TRACE: Generating Traces from Mobility Models for Distributed Virtual Environments

Emanuele Carlini[1(✉)], Alessandro Lulli[1,2], and Laura Ricci[1,2]

[1] Istituto di Scienza e Tecnologie dell'Informazione (ISTI), Consiglio Nazionale delle Ricerche (CNR), Rome, Italy
{emanuele.carlini,alessandro.lulli,laura.ricci}@isti.cnr.it, {lulli,ricci}@di.unipi.it
[2] Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Abstract. The development and evaluation of a proper mobility model is an essential feature to evaluate a system that manages a virtual world. In distributed virtual environments, this is also more important because each avatar requires a consistent view of the world that usually is splitted on multiple machines. Several models have been proposed in the literature to describe avatars' mobility, but a single environment supporting the generation of traces from different models to enable a simple comparison of them is still lacking. In this work we present a tool that implements popular mobility models and supports the generation of traces generated by them. This may help developers to easily validate their systems using several mobility models. Our tool provides a unified format to describe the traces, enables the generation of traces for thousands of avatars and defines an API enabling the integration of additional models.

1 Introduction

A common trait of many virtual environments is the fact that the behaviour of avatars depends mostly on what happen in their immediate surroundings. This fact, referred to as *locality*, has been widely exploited to optimize the management of the virtual environments operations at system level. In the last decade many approaches have been proposed to foster the transition of virtual environments from client-server to distributed applications, referred to as Distributed Virtual Environments (DVEs) [14]. Most DVEs architectures heavily rely on the concept of locality to split the virtual world and distribute each part of the world to different machines. In this scenario, each avatar needs to reconstruct its local view of the virtual world by interacting with the host of the nearby avatars. Also, the machines that handle the world need to cooperate to provide a consistent view of the virtual environment. This kind of approach holds for approaches based on unstructured [9,15] and structured [2,10] peer-to-peer technologies, as well as more centralized technologies like cloud computing [13]. For example, in Voronoi-based DVE approaches, the world is assigned to the hosts of the avatars according to a tessellation of the virtual world, which depends on the position of the avatars [9,15]. When avatars move, the assignments change accordingly.

F. Desprez et al. (Eds.): Euro-Par 2016 Workshops, LNCS 10104, pp. 272–283, 2017.
DOI: 10.1007/978-3-319-58943-5_22

Therefore, the description of how avatars move in a DVE is essential to design, validate and compare different DVE architectures.

Performing the above actions in a real setting is an expensive and difficult task, since it requires to organize the setting on multiple machines and involve multiple persons each moving an avatar in the virtual world. Therefore, the solution adopted by the researchers is to simulate avatars' movement in order to validate the specific DVE architecture. Normally, two common ways are considered to simulate avatars' movements, i.e. traces taken from an instance of a real virtual environment application, or synthetic traces generated from mobility models. Real traces are usually a good mean of validation, as they represent what is the actual behaviour of avatars in the virtual worlds [7]. However, they suffer from the same issues of testing a DVE in a real setting: it is difficult to collect real traces and, in particular, they may be not suitable to validate the system on an extreme or specific scenario. Synthetic traces are usually not extremely precise in simulating avatars' movement but present many clear advantages in contrast with real traces [17], such as (i) *scalability*, to stress the DVE support in limit situations, (ii) *reproducibility*, as synthetic traces can be reused on different systems in order to have a common ground for comparison.

As a consequence the best way to evaluate a DVE architecture is to exploit a combination of real and synthetic traces. In this paper we focus on the latter, and we provide the description of a software library we developed to generate synthetic traces using mobility models. Mostly, all the approaches generate synthetic traces with custom specifically developed solutions. This has several drawbacks: (i) it is hard to compare different systems on the same scenario, as exact details on how the traces are generated are usually not released; (ii) researchers spend time to code and test traces and trace generators; (iii) there are no clear reference mobility models that are targeted by the DVE community; (iv) it is difficult to reuse traces because usually they are encoded using specific formats. In order to overcome these drawbacks, we developed TRACE, a software library that generate avatar positions according to mobility models. Initially, we have used TRACE for internal research (such as in [4,5]) but eventually we have made it available for the whole DVE community. TRACE: (i) provides means to generate traces for a wide variety of DVE-based mobility models; (ii) allows to export and reload traces for later uses and comparisons; (iii) works in memory (Java) and with a separate visual tool; (iv) is fully configurable, both on mobility models and the map; (v) is designed for an easy integration of new and personalized mobility models; (vi) uses a unified format in all the mobility models used.

In this paper we present the main features and characteristics of the tool, unravelling important under-the-hood decisions that makes it easy and practical to use. We provide an overview of the tool's API. We show an example of integration of TRACE in an existing DVE support, showing how different traces can be used to test and validate the support.

2 Related Works

Although mobility models have been extensively used, in the last years, in several applicative domains, and in particular for DVE, no tool able to generate multiple models on demand currently exists. A mobility model is usually implemented and used in isolation. For instance, mobility models are one of the most important factors to validate gaming overlays. VON [9], Mopar [18], pSense [16] and Gross *et al.* [6] is only a brief list of the most popular P2P game overlays that use just random based walker models or random walk between hotspots. Those models are popular thanks to their simplicity: a random walk model only requires a few lines of code and the community generally accepts it as a model able to describe several gaming scenarios. However, different and more complex models exist, that are able describe specific scenarios more precisely. BlueBanana [12] is inspired by the virtual world defined by Second Life. In this world, players gather around a set of hotspots, which usually correspond to towns, or, in general, to points of interest of the virtual world. Using the Least Action Planning trip (LAPT) [11] the avatars select hotspots in close proximity with higher probability. When an avatar visits an hotspot, it stays there for a time drawn from a truncated-Pareto distribution and then moves to another hotspot. In RPGM [8], each player belongs to a group and it moves by following the movement of its group, in order to model the players habit to gather in teams. Similar to the previous, a subset of the authors of this work defined a mobility model called WOW [3]. wow considers also hotspots where players are placed at the start of the game and spawn after death. This model takes into account the team-oriented nature of the scenario, where moving in group is encouraged by the game semantics. However, an avatar may decide to move alone by itself, for instance to take the enemy by surprise. All the above models have been defined and used on specific scenarios. However we think it is important to unify how the models are generated and how they are used.

An interesting approach is that of the game trace archive [7], which collects different real traces with the main aim of defining a common format to collect and record game traces so that these can be easily used. In April 2016 the archive includes 12 traces. Even if this environment presents some similarity with our work, it does not include a mobility model generator.

Triebel *et al.* [17] study both the mobility of avatars and their interactions. They compare the movement of avatars guided by mobility models versus movements generated by artificial intelligence techniques. Although the latter provides better results, using simple mobility model such as random way point and a random model based on hot spots, give close results, in particular the one based on hot spots. Artificial intelligence movements take into account also the context of the game, must be built specifically for each game and they base their movement on the mobility models. For all these reasons, although specific solutions may get marginal improvements on the validation, we think that the generality of the mobility models is an important way to validate games.

3 The Tool

TRACE is a open-source Java library[1] specifically designed for the experimentation of DVEs that generate traces from mobility models, unifies the output of the models and provides an API to enrich TRACE with additional mobility model implementations. In the following of this section we describe the main characteristics of TRACE, its architecture and the functionalities provided.

TRACE has been primarily designed with the idea of focusing on experimentation and evaluation of distributed virtual environments, therefore most of the terminology used in this section refers to such field. However, we believe that TRACE is flexible enough to be used in other contexts in which a number of entities move across a (virtual) area. In DVEs, avatars are the digital agents of the users in the virtual environment and are associated with a position in the virtual world. They are the moving unit considered in TRACE. Other than avatars, TRACE gives the possibility to specify *static* entities, namely passive objects and hotspots. Passive objects are entities that have a state and can be interacted by avatars (e.g. doors), but unlike avatars are not controlled by an human user. The hotspots are those areas of the virtual environment corresponding to places of interest and where usually is present an higher density of passive and active entities.

In a nutshell, TRACE (Fig. 1 provides an high-level overview of TRACE and Table 1 provides a list of the most important classes of TRACE) takes in input the definition of the virtual environment and the mobility model and outputs

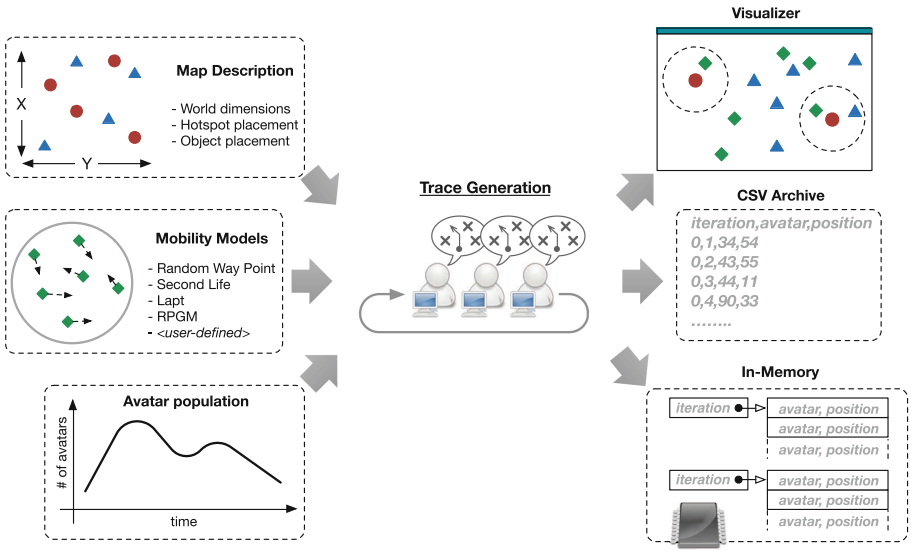


Fig. 1. TRACE overview: inputs and outputs

[1] Publicly available at: https://github.com/hpclab/trace.

Table 1. Notable interfaces and classes of TRACE

`AMobilityModel`	Abstract class to define mobility models. The core method is `move` in which the movement of the generic avatar is defined according to the iteration
`AStaticPlacement`	Abstract class to define placement function for static entities, such hotspots and passive objects. This class is called once during the initialization of the virtual environment
`IAvatarNumberFunction`	Interface to define the amount of avatars at any iteration. It is called by the engine before the computation of each iteration to adjust the avatar population

the resulting traces. All the inputs are defined in a *configuration* file composed by a list of key-value tuples that contains all the necessary information for the generation of the traces. Map description defines the rectangular area of the virtual environment, including its size, hotspots and how to assign the position of the passive objects. Note, both hotspots and passive objects can influence the movements of the avatars, but they are not essential for the generation of the traces. Nevertheless, the placements of hotspots and passive objects is totally configurable via the class `AStaticPlacement`, which can be extended to place static entities according to a user-defined function (e.g. randomly across the area of the environment or with high probability placement in hotspots) or by providing a list of points if there is the need to simulate a specific virtual environment. Avatar number represents the amount of avatars in the virtual environment. Frequently, DVE frameworks exploits peer-to-peer protocols to assure scalability and cost effectiveness. In order to validate a framework is therefore necessary to see how it behaves in scaling up and down, according to the typical churn that characterizes DVE workloads. TRACE gives the opportunity to model the churn with the interface `IAvatarNumberFunction`, which allows to define the number of avatars at any iteration. Note that TRACE also allows for a fine grained control of the churn, as it is possible to understand which avatars left (or entered) because TRACE keeps avatar id consistent across iterations.

The definition of a mobility model is one of the core parts of TRACE, and can be done by extending the `AMobilityModel` interface. A mobility model defines how a generic avatar shall move within the boundaries of the virtual environment, and this behaviour is then replicated for all avatars in the DVE. TRACE considers discrete time iterations, and at each iteration avatars move according to the mobility model specified. In particular, during iteration t avatars move independently without the knowledge of each other position at iteration t; however they can have a read-only access to positions of avatars at iteration $t-1$. A common issue when generalizing the generation of traces is that any mobility model can have its own configuration with specific parameter. TRACE resolves this issue by allowing a free definition of the parameters inside the configuration file, leaving to the developer the responsibility of matching the correct parameter within the

implementation of the mobility model. For example, the Blue Banana mobility model (whose implementation is described in detail in Sect. 4) is heavily focused on hotspots and therefore define specific properties such as the probability for an avatar of being inside the area of an hotspot.

According to the configuration file, TRACE creates the mobility traces, iteration by iteration, completing each avatar movements before dealing with the next iteration. The `MapVirtualEnvironment` object stores all the information about movements of the avatar, hotspots and passive objects. This class can be accessed in a read-only fashion to be used right away when the generation of the traces is done contextually to the experimentation. Apart from such in-memory data structure, TRACE provides two additional and optional output features, namely *logfile archive* and *visualizer*. These two features can be active at the same time.

With logfile activated, a dump of `VEMap` is saved on disk in a format that represents the movement of all the entities in the virtual environment. Regardless of the model used, TRACE builds a compressed archive consisting of the following files: *(i) configuration*, which contains all the variables to replicate the scenario; *(ii) avatars*, which stores the movement of the avatars; *(iii) hotspots*, which stores the position of the hotspots; *(iv) objects*, which stores the position of the passive objects in the game; *(v) bandwith*, which provides statistics regarding the number of objects in the AoI of each avatar; *(vi) aoiStat*, which provides statistics regarding the number of avatars in the AoI of each avatar. The avatars file contains a snapshot of the position of all the avatars in each time step in a CSV format containing the following values: time step, unique avatar identifier, position of the avatar in the map as a couple (x, y). The resulting file can be loaded at a later time to be used in different experimental evaluation. With the visualizer activated, TRACE provides a graphical representation of the avatars moving across the map. Although this option may slow down the generation of the traces, it results very useful to tune the parameters of a mobility model in order to obtain specific behaviour from avatars.

TRACE comes bundled with the following mobility models already implemented and ready to be used[2]: (i) Random Way Point [1], (ii) RandomWalk, (iii) Lapt [11], and (iv) Blue Banana [12]. In order to provide an hand-on overview on the utilization of TRACE, the next section describes in details BlueBanana and how it has been implemented within TRACE.

4 Case Study: Blue Banana

Avatars move on the map according to realistic mobility traces that have been computed according to the mobility model presented by Legtchenko et al. [12], which simulates avatars movement in a commercial MMOG, Second Life[3]. We provided a preliminary implementation of this mobility model, as well as a comparison with other mobility models in [3], In the model, avatars gather around

[2] More mobility models are under development and will be added in the future.
[3] http://secondlife.com/.

Algorithm 1. *AMobilityModel.move()* implementation: BlueBanana

```
   Input  : map: a Map representing the virtual world
            t: the current time
            avatarList: the avatars position at time t − 1
   Output: the position of the avatars at time t
 1 List next = avatarList
 2 forall Avatar a∈avatarList do
 3 |   State nextState = markovChain.getNextState(a, markovChain.getState(a))
 4 |   if nextState = E then
 5 |   |   Point current = a.getPosition()
 6 |   |   next(a) = current.explore()
 7 |   else if nextState = T then
 8 |   |   Point t = map.getRandomPoint()
 9 |   |   Point current = a.getPosition()
10 |   |   next(a) = current.moveToward(t)
11 |   else
12 |   |   do nothing
13 |   end
14 end
15 return next
```

a set of *hotspots*, which usually correspond to towns, or in general to points of interest in the virtual world. Each hotspot has a circular area characterized by a center and by a radius. Traces generation goes through two phases: *initialization* and *running*.

In the initialization phase, the area of the virtual environment is divided in *hotspot area* and *outland area*. The percentage of the hotspot area is defined by p_{hot} and, consequently $1 - p_{hot}$ represents the outland area. The hotspots are placed randomly in the virtual environment. The number of hotspot is defined by the parameter H_{num}. Their radius is computed such that the total area covered by the hotspots is in accordance to p_{hot}. The parameter p_{den} defines the probability that an avatar would be initially placed in an hotspot, whereas $1 - p_{den}$ defines the probability for an avatar to be initially placed in outland. If the avatar is placed in the outland, its position is chosen uniformly at random on the whole map. Otherwise, an hotspot for the avatar is randomly selected and the avatar is positioned inside the hotspot. The position inside the hotspot is chosen by considering a Zipfian distribution, so to ensure an higher density of players near the center of the hotspot.

The running phase moves the avatars across the virtual environment. The movements are driven by a Markov chain, whose transition probabilities are taken from the original paper [12]. The possible states for an avatar is the following:

- *Halt*(H): the avatar remains in place;
- *Exploration*(E): the avatar explores a specific area. If the avatar is moving inside an hotspot, the new position is chosen according to a power law distribution. Otherwise, the new position is chosen at random;
- *Travelling*(T): the avatar moves straight toward another point in the virtual environment. The new point is chosen in accordance with p_{den}.

Initially every avatar is in state H. At each step t, the model decides the new state according to the probability of moving between states defined in the Markov chain. This mobility model exposes a fair balance between the time spent by avatars in hotspots and outland.

To integrate such model in TRACE the following steps are required:

- *configuration*: it is required to load all the model specific configuration variables such as p_{hot}, H_{num} and p_{den};
- *additional functionalities*: since this model requires a Markov Chain to move the avatars between different states, i.e. (H, E, T), we implemented an utility class to easily know, given a state, which is the next state of the avatar;
- *AMobilityModel*: the core of the model is the implementation of the *AMobilityModel* interface. Specifically, it is required to implement the *move* method where TRACE provides the position of the avatars at time $t-1$ as well as an object describing the virtual environment where is possible to find the position and size of the hotspots and objects. The model must return the position of the avatars at time t. Refer to Algorithm 1 to an example of the method's definition and implementation.

For what concerns the initialization phase, our implementation at time 0 follows the specification of the initialization phase provided in the original paper. During the running phase, we generate a new position for each avatar (Line 2) and the new state of the avatar according to its previous state (Line 3). Based on the next state, we follow the specification of the model for the *Travelling* state (Line 7), *Exploration* state (Line 4) and *Halt* state (Line 11). We collect all the new positions in a list and we return all the new positions (Line 15).

Finally, to use the new implemented model, it is required to modify the configuration of TRACE, giving a name to the new model, for instance "BlueBanana", and providing the package and class name where it is implemented. Next, it is necessary to set, in the configuration, the property "model BlueBanana", as well as all the configuration parameters required by the model. The execution will use the selected model and generate the traces accordingly.

5 Experimental Results

We implemented TRACE in Java and we make the code publicly available[4]. For all the experiments, we considered a virtual environment composed by a squared region with side having 1500 points. Each avatar has a circular AoI, whose

[4] https://github.com/hpclab/trace.

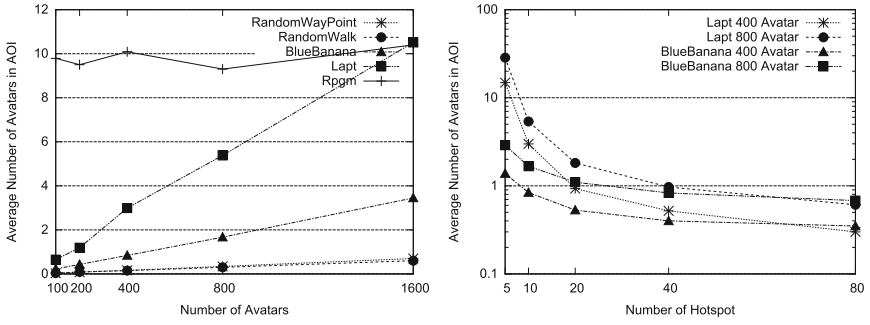
radius is 15 points. Each hotspot has a circular shape, whose radius is 100 points. The simulations ran on a machine equipped with Java 7, 128 Gb of RAM, an AMD Opteron(TM) Processor 6276 with 32 cores @1.4 Ghz. In the following, we present results showing some properties of the models implemented in TRACE. In particular, the avatars' crowding in the virtual world (Sect. 5.1) and the estimated bandwidth consumption to transmit objects of the virtual world (Sect. 5.2). We conclude our experiments with an evaluation of the computational time to generate a mobility model and the scalability of TRACE (Sect. 5.3).

5.1 Evaluating the Crowding Generated

With the terms crowding we refer to the evaluation of the number of avatars present in each avatar's AoI. This metric assesses how much communication is required to keep updated the vision of the avatars with respect to the other players in the game.

Figure 2a shows the average number of avatars in the AoI of each avatar for all the models produced by TRACE. On the X axis is represented the number of avatars present in the virtual world, on the Y axis the average number of avatars in a AOI. We generate for each model a trace having a number of avatars in the range $[100, 1600]$. It is interesting to note that with RPGM we obtain similar results in all the configurations. This result is expected because we configure RPGM in order to keep the number of groups equals to 1/20 of the number of avatars. The two models based on random movements are the ones having the less number of avatars in the AoI. Instead, LAPT is the model having the larger increase of crowding as the number of avatars grows, because all the avatars move only between hotspots. With BLUEBANANA this effect is mitigated because a percentage of the avatars is free to move outside the hotspots.

For what concerns LAPT and BLUEBANANA, the models that take in consideration the hotspots, Fig. 2b shows the impact of the number of hotspots using



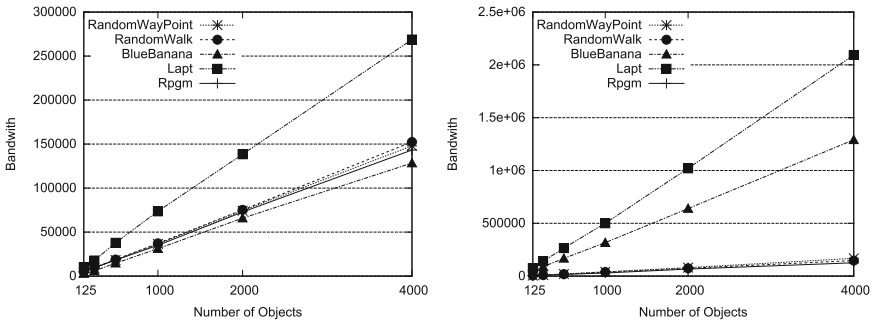
(a) Average number of avatars in AOI (b) Average number of avatars in AOI with different number of hotspots

Fig. 2. Evaluation of Optimizations

the same metric of the previous figure. Note the log scale on the Y axis. When the number of hotspots is kept low, LAPT is, in both the configurations, the model having a larger crowding factor. However, when the number of hotspots increases, the two models behave similarly.

5.2 Evaluating the Bandwidth to Transmit Objects

In this set of experiments, we evaluate the ability of TRACE to model the avatars and objects placement. In particular, when an object enters the AoI of an avatar, a transmission of the object to the avatar is required, resulting in a bandwidth consumption. We measure the total number of objects transmitted when increasing the total number of objects in the virtual world. We test the two methodologies to distribute the objects, respectively the uniformly at random in Fig. 3a, and higher probability in the hotspots in Fig. 3b. For the uniformly at random placement, all the models behave similarly and have a linear increase of the bandwidth with respect to the number of objects. Only LAPT have a little more bandwidth requirement but in the same order of magnitude. Instead, when the objects are more present in the hotspots area, Fig. 3b, the two models, LAPT and BLUEBANANA, as expected, require more bandwidth, because the avatars are more present in the hotspots area.



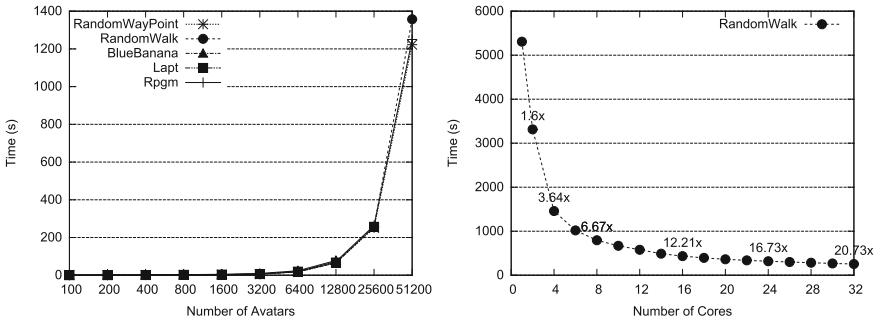
(a) Object placement: uniformly at random

(b) Object placement: higher probability in hotspots

Fig. 3. Evaluation of Bandwidth consumption

5.3 Evaluating the Computational Time and Scalability

Finally, we test the computational time required by TRACE to generate the traces. Figure 4a depicts the computational time when requesting a different number of avatars moving in the virtual world. As expected, the time increases when increasing the number of avatars but it is acceptable also with a large number of avatars, as well as 51 200 avatars. All the mobility models behaves similarly. Due to this, we perform the scalability of TRACE only with the RW model



(a) Computational time with different number of avatars

(b) The scalability of TRACE

Fig. 4. Evaluation of computational time

(we confirm that with other models the shape of the curve is identical). We are able to test our tool with a scenario having a number of cores in the range $[1, 32]$. We obtain a good scalability of TRACE. For instance, with 8 cores we obtain a speed-up of 6.67 to a maximum of 8 and with 12 cores a speed-up of 12.21 to a maximum of 16.

6 Conclusions

This paper described the design and the main features of TRACE, a software toolkit for the generation of mobility traces targeting DVEs. We showed that is possible to implement a mobility model and create personalized mobility traces with few lines of code, by extending the described programming interface. TRACE is able to manage thousands of avatars concurrently, and its experimental evaluation showed its good scalability when multiple cores are used for the generation of traces. In conclusion, we believe that TRACE can be an effective tool to facilitate the evaluation of DVEs frameworks and to implement effective mobility models. In the future, we plan to extend the tool by providing even more options for the generation of traces, as for example an command-line interface to generate traces in a programmatic way.

References

1. Bai, F., Helmy, A.: A Survey of Mobility Models. Wireless Adhoc Networks, vol. 206. University of Southern California, USA (2004)
2. 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. ACM SIGCOMM Comput. Commun. Rev. **38**(4), 389–400 (2008)
3. Carlini, E., Coppola, M., Ricci, L.: Evaluating compass routing based aoi-cast by mogs mobility models. In: Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, pp. 328–335. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2011)

4. Carlini, E., Dazzi, P., Mordacchini, M., Lulli, A., Ricci, L.: Community discovery for interest management in DVEs: a case study. In: Hunold, S., Costan, A., Giménez, D., Iosup, A., Ricci, L., Gómez Requena, M.E., Scarano, V., Varbanescu, A.L., Scott, S.L., Lankes, S., Weidendorfer, J., Alexander, M. (eds.) Euro-Par 2015. LNCS, vol. 9523, pp. 273–285. Springer, Cham (2015). doi:10.1007/978-3-319-27308-2_23
5. Carlini, E., Ricci, L., Coppola, M.: Flexible load distribution for hybrid distributed virtual environments. Futur. Gener. Comput. Syst. **29**(6), 1561–1572 (2013)
6. Gross, C., Lehn, M., Münker, C., Buchmann, A., Steinmetz, R.: Towards a comparative performance evaluation of overlays for networked virtual environments. In: 2011 IEEE International Conference on Peer-to-Peer Computing (P2P), pp. 34–43. IEEE (2011)
7. Guo, Y., Iosup, A.: The game trace archive. In: Proceedings of the 11th Annual Workshop on Network and Systems Support for Games, p. 4. IEEE Press (2012)
8. Hong, X., Gerla, M., Pei, G., Chiang, C.C.: A group mobility model for ad hoc wireless networks. In: Proceedings of the 2nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 53–60. ACM (1999)
9. Hu, S.Y., Chen, H.F., Chen, T.H.: VON: a scalable peer-to-peer network for virtual environments. IEEE Netw. **20**(4), 22–31 (2006)
10. Kavalionak, H., Carlini, E., Ricci, L., Montresor, A., Coppola, M.: Integrating peer-to-peer and cloud computing for massively multiuser online games. Peer-to-Peer Netw. Appl. **8**(2), 301–319 (2015)
11. Lee, K., Hong, S., Kim, S.J., Rhee, I., Chong, S.: Slaw: a new mobility model for human walks. In: INFOCOM 2009, pp. 855–863. IEEE (2009)
12. Legtchenko, S., Monnet, S., Thomas, G.: Blue banana: resilience to avatar mobility in distributed MMOGs. In: 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 171–180. IEEE (2010)
13. Nae, V., Prodan, R., Fahringer, T.: Cost-efficient hosting and load balancing of massively multiplayer online games. In: 2010 11th IEEE/ACM International Conference on Grid Computing (GRID), pp. 9–16. IEEE (2010)
14. Ricci, L., Carlini, E.: Distributed virtual environments: from client server to cloud and P2P architectures. In: 2012 International Conference on High Performance Computing and Simulation (HPCS), pp. 8–17. IEEE (2012)
15. Ricci, L., Carlini, E., Genovali, L., Coppola, M.: AOI-cast by compass routing in delaunay based DVE overlays. In: 2011 International Conference on High Performance Computing and Simulation (HPCS), pp. 135–142. IEEE (2011)
16. Schmieg, A., Stieler, M., Jeckel, S., Kabus, P., Kemme, B., Buchmann, A.: pSense-maintaining a dynamic localized peer-to-peer structure for position based multicast in games. In: Eighth International Conference on Peer-to-Peer Computing P2P 2008, pp. 247–256. IEEE (2008)
17. Triebel, T., Lehn, M., Rehner, R., Guthier, B., Kopf, S., Effelsberg, W.: Generation of synthetic workloads for multiplayer online gaming benchmarks. In: Proceedings of the 11th Annual Workshop on Network and Systems Support for Games, p. 5. IEEE Press (2012)
18. Yu, A.P., Vuong, S.T.: MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In: Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video, pp. 99–104. ACM (2005)