A Walkable Kademlia Network for Virtual Worlds

Matteo Varvello[†⋆], Christophe Diot[†], Ernst Biersack[⋆]
[†] Thomson, Paris, France
[⋆] Institut Eurecom, Sophia-Antopolis, France
{matteo.varvello,christophe.diot}@thomson.net, ernst.biersack@eurecom.fr

***Abstract*—Virtual worlds are digitalizations of the real world where users can live a virtual life. Commercial virtual worlds rely on a Client/Server architecture, which has serious scalability limitations. Peer-to-Peer (P2P) and Distributed Hash Tables (DHTs) are a cheap and scalable alternative to a Client/Server approach. However, current DHT designs are not flexible enough to support such complex applications. We introduce Walkad, a Kademlia extension specifically designed to efficiently manage virtual worlds. The Walkad design leverages on the Kademlia DHT and on a novel indexing algorithm based on a reverse binary trie. We evaluate Walkad via emulation, and using traces extracted from Second Life. Our preliminary results show that Walkad is an efficient P2P design for virtual worlds. Walkad guarantees to its users a fast discovery of the virtual world, while load balancing the virtual land responsibilities among peers.**

I. Introduction

In the last years, we observed an explosive growth of virtual worlds. A virtual world consists of a virtual land populated by *objects* where human-controlled *avatars* can move, interact, or even trade. Second Life is probably the most popular virtual world with more than 15 Millions registered users [17].

Virtual worlds are implemented using a Client/Server (C/S) architecture. A server stores a copy of all objects that reside on the virtual land. The clients run stateless applications that allow users to explore the virtual world through the eyes of their avatar. To do so, the clients send *range queries* to the server, i.e., requests for the objects which spatial coordinates are located within a given range. In practice, avatars identify the set of objects, e.g., trees, cars or buildings, located in their surroundings by sending to the server a range query with range equal to the avatar visibility area.

Range queries in virtual worlds can be divided into *local* and *non-local*. A local query consists in a request for objects located in the avatar surroundings, i.e., the query's range is close to the avatar coordinates. For example, avatars generate local queries when they walk, run or fly, in order to constantly update their visibility area. A non-local query is a request for virtual objects that are located far away from the avatar coordinates. For example, avatars generate non-local queries when they suddenly cover a very large distance via the teleport operation. Local queries must be answered quickly to ensure a good user experience. Conversely, a higher delay in answering non-local queries may be tolerable [16].

Both local and non-local queries in virtual worlds are easy to manage with a C/S architecture. However, this architecture exhibits poor scalability and high cost [17]. A scalable alternative is to use a Peer-to- Peer (P2P) approach, where the virtual world is maintained exploiting the users' resources [2][5].

Distributed Hash Tables (DHTs) are popular P2P architectures used to store and retrieve content [7][11]. DHTs use a hash function (e.g., SHA-1 [9]) to distribute content fairly among peers. This design is very efficient to build a scalable P2P lookup system, but allows only to address content specifically. Thus, is not flexible enough to handle range queries. This limitation of DHTs is a bottleneck for their applicability to new applications such as virtual worlds.

In this work, we design and evaluate Walkad, a distributed architecture for the management of range queries in virtual worlds (Section IV). We design Walkad as an extension of the Kademlia DHT [7]. The reasons for this are twofold. First, the Kademlia design is very convenient for an extension to support range queries. Second, Kademlia is a very popular DHT successfully adopted by Kad [12], the P2P network used to locate content in the eMule filesharing application. Kad can also be an interesting platform for testing new distributed applications under realistic conditions [16].

Walkad organizes the Kademlia keyspace in a *reverse binary trie*, i.e., a tree-based data structure where nodes of each level of the tree are labeled using the Gray Code [4]. In this way, Walkad maps closeby portions of the virtual world, named *cells*, to keys that are closeby in the Kademlia keyspace. In other words, a Walkad peer responsible of a cell maintains routing information towards peers responsible for closeby cells. Therefore, local queries, which are the most popular [6], are quickly answered by "walking" across the Kademlia routing tables.

We evaluate Walkad via network emulation [13] (Section V). We build a Walkad network with a maximum of 1024 peers and we use object traces from Second Life [17] to simulate a realistic virtual world. Finally, we use synthetic traces of avatar movements to study different types of range queries.

Our preliminary results show that in a virtual world made of five Second Life regions indexed in a Walkad network of 1024 peers, local queries are answered in less than $150\ ms$ (in average), while non-local queries require about $200\ ms$. This result is very promising if we consider that acceptable latency values in C/S virtual worlds varies between $300\ ms$ and $1\ sec$ [3]. In addition, Walkad distributes equally the object load to peers as the network and virtual world sizes increase.

II. Related Work

Range-queries over DHTs have been a very fertile research area. We have identified two pieces of work that are relevant

to Walkad. Ramabhadran et al. [10] design the Prefix Hash Tree (PHT), a distributed data structure that enables range queries over any DHT. PHT organizes keys in the DHT as a binary trie, and uses the DHT lookup operation to handle range queries. Walkad is different from [10] since it integrates the indexing algorithm with the underlying routing algorithm. This design rational is similar to the one used in P-Grid [1]. P-Grid uses a self-organization process to structure peers in a binary trie. Thus, P-Grid dramatically reduces the number of routing hops to answer range queries compared to PHT. Walkad is different from P-Grid since it is designed as a Kademlia extension and leverages on a reverse binary trie. Moreover, Walkad performance are optimized for the management of local queries.

III. Background

We now introduce Kademlia and the Gray code. Then, we describe a simple approach to range queries over a DHT-based virtual world such as found in [16].

A. Kademlia

Kademlia is a structured P2P network, where peers and content are identified by a random 160-bit identifier [7]. Given two identifiers, a and b, Kademlia defines the *distance* between them as their bitwise exclusive or (XOR).

A Kademlia peer keeps for each $0 \leq i < 160$ bit of its identifier a list of peers with XOR distance $2^i \leq d < 2^{(i+1)}$ from itself. These lists are called *k-buckets*, where k defines the maximum number of entries per bucket. The entries in the n^{th} k-bucket have a different n^{th} bit from the peer identifier.

Routing in Kademlia is done iteratively. A message to a destination key is simply forwarded to one of the peers from the bucket with the longest common prefix to the target key. To store and search a *<key,value>* pair, a peer locates the closest peers to a key. The k-bucket structure allows Kademlia to contact only $O(log(N))$ peers during a lookup.

B. Gray Code

The Gray Code [4] is a binary numeral system where two successive values differ in only one digit. The $(n+1)$-bit Gray Code is constructed as follows: (1) we *reflect* the bits of the n-bit Gray Code, i.e., we list them in the reverse order, (2) we prefix the original bits with a 0, and the reflected bits with a 1, (3) we concatenate the reverse list to the original list.

Figure 1 shows an example of a *3*-bit Gray Code computation. On the left portion of Figure 1, we can see the *1*-bit Gray Code, i.e., the most basic Gray Code, $G = \{0, 1\}$. In order to compute the *2*-bit Gray Code, we reflect $G = \{0, 1\}$ obtaining $G^{'} = \{1, 0\}$. Then, we prefix the original bits (G) with a 0, and the new bits ($G^{'}$) with a 1, obtaining the *2*-bit Gray Code, $G = \{00, 01, 11, 10\}$. The same procedure is applied to compute the *3*-bit Gray Code.

C. A Simple Approach to Range Queries

A *binary trie* is a tree-based data structure that uses prefix bits to direct branching in a tree. Conventionally, a 0 represents a left branch and a 1 represents a right branch. Each node in the trie is associated to a label composed by the set of bits indicating the path in the trie to reach the node.

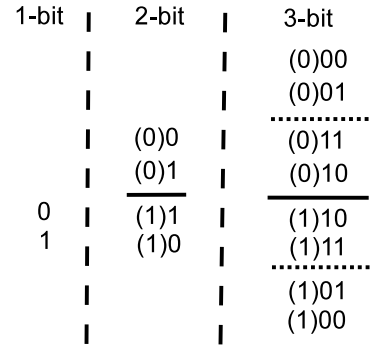


Fig. 1. Example of the Gray Code computation

Ramabhadran et al. [10] were the first to propose the usage of a trie to handle range queries over a DHT. Successively, Varvello et al. [16] use a similar approach to deploy a virtual world over the Kad DHT [12]. In the following, we refer to the architecture described in [16] as a *simple* approach. This architecture is also representative of [10].

We call a *cell* a portion of the virtual world, and we say that originally the virtual world is composed of a single cell. Then, the world is recursively divided into multiple cells as objects are created. A cell division occurs when the number of virtual objects becomes larger than a threshold D_{max}. Assuming a two dimensional space, a cell is first split on the vertical dimension, then on the horizontal, and so on. Similarly, two adjacent cells are merged when the sum of the virtual objects they contain is smaller than D_{min}, with $D_{min} < D_{max}$. In this way, we avoid the system to oscillate between splitting and merging cells in case of very dynamic environments.

The sequence of splits that generate a cell identify a path in the trie. A cell is uniquely identified by the label of the leaf corresponding to the computed path in the trie. We call *cell-ID* the key of the DHT associated to a cell. The cell-ID is computed by hashing the label of the leaf in the trie corresponding to the cell. This hash operation [9] ensures the desired balanced distribution for the cell-IDs in the keyspace.

A range query spanning a portion of the virtual world is solved as follows. First, we locate the leaves in the trie corresponding to the virtual cells that intersect the query's range. Then, we hash the labels of the leaves to obtain the corresponding cell-IDs. Finally, we perform the lookup in the DHT of the derived cell-IDs.

IV. Walkad

In this Section, we describe the Walkad key indexing algorithm, as well as the main Walkad operations. Then, we analyze the complexity of range queries in Walkad.

A. Key Indexing

We first define the notion of "locality" in a cell-based virtual world as described in Section III-C. Similarly, we say that two cells are *neighbor* cells if: (1) they are adjacent, i.e., they have a side in common, or (2) they are symmetric according to the

axis used in previous split operations. We say that two cell-IDs are *neighbors* when their cell-IDs have a Hamming distance of one, i.e., when they differ only by one bit. By definition, a cell-ID with l significant bits has l neighbor cell-IDs. To illustrate this, we consider an example of a one-dimensional virtual world (Figure 2). We denote the *i*-th cell/cell-ID generated by l splits respectively as C_i^l and k_i^l.

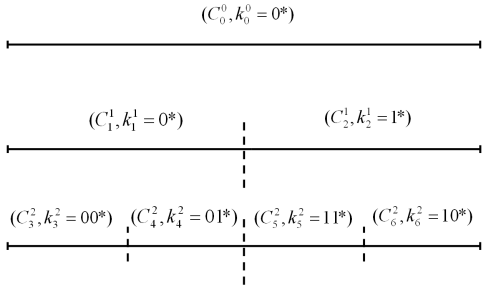


Fig. 2. 1-dimensional virtual world indexed by Walkad

The top portion of Figure 2 shows the initial cell organization. At this stage, there is only one cell, C_0^0, which covers the whole virtual world. The middle part of Figure 2 shows the virtual world configuration after the first split, where two new cells are created, C_1^1 and C_2^1. These two cells are obviously neighbors. The bottom part of Figure 2 shows the result of splitting again both cells, obtaining cells C_3^2, C_4^2, C_5^2 and C_6^2. Let's consider cell C_3^2. Its neighbor cells are cell C_4^2 which is adjacent and cell C_6^2 which is symmetric to C_3^2 according to the long dashed line crossing the middle of the line (indicating a previous split). A generic cell C_i^l generated after l split operations has l neighbor cells in the virtual world.

Walkad organizes the cell-IDs in a *reverse binary trie* to associate neighbor cell-IDs to neighbor cells. In a reverse binary trie, the nodes of each level of the trie are labeled with the Gray Code [4] (Section III-B).

We now explain how we organize the cell-IDs in a reverse binary trie considering the example of Figure 2. For convention, cell C_0^0 is assigned cell-ID $k_0^0 = 0*$. When C_0^0 splits, two neighbor sub-cells are created. We generate the corresponding cell-IDs by taking $k_0^0 = 0*$, and setting the least significant bit respectively to 0 and 1. We thus obtain two cell-IDs, $k_1^1 = 0*$ and $k_2^1 = 1*$, which have a Hamming distance of one. The intuition is that to build a reverse binary trie in some cases we need to reverse the added bits (i.e., add a 1 to the left cell-ID). Figure 2 shows that splitting cell C_2^1 required setting the least significant bit to 0 for $k_6^2 = 10*$ and to 1 for $k_5^2 = 11*$ to guarantee that cells C_3^2 and C_6^2, which are neighbors, are assigned neighbor cell-IDs as well. We see that the code generated at level 2 of the trie is the 2-bit Gray Code.

The generalization to the multi-dimensional case is straightforward, as the cell split mechanism is applied independently to each dimension of the virtual world [15].

The Walkad indexing algorithm as described above generates a distribution of cell-IDs within the keyspace that follows the shape of the trie. Therefore, an unbalanced trie will result in an unbalanced distribution of cell-IDs and so of load among peers. In order to restore the uniform distribution of the cell-IDs, we divide the world in regions (as in Second Life), and we allocate to each region a *region-ID*. Then, we perform a XOR operation between the cell-IDs and the region-ID. In this way, the Hamming distance property defined among cell-IDs of the same region is maintained and load balancing is achieved among cell-IDs of different regions.

B. Walkad Operations

Walkad uses only the classic Kademlia operations without requiring any changes to the routing algorithm. Moreover, Walkad nodes perform some additional operations related to the construction and management of the virtual world. We now describe these operations.

A generic cell-ID in Walkad is coded into a 160-bit Kademlia key by setting the non prefix bits to 0, e.g., obtaining $100...0$ for cell-ID $1*$. We call a *coordinator* the peer responsible for a cell. The coordinator for a cell C_i^l indexed by the cell-ID k_i^l is the XOR closest peer to k_i^l as defined by Kademlia. For each (cell/cell-ID) pair there are R coordinators, i.e., each object and consequently cell is replicated at R peers. This replication factor allows Walkad to sustain churn [16].

Initialization - A Walkad bootstrap node identifies the R coordinators of the initial cell C_0^0 by performing a Kademlia lookup for cell-ID k_0^0. At this time, these peers are responsible for the entire virtual world. They store all objects created in C_0^0 and answer all range queries.

Split - When a cell C_i^l is split in cells C_{2i+1}^{l+1} and C_{2i+2}^{l+1}, its coordinators do the following operations: (1) select the coordinators for C_{2i+1}^{l+1} and C_{2i+2}^{l+1} by performing a Kademlia lookup for cell-IDs k_{2i+1}^{l+1} and k_{2i+2}^{l+1}, (2) transfer to the coordinators of C_{2i+1}^{l+1} and C_{2i+2}^{l+1} the list of C_i^l neighbor cell/cell-IDs , (3) distribute the virtual objects currently located in C_i^l to the coordinators of cell C_{2i+1}^{l+1} and C_{2i+2}^{l+1} according to the object coordinates. Note that a merge operation can be done similarly.

Coordinator Selection - A peer selected to be a coordinator for a cell C_i^l with cell-ID k_i^l does the following operations: (1) derive the l cell-IDs with Hamming distance equal to 1 from k_i^l, (2) compare each of these cell-IDs with the list of cell-IDs received during the split in order to identify the existing neighbor cell-IDs, (3) perform a Kademlia lookup for each existing neighbor cell-ID to populate its k-buckets with the routing information towards the neighbor coordinators, (4) inform the coordinators of the neighbor cells that a new cell was created.

Range Query - We suppose that a peer P submitting a range query already knows the R coordinators of the cell where its own avatar is located. P sends the range query to one of these coordinators. The coordinator answers the query or a portion of it according to the information it has about the neighbor cells. Then, it sends back to P the information it may know, i.e., routing information towards the coordinators for the cells intersecting with the query's range. In case a coordinator has not a complete view of the entire range, it forwards the query to the coordinators it knows that manage

the closest cells to this range. Intuitively, these coordinators will have a more detailed view of this portion of the virtual world. This procedure is done iteratively until the range is completely covered. Finally, P directly contacts the set of coordinators responsible of the query's range to retrieve the information about the virtual objects located in this portion of the virtual world.

C. Range Queries Cost Analysis

We now analyze the cost of range queries in Walkad in terms of number of routing hops. Let k be the size of a k-bucket and R the number of coordinators per cell. N is the number of active peers and m the number of cells composing a region of the virtual world. We assume $N >> (R*m)$ such that each cell-ID is stored at R different coordinators.

In case of a uniform distribution of cells within the world, all the leaves of the trie are at the same level l. In this case, every cell has l neighbor cells associated to l different cell-IDs. Therefore, local queries are answered in a single routing hop, while non local queries require $log(m)$ routing hops.

Figure 4 shows an example of a two dimensional virtual world composed by a uniform distribution of cells. The arrows indicate neighbor (cell/cell-ID) pairs as well as routing information, e.g., the coordinators for cell-ID $001*$ keeps routing links towards the coordinators for cell-IDs $000*$, $011*$ and $101*$. We clearly see that all local queries require a single routing hop. Let's consider a non-local query, e.g., an avatar P located in the cell identified by the cell-ID $001*$ that wants to teleport to the cell identified by cell-ID $110*$. In this case, the query goes through the coordinators of cell-IDs $101*$, $111*$ and $110*$ to be solved, i.e., it requires $log(m)$ routing hops.

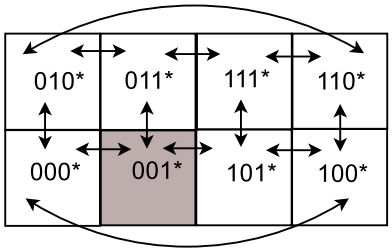
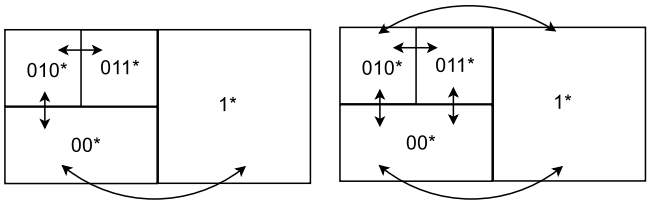


Fig. 4. Uniform cell division of a 2-D virtual world

A skewed distribution of cells within the virtual world results in an *unbalanced* trie. In this case, a cell which is close to the root of the trie may have more neighbor cells than neighbor cell-IDs. Let's consider two cell-IDs k_i^l and k_j^f respectively located at level l and f of the trie with $f < l$. In this case, there are $(R * 2^{(l-f)})$ coordinators "colliding" on the same k-bucket of the coordinators for cell-ID k_j^f. If $k < (R * 2^{(l-f)})$, the coordinators of k_j^f will select only a subset of the $2^{(l-f)}$ neighbor cells to maintain a direct route towards their coordinators.

Figure 5 shows an example of a two dimensional virtual world composed by a skewed distribution of cells. The arrows indicate routing links among the coordinators, respectively in the case of $k = R$ (Figure 5(a)) and $k = 2R$ (Figure 5(b)). We can see that the neighbor cells for the cell identified by cell-ID $1*$ are the cells indexed by $011*$ and $00*$, which are adjacent, and $010*$ which is symmetric to the vertical split. However, by definition, cell-ID $1*$ has only a single neighbor cell-ID that is $0*$. Therefore, if $k = R$ the coordinators for cell-ID $1*$ keep a single link towards the coordinators for cell-ID $00*$, whereas if $k = 2R$ these coordinators keep two links towards the coordinators for respectively cell-ID $010*$ and $00*$.



(a) k=R (b) k=2R

Fig. 5. Skewed cell division of a 2-D virtual world

We now derive an expression for the number of routing hops in Walkad in case of an unbalanced trie. Let's consider again two generic cell-IDs k_i^l and k_j^f located at level l and f of the trie. In case k_i^l and k_j^f are neighbor cell-IDs, a query from k_i^l to k_j^f or vice-versa is local and require $\frac{(|l-f|)}{\lfloor log(\frac{k}{R}) \rfloor}$ routing hops. In case k_i^l and k_j^f are not neighbor cell-IDs, the query is non-local and require an intermediate step at a cell-ID k_k^c that has routing information towards the destination cell-ID. Therefore, the number of routing hops is equal to $\frac{(|l-c|+|c-f|)}{\lfloor log(\frac{k}{R}) \rfloor}$. Note that in the worst case the number of routing hops is $O(m)$ for both local and non-local queries.

In conclusion, the number of routing hops in Walkad varies between $O(1)$ and $O(m)$ according to the skewness of the cell distribution and the type of range query. Intuitively, prefix expansion [14] can be used to reduce the number of routing hops in case of a very unbalanced trie. For comparison, the simple approach (Section III-C) and P-Grid [1] require respectively $log(N) * log(m)$ and $log(N)$ routing hops for both local and non-local queries.

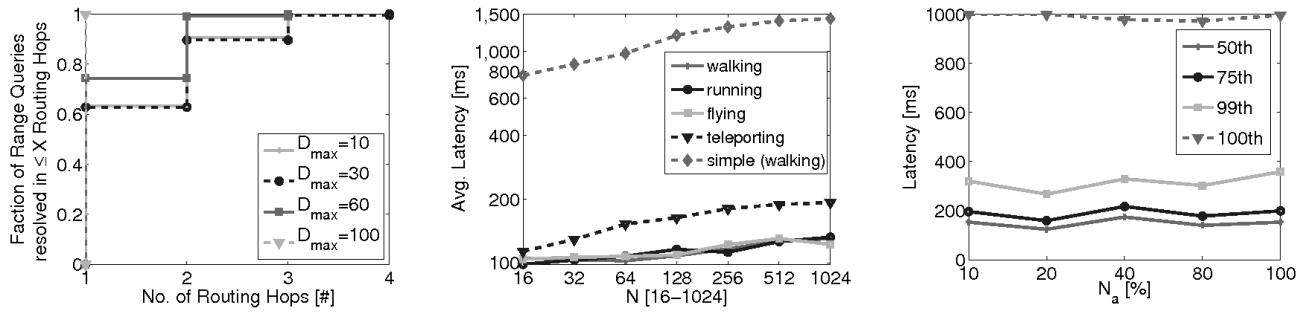
V. Experimental Evaluation

We now perform a preliminary evaluation of Walkad applied to Second Life. We focus on routing hops and latency to answer range queries, and on load balancing properties[1]. For comparison, we also present some results obtained with the simple approach described in Section III-C.

We deploy up to 1024 peers (i.e., avatars) on a local cluster, and we use Modelnet [13] to emulate wide-area latencies and bandwidths. We use a synthetic Internet topology generated by Inet [18]. We use a classic Kademlia setup with k-bucket size $k = 20$ [7], and $R = 10$ [16].

We construct a realistic virtual world using object traces from five popular Second Life regions [17]. The number of

[1]The analysis of the cost for the retrieval of the objects, as well as the virtual world consistency and persistency under churn is left for future work.



(a) $D_{max} = [10; 30; 60; 100]$; $N = 1024$; Avatar=$[Walk]$

(b) $D_{max} = 10$; $N = [16 - 1024]$; Avatar=$[Walk; Run; Fly; Teleport]$

(c) $D_{max} = 10$; $N = 1024$; $N_a = [10; 20; 40; 80; 100]\%$; Avatar=$[Walk]$

Fig. 3. Routing Hops and Latency ; 5 Second Life regions

objects per region varies between 70 and 350 objects. A bootstrap Walkad node computes the virtual world division in cells, and informs all coordinators of their role.

When an avatar moves out of a cell, it generates a range query that spans the cell containing the new avatar coordinates, i.e., we approximate the visibility area of an avatar to the cell where its coordinates are contained. We use synthetic traces for avatar movements generated via the Random Waypoint Mobility model [8] with different speeds to simulate avatar walking (1 m/s), running (3 m/s), flying (10 m/s) and teleporting (100 m/s). The avatar traces lasts for one hour. We use synthetic traces for avatar movements and not real avatar traces [17] in order to define bounds of Walkad performance under controlled avatar behaviors.

A. Routing Hops and Latency

We analyze first the number of routing hops as a function of D_{max}, the maximum number of objects per cell. By varying D_{max} we simulate different divisions of the virtual world. We consider a Walkad network composed by 1024 peers and a single avatar walking in the virtual world. This means that all range queries are local and the "load" in terms of concurrent number of queries in the network is small.

Figure 3(a) shows that 90% of the local queries in Walkad require only one or two routing hops to be solved. This percentage becomes even larger as we increase D_{max}, e.g., when $D_{max} = 100$ all local queries are answered in a single hop with no exception. In fact, increasing the number of objects per cell systematically results in a more simple cell organization of the virtual world. However, if we focus on the curves obtained with $D_{max} = [30; 10]$ we notice that the number of routing hops is comparable or even smaller for $D_{max} = 10$ which seems contradictory. The cause of this phenomenon is that for $D_{max} = 10$, the condition $N >> (R * m)$ is not verified. Therefore, peers are coordinators of multiple cells, and the Walkad indexing algorithm tends to aggregate closeby cells at the same peer. The side effect is a reduction in the number of effective routing hops especially for local queries.

We now evaluate the *latency*, i.e., the time required to answer range queries, as a function of the network size N and type of range query (see Figure 3(b)). To generate different range queries, we consider a single avatar walking, running, flying and teleporting in the virtual world. Figure 3(b) plots also the latency values for a simple approach (see Section III-C). For the simple approach, we only consider the case of an avatar walking as the type of range query does not impact the way routing performs.

Figure 3(b) shows that range queries generated by an avatar walking, running or flying are all resolved in about the same time, i.e., $100 - 130\ ms$ in average. In fact, all these movements generate local queries. Conversely, non local queries generated by an avatar teleporting in the virtual world require about twice the time, e.g., up to $200\ ms$. For comparison, the average latency for the simple approach is between $800\ ms$ and $1500\ ms$, i.e., 8 times larger than in Walkad. Figure 3(b) shows also that the overall latency only slightly increases with the size of the network N. In fact, the number of routing hops to solve range queries in Walkad depends on the size of the virtual world rather than on the size of the network (see Section IV-C). However, when the network is very small, peers are coordinators of multiple cells, thus reducing the number of routing hops and latency as well.

We now evaluate the impact of load, i.e., the concurrent number of range queries, on the latency. We set $D_{max} = 10$, $N = 1024$, and we vary N_a, i.e., the fraction of peers whose associated avatar walks in the virtual world. We consider only the case of local queries. Figure 3(c) shows different percentiles of the distribution of latency values as a function of load. We observe that Walkad is very robust to load, as the overall latency is not impacted by a large value of N_a. The fluctuations we observe for each curve depend on the different overlay organizations in the experiments. Figure 3(c) shows another interesting result: 75% of the latency values are smaller than $200\ ms$. This confirms that Walkad is very efficient in managing local queries as already observed in Figure 3(a) and 3(b). Only 1% of the latency values are significantly higher, and reach a maximum of $1000\ ms$. These values occur when local queries involve cells at different levels in the trie. However, even the largest latency value we observe in Walkad is significantly smaller than the average latency value we observe with the simple approach (see Figure 3(b)).

B. Load Balancing

We now analyze the Walkad load balancing properties, i.e., how the responsibility of the virtual world is distributed among peers. We choose $D_{max} = 10$. Figure 6 shows several percentiles of the distribution of the fraction of cells per peer as a function of N. Load balancing is achieved when each peer manages the same fraction of the cell-IDs, i.e., when all percentiles of the distribution for a given N assume the same value. Note that this is always the case for the simple approach.

Figure 6 shows that increasing the size of the Walkad network the load is rapidly distributed fairly among the active peers. For example, when N is larger than 256, for about 90% of the peers the difference in the fraction of cell-IDs they manage is smaller than 1%. The remaining 10% of the peers are responsible of a larger fraction of cell-IDs. This is due to the fact that we are considering a small virtual world composed by only five Second Life regions. Therefore, the global cell-ID organization is still impacted by the specific cell-ID organization within each region.

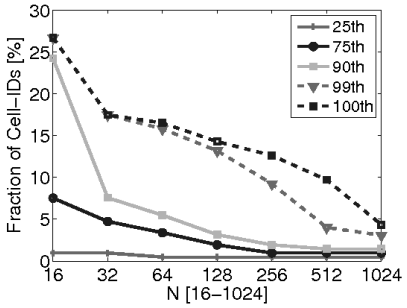


Fig. 6. Some percentiles of the distribution of the fraction of cell-IDs per peer; Five Second Life regions ; $D_{max} = 10$; $N = [16 - 1024]$

We also evaluate the distribution of the virtual world responsibilities as a function of the virtual world size, i.e., the number of Second Life regions indexed with Walkad (Figure 7). To do so, we set $D_{max} = 10$, $N = 1024$ and we vary the number of Second Life regions betwen 1 and 100 using the dataset in [17]. Figure 7 shows that when the virtual world is small, i.e., composed by less than five regions, we cannot identify a general trend for the curves. In fact, the different object compositions strongly impact the general distribution of the cell-IDs. By focusing on a number of regions ≥ 10, we observe that the division of the virtual world responsibilities becomes more and more uniform as the size of the virtual world increases. For example, in a virtual world composed by 100 Second Life regions only 1% of the peers store a larger portion of the virtual world, which consists in worst case of only 1% of the entire virtual world.

VI. Conclusions and Future Work

This paper has presented *Walkad*, a Kademlia-based Peer-to-Peer (P2P) network designed to manage range queries in virtual worlds. Walkad leverages on the Kademlia routing protocol and on an indexing algorithm based on a reverse binary trie. This indexing algorithm is designed to favor local queries which are the most frequent in virtual worlds [6]. We evaluate a prototype of Walkad via network emulation

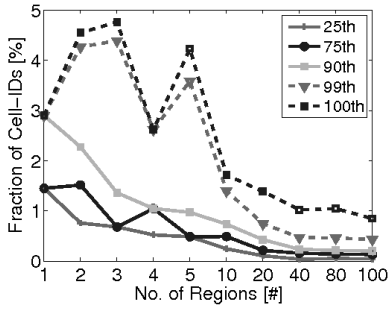


Fig. 7. Some percentiles of the distribution of the fraction of cell-IDs per peer ; $D_{max} = 10$; $N = 1024$; No. of regions=$[1 - 100]$

with up to 1024 peers. We simulate a realistic virtual world using object traces from Second Life. Our results show that Walkad efficiently handles range queries generated by avatars moving in a virtual world. Moreover, the management load of the virtual world is fairly distributed among peers as both the network and virtual world grow. As future work, we will extend the evaluation of Walkad and build a complete P2P architecture for virtual worlds.

Acknowledgements

The authors would like to thank Fabio Picconi for his precious help with the Modelnet setup and for his insightful comments and suggestions. This research was supported by the EU FP7 "Nano Data Centers" project.

References

[1] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: a Self-Organizing Structured P2P System. *SIGMOD Rec.*, 32(3):29–33, 2003.
[2] A. Bharambe, J. Pang, and S. Seshan. Colyseus: A Distributed Architecture for Online Multiplayer Games. San Jose, CA, May 2006.
[3] M. Claypool and K. Claypool. Latency and Player Actions in Online Games. *Commun. ACM*, 49(11):40–45, 2006.
[4] F. Gray. Pulse Code Communication. U.S. Patent 2,632,05, March 1953.
[5] S.-Y. Hu, J.-F. Chen, and T.-H. Chen. VON: A Scalable Peer-to-Peer Network for Virtual Environments. *Network, IEEE*, 20(4):22–31, 2006.
[6] C.-A. La and P. Michiardi. Characterizing User Mobility in Second Life. In *WOSN*, Seattle, USA, August 2008.
[7] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *IPTPS*, Cambridge, MA, USA, March 2002.
[8] Mobility Models. http://ica1www.epfl.ch/RandomTrip/.
[9] NIST. Secure hash standard. Federal Information Processing Standard, FIPS-180-1, April 1995.
[10] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief Announcement: Prefix Hash Tree. In *PODC*, page 368, 2004.
[11] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, London, UK, 2001.
[12] M. Steiner, T. En Najjary, and E. W. Biersack. A Global View of KAD. In *IMC*, San Diego,CA,USA, Oct 2007.
[13] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kosti, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *OSDI*, Boston, MA, USA, December 2002.
[14] G. Varghese. *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufman, 2006.
[15] M. Varvello. A Walkable Kademlia Network for Virtual Worlds. Technical Report CR-PRL-2009-01-0002, Thomson, 2009.
[16] M. Varvello, C. Diot, and E. Biersack. P2P Second Life: experimental validation using Kad. In *Infocom*, Rio De Janeiro, Brazil, April 2009.
[17] M. Varvello, F. Picconi, C. Diot, and E. Biersack. Is There Life in Second Life? In *Conext*, Madrid, Spain, Dec. 2008.
[18] J. Winick and S. Jamin. Inet-3.0: Internet Topology Generator. Technical Report CSE-TR-456-02, University of Michigan, 2002.