Vector-Field Consistency for Multiplayer Games
Estado da Arte

Dinis Lage
{dinis.lage@ist.utl.pt}

Instituto Superior Técnico de Lisboa/INESC-ID
Av. Rovisco Pais, 1049-001 Lisboa

Resumo: Os jogos para multi-jogador são um dos maiores entretenimentos actuais à escala mundial. Com a disseminação da Internet torna-se essencial utilizar de forma eficaz os recursos de comunicação existentes de forma a permitir a interacção entre os jogadores da melhor forma possível. Existem já inúmeras técnicas e modelos que visam melhorar a jogabilidade deste tipo de jogos e também aumentar o número de participantes que podem participar numa mesma sessão.

Palavras-chave: jogos multi-jogador, *locality awareness*, escalabilidade, jogabilidade

1 Introdução

Os jogos para multi-jogador representam actualmente uma das formas mais populares de comunicação em grupo na Internet. A natureza de tempo real de muitos destes jogos tem especial interesse pois estes requerem tempos de resposta e de comunicação curtos. Controlando a quantidade de informação que é transmitida no decorrer do jogo, adaptando os limites de consistência dependendo da localidade de interesse de cada jogador, é possível aumentar a quantidade de jogadores e melhorar a qualidade com que interagem num mundo virtual.

O trabalho proposto consiste em adaptar e implementar do modelo VFC (*Vector-Field Consistency* [1]) num ambiente distribuído de computadores pessoais no âmbito de um jogo *shoot'em up* e em analisar o impacto do modelo na jogabilidade.

É também de interesse avaliar a possibilidade da implementação do algoritmo VFC de forma distribuída, sem necessidade de existência de um servidor que controle o estado do jogo e a comunicação necessária no desenrolar do jogo.

Tendo como objectivo a facilidade de extensão do código do jogo será também avaliada a complexidade requerida de forma a ser possível utilizar o VFC na comunicação no jogo.

Este artigo está organizado da seguinte forma: na secção 2 começa-se por uma introdução histórica de mecanismos de consistência; na secção 3 são discutidas algumas abordagens que utilizam a localidade de interesse em jogos multi-jogador; a secção 4 introduz os principais tipos de jogos para multi-jogador e os factores que os caracterizam; na secção 5 é feita uma análise das arquitecturas mais comuns em diversos géneros de jogos multi-jogador; na secção 7 é introduzido o VFC e são discutidas algumas vantagens deste modelo; na secção 7 é apresentado o planeamento do trabalho a desenvolver ao longo deste trabalho; são tiradas algumas conclusões na secção 8.

2 Mecanismos de consistência

As técnicas de replicação tradicionais tentam manter a consistência de cópia única dos dados, dando aos utilizadores a ilusão de que apenas existe uma cópia altamente disponível. Existem diversas formas de atingir esse objectivo mas, geralmente, o mecanismo passa por bloquear o acesso aos dados sempre que não seja possível verificar que estão actualizados. Por esta razão estas técnicas são chamadas de "pessimistas". O desempenho verificado em redes locais é bom devido à baixa latência e alta disponibilidade. O mesmo não pode ser esperado de uma rede de larga escala, como o caso da Internet, por três motivos principais.

Primeiro, mesmo com a evolução da tecnologia, a Internet continua lenta e pouco fiável. Além disso, o uso de dispositivos móveis, com ligações intermitentes, é cada vez mais popular.

Em segundo lugar, os algoritmos pessimistas são pouco escaláveis. É difícil construir um sistema replicado pessimista de larga escala, com actualizações frequentes aos dados, pois o aumento do número de nós deteriora a taxa de transferência e disponibilidade do sistema.

Por último, existem actividades nas quais o utilizador requer que a partilha dos dados seja feita de forma assíncrona. Algumas actividades requerem que as pessoas estejam em ambientes de relativo isolamento. É então melhor permitir o acesso concorrente e tratar eventuais conflitos posteriormente do que bloquear o seu acesso.

Para diversos problemas nos quais se pode tolerar alguma inconsistência nos dados é então melhor optar por um mecanismo de consistência optimista. A replicação optimista consiste num conjunto de técnicas que permitem partilhar dados de forma eficiente em redes de larga escala e ambientes móveis. Ao contrário das técnicas de consistência pessimista, na replicação optimista as réplicas são coordenadas de forma assíncrona, permitindo que os dados sejam acedidos tanto para leitura como escrita, baseando-se na assunção de que os problemas, se ocorrerem, são raros. As actualizações são propagadas à posteriori e é então que são resolvidos os conflitos que possam existir.

Os algoritmos de replicação optimista têm diversas vantagens sobre os pessimistas: melhoram a disponibilidade dos dados; são flexíveis quanto à topologia da rede, mesmo com canais de comunicação variáveis e desconhecidos; devem escalar bastante melhor pois necessitam de pouca sincronização entre as réplicas; tanto as réplicas como o utilizador tornam-se mais autónomos, como o caso de controlo de versões como o CVS [2], SVN [3]; por último, a resposta dos algoritmos optimistas é rápida pois podem tentar aplicar as alterações assim que são submetidas.

No entanto, existe um grande desafio nas técnicas optimistas que é a possibilidade de existir divergência entre réplicas e operações concorrentes. Portanto, só podem ser aplicadas estas técnicas quando o sistema pode tolerar conflitos ocasionais.

Para uma análise mais profunda sobre replicação optimista ver [4].

Em aplicações que podem tolerar consistência relaxada foram propostos diferentes modelos de consistência optimista [5-8]. Infelizmente, tipicamente, os modelos optimistas não garantem limites na inconsistência dos dados que são fornecidos ao utilizador. Para tentar solucionar e delimitar a inconsistência dos objectos surgiram então modelos que tentavam optimizar o balanceamento entre disponibilidade e consistência.

As garantias de tempo real [4] permitem que réplicas de objectos obsoletos possam continuar a ser usados durante um determinado intervalo de tempo sem que seja verificada a sua actualidade.

A delimitação numérica baseia-se na noção de definição de limites aceitáveis para a quota de actualizações possíveis para cada réplica. Quando esgotada a quota de uma réplica só poderão voltar a ser efectuadas actualizações a essa réplica quando esta for tornada consistente com as restantes. Este conceito foi inicialmente introduzido com o TACT [9].

O TACT (*Tunable Availability and Consistency Tradeoffs*) é um modelo multi-dimensional que propõe a utilização da delimitação numérica em conjunto com a delimitação ordeira. Dessa forma tenta-se explorar o espaço existente entre consistência forte e estritamente optimista. Assim, ao existir um limite de consistência por cada réplica torna-se possível ao sistema calcular, com uma probabilidade esperada, por exemplo, se irá advir algum conflito de uma escrita ou se uma leitura pode observar escritas que posteriormente serão revertidas.

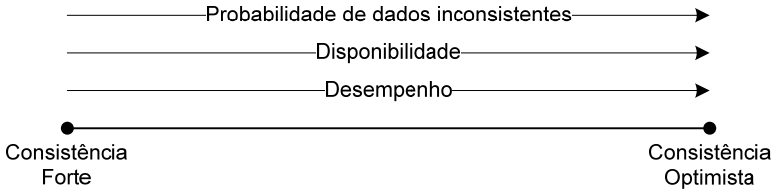

Figura 1 – Relação entre consistência, disponibilidade e desempenho

Na Figura 1 está representada a relação entre disponibilidade, desempenho e a probabilidade da existência de dados inconsistentes ao relaxar os requisitos de consistência de consistência forte para a optimista. Ao percorrer de consistência forte para optimista, o desempenho e a disponibilidade do sistema aumentam. No entanto, também aumenta a probabilidade de diferentes acessos poderem retornar dados inconsistentes. No TACT torna-se possível que cada aplicação ajuste o seu nível de consistência conforme seja necessário.

3 *Locality Awareness* em jogos multi-jogador

As noções presentes em *locality awareness* estão intrinsecamente relacionadas com o conceito de *interest management*, usado para filtrar grandes quantidades de dados em simulações distribuídas de larga escala [10]. Ao comunicar apenas dados com interesse para os outros nós consegue-se optimizar a utilização do canal de comunicação.

Os autores de [11] propõem a utilização de *locality awareness* para fazer o balanceamento da carga em MMG (*Massive Multiplayer Games*). Baseando-se em informação localizada, o algoritmo proposto faz o balanceamento da carga e reduz a comunicação entre servidores, tentando também evitar a mudança de atribuição de responsabilidade das diferentes zonas. Dado que com esta abordagem é possível a existência de *hotspots* (concentração grande parte dos jogadores numa mesma zona), é implementada uma heurística para calcular quando deve ser reduzida a carga num servidor (dividindo zonas muito populadas) ou aproveitados recursos de servidores com pouca carga (agrupando várias zonas).

Em [12] é proposta uma aproximação direccionada à visibilidade dos jogadores. Nessa arquitectura é aplicada uma divisão do mundo virtual por camadas. Essa partição é feita apenas quando o jogo é carregado inicialmente para que não haja necessidade de redesenhar as zonas no decorrer do jogo. A ideia por trás da partição por camadas é que ao aumentar a granularidade das zonas consegue-se diminuir o tamanho de cada zona. Ao diminuir o tamanho, diminui-se também a probabilidade de existência de um grande número de utilizadores nessa área. Na Figura 2 é apresentado um esquema representativo do tipo de divisão feita. A zona 1 representa uma zona com sobrecarga de utilizadores, cada um dos servidores contíguos assume responsabilidade pela subárea que lhe corresponde. Desta forma é preservada a localidade pois a carga é partilhada apenas por zonas subjacentes. O algoritmo também prevê a existência de múltiplas zonas de sobrecarga contíguas. A divisão hexagonal permite uma regularidade no particionamento aquando da existência de múltiplas zonas de sobrecarga. Na implementação dos autores conseguiu-se uma redução substancial no número de mensagens que era necessário trocar ao longo da interacção.

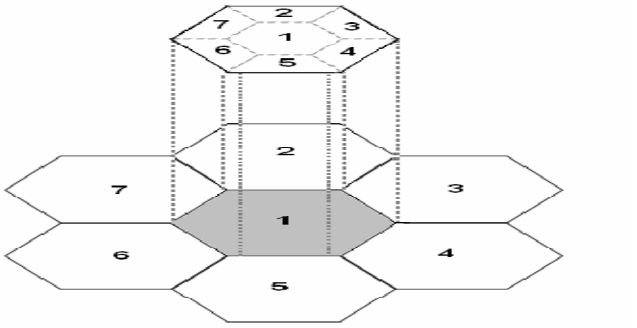

Figura 2 – Partição por duas camadas (de [12])

Em [13] é discutida a utilização de uma infra-estrutura ao nível da rede, em que algumas das funções do servidor são executadas em plataformas computacionais (*booster boxes*) que são co-alocadas com os *routers* e que têm noção do estado da rede. Desta forma consegue-se reduzir a carga nos servidores e é argumentado que será então possível realizar tarefas que actualmente não são fazíveis.

Outra solução de interesse é a utilização de uma implementação de suporte para MMGs em *peer-to-peer*, proposta apresentada em [14], onde é explorado o facto de que nesse tipo de jogos os jogadores apresentam localidade de interesse. Assim sendo torna-se possível formar grupos auto-organizados com base na sua localização no espaço virtual.

4 Principais tipos de jogos para multi-jogador

Para se conseguir melhorar o desempenho de jogos interactivos é necessário entender o tipo de tráfego gerado em cada tipo de jogo [15]. Os requisitos de comunicação em rede da maioria dos jogos podem ser inseridos numa das seguintes categorias ou, em alguns casos, possuir um conjunto de elementos de cada um dos grupos:

- *First Person Shooters* (FPS) – este tipo de jogo envolve grande percentagem de combate que requer um tempo de resposta baixo. É necessário grande número de mensagens, tal como eventos de actualização da posição ou acções dos jogadores. Em [16] foi feito um estudo que mostra que a latência começa a ser detectada aos

100ms e que torna a jogabilidade impossível aproximadamente aos 200ms. Este tipo de jogo é o mais exigente em termos de requisitos de latência.

- *Role Playing Games* (RPG) – visualmente, este tipo de jogo, é semelhante aos FPS mas o ritmo de interacções dos jogadores é geralmente menos intenso. Os RPGs online destinam-se geralmente a suportar um largo número de utilizadores, como tal, requerem uma boa escalabilidade. Este é o tipo de jogo mais exigente em termos de quantidade de fluxos de jogo concorrentes.
- *Real-Time Strategy* (RTS) – neste tipo de jogo o efeito da latência, mesmo que facilmente detectável pelo jogador, não interfere na jogabilidade nem no desempenho do jogador devido à sua natureza que claramente favorece a estratégia sobre aspectos de tempo real tal como estudado em [17] e [18].

Na Tabela 1 é apresentado um conjunto de características dos diferentes tipos de jogos multi-jogador. Actualmente, os MMORPG (*Massively Multiplayer Online Role-Playing Games*) conseguem suportar uma grande quantidade de jogadores (até milhares) utilizando paralelização por zonas, dividindo o mundo em diversas zonas independentes para processamento concorrente em vários servidores. Nos jogos FPS e RTS, cujos cenários são habitualmente mais reduzidos, não foi ainda conseguida uma escalabilidade para sessões com uma quantidade massiva de participantes pois a aproximação por zonas não é adequada a estes. Tanto nos FPS como nos RTS os jogadores tendem a concentrar as suas entidades em determinadas zonas onde existe mais acção.

Tipo de Jogo	**Nº servidores**	**Nº máximo jogadores**	**Latência máxima (ms)**	**Tamanho dos cenários**	**Nº de entidades manipuladas**
FPS	1	12 a 32	100	Pequeno	Uma
RPG (MMORPG)	1 (Centenas)	2 a 12 (Milhares)	500	Grande	Uma
RTS	1	6 a 12	1000	Médio	Dezenas a centenas

Tabela 1 – Principais características dos principais tipos de jogos para multi-jogador

5 Arquitectura de jogos multi-jogador

Actualmente a maioria dos jogos comerciais são implementados com uma arquitectura cliente-servidor, quer seja com apenas um servidor ou usando um cluster de servidores. Cada cliente possui então uma ligação independente ao servidor e toda a comunicação é feita através dele. Desta forma conseguem-se reduzir as possibilidades de *cheating*, assegurar o anonimato dos jogadores e simplifica-se a administração. Nesta arquitectura a forma como pode ser abordado o tráfego difere, entre o de cliente para servidor e o de servidor para cliente, pelos seguintes factores:

- Servidor para cliente: os servidores identificam grupos de clientes para os quais enviam a mesma informação. Podem então utilizar *multicast* de forma a melhorar a eficiência na utilização da rede ou, mesmo que tal não seja possível, permite definir prioridades ou agrupar conjuntos de mensagens.
- Cliente para servidor: os clientes podem gerar eventos tanto pela interacção dos utilizadores como periodicamente. Geralmente os clientes apenas comunicam com um servidor e não directamente com outros clientes.

Em quase todos os jogos comerciais do grupo de FPS a arquitectura utilizada é a de cliente-servidor simples, em que existe apenas um nó servidor. Cada servidor tem então um limite de participantes bastante reduzido. Por essa razão, jogos como *Quake*, *Half-Life*, *Unreal Tournament*, têm milhares de servidores disponíveis mundialmente. A latência afecta intensamente este tipo de jogos e por essa razão são regularmente utilizadas técnicas de compensação de latência que envolvem essencialmente a predição do lado do utilizador. Utilizando esta técnica os comandos do utilizador podem ser imediatamente interpretados (por exemplo: movimento, disparo da arma), partindo do princípio que esses comandos serão aceites pelo servidor e, caso haja alguma inconsistência, são posteriormente corrigidos com a resposta por parte do servidor [19].

A concentração de diversas entidades num espaço virtual pequeno torna necessária a subdivisão de responsabilidade da acção que aí se desenrola. Tendo em atenção esse facto, em [20] foi estudada a utilização de clusters de servidores de forma a tentar aumentar o número de jogadores suportados em FPS. Embora torne possível a participação de mais jogadores em cada sessão, esta escalabilidade não se aproxima eficazmente capacidade de participação massiva conseguida em jogos RPG.

A utilização de uma rede de servidores é vulgarmente utilizada em jogos MMORPG. Os cenários vastos, a latência máxima que possibilita boa jogabilidade relativamente relaxada (cinco vezes superior aos FPS), apenas uma entidade controlada (ao invés de dezenas ou centenas dos RTS), fazem com que neste tipo de jogos seja possível implementar técnicas de replicação por clusters de servidores que partilham a responsabilidade de diferentes zonas do mundo virtual. Muitas vezes, de forma a explorar uma determinada secção do mundo virtual, os jogadores têm que se ligar explicitamente a um servidor específico que está responsável por essa mesma secção. Exemplos de grande sucesso entre os MMORPG são o *World of Warcraft* [21] e o *Everquest* [22].

Existem também jogos para multi-jogador que utilizam modelos descentralizados, como é o caso do *MiMaze* [23] e do *Age of Empires* [17], cujo design tem grandes limitações em termos de escalabilidade pois o estado do jogo é transmitido para todos os jogadores. Alternativas a este comportamento incluem o *AMaze* [24] e o *Mercury* [25], duas implementações de jogos FPS, que se baseiam em comunicação em grupos consoante a localização actual dos jogadores. Embora a abordagem de *peer-to-peer* seja mais flexível e reduza o custo de estabelecimento dos jogos tem como desvantagem o aumento dos riscos de segurança dado que o jogo está distribuído por diversos nós. O *NPSNET 3D* [26] é um simulador de veículos, categoria que pode ser inserida nos FPS, que utiliza também comunicação em grupos de interesse no espaço virtual.

O *MiMaze* é um jogo em que cada jogador é representado por um avatar 3D semelhante ao “Pacman” que se movem por um labirinto tentando “matar” o maior número possível de adversários. Este jogo tem uma arquitectura totalmente distribuída e utiliza *multicast* para comunicação entre os jogadores. Todas as acções têm que chegar, ser processadas e mostradas a todos os outros jogadores com o menor atraso possível.

No *Age of Empires* é utilizada uma abordagem em que cada participante no jogo executa uma simulação idêntica do jogo de forma a ser possível apenas transmitir os comandos executados e assim reduzir a quantidade de informação que seria necessário transmitir devido à quantidade de entidades que cada jogador manipula. A comunicação é feita de todos para todos. É também utilizado um sistema de cálculo da velocidade de jogo possível que está dependente tanto da ligação entre os participantes como também

da capacidade de processamento de cada um dos terminais de jogo. Tratando-se de um modelo *peer-to-peer* em que todos os nós têm que executar exactamente a mesma simulação esta arquitectura é robusta contra *cheating* (por exemplo: aumentar recursos disponíveis, número de unidades) por parte de um jogador pois este seria considerado como dessincronizado.

Uma arquitectura distribuída que utiliza publicação/subscrição consoante a área de interesse (por exemplo: visibilidade, jogo em equipa) é utilizada no *Mercury*. A responsabilidade das zonas de interesse é distribuída, de forma uniforme, pelos nós participantes. É utilizado um protocolo de *routing* semelhante ao *Chord* [27]. Tal como no *Chord*, os membros estão organizados num anel lógico. O *Mercury* foi desenhado de forma a permitir pesquisa baseada em conteúdo em vez de apenas identificar o nó responsável por um determinado identificador de recurso. No desenrolar do jogo cada nó subscreve o conteúdo que têm interesse na acção do jogador.

O *NPSNET 3D*, sendo um simulador de veículos, tem diversas semelhanças com os FPS, tendo portanto requisitos apertados em termos de latência máxima que permite boa jogabilidade. Assenta sobre uma arquitectura *peer-to-peer* na qual é feita uma divisão do espaço virtual em conjuntos de hexágonos que representam diferentes grupos de interacção. O conjunto de grupos (pelo menos um) a que cada jogador pertence vai alterando à medida que se movimenta. Existem também grupos diferentes consoante o tipo de requisitos de consistência.

Jogo	Tipo	Arquitectura	Comunicação
Quake	FPS	Cliente/servidor	TCP/UDP
World of Warcraft	RPG	Rede de servidores	TCP/UDP
MiMaze	FPS	Peer-to-Peer	Multicast
Age of Empires	RTS	Peer-to-Peer	Todos para todos
Mercury	FPS	Peer-to-Peer	Publish-subscribe
NPSNET 3D	Simulador (FPS)	Peer-to-Peer	Grupos Multicast

Tabela 2 – Exemplos de jogos e suas características

6 Discussão

O VFC [1] é um modelo de consistência optimista que permite delimitar a divergência possível para as réplicas de um objecto. No VFC a consistência das réplicas é ajustada selectiva e dinamicamente com base no estado do jogo que se encontra a decorrer. Ao mesmo tempo gere como é que o grau de consistência altera ao longo da execução do jogo e como é que os requisitos de consistência são especificados.

O grau de consistência necessário em cada momento é calculado através da distância a pontos de observação, chamados de pivots, que podem ser, por exemplo, as posições dos jogadores. Quanto menor a distância a um pivot mais forte é a consistência necessária.

Os requisitos de consistência são controlados através de um vector tridimensional que define os graus de consistência. Cada dimensão representa um limite de divergência de tempo (atraso), sequência (número de alterações) e valor (magnitude das alterações).

Existem diversas vantagens inerentes ao VFC. Trata-se de um modelo flexível e facilmente percebido por programadores de jogos visto que o modelo de consistência baseado em pivots é intuitivo e a configuração de parametrização é minuciosa,

permitindo ao programador especificar os requisitos de consistência para um conjunto de cenários de jogo vasto. Do ponto de vista do jogador, o VFC permite que o jogo desenrole com harmonia no sentido em que, no que respeita o jogador, todas as regras são respeitadas, e os utilizadores possuem toda a informação relevante para a tomada de decisões de jogo. Ao definir diferentes prioridades consoante a importância das actualizações que acontecem no decorrer do jogo, o VFC utiliza de forma competente os recursos de largura de banda e disfarça a latência.

Em termos de aplicabilidade em princípio o VFC será de maior utilidade em jogos que se integrem tanto no grupo de FPS como de RPG. Neste tipo de jogos é fácil identificar quais os pivots que é necessário modelar e os seus raios de interesse. Nos RTS, dado o elevado número de entidades controladas pelo utilizador, torna-se mais difícil identificar os pivots e raios de interesse. Os benefícios que advêm da integração do modelo VFC serão portanto mais reduzidos.

7 Trabalho a desenvolver

Tendo como base a proposta apresentada em [1] e o trabalho respectivo já realizado, é agora proposto o desenvolvimento de uma implementação do modelo VFC num jogo a ser desenvolvido tendo como base um jogo template disponibilizado como ferramenta de suporte inicial para desenvolvimento de jogos no *XNA Game Studio 2.0*. O jogo utilizado é o *Net Rumble*, um *shooter* 2D em que os jogadores competem numa arena em estilo *death match.*

Inicialmente, a arquitectura a utilizar será de cliente-servidor. Assim será possível ter apenas um local no qual é feita toda a monitorização e controle dos diferentes limites de consistência. Esta é a arquitectura mais utilizada no domínio comercial dos jogos, tanto pelas suas vantagens de controlo de segurança como pelo anonimato dos clientes. Por esse motivo o estudo da falibilidade e verificação das melhorias possíveis com o método de *Vector-Field Consistency* nesta arquitectura ganha especial interesse para aplicações práticas, tanto em futuros sistemas como na integração dos sistemas actuais.

Posteriormente também terá grande interesse a implementação do modelo VFC numa arquitectura baseada em *peer-to-peer* visto que se conseguiria tirar partido dos recursos de comunicação em rede de forma mais eficiente.

8 Conclusão

A comunicação nos jogos para multi-jogador é alvo de grande estudo e teste à escala mundial. Foram apresentadas diversas abordagens que tentam optimizar a utilização da rede e resolver problemas de escalabilidade nos diversos tipos de jogos. Cada implementação requer uma relação de compromisso em termos de consistência entre os diversos participantes no jogo. Dados os diferentes requisitos de cada tipo de jogo é necessário utilizar o conjunto de técnicas que melhor se adapta de forma a optimizar a jogabilidade sentida pelos utilizadores.

9 Referências

[1] N. Santos, L. Veiga, and P. Ferreira, “Vector-Field Consistency for Ad-Hoc Gaming.”

[2] P. Cederqvist, *Version Management with CVS*, Network Theory Ltd., 2002.

[3] M. Pilato, *Version Control With Subversion*, O'Reilly \& Associates, Inc, 2004.

[4] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Computing Surveys (CSUR)*, vol. 37, 2005, pp. 42-81.

[5] D.B. Terry et al., "Managing update conflicts in Bayou, a weakly connected replicated storage system," *Proceedings of the fifteenth ACM symposium on Operating systems principles*, 1995, pp. 172-182.

[6] R.G. Guy et al., "Implementation of the Ficus replicated file system," *USENIX Conference Proceedings*, vol. 74, 1990, pp. 63-71.

[7] J. KISTLER and M. SATYANARAYANAN, "Disconnected Operation in the Coda File System," *ACM Transactions on Computer Systems*, vol. 10, 1992, pp. 3-25.

[8] R. Guy et al., "Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication," *Advances in Database Technologies: ER'98 Workshops on Data Warehousing and Data Mining, Mobile Data Access, and Collaborative Work Support and Spatio-Temporal Data Management, Singapore, November 19-20, 1998: Proceedings*, 1999.

[9] H. Yu and A. Vahdat, "Design and evaluation of a conit-based continuous consistency model for replicated services," 2002, pp. 239-282.

[10] K.L. Morse, "Interest Management in Large-Scale Distributed Simulations," 1996.

[11] J. Chen et al., *Locality aware dynamic load management for massively multiplayer games*, ACM Press New York, NY, USA, 2005.

[12] I. Kazem, D.T. Ahmed, and S. Shirmohammadi, "A Visibility-Driven Approach to Managing Interest in Distributed Simulations with Dynamic Load Balancing."

[13] D. Bauer, S. Rooney, and P. Scotton, "Network infrastructure for massively distributed games," *Proceedings of the 1st workshop on Network and system support for games*, 2002, pp. 36-43.

[14] B. Knutsson et al., "Peer-to-peer support for massively multiplayer games," *IEEE Infocom*, 2004.

[15] C. Majewski, C. Griwodz, and P. Halvorsen, "Translating latency requirements into resource requirements for game traffic."

[16] L. Pantel and L.C. Wolf, “On the impact of delay on real-time multiplayer games,” *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, Miami, Florida, USA: ACM, 2002, pp. 23-29.

[17] P. Bettner and M. Terrano, “1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond,” *Presented at GDC2001*, vol. 2, 2001, p. 30p.

[18] N. Sheldon et al., “The effect of latency on user performance in Warcraft III,” *Proceedings of the 2nd workshop on Network and system support for games*, Redwood City, California: ACM, 2003, pp. 3-14.

[19] Y.W. Bernier, “Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization.”

[20] J. Müller et al., “Scaling multiplayer online games using proxy-server replication: a case study of Quake 2,” *Proceedings of the 16th international symposium on High performance distributed computing*, Monterey, California, USA: ACM, 2007, pp. 219-220; http://portal.acm.org/citation.cfm?id=1272399.

[21]“World of Warcraft Community Site”; http://www.worldofwarcraft.com/index.xml.

[22]“EverQuest – Massively Multiplayer Online Fantasy Role-Playing Game”; http://everquest.station.sony.com/.

[23] C. Diot and L. Gautier, “A distributed architecture for multiplayer interactive applications on the Internet,” *Network, IEEE*, vol. 13, 1999, pp. 6-15.

[24] E.J. Berglund and D.R. Cheriton, “Amaze: A Multiplayer Computer Game,” *IEEE Softw.*, vol. 2, 1985, pp. 30-39.

[25] A.R. Bharambe, S. Rao, and S. Seshan, “Mercury: a scalable publish-subscribe system for internet games,” *Proceedings of the 1st workshop on Network and system support for games*, Bruanschweig, Germany: ACM, 2002, pp. 3-9.

[26] M.R. Macedonia et al., “Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments,” *IEEE Computer Graphics and Applications*, vol. 15, 1995, p. 3845.

[27] I. Stoica et al., “Chord: A scalable peer-to-peer lookup service for internet applications,” *SIGCOMM Comput. Commun. Rev.*, vol. 31, 2001, pp. 149-160; http://portal.acm.org/citation.cfm?id=964723.383071.