An Evaluation of Push-Pull Algorithms in Support of Cell-Based Interest Management

Rob Minson
School of Computer Science
University of Birmingham
Edgebaston, Birmingham
B152TT, UK
R.Minson@cs.bham.ac.uk

Georgios Theodoropoulos
School of Computer Science
University of Birmingham
Edgebaston, Birmingham
B152TT, UK
G.K.Theodoropoulos@cs.bham.ac.uk

Abstract

Several approaches for scalable interest management (IM) within real-time distributed virtual environments (DVEs) have been proposed based upon some division of the data-space in to disjoint volumes or cells. *Responsibility for the entire space can then be distributed. Any such approach, however, must implement some mechanism for propagating the query and update messages around the distributed system. The efficiency of this process can greatly effect the scalability of such systems. In this paper we evaluate an adaptive approach to this problem, designed for use in cell-based systems in general.*

1 Introduction and Related Work

Large Distributed Virtual Environments (DVEs) consisting of many clients interacting in a shared space, pose well recognised scalability challenges. They also have properties - namely the limited proportion of the space with which an individual client interacts - which make it possible to meet these challenges through interest management (IM). Particularly in the case of real-time 3D worlds the fundamental mode of interest expression is the *range-query*. In such systems the IM problem is reduced to the problem of comparing range queries against updates and forwarding to the querier only those which match.

The matching task can be performed by a central node. Such a solution is used, for example, in all modern First Person Shooter (FPS) distributed multiplayer games such as Quake [15]. This solution, though robust for small numbers of participants, does not scale and much research has sought to distribute the task of filtration. In many approaches the logical space over which range queries are applied is divided in to contiguous, non-overlapping cells (for example, in to a uniform grid, though non-uniform, non-rectangular decompositions are common). We collectively term these approaches 'cell-based interest management'.

In this paper we present and evaluate an algorithm designed for deployment within a generic reference model of a cell-based framework. The aim of the algorithm is to take an adaptive approach to the distribution of query and update information in order to minimise the communication overhead of the matching process.

The remainder of the paper is organised as follows: section 2 reviews the large body of existing work in to cell based IM; section 3 outlines our algorithm and briefly details previous work for a different form of data-access pattern upon which the algorithm is based; section 4 describes the methodology for our evaluation of the algorithm; section 5 presents results of the evaluation; and section 6 discusses the direction of future work implied by these results.

2 Related Work

Two disjoint branches of research have both looked extensively at various cell-based approaches to IM. We review this work here.

2.1 IM in Distributed Simulation

In the area of real-time distributed simulation, the earliest approaches appeared in support of the Distiributed Interactive Simulation protocol (DIS). Both the work of the NPS-NET group [21] and of Van Hook et. al. [13] used an approach with cells mapped on to multicast groups at the network level to perform a coarse-grained, highly decoupled form of matching. Data Distribution Management (DDM) is the IM framework integrated in to the specification of the High Level Architecture (HLA) for distributed simulation [17] which largely replaced DIS as the key framework for real-time distributed simulation. In the DDM architecture application nodes ('federates') express potential events (publications) and their interests in them (subscriptions) through n-Dimensional *regions*. Subscription regions in particular are of interest to us as they are the HLA equivalent of a range query. Most proposed implementations of this framework again use IP multicast to divide the entire n-Dimensional space in to a grid. When, as with the earlier DIS work, this assignment is static [30], this approach is termed 'fully distributed' as no communication is required between federates to locate which other federates contain data for a particular cell as rendevous duties are handled by the IP multicast implementation. When the assignment is dynamic (for example if load balancing dictates a need to alter the cell granularity in different regions of the space) some form of centralised/broadcast coordination is required [14, 4, 8]. Though multicast seemed a highly attractive prospect for implementation of DDM, experiences [29, 12] indicate its deployment has not been efficient or widespread enough to allow these techniques to achieve the performance suggested by experimental results.

2.2 IM through Peer-to-Peer Overlays

The other community that has looked extensively in to IM using cell-based approaches is the structured peer-to-peer (P2P) overlay community. A structured P2P overlay (also commonly referred to as a 'Distributed Hash Table' - DHT) is an application-layer routing structure over a set of network nodes that can locate a named data item within an n node network in a number of hops $O(logn)$ [31, 38, 33]. These systems position both data items and nodes in the logical space of the overlay using a single hash function on some unique identifier of both types of object (e.g. an IP address for a node, or an attribute handle for a variable). The node with the position closest to a given object's position assumes responsibility for it. This creates a problem for referencing volumes of variable size (as in a range query) as the DHT abstraction does not readily support this kind of *associative* memory access. Many approaches solve this problem by a static cell decomposition where each cell's coordinate in the n-Dimensional grid is treated as its identifier [18, 11, 32, 1, 35, 9][1]. Other systems [3, 34] take a slightly different approach of using the application's coordinate system itself rather than the hash space for routing in the overlay. This has the advantage of allowing a dynamic cell decomposition but the disadvantage of increased probability of the need for explicit load balancing amongst cells leading to a higher cost of maintaining efficient route information. In recent years several architectures have appeared which propose cell-based IM over a DHT as an infrastructure for scalable P2P multiplayer games [5, 18, 39].

[1]Note that, to some degree, these approaches can be seen as equivalent to fixed-grid DDM implementations using application-layer multicast protocols based on a DHT.

3 Design

In previous work we presented and evaluated a framework which aimed to reduce total message load across a system of distributed variables by switching responsibility for data flow between readers and writers. This framework, first presented in [23] is briefly described here.

3.1 The Basic Push-Pull Framework

We abstractly represent a system as a set of uniquely identified variables and a set of nodes each hosting an application which issues reads and writes to the set of variables. Each variable is associated with a particular *owner* node. All writes to a variable are sent to that variable's owner, thus ensuring consistency via this *master* copy. All nodes who are not the owner of a particular variable are termed *replicators* of that variable. These maintain a *proxy* locally which can be read and written by the hosted application in a transparent way. The state of proxies can be maintained in one of two processing modes:

- *Push* wherein each write at the master results in an update message sent to the proxy
- *Pull* wherein each read at the proxy results in a read-request/read-response exchange initiated by the proxy

The general operation of the system is shown in figure 1.

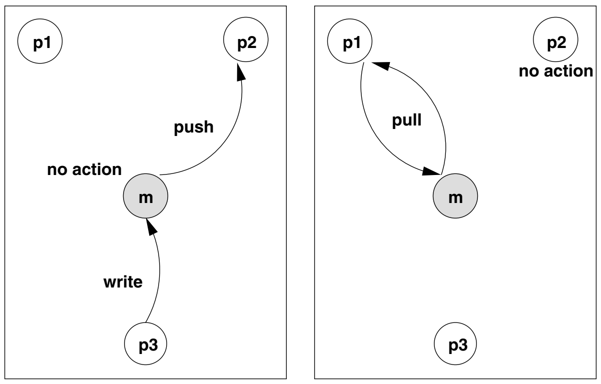


Figure 1. Operation of the push-pull system. Three proxy variables, p_1,p_2 and p_3 replicate the master variable m. p_1 is in pull mode while p_2 is in push mode. On the left is the behaviour when a write occurs, on the right, when a read occurs at both p_1 and p_2.

For each master-proxy pair in the system the framework implements an online algorithm which continually selects over time the processing mode that will produce the smallest number of messages. We call this action of selection an 'edge-flip' and define a handshake protocol that allow either a master or a proxy to initiate one. In previous work [24] we used a simulated testbed to evaluate various such algorithms against Push-only, Pull-only and Optimal baselines finding that a simple heuristic based approach - the *Demand-Push* algorithm - produced a result acceptably close to optimal when read-write patterns had at least some degree of predictability.

3.2 Applying Push Pull to Cell-Based IM

In high-level terms our approach for applying Push-Pull to cell-based IM frameworks is to consider the contents (or observers) of a cell as being equivalent to a variable, and the processes of querying and updating the objects within a cell as analogous to the reads and writes which occur on that variable. This design derives from one of three first presented in [25].

We assume a system which conforms to the following properties:

- A system composed of several nodes (computational locations in a physical network) runs a DVE application wherein some set of nodes periodically issue both updates to data items and range queries over volumes of the space. This set of nodes are termed application nodes.
- A range query is a persistent subscription to a volume, updates to which must be forwarded to the querying node. Updates, however are not persistent in the sense that a newly arriving range query over volume V issued at time $t(V)$ is **not** matched to updates U if $t(U) < t(V)$.
- Cells of unspecified dimensionality are assigned in some way to nodes in the system, these nodes are termed *cell hosts* . Though these cell hosts *may* also be application nodes (i.e. a fully P2P system), our design assumes that they are not (though this assumption does not change the algorithms used in any way).
- Given a range query or update an application node can determine to which cell(s) the given object corresponds and, through some unspecified mechanism (e.g. DHT routing; static assignment), can establish network communication with the relevant cell hosts.

Within this system, cell hosts store a set of ids of application nodes, identifying which application nodes have range queries overlapping some portion of the volume of the cell. Such a publish-subscribe system inherently trades off the granularity of filtering against the frequency of change of an application node's subscription set. Whilst this issue is not directly addressed by our algorithm, it does have an important interaction with performance which we evaluate in section 5.

If a cell is then considered a variable, this set of ids is therefore considered the variable's *value*. Querying application nodes modify the value (they are 'writers') while updating application nodes require the value in order to send updates to application nodes in the subscribing set (they are 'readers'). In this architecture the flow of reading and writing is depicted in figure 2. In this figure two application nodes, u_1 and u_2, are issuing updates to a cell host to which two other application nodes, q_1 and q_2, are subscribed. u_1 is considered to be in 'push-mode' as the cell pushes all new subscriptions to it, whilst u_2 is considered to be in 'pull-mode' as the cell sends the subscription set only upon receiving a request. Both updaters send the matching updates directly to the subscribing queriers.

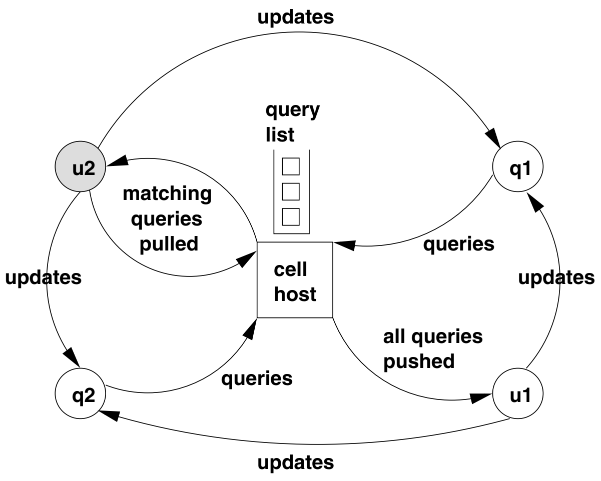


Figure 2. A literal application of Push-Pull to cell-based IM

3.2.1 Algorithms

This very literal application neatly demonstrates the basic principle upon which we will apply Push-Pull in this architecture. However, this first design is quite inefficient if one considers the flow of data in the case of u_2. Here the application node first sends an update notification to the cell host, which then responds with a subscription set, the updates finally being forwarded to each member application node in the set. This number of steps can be greatly reduced if u_2 were to simply send the update itself, requesting the cell host to forward the update to the subscribing set. u_2 itself has no particular interest in the subscribing set, as it will have to request it again the next time it updates anyway. The two alternative approaches are depicted in figure 3.

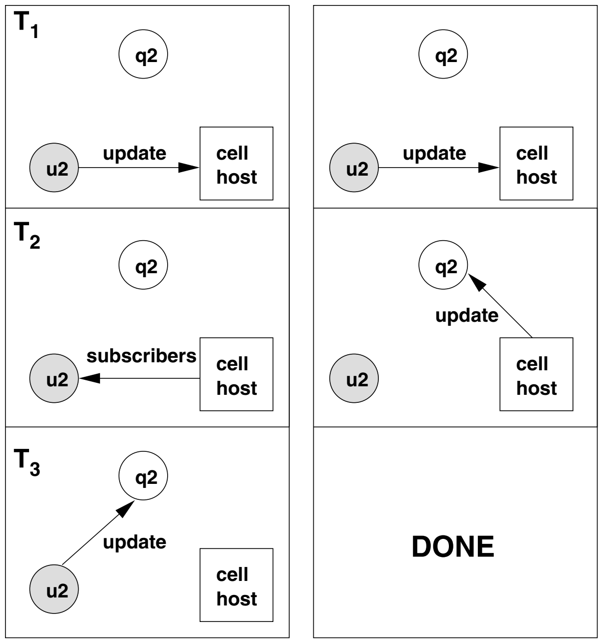


Figure 3. Steps taken by u_2 in the literal design (left) and the more rational design (right)

Note that in terms of the Push-Pull algorithm, we need to change nothing, as the optimisation task is still a question of predicted read rate (rate of updates issued) vs. predicted write rate (rate of change of subscription set at the cell). Minimising the message load across the updater-cell host link is still a function of these two factors.

To reflect the fact that 'push' and 'pull' no longer satisfactorily characterise the choice of processing model, we will instead adopt the terms cell-filtering (previously 'pull') and client-filtering (previously 'push')[2].

4 Evaluation: Methodology

To evaluate our algorithm we use a trace-driven simulation of a network of cell hosts and application nodes. In the simulation we feed player action inputs to application nodes which translate these (according to the particular algorithm being evaluated) in to range query and update messages to either other application nodes or to cell hosts. Since we are not currently interested in evaluating a particular form of routing architecture we simply model the system as a fully connected graph where any node can message any other node in a single hop. This architecture is depicted in figure 4.

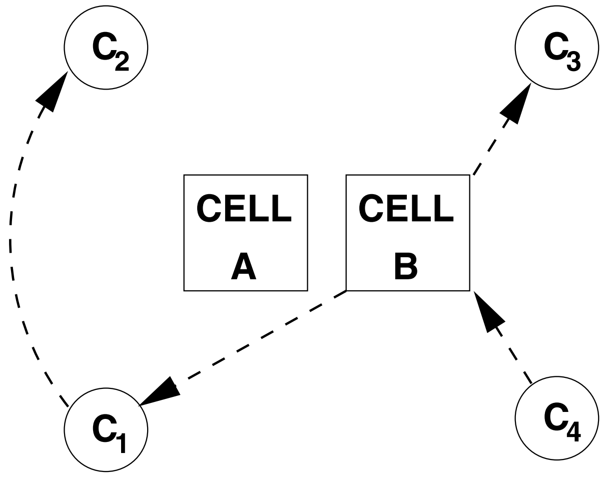


Figure 4. A depiction of the simulation setup. Client c_1 is using client-filtering while c_4 is using cell-filtering

The traces used to drive the simulation are harvested from a running instance of the First-Person Shooter (FPS) game Quake 2 [15]. The Quake series of games is a popular and, in design terms, highly influential reference model for the FPS genre. The engine itself has been used in several other popular FPS titles [37, 36]. The series has pursued a far more open model of development than many, with both a very open architecture allowing for extensive user-created content and the release of source code under a GPL license [15]. The combination of Quake's ubiquity as a reference model and its openness as an architecture have lead to its widespread adoption as a research testbed for traffic analyses [2, 19] and distributed architectures [6, 27] of multiplayer games.

The Quake 2 game logic is built around a Client-Server architecture in which a client sends control messages to a server which responds with game states (in single-player mode this architecture is retained, with messaging occuring on the loopback interface). A built in function allows the recording of 'demo' traces in the form of '.dm2' files which represent the precise message exchange across the server-client interface. Quake 2 supports scripted dummy players, or 'bots', designed for training real players for multiplayer competition. The game also supports custom maps which can be composed in various editing applications (we used the GtkRadiant package [16]).

By running a single instance of Quake 2 with some number of bots playing against each other using a given map, we can quickly and efficiently parameterise and harvest dm2 traces which approximate closely the real application be-

[2]Note that this also sits more comfortably with the terminology of existing IM taxonomies for information processing (e.g. [26]).

haviour of a network of human players.

For our experiments we created a map whose dimensions were at the limit of the Quake 2 engine's storage capacity (3072 game units[3]). The map was a cube consisting of 12 floors each separated in to 4 rooms and joined by staircases at each corner. Figures 5 and 6 respectively show screenshots of a cross section of the entire map and a single level from above.

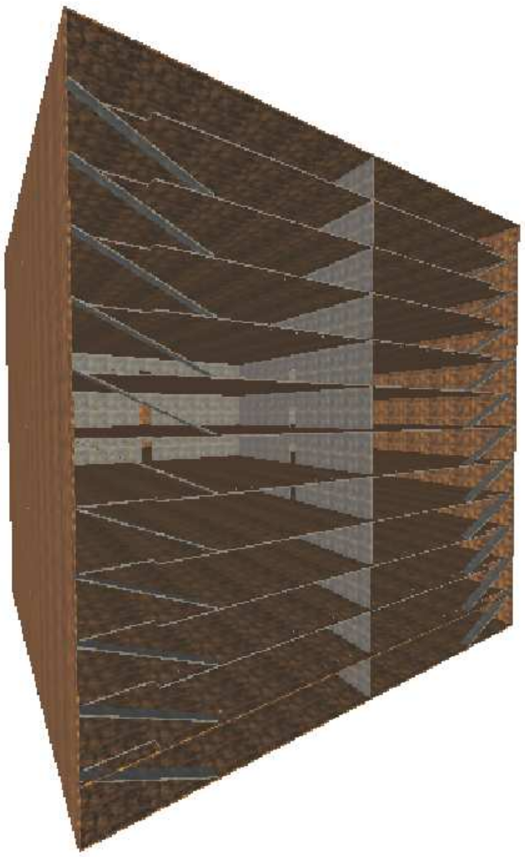


Figure 5. A cross section of the map used to generate traces.

Over this map we set up games between varying size populations of bots (between 2 and 28). The bots would play for approximately 5 minutes, which equates to just over 3000 frames on the hardware used. In post-processing the trace was trimmed of all but the final 3000 frames, this allowed us to obtain uniform length whilst ensuring that play had settled to a reasonable pattern of interaction before the beginning of the trace itself.

4.1 Quake 2 Trace Schema

The game logic of Quake 2 is a simple time-stepped simulation where the complete state of the game world (the union of the individual states of all entities in the world) is termed a 'frame'. The dm2 schema (which is more exhaustively detailed in [10]) is then a sequence of frame objects, each of which contains a set of entity states.

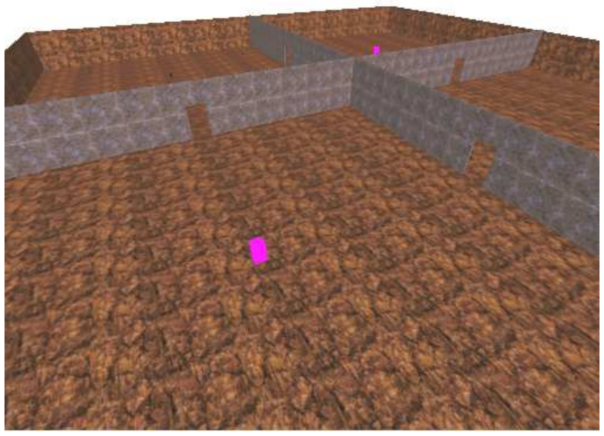


Figure 6. A single level of the map from above. The small square object represents the start location of a bot.

Using these state tuples to represent updates issued by nodes to the shared world view, we can both assign individual updates to application nodes and also determine the dimensions of an application node's range query in a given frame.

This pre-processing turns a dm2 file in to a usable trace to drive our simulation from the perspective of application nodes. In addition to this we must divide the whole state space in to cells to be assigned to cell hosts. This parameter is termed 'grid-granularity'.

4.2 Evaluation Metrics

The problem of IM thus far has been characterised as how to reduce the message load a distributed application places on a network. In fact the problem is strictly that of how to reduce the message *latency* experienced by an application.

In our experiments we consider only total message load, with our simulation using a network abstraction in which a single message from one node to another has an associated *cost* of 1. Although this metric does not explicitly capture the effect of network congestion on the latency experienced by application nodes we chose it as it is agnostic to both network topology and messaging architecture (e.g. IP multicast or DHT routing over IP). Furthermore, latency-compensation techniques in several DVE implementations make the true per-node latency the result of several complex interactions of IM algorithm, messaging architecture, topology *and* techniques like dead-reckoning and frame-aggregation. For each of these parameters a reference model would need to be applied, eventually making the actual effect of the algorithm extremely hard to isolate.

Whilst this kind of 'in the wild' analysis of an IM algorithm is very important, this current study will only consider the metric of total messages exchanged between nodes.

5 Evaluation: Results

We chose to study the behaviour of the algorithm with relation to two parameters: the granularity of the grid; and the number of players. Each of these parameters directly effects both the flow of data between nodes (larger granularity leads to fewer updates being sent to queriers; larger populations leads to greater likelehood of interaction) and the constituents of the network itself (one cell host per-grid cell; one application node per player).

We express grid granularity as the per-dimension division, so varying from 1 (basically pure client-server) to 8 increases the cell count logarithmically from 1^3 to 8^3.

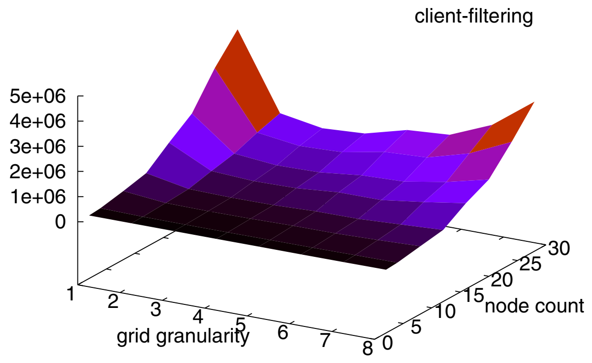
We ran experiments for each permutation of these parameters, using each of cell-filtering, client-filtering and Push-Pull algorithms.

We also defined a hypothetical 'Optimal' algorithm for benchmarking purposes. This algorithm is generated by a function which scans a known network-wide sequence of update and query operations that will occur. It then uses an heuristic search of the set of all possible sequences of edge-flips to find the one which generates the fewest messages. This algorithm is a variation of the basic Push-Pull form described in [24].

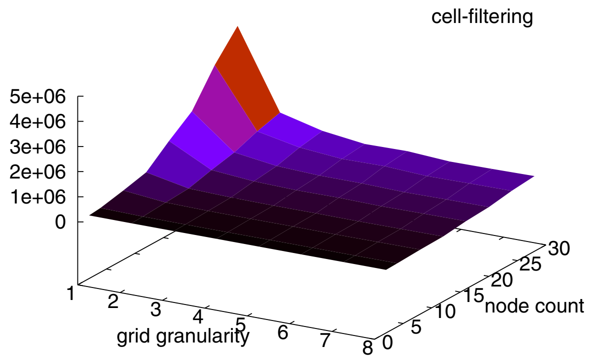
The total count of all messages in the network for each point in this space is shown in figures 7 and 8.

These two figures illustrate two important characteristics: the relationship between granularity and population count; and the relationship of each algorithm to the others.

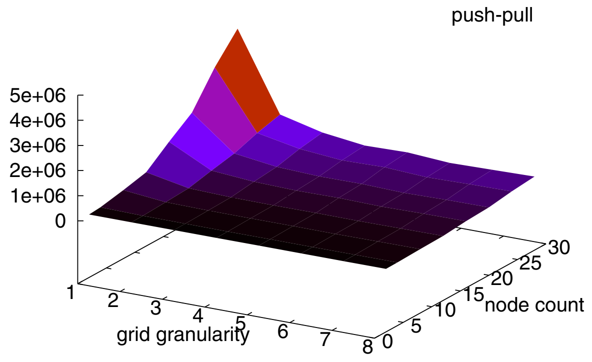
The most important comparison in figure 7 is between figures 7(a) and 7(b). Here we see in the case of client-filtering that both high and low granularities increase message load, whilst in the case of cell-filtering the effect is seen only at low granularities. The cause of low granularity message load is the number of extraneous updates sent to subscribers, this is of course something all three algorithms suffer from. As granularity increases logarithmically, so we see a direct correlation in the reduction of message load. However, in the case of client-filtering more cell hosts means more queries being propagated to *all* application nodes. The fundamental n^2 nature of the client-filtering algorithm (each range query is sent to $n-1$ other nodes) means that quite quickly this type of message dominates the load. This is not the case in cell-filtering as queries



(a) client-filtering

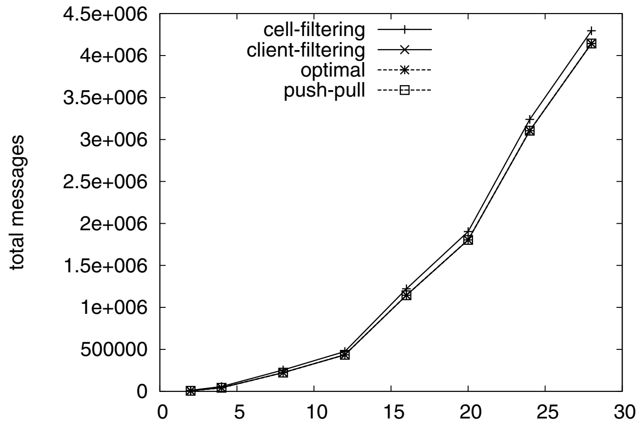


(b) cell-filtering

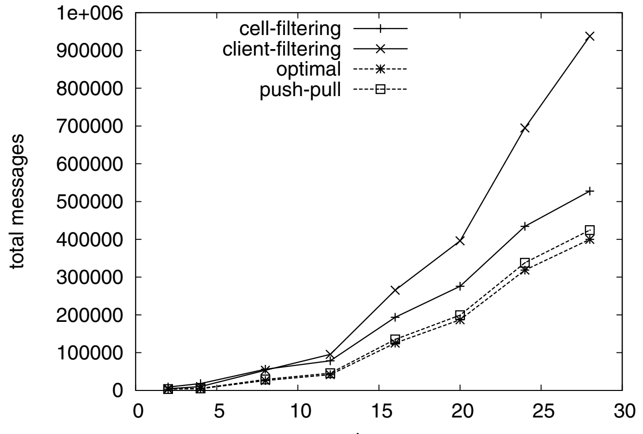


(c) Push Pull

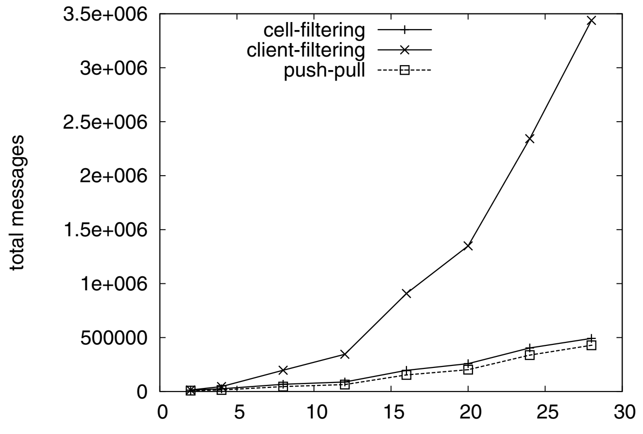
Figure 7. Total messages sent for each combination of player count and grid granularity by different algorithms.



(a) 1x1x1 Grid



(b) 4x4x4 Grid



(c) 8x8x8 Grid (Optimal algorithm not shown due to complexity of computation in an 8^3 cell network)

Figure 8. Total messages sent for each algorithm across different granularities.

stop at the cell host. Ultimately however, even in the case of cell-filtering, there is always a point at which the advantage of finer-grain filtering of updates becomes less than the burden of application nodes sending queries to cell hosts.

This tradeoff is the key characteristic of figure 7 and looking at figure 7(c) we can see that the Push-Pull algorithm most closely approximates cell-filtering in this respect. From this we can infer that Push-Pull is recognising the high load of queries being received at application nodes when grid granularity is high. It is responding to this by flipping the vast majority of application nodes to be in cell-filtering mode.

This facet of Push-Pull (choosing the best filtering mode on aggregate) is important but its real strength is in providing an edge-wise selection. This leads us to expect Push-Pull to produce fewer messages than either of the other algorithms, even when one is orders of magnitude worse than another on aggregate (i.e. as client-filtering is at high granularities). Figure 8 shows this behaviour with Push-Pull consistently producing fewer messages than than either of the other approaches, even when (as in 8(c)) one is extremely poorly performing.

6 Conclusions and Future Work

We have shown that, with some slight conceptual modifications, the basic Push-Pull framework can be applied to the problem of distributing query and update information in cell-based IM frameworks. The core algorithm needs no modification in this new setting and, predictably therefore, we closely replicate the original algorithm's ability to minimise total message load in the network.

In this paper there are two aspects of the algorithm's behaviour we have not yet evaluated: the first is its effect on the distribution of load (both in terms of bandwidth and in terms of computation) around the network; the second is the precise effect of total network load on the perceived latency at the application nodes.

We are currently studying Push-Pull's effect on load distribution. Whilst results of these analyses are out of the scope of this paper, they have produced some unexpected indicators that suggest that the behaviour of an implementation of the algorithm may be strongly dependent on the network's topology and constitution.

As discussed in section 4 a systematic evaluation of latency is inherently a more specific evaluation of a particular messaging architecture, IM algorithm and application behaviour. Nonetheless, this study of the algorithm 'in the wild' is a very important piece of future work. This work will also reflect our findings on load distribution mentioned above.

Finally, we have here focussed on applying the algorithm to classical DVE implementations (distributed virtual reality; first-person shooter games; etc.). There are applications, such as massively multiplayer online games [7, 28] or virtual worlds [20, 22] which are less sensitive to latency, have more gradual changes in patterns of interaction and which, in some cases, exhibit far higher degrees of interest-limitation. Such application types have not been heavily studied by either the DDM or the P2P overlay communities mentioned above (partly due to the comparative lack of open-source examples or reliable reference models for such applications). Given the characteristics of these applications they appear to have good potential for applying adaptive interest management approaches such as the one described herein.

References

[1] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Proceedings fo the Second International Conference on Peer-to-Peer Computing (P2P'02)*, pages 33–40, 2002.

[2] G. Armitage. An experimental estimation of latency sensitivity in multiplayer quake 3. In *Proceedings of the 11th IEEE International Conference on Networks*, 2003.

[3] A. Barambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of the ACM SIGCOMM 2004*, pages 353–366, 2004.

[4] A. Berrached, M. Beheshti, O. Sirisaengtaksin, and A. deKorvin. Approaches to multicast group allocation in HLA data distribution management. In *Proceedings of the 1998 Spring Simulation Interoperability Workshop*, 1998.

[5] A. Bharambe, J. Pang, and S. Seshan. A distributed architecture for multiplayer games. Technical Report CMU-CS-05-112, Carnegie-Mellon University, 2006.

[6] A. R. Bharambe, S. Seshan, and J. Pang. A distributed architecture for interactive multiplayer games. Technical Report CMU-CS-05-112, School of Computer Science, Carnegie Mellon University, January 2005.

[7] Blizzard Entertainment. World of warcraft. World Wide Web, `http://www.worldofwarcraft.com/`, March 2006.

[8] A. Boukerche, A. J. Roy, and N. Thomas. Dynamic grid-based multicast group assignment in data distribution mangagement. In *Proceedings of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications (DSRT 2000)*, 2000.

[9] A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer. Range queries in trie-structured overlays. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer computing*, 2005.

[10] T. Ferguson. Quake ii network protocol specs. World Wide Web, `http://www.csse.monash.edu.au/~timf/bottim/q2net/q2network-0.03.html`, Published 1998, Retrieved 2007.

[11] A. Harwood and E. Tanin. Hashing spatial content over peer-to-peer networks. In *Proceedings of the Australian Telecommunications, Networks and Applications Conference*, December 2003.

[12] B. Helfinstine, D. Wilbert, M. Torpey, and W. Civinskas. Experiences with data distribution management in large-scale federations. In *Proceedings of the 2001 Fall Simulation Interoperability Workshop*, 2001.

[13] D. J. V. Hook, S. J. Rak, and J. O. Calvin. Approaches to relevance filtering. In *Proceedings of the 11th DIS Workshop*, 1994.

[14] M. Hyett and R. Wuerfel. Implementation of the data distribution management services in the RTI-NG. In *Proceedings of the 2001 Fall Simulation Interoperability Workshop*, 2001.

[15] id Software. Quake technology page. World Wide Web, `http://www.idsoftware.com/business/techdownloads/`, December 2004.

[16] id Software. Qe radiant website. World Wide Web, `http://www.qeradiant.com/`, 2007.

[17] IEEE. IEEE 1516 (Standard for Modelling and Simulation High Level Architecture Framework and Rules), 2000.

[18] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-Peer support for massively multiplayer games. In *Proceedings of INFOCOMM 2004*, March 2004.

[19] T. Lang, P. Branch, and G. Armitage. A synthetic traffic model for quake3. In *Proceedings of the ACM International Conference on Advances in Computer Entertainment Technology*, pages 233–238, 2004.

[20] Linden Labs. Second life. World Wide Web, `http://secondlife.com/`, December 2004.

[21] M. Macedonia, M. Zyda, D. Pratt, D. Brutzmann, and P. Barham. Exploiting reality with multicast groups: A network architecture for large-scale virtual environments. *IEEE Computer Graphics and Applications*, 15(3):38–45, 1995.

[22] Mindark. Entropia universe. World Wide Web, `http://www.entropiauniverse.com/`, May 2007.

[23] R. Minson and G. Theodoropoulos. An adaptive interest management scheme for distributed virtual environments. In *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation (PADS), 2005*, pages 273–281. IEEE, 2005.

[24] R. Minson and G. Theodoropoulos. Adaptive interest management via push-pull algorithms. In *Proceedings of the 10th International Symposium on Distributed Simulation and Real Time Systems*. IEEE, 2006.

[25] R. Minson and G. Theodoropoulos. Adaptive support of range-queries via push-pull algorithms. In *Proceedings of the workshop on principles of advanced and distributed simulation (PADS)*, 2007. To appear.

[26] K. L. Morse. *An Adaptive, Distributed Algorithm for Interest Management*. PhD thesis, University of California, Irvine, 2000.

[27] J. Müller, A. Gössling, and S. Gorlatch. On correctness of scalable multi-server state replication in online games. In *Proceedings of the 5th Workshop on Network and Systems Support for Games*, 2006.

[28] NC Soft. City Of Heroes. World Wide Web, `http://www.cityofheroes.com/`, December 2004.

[29] M. J. Pullen, M. Myjak, and C. Bouwens. Rfc:2502, limitations of internet protocol suite for distributed simulation in the large multicast environment. World Wide Web: `http://www.ietf.org/rfc/rfc2502.txt`, February 1999.

[30] S. J. Rak, M. Salisbury, and R. S. MacDonald. HLA/RTI data distribution management in the synthetic theater of war. In *Proceedings of the Fall 1997 DIS Workshop on Simulation*, 1997.

[31] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 161–172, 2001.

[32] S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Range queries over DHTs. Technical Report IRB-TR-03-009, Intel Research, Berkley, 2003.

[33] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2003.

[34] O. D. Sahin, A. Gupta, D. Agrawal, and A. E. Abaddi. A peer-to-peer framework for caching range queries. In *Proceedings of the 20th International Conference on Data Engineering (ICDE '04)*. IEEE, 2004.

[35] C. Schmidt and M. Parashar. Enabling flexible queries with guarantees in P2P systems. *IEEE Internet Computing*, 8(3):19–26, 2004.

[36] Steam. Counter-Strike. World Wide Web, `www.counter-strike.net`, May 2007.

[37] Steam. Half-Life 2. World Wide Web, `http://half-life2.com/`, May 2007.

[38] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 149–160, 2001.

[39] S. Yamamoto, Y. Murata, K. Yasumoto, and M. Ito. A distributed event delivery method with load balancing for mmorpg. In *Proceedings of the 4th ACM SIGCOMM Workshop on Network and System Support for Games*, 2005.