BerlinMODPlayer: A tool for generating GPS coordinate updates

Jan Kristof Nidzwetzki
Ralf Hartmut Güting
Faculty of Mathematics and
Computer Science
FernUniversität Hagen
Hagen, Germany
{jan.nidzwetzki,rhg}@fernuni-hagen.de

July 27, 2016

Abstract

BerlinMOD is a benchmark for spatio-temporal database management systems. The benchmark generates trips of moving vehicles within Berlin. BerlinMODPlayer is a player for such trips. The Player reads the data generated by BerlinMOD and generates a stream of GPS coordinates therefrom. The stream of GPS coordinates can be used to benchmark stream processing systems.

1 Introduction

Stream processing systems are used to process the content of data streams. Such data streams are often generated when moving or changing objects are observed and the observation is recorded progressive. For example, the changing price of a stock, the speed of an engine or the position of a vehicle.

BerlinMOD [6] is a benchmark that generates trips of moving vehicles within Berlin. BerlinMODPlayer is a software that reads the generated trips and creates a stream of GPS coordinates therefrom. That stream is written to a TCP socket. In a real world scenario, such data stream is generated by a fleet of vehicles moving through the streets of a city and sending their GPS coordinates to a central system.

1.1 Basics

A movement beginning at the coordinate $(xstart, ystart)$ and ending at the coordinate $(xend, yend)$ will be called *trip* in this paper. A trip is composed of smaller movements, called *units* (as illustrated in Fig. 1). Trips and units

are describing a movement, starting at time t_0 and ending at time t_1. Inside the interval $[t_0, t_1]$ the trip is called *valid*. Outside this time interval, the trip is *invalid*.

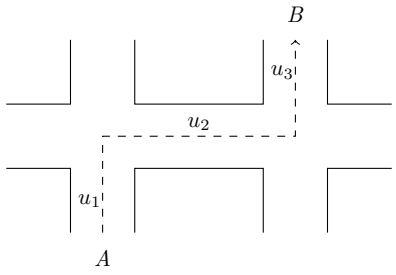


Figure 1: The trip on a map from point A to point B is composed of three units (u_1, u_2, u_3).

1.2 BerlinMod

BerlinMOD is a benchmark for spatio-temporal database management systems. This benchmark includes a generator for moving objects and a set of queries to benchmark the performance of a database management system. In this paper, the data generator is used to create the input data (further called *the input file*) for BerlinMODPlayer.

The basic idea of the data generator of BerlinMOD is to define a set of vehicles and to generate trips on the street network of Berlin. Each vehicle has a *HomeNode*, representing the holder's residence and a *WorkNode*, representing the working place of the holder. Additionally, a set of nodes in the *Neighborhood* around the HomeNode are used. The data generator creates trips from the HomeNode to the WorkNode and trips to the nodes in the neighborhood. For more details about the creation of trips, see [6].

The data generator of BerlinMOD consists of a set of quires for the the extensible database management system Secondo [7]. In section 3 the installation and the configuration of the data generator will be discussed.

1.3 Related work

Testing the performance of a computer system is a very common task. For this purpose, many programs do exist. The software company Oracle offers a *load generator* and a *CSV data generator* [3] to analyze the performance of the *Orcale Event Processing System*. Similar to this work, the data is generated in a first step. Then, the generated CSV data will be send to the software system. However, the generation of spatial data is not supported.

Mockaroo [11] is an online service for generating CSV files with realistic data such as names, ip addresses, email addresses or credit card numbers. Generating GPS coordinates is not supported by mockaroo. *Seagull* [16] is an open source

software for generating protocol traffic, e.g. creating *radius*, *sip* or *h248* requests. Nevertheless, database management systems or generating spatial data is currently not supported.

TPC-C [13] is a benchmark to test transactional databases. The benchmark describes common queries of a wholesale parts supplier and is widely used nowadays. *TPC-H* [4] is an other benchmark for databases, it focuses on the generation of ad-hoc queries and concurrent data modification. *HammerDB* [9] is a practical implementation of the TPC-C and the TPC-H benchmark.

2 BerlinMODPlayer

BerlinMODPlayer is part of the Secondo CVS project and is located in the directory `Tools/Generators/bmod_player`. The following subsections will describe the basic concepts of the BerlinMODPlayer. In addition, the architecture of the software will be discussed. BerlinMODPlayer simulates the movement of vehicles. Consequently, the execution of the software will also referred as *simulation*.

2.1 Command line arguments

The software accepts some arguments, to influence the program behavior and the generated data stream. So, the begin time and the end time of the simulation can be settled, the destination host and port can be specified, to name just a few of the arguments. For a full description of the command line arguments see Table 1.

Parameter	**Name**	**Description**
`-i`	Input file	The file name of the input file.
`-u`	Output URL	The format and the transport for the output. (see Section 2.3).
`-s`	Simulation Mode	The simulation mode that should be used (see Section 2.6).
`-o`	Output file	The output file for the statistics (see Section 2.2).
`-b`	Begin time offset	*Optional:* The time when the simulation should begin. If the parameter is omitted, the simulation begins with the first line of the input file. The time has to be specified in the format: `yyyy-mmm-dd hh:mm:ss`
`-e`	End time offset	*Optional:* The time when the simulation should end. If the parameter is omitted, the simulation ends with the last line of the input file.

Table 1: Command line arguments of BerlinMODPlayer.

Example: Run the BerlinMODPlayer and send the GPS updates to localhost port 10 025 (TCP) in CSV format using the adaptive simulation mode. The statistics are written into the file `statistics.txt`. The file `trips.csv` contains the data generated by BerlinMOD. In Section 3, the creation of such a file is demonstrated.

```
./bmodplayer -i trips.csv -u tcp://localhost/10025 -s adaptive
     -o statistics.txt
```

2.2 Statistical Output

During the program execution, statistical information of the simulation is written to an output file. This file contains a line for every second of the simulation. Every line consists of five fields: (*i*) the first field contains the number of seconds passed since the simulation has begun. (*ii*) The second field contains the amount of read lines from the input file. (*iii*) The third field contains the amount of written lines on the network socket. (*iv*) The fourth field contains the amount of read tuples in the last second. (*v*) The last field contains the amount of sent tuples in the last second.

```
#Sec    Read    Write   Diff read   Diff send
0       0       0       0           0
1       65554   55460   65554       55460
2       135112  124987  69558       69527

[...]

### Total execution time (ms): 11004
### Read: 459974
### Send: 459974
```

The format of the file can easily processed with tools like *gnuplot* [10] (see Section 3.4). The last three lines of the file describe the total execution time of the simulation and the total amount of read and written coordinates.

2.3 The GPS coordinate data stream

BerlinMODPlayer generates a stream of GPS coordinates. The output URL determines the *output transport* and the *output format*. Currently, two different output formats and transports are supported: (*i*) the data can be send in the CSV format (*Comma-separated values*) via a TCP socket or (*ii*) the data can be send in the JSON format (*JavaScript Object Notation*) [5] via HTTP requests.

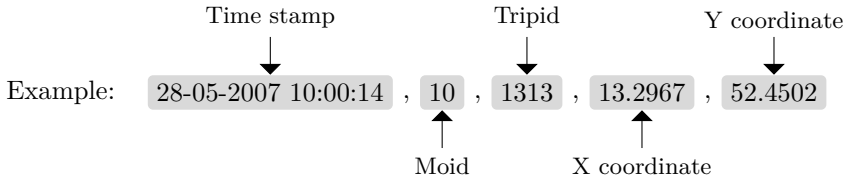
Both formats and transports are discussed in the next sections. Table 2 describes the structure of the supported output URLs.

Output URL	Description
`tcp://myhostname/myport`	Send the output in CSV format via TCP to the host *myhostname* on port *myport*.
`http://myhostname/document`	Send the output in JSON format via HTTP-Requests to the URL *http://myhostname/document*.

Table 2: Supported output URLs.

2.3.1 CSV format and TCP output

Every GPS coordinate is represented as a line with five fields: (*i*) The time stamp when the coordinate is measured, (*ii*) the object, i.e. the vehicle, who has measured the coordinate, (*iii*) an id for the trip and (*iv*) the X and (*v*) the Y coordinate.



The generated coordinate stream consists of lines in the format described above, ordered by the time stamp of the lines. For example:

```
28-05-2007 10:00:14,10,1313,13.2967,52.4502
28-05-2007 10:00:15,22,3107,13.2178,52.5086
28-05-2007 10:00:15,112,16315,13.2063,52.5424
28-05-2007 10:00:15,6,751,13.322,52.4626
```

After the last coordinate update is send, the ascii character *end of transmission* (EOT) will be send, to indicate the end of the data stream. Afterwards, the tcp connection will be closed.

2.3.2 JSON format and HTTP output

Alternatively, the GPS coordinate stream can send via HTTP PUT-Requests. For that kind of output, the GPS coordinates are converted into the JSON format. One coordinate in JSON format looks like:

```
{
   "Id":"100000000000000",
   "Position":{
     "date":"2015-03-06T23:20:01.000",
      "x":13.141600000000002,
```

```
      "y":13.141600000000002
   }
}
```

Each coordinate update is send via a single HTTP request to the URL, indicated by the output URL.

2.4 Process the GPS coordinate stream with SECONDO

SECONDO contains an operator called `csvimport`, to import csv separated data into a relation. This operator can read the data from a file or from a network socket. In this paper, the operator is used to import the generated GPS coordinate stream into SECONDO.

The Operator requires at least three arguments: (*i*) The data source, (*ii*) the amount of lines to skip and (*iii*) the used comment character. All lines beginning with the comment character will be skipped. The operator reads each line from the data source and parses the content, until the input is processed completely. When the data is read from a file, the end of the input is indicated by reaching the end of the file. When the data is read from a network socket, the end is indicated by the ascii character EOT.

Example: The following query in SECONDO will open a TCP socket on port 10 025 and read all the received GPS coordinate updates from the TCP socket. Every coordinate update is converted into a tuple and send to the operator `count`. After the query finishes, the total amount of received tuples is printed to the console.

```
query [ const rel(tuple([Time: string, Moid: int, Tripid: int,
          X: real, Y: real])) value() ] csvimport
          ['tcp://10025', 0, ""] count;
```

2.5 Architecture

The software is developed with an architectural focus on producing high amounts of GPS updates. BerlinModPlayer is written in C++ and uses the *POSIX thread library* [2] to create multiple threads and utilize the available hardware as good as possible (see Figure 2).

The input data is read by a *Producer Thread* and placed into a vector. In this vector, the data can be reordered or processed otherwise, depending on the simulation mode.

Occasionally, the content of the vector is moved to a fifo. A second thread, the *Consumer Thread*, reads the data from the fifo and writes them to the network socket. The fifo is the compound between the Producer and the Consumer Thread. Both threads are accessing the fifo simultaneously. To prevent race conditions, the access to the fifo needs to be synchronized. Due to the synchronization, only one thread can access the fifo at the same time. If both

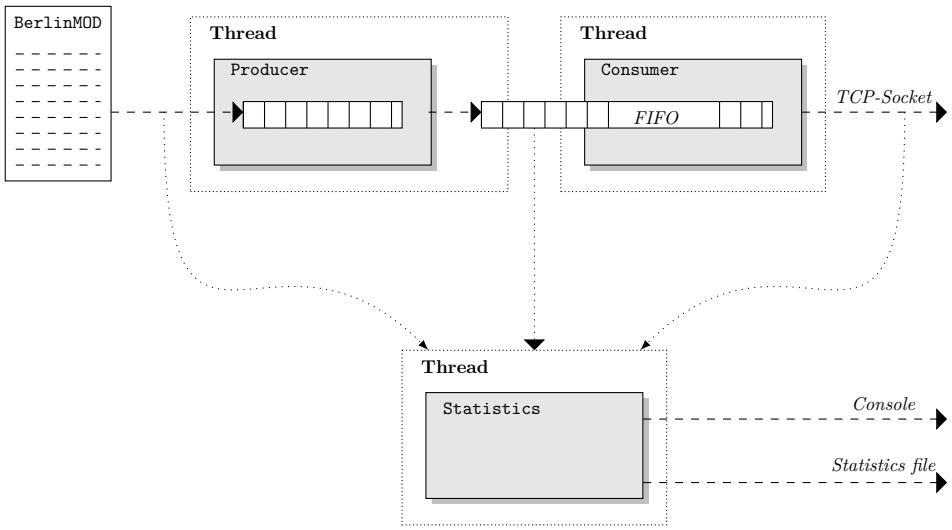


Figure 2: The architecture of the BerlinMODPlayer. The software uses three threads to do tier work: (*i*) A producer thread, (*ii*) a consumer thread and (*iii*) a statistics thread.

threads want to access the fifo one thread will be blocked and has to wait until the other thread has finished the access.

To ensure that both threads can perform their work without waiting much time, the above mentioned vector was introduced. With this vector, the Producer Thread can read data into the memory and placing it into the vector without interfering the consumer thread. Most of the time, the Consumer Thread can access the fifo exclusively. Only in the moment, when the data from the vector is moved to the fifo, simultaneous access to the fifo is needed.

A third thread, the *Statistics Thread*, is collecting statistical information about the read and written data and writes it to the console and into a file.

2.6 Simulation Modes

BerlinMODPlayer provides currently two different simulation modes: (*i*) fixed and (*ii*) adaptive. The simulation mode determines how the trips, generated by BerlinMOD, are handled.

Fixed: This simulation mode reads the units of the trips and sends the begin and the end coordinates of a unit to the TCP socket. The advantages of this mode are the predicable amount of generated GPS coordinates and the exact repeatability of the simulation. The drawback is, that the amount of generated coordinates is bound to the read performance of the hard disk, because the coordinates need to be read from that device (see Algorithm 1).

Algorithm 1 The fixed simulation mode

```
1 #define TERMINAL NULL;
2 #define QUEUESIZE 1000
3 queue myQueue;
4
5 Thread Producer(myQueue):
6     File inputfile;
7     inputfile.open();
8
9     while(! inputfile.eof()) {
10        while(queue.size >= QUEUESIZE);
11
12        String line = inputfile.readline();
13
14        // Extract the begin and the end position of every unit
15        Position beginPosition = convertToUnit((line)).begin();
16        Position endPosition = convertToUnit((line)).end();
17
18        // and add them into the queue
19        myQueue.add(beginPosition);
20        myQueue.add(endPosition);
21    }
22
23    inputfile.close();
24
25 Thread Consumer(myQueue):
26    TcpSocket mySocket;
27
28    mySocket.open();
29    while(queue.isEmpty() == true);
30
31    while(true) {
32       // Remove queue top
33       Position myPosition = queue.getAndRemoveTop();
34
35       if(myPosition == TERMINAL) {
36         break;
37       }
38
39       // and send element
40       mySocket.send(myPosition);
41    }
42    mySocket.close();
```

Adaptive: This simulation mode simulates the movement of the vehicles in real time. At the beginning of the simulation, the times tamp of the begin of the first unit is kept in memory. Subsequently, this time stamp is updated every second and indicates the current time of the simulation.

This mode implements a sweep line algorithm [15] [1, p. 22] to simulate the movement of the vehicles. The units of the vehicle trips are divided into three groups: (*i*) *dead*, (*ii*) *active* and (*iii*) *sleeping*. The simulation time represents the sweep line. The dead units are behind the sweep line, the active units are crossed by the sweep line and the sleeping units are ahead of the sweep line. According to the simulation time, the active units are loaded into memory and dead units are removed from memory. BerlinMODPlayer calculates and tries to send a position update every second for every active unit (see Algorithm 2).

The advance is, that the input data is interpolated in memory. So, the amount of generated coordinate updates is not bound to the performance of the hard disk. The drawback of this mode is, that the amount of generated coordinates is not predictable. So, the simulation can not be repeated exactly.

3 Usage

In the next three steps, the usage of BerlinMODPlayer is demonstrated. In the first step, BerlinMOD is started to produce the simulation data. In the second step, the produced data is converted and exported into the appropriate file format. In the last step, BerlinMODPlayer is used to run a simulation based on the data.

3.1 Prepare the Simulation with BerlinMOD

A working SECONDO installation is required for that step. Information about installing SECONDO can be found on the SECONDO website [8].

Download BerlinMOD: The followings commands will download and unpack BerlinMOD:

```
wget http://dna.fernuni-hagen.de/secondo/BerlinMOD/
  Scripts_OptimizerCompilant-2015-01-28.zip
unzip Scripts_OptimizerCompilant-2015-01-28.zip
```

Before running BerlinMOD, it is recommended to adjust the variables `P_NUMCARS` and `P_NUMDAYS` in the file `BerlinMOD_DataGenerator.SEC`. The first one determines how much vehicles should be generated, the second one determines for how many days the data should be generated. For example, the number of days can be set to one and the number of vehicles can be set to 10 000.

Algorithm 2 The adaptive simulation mode

```
1 #define TERMINAL NULL;
2
3 queue myQueue;
4
5 Thread Producer(myQueue):
6     File inputfile;
7     time_t simulationTime = getSimulationTime();
8
9     inputfile.open();
10
11    while(! inputfile.eof()) {
12        String line = inputfile.readline();
13        Unit unit = convertToUnit(line);
14
15        // Wait until the unit is active
16        do {
17            simulationTime = getSimulationTime();
18        } while(unit.begin() > simulationTime);
19
20        myQueue.add(unit);
21    }
22
23    inputfile.close();
24
25 Thread Consumer(myQueue):
26    TcpSocket mySocket;
27    time_t lastSimulationTime;
28    time_t currentSimulationTime
29    mySocket.open();
30
31    // Wait for queue to fill
32    while(queue.isEmpty() == true);
33
34    lastSimulationTime = getSimulationTime();
35
36    // Main loop: one loop per second
37    while(true) {
38
39       // Wait for next second
40       do {
41          currentSimulationTime = getSimulationTime();
42       } while(currentSimulationTime > lastSimulationTime);
43
44       // Try to send a position update for every active unit
45       for(int i = 0; i < queue.size(); i++) {
46         Unit unit = queue.getElement(i);
47
48         // Unit is dead, remove them
49         if(unit.end() < currentSimulationTime) {
50            queue.removeElement(i);
51            continue;
52         }
53
54         mySocket.send(interpolateCoordinates(unit,
     currentSimulationTime));
55
56         if(myElement == TERMINAL) {
57            break;
58         }
59       }
60       lastSimulationTime = currentSimulationTime;
61    }
62    mySocket.close();
```

Using a high number of vehicles is suggested, especially when using the adaptive simulation mode. Otherwise the simulation will generate only a few GPS updates every second. Assume, only three vehicles are moving in a particular time frame, you receive only three GPS updates every second.

After adjusting these variables BerlinMOD can be started with the following command:

```
SecondoTTYNT -i BerlinMOD_DataGenerator.SEC
```

3.2 Export the generated data

After the last command finishes, the trips are stored in SECONDO. These trips have to be exported now. There are two methods to export the data, resulting in different coordinate models. It is possible to (*i*) export the trips with *bbbike* coordinates [14] and (*ii*) export the trips with *World Geodetic System 1984* (wgs84) coordinates [12]. Depending on your field of application, chose one of the export commands.

Export trips with bbbike coordinates:

```
open database berlinmod;
query dataMtrip feed
   project[Moid,Tripid, Trip]
   projectextendstream[Moid, Tripid; Unit : units(.Trip)]
   projectextend [ Moid, Tripid; Tstart : inst(initial(.Unit)),
                   Tend : inst(final(.Unit)),
                   Xstart : getx(val(initial(.Unit))),
                   Ystart : gety(val(initial(.Unit))),
                   Xend : getx(val(final(.Unit))),
                   Yend : gety(val(final(.Unit))) ]
   sortby[Tstart]
   csvexport['trips.csv',FALSE,TRUE] count;
```

Export trips with wgs84 coordinates:

```
open database berlinmod;
query dataMtrip feed
   project[Moid,Tripid, Trip]
   projectextendstream[Moid, Tripid; Unit : units(.Trip)]
   projectextend [ Moid, Tripid; Tstart : inst(initial(.Unit)),
                   Tend : inst(final(.Unit)),
                   Xstart : getx(berlin2wgs(val(initial(.Unit)))),
                   Ystart : gety(berlin2wgs(val(initial(.Unit)))),
                   Xend : getx(berlin2wgs(val(final(.Unit)))),
                   Yend : gety(berlin2wgs(val(final(.Unit)))) ]
   sortby[Tstart]
   csvexport['trips.csv',FALSE,TRUE] count;
```

After executing one of the export commands, a file named `trips.csv` is created in the current directory.

3.3 Run the simulation

Start SECONDO and run the following command to open a TCP port 10 025, read all produced GPS updates and store them into the relation GPS.

```
let GPS = [ const rel(tuple([Time: string, Moid: int, Tripid: int,
          X: real, Y: real])) value() ] csvimport
          ['tcp://10025', 0, ""] consume;
```

On the command line of the system, execute the following command to run the simulation:

```
./bmodplayer -i trips.csv -u tcp://localhost/10025
        -s fixed -o statistics.txt  -b '2007-05-28 10:00:14'
        -e '2007-05-28 12:00:14'
```

3.4 Analyze the output file of the simulation

As specified in the command above, the statistics about the simulation are written into the file `statistics.txt`. BerlinMODPlayer ships with two templates (`statistics.plot` and `statistics_diff.plot`) for gnuplot to plot this data. The first template draws the total amount of read and send lines (the total plot), the second one draws the read and send lines per second (the differential plot). The plot of an example simulation is shown in Pic 3 and in Pic 4.

The plot of the data can be easily created by copying the gnuplot template and the statistics file into the same directory and running the following commands:

```
gnuplot statistics.plot
gnuplot statistics_diff.plot
```

After executing the two commands, the four files `statistics.gif`, `statistics.pdf`, `statistics_diff.gif` and `statistics_diff.pdf` are created. The first two files contain the total plot in the format GIF and PDF. The last two files contain the plot for the differential plot in the same formats.

4 Conclusion

In this paper, a load generator for GPS coordinates, called BerlinMODPlayer, was described. BerlinMODPlayer reads trips generated by BerlinMod and generates a stream of GPS coordinates from that data. In a real world scenario, such data can be generated by vehicles, each equipped with a GPS receiver.

The player provides two different simulation modes. The simulation mode determines how the input data is handled. The fixed simulation mode uses only

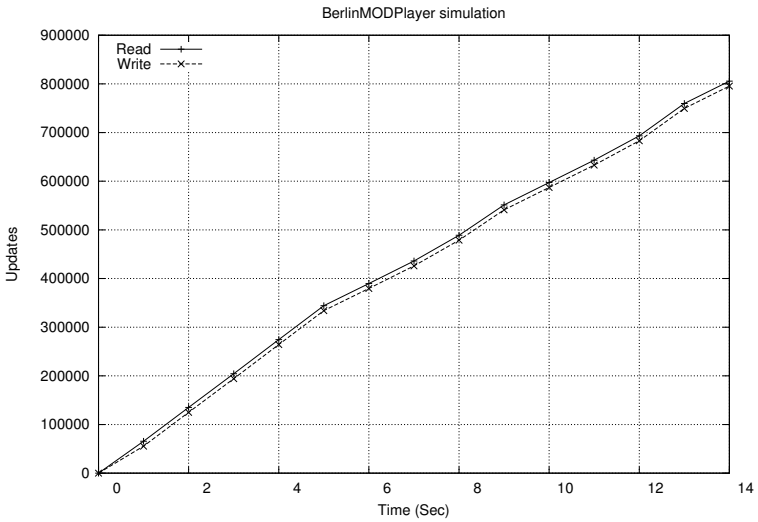


Figure 3: The progress of a simulation using the fixed simulation mode. The total amount of read and send lines are shown.

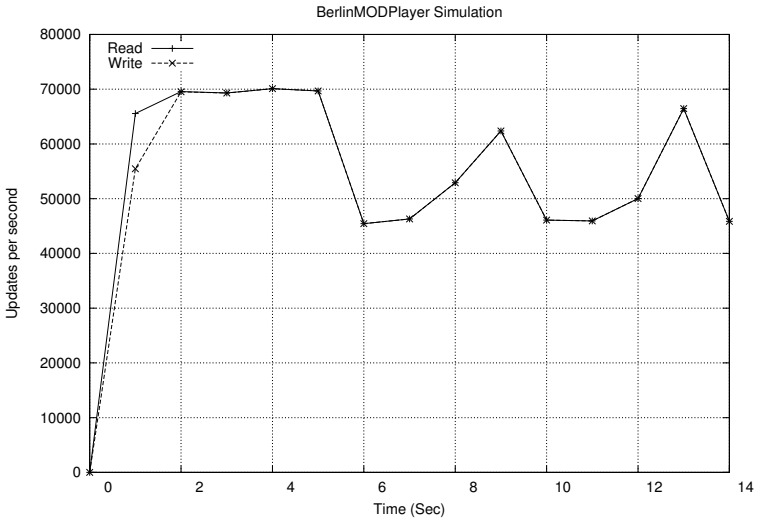


Figure 4: The progress of a simulation using the fixed simulation mode. The amount of read and send lines per second are shown.

the by BerlinMod generated GPS coordinates. The adaptive mode interpolates the movements of the vehicles. Statistical data about the stream is written to a file. This file can be easily analyzed and plotted with gnuplot.

References

[1] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[2] David R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[3] Oracle Corporation. Application Development Guide - Using the Load Generator to Test Your Application. http://docs.oracle.com/cd/E13157_01/wlevs/docs30/create_apps/loadgen.html, 2015. [Online; accessed 15-May-2015].

[4] Transaction Processing Performance Council. TPC BENCHMARK H (Decision Support) Standard Specification. http://www.tpc.org/tpch/. [Online; accessed 15-May-2015].

[5] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006. Obsoleted by RFC 7159.

[6] Christian Düntgen, Thomas Behr, and Ralf Hartmut Güting. Berlinmod: a benchmark for moving object databases. *VLDB J.*, 18(6):1335–1368, 2009.

[7] Ralf Hartmut Güting, Thomas Behr, and Christian Düntgen. Secondo: A platform for moving objects database research and for publishing and integrating research implementations. *IEEE Data Eng. Bull.*, 33(2):56–63, 2010.

[8] FernUniversität Hagen. Website of the software SECONDO. http://dna.fernuni-hagen.de/secondo/, 2015. [Online; accessed 15-May-2015].

[9] HammerDB. HammerDB an open source database load testing and benchmarking tool. http://www.hammerdb.com, 2015. [Online; accessed 15-May-2015].

[10] Philipp K. Janert. *Gnuplot in Action: Understanding Data with Graphs*. Manning Publications Co., Greenwich, CT, USA, 2009.

[11] LLC. Mockaroo. Mockaroo - realistic data generator. https://www.mockaroo.com/, 2015. [Online; accessed 15-May-2015].

[12] National Imagery and Mapping Agency. Department of Defense World Geodetic System 1984: its definition and relationships with local geodetic systems. Technical Report TR8350.2, National Imagery and Mapping Agency, St. Louis, MO, USA, January 1984.

[13] Francois Raab. TPC-C - The Standard Benchmark for Online transaction Processing (OLTP). In Jim Gray, editor, *The Benchmark Handbook*. Morgan Kaufmann, 1993.

[14] Slaven Rezic. BBBike - a route-finder for cyclists in Berlin and Brandenburg. http://www.bbbike.de, 2015. [Online; accessed 15-May-2015].

[15] Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, SFCS '76, pages 208–215, Washington, DC, USA, 1976. IEEE Computer Society.

[16] HP OpenCall Software. Seagull: an Open Source Multi-protocol traffic generator. http://gull.sourceforge.net/, 2015. [Online; accessed 15-May-2015].