only from the communication delay to the client but also from the computation delay of complex AI logic.

Periodically, the server sends a *glimpse* of the game state to the client, and the client responds with *advice*. A glimpse is a snapshot of limited scope, containing data of proximate relevance to the AI's subject. This glimpse is input to the client-side AI computation. The output of the computation is advice for the server-side AI, typically in the form of parameters and coefficients. Because glimpses and advice consume bandwidth, it is important to keep them fairly small.

It is advantageous to design the server-side AI to be tolerant of stale advice. Communication delays, varying client CPU availability, and the possibility of client failure preclude any guarantee of promptly returned advice. Thus, the server-side AI should not require advice for any particular execution frame; rather, advice should be useful over a range of execution time. Punctuality may be beneficial, but it should not be critical. In the extreme case in which no advice is received for an extended period of time, the server-side AI should have a *fallback* mode in which it can operate independently of client advice.

There are several advantages to designing the client-side AI to be *stateless*. By this, we mean that each glimpse-advice computation is independent of prior computations, with no client-side state carried forward. If a client fails or becomes disconnected, and the server hands off the computation to another client, the new client can immediately pick up where the previous one left off. In addition, the server can temporally limit the effect of each client on the server-side AI by *preemptively migrating* the client-side AI: assigning successive computations for the same subject to different clients.

There are also advantages to designing the client-side AI to be *deterministic*, meaning that identical glimpses produce identical advice. To tolerate failures and exploits, the server can redundantly issue the same glimpse to multiple clients, effectively replicating the client-side AI. To deal with simple failures, the server can accept the first advice it receives. Alternatively, to deal with attempted client exploits, the server can wait for multiple replies and use plurality voting to determine the correct advice; however, this may increase latency as the server waits for replies from multiple clients. If the client-side AI computation needs to include randomness, the seed for the random-number generator should be selected by the server and sent with the glimpse to clients, thereby keeping the client-side AI deterministic.

5 Example – tactical navigation

To evaluate the feasibility of offloading AI to clients, we consider the problem of tactical navigation. This is a particularly challenging task because it is highly sensitive to latency. The intent of the example is not to demonstrate a particularly impressive AI; rather, it is to illustrate how AI calculations can be effectively partitioned in a way that tolerates the latency of remote computation.

5.1 Classic navigation

The conventional approach to game-AI navigation [19,21] is first to select a goal and then to move toward that goal via a series of predetermined waypoints. If the goal is an opponent, then when the opponent comes within a defined range, navigation switches to a mode of random selection among preprogrammed attack movements such as charging, feinting, and strafing.

The main benefit of this approach is computational efficiency, since the complex logic for selecting a new goal is performed sporadically rather than reevaluated on every frame, and detailed path calculations are performed offline prior to game execution.

However, this approach has at least two significant weaknesses. First, it only allows for one goal at a time. In contrast, humans can simultaneously weigh several goals and devise a path that optimizes over all of them. Second, this approach does not readily adapt to quickly changing circumstances, such as the virtual locations of teammates and opponents. Consequently, this approach cannot execute interesting and intelligent movement patterns, such as the complex retreating behavior described in Section 2.

5.2 Improvement: aggregate influence fields

We address the two weaknesses noted above with a more flexible approach to tactical game navigation. Rather than navigating toward a single selected goal, the approach is to calculate a vector field that characterizes the collective influence of all entities in the vicinity, and then to move in the direction indicated by the field. This field optimizes over both the explicit primary goal and implicit secondary goals, and it can be readily recalculated as entities move.

From the subject's perspective, each other entity exudes an attractive or repulsive radial influence field with a magnitude of

$$\|\mathbf{v}\| = |W| d^{-m}$$

where d is distance from the subject, m is a decay factor, and W is a weight. Most entities are attractive, such as the primary goal, targeted opponents, weapons, ammunition, and health packs. Some entities may be repulsive, such as a powerful opponent who is currently attacking the subject.

The aggregate influence field is the sum of the fields from nearby entities. For a subject at point $\mathbf{p}$ in virtual space, the aggregate field $\mathbf{f}$ from a set of N entities can be calculated as

$$\mathbf{f}(\mathbf{p}) = \sum_{k=1}^{N} W(k) \|\mathbf{p}_k - \mathbf{p}\|^{-m(k)-1} (\mathbf{p}_k - \mathbf{p})$$

where the weight W and decay factor m are functions that vary per entity. In general, the weight and decay functions are affected by the state of the subject; for example, as the subject's health decreases, it becomes more repulsed by attacking opponents and more attracted by health packs.

We can reformulate the above expression for $\mathbf{f}$ as

$$\mathbf{f}(x, y) = \\ \sum_{k=1}^{N} W(k) \left((x_k - x)^2 + (y_k - y)^2 \right)^{-m(k)/2-1/2} \left((x_k - x)\mathbf{i} + (y_k - y)\mathbf{j} \right)$$

where $\mathbf{i}$ and $\mathbf{j}$ respectively represent unit vectors in the X and Y dimensions. By simplifying the latter formulation, the cost of calculating the aggregate field is five additions, six multiplications, and one exponentiation per entity in the subject's vicinity.

Fig. 1a illustrates a sample aggregate influence field. The subject is repelled by the attacking enemy, attracted to the other enemy, and more attracted to the health pack.

5.3 Offloading: Taylor-approximate fields

The cost calculating the aggregate influence field is proportional to the count of entities in the area. To offload the bulk of this effort to a client, we use AI partitioning, as described in Section 4. Specifically, in the server-side AI, we replace the calculation of the actual influence field $\mathbf{f}$ with the calculation of a second-order two-dimensional Taylor series approximation:

$$\mathbf{f}(\tilde{x} + \Delta x, \tilde{y} + \Delta y) \approx \\ \left(A_0 + \mathrm{A}_1 \Delta x + \mathrm{A}_2 \Delta y + \mathrm{A}_3 \Delta x^2 + \mathrm{A}_4 \Delta y^2 + A_5 \Delta x \Delta y \right) \mathbf{i} \\ + \left(B_0 + \mathrm{B}_1 \Delta x + \mathrm{B}_2 \Delta y + \mathrm{B}_3 \Delta x^2 + \mathrm{B}_4 \Delta y^2 + B_5 \Delta x \Delta y \right) \mathbf{j}$$

The cost of this computation is merely 12 additions and 13 multiplications, irrespective of the count of entities in the vicinity. The Taylor-series coefficients, A_0 to A_5 and B_0 to B_5, are computed by the client-side AI, based on a glimpse of the game state provided by the server when the subject is at point ($\tilde{x}$, $\tilde{y}$) in virtual space.

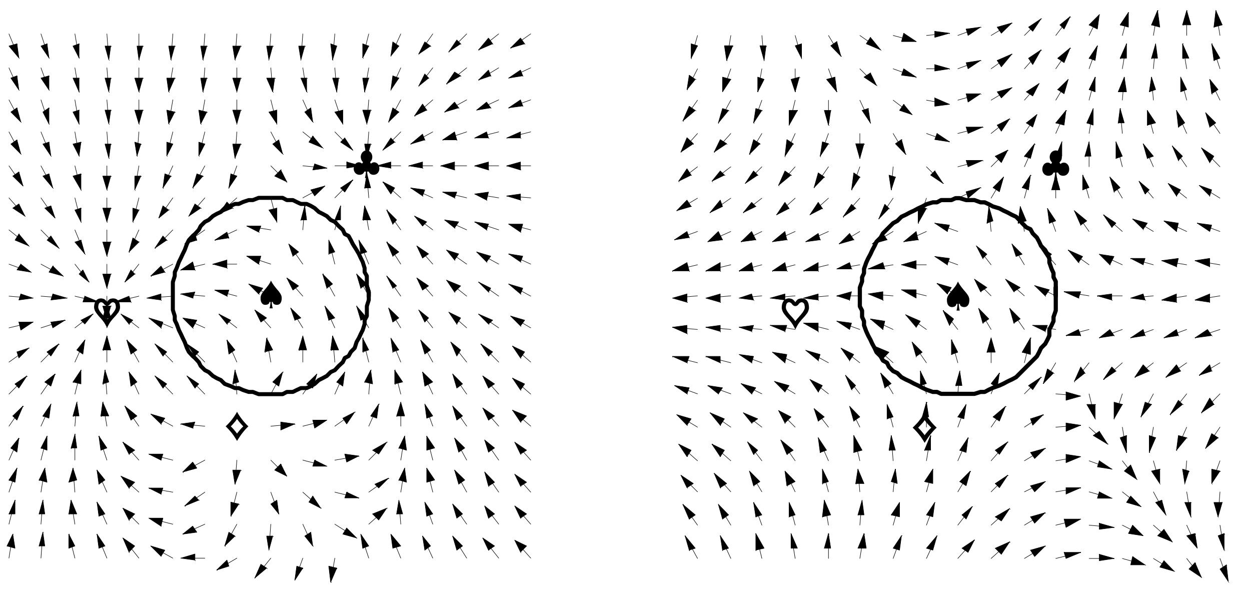

Fig. 1: Aggregate influence field – subject ♠, attacking enemy ♢, non-attacking enemy ♣, health pack ♡

Fig. 1b illustrates a 2D Taylor series approximation of the aggregate influence field in Fig. 1a. Within the region highlighted by the circle, the approximation closely follows the actual influence field.

The edges of Fig. 1 show that the approximation can be wildly wrong, as the Taylor-series accuracy diminishes with distance from the point $(\tilde{x}, \tilde{y})$. Therefore, as the subject moves away from this point over time, the advice returned by the client-side AI becomes less valuable. In addition, even if the subject stays in place, the positions of the other entities will change, rendering the advice stale. We quantify this effect in the next section.

6 Evaluation

To determine how well AI partitioning can work in practice, we built a prototype of our Taylor-series-based tactical navigation. Because we want our experiments to evaluate partitioned AI performance even at high latencies, we do not provide a server-side fallback mode. We also do not implement replication; however, the client-side AI is both stateless and deterministic, so replication would be fairly straightforward to add.

6.1 Prototype implementation

Due to our unfamiliarity with available MMOG code bases [7,8], we evaluate the above mechanism in an open-source first-person-shooter (FPS) game, specifically Quake III. Although FPS games are quite different from MMOGs in many respects, the basic game loop and combat AI logic are very similar [1,19].

We built a prototype implementation of partitioned AI inside Quake III's bot code. We employ Quake III's standard AI [21] for every aspect of bot control except the direction of motion, which we determine by a Taylor-approximate influence field. Notably, we did not modify the target-selection logic or shooting accuracy.

Because our focus is on the AI-offloading technique, rather than on bot tactics, we did not perform extensive experimentation for optimal weights and decay factors in the field equations. Instead, we thought of a few basic strategies for tactical bot behavior. We then tinkered with parameter settings and parametric functions until we found a set that seemed to work reasonably well. The main considerations are that higher weights result in a greater influence, and higher decay factors increase the localization of the influence. Ultimately, we settled on four strategies of increasing sophistication:

seekplayer: The goal, identified by existing Quake III bot logic, is strongly attractive even at a distance ($W = 60$, $m = 1$); other bots are attractive when they are nearby ($W = 20$, $m = 2$); and items, such as weapons, ammunition, and health packs, are neutral.

seekall: Like seekplayer, except that nearby items are also attractive ($W = 20$, $m = 2.5$) if considered *useful*. An item is designated useful if acquiring it will change the bot's state: a weapon of a type not currently possessed, ammunition if the bot has less than the allowable limit, and a health pack if the bot has less than perfect health.

selfaware: Like seekall, except the state of the bot can cause some useful items to be designated as *critical*. This applies to weapons and ammunition when the bot has no ammunition left, and to health packs when the bot is below 20% health. Critical items are extra attractive ($W = 40$, $m = 2$).

avoidattacker: Like selfaware, except that when the bot's health is below 50%, it is repulsed by the last enemy to have damaged it recently. The repulsion is determined by whether the attacker wields a range weapon, in which case it is moderately repulsive when nearby ($W = –10$, $m = 2$), or a melee weapon, which makes it more repulsive but only when very close ($W = –30$, $m = 3$).

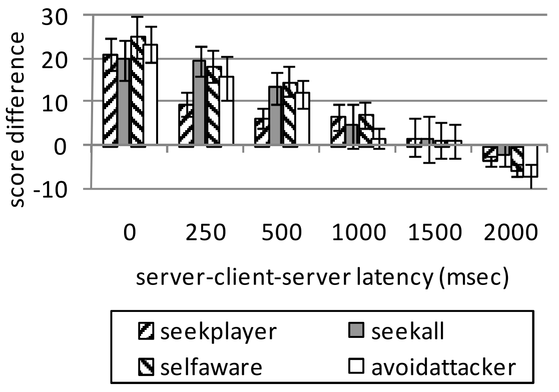

Fig. 2: Score difference for various strategies

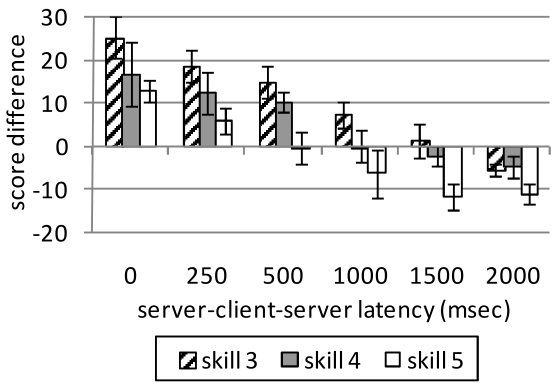

Fig. 3: Score difference for various standard bot levels

6.2 Experiments

Our virtual test environment is a flat space with no fixed obstacles, intended to resemble an open outdoor area as can be found in an MMOG. Since Quake environments must be bounded, we use the Quake map editor to build the largest allowable map, and we locate all items of interest in rough proximity to the center, to make the bots disinclined to go near the edges of the world. We also severely limit the availability of ammunition, to produce a relatively even balance of melee and ranged combat.

We evaluate our implementation by staging a series of six-bot, all-against-all "deathmatch" tournaments. Each tournament consists of nine ten-minute games, each involving three enhanced bots and three standard bots. By default, all bots are set to median skill (level 3 in the range from 1 to 5). To simulate wide-area network latency, we add a tunable delay to the client-server communication path. For each tournament, we measure the mean score difference between the enhanced bots and the standard bots. We vary the round-trip latency, the navigation strategy of the enhanced bots, and the skill level of the standard bots.

6.3 Results

Fig. 2 plots the difference between the mean scores of enhanced bots and the mean scores of standard bots, as a function of round-trip latency, for the four strategies enumerated above. It also plots 95% confidence intervals for these score differences. Even with latencies up to one second, our enhanced bots outperform the standard bots. We stress that this improvement has nothing to do with targeting or shooting accuracy, which we did not change. We replaced only the navigation logic.

The ability to tolerate a round-trip latency of one second suggests that the partitioning technique would be effective even in the high-latency environment described in Section 3. As the latency increases beyond this point, the standard bots begin to outperform our enhanced bots, as the advice from the client-side AI becomes increasingly stale. This suggests that a threshold of one second is a reasonable cutoff point for invoking a fallback navigation mode, as mentioned in Section 4. An obvious choice for fallback code is the standard bot navigation logic, which requires no advice from the client.

The confidence intervals for the four strategies overlap each other, implying that the four strategies are not significantly different in their effectiveness, or at least that any differences are too minor for our small set of experiments to demonstrate. Perhaps the ability to optimize over multiple goals, which is present in all four strategies, is the dominant factor responsible for the improved performance of field-based bots over standard bots.

We are somewhat surprised that sophisticated decision-making, as performed by the selfaware and avoidattacker strategies, has little effect on the game outcome, relative to our simpler enhancement strategies. However, this may be due to a quirk of FPS games that is not representative of MMOGs, namely that dying can be beneficial. A respawned bot regains full health and a partly loaded weapon, which is often an improvement over the bot's state prior to being killed. Thus, a strategy designed to avoid dying may not improve game score.

Fig. 3 plots the score difference of level-3 enhanced selfaware bots versus standard bots of skill levels 3, 4, and 5. As already shown in Fig. 2, enhanced bots outperform standard bots of the same level even with latencies up to one second. In addition, at lower latencies, enhanced bots outperform standard bots of higher skill level, despite the fact that these higher-level bots have better target-selection logic and greater shooting accuracy. This is even more noteworthy considering that our replacement of the navigation logic had the side effect of removing classic combat maneuvers such as feinting and strafing, which the standard bots continued to employ.

Overall, our experiments show that an enhanced AI, partitioned into server and client components, can improve on an AI's abilities, even with high round-trip latency.

7 Related work

Several prior researchers have investigated offloading workload from game servers to clients. Kabus et al. [11] forms clients into a multicast tree that disseminates server updates. FreeMMG [6] is a hybrid model, mainly peer-to-peer but with a server to help failed clients recover. Others have proposed completely replacing servers with peer-to-peer systems [9,13,17] running on clients.

In a different context, researchers have explored client-donated CPU cycles for distributed computing tasks, such as SETI@home and the other BOINC projects [2], and Folding@home and Genome@home [14].

The use of vector fields for directional guidance originated in the AI community, where it was applied to actual physical robots [5,12]. Mamei et al. [16] employed vector fields for strategically coordinating bots in Quake III; this is a sophisticated AI task to which our offloading technique could perhaps be applied directly.

8 Conclusions and future work

In this paper, we propose enhancing the AI of game servers by offloading computation to clients. To address the problem of latency, we partition each computation into a critical tight-loop server-side AI and an advice-giving client-side AI. As an exemplar, we design an enhanced AI for tactical navigation based on influence fields, and we partition it using Taylor series approximation. Prototype experiments show substantial improvement in AI abilities, even with round-trip latencies up to one second.

A further step with our prototype is to replicate the client-side AI and test its ability to deal with client failure. Another step is to add a local fallback mode to the server-side AI and investigate the transition between advised and unadvised AI behavior. A minor but practically important improvement is to execute the client-side AI on a low-priority thread, to ensure that it does not disturb the gameplay of the user on the client machine. Moreover, we would like to implement partitioned AI in an MMOG and conduct a user study to see whether it improves the game as we expect.

One aspect of client exploitation we did not consider is information leakage, in which clients inspect glimpses from the server to learn details of game state they should not be allowed to observe. We would like to investigate anonymization and obfuscation techniques to limit client visibility into offloaded computations.

9 Acknowledgements

The authors thank Mike Calligaro for insightful early discussions of ideas in this paper, Jeff Pang for his assistance with the Quake III code base, and the anonymous NOSSDAV reviewers for their very helpful suggestions in improving the paper.

References

[1] T. Alexander, *Massively Multiplayer Game Development*, Charles River, 2003.

[2] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," 5th IEEE/ACM *GRID*, 2004.

[3] Blizzard, "Minimum system requirements for World of Warcraft," *World of Warcraft Community Site*, http://www.blizzard.com/support/wow/?id=aww0823p

[4] G. Block, "World of Hackcraft: Fooling The Warden," *CTOFORADAY: The Journal of Gregory Block*, http://www.ctoforaday.com/articles/000059.html

[5] J. Borenstein, Y. Koren, "Real-time Obstacle Avoidance for Fast Mobile Robots," IEEE Trans. Systems, Man, and Cybernetics, 19 (5), 1989.

[6] F. R. Cecin, R. de Oliveira Jannone, C. F. R. Geyer, M. G. Martins, J. L. V. Barbosa. "FreeMMG: A Hybrid Peer-to-Peer and Client-Server Model for Massively Multiplayer Games," *NetGames*, 2004.

[7] Crossfire, "Crossfire – The Multiplayer Adventure Game," http://crossfire.real-time.com/

[8] Daimonin, "Daimonin MMORPG – Free Fantasy Online Multiplayer Game," http://www.daimonin.net/

[9] C. GauthierDickey, D. Zappala, V. Lo. "A Fully Distributed Architecture for Massively Multiplayer Online Games," *NetGames* 2004.

[10] T. Jehaes, D. De Vleeschauwer, T. Coppens, B. Van Doorselaer, E. Deckers, W. Naudts, K. Spruyt, R. Smets, "Access Network Delay in Networked Games," *NetGames* 2003.

[11] P. Kabus, W. Terpstra, M. Cilia, A. Buchmann. "Addressing Cheating in Distributed MMOGs," *NetGames* 2005.

[12] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," IEEE Robotics and Automation, 1985.

[13] B. Knutsson, H. Lu, W. Xu, B. Hopkins. "Peer-to-Peer Support for Massively Multiplayer Games," IEEE *INFOCOM* 2004.

[14] S. M. Larson, C. D. Snow, M. R. Shirts, V. S. Pande, "Folding@Home and Genome@Home: Using Distributed Computing to Tackle Previously Intractable Problems in Computational Biology," Computational Genomics, 2002.

[15] J. Lee, "Wage Slaves," *Computer Gaming World*, 07.05.2005, http://www.1up.com/do/feature?cId=3141815

[16] M. Mamei, F. Zambonelli, "Motion Coordination in the Quake 3 Arena Environment: A Field-Based Approach," *E4MAS*, 2004.

[17] M. Merabti, A. El Rhalibi. "Peer-to-Peer Architecture and Protocol for a Massively Multiplayer Online Game," IEEE *Globecom* 2004.

[18] PC Pitstop, "PC Pitstop General Statistics: Processor MHz Ranges," *PC Pitstop Research*, http://www.pcpitstop.com/research/cpurange.asp

[19] B. Schwab, *AI Game Engine Programming*, Charles River, 2004.

[20] Valve, "Valve Survey Summary," *Steam*, http://steampowered.com/status/survey.html

[21] J. M. P. van Waveren, "The Quake III Arena Bot," MS Thesis, TU Delft, 2001.

[22] R. C. Wood, "The Seven Deadly Sins of MMO Developers, Part II: Brain-dead AI," Windows Live Spaces - *Cranius*, http://cranius. spaces.live.com/blog/cns!6174BA0350BAA452!418.entry

[23] B. Zhang, T. S. E. Ng, A. Nandi, R. Riedi, P. Druschel, G. Wang, "Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space," *IMC* 2006.