ISR Institute for Software Research
University of California, Irvine

RCAT: A Scalable Architecture for Massively Multiuser Online Environments

Thomas Debeauvais
University of California, Irvine
tdebeauv@uci.edu

Arthur Valadares
University of California, Irvine
avaladar@ics.uci.edu

Cristina V. Lopes
University of California, Irvine
lopes@ics.uci.edu

November 2013
ISR Technical Report # UCI-ISR-13-2

Institute for Software Research
ICS2 221
University of California, Irvine
Irvine, CA 92697-3455
www.isr.uci.edu

www.isr.uci.edu/tech-reports.html

RCAT: A Scalable Architecture for Massively Multiuser Online Environments

Thomas Debeauvais, Arthur Valadares, Cristina V. Lopes
Institute for Software Research
and Department of Informatics,
UC Irvine
{tdebeauv, avaladar, lopes}@uci.edu

Abstract

We describe RCAT, an architecture designed to scale real-time online environments up to thousands of interacting users in an application-independent manner. RCAT takes inspiration from the principles that allow Web applications to scale horizontally, while being aware of the challenges brought up by real-time multi-user interactions. We present a performace study that shows where the bottlenecks of RCAT are, and the thresholds at which additional cores are needed. Our laboratory experiments with a reference application that produces 2.5 updates/second show that 229 interacting users can be appropriately served with 16 cores. More generally, our experiments show that to support twice the number of users, the server side needs four times more cores, independent of the update frequency of the application. This finding is important in order to engineer the next generation of cloud-based massive multiuser environments.

1 Introduction

Massively Multiuser Online games (MMOs) such as World of Warcraft, EVE Online, and Second Life are virtual 3D environments in which players interact in real-time with each other inside a virtual world. But interactions in current MMOs are not truly massive: designers craft the games so that users can only interact with at most around a hundred other users at the same time. When more than the expected number of players gather together, the game usually crashes[1].

The demand for high user concurrency is now going well beyond games. Applications such as Facebook and Twitter distribute massive numbers of events among massive numbers of users. While these applications are not exactly

[1] See a virtual flash mob in World of Warcraft at http://www.youtube.com/watch?v=m7FW0BK2fUo

UCI ISR Technical Report # UCI-ISR-13-2. Nov. 2013.

real-time, the updates are relatively fast-paced, and therefore present similar challenges to those seen in online games. As Web technologies become more capable of supporting rich 2D and 3D media, the line between online games and (serious) Web applications will become fuzzier. The goal of our work is to be able to support the next generation of cloud-based massively multiuser online environments, such as Massive Online Open Courses (MOOCs) [5] and social environments for medical applications [42].

Looking at how MMOs have approached scalability can provide insights to scale other types of multiuser applications. MMO players are generally seen as event producers and consumers: each player generates a stream of events that other players are interested in. For example, if a player orders her character to move forward, other players should be notified or the character's movement, update their local state of the world, and render the updated state. If all the players subscribe to all the other players, then the number of event messages to deliver increases quadratically with the number of players [37]. Quickly, the system is overwhelmed by the number of events, and the latency (i.e. the time taken to forward an event) increases. The game becomes much less responsive. Thus, there is a trade-off between scalability and responsiveness. That is, if the number of users increases, so does the latency.

To give the illusion that players can interact with thousands of other players, MMOs use interest management techniques [9]. They assume that players are only interested in events happening near them in the virtual world. The world is therefore partitioned in self-contained regions (e.g. a city or a forest), and each region is handled by a different process. Systems implementing space-partitioning assume that players are interested in events generated in their current region, and maybe also in the adjacent regions, but never in the entire world. Using this partitioning strategy, MMOs have been able to support hundreds of users while keeping the latency relatively low.

Space-partitioning works well until too many players decide to meet in the same region. For example, the maximum number of directly interacting users ever achieved in a commercial MMO is around 3,000 in the game EVE Online[2]. This record was achieved thanks to a very expensive hardware infrastructure, a tiered software architecture, and a game design solution called time dilation. Time dilation compensates for the server load by slowing down the region's time. Regions may support more players, but some have found time dilation "absolutely unplayable"[3]. MMOs that can not afford this kind of infrastructure, architecture, or game design tricks have much lower player limits: World of Warcraft supports around 120 users per region[4], and Second Life up to 100[5].

Research has tried to dynamically adapt the shape and size of regions to the distribution of players. However, solutions such as dynamic binary space partitioning suffer from a high overhead due to synchronization and data handover between regions [36]. In fact, the CAP theorem states that a partitioned

[2] See `http://themittani.com/news/asakai-aftermath-all-over-cobalt-moon`

[3] See `https://forums.eveonline.com/default.aspx?g=posts&t=108331`

[4] See `http://www.wowwiki.com/Wintergrasp#Queuing`

[5] See `http://wiki.secondlife.com/wiki/Limits#Land`

system can not be both highly available (i.e. responsive) and strongly consistent [11, 12]. Since a scalable MMO system involves multiple partitioned processes, the MMO developer inherently has to trade some consistency to stay responsive.

In the last two decades, the number of MMO players has tremendously increased, but so as the number of Web users. To scale, Web applications follow a very constrained architectural style called REpresentational State Transfer (REST) [22]. One of the pillars supporting REST is the statelessness of the HTTP protocol: web servers do not remember any client data between two requests. This way, it is easy to scale a web application by adding more machines running the exact same application code.

In this work, we explore how REST could be applied to scale MMOs. Even though some game developers doubt that REST can be applied to MMOs [51], the suitability of REST for MMOs remains an open question. Moreover, recent web technologies such as HTML5 WebSockets or WebGL show serious opportunities for browser-based MMOs.

Although scalability is our main concern, we keep in mind the trade-off with consistency and responsiveness. Other requirements such as fault-tolerance (if a machine falls, it is at minimal cost for the system) or resilience (recovering quickly from peaks or crashes) are desirable, but they are not the main focus of this paper.

The contributions of this work are as follows:

- We make a first attempt at conceptually harmonizing REST principles with MMOs
- We develop an architecture (RCAT) and its reference implementation, that is based on those principles
- We report performance characteristics of a prototypical RCAT application that allow us to estimate upfront the resources necessary to deploy these applications on the cloud given specific concurrency targets

In the rest of this paper, we first cover the current techniques used to scale MMOs to large numbers of users. Then we introduce RCAT, our own architecture designed to scale to large number of users without being bound to a particular type of application. We then detail the reference implementation and an application based on this architecture. Finally, we report our laboratory experiments, discuss the implications, and conclude.

2 Overview of Techniques for Scaling Up Multiuser Games

A large body of academic literature, and several practical techniques seen in the industry, focus on scalability, consistency, and responsiveness for MMOs. These techniques are scattered in various fields of computer science, and target different parts of the scalability problem. This section revisits some of the most well-known techniques for scaling up these systems.

Probably the most popular current technique for scaling MMOs is space partitioning. We detail what space partitioning is in the next subsection. We also describe other game-specific techniques that help scaling while maintaining consistency and responsiveness. Then we look at scaling techniques applicable to MMOs from the database community, and finish with approaches from the network and systems communities.

2.1 Space Partitioning

Space partitioning consists of splitting the game world in multiple regions, and assigning each region to a process of the MMO system. In client-server architectures, the processes in charge of regions are game servers, while in peer-to-peer architectures, each peer is in charge of one region. Even though the practical feasibility of peer-to-peer architectures is subject to debate (e.g. Miller and Crowcroft say not feasible [41], Hu et al. say feasible [29]), client-server and peer-to-peer really are two sides of the same coin: they both partition a game state over multiple machines.

The assumption behind space-partitioning dates back from the seminal virtual world of the early 90s called DIVE [15]. In DIVE, users can transfer from region to region, but they only need to know what happens in their current region. DIVE follows a client-server architecture. Each server is in charge of processing the client requests concerning the virtual objects in its region, as well as the handover of users from and to other regions. This model works well until a server has to handle too many users or objects in its region. There has been a lot of academic work on space-partitioning [30, 53] Most current commercial MMOs use this static space-partitioning model. To prevent server crashes, game operators have resolved to *instancing*: they instantiate replicas of a particularly popular region, and cap the number of players in the region. The World of Warcraft dungeon raids are a current example of instancing.

But instancing replicas of a region does not make that region scale. Moreover, in virtual worlds like Second Life, where users can create objects at run time, the distribution of objects and users is non-uniform, and the behavior of objects and users hard to predict [31]. Dynamic space-partitioning aims at solving this problem by adapting the size and shape of regions to the object and user distribution, so as to balance the computational and bandwidth load across peers (in peer-to-peer) or servers (in client-server). There are several approaches to dynamic space partitioning. One approach treats regions as cells of a Voronoi diagram, whether in peer-to-peer [13] or in client-server [4]. Another approach treats regions as a set of adjacent microcells [18, 20, 49]. Yet recent work suggests that the amount of interactions between two regions can result in a high load for the two regions, thereby making spatial partitioning not suitable for all object or player distributions and behaviors [16]. In short, space-partitioning is an optimization that is only efficient in certain MMO use cases, but not all.

2.2 Other Game-Specific Techniques

Interest management consists of notifying users and objects only about the users, objects, locations, or events they are interested in. For example, an avatar could be interested in only avatars and objects within 100 meters, or in chat messages directly addressed to that avatar. In academia, interest management has often involved publish-subscribe infrastructures [9], whether through spatial queries in peer-to-peer [7], communication channels maintained by a centralized server [21], or a cloud-based infrastructure [43]. Commercially, interest management is central in EVE Online, a futuristic MMO. EVE's servers conceptually group nearby spaceships in self-contained and isolated spheres. Each sphere has a particular channel, to which all players in the sphere subscribe to. This is an improvement over the traditional space-partitioning, as region servers now only need to compute spheres and broadcast messages within spheres rather than to the entire region [10]. Sirikata [28] uses a twist on interest management: rather than computing it spatially, they compute it based on the avatar's view of the world. Thus clients receive more updates concerning objects in their frustum than updates concerning hidden objects. Once again, not all MMOs may be able to use interest management, visibility, or publish-subscribe techniques efficiently due to their design.

Data prioritization is a mechanism frequently used in MMOs to reduce the bandwidth between peers or from the server(s) to the clients. It assumes that some messages are critical, while others are not. When bandwidth or computational resources become scarce, only critical messages need to be forwarded. This has been abundantly studied in peer-to-peer academic research [6, 45].

Time dilation is a technique commonly used in commercial MMOs. It consists of slowing down the simulation time, mostly because the server's physics engine can not keep up with the load. While time dilation is more graceful than a complete server crash, it remains game-specific and severely reduces the responsiveness of the game. EVE Online and Second Life are two MMOs using time dilation[6].

And finally, some online games like first-person shooters place a strong emphasis on latency. For those games, it is better to be wrong but on time, than right but late [39]. Commercially, the Source Engine from Valve follows an *optimistic* client-server architecture[7]: if player A hits player B, A actually only predicts that B is hit. Client A sends the hit message to the server. If the server determines that player B was indeed hit when A fired, i.e. several frames ago, it forwards the hit to B. But B could have reacted faster than the latency between A and the server. In that case, A's view would be slightly inconsistent with B's and the server's. Thus the server would determine that A actually missed his shot. The server notifies A of the missed shot, and A rollbacks its state. This rollback results in local inconsistencies [17]. In academic research, Gupta et al. suggest that each client should run the game logic, and the server only act as

[6] See `http://community.eveonline.com/news/dev-blogs/3412` and `http://wiki.secondlife.com/wiki/LlGetRegionTimeDilation`

[7] `https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking`

a message forwarder and be in charge of persistence [26]. Clients apply their own actions to an optimistic model, and actions received from the server to a stable model. If applying the same action to the optimistic and stable models result in two different models, then the system needs to rollback. The client asks the server to broadcast a fix it proposes, all other clients execute the fix, and if they conflict, send their own fix to the fix. This can result in long chains consuming a lot of bandwidth and CPU. Thus the server rejects fixes past a certain chain length. Both academic and commercial approaches are examples of the consistency-responsiveness trade-off mentioned earlier.

2.3 Data Management

The commercial MMO Guild Wars 2 has had trouble scaling the number and complexity of *AI scripts* handled by its servers [35]. A research group at Cornell also observed this trade-off between the complexity of AI scripts and their quantity. They proposed a data-driven declarative language that factors similar queries together [52], thereby reducing the AI computations from quadratic to linear with the number of entities in the world. Pikko and BigWorld, two commercial middleware platforms for MMOs, recommend offloading AI scripts outside of the server, as if they were normal clients [3, 8]. This way, the server can dedicate more CPU processing client requests rather than complex AI scripts. However, there may be a significant bandwidth increase compared to the case where AI is run within the server.

Persistence-wise, a *central database* is usually a bottleneck in client-server architectures. This problem has been solved in commercial MMOs in three ways. First, they only persist the state of all connected avatars to the database in a batch every few minutes. While this alleviates the load on the database, avatar states have to rollback when a server crashes. The second technique addressing the database bottleneck in commercial MMOs is the use of a query manager: a machine stands between the database and the game servers as a buffer and a cache, as in the MMO Tibia [47]. A third approach consists of investing in expensive hardware for the database, also known as vertical scaling. For example, as of March 2011, each of the two machines hosting the database of EVE Online had 512 GB of RAM, 32 logical cores, 18 solid state drives, and used a 32 Gbps Infiniband link to communicate with the tier of game servers[8].

We saw earlier that spatial-partitioning was not always the best data partitioning scheme. In Darkstar (now called RedDwarf[9]), an entity's data is stored on a node independent of the entity's location in the game world [50]. In this way, Darkstar resembles the distributed memory caching of memcached[10]. But the main feature of Darkstar is its transactional tasks. Server reads can be performed on the local cache, but writes must be sent to a central server to check for conflicts with other writes. If no conflicts are detected, the transaction is executed, otherwise, the transaction fails and runs again after a period of time.

[8] See http://community.eveonline.com/devblog.asp?a=blog&nbid=2292

[9] See http://sourceforge.net/apps/trac/reddwarf/

[10] See http://memcached.org

This approach may add latency to some transactions, and particularly to those related to popular entities (e.g. thousands of players attacking the same boss). Darkstar clearly stands on the consistency side of the consistency-responsiveness spectrum.

2.4 Network and System Approaches

Several system architectures have been proposed for MMOs. A very common architecture seen in client-server academic prototypes and commercial MMOs alike is the *tiered architecture*. Clients connect to a tier of client managers (also known as proxies). Proxies forward messages between clients and the tier of game servers, running the game logic. Game servers persist their state in the database tier. Proxies are useful to externalize the load due to client handling on the server (e.g. socket management and data prioritization) [24]. Even though the average MMO packet size is less than 30 bytes [27], the number of messages to send is between linear and quadratic with the number of players. Tiered architectures were already mentioned in research in the early 2000's [23, 40], and have been used in Intel's DSG [34] and many commercial MMOs [2, 10, 32].

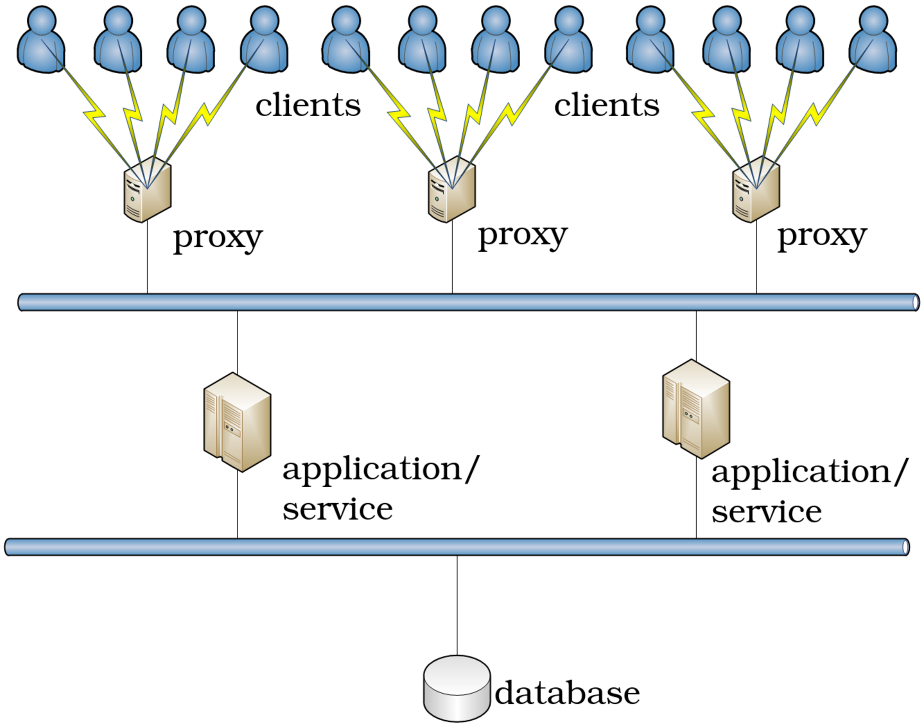


Figure 1: Tiered architecture

Intel's DSG does more than just a tiered architecture. It aims at breaking down the monolithic simulator-centric architecture and offloading *services* to external processes [34]. While external services provides a nice separation of concerns, it adds extra latency due to the extra hop in forwarding tasks to services. Once again, we see the responsiveness-scalability trade-off. Carlini et al. mixed peer-to-peer and cloud by using virtual nodes as an abstraction layer for the physical node location. Virtual nodes allow for transparent node relocation, is cheaper than a pure client-server solution, and more reliable than a pure peer-to-peer solution [14]. S-VON is a peer-to-peer Voronoi-based overlay network with super peers. While a pure peer-to-peer Voronoi-based overlay network only accounts for in-game proximity, the super peers also take into account the network proximity to reduce the latency [29].

Another system approach consists of using *lockless* tasks to avoid deadlocks between cores. This has been tried academically in [46] and comercially in the MMO called TERA [33]. Compared to offloading services to other machines, lockless game servers do not suffer from additional latency due to the extra hop. However, lockless game servers can only scale up to the number of cores in one machine. We see once again the trade-off between scalability and responsiveness.

3 RCAT

The spectrum of solutions for scaling up MMOs is large, and includes many application-specific optimizations, some of which were not covered in the previous section. We seek to develop a middleware that has the following characteristics:

- It is application independent, and it can support a variety of massive multiuser applications, from social networks to synchronous online education to real-time 3D games. That way the knowldege gained in developing one type of application is not lost when having to develop another type of application.
- It can scale horizontally, i.e. the demand for larger number of interacting users can be met simply by adding more servers doing the same functions.

In order to meet these goals, we approach the problem from an architectural perspective. Architectures, or more precisely architectural styles, are constraints over all things that can be done [48]. By setting constraints on applications in specific ways, we establish a set of principles that, on the one hand, disallow many potentially valid application-specific short-cuts, but that, on the other, ensure that the goals of knowledge sharing and horizontal scaling are met.

In this section, we first describe some of the foundations of RCAT, and then describe RCAT itself.

3.1 REST

Before the Web established itself as the main platform for Internet applications at the global scale, many other platforms had been proposed during the 80s and the 90s [25, 44]. The Web has given us two very important advantages over the competition at the time: (1) it created a common foundation for the development of a variety of applications; web developers can create many applications in many different domains using many different frameworks, languages and tools, but the architectural foundations are all the same; and (2) it allowed web applications to start small (one web server) and gracefully grow to server farms if the businesses so demand.

We would like MMOs to have these same properties. As such, RCAT is inspired by the architecture of Web applications, in particular REST [22]. REST stipulates five main architectural constraints that we adapt to the context of MMOs:

- *Client-server*: The state of the system is stored on the server-side, not on the client-side. Client-server architectures are commonplace in MMOs.

- *Tiers*: hardware proxies, load balancers, and software-level intermediaries are recommended to decouple clients from servers. Tiered architectures are not a new topic in MMO architecture [1].

- *Stateless protocol*: No client-specific information (i.e. context) can be stored on the server between two client requests. Our intuition is that servers that contain state cannot scale because of the overhead of potentially synchronizing the parts of the state that they need to share. This is a problem we mentioned in the previous section. REST recommends persisting the state to a database or the clients. This way, the tier of application servers can scale by simply adding more hardware running the application.

- *Uniform interface*: This involves two parts. First, a RESTful server interface should be able to receive messages with different formats, and answer client requests in the format they ask for. Thus metadata should accompany the client request to help the server understand how to interpret and answer the request – timestamps for example. Second, each resource should have a unique address (a URI). For example, the host machine's IP and port, and the GUID of an object suffice to identify and access that object uniquely.

- *Caches*: Caches are intermediaries between system components. Their use is highly recommended, since they can reduce computations and traffic considerably. Caches contain information that changes infrequently. In MMOs, players' positions are generally not cacheable, but textures and assets are. We saw earlier that several academic and commercial approaches replicate/cache game entities in adjacent regions for performance reasons.

We must be cautious when applying REST naively to MMOs: at first sight, the statelessness of the protocol between clients and application servers seems to be conflicting with the requirements of MMOs. A stateless protocol implies that either a) clients have to enclose to their request all the required data for the server to process their requests, or b) the server has to retrieve the necessary data from the database. In MMOs, clients clearly cannot enclose all necessary data with their request, because this can potentially mean the whole game state. Enclosing the whole game state is doable for a 2-player game of Tic-tac-toe or even Chess, but not reasonable for an MMO: bandwidth will quickly become limiting [51]. Thus servers need to fetch data from the database for each client request. As we show in the evaluation section, compared to the current monolithic architecture of commercial MMOs, REST can result in a higher latency to process client requests, more bandwidth consumed between server and database, and the database can become a computational bottleneck.

3.2 The RCAT Architecture

RCAT (RESTful Cient-server ArchiTecture) consists of four tiers: proxies, game servers, caches, and database. We first introduced this architecture in [19]. RCAT is very similar to the architecture shown in Figure 1. Proxies handle the communication with clients, game servers handle computation of the game logic or simulation, and the database ensures the persistence. Each tier isolates a different performance requirement, and therefore a potential bottleneck. This architecture aims at being game-agnostic, i.e. it does not embed any notion of a virtual space in it.

Proxies isolate the quadratic broadcast problem from the game servers. As mentioned before, proxies are common components in MMO architectures and on the web. Clients and game servers hold permanent connections with the proxies. Their role is simply to forward messages between clients and game servers. The game operator or developer can decide how the proxy forwards client requests to the servers: round-robin, fixed (a given proxy always forwards messages to the same game server), or per-client/"sticky" (all the messages from a given client are sent to the same server). Beside message forwarding, proxies can prioritize, bucket, piggyback, or filter messages based on certain network heuristics (e.g. upload rate or TCP window size). However, they do not have any knowledge whatsoever about the game (e.g. areas of interest or friend lists). Proxies can also help mitigating denial of service attacks against game servers.

Game servers receive and process client requests according to the game logic. When a server receives a request from a client (through a proxy), it computes which clients have to be notified, and broadcast to all the proxies the response to forward as well as the concerned clients. Servers can perform any game logic treatment, from low-latency state updates (e.g. avatar movement) to more reliable bulk-transfer content delivery (e.g. textures, or streaming the world state when a client first logs in). Servers can be added on the fly to the server tier at no synchronization cost. They only have to notify the proxies when they join in, and retrieve data from their cache.

Caches live on the same machines as the game servers so as to provide for the game servers a quick access to the data, and to alleviate the load on the database. Determining which object lives in which cache, so as to optimize data accesses, is the concern of the developer. For example, in space partitioning, two objects that are near each other in the game world would live in the same cache. Other types of partitioning include by users, as is currently done by Zynga [54], or by data types.

Caches are only a temporary storage to improve performance; it is up to the developer to specify which objects can be cached, the delay for cached objects to become stale, and whether replicas should be instantiated on other machines for faster reads. Basically, the cache tier is the place where the developer specifies the degrees of latency and consistency she needs for which objects.

Finally, the bottom tier is the database. It ensures persistence, and may still be directly accessed by the game server for transactional operations that must be ACID (atomic, consistent, isolated, and durable). When a cached object expires, the cache may retrieve the object's state from the database. If an object living in the cache must be strongly consistent, all the writes performed on the object should also be forwarded to the database. But if latency is important, the object's state can be flushed to the database only periodically.

4 RCAT Reference Implementation

To be able to implement several games, and test which strategies would work best in which cases, we implemented RCAT as a middleware in Python, running on the Linux operating system. This reference implementation of RCAT supports clients running as JavaScript applications on regular Web browsers and interacting with regular Web servers on the backend. The source code of RCAT, as well as two games built on top of it, are available at `https://github.com/gentimouton/rcat`. The components of the middleware are shown in Figure 2. The two higher-level components are the proxy and the game server.

The proxy communication is performed by Tornado, a non-blocking single-threaded web server supporting WebSockets[11]. The proxy has two URL access points: **/client** and **/server**. Clients connect to **/client**, where the client handler component generates a session user ID. The game handler then forwards client messages to a game server. Servers reply to the clients through the **/server** URL access point of the proxy. Messages sent to **/server** are treated by the game handler component, which determines the clients that should receive the message.

In the game server, the game logic and mapper components are the game-specific modules of the game server. The game logic contains the game rules. The mapper provides data services to the game logic through an API. Both are plugged into the RCAT's middleware, and interface with components from the proxy layer (the proxy connector) and the data layer (the persistence manager and the object manager).

The proxy connector provides two abstractions to the game logic. First, it de-multiplexes the WebSocket connections from the game server to the proxies

[11] Tornado has been used extensively in real-time web applications such as FriendFeed and Facebook's timeline. See `http://www.tornadoweb.org`

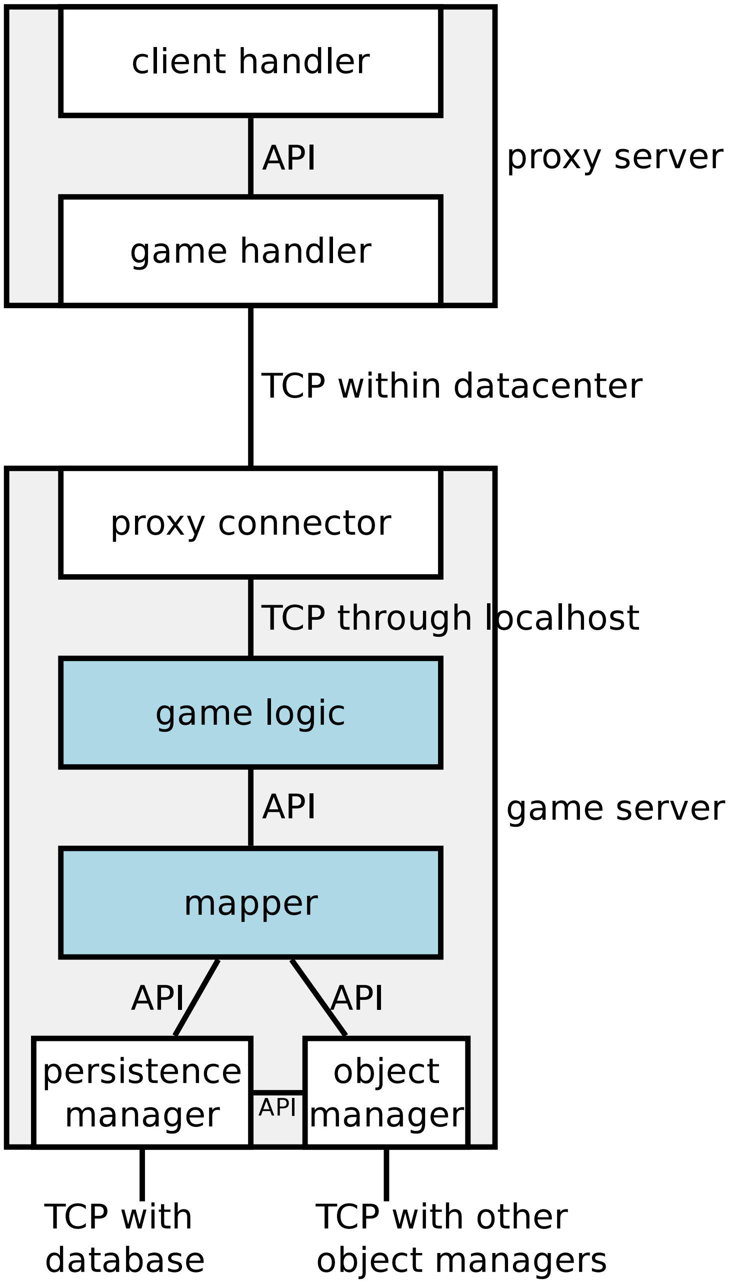


Figure 2: The RCAT components. The blue components are game-specific and implemented by the game developer.

into a single one. Second, it hides from the game logic the location of proxies and clients. If the game logic component wishes to message users, it simply sends the message and the set of recipients (i.e. a set of user IDs). The proxy connector parses the set, checks which proxies those users are connected to, and forwards the message to the appropriate proxies.

The persistence and object managers abstract away database specific protocols and the handling of objects in local and remote caches. Each provides an API to the mapper. Both are implemented in Python. Any database supported by SQL Alchemy can be used, but for our implementation we only tried MySQL.

The persistence manager provides a centralized, available, and consistent data access through SQL Alchemy[12], a database object-relational mapper.

The object manager provides a distributed data access, and the tools for managing data availability or consistency. The object manager was implemented by us and provides an API to 1) manipulate data stored in local or remote caches, and 2) relocate data stored in remote caches to the local cache.

WebSockets

The RCAT reference implementation uses WebSockets in order for the clients to receive messages from the servers. Here we give a brief description of WebSockets.

A WebSocket is a standard TCP connection initiated over HTTP, and growing in popularity for real-time web applications. The use of TCP in MMOs is a controversial debate in both research and industry. Some game developers would even rather implement a reliable protocol on top of UDP rather than use TCP[13]. We opt for TCP for three reasons. First, TCP removes the effort of controlling packet ordering and retransmission. Adequately configuring the TCP retransmission rate can greatly reduce the latency [24]. Second, MMOs generally use TCP, probably because clients send only between 1 and 10 messages per second [16, 24]. Using TCP makes our approach more valid and applicable. And finally, TCP has many congestion control mechanisms, such as Nagle's algorithm, that may help on the server-side when broadcasting the same message to many connections. Such solutions generally trade latency for bandwidth, or vice-versa. They are yet another illustration of the scalability-responsiveness trade-off in MMOs.

5 RCAT Reference Application: Jigsaw Puzzle

In order to study the performace characteristics of applications built with the RCAT architecture we have implemented a multiplayer client-server game using the reference RCAT implementation described above. Our virtual jigsaw puzzle does not impose any limit on the number of players. Moreover, a multiplayer jigsaw puzzle provides an interesting set of requirements:

- Players may scroll from one end of the board to another instantly,

[12] See `http://www.sqlalchemy.org/`

[13] See `http://gafferongames.com/networking-for-game-programmers/udp-vs-tcp/`

- Players may grab any piece at any time,
- Players must know immediately the pieces that are grabbed, moved, and dropped by other players,
- Players may zoom in or out, to see the overall picture, or to check for small details.

A screenshot of the HTML5 client is shown in Figure 3. Clearly, space partitioning will not be as effective for a multiplayer jigsaw puzzle as for MMOs with clearly-defined regions like World of Warcraft. However, interactions between players are minimal, and players may be likely to pick the same piece again. Thus we configure the proxy to "stick" each client to a server: all the messages sent by that client are forwarded to the same server. Clients are stuck to game servers in a round-robin fashion. We also configure the mapper to partition by players. When a game server receives a message concerning a particular piece from a particular player, it checks if the piece is cached in another server. If it is, then the server retrieves the remote piece, and places it in the local cache. If the piece is not cached anywhere, the server retrieves it from the database, and places it in the local cache. In short, whenever the server needs a piece, it will relocate that piece to the local cache.

6 Experiments and Results

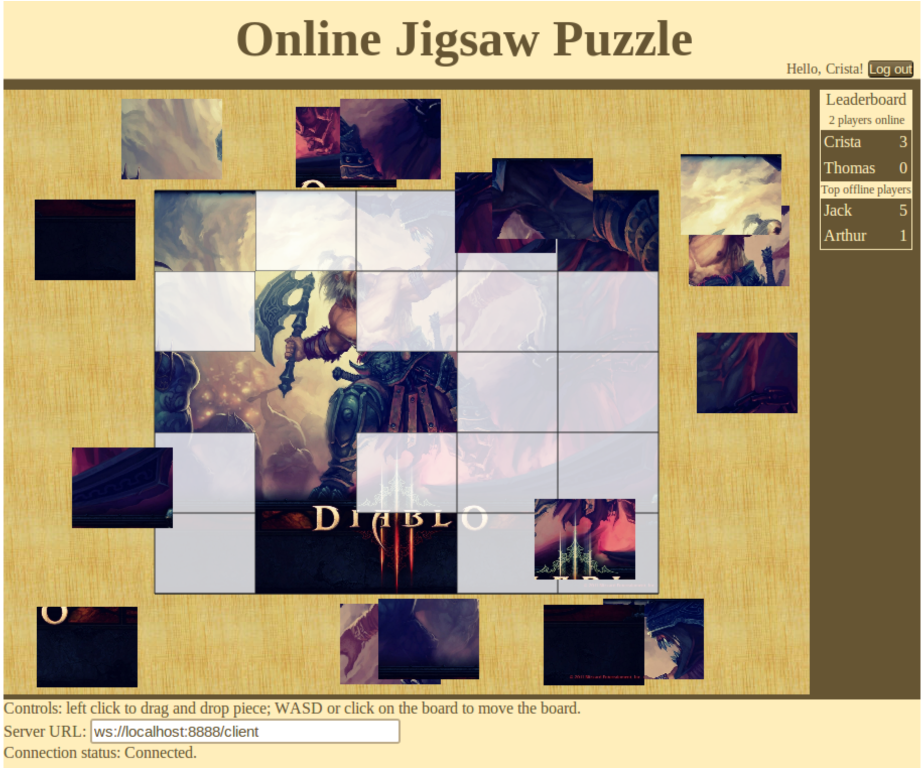
In this section, we confirm that the bottlenecks we anticipated on the proxy and database tiers actually exist in practice. We show how we scaled the number of players in the jigsaw puzzle with the number of cores and machines by using the RCAT middleware.

6.1 Experiment 1: Proxy and Database Bottlenecks

To confirm that the bottlenecks we anticipate on the proxy and database tiers actually exist in practice, earlier on we developed a proof-of-concept of a full-broadcast multi-player movement game following the RCAT architecture, but using a much simpler server side. This study is documented in [38] and the code available at `https://github.com/gentimouton/rcat-gs`. We include the main findings of that study here for the purposes of illustrating the basic performace characteristics of RCAT applications.

In this earlier study, both the proxy and the game server are implemented in C# using the .NET framework. The proxy accepts WebSocket connections through the multi-threaded Alchemy WebSockets server[14]. A graphical client was implemented in JavaScript using the HTML5 canvas for rendering, and the WebSocket API to connect to the proxy. We used bots, implemented in Java, to stress test the system. Everything ran in Windows 7.

[14] See `http://alchemyWebSockets.net/`



Online Jigsaw Puzzle

Hello, Crista!

Leaderboard	
2 players online	
Crista	3
Thomas	0
Top offline players	
Jack	5
Arthur	1

Controls: left click to drag and drop piece; WASD or click on the board to move the board.

Server URL: ws://localhost:8888/client

Connection status: Connected.

Figure 3: Multiplayer jigsaw puzzle with 25 pieces.

We are not trying to show that our architecture scales, but rather whether the proxy can be a bottleneck. To validate this hypothesis, we only need, and use, one proxy and one server. The commodity machines we use are Optiplex 980, with eight 2.8-Ghz i7 cores. We launch up to 50 bots, in increments of 5, to the proxy. Each bot sends 20 position messages per second. The proxy and database run on one machine, the server on another, and bots run on three other different machines (five machines total). As shown in Figure 4, the bandwidth from the proxy to the clients increase quadratically. Thus we can not expect to be able to scale the number of proxies linearly with the number of clients.

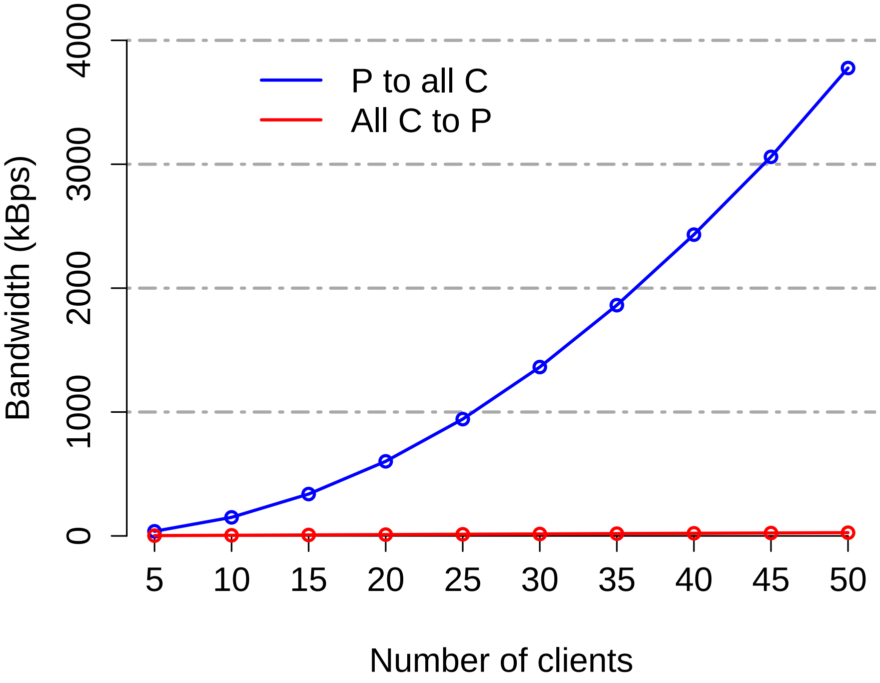


Figure 4: Bandwidth between all clients (all C) and proxy (P).

In this simple game, every time the game server receives a position message from a client, it updates in the database the position of the client's avatar, retrieves the list of connected clients, and sends to the proxy, in a single message, 1) the message to be forwarded to the clients, containing the avatar's new position, and 2) the list of clients to send the message to (i.e. everyone currently connected). To estimate the magnitude of the database bottleneck, we compare a scenario where the game server has to retrieve the list of all connected users for every client message, and another scenario where the game server caches the list of users, and does not have to ask the database every time.

As shown in Figure 5, the bandwidth from the server to the database increases linearly with the number of users, and decreases slightly when the list of current clients is cached (compare the two “S to DB” curves). More strikingly, the bandwidth from the database to the server increases quadratically with the number of clients. By caching the list of users, we have reduced a quadratic increase into a linear increase (compare the two “DB to S” curves). This proves that the database can be a central bottleneck, and that caching can be an effective strategy.

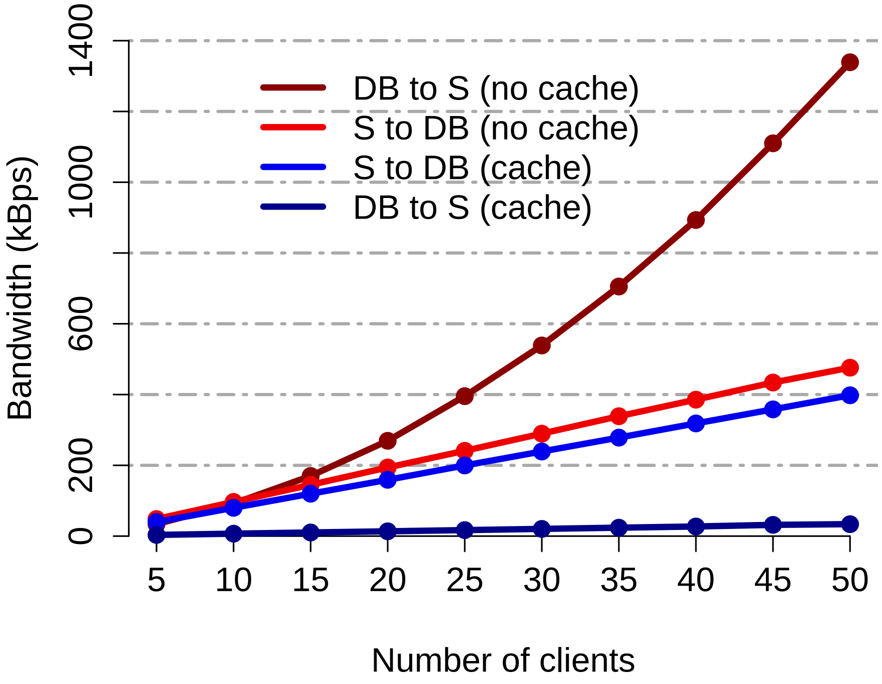


Figure 5: Bandwidth between database and server with or without cache.

6.2 Experiment 2: Scaling Up the Number of Players

The experimental results reported in this subsection are based on our RCAT reference application, the multiplayer online jigsaw puzzle.

6.2.1 Experimental Setup

Since all players must know about the jigsaw pieces that all other players are acting on, we let TCP delay and piggyback the transmission of ACKs (TCP_QUICKACK socket flag kept at 0). We keep the default value of 40 ms of delay before sending an ACK[15]. We also enable Nagle’s algorithm (TCP_NO_DELAY

[15] See http://lwn.net/Articles/502585/

socket flag kept at 0). The bots, proxies, and servers apply Nagle's algorithm and delayed ACKs.

The bots follow a scripted behavior. On both machines, a new bot joins the game every 10 seconds. Each bot starts by grabbing one piece, and moves it by several game units X times per second, with X a configurable parameter. This bot behavior results in no piece having to be relocated between caches. We acknowledge that with real users, piece relocations will happen. But unless many users fight for the same piece, these relocations should happen relatively rarely. We stop the script manually when we observe that the CPU capacity of the proxies or game servers are completely used. For example, the sum of the proxies' CPU may reach 400% if there are 4 proxies. However, Tornado recommends offloading tasks consuming considerable CPU to a thread pool. In our case, the game server keeps processing messages through Tornado, but offloads the game logic tasks to other threads. Consequently, the CPU for a single game server may reach more than 100% if the game logic threads are allocated to another core.

Bots measure the round-time-trip latency (latency, for short) using game messages containing globally unique identifiers. Each proxy and game server instance measures its (user plus system) CPU consumption, and the frequency of voluntary and involuntary context switches every five seconds using the Linux getrusage command[16]. We want to check how the number of proxies and the number of game servers scale with the number of bots for different message frequencies. We define the *maximum capacity* of the system as the highest number of connected clients when the 99^{th} percentile of the latency is below 100 ms.

The commodity machines used for the experiments are Dell Core i7 2600, each with four hyper-threaded 3.4-Ghz processors. Up to two machines run proxy instances, up to two others server instances, one other the database, and two others the bots. All machines reside on the same 1-Gbps LAN, and are one hop from each other. Experiment names follow the format "X/Y/Z", standing for X cores running one proxy each, Y cores running one game server each, and bots sending Z messages per second. For example, in the scenario 2/2/5, there are two proxy instances running on the same machine, two game servers sharing one other machine, and bots on two other machines send five messages per second. In the scenario 4+4/4+4/2.5, there are four proxies running on one machine, four proxies on another, four game servers on a third machine, and four game servers on a fourth machine.

6.2.2 Results

Figure 6 shows that the maximum capacity is reached at 229 clients for the scenario 4+4/4+4/2.5. In this scenario, the proxy is the bottleneck, and the server reaches slightly past half of its total CPU capacity (400% out of 800% available). On the one hand, the total CPU consumed by the game servers

[16] See `http://linux.die.net/man/2/getrusage`

increases linearly with the number of clients. On the other hand, the CPU consumed by the proxies increases quadratically at first, then flattens around 100 clients, and finally seems to increase linearly. We remark this behavior in most of the scenarios where the proxy is a bottleneck.

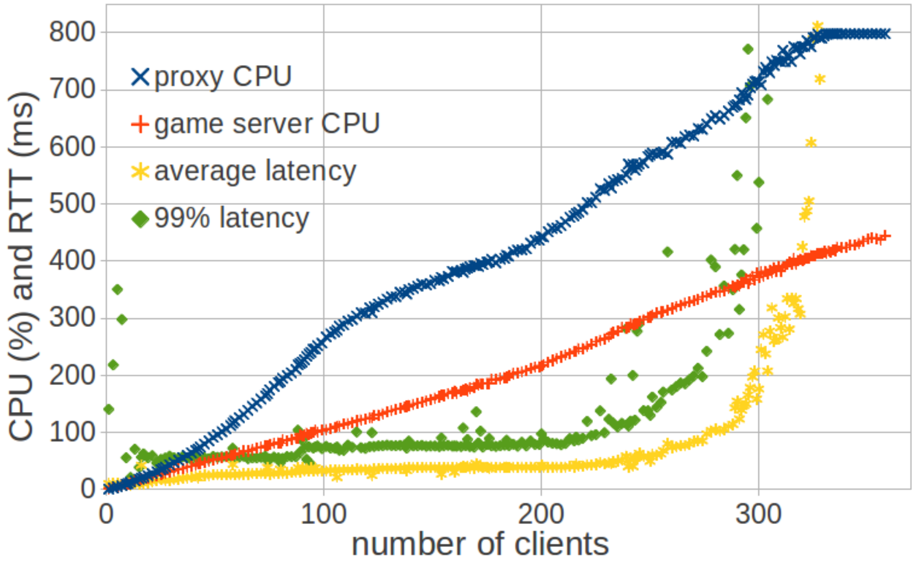


Figure 6: Evolution of the proxies CPU, game servers CPU, average latency, and 99^{th} percentile of the latency with the number of clients when using 4+4 proxies and 4+4 game servers. The last 99^{th} percentile of the latency to be below 100 ms is the maximum capacity: 229 clients.

Figure 7 shows that the number of voluntary context switches also reaches a plateau around 100 clients. Voluntary context switches occur when Tornado polls the client sockets, while involuntary context switches are triggered by the kernel scheduler[17]. Thus there may be a bottleneck on the proxy well before the maximum capacity is reached. On the plus side, this bottleneck does not seem to impact the latency observed on the client side. It is possible that the TCP congestion control mechanisms (e.g. piggybacks and ACK delays) buffer the extra delay taken by the proxy.

However, these mechanisms are not enough to prevent the number of voluntary context switches to fall and the latency to spike around 200 clients. Context switches remain the principal reason: since each core ticks 250 times per second[18], the maximum number of context switches, voluntary or not, for a system with eight proxies is $8 \times 250 = 2000$ per second. This is the cap reached around 190 clients. The proxies' cores are overwhelmed.

[17] See `http://www.lindevdoc.org/wiki/Involuntary_context_switch`

[18] The default value of a jiffy is 250 Hz, see `http://man7.org/linux/man-pages/man7/time.7.html`

Interestingly, the proxies are only using half of their CPU capacity when this happens. Clearly, the proxies' CPU is not a sufficient metric by itself to assess the quality of experience of the players, or even the load on the system.

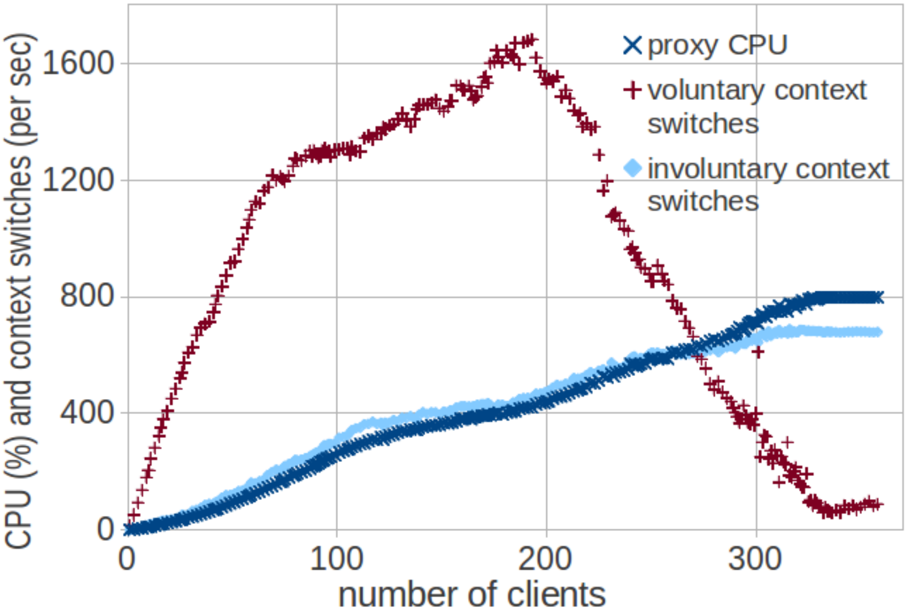


Figure 7: Evolution of the proxies CPU and context switches with the number of clients when using 4+4 proxies and 4+4 game servers.

Figure 8 plots the maximum capacities for various scenarios. Simply put, to be able to handle twice more players, the system needs four times more cores. And this is independent of the message frequency. It is not because a game is slower-paced that it scales more linearly. Yet slower-paced games can handle more players per core.

Finally, Figure 9 illustrates that message frequencies can result in different bottlenecks. We take the scenario with eight proxies and eight game servers as an example. At 2.5 messages per second, the maximum capacity (229 clients) is reached while the game servers only consume around 250% CPU. Meanwhile, the proxies consume twice as much CPU. In contrast, when clients send 10 messages per second, the game servers consume more CPU than the proxies when reaching the overall system's maximum capacity (125 clients). In this scenario, it is difficult to know which of the proxies or the game servers is the bottleneck. Yet it seems, surprisingly, that handling many slow-paced connections is more costly than handling few active connections.

Thus for slow message rates, the proxy is the bottleneck. But for fast message rates, the game server may be limiting. In fact, the game logic of our jigsaw puzzle is very simple (e.g. no collision detection). Therefore the server CPU

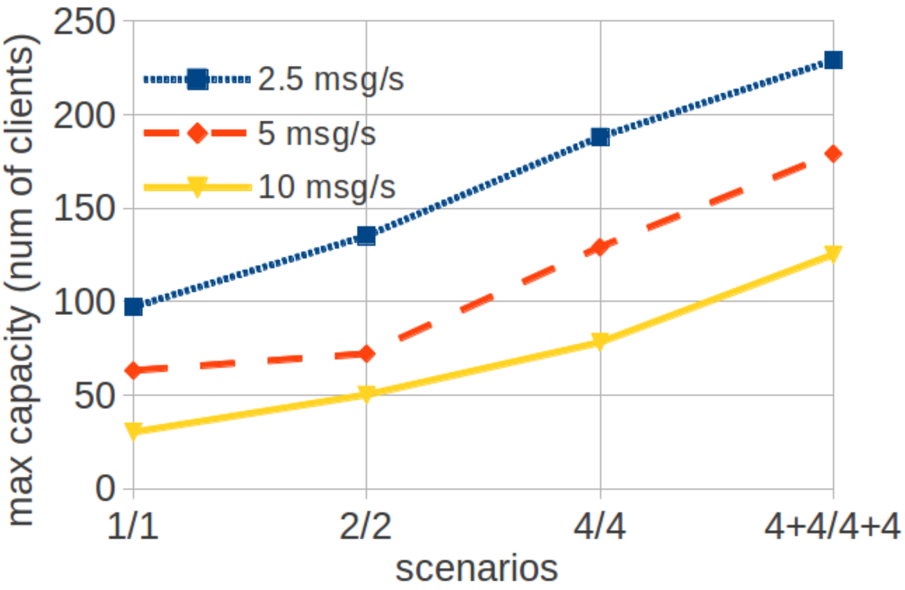


Figure 8: Scaling the number of clients with the number of cores and machines, when bots send 2.5, 5, and 10 messages per second.

may actually increase faster with the message frequency than shown in Figure 9. Provisioning how many proxies and game servers are needed for a given game is not a trivial: 1-to-1 ratios may rarely be optimal.

7 Discussion

Research has paid much of attention to MMOs that are easy to partition spatially. We showed one example of MMO where space-partitioning does not apply: a multiplayer jigsaw puzzle. We partitioned the data by user, but did not try other partitioning schemes. Which type of partitioning is best for a multiplayer jigsaw puzzle remains an open question. In fact, this question is not specific to MMOs: developers of other multiuser online applications, such as massive open online courses, are facing the same challenges. We are actively looking for more examples of massively multiuser online applications, as they may highlight new scalability challenges and solutions.

But space partitioning may still be appropriate for many MMOs. In fact, picking the appropriate load-partitioning algorithm is a problem common to any massively multiuser application. In our jigsaw puzzle, we originally considered a space partitioning scheme for the jigsaw puzzle MMO: cutting the board into square regions, and assigning a server per region. We quickly realized that this approach would be very inefficient, since users would move pieces between

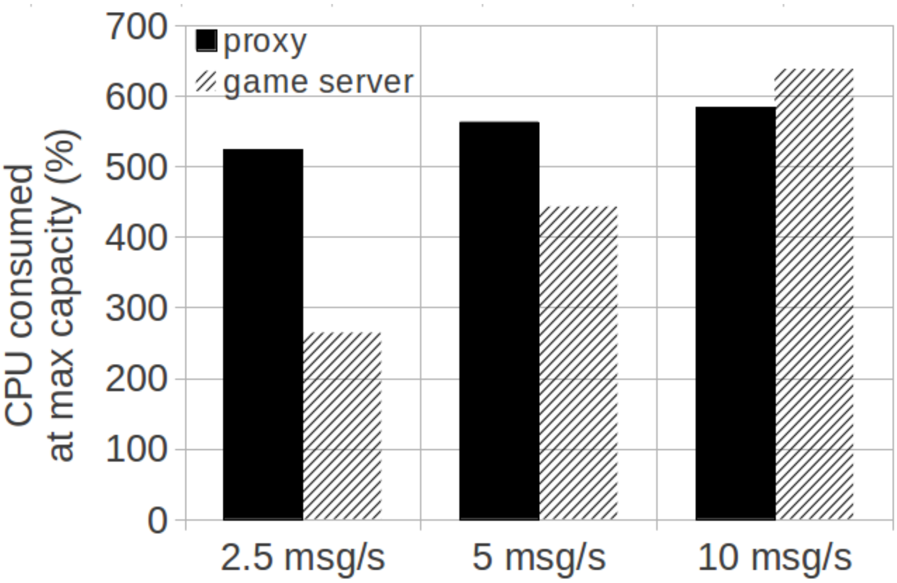


Figure 9: CPU consumed when maximum capacity is reached with 4+4 proxies and 4+4 game servers.

regions as often as within regions. This would cause data handovers too often, and result in poor performance.

Thus we opted for a pair (client, puzzle piece) for load balancing. Whenever a client attempts to move a piece, the piece is relocated to the server treating the client's request. This way, the data for the piece is always located where the client's request is being treated. Another successful approach may have been to move the clients messages to the application server where the jigsaw piece is cached.

The implementation of RCAT is in active development. While some parts are application-independent, the game developer needs to write the application-specific parts. We use Python for the mapper, and therefore also for the game logic, as the persistence and object managers only provide a Python API. As the middleware development progresses, different implementations of the object manager and persistence manager will provide more flexibility and choice for a wider range of applications.

Currently, each game server instantiates its own object manager. But game servers running on the same machine could use the same object manager. Implementing this feature may require game servers to be able to tell whether other game servers are local or remote, or to slightly change our architecture.

We noted throughout our experiments that a bottleneck may hide another. Context switches on the proxies may be the limiting factor to scale, and not the proxies CPU. Moreover, the number of voluntary context switches seem to spike right before the latency starts spiking. Thus context switches may prove to be a more adequate indicator than the CPU to gauge whether the current load on the system actually impacts the players' experience. It may also prove to be a useful metric for provisioning.

More generally, our work informs the provisioning of massively multiuser online environments. For an application to support twice more users, the operator needs to deploy four times more hardware. In the case of the Jigsaw puzzle, and assuming the test conditions of our experiment (i.e. all users moving pieces all the time), we estimate that we would be able to support nearly 1,000 users with 256 cores. Being able to do these upfront estimations is a major engineering feat with business implications. Operators may also need to provision a sufficient network infrastructure to support the client connections as well as communication between proxies and application servers. In our experiments, a single 1-Gbps switch was sufficient. Thus network provisioning may only be a concern well after the 200-user mark.

8 Conclusion

In this article, we have presented RCAT, a scalable and adaptable three-tiered architecture for massively multiplayer online games. RCAT aims at delivering the flexibility of choosing the individual trade-offs found in the extensive variety of MMOs, while providing a solid infrastructural middleware that abstracts the complexities of scaling through a distributed system. RCAT also provides a common platform for developers and researchers to develop applications and compare results of different partitioning schemes.

We have demonstrated the potential of the architecture by presenting a multiplayer jigsaw puzzle, featuring full broadcasting of events, data locality, and massive number of users. Despite the unavoidable and essential quadratic event distribution, we can support 229 concurrent clients sending 2.5 messages per second, and 125 clients sending 10 messages per second, using commodity servers with 16 cores total. Scaling to higher numbers of clients is possible simply by deploying more proxies and game servers such that a target of twice the number of users requires 4x the number of cores.

As the Web becomes more supportive of rich interactive applications, we believe our work can be the foundation for a variety of massive multi-user applications.

References

[1] Abrams, H.: Three-tiered interest management for large-scale virtual environments. In: In Proceedings of the ACM symposium on Virtual reality software and technology (1998)

[2] Almroth, D.: Pikko Server. In: Erlang User Conference (2010). URL http://www.erlang-factory.com/conference/ErlangUserConference2010/speakers/DavidAlmroth

[3] Almroth, D., Lonnholm, C.: PikkoTekk - Horizontal scalability. Tech. rep., MuchDifferent (2011). URL http://download.muchdifferent.com/pikkoserver/pikko-server-scalability.pdf

[4] Almroth, D., Lonnholm, C.: PikkoTekk technical summary. Tech. rep., MuchDifferent (2011). URL http://download.muchdifferent.com/pikkoserver/pikko-server-tech-summary.pdf

[5] Baldi, P., Lopes, C.: The universal campus: An open virtual 3-d world infrastructure for research and education. eLearn **2012**(4) (2012). DOI 10.1145/2181207.2206888. URL http://doi.acm.org/10.1145/2181207.2206888

[6] Bharambe, A., Douceur, J.R., Lorch, J.R., Moscibroda, T., Pang, J., Seshan, S., Zhuang, X.: Donnybrook: enabling large-scale, high-speed, peer-to-peer games. In: Proceedings of the ACM SIGCOMM 2008 conference on Data communication, SIGCOMM '08, pp. 389–400. ACM, New York, NY, USA (2008). DOI 10.1145/1402958.1403002. URL http://doi.acm.org/10.1145/1402958.1403002

[7] Bharambe, A.R., Rao, S., Seshan, S.: Mercury: a scalable publish-subscribe system for internet games. In: Proceedings of the 1st workshop on Network and system support for games, NetGames '02, pp. 3–9. ACM, New York, NY, USA (2002). DOI 10.1145/566500.566501. URL http://doi.acm.org/10.1145/566500.566501

[8] BigWorld: Server overview. Tech. rep. (2009)

[9] Boulanger, J.S., Kienzle, J., Verbrugge, C.: Comparing interest management algorithms for massively multiplayer games. In: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games, NetGames '06. ACM, New York, NY, USA (2006). DOI 10.1145/1230040.1230069. URL http://doi.acm.org/10.1145/1230040.1230069

[10] Brandt, D.H.: Scaling EVE Online, under the hood of the network layer. Tech. rep., CCP Games (2005)

[11] Brewer, E.: Towards Robust Distributed Systems (2000). URL http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[12] Brewer, E.: Cap twelve years later: How the "rules" have changed. Computer **45**(2), 23–29 (2012). DOI 10.1109/MC.2012.37

[13] Buyukkaya, E., Abdallah, M.: Data management in voronoi-based p2p gaming. In: Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE, pp. 1050–1053 (2008). DOI 10.1109/ccnc08.2007.239

[14] Carlini, E., Coppola, M., Ricci, L.: Integration of p2p and clouds to support massively multiuser virtual environments. In: Proceedings of the 9th Annual Workshop on Network and Systems Support for Games, NetGames '10 (2010). URL http://portal.acm.org/citation.cfm?id=1944796.1944813

[15] Carlsson, C.: DIVE A multi-user virtual reality system. Virtual Reality Annual International pp. 394–400 (1993). DOI 10.1109/VRAIS.1993.380753. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=380753http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=380753

[16] Chen, K.T., Lei, C.L.: Network game design: hints and implications of player interaction. In: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games, NetGames '06. ACM, New York, NY, USA (2006). DOI 10.1145/1230040.1230083. URL http://doi.acm.org/10.1145/1230040.1230083

[17] Cronin, E., Filstrup, B., Kurc, A.R., Jamin, S.: An efficient synchronization mechanism for mirrored game architectures. In: Proceedings of the 1st workshop on Network and system support for games, NetGames '02, pp. 67–73. ACM, New York, NY, USA (2002). DOI 10.1145/566500.566510. URL http://doi.acm.org/10.1145/566500.566510

[18] De Vleeschauwer, B., Van Den Bossche, B., Verdickt, T., De Turck, F., Dhoedt, B., Demeester, P.: Dynamic microcell assignment for massively multiplayer online gaming. In: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games, NetGames '05, pp. 1–7. ACM, New York, NY, USA (2005). DOI 10.1145/1103599.1103611. URL http://doi.acm.org/10.1145/1103599.1103611

[19] Debeauvais, T., Nardi, B.: A qualitative study of Ragnarök Online private servers: in-game sociological issues. In: Proceedings of the Fifth International Conference on the Foundations of Digital Games, FDG '10 (2010)

[20] Denault, A., Cañas, C., Kienzle, J., Kemme, B.: Triangle-based obstacle-aware load balancing for massively multiplayer games. In: Proceedings of the 10th Annual Workshop on Network and Systems Support for Games, NetGames '11, pp. 4:1–4:6. IEEE Press, Piscataway, NJ, USA (2011). URL http://dl.acm.org/citation.cfm?id=2157848.2157853

[21] Fiedler, S., Wallner, M., Weber, M.: A communication architecture for massive multiplayer games. In: Proceedings of the 1st workshop on Network and system support for games, NetGames '02, pp. 14–22. ACM, New York, NY, USA (2002). DOI 10.1145/566500.566503. URL http://doi.acm.org/10.1145/566500.566503

[22] Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000)

[23] Griwodz, C.: State replication for multiplayer games. In: Proceedings of the 1st workshop on Network and system support for games, NetGames '02, pp. 29–35. ACM, New York, NY, USA (2002). DOI 10.1145/566500.566505. URL http://doi.acm.org/10.1145/566500.566505

[24] Griwodz, C., Halvorsen, P.: The fun of using tcp for an mmorpg. In: Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video, NOSSDAV '06, pp. 1:1–1:7. ACM, New York, NY, USA (2006). DOI 10.1145/1378191.1378193. URL http://doi.acm.org/10.1145/1378191.1378193

[25] Grosso, W.: Java RMI. O'Reilly (2001)

[26] Gupta, N., Demers, A., Gehrke, J., Unterbrunner, P., White, W.: Scalability for virtual worlds. In: Proceedings of the 2009 IEEE International Conference on Data Engineering, ICDE '09, pp. 1311–1314. IEEE Computer Society, Washington, DC, USA (2009). DOI 10.1109/ICDE.2009.228. URL http://dx.doi.org/10.1109/ICDE.2009.228

[27] Harcsik, S., Petlund, A., Griwodz, C., Halvorsen, P.: Latency evaluation of networking mechanisms for game traffic. In: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games, NetGames '07, pp. 129–134. ACM, New York, NY, USA (2007). DOI 10.1145/1326257.1326280. URL http://doi.acm.org/10.1145/1326257.1326280

[28] Horn, D., Cheslack-Postava, E., Azim, T., Freedman, M.J., Levis, P.: Scaling virtual worlds with a physical metaphor. IEEE Pervasive Computing **8** (2009)

[29] Hu, S.Y., Wu, C., Buyukkaya, E., Chien, C.H., Lin, T.H., Abdallah, M., Jiang, J.R., Chen, K.T.: A spatial publish subscribe overlay for massively multiuser virtual environments. In: Electronics and Information Engineering (ICEIE), 2010 International Conference On, vol. 2, pp. V2–314–V2–318 (2010). DOI 10.1109/ICEIE.2010.5559789

[30] Iimura, T., Hazeyama, H., Kadobayashi, Y.: Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games, NetGames '04, pp. 116–120. ACM, New York, NY, USA (2004). DOI 10.1145/1016540.1016549. URL http://doi.acm.org/10.1145/1016540.1016549

[31] Itzel, L., Heger, F., Schiele, G., Becker, C.: The quest for meaningful mobility in massively multi-user virtual environments. In: Network and Systems Support for Games (NetGames), 2011 10th Annual Workshop on, pp. 1 –2 (2011). DOI 10.1109/NetGames.2011.6080991

[32] Koo, S.: How to support an action-heavy MMORPG from the angle of server architecture. In: Chinese Game Developers Conference (2010)

[33] Koo, S.: TERA: Evolving MMORPG combat. Game Developer Magazine (2012)

[34] Lake, D., Bowman, M., Liu, H.: Distributed scene graph to enable thousands of interacting users in a virtual environment. In: Proceedings of the 9th Annual Workshop on Network and Systems Support for Games, NetGames '10 (2010). URL `http://portal.acm.org/citation.cfm?id=1944796.1944815`

[35] Lewis, M.: Managing the masses. In: Game Developers Conference, GDC (2012)

[36] Liu, H., Bowman, M.: Scale virtual worlds through dynamic load balancing. Distributed Simulation and Real Time Applications **0**, 43–52 (2010). DOI http://doi.ieeecomputersociety.org/10.1109/DS-RT.2010.14

[37] Liu, H., Bowman, M., Adams, R., Hurliman, J., Lake, D.: Scaling virtual worlds: Simulation requirements and challenges. In: Winter Simulation Conference'10 (2010)

[38] Lopes, C., Debeauvais, T., Valadares, A.: Restful massively multi-user virtual environments: A feasibility study. In: Games Innovation Conference (IGIC), 2012 IEEE International, pp. 1–4 (2012). DOI 10.1109/IGIC.2012.6329833

[39] Mauve, M.: How to keep a dead man from shooting. In: In Proceedings of the 7 th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS) 2000, pp. 199–204 (2000)

[40] Mauve, M., Fischer, S., Widmer, J.: A generic proxy system for networked computer games. In: Proceedings of the 1st workshop on Network and system support for games, NetGames '02, pp. 25–28. ACM, New York, NY, USA (2002). DOI 10.1145/566500.566504. URL `http://doi.acm.org/10.1145/566500.566504`

[41] Miller, J.L., Crowcroft, J.: The near-term feasibility of P2P MMOG's. In: NetGames '10 (2010)

[42] Mousavi, H., Khademi, M., Dodakian, L., Cramer, S.C., Lopes, C.V.: A spatial augmented reality rehab system for post-stroke hand rehabilitation. Studies in health technology and informatics **184**, 279 (2013)

[43] Najaran, M.T., Krasic, C.: Scaling online games with adaptive interest management in the cloud. In: Proceedings of the 9th Annual Workshop on Network and Systems Support for Games, NetGames '10, pp. 9:1–9:6. IEEE Press, Piscataway, NJ, USA (2010). URL `http://dl.acm.org/citation.cfm?id=1944796.1944805`

[44] Orfali, R., Harkey, D., Edwards, J.: The Essential Distributed Objects Survival Guide. Wiley (1995)

[45] Pang, J.: Scaling peer-to-peer games in low-bandwidth environments. In: In Proc. 6th Intl. Workshop on Peer-to-Peer Systems (IPTPS (2007)

[46] Raaen, K., Espeland, H., Stensland, H.K., Petlund, A., Halvorsen, P., Griwodz, C.: A demonstration of a lockless, relaxed atomicity state parallel game server (lears). In: Proceedings of the 10th Annual Workshop on Network and Systems Support for Games, NetGames '11, pp. 19:1–19:3. IEEE Press, Piscataway, NJ, USA (2011). URL `http://dl.acm.org/citation.cfm?id=2157848.2157870`

[47] Rudy, M.: Inside tibia - the technical infrastructure of an mmorpg. In: Game Developers Conference, Europe, GDC Europe (2011)

[48] Taylor, R.N., Medvidovic, N., Dashofy, E.M.: Software Architecture: Foundations, Theory, and Practice. Wiley Publishing (2009)

[49] Van Den Bossche, B., De Vleeschauwer, B., Verdickt, T., De Turck, F., Dhoedt, B., Demeester, P.: Autonomic microcell assignment in massively distributed online virtual environments. Journal of Network and Computer Applications **32**(6), 1242–1256 (2009). DOI 10.1016/j.jnca.2009.04.001. URL `http://linkinghub.elsevier.com/retrieve/pii/S1084804509000575`

[50] Waldo, J.: Scaling in games and virtual worlds. Commun. ACM **51** (2008)

[51] Watte, J.: REST and games don't mix. `http://engineering.imvu.com/2010/12/18/rest-and-games-dont-mix/`

[52] White, W., Demers, A., Koch, C., Gehrke, J., Rajagopalan, R.: Scaling games to epic proportions. In: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, SIGMOD '07, pp. 31–42. ACM, New York, NY, USA (2007). DOI 10.1145/1247480.1247486. URL `http://doi.acm.org/10.1145/1247480.1247486`

[53] Yamamoto, S., Murata, Y., Yasumoto, K., Ito, M.: A distributed event delivery method with load balancing for mmorpg. In: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games, NetGames '05, pp. 1–8. ACM, New York, NY, USA (2005). DOI 10.1145/1103599.1103610. URL `http://doi.acm.org/10.1145/1103599.1103610`

[54] Zubek, R.: Scalability for social games. In: Game Developers Conference Online, GDC Online (2010)