EXPLOITATION OF HPC INFRASTRUCTURE SERVICES FOR REAL-TIME CRITICAL SMALL REQUESTS

Jiří Ševčík[(a)], Martin Golasowski[(b)], Jan Martinovič[(c)], David Vojtek[(d)], Jan Faltýnek[(e)]

[(a),(b),(c),(d),(e)]IT4Innovations, VŠB - Technical University of Ostrava,
17. listopadu 15/2172, 708 33 Ostrava, Czech Republic

[(a)]jiri.sevcik@vsb.cz, [(b)]martin.golasowski@vsb.cz, [(c)]jan.martinovic@vsb.cz ,[(d)]david.vojtek@vsb.cz
[(e)]jan.faltynek@vsb.cz

ABSTRACT

Modern smart devices are often supported by innovative online services which bring new challenges for cloud and high-performance computing industry. These services have to be designed to handle a dynamic load from various sources while maintaining efficient operation in a given computing environment. Operation of such services has to conform to a multi-criteria service level agreement which can include availability, response time, energy efficiency and other factors. In the field of high-performance computing, this is accomplished by using specialized mechanisms and tools for efficient scheduling and resource allocation. This paper proposes a system which can be used to implement such service on a high-performance computing infrastructure. We demonstrate our approach on an experimental server-side traffic navigation service. The solution is applicable both for high-performance and cloud computing.

Keywords: high-performance, cloud, quality of service, routing navigation, service level agreement.

1. INTRODUCTION

A number of smart devices integrated into our daily lives increases every day. There is a lot of effort supported by public and private entities often focused on increasing the quality of everyday life while consuming less energy. Such activities often span multiple disciplines and involve topics like optimization of public and personal transport, fine control over heating and cooling systems in commercial buildings, weather nowcasting, waste management logistics and others. These activities often make use of the so-called edge computing which is a combination of small embedded computing devices (IoT) and a central data storage accompanied by powerful computing resources like high-performance computing (HPC) clusters.

Edge computing offers a degree of autonomy for the individual devices such that the sensor itself can directly affect another device (switch, actuator) without the need to contact the central unit. This is especially useful in situations with little power available and limited network connectivity in remote locations. The device can contact the central unit only in case there are interesting data available. The central unit stores the received data and executes further processing and aggregation of data. A result of the processing can be further used to perform simulations and projections. The processing pipeline can consist of several steps which can involve heavy computations like training of a deep neural network, filtering or clustering. The same pipeline can be used either to receive a continuous stream of data in an online mode or to process a set of historical data in a batch mode.

In this paper, we propose a system which can be used to execute a single pipeline of tasks on the HPC cluster in above mentioned modes. The system consists of a master process which handles the incoming data sets and a set of worker processes which communicate with the management process. The master process manages the distribution of the work to the workers using a custom message passing protocol.

In the online mode, the master process of the system resides outside of the cluster and handles deployment of the individual workers by submitting jobs to the cluster scheduler. The system is also prepared to handle dynamic loads by using external tools for autotuning (Gadioli, Palermo and Silvano 2015, Silvano 2017) which provides a number of resources for individual stages of the pipeline to handle variations in the request load while maintaining efficient operation of the service.

In the batch mode, the master and worker processes run together in a single job and process a predefined set of data. The workers can be run as a native Linux application directly on a compute node which introduces a possibility to use a massive parallel code and highly optimized math and data processing libraries. In both mentioned modes the defined pipeline remains the same, the only difference is in location of the master process. In this way, we focus on improving the usability of the HPC platform.

HPC infrastructure provides a vast amount of computational resources. Current clusters usually consist of multiple types of devices, including CPUs, GPUs or other types of specialized accelerators, connected by high bandwidth network with redundant

topology. In order to exploit the power efficiently, the applications have to run with as little overhead as possible. This is achieved by using specialized parallel programming models like MPI, OpenMP or CUDA.
On the other side, cloud services provide vastly easier access to a large number of computing resources through simple APIs. This abstraction is usually at the cost of overhead introduced by virtualisation or similar technology. Also, high bandwidth network typical for HPC installations is less common in the cloud environment. The central unit of such a smart system can be a combination of multiple types of resources.
There are similar tools which can be used to implement such a system. The streaming interface of Apache Kafka (Dunning and Friedman 2016) can serve as a message broker using the consumer-producer pattern and can be used by clients from different programming languages via its own binary protocol. However, it does not offer integration with HPC cluster scheduler and due to its robustness and vast array of features is unsuitable for lightweight job batch processing of data.
There is also a number of systems coming mainly from the biochemistry field which allows specification of data processing pipelines by using the Common Workflow Language (CWL). One example of such system is Toil (Vivian, et al. 2017), which supports specification of such workflows, allows integration with many common Cloud and HPC environments. It is suitable for launching extensive workflows with many stages. It does not offer a possibility for running an online service and integration of the autotuning interface. There are also other systems which main aim is to utilize workflows mainly in a specialized case (Al-Ali, et al. 2016) or offering a system for executing pipelines independently of their types (Cima, et al. 2018).
We demonstrate the functionality of the proposed system on an experimental server-side traffic navigation service deployed on an HPC cluster.
This paper is organized as follows: Section 2 describes the architecture, its components and main features. Next section is dedicated to the description of communication between the system parts. Extensibility of the system is also introduced. Section 5 provides an overview of Quality of Service requirements. Description of usage of external data is also provided. In Section 7, navigation use case for proposed architecture is presented. Testing methodology and its results are provided in Section 8. In the end, we mention the possibility of future work.

2. SOLUTION ARCHITECTURE OVERVIEW

Our proposed solution is a specialized architecture which is divided into two main parts - a management system part and a computation part. This paper is focused mainly on the second part, but the first part will be briefly described as well.

2.1. Management system

The management system part facilitates communication with service clients and routing workers. It is divided into two services – proxy and management service. Both are designed to run completely independently on different machines and communication between them is secured by TCP/Linux sockets. Both parts use multiple threads and because of this fact, computation resources could be exploited very efficiently. For the best performance in the area of network communication, it is recommended to run both parts of the system on the same machine using Linux-based sockets because an amount of transmitted data could be very high (Stevensson, Fenner and Rudoff 2004).
The proxy service is responsible for direct communication with clients. After accepting the connection, the request is parsed and sent to the management service. Computed result is then returned, and proxy service sends it to the client, so no additional computation or modification is done. The proposed solution of that part allows separation of the pre-processing of the request and the computation. This separation is useful not only in case of the scattering load on different machines, but it is also a security feature because clients do not have a direct access to part of the system part responsible for computation. Protocol for this client-server communication is based on HTTP REST protocol but it is possible to develop and use a different form for the communication, for example, binary based streaming or permanent connection between client and server side.
The management service communicates with the proxy service and with the part of the system which is responsible for the computation. Scheduling and allocation of the computing resources will be done at this part. An important part of work is also transformation of the data which are used as an input or output for the computation part. Transformation after receiving and before sending the data is necessary for two main reasons which are based on a communication protocol and also on the data transfer performance. The protocol is described in Section 3 of this article. For the data transfer performance, only relevant data, which are really necessary for the next computation are sent to the next computing part.
To obtain statistics, it is possible to enable optional logging to an external database. This could be done in both of parts of the system.
Because of the separation of the management system and the part responsible for the computation, both parts could be replaced by different modules which implement the same communication protocol. This can be useful for example in the case when there is a need for a different type of work scheduler.

2.2. Computation worker

As it was stated, our solution separates communication and scheduling from computation. This part is marked as a computation worker and its architecture is designed to exploit all available computing resources in an

efficient way. This separation allows the system to be efficient and to provide high request throughput.
The worker communicates only with the management service so every request for computation has to be passed through it. Thus, every information or statistics from the computation in the worker has to be sent into that part of the system.
The worker internal architecture allows multi-threading. This allows to exploit all available resources effectively and also it is possible to process many requests concurrently. The system was also designed with emphasis on usage of usage locks to ensure efficient concurrent communication and memory access. Amount of threads can be configured dynamically which is an advantage for example in the case where resources are limited and there is a need to run multiple types of workers on a single device.
Architecture of the worker is separated into two parts to ensure efficient implementation of various types of tasks. The first is a core, which is responsible for communication with the management system which also includes service messages which are part of the service communication protocol. Actual computation is performed in the second part of the worker which uses the provided worker

2.2.1. Worker mode

The purpose of modes is fact that the whole architecture can be used for resolving tasks which could contain a lot of different computations. Internally, the modes represent the states of a deterministic finite automaton. The messages exchanged between the different parts of the system represent transitions between the different states of the request
The mode at the beginning receives a message which contains input information which is necessary for a performing the computation. After performed computation, mode creates a message with results and this message is passed to the core part of the worker and then transmitted to the management system. As it is described below, messages could contain various types of values. Thanks to this fact, the mode itself could execute a lot of different types of computation and messages with result could also vary as well.
For a better scalability, every instance can run only one mode at a time, but modes can be changed dynamically during the worker life cycle. This fact corresponds with the proposed approach for a high scalability.

3. COMMUNICATION

There are three different types of communication in the system:

- communication with external sources - client with the proxy service,
- internal communication between the proxy and the management part,
- internal communication between the management and workers.

The first type of communication is based on HTTP REST and the second type is based on binary based object passing. For the third, the most crucial part, the specialized internal protocol is implemented. It is based on binary data format which ensures better throughput in comparison with the text-based HTTP protocol used by web services. The protocol uses only signed and unsigned numbers. Combination of both is allowed. For a special occasion, the string could be also used.

3.1. Communication workflow

The communication design uses two separate threads. Both of these threads use input and output message queue which is shared with all other threads. Communication with the management service is currently based on sockets and could be both TCP or UNIX based.
Input thread waiting for an incoming message and after successful read and base validation is that message pushed to the input queue. On the other hand, the output thread is waiting for new messages in the output queue and without any further validation send these messages into the management server.
Each computation thread operates in a loop. At the beginning of the loop, a message from the input queue is removed and then processed. Result of that computation processing is returned as an output message and pushed into the output queue.
With this design, there are no delays or possible deadlock which can occur in case of sharing communication socket.

3.2. Protocol

The protocol is divided into two parts – input and output messages. For a possible extensibility, both parts have mandatory and optional parameters and contain a header. The header is the same for both types and includes values which are used for validation and identification:

1. ID – unique value for each message during a whole run of the service. Value is generated in management service.
2. Origin – original source of the message.
3. Type – type of the message. An important value for a next processing of the message.
4. Worker type – used mostly for control that request will be handled by according worker.

After receiving the request, validation is carried out and the message is pushed for a next processing.

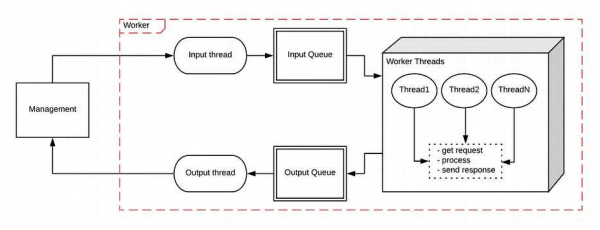

Figure 1: Flow of communication

3.2.1. Input messages

Processing of that message depends on value of the Type field. For example, in case of stopping or reloading the worker, no additional parameters in the message are used. But in the case of route request, other message parameters are parsed, and computation is done with these parameters.

3.2.2. Output messages

In that message type, various values can be used. Their type depends on requested computation. The header has the same structure as described above but Type is changed according to the type of the result.

3.3. Special messages

Currently, we integrated some specialized message types which are not directly used for the computation. These messages are handled directly in the input thread.

- reload – reload routing graph. It is possible to reload only selected parts,
- change type of worker – due to adaptivity, the worker could easily change its computation purpose just-in-time without the need of restarting and subsequent initialization,
- kill – stop the worker,
- ping – similar to network ping utility.

4. EXTENSIBILITY

The system can be extended with more functionality. In the case of adding a new type of worker, changes have to be added in both of management service and worker. In management, new protocol parameters has to be specified and also its passing into the worker. On the worker side, there are base rules on how to add new functionality. First of all, the new Worker type has to be added into enumeration with available types, and optionally, a new Type of message can be added. Next, the handler for the new type of the message must be created and integrated into input thread. It is also possible to exploit inheritance and extend existing message structures and handlers.

It is necessary, in the case of creating new message types, to keep the structure of the base header which was described in Section 3. Any additional parameters have to be processed by the newly created handler. The system itself contains a lot of utilities for parsing messages, decoding from/to binary form, so no additional work in that area is necessary.

There is a possibility to replace communication system with management service for use on a different architecture (current implementation uses Linux-based sockets). Moreover usage of different communication styles e.g. pipes or shared memory is possible to implement.

Due to a logical separation of handlers and messaging system, these improvements require changes only at one place of the code so other parts will not be affected. It is possible to change the whole communication protocol if necessary entirely, but that change will require a bigger modification of the worker core.

As we can see, adding a new handler is simple so a current system can be extended easily which is useful if there is a demand for high flexibility. After this step, new pipelines can be specified.

Because of fact, that the system has to be to modifiable and easily extensible, all of the parts of its architecture are designed to do these possible changes.

5. QUALITY OF SERVICE REQUIREMENTS

There are multiple points of view on the Quality of Service (QoS) and Service level agreement (SLA) (Chu 2014) requirements. The most important criteria is the response time which means, that the system has to be able to process a certain amount of requests in the selected time. This requirement can be further refined into two tasks. The first is dealing with SLA which is chosen for single computation e.g. all requests have to be computed in this time. The second is focused on time which is required for the computation of the whole demanded requests.

These requirements could be achieved thanks to the dynamical ability of allocation of additional computation resources on-demand. As it is shown in a section with experiments, the system is able to determine a minimal amount of resources for fulfilling demanded SLA.

6. USAGE OF EXTERNAL DATA

The system is designed to be autonomous without the need for other external data sources e.g. databases or other services running outside HPC environment. Main reason is coming from the fact that computation is carried out on dynamically allocated resources (nodes) which are inside HPC network and access to the externally running systems or sources is not allowed on the network layer.

In cases, where access to the database is necessary, the management system offers functionality for communication with this source. But for enabling it, it has to be an ad-hoc solution considering necessity of its usage and also the network has to be configured which could be a problem in some cases. The next important fact is a security of the computation and a whole HPC environment (Nowak, et al. 2013).

7. SERVER-SIDE TRAFFIC NAVIGATION USE CASE

Our case of the architecture was to implement a system which is responsible for a computation of routes for vehicles navigation. This scenario uses all of the three parts of the architecture – the management, the proxy and also worker modes. Each part extended the system functionality for parts which are necessary for the routing navigation server-side. Also specialized algorithms for a route computation are used as well.

The service also offers a system for communication with external services. In our case, this system is used for receiving notification about the actual traffic situation which is represented as changes in a routing graph. Next usage is for communication with a system responsible for autotuning.

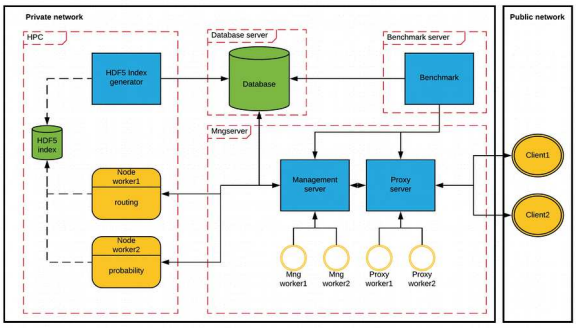

Figure 2: Base architecture scheme

7.1. Used worker modes

Currently, there are four worker modes:

1. One to one routing (Simple) - computes single path – Dijkstra and A* algorithm.
2. Alternative routing (Alternatives) - k-shortest paths based on the plateau algorithm.
3. Probabilistic Time-Dependent Routing (PTDR) (Golasowski, Tomis, Martinovič, Slaninová, and Rapant 2016) - estimates the distribution of the path travel time.
4. GPS translate (GPS to Internal) - translates GPS coordinates to routing graph nodes identifier.

7.2. Request handling

With current functionality, the system can handle client requests for computing routes based on origin/destination (O/D) parameters. For internal usage, an identifier is used as O/D, but for an external client requests, GPS coordinates are demanded as an input. In a basic case, the client wants only one route (the first mode - Simple) or demand alternatives routes (the second mode - Alternatives).

In some cases, for client request it is necessary to perform more than one computation with the use of different modes. This is, in the current state of work, a case, in which alternative routes has to be computed and based on a result of this computation, PTDR algorithm is carried out. To achieve this, two workers with different modes have to be available. There is also another case in which is the necessary usage of the fourth mode. This can occur in a case when the first or second mode does not use functionality for GPS to Internal, so separate workers with that functionality have to be available.

As it was mentioned before, single worker instance can be run just only in one mode and then is available for computing only one kind for the request. For solving cases with a request containing multiple computation steps, a specialized request pipelining is used.

7.3. Request pipelining

For the description of pipelining functionality, the case with the computation of Alternatives with integrated functionality of GPS to Internal and following usage of PTDR will be used as an example.

After passing the request into the management service, the internal system finds out which parts have to be computed for the given request. In our example, the system creates an info structure in which it is saved information that Alternatives and PTDR computation have to be carried out. This information is stored during a whole lifetime of the request. The system next work in these steps:

1. Send data for computation into worker running in the Alternative mode.
2. Receive resulted data from the worker and store them in the info structure.
3. Prepare data for the form suitable as an input into the PTDR mode.
4. Send data to worker running in PTDR mode.
5. Receive data from the worker, transform them into a form suitable for the proxy service and send them into this service.
6. Remove info structure belonging to the current request.

Transformation after receiving and before sending the data is necessary for two main reasons which are based on a communication protocol and also on the data transfer performance. The protocol is described below in this article. For the data transfer performance, only relevant data, which are necessary for the next computation, are sent to the workers. In this example case, step 2 return computed routes in GPS coordinates and also in internal identifiers which are used as an input for the step 3. Identifiers have a smaller size and therefore performance and communication are faster because they are used as an input for the next step.

After getting results from step 5, route result obtained in step 2 are re-ordered based on these results and finally sent back to the client through the proxy service. This example could be extended by two additional steps in case of usage of worker running in GPS to Internal mode. In that case, one step will be added before step 1 (translation from GPS to internal identifiers) and the next one will be added between step 2 and 3 (obtained

routes will be translated from internal identifiers into GPS format).
Described pipelining is adaptable for various schemes, but appropriate methods for data transformation and preparation have to be developed for its proper usage.

7.4. Data sources

The data used by workers can be obtained from various sources, such as traffic monitoring or weather prediction. Because of a high amount of stored data, an HDF5 file format was chosen as the main type for storage. This standardized format enables a solid compression for storing data and also offers a mechanism for fast access to the required part of data. Thanks to this characteristic, dealing with HDF5-based sources is fast and efficient in terms of input/output. As others data sources, SQLite with its SpatiaLite superstructure is used as well.
These sources are processed to a form suitable for the individual workers and stored in appropriate formats:

- Routing index (routing graph) - contains a graph representation of a road network used for route computation. The graph is stored in a structured HDF5 file.
- Routing index changes – particular speed changes in a road network.
- Spatial data structure based on SpatiaLite (Marchesini, Neteler, Frigeri, Casagrande, Cavallini and Furieri 2014) which provides GPS to internal IDs which are necessary for routing computations.
- Probabilistic speed profiles - used by the PTDR worker, stored in an HDF5 file.

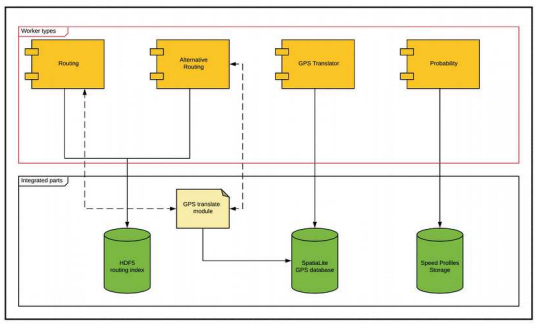

Figure 3: Worker modes with data sources

7.4.1. Data acquisition

Open Street Map dataset (OSM) (Ptošek and Slaninová 2018) is a main data source for the routing index but only OSM line data with tag value highways. The result of our processing is a road network. Transformation of a spatial and non-spatial attribute of OSM highways lines is performed so the resulted road network meets basic needs for usage in server-side routing. These criteria are:

- The road cannot self-intersect or self-overlaps.
- One road cannot overlap another one.
- The roads can intersect only when it is logically correct (e.g. roads intersection or change of non-spatial attribute of road).
- Each road and its direction has own spatial representation.

Next step in processing is a derivation of nodes from basic road network. Each node has a flag, so roads intersection, dead-end, border node, one-way dead-end and pseudo-node can be distinguished. In the following step, aggregation of the road network features into segments to eliminate all pseudo-nodes is carried out. Loops and multiple edges for the segment network are allowed. Partition of roads, nodes and segments according to geographic areas into smaller sets is also allowed. As a result, SQL HDF5 set, in SQL OpenLR set (OpenLR 2012), benchmark set and other product from the nodes and the segment network is created. In the last step SpatiaLite database for each partition, with basic network analysis capabilities is produced as well.
The routing index is based on data obtained in this processes. For a creating of the routing index file is used a special process which is adapted for an effective loading of the file in the routing worker. This index is loaded at the start of the worker and data are read-only for its computation threads.
For achieving reliable routes, the system has to be able deal with an actual data which can vary with the base data. Thus, the system has to be able to work and integrate new information. This is done through a next HDF5-based file. The file with changes represents an actual speed on the road network. The changes contain information about actual traffic jams, weather conditions, etc. External processes generate the file and every running worker is notified about this new data changes. For saving storage space, only changes from the previous state are stored. Changes in the graph could be done only through the main thread and just after a notification from the management service.

8. EXPERIMENTS

Two types of experiments were carried out according to the topic of this article. Both are focused on how many resources are needed for handling all requests in a given time which is differently specified for each experiment. Types of these tests are focused on previously mentioned QoS and SLA for the current system.

8.1. Experiments dataset

The points have been selected from geographical areas defined by Local Administrative Units (LAU1) corresponding to the three biggest cities in the Czech Republic (Praha, Brno and Ostrava). The dataset is then created as a cartesian product of the entire set to create pairs of origin and destination location. The used probabilistic speed profiles emulate the behaviour of the real world traffic using a custom Markov chain model. The whole dataset contains 25,560 pairs from which the

first 2,000, 500 and 250 were chosen for experiment cases.

8.2. Quality of Service for a single request

In this scenario, tests are focused on detecting how many resources are needed for resolving every incoming routing request in a given time. Because the current workers can run in different modes, two types of measurement for different modes were performed. In the first, 2,000 parallel request for single route response (worker mode 1) was created and in the second one, 500 parallel requests with 10 alternatives (worker mode2). In every iteration, one computation worker (node) were added. Results of these tests are shown in Figures 4. and 5.

For the first test, SLA time for resolving every request was chosen to be 1 second. As the results show, for achieving this time, at least 6 workers (nodes) are necessary. For the second test, SLA was 5 seconds and so 7 workers are necessary for fulfilling this SLA.

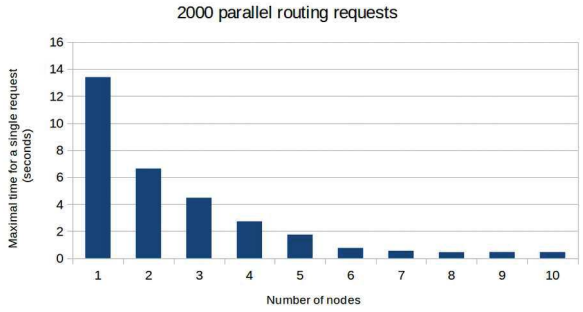

Figure 4: Single request test – 2,000 requests

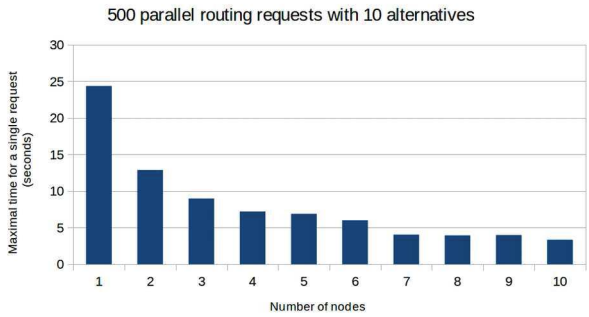

Figure 5: Single request test – 500 requests with 10 alternatives

8.3. Quality of Service for all requests

This test is based on fact that there are cases in which is it not necessary to resolve every routing request in such time as in the previous scenario. For this case, the test is finding how many resources are necessary for resolving all routing requests in a given time. Two different types of the worker mode are used in that scenario.

Configuration for this test case was chosen as one constant Alternative worker and an increasing amount of PTDR workers (worker mode 2 and 3). As the PTDR configuration, sample count was set 100 and departure usual Monday at 6 AM.

200 requests with 5 alternatives routes and PTDR were executed and SLA time for completing all request was 30 seconds. As is it shows in Figure 6, 8 nodes are necessary for the chosen time.

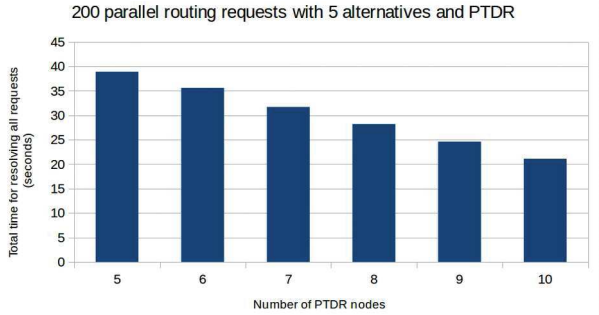

Figure 6: All requests test – 200 requests with 5 alternatives and PTDR.

8.4. Tests evaluation

As the results proved, the system is able to achieve demanded SLA times with a proper amount of resources. Tests also confirm a scalability with a growing amount of computation resources.

9. CONCLUSION AND FUTURE WORK

We proposed and created a highly scalable system for usage on HPC architecture which offers efficient resource allocation and scheduling as well. Thanks to its design, this solution could be applied in different areas of computation and extensibility could be done easily. This allows the creation of advanced pipelining with a big amount of different worker modes.

As the experiments on our use case proved, the system is able to fulfil different demands in the area QoS and SLA. Also, the scalability in case of usage of increasing number of computation resources was confirmed.

Future work will be done mainly in the area of automatic resources allocation and integration of external tools used for autotuning. Another goal is an improvement in scalability.

ACKNOWLEDGEMENT

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

REFERENCES

Silvano, Cristina, et al., 2017 The ANTAREX tool flow for monitoring and autotuning energy efficient HPC systems. SAMOS 2017 - International Conference on Embedded Computer Systems: Architecture, Modeling and Simulation. 2017.

Gadioli D., Palermo G. and Silvano C., 2015. Application autotuning to support runtime adaptivity in multicore architectures. SAMOS 2015 - International Conference on Embedded Computer Systems: Architecture, Modeling and Simulation, 2015 International Conference, IEEE, pp. 173-180.

Chu W.C.C., et al., 2014. An Approach of Quality of Service Assurance for Enterprise Cloud Computing (QoSAECC). International Conference on Trustworthy Systems and their Applications - Taichung, pp. 7-13.

Marchesini I., Neteler M., Frigeri A., Casagrande L., Cavallini P. and Furieri A., 2014. GIS Open Source: GRASS GIS, Quantum GIS and SpatiaLite [Kindle Edition]. Dario Flaccovio Editore.

Cima V., Böhm S., Martinovič J., Dvorský J., Ashby T.J. and Chupakhin V., 2018. HyperLoom Possibilities for Executing Scientific Workflows on the Cloud. Barolli L., Terzo O. (eds) Complex, Intelligent, and Software Intensive Systems - CISIS 2017. Advances in Intelligent Systems and Computing, vol 611. Springer, Cham.

Ptošek, V. and Slaninová K., 2018, Multinode Approach for Map Data Processing, 5th International Doctoral Symposium on Applied Computation and Security Systems (ACSS)

OpenLR™ White Paper, 2012. Available from: http://www.openlr.org/data/docs/OpenLR-Whitepaper_v1.5.pdf [accessed 23 July 2018]

Golasowski M., Tomis R., Martinovič J., Slaninová K. and Rapant L., 2016. Performance evaluation of probabilistic time-dependent travel time computation. IFIP International Conference on Computer Information Systems and Industrial Management, Springer, pp. 377-388.

Nowak M., Frankowski G., Meyer N., Yilmaz E., Erdogan O., Nominé J.P. and Robin F., 2013. Security in HPC Centres. Available from: http://www.prace-ri.eu/IMG/pdf/wp79.pdf [accessed 10 June 2018]

Dunning T. and Friedman E., 2016. Streaming architecture: new designs using Apache Kafka and MapR streams. O'Reilly Media, Inc..

Vivian J., Rao A.A., Nothaft F.A., Ketchum C., Armstrong J., Novak A. and Schmidt H., 2017. Toil enables reproducible, open source, big biomedical data analyses. Nature biotechnology 35, pp. 314-316.

Stevenson W.R., Fenner B., Rudoff A.M., 2004. UNIX Network Programming. Addison-Wesley Professional.

Al-Ali R., Kathiresan N., El Anbari M., Schendel E. and Abu Zaid T., 2016. Workflow optimization of performance and quality of service for bioinformatics application in high performance computing. Journal of Computational Science – Volume 15, pp. 3-10.