Computer Networks 56 (2012) 1731–1744

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

P2P consistency support for large-scale interactive applications

Yi Hu *, Laxmi N. Bhuyan, Min Feng

Department of Computer Science and Engineering, University of California at Riverside, 900 University Ave., Riverside, CA 92521, USA

ARTICLE INFO

Article history:
Received 20 May 2011
Received in revised form 7 January 2012
Accepted 16 January 2012
Available online 31 January 2012

Keywords:
P2P networks
Consistency maintenance
Computer games

ABSTRACT

Peer-to-Peer (P2P) systems have been widely used by networked interactive applications to relieve the drawback and reduce the reliance on well-provisioned servers. A core challenge is to provide consistency maintenance for a massive number of users in a P2P manner. This requires propagating updates on time by only using the uplink bandwidth from individual users instead of relying on dedicated servers. In this paper, we present a P2P system called PPAct to provide consistency maintenance for large-scale fast-interactive applications. We use massive multi-player online games as example applications to illustrate PPAct. The design can be directly applied to other interactive applications. We adopt the Area-of-Interest (AOI) filtering method, which is proposed in prior works [1,2], to reduce bandwidth consumption of update delivery. We solve the AOI's critical problem of bandwidth shortage in hot regions by dynamically balancing the workload of each region in a distributed way. We separate the roles of view discovery from consistency maintenance by assigning players as "region hosts" and "object holders." A region host is responsible for tracking objects and players within a region, and an object holder is responsible for sending updates about an object to interested players. Lookup queries for view discovery are processed by region hosts, while consistency maintenance of objects is taken by object holders. Separating the roles not only alleviates the workload overflow in hot regions, but also speeds up view discovery and update delivery. Another key idea is that peers contribute spare bandwidth in a fully distributed way to forwarding updates about objects of interest. Thus popular, high-demand objects will have more peers forward updates. We also present how to select capable and reliable players for region hosts and object holders.

A P2P network simulator is developed to evaluate PPAct on two major types of online games: role-playing games (RPGs) and first-person shooter (FPS) games. The results demonstrate that PPAct successfully supports 10,000 players in RPGs and 1500 players in FPS games. PPAct outperforms SimMud [2] in RPGs and Donnybrook [3] in FPS games by 40% and 30% higher successful update rates respectively.

1. Introduction

Modern networked interactive applications, such as online games [3], performance monitoring [4], and collaborative editing [5,6], have moved from dedicated server designs to P2P paradigms for better scalability and robustness. A core challenge for P2P systems is to provide consistency maintenance for all users by only using resources from these users.

P2P managed massive multi-player online games (MMOGs) are representative applications that have been extensively studied [1–3]. Some online game platforms, e.g., Microsoft XBox Live [7], have deployed their P2P services for commercial use. P2P systems for managing MMOGs have the following goals: (1) scalability to serve a large number of players; (2) consistency support for high-frequency interactions; (3) compatibility to different types of games; (4) easy configurability by heterogeneous players. Unfortunately, there is no existing P2P system that achieves all these goals. In this paper, we design a P2P

* Corresponding author.
E-mail address: yihu@cs.ucr.edu (Y. Hu).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2012.01.012

system, PPAct, that accomplishes all these goals supporting large-scale fast-interactive applications. MMOG is used as an example to illustrate how these goals are satisfied.

Insufficient uplink bandwidth. Online game applications require that each player should have a correct view of game status. The views of all players should be consistent. Thus, a player needs to get timely updates. In the worst case, a player needs every update from all other players to maintain consistency. The total number of updates sent by players is quadratic to the total number of players. However, the total supply of uplink bandwidth used for update delivery increases only linearly with the number of players. The asymmetric bandwidth allocation is dominated by downlink and thus uplink bandwidth is quite limited for normal players. The mismatch between the supply and demand of uplink bandwidth is a major obstacle to good scalability.

We adopt Area-of-Interest (AOI) filtering [1,2] to reduce the number of updates sent to each player. The game map is partitioned into regions, and each player receives updates about surrounding regions. These regions are called *view regions*. Each player subscribes to appropriate view regions in real-time. However, the AOI filtering only alleviates bandwidth shortage but does not solve it fundamentally.

We make additional bandwidth savings by reducing the overhead of view region discovery. We organize all regions into a two-dimensional DHT (2-D DHT) based on their horizontal and vertical positions. As game mobility studies [8] suggest, player movements are mostly continuous. A 2-D DHT provides low-cost region discovery for continuous movements. In addition, the frequency of receiving updates from subscribed regions is tuned temporarily to offset the gap between the supply and demand of bandwidth due to update bursts. Since new updates are sent out periodically, players may ignore obsolete updates in favor of newer ones.

Workload imbalance. The interests of players are usually clustered to a few regions according to the power law distribution of player population density [9]. The players who are responsible for sending updates about the hot regions may not have enough bandwidth to deliver them on time, which is called the AOI "hot spot" problem. A straightforward solution is to adjust region size related to popularity as in the previous works [10,1]. A popular region is divided into several small regions while an unpopular region is kept large. However, varying region size complicates region partitioning in P2P systems. Players change interests as the game evolves, consequently, locations and sizes of hot areas keep changing. It is hard to correctly partition regions because all players should be synchronized and provided with such frequently changing information. Another solution is migrating peers from unpopular regions to popular regions to share the workload as in the previous work [11]. However, it is difficult to correctly perform migration in dynamic P2P systems. In PPAct, we use a uniform region size so that region partitioning information is easily maintained by every player. In this paper, we use peers and players interchangeably, both mean the users of a P2P system.

We solve the AOI "hot spot" problem by breaking down the workload of each region. We separate the object discovery from the object consistency maintenance by assigning each region to a *region-host* and each object to an *object-holder*. A *region-host* is a peer who tracks objects inside a region and players who want to receive updates about this region. An *object-holder* is a peer who holds the primary copy of an object and sends out updates about the object. All objects inside a region are registered to the region-host by their object-holders. Thus, players only query the region-host to subscribe to all objects within that region.

We further break down the workload of update delivery by having each player contribute spare bandwidth to subscribed objects. This avoids overloading object holders of popular objects. Since demand for receiving updates of an object is determined by the number of object subscribers, our scheme ensures that the objects with higher demands have more players to contribute. Moreover, bandwidth is contributed in a fully distributed manner as players piggyback bandwidth information onto their region discovery queries without adding extra overhead.

From our observation, the workload of a region host is more stable than that of an object holder. The number of regions assigned to a region host varies by the region host's capacity. When assigning multiple regions to a peer, we avoid assigning neighboring regions to the same peer so the regions are less likely to have player crowds simultaneously. A region host should also be reliable since it is the contact point for players and object holders. We develop an analytical model for region host selection with both capacity and reliability threshold requirements.

Incompatibility to various games. There are two types of popular online games: action games and role-playing games (RPGs). Each type has its distinctive features and thus different application requirements. An action game requires players to use quick reflexes, accuracy, and timing to overcome obstacles. A popular sub-genre is called first person shooter (FPS). Thus, an action game involves frequent updates and imposes stringent latency bounds for update delivery. Most RPGs cast the player in the role of one or more "adventurers" who specialize in specific skill sets and progress through a predetermined storyline. Compared to FPS games, RPGs have lower update rates and tolerate to longer update latency, but support a larger number of players. RPGs have updates from a great number of Non-Player Characters (NPC) and player avatars, unlike FPS games, which only have updates from player avatars [12]. Existing techniques may address either game type but overlook adaptability. For instance, Donnybrook [3] works well for FPS games, but only scales up to 900 players and does not handle NPCs for RPGs. Another work, SimMud [2], manages NPCs for RPGs, but can rarely meet the latency bound of FPS games.

PPAct is the first P2P system designed to be compatible for different types of games. The separation of region-hosts from object holders not only speeds up update delivery but also provides flexible consistency maintenance for both avatars and NPCs. With all these fully distributed techniques, PPAct satisfies the stringent update latency bound of FPS games and provides good scalability for RPGs. Our extensive simulations show that PPAct successfully supports 10,000 players in RPGs with successful update rate

40% higher than that of SimMud [2], and 1500 players in FPS games with successful update rate 30% higher than that of Donnybrook [3]. The paper makes the following contributions while designing the new PPAct system for maintaining consistency in a large interactive application.

(1) We decouple the roles of view discovery and consistency maintenance to speed up lookup and update delivery for the stringent latency requirements of FPS games, and to flexibly support object consistency maintenance for RPGs.
(2) We propose a scheme that balances the workload of update delivery dynamically by having each player contribute its spare bandwidth to its subscribed objects. This solves the AOI's problems of bandwidth shortage in hot regions.
(3) We present schemes for selecting region hosts and object holders with capability and reliability requirements.
(4) We develop a simulator to extensively evaluate the performance of PPAct on two major types of online games. The performance of our work is compared with two recent P2P game systems: Donnybrook [3] on FPS games and SimMud [2] on RPGs. The major design factors are also verified through simulation results.

The rest of the paper is organized as follows. Section 2 gives the background on consistency maintenance of online games. Section 3 and Section 4 present PPAct techniques for fulfilling our design goals. Section 5 shows simulation results. Section 6 reviews prior work, and Section 7 concludes the paper.

2. Background

In most MMOGs, each player plays a role in a virtual world. Each player is represented in the game by an avatar. A game includes both immutable and mutable objects. Immutable objects (e.g., landscape) are downloaded by game client software. Mutable objects are updated by either players (e.g., avatars, food, tools) or automated algorithms (e.g., NPCs). Every mutable object is represented by its state. For example, an avatar's state may include its position, health, possessions; a tool's state may include its position, shape or content. An NPC's state is similar to an avatar's. The only difference is that each NPC is associated with an automated algorithm for execution. The consistency maintenance updates the game states of all mutable objects correctly for every player. From here on, we use the term "object" to mean a mutable object.

Client–server systems use the primary copy consistency model for maintaining game state. The server maintains a primary copy of every object in the game and periodically sends updates to players. A game is implemented as a discrete event loop. Each loop iteration is called a "frame." The server broadcasts an update to every player by the end of each frame. The update broadcast includes updates from all mutable objects.

P2P systems follow the primary copy consistency model in client–server systems. Each object has a primary copy where all updates are serialized and sent out. However, all primary copies that were originally maintained by servers are transferred to different players, who are responsible for serializing and delivering updates every frame. Using bandwidth and computation resources from players relieves the reliance on game servers, which thus only do external support, such as authentication. The challenge is how to fully take advantage of the limited resources from a large number of unreliable players to achieve better scalability and robustness compared to a handful of resource abundant and dedicated servers. This challenge motivates the design of PPAct system.

3. Rendezvous enabled range query processing and subscription

In this section, we first give an overview of how a player obtains its view in PPAct. Then, we present the region partitioning scheme, and the update forwarding scheme. Finally, we present how to map regions to region hosts and objects to object holders.

3.1. Overview

We apply AOI filtering to reduce bandwidth consumption due to sending updates. AOI filtering takes advantage of spatial locality of player interests. Thus, each player gets a confined view in the game, which is an area centered at its avatar's current position with radius R, instead of the entire game map.

The latency bound for acquiring a view is stringent. For example, FPS games require that players' views should be synchronized every 150 ms [3]. This requires a player to receive updates about all objects in its view within every 150 ms. When a player is moving, its view changes accordingly. Hence, the latency bound for receiving an update covers two steps: view look-up and update delivery.

We separate the roles of view discovery from object consistency maintenance by assigning *region hosts* and *object holders*. A region host tracks both objects in a region and players whose views include the region. An object holder maintains the primary copy of an object and sends updates about the object to subscriber players. Players subscribe to all objects within their views for receiving updates. The subscription is performed by sending lookup requests to the region host, because dynamic movements of objects make it difficult for individual players to track all updates. After determining his subscribed regions, a player discovers those region-hosts through contact. In PPAct, region hosts are organized into a two-dimensional DHT overlay based on regions' horizontal and vertical positions. Thus, players are provided with two-hop lookup for frequent continuous movements and logarithmic lookup for the seldom random movements.

To assist region-hosts in tracking objects, object holders register objects with region hosts of their current regions. Once an object moves out of a region's boundary, the holder unregisters from the previous region-host and

registers with the new one. No real-time information, such as object's exact location, is involved in the registration. There is no need to update information at the region-host once an object is registered. Therefore, maintenance overhead of registration is low.

In summary, there are three steps for a player to get a view as shown in Fig. 1.

(1) **Lookup.** A player's view is computed based on its avatar's current position, and a lookup query is sent to the region host of each view region that has not yet been subscribed. An unsubscription message is sent to each region that has gone out of view but previously subscribed.
(2) **Subscription.** Every region-host maintains a *subscription list*, recording players who subscribed to this region in the last frame. An *unsubscription list* is maintained similarly. The two lists are sent to each object-holder who has registered with this region.
(3) **Update delivery.** After receiving the two lists, each object-holder sends out updates to all subscribers in every frame.

3.2. Region partitioning

The map of a game is partitioned into a set of disjoint regions. A region can be any shape defined by a center point and a radius. We choose circle regions with uniform radius *r*. Fig. 2(a) shows an example of region partitioning. The dotted circle represents a player's view where the center is its avatar's position. A player's view consists of seven adjacent regions. In this way, each view region can be flexibly subscribed when the player is moving and its view is changing.

Region size is an important factor to the efficiency and overhead of subscription, as the subscription is performed in unit of a region. If a region is so large that it covers the entire view, it is hard to accurately define the view. A range query from each player to look up view regions is transformed to a set of exact match queries, each of which locates a view region. Thus, large regions are too rough to approximate the range query. Fig. 2(b) shows a region partitioning where three regions cover a player's view. It is possible that some objects are inside the three regions but outside of the player's view. They will be unnecessarily included in the player's range query which wastes network resources. On the other hand, if a region is so small that a view consists of dozens of regions, the excessive overhead of query processing and object registration hand-off will degrade performance.

Vague region boundary. We apply *vague boundaries* between neighboring regions to reduce hand-off overhead when objects are frequently moving back and forth across boundaries. The shaded area in Fig. 2(a) shows the intersections of neighboring regions. The shaded area is called the "vague boundary" since there is no clear borderline between neighboring regions. An object-holder switches registration only when the object enters a new region and moves out of the shaded area. Objects in the shaded area delay switching registrations to avoid unnecessary hand-offs. Such delayed switchings do not negatively affect the results of range queries. The objects, which are excluded from the query results because of vague boundaries, reside at the border of the player's view. They are less visible than other closer objects that are returned in the results anyway. Hence, the player will not notice the difference.

3.3. Update forwarding and burst handling

Limited uplink bandwidth and heterogeneous object popularity give rise to the AOI "hot spot" problem [1,2], where some object-holders are overloaded by an excessive number of subscribers. Other object-holders with surplus uplink bandwidth may only have a handful of subscribers or perhaps none at all.

In PPAct, we balance workload by making subscribers contribute their spare bandwidth to forwarding updates to other subscribers. The procedure of update forwarding is described as follows.

First, each subscriber attaches a forwarding quota to its lookup query for each of its view region. A subscriber's forwarding quota is equal to its available bandwidth divided by the size of an update, where the available bandwidth is the total bandwidth subtracted by the bandwidth reserved for serving as a region host if it is. The forwarding quota indicates how many recipients a subscriber can forward to under the one hop delay constraint. A frame in FPS is no more than 150 ms and in RPG may be up to 180 ms [13]. Within a frame, at most three steps – subscribing, update delivering, and update forwarding should be completed. Thus, on average each step can use 1/3rd of the time, and one hop delay is set to be 50 ms to satisfy both categories. The measurement statistics of P2P online games in [4] confirm the feasibility of this setting, where more than 80% one hop delay is less than or equal to 50 ms.

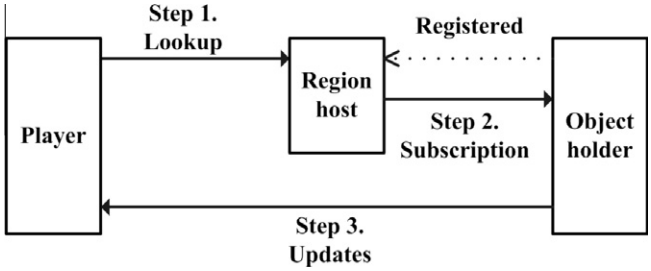


Fig. 1. The procedure for getting a view.

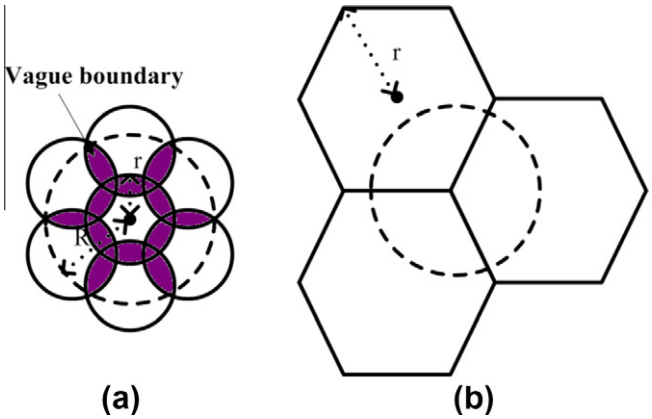


Fig. 2. Examples of region partitioning.

Second, after a region host collects the forwarding quota from all subscribers of the region, it evenly distributes the forwarding quota to every object holder in this region. Thus, the subscription list that a region host sends to each object holder in the region includes all subscribers and their forwarding quota dedicated to the object.

Finally, after obtaining the subscription list, an object holder selects a subset of subscribers as forwarders. It then sends update to each forwarder with a forward list according to each forwarder's quota. The number of forwarders is the maximum number of recipients the object holder can send to under the one hop delay constraint. Subscribers with larger forwarding quota are preferred when selecting forwarders.

Each region host evenly distributes the forwarding quota among object holders because all objects in a region are subscribed as a whole and each object holder needs the same amount of bandwidth for sending updates.

When more peers subscribe to a region, more forwarding quota is contributed to sending updates about the region. Therefore, with the forwarding scheme, the amount of bandwidth contribution is proportional to the demand for sending updates. Moreover, our forwarding scheme avoids sending updates to non-subscribers. We only use one-hop forwarding due to the extended delay and unreliability of multi-hop forwarding.

Temporary update bursts may congest region-hosts or object-holders. PPAct provides a differentiated subscription service to adjust bandwidth consumption when needed. Differentiation is applied to the frequency of receiving updates and the number of objects to be subscribed. Specifically, when a region-host finds that updates cannot be delivered to all subscribers on time, it halves the update delivery by dividing the subscribers into two groups and sends updates to each group every two frames. Such tuning is only applied as a temporary solution and the normal subscription is resumed immediately after bursts.

3.4. Mapping regions to region hosts

We select peers (i.e., players) to be candidate region hosts by using the scheme presented in Section 4.2. Such candidate region hosts form a DHT, called candidate host DHT. Each candidate host in the DHT has an ID, called candidate ID. Each region also has an ID, called region ID. We use its x and y coordinates as its region ID to preserve the locality of neighboring regions in the 2D-DHT.

When a region (x, y) needs a host, a candidate host is selected from the candidate host DHT by hashing the region ID (x, y) into a candidate ID. The candidate host then becomes the host of the region (x, y) and uses the region ID (x, y) to join the 2D-DHT. If this newly selected host still has spare bandwidth satisfying the threshold requirement defined in Section 4.2, it stays in the candidate host DHT. Otherwise, it leaves the candidate host DHT.

A failed region host is replaced by a candidate host instead of using the DHT failure handling in 2D-DHT to prevent a host from maintaining multiple neighboring regions. Hot regions are always clustered together and can easily overload a host if they share the same host. With new hosts selected from the candidate host DHT through hashing, it is less possible for neighboring regions to share the same host.

The maintenance overhead of the candidate host DHT is low because only stable nodes join it, and no extra communication or computation is needed.

3.5. Mapping objects to object holders

Each object has an object ID, and each object holder has a holder ID. All object holders form an object holder DHT. Each object has a primary copy, which is maintained by its object holder. Avatars and NPCs are treated differently. The object holder of an avatar is its player since it is mostly accessed by its player. The object holder of an NPC is selected from the object holder DHT by hashing the object ID. When an avatar's holder (i.e., its player) fails, we do nothing because the avatar is gone. When an NPC's holder fails, a new holder is selected according to the DHT failure handling in the object holder DHT.

Maintenance overhead of the object holder DHT is low because only stable peers join and no extra communication or computation is needed. Using the object holder DHT to

store object information takes advantage of DHT data replication and failure handling to ensure availability of NPCs.

We use an uptime threshold to select reliable peers as object holders, which form the object holder DHT. We use the update forwarding scheme (as described in Section 3.3) to prevent a holder from being overloaded by sending updates about the objects it holds.

3.6. Summary of region hosts and object holders

Table 1 summarizes the features of region hosts and object holders.

Fig. 3 shows the procedure for selecting players to be region hosts or object holders. We first assign each player as the object holder of its avatar. Then, we examine each player using the region host selection scheme presented in Section 4. If it has passed the selection threshold, it becomes a region host. Whether it is a region-host or not, a player becomes an object holder of NPC when it is identified as stable and has spare bandwidth. We use an uptime threshold to identify a stable player. If the uptime of a player is above the threshold, the player is identified to be stable; otherwise, it is unstable. The threshold for object holders is set to rule out the short-live unstable players. If a player leaves the DHT only a short time after joining, failure recovery overhead will outweigh its service. In simulation, we set the threshold for object holders to 10 min based on the churn studies in [14,15].

4. Region-host organization and selection

In this section, we introduce how to organize regions into a 2-D DHT and how to route lookup queries. We then analytically model the region host selection with both reliability and capability threshold requirements.

4.1. Region host DHT

Regions are organized into a multi-dimensional DHT that we use to process real-time lookup queries, where the number of dimensions in the DHT equals to the number of dimensions in the query attribute. As PPAct works on two-dimensional geographic queries, we organize regions into a two-dimensional DHT (2D-DHT) with two groups of DHTs, G_x and G_y. G_x is the group of DHTs built on x-axis, and G_y is the group of DHTs built on y-axis. Each DHT in $G_x(/G_y)$ is built on a set of regions that share the same $y(/x)$ coordinates. A region (x_0, y_0) is included in two DHTs, one $DHT_x \in G_x$ with $y = y_0$ and the other $DHT_y \in G_y$ with $x = x_0$. Therefore, each region-host has two routing tables: one is built for x-axis (DHT_x) and the other is for y-axis (DHT_y).

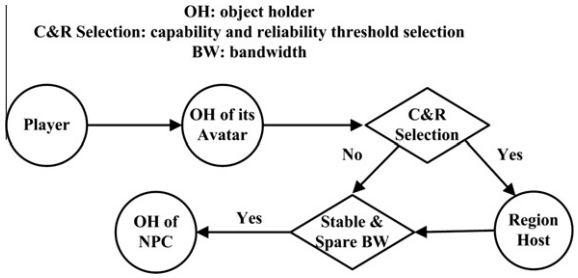


Fig. 3. The procedure of selecting a region host and an object holder.

Our 2D-DHT design provides fast lookups for players. This is because a region host has leaf sets of DHT_x and DHT_y including all its neighboring regions along either axis. In addition, the movements of players and objects in a game are mostly continuous. Thus, relying on the last-time contacted region host, a lookup takes 2-hops for moving vertically or horizontally and 3-hops for moving diagonally. The worst-case logarithmic-hop lookup is only performed for rare non-continuous movement.

Graphically, DHT_x is built on a row of regions with the same y coordinate as the owner's region and DHT_y is built on a column of regions similarly. As shown in Fig. 4, routing from a source region A to a destination region B follows a horizontal route in DHT_x of A to a region C with $y_C = y_A$ and $x_C = x_B$, then a vertical route in DHT_y of C to B.

The routing principles in 2D-DHT are the same as in the original DHT. We choose a representative DHT overlay – Pastry [16] as the routing substrate in PPAct. Pastry is widely used, for example, PAST [17] and SCRIBE [18] are built on top of Pastry. The routing in a 2D-DHT consists of two continuous routes, one from a $DHT_x \in G_x$ and the other from a $DHT_y \in G_y$. An example is shown in Fig. 5, routing from the source region at (350,479) to the destination region at (813,648). Given a total of $N * M$ regions, a $DHT_x \in G_x$ is built on N region and a $DHT_y \in G_y$ is built on M regions. In the worst case, a route to any region takes $\log(N) + \log(M) = \log(N * M)$ hops. The total number of hops is the same as original DHT routing, independent of the number of dimensions.

4.2. Analysis of region-host selections

A sufficient level of DHT performance is ensured by maintaining overlay connectivity. This requires that each DHT node should periodically probe other nodes, perform content replication, and execute failure recovery when necessary. Reliable DHT nodes reduce maintenance overhead. If a region host only serves a short time before leaving, failure recovery overhead outweighs its service contribution. In

Table 1
A summary of region hosts and object holders.

	Region hosts	Object holders
Major functions	Track objects and players in a region	Maintain object consistency and send updates
Workload balance	Threshold selection region assignment	Subscribers help forward updates
Failure handling	Replaced by a new host from the candidate host DHT	Follow the DHT failure handling

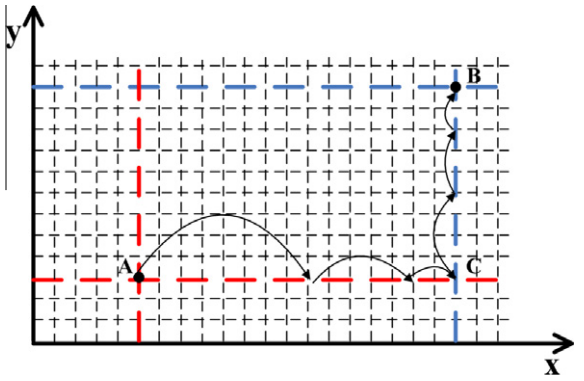


Fig. 4. Routing from region host A to region host B.

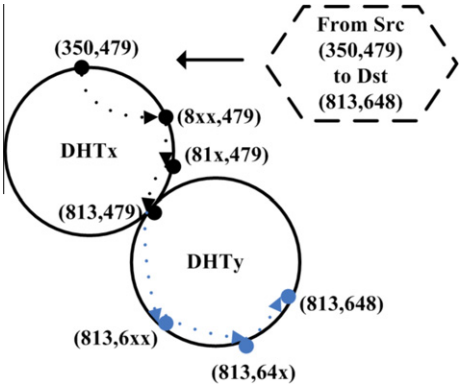


Fig. 5. An example of PPAct routing.

addition, a region host should have enough bandwidth to send subscription lists to object holders on time.

We propose a distributed scheme to select the most stable peers with adequate bandwidth to be region hosts. We focus on selecting peers as candidate region hosts, which form a candidate host DHT (called R-DHT). These candidate region hosts (i.e., R-DHT nodes) are mapped to regions through hashing and join the 2D-DHT as described in Section 3.4.

To estimate the number of qualified region hosts, we first analyze the distribution of peers' reliability and capability. Then, we set an appropriate threshold for selection.

We consider a system with n players in steady-state. Each node switches between two states: ON when the node gets online during its uptime and OFF when the node goes offline during its downtime. A node's uptime is the time interval between joining and leaving. Its downtime is the time interval from departure to re-joining. $U(\cdot)$ is the cumulative distribution function (CDF) of the node uptime, and $\bar{u}$ is the average uptime. $D(\cdot)$ is the CDF of the node downtime, and $\bar{d}$ is the average downtime. A node's availability a is measured by the probability that a node is in the ON state, $a = \frac{\bar{u}}{\bar{u}+\bar{d}}$. The expected number of nodes in the ON state is an.

We first examine nodes with expected uptime above T. The information about the node uptime can be obtained either by sampling or by prediction techniques [19,20]. Let n' be the number of nodes above the threshold T and in their uptime, $n' \leqslant n$. $\rho = \frac{n'}{an}$ represents the ratio of such nodes, $0 < \rho \leqslant 1$. $U'(\cdot)$, $D'(\cdot)$, $\bar{u}'$, $\bar{d}'$ and a' represent the counterpart attributes of the nodes with expected uptime above T as $U(\cdot)$, $D(\cdot)$, $\bar{u}$, $\bar{d}$ and a of all nodes, respectively. Our analysis assumes a node maintains $\log(n')$ overlay connections which is the same as with the general DHT.

Measurement results [21,22] demonstrate that node uptime is well modeled by a long-tailed distribution. We adopt a shifted Pareto distribution to depict the independence of node uptime as [23]. The probability density function (PDF) of node uptime $u(t)$ is shown in Eq. (1), where $\alpha > 1,\ \beta > 0$

$$u(t) = \frac{\alpha}{\beta}\left(1+\frac{t}{\beta}\right)^{-(\alpha+1)} \tag{1}$$

And the CDF of the node uptime $U(t)$ is shown in Eq. (2), where $\alpha > 1,\ \beta > 0$

$$U(t) = 1 - \left(1+\frac{t}{\beta}\right)^{-\alpha} \tag{2}$$

The smaller value of α means a stabler system with longer node uptime. The PDF of selected R-DHT node uptime $u'(t)$ is derived in Eq. (3)

$$u'(t) = \begin{cases} 0 & t < T \\ \frac{u(t)}{\int_{t=T}^{\infty} u(t)dt} = (1+\frac{T}{\beta})^{\alpha} u(t) & t \geqslant T \end{cases} \tag{3}$$

And the CDF of the selected R-DHT node uptime $U'(t)$ is derived in Eq. (4)

$$U'(t) = \begin{cases} 0 & t < T \\ \frac{U(t)-U(T)}{\int_{t=T}^{\infty} u(t)dt} = (1+\frac{T}{\beta})^{\alpha}(U(t)-U(T)) & t \geqslant T \end{cases} \tag{4}$$

The stability of the selected R-DHT nodes is measured by their average uptime $\bar{u}'$ derived in Eq. (5)

$$\bar{u}' = \int_{t=T}^{\infty} t \cdot u'(t)dt = \frac{\beta+\alpha T}{\alpha-1} \tag{5}$$

The higher the threshold T, the better the system stability $\bar{u}'$. This system stability determines the frequency of maintenance probing and thus the expected overhead.

To calculate the ratio of selected R-DHT nodes over all nodes, we compute the values of n' and ρ as a function of the selection threshold T. According to Little's Law, the average number of nodes in a stable system equals their average arrival rate multiplied by their average uptime in the system. Applying to all nodes in the ON state, we get $an = \lambda\bar{u}$, where λ is the average arrival rates of nodes to the ON state. Applying Little's Law to the selected R-DHT nodes in the ON State, we get $n' = \lambda(1-U(T))\bar{u}'$. Substituting the variables an, n', $U(T)$ and using $\bar{u} = \int_{t=0}^{\infty} t \cdot u(t)dt = \frac{\beta}{\alpha-1}$, the percentage of selected R-DHT nodes ρ is derived in Eq. (6), which estimates how many players are qualified for the threshold T

$$\rho = \frac{n'}{an} = \left(1+\frac{T}{\beta}\right)^{-\alpha}\left(1+\frac{\alpha}{\beta}T\right) \tag{6}$$

Next, we examine nodes with bandwidth capacity above a threshold B. The bandwidth consumption of a region-host comes from three aspects: R-DHT maintenance

workload B_1, bandwidth reserved as the object-holder of its avatar B_2, and bandwidth reserved as a region-host B_3. $B = B_1 + B_2 + B_3$. PPAct sets probe frequency to be once per average R-DHT node uptime, so that $B_1 = \frac{1}{u'}\log(n')p_m$, where p_m is maintenance packet size, and $\log(n')$ is the number of overlay connections a R-DHT node maintains. B_2 changes with the number of subscribers to the object, and B_3 changes with the number of objects in the region. According to the traffic analysis in Fig. 14, B_2 and B_3 are empirically set to be 500 Kb and 1 Mb to satisfy basic demand. Update forwarding and burst handling help deal with the dynamic workloads.

The ratio of nodes qualified for the bandwidth capacity threshold B is denoted as μ. We analyze μ with capacity threshold B in the same way as we analyze ρ with the reliability threshold T. The heterogeneous P2P node capacity is modeled by a bounded Pareto distribution with lower bound L and upper bound H, representing the diverse bandwidth from dial-up modem connections to cable connections. The PDF of the node capacity $f(x)$ is shown in Eq. (7), where a node capacity is between upper and lower bounds $L \leqslant x \leqslant H$, and the shape parameter is γ ($\gamma > 0$)

$$f(x) = \frac{\gamma L^{\gamma} x^{-\gamma-1}}{1-(\frac{L}{H})^{\gamma}} \tag{7}$$

The CDF of the node capacity $F(x)$ is shown in Eq. (8)

$$F(x) = \frac{1-L^{\gamma}x^{-\gamma}}{1-(\frac{L}{H})^{\gamma}} \tag{8}$$

Given there are total n nodes with bounded distribution from L to H, the number of nodes ranging from threshold B to H is $(F(H)-F(B))n$. Hence, the percentage μ of nodes selected with capacity threshold B is in Eq. (9)

$$\mu = \frac{(F(H)-F(B))n}{n} = \frac{(\frac{L}{B})^{\gamma}-(\frac{L}{H})^{\gamma}}{1-(\frac{L}{H})^{\gamma}} \tag{9}$$

Finally, the number of R-DHT nodes qualified for both thresholds is $n \cdot \rho \cdot \mu$. Given the total N regions, PPAct sets the reliability threshold T^* as in Eq. (10) to get the best reliability constrained by that the total region-host bandwidth reservation from all selected nodes is sufficient to serve all regions. The average bandwidth capacity of the selected nodes is given by $\bar{b}' = \int_{x=B}^{H} x \cdot f(x)dx$

$$T^* = arg\max T \quad s.t.\ n\rho\mu(\bar{b}' - B_1 - B_2) \geqslant N * B_3 \tag{10}$$

5. Performance evaluation

5.1. Experimental methodology

We developed a simulator to evaluate the efficiency of PPAct in FPS games and RPGs.

Game settings. Two game workload generators are developed: one for FPS games and the other for RPGs. Game traffic characteristics are based on trace data of Counter Strike [24] for FPS games and ShenZhou Online games [12] for RPGs, as summarized in Table 2. The actions of the players and their movements in FPS games are based on the networked game mobility model [8], which simulates the real FPS player behaviors. Those in RPGs are extracted from the trace ShenZhou Online games in [12], including both player-to-player and player-to-object interactions. The map simulated for FPS games takes an average peer $5 * 10^3$ s to walk from one end to the other, and that for RPGs takes an average of $5 * 10^4$ s. The map is partitioned into $10 * 10$ regions in FPS games and $100 * 100$ regions in RPGs by default. These default values were selected after careful evaluation of the impact of region size on performance, given in Section 5.2.

Network model. Each player is on an individual machine in a simulated network, representing a general Internet player experience. We adopt the widely used statistics of the player bandwidth capacity collected at US Broadband report [25]. The upload capacity of game players is shown in Fig. 6, which is well approximated by a Pareto distribution with a range from 256 Kb/s to 10 Mb/s. We simulate wide geographic areas where players come from. The inter-player round-trip time (RTT) in an n-player game is simulated by drawing n nodes from the Xbox 360 player data set [26] that is spread over the Western United States. The mean, median, and standard deviation of inter-player RTT of this data set are 81 ms, 64 ms, and 63 ms. Vivaldi 3D coordination system [27] is used to extrapolate the RTT values between pairs of players who did not probe each other in the data set. We use a two-state Gilbert model [28], which models packet loss property of Internet paths, setting loss rate to 1% and mean loss burst time to 100 ms.

Performance metric. We mainly use three metrics to measure performance of PPAct in maintaining consistency for real-time games. The *successful action rate* is the ratio of the number of actions completed over the total number of actions issued by players. An action refers to an update that a player issued on an object. An action is completed when the update is successfully received by the object holder on time. The *successful subscription rate* is the ratio of the number of subscriptions received by object holders over the total number of subscriptions requested by players. A subscription refers to completion of step 1 and step 2 in Fig. 1. A player requests a subscription to each object in its view. The *successful update rate* is the ratio of the number of updates received by subscribers over the total number of updates sent out by object holders. An update refers to the completion of step 3 in Fig. 1. We use 150-ms deadline for FPS games and 180-ms for RPGs according to [13]. Each result shown in the figures is the average of 50 rounds of simulations, and each round executes $5 * 10^3$ s.

Table 2
A summary of game traces.

Trace	Counter Strike	Shenzhou Online
Date	April 11, 2002	August 29, 2004
Start time	08:55	15:00
Period	7 days, 6 h, 1 m	20 h
Established connections	16,030	112,369
Total packets	500 M	1356 M
Mean payload size	32 bytes	32 bytes
Mean packet size	87 bytes	84 bytes

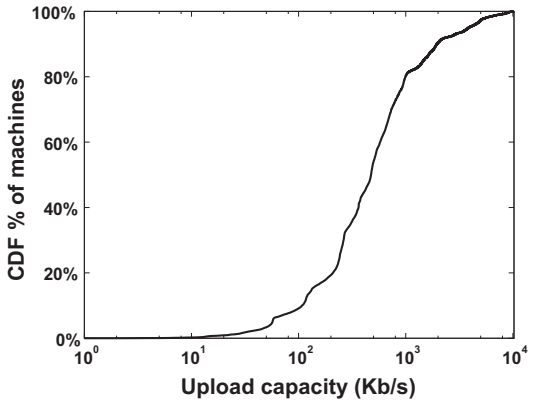


Fig. 6. CDF of peers' upload capacities.

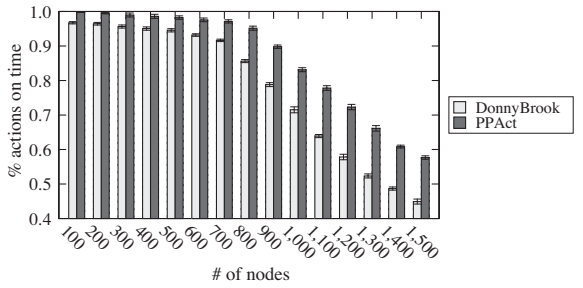


Fig. 7. Successful action rates in FPS games. Error bars show 95% confidence intervals.

5.2. Evaluation results

Scalability for FPS games. We evaluate PPAct for FPS games compared to Donnybrook [3], which is a seminal work on P2P managed FPS games. The successful rates of actions, updates and subscriptions are shown in Figs. 7–9. PPAct outperforms Donnybrook in all three metrics. An important reason is that Donnybrook requires every player to broadcast a guidance message to every other player each second. Getting rid of the broadcast overhead, PPAct saves more bandwidth for delivering updates, actions and subscriptions. Since the broadcast overhead grows exponentially when the number of players increases, PPAct has more advantages over Donnybrook when the system becomes larger.

Comparing results in Figs. 7–9, the successful subscription rate is the lowest of the three. This is because every time a subscription is sent to a different region host, while actions and updates are sent to the same object holders or subscribers until a subscription is changed. Thus, completing a subscription incurs an extra lookup delay over completing an action or an update. The subscription performance is still acceptable because of the constant hop lookup supported by both PPAct and Donnybrook. A Donnybrook subscription takes one hop because the broadcast lets every node know all others. Most PPAct subscriptions take 2 or 3 hops as shown in Fig. 10. Since players move

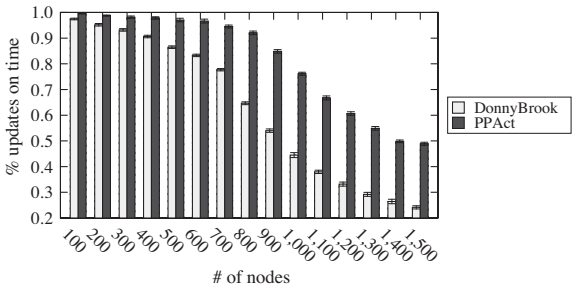


Fig. 8. Successful update rates in FPS games. Error bars show 95% confidence intervals.

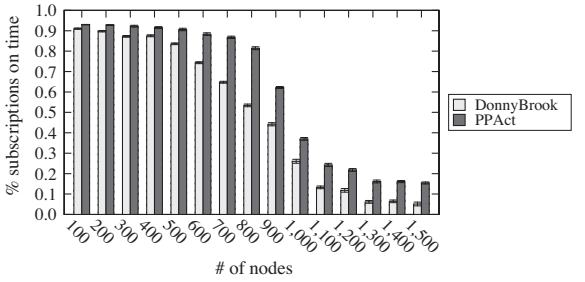


Fig. 9. Successful subscription rates in FPS games. Error bars show 95% confidence intervals.

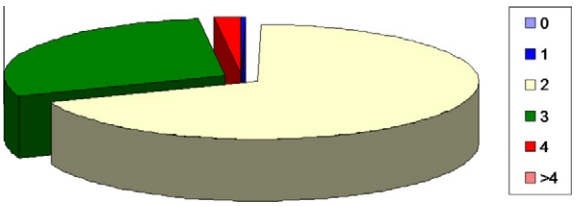


Fig. 10. Subscription hop counts in PPAct.

continuously most of the times, a new region is adjacent to the previously subscribed one. Such a subscription takes 2 hops in 2D-DHT. The 3 or 4 hops are caused by node churn, for another hop is taken to contact the new host. PPAct achieves higher successful subscription rates than Donnybrook under node churn because of overwhelming broadcast overhead in Donnybrook.

To maintain high success rates of actions and updates, we choose a larger subscription area than the player's view. As a result, when a player moves, most of the new view regions intersect with previously subscribed regions. This masks subscription delay.

Scalability for RPGs. Since Donnybrook does not support NPCs in RPGs, we compare with SimMud [2], a pioneering work on P2P managed RPGs. Both PPAct and SimMud use region partitioning techniques and DHT routing. By default, the number of NPCs is 10^4. Figs. 11–13 show that PPAct achieves significantly better results than SimMud by all three metrics. This is because every SimMud region host must be the object holder for all objects

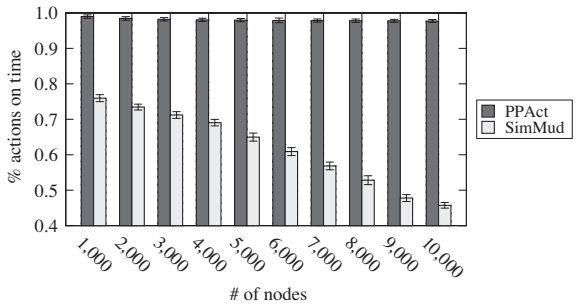


Fig. 11. Successful action rates in RPGs. Error bars show 95% confidence intervals.

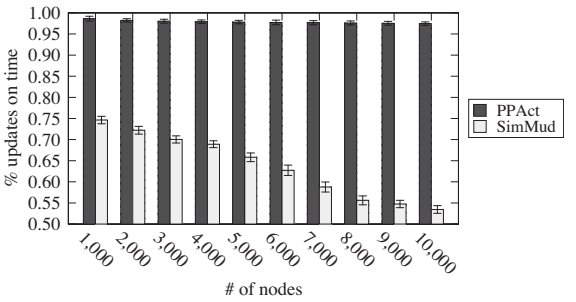


Fig. 12. Successful update rates in RPGs. Error bars show 95% confidence intervals.

in that region, which may be overloaded easily. PPAct separates the workload of region hosts from object holders to avoid overloading. In addition, PPAct selects reliable and capable players to be region hosts as modeled in Section 4.2, while SimMud randomly selects players to be region hosts. As a result, the 2D-DHT in PPAct is more robust and efficient than the DHT in SimMud. As shown in Fig. 13, SimMud incurs a longer subscription delay than PPAct. This is because SimMud organizes all players into one DHT, and a subscription takes logarithmic hops.

PPAct supports RPGs with both successful action rate and update rate above 98% up to 10^4 players. To our knowledge, it is the first one to support P2P managed online games with tens of thousands players.

Impacts of population density. Comparing the results in Figs. 8 and 12, the scalability of PPAct is ten times more in RPGs than in FPS games. This is because the density of players in a region is one tenth in RPGs than in FPS games, but the region size is the same in both. The map in RPGs is 100 times larger than that in FPS games and the number of regions in RPGs is also 100 times more than that of FPS games. When the player population grows 10 times with game map increasing 100 times, the population density becomes 1/10. The reduced population density results in reduced update delivery overhead, which dominates overall traffic in PPAct as shown in Fig. 14. Increased scalability in PPAct is also a result of our dynamic workload balance scheme. When serving the same 1000 players, the performance of SimMud in RPGs as shown in Fig. 12 is still lower than that of PPAct in FPS games as shown in Fig. 8, because SimMud does not handle the AOI "hot spot" problem. Thus, addressing clustered workload of hot regions is critical to scalability.

Impacts of region size. Generally, there are trade-offs in using large regions or small regions. Large regions speed up lookup and reduce query processing overhead, as fewer regions cover the same view and fewer region hosts are queried. However, large regions reduce granularity and accuracy of range query processing. Oversized regions either waste resources for processing extra areas or provide insufficient views. Oversized regions also increase workload imbalance, since hot areas are covered by a fewer number of regions. To the contrary, small regions lower the workload of each region for better workload balance. A host may choose to take charge of several small regions

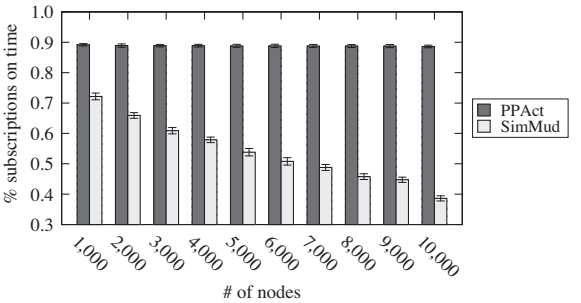


Fig. 13. Successful subscription rates in RPGs. Error bars show 95% confidence intervals.

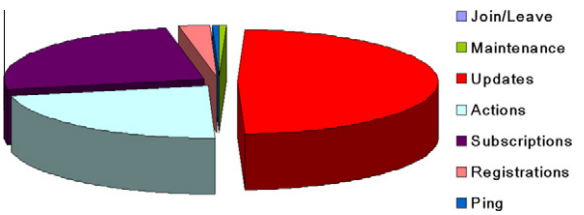


Fig. 14. PPAct traffic analysis.

that are unlikely to have simultaneous workload crowds. Whereas, small regions incur higher lookup overhead and object hand-off overhead (i.e., the overhead incurred by switching registration among regions when an object moves) because objects move across regions more frequently.

We evaluate the impact of region size by partitioning the RPG map into $40*40$, $50*50$, and so on up to $300*300$ regions. The results are shown in Figs. 15–17. Subscription rate in Fig. 17 improves when the number of regions increases from $40*40$ to $100*100$, and degrades with further increases. The improvement comes from the decreased population density with smaller region size, while further reducing region size imposes excessive subscription overhead when players move. However, the degradation is only reflected in successful subscription rate. The successful action rate and update rate are

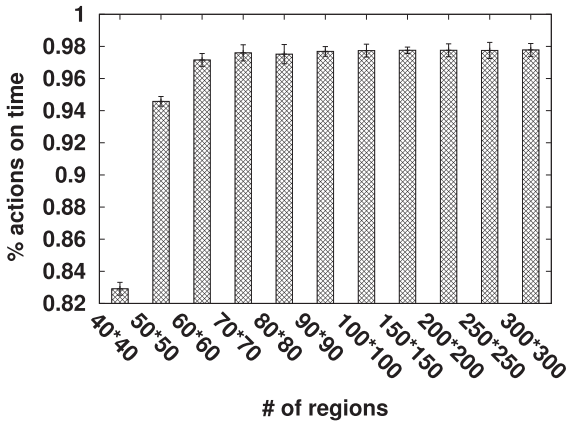


Fig. 15. Successful action rates in RPGs with various scales of regions. Error bars show 95% confidence intervals.

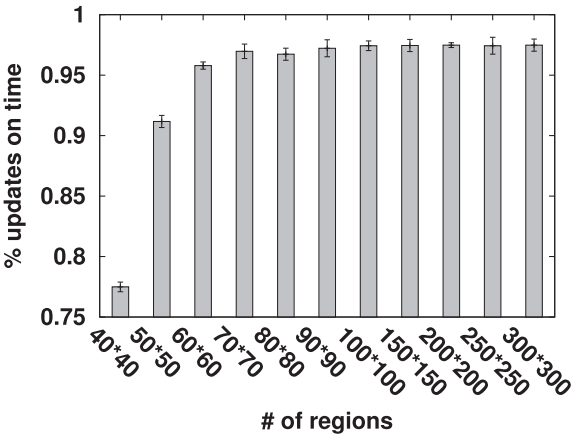


Fig. 16. Successful update rates in RPGs with various scales of regions. Error bars show 95% confidence intervals.

maintained high as in Figs. 15 and 16. Since subscription overhead in PPAct is only a minor part of overall traffic as shown in Fig. 14, we choose the size of $100 * 100$ regions as the default region size in our simulations.

Impacts of object number. We evaluate the impact of number of mutable objects by increasing the number of NPCs from 10^4 to 10^5 in RPGs with 10^4 players. The results in Figs. 18–20 show that the performance of PPAct degrades only slightly while increasing the number of NPCs. This is because update delivery overhead is mainly affected by the number of receivers for each update not the number of mutable objects. The number of receivers per each update is reflected by the population density. Therefore, even with an increased number of NPCs the overall traffic is in the same order when the population density is kept the same.

Impacts of churn. According to churn studies in different P2P applications [14], we model the inter arrival time of players by a Weibull distribution with shape parameter $k = 0.6$. We vary the scale parameter to simulate the average leave rate ranging from 10% to 50%. Results under different churn rates do not have presentable differences, so we do not show them separately. All our results are measured under an average leave rate of 25%. PPAct performs robustly against node churn because we select reliable players as region hosts. We also observe that short lives of other players do not have noticeable negative effect on overall performance.

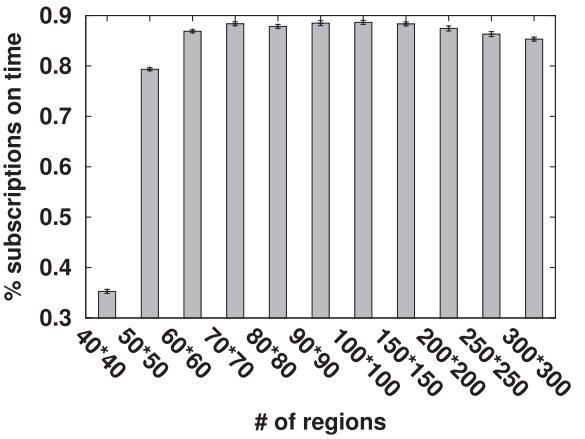


Fig. 17. Successful subscription rates in RPGs with various scales of regions. Error bars show 95% confidence intervals.

6. Related work

6.1. P2P managed online game systems

In the literature, two major techniques are proposed to reduce bandwidth consumption of update delivery. One is Area-of-Interest (AOI) filtering [1,2], where updates about an object is only sent to the players within the same region as the object. This method works well when players have limited proximity to each other, but does not help when players are clustered in a few regions, which is unavoidable due to the power law distribution of the player

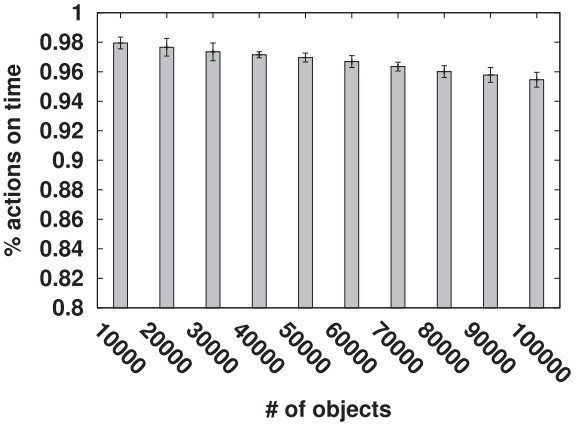


Fig. 18. Successful action rates in RPGs with various scales of objects. Error bars show 95% confidence intervals.

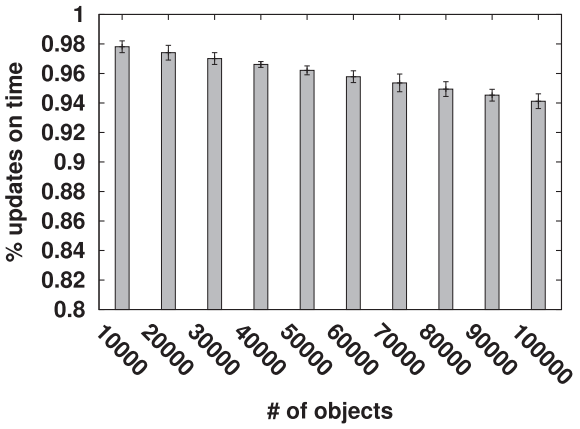


Fig. 19. Successful update rates in RPGs with various scales of objects. Error bars show 95% confidence intervals.

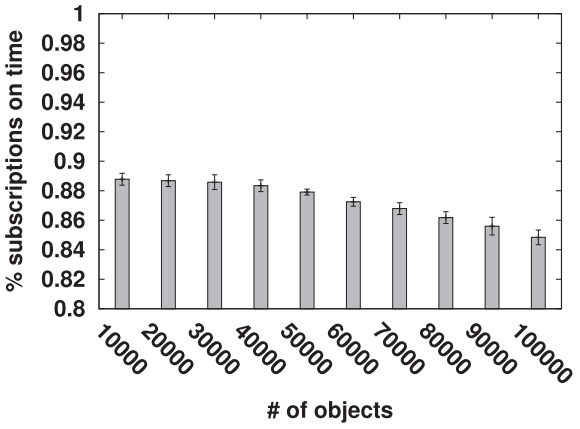


Fig. 20. Successful subscription rates in RPGs with various scales of objects. Error bars show 95% confidence intervals.

population density [9]. As a result, demand in such *hot regions* still grows quadratically, and AOI filtering does not fundamentally solve the bandwidth shortage problem. Another technique is proposed in Donnybrook [3], which limits each player to receive only a constant number of real-time updates. The bandwidth consumption of sending real-time updates is then linearly proportional to the number of players. Since every player still has to be aware of the status of all the others, each Donnybrook player is required to broadcast less frequent guidance messages. Therefore, total bandwidth consumption in Donnybrook remains asymptotically quadratic to the total number of players.

We follow the AOI filtering but solves its "hot spot" problem by having players contribute their spare bandwidth to update delivery in a distributed way. We also take advantage of players' movement patterns to build a 2-D DHT for reducing the lookup overhead and delay. Thus, more bandwidth is saved for sending out updates on time.

For better workload distribution, we decouple view discovery and update delivery. The Colyseus system [1] also decoupled object discovery and replica synchronization, but they assumed objects were properly placed with minimal interactive latency without any specific methods. We present efficient object placement schemes, peer selection schemes, and failure recovery processes.

6.2. Load balance on P2P networks

Data replication and load redistribution are two major solutions to the imbalance workload problem in P2P networks. Workload imbalance is due to skewed data popularity and heterogeneous peer capability. Data replication alleviates overloaded nodes by providing extra targets for incoming requests [29–32]. Data migration requires actual data transfers between nodes to balance workload, which has two major forms: data item exchange and node migration. Optimal load balance requires both item exchanges between neighbor nodes and global node migrations [33]. Some P2P systems use topology adjustment to balance workload. A node in the Mercury system [11] uses sampling to estimate other nodes' workloads and adjusts its long link connections for load balance. The authors in [34] propose that each node tunes its routing table size to balance query workload and avoid congestion. These methods focus on static traffic, while PPAct addresses load balance for dynamic traffic and avoids data transfers or node migrations.

To balance workload for update delivery, Donnybrook [3] also has each player advertise its spare bandwidth to forward updates. However, in Donnybrook [3] each player must broadcast its spare bandwidth, which requires global knowledge of all players and strictly limits system scalability.

6.3. P2P systems support for range queries

Distributed Hash Table (DHT) provides a scalable and robust substrate to build structured P2P systems. However, DHT only supports exact-match queries because the uniform hashing of DHT destroys data locality required by the range query processing. Existing P2P index techniques to support range queries fall into two categories: (1) DHT-preserved indexing, which maintains the original DHT and builds an overlay index-structure on top of the DHT (e.g., Prefix Hash Trie (PHT) [35], multi-dimensional Lightweight Hash Tree (mLIGHT) [36], Distributed Segment Tree (DST) [37] and Range Search Tree (RST) [38]); (2) DHT-modified indexing, which modifies the internal structure of DHT and develops certain locality-preserved overlay (e.g., the multi-ring structure in Mercury [11], the space kd-tree index in SkipIndex [39]). However, all these methods are inadequate to support real-time range queries, where the answer is time related and the object attribute is changing dynamically. Every change of an object's attribute value leads to changes on PHT or object movements from one node to another. This results in cascading changes and requires re-running load balance algorithms. PPAct successfully enhances the DHT structure to support real-time

range queries by properly partitioning regions and making a region as the unit of range index.

7. Conclusion

In this paper, we proposed a P2P system called PPAct to provide consistency maintenance for large-scale fast-interactive applications. We presented the design and evaluation of PPAct by taking massive multi-player online games as example applications. Our work focused on three major problems of consistency maintenance: insufficient uplink bandwidth for the update delivery, workload imbalance among players, and incompatibility for various types of games.

We adopted the AOI filtering method to reduce bandwidth consumption of update delivery and made players contribute their spare bandwidth in a distributed way to solve the AOI "hot spot" problem. We separated the roles of view discovery and consistency maintenance by assigning peers as region hosts and object holders. This speeds up the players lookup and reduces the delay of update delivery to meet the stringent latency requirement of FPS games. Meanwhile, this separation also flexibly handles the object consistency maintenance for RPGs.

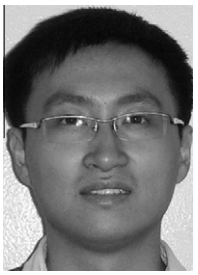
We conducted extensive simulations to evaluate the performance of PPAct on two types of popular online games: FPS games and RPGs. The performance of PPAct is compared with Donnybrook [3] on FPS games. Results show that PPAct scales well to support 1500 players with a success update rate 30% higher than that of Donnybrook. The performance of PPAct is compared with SimMud [2] on PRGs. Results show that PPAct scales well to support 10,000 players with a success update rate 40% higher than that of SimMud.

References

[1] A. Bharambe, J. Pang, S. Seshan, Colyseus: a distributed architecture for online multiplayer games, in: NSDI, 2006.
[2] B. Knutsson, W.X.H. Lu, B. Hopkins, Peer-to-peer support for massively multiplayer games, in: IEEE INFOCOM, 2004.
[3] A. Bharambe, J.R. Douceur, J.R. Lorch, T. Moscibroda, J. Pang, S. Seshan, X. Zhuang, Donnybrook: enabling large-scale, high-speed, peer-to-peer games, in: ACM SIGCOMM, 2008.
[4] S. Agarwal, J.R. Lorch, Matchmaking for online games and other latency-sensitive P2P systems, in: ACM SIGCOMM, 2009.
[5] G. Oster, P. Urso, P. Molli, A. Imine, Data consistency for P2P collaborative editing, in: CSCW, 2006.
[6] D. Li, R. Li, Ensuring content and intention consistency in real-time group editors, in: IEEE ICDCS, 2004.
[7] XboxLIVE, 2008. <http://www.xbox.com/en-US/live>.
[8] S.A. Tan, W. Lau, A. Loh, Networked game mobility model for fist-person-shooter games, in: NetGames, 2005.
[9] D. Pittman, C.G. Dickey, A measurement study of virtual populations in massively multiplayer online games, in: NetGames, 2007.
[10] M. Varvello, C. Diot, E. Biersack, P2P second life: experimental validation using Kad, in: IEEE INFOCOM, 2009.
[11] A. Bharambe, M. Agrawal, S. Seshan, Mercury: supporting scalable multi-attribute range queries, in: ACM SIGCOMM, 2004.
[12] K.T. Chen, P. Huang, C.L. Lei, Game traffic analysis: an mmorpg perspective, Computer Networks 51 (3) (2006).
[13] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, M. Claypool, The effects of loss and latency on user performance in unreal tournament 2003, in: NetGames, 2004.
[14] D. Stutzbach, R. Rejaie, Understanding churn in peer-to-peer networks, in: ACM IMC, 2006.
[15] J. Falkner, M. Piatek, J.P. John, A. Krishnamurthy, T. Anderson, Profiling a million user DHT, in: ACM IMC, 2007.
[16] A. Rowstron, P. Druschel, Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, in: International Conference on Distributed Systems Platforms, 2001.
[17] A. Rowstron, P. Druschel, Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility, in: ACM SOSP, 2001.
[18] M. Castro, P. Druschel, A. Kermarrec, A. Rowstron, Scribe: a large scale and decentralized application level multicast infrastructure, IEEE J-SAC 20 (8) (2002) 1489–1499.
[19] J. Mickens, B. Noble, Exploiting availability prediction in distributed systems, in: NSDI, 2006.
[20] J. Mickens, B. Noble, Predicting node availability in peer-to-peer networks, in: SIGMETRICS POSTER, 2005.
[21] S. Saroiu, P. Gummadi, S. Gribble, A measurement study of peer-to-peer file sharing systems, in: ACM MMCN, 2002.
[22] D. Leonard, V. Rai, D. Loguinov, On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks, in: IEEE SIGMETRICS, 2005.
[23] C.C. Wang, K. Harfoush, On the stability-scalability tradeoff of DHT deployment, in: IEEE INFOCOM, 2007.
[24] W. Chang Feng, F. Chang, W. Chi Feng, J. Walpole, A traffic characterization of popular on-line games, IEEE/ACM TON 13 (3) (2005).
[25] Broadband report, 2008. <http://www.dslreports.com>.
[26] Y. Lee, S. Agarwal, C. Butcher, J. Padhye, Measurement and estimation of network qos among peer xbox 360 game players, in: PAM, 2008.
[27] F. Dabek, R. Cox, F. Kaashoek, R. Morris, Vivaldi: A decentralized network cooridante system, in: ACM SIGCOMM, 2004.
[28] E.N. Gilbert, Capacity of a burst-noise channel, The Bell System Technical Journal 39.
[29] C. Zhang, Z. Zhang, Trading replication consistency for performance and availability: an adaptive approach, in: IEEE ICDCS, 2003.
[30] S. Susarla, J. Carter, Flexible consistency for wide area peer replication, in: IEEE ICDCS, 2005.
[31] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, P. Keleher, Adaptive replication in peer-to-peer systems, in: IEEE ICDCS, 2004.
[32] S. Tewari, L. Kleinrock, Proportional replication in peer-to-peer networks, in: IEEE INFOCOM, 2006.
[33] P. Ganesan, M. Bawa, H.G. Molina, Online balancing of range-partition data with applications to peer-to-peer systems, in: VLDB, 2004.
[34] H. Shen, C. Xu, Elastic routing table with provable performance for congestion control in DHT networks, in: IEEE ICDCS, 2006.
[35] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, J. Hellerstein, A case study in buliding layered DHT applications, in: ACM SIGCOMM, 2005.
[36] Y. Tang, J. Xu, S. Zhou, W.C. Lee, m-light: indexing multi-dimensional data over DHTs, in: IEEE ICDCS, 2009.
[37] C. Zheng, G. Shen, S. Li,, S. Shenker, Distributed segment tree: support of range query and cover query over DHT, in: Fifth International Workshop on Peer-to-Peer Systems IPTPS, 2006.
[38] J. Gao, P. Steenkiste, An adaptive protocol for efficient support of range queries in DHT-based systems, in: IEEE ICNP, 2004.
[39] C. Zhang, A. Krishnamurthy, R.Y. Wang, Brushwood: distributed trees in peer-to-peer systems, in: IEEE IPTPS, 2005.

Yi Hu received her B.Eng. degree in Computer Science from City University of Hong Kong in 2006. Currently, she is a PhD student, advised by Dr. Laxmi N. Bhuyan, in Computer Science and Engineering Department at University of California, Riverside.

Laxmi N. Bhuyan is Distinguished Professor and Chairman of Computer Science and Engineering Department at the University of California, Riverside (UCR). Prior to joining UCR in January 2001, he was a professor of Computer Science at Texas A&M University (1989–2000) and Program Director of the Computer System Architecture Program at the National Science Foundation (1998–2000). He has also worked as a consultant to Intel and HP Labs.



Min Feng received his B.Eng. degree in Computer Science and Technology from Tongji University, Shanghai in 2005, and his M.Phil. degree in Computer Science from City University of Hong Kong in 2007. Currently, he is a PhD student, advised by Dr. Rajiv Gupta, in Computer Science and Engineering Department at University of California, Riverside.