SERVER-SIDE NAVIGATION SERVICE BENCHMARKING TOOL

Martin Golasowski, Kateřina Slaninová, Jiří Ševčík, Vít Ptošek, David Vojtek
IT4Innovations
VŠB - Technical University of Ostrava, Ostrava, Czech Republic
martin.golasowski@vsb.cz, katerina.slaninova@vsb.cz, jiri.sevcik@vsb.cz, vit.ptosek@vsb.cz, david.vojtek@vsb.cz

Abstract: Convenient access to a complex benchmarking and monitoring tool was the main motivation for implementation of the proposed benchmarking tool. This paper provides a brief summary of its design and implementation. Its main purpose is to test an experimental server-side navigation service which has to comply to a multi-criteria service level agreement. The service is deployed on a heterogenous high performance computing infrastructure which is monitored by the tool. Consistent test environment is provided by the tool for transparent analysis and optimization of the service performance.

Keywords: navigation, routing, benchmark, high performance computing

1 Introduction

There is a significant demand for new forms of vehicle routing algorithms. For example, Probabilistic time-dependent vehicle routing problem which could use precise route computation as an input was proposed in [1]. In the context of smart city, large number of routing requests can be issued in a very short time while there can be periods with very low or nonexistent traffic. Service which provides such functionality has to be flexible enough in order to satisfy the dynamic load while maintaining a given service level agreement (SLA).

These services are usually deployed on a heterogenous architecture consisting for example of both virtual machines and high performance computing (HPC) nodes. The SLA therefore consists of multiple criteria, such as response time or resources budget. These criteria can change over time. Maintaining the SLA over time is possible by using technologies from the ANTAREX project [10], such as the autotuning framework and a domain specific language.

In order to design and develop such a service a proper benchmarking and monitoring tool has to be provided. The goal of the tool is to provide users a way how to test the mentioned multi-criteria SLA against a given set of parameters of the service and its underlying technologies. The tool should provide a consistent and robust benchmarking environment which should be used in a transparent way for optimizing the service. Simultaneously, the tested service is provided as a black box and the tool only exposes only a subset of the service parameters which are relevant for testing. The tool can be used to obtain unified presentation of the test results, which is convenient for comparison of different parameter settings of the service.

The service provides navigation on a graph representation of the road network. It finds a path between a pair of two nodes or between a set of waypoints according to a selected criteria (speed, road/vehicle type, etc.). The performance metric of the service in our case is request throughput per second and average time per request. The input data set consists of a set of origin/destination pairs and parameters of the routing pipeline (PTDR, sample count, routing algorithm type, etc.). The service implements a subset of the HTTP communication protocol and uses simple JSON-based data structures.

1.1 Related work

There is an increased demand for architectures following the Anything as a Service (XaaS) paradigm [4]. Even though there are many robust and complicated protocols for providing such services; APIs based on top of the HTTP protocol are quite popular these days.

Any service provided over a computer network needs to be properly tested in order to determine possible performance bottlenecks in the architecture or possible errors in its output. Obtaining an energy efficient operating mode of the service is also a significant factor in testing, especially in big data centres [2, 3]. Demand for energy efficient adaption of this paradigm on powerful computing nodes is getting more significant with the overlap of cloud and traditional high-performance computing.

There are many benchmarking tools for HTTP-based services implemented for various platforms. They are usually implemented as a stand-alone terminal applications [5, 6]. Input specification is often realised by a template of the request or by a set of pre-defined requests to be sent. Some of the benchmarks provide more comprehensive statistical evaluation of the test, however, the output is often in plain text form, which is harder to parse and prone to errors. On the other hand, there are comprehensive tools, such as Locust [7], which provides all-in-one solution together with the analysis and visualisation front-end and result database. Such tools offer many features, which are unnecessary for our use-case and can be hard to alter and optimise. Therefore, we propose a custom light-weight solution which is designed for services running on heterogenous HPC infrastructure.

Next Section 2 is focused the benchmarking process including an example of the server-side navigation service benchmark used within the ANTAREX project [10]. The benchmark implementation is described in Section 3 and the asynchronous back-end which provides an execution and management of large number of small tasks follows in Section 3.2. Section 4 then concludes the paper.

2 Benchmarking Process

Benchmarking can be described as a process in which performance of a system is evaluated on a set of its parameters and input data. Performance of the system is then evaluated by relevant set of metrics (throughput, load, energy consumption, etc.). Subject of the testing is a relation between the parameter values and the selected metrics. Tested parameters can be for example I/O strategy, memory access pattern or various source code optimizations. The system is tested against a set of the parameter values while input data and run-time environment remain static.

Crucial property of a good benchmark is consistency. Individual tests share the same conditions through the entire test. It is often hard to assure completely consistent conditions for all the tests in real-world scenarios (for example in multi-task operating systems), therefore influence of the external factors should be kept to a sane minimum. For example, unnecessary services should be turned off as well as scheduling of maintenance tasks (backup, logging, etc.), in order to minimise performance fluctuation of the target hardware platform.

2.1 Server-side Navigation Benchmark

In our case, the subject for benchmarking is the server-side navigation service. Core of the service is the navigation pipeline which is deployed on a HPC cluster. This design allows flexible optimization of the pipeline according to various criteria to correspond with the current SLA.

Parameters of the benchmark are divided into three parts.

1. **Data sets** - type, size
2. **Static routing** - routing algorithm, route type
3. **Probabilistic time-dependent routing** - enabled, number of samples, *autotuning strategy*

The data set of origin-destination point pairs is sent to the service from a large number of concurrent processes. This approach allows us to simulate a large number of concurrent users

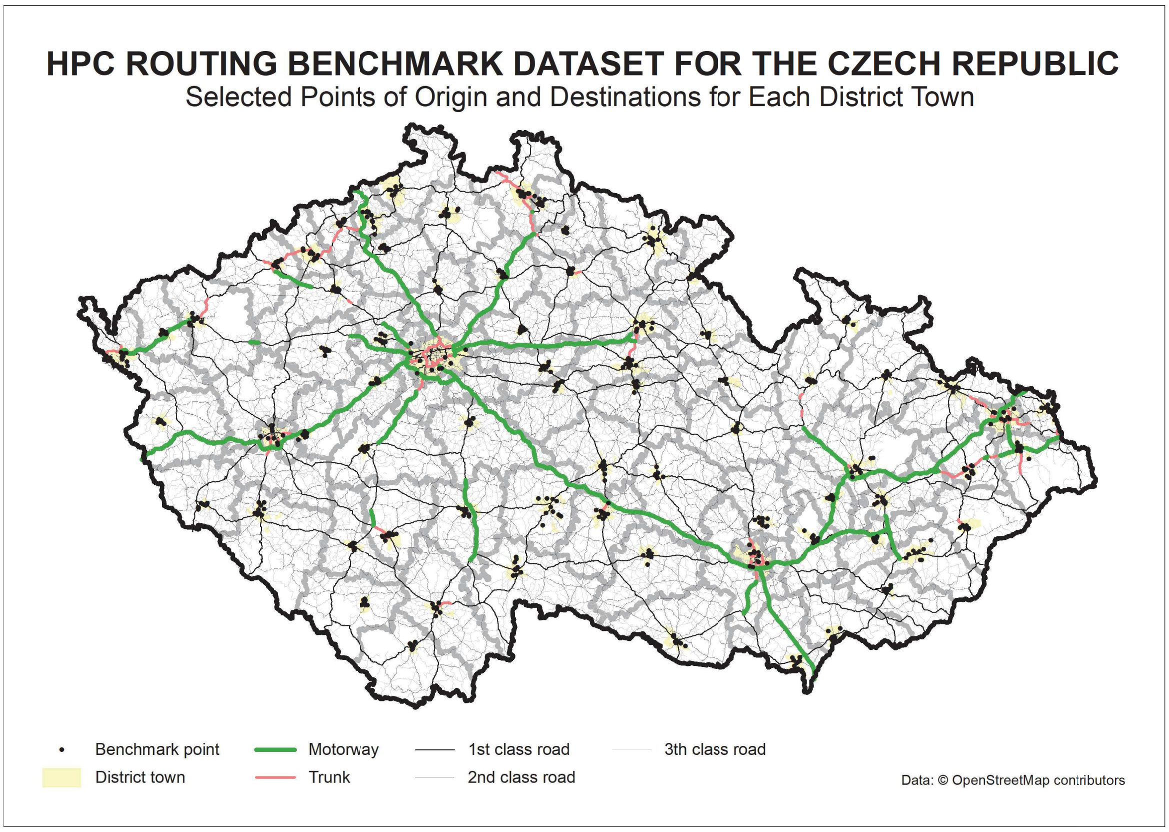

HPC ROUTING BENCHMARK DATASET FOR THE CZECH REPUBLIC
Selected Points of Origin and Destinations for Each District Town
Benchmark point
District town
Motorway
Trunk
1st class road
2nd class road
3th class road
Data: © OpenStreetMap contributors

Figure 1: HPC Routing Benchmark Dataset

of the service. The service then logs each individual requests. A benchmark is defined by the time-frame between sending the first pair of the input data to handling reception of a response for the last pair of the input data. The metrics are computed from the service log within the time-frame. Total amount of the requests sent is controlled by initial size of the selected data set and its size parameter, which can be used to artificially inflate the number of sent requests and to create specialized use case (such as smart city).

2.2 Data Sets

The points in current version of the benchmark are selected from the region of the Czech Republic, see Figure 1. The region is divided to 77 units according to the LAU1 (Local Administrative Units Level 1) division, which corresponds to individual districts of the Czech Republic. Each district has its administrative center which usually corresponds to its biggest city. There are 72 district towns as some of districts share one district town. There are 10 randomly selected points, that correspond to nodes of a routing graph, inside each of the 72 district cities. The resulting point data set is stored in a PostgreSQL database with a spatial extension PostGIS as a basic data set. Source data for the basic dataset are obtained from the Open Street Map project (OSM)[11].

Since the source data for the routing graph are updated regularly, the IDs of the graph nodes (the selected points of the basic dataset), can change. Therefore, it is practical to generate derived data sets only by a database query stored as a procedure or materialized view. The data sets are versioned according to the current processed OSM data.

In the current version of the benchmark, there are three data sets, defined by their size and type of the location:

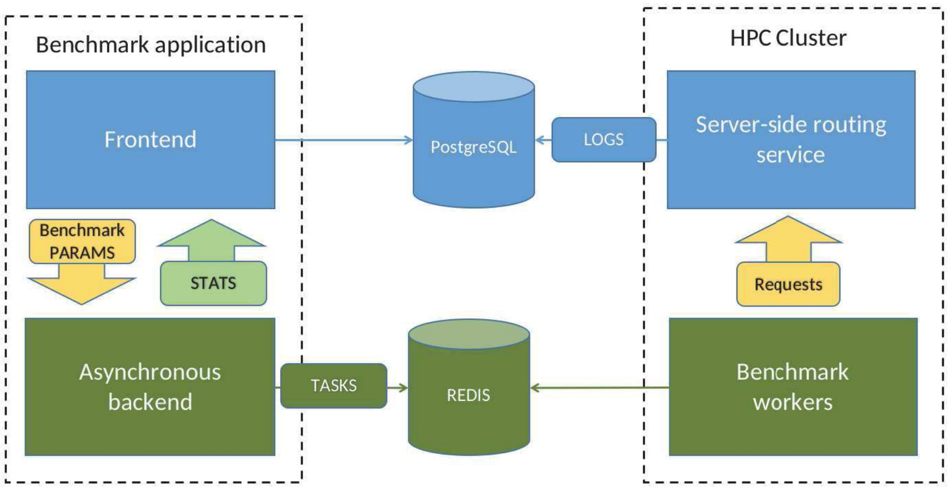


Figure 2: Overview of the application architecture

- **Smart city** - points within single LAU (city, 90 points)
- **Transit** - points between three main cities (Praha, Brno, Ostrava, 300 points)
- **All** - all 72 LAU1 regions (250 000 points)

3 Benchmark Application

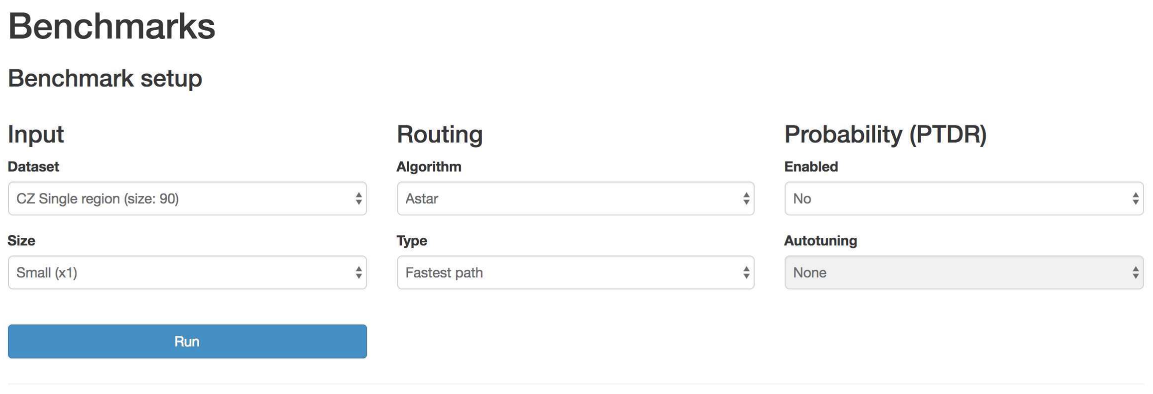
The application design is roughly divided into several parts. The monitoring and benchmarking front-end is the first part while the asynchronous task and database back-end is the second part. Overview of the application architecture is in the Figure 2. Web applications are inherently synchronous and stateless which makes them unsuitable for our purposes. The user front-end should only serve as a tool for control and monitoring of the benchmarking process. Therefore, an asynchronous approach has to be implemented, such that the process execution is independent on the user requests to the application front-end.

3.1 Implementation

The frontend is implemented in a popular Python web framework Flask [8]. In the context of benchmarking, it provides an user interface for setup, execution and presentation of the benchmark results. The main page contains a form for submitting the benchmark task and a table of results of the past runs. Detail of a particular benchmark run can be accessed from the results table (Figure 3). The detail presented in Figure 4 contains information about parameters of the viewed run and its results. In this particular case, the presentation of the results is split into data about the run itself and several metrics computed from the service log. Content of the entire page is generated on the server side directly from the service logs. This approach allows fairly easy extension of the analysis with new types of metrics.

3.2 Backend

The asynchronous back-end of the application is implemented with the help of the Celery framework [9]. It provides a convenient way of execution and management of large number of small tasks asynchronously and in a distributed fashion. The client part of the framework is



Benchmarks

Benchmark setup

Input

Dataset: CZ Single region (size: 90)

Size: Small (x1)

Routing

Algorithm: Astar

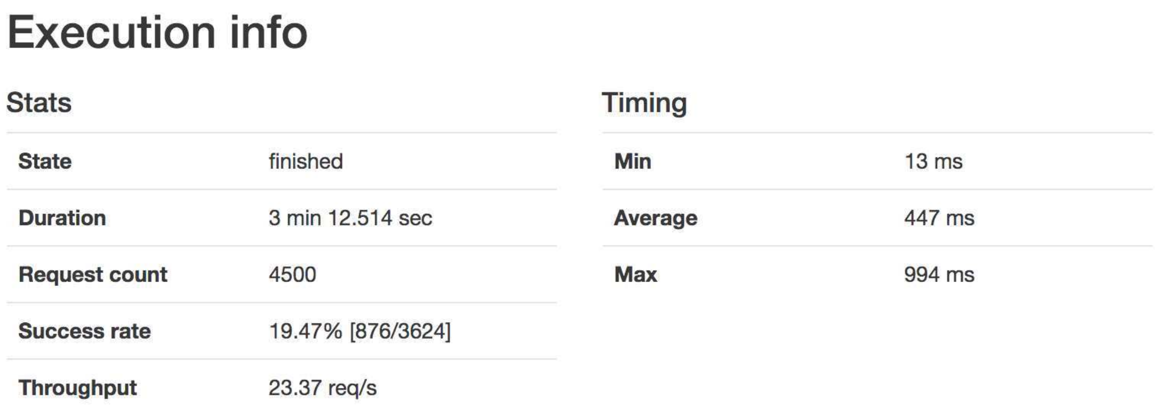
Type: Fastest path

Probability (PTDR)

Enabled: No

Autotuning: None

Figure 3: Interface for parameter specification



Execution info

Stats

State	finished
Duration	3 min 12.514 sec
Request count	4500
Success rate	19.47% [876/3624]
Throughput	23.37 req/s

Timing

Min	13 ms
Average	447 ms
Max	994 ms

Figure 4: Visualisation of the benchmark result

used by the front-end to create the benchmarking task and pass the parameters. The Celery then uses a type of in-memory concurrent storage (such as REDIS or RabbitMQ) to distribute the task across available worker processes. The worker processes communicate with the client through the in-memory storage, which is also used for storing the intermediate results. This mechanism is very useful in our case as it can be used to launch a large number of tasks which are processed by a pool of workers running on a HPC cluster. Celery takes care of the task scheduling itself and the benchmarking pipeline is implemented using the available task synchronization constructs.

The individual service requests are executed in a celery worker tasks. Each request is sent to the service in a blocking call to HTTP requests library and the result of the call is stored in the in-memory database.After all requests are sent, all results are passed to a finalization task, which computes aggregated statistics and stores them in the SQL database for presentation on the front-end.

The worker processes are deployed on a HPC cluster to ensure proper saturation of the service in order to maintain consistency of the benchmark.

4 Conclusion and future work

The authors presented benchmarking application which is used for transparent testing and further optimization of a server-side navigation service. The application implements an initial version of a consistent benchmarking environment. It can be further extended for testing other network-based services. Current version is used for testing of the autotuning framework and other tools developed within the H2020 project ANTAREX as well as for performing acceptance tests of the navigation service. It will be also used for testing of various types of navigation algorithms for development of vehicle routing problem algorithm.

In the next version, we plan to implement another type of the test, which will bypass the network connection handling layer of the service and test only the routing pipeline running on the cluster.

5 Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science - LQ1602", by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project "IT4Innovations National Supercomputing Center LM2015070", and by ANTAREX, a project supported by the EU H2020 FET-HPC program under grant 671623.

References

[1] Režnar, T., Martinovič, J., Slaninová, K., Grakova, E., Vondrák, V.: Probabilistic time-dependent vehicle routing problem (2016). Central European Journal of Operations Research, pp. 1–16.

[2] Yichao Jin and Yonggang Wen and Qinghua Chen: Energy efficiency and server virtualization in data centers: An empirical investigation (2012). In 2012 Proceedings IEEE INFOCOM Workshops, pp. 133–138.

[3] K. Le and R. Bianchini and T. D. Nguyen and O. Bilgir and M. Martonosi: Capping the brown energy consumption of Internet services at low cost (2010). In International Conference on Green Computing, pp. 3–14.

[4] Y. Duan and G. Fu and N. Zhou and X. Sun and N. C. Narendra and B. Hu: Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends (2015). In 2015 IEEE 8th International Conference on Cloud Computing, pp. 621–628.

[5] The Apache Benchmark. (2017) The Apache Software Foundation
https://httpd.apache.org/docs/2.4/programs/ab.html [Accessed 20/05/2017].

[6] Siege (2012) Jeffrey Fulmer at al.
https://www.joedog.org/siege-home/ [Accessed 20/05/2017].

[7] Locust (2016). Locust
http://locust.io/ [Accessed 20/05/2017].

[8] Flask (2010). Armin Ronacher
http://flask.pocoo.org/ [Accessed 20/05/2017].

[9] Celery - Distributed Task Queue (2017). Ask Solem
http://docs.celeryproject.org/en/latest/index.html [Accessed 20/05/2017].

[10] Silvano, C., Agosta, G., Cherubin, S., Gadioli, D., Palermo, G., Bartolini, A., Bispo, J., et. al. (2016). The ANTAREX approach to autotuning and adaptivity for energy efficient HPC systems. In Proceedings of the ACM International Conference on Computing Frontiers (pp. 288–293). ACM.

[11] Haklay, M., Weber, P. (2008). Openstreetmap: User-generated street maps. IEEE Pervasive Computing, 7(4), pp. 12-18.