IMPLEMENTING COMPUTATIONS WITH DYNAMIC TASK DEPENDENCIES IN THE DESKTOP GRID ENVIRONMENT USING EVEREST AND TEMPLET WEB

S.V. Vostokin [1,a], O.V. Sukhoroslov [2], I.V. Bobyleva [1], S.N. Popov [3]

[1] *Samara National Research University, 34 Moskovskoe Shosse, Samara, 443086, Russia*

[2] *Institute for Information Transmission Problems of the Russian Academy of Sciences, 19 Bol'shoi Karetnyi per., 127051, Russia*

[3] *Innopolis University, 1 Universitetskaya Str., Innopolis, 420500, Russia*

E-mail: [a] easts@mail.ru

The article presents a proof of technology solution for enterprise desktop grid (EDG) applications with complex dependencies between tasks. The EDG applications are in the demand in growing fields of computational science like data science, neuroinformatics, bioinformatics, and mathematical modeling of complex systems. The use of the EDG applications in these fields is attractive regarding minimum hardware costs and affordability, but it remains hard for both programming and deployment. To demonstrate the approach for solving this problem, a prototype application based on the block sorting algorithm is developed. Specific requirements for the proof of technology application include: (1) the use of idle computers; (2) running on heterogeneous equipment; (3) the simplicity of application components deployment; (4) the possibility of long-term fault-tolerant calculations; (5) the calculations with a large number of tasks and complex dependencies between them. The application development and deployment are discussed. An experimental study of distributed sorting of a large data array in an EDG environment shows how these requirements are met.

Keywords: distributed computing, enterprise desktop grid, actor model, microservice, block sorting algorithm

1. Introduction

Enterprise desktop grid (EDG) systems are extensively used for high-performance computing. Their advantage in comparison to dedicated cluster systems is the possibility of almost unlimited scaling of computing power and the amount of stored data. They flexibly utilize idle resources and provide fault tolerant computing. These features make desktop grids very attractive for computing in corporate networks, since there are many cases in which desktop grids can replace expensive cluster systems [1].

However, there is a problem with the special point of view on parallel computing in the enterprise desktop grids. The problem is that the complex calculation should be decomposed into a set of independent subtasks, and the correct order of their execution should be organized. For parameter sweep applications, when data parallelism is used, the decomposition is trivial. But the decomposition becomes difficult for other classes of parallel applications.

There are many different methods of calculations based on the cooperation of processes (for example, using MPI technology) that have been developed for high-performance computing on clusters. However, such methods are not suitable for computing within enterprise desktop grids, as they require non-trivial mechanisms to provide communication between processes, fault-tolerant computing, and balancing the computational load. To overcome this problem, a number of tools were introduced that allow one to describe calculations as a static data dependency graph or static control dependency graph [2,3].

The purpose of this study is to demonstrate the possibility of dynamic task generation for applications with complex task dependency graph in the EDG environment. This technology can potentially expand the range of EDG applications with applications from new areas of computational science, which are usually run on cluster systems. Important examples of such areas are data science, bioinformatics, neuroinformatics, and mathematical modeling.

In the paper we discuss the development of the block-sorting application for large data arrays and the implementation of the method for dynamic generation of task dependency graph. We show the usage of Everest [4] and Templet Web [5] systems to deploy the application in EDG environment. In the experimental research, we show the possibility of performing long-term calculations with a large number of interdependent tasks for the proposed method of building and deploying EDG application. In conclusion, the possibilities of applying the described method of calculations are considered.

2. The prototype application design

Application structure overview. We propose the following application design to implement calculations with complex dependencies between tasks. The application consists of three loosely connected components. The orchestrator (the first component) forms a sequence of tasks. The execution semantics of the orchestrator is based on the model of Hewitt actors [6]. Its code structure represents a set of interconnected microservices [7]. To maintain the code structure, we use a markup language and a special preprocessor named Templet [8]. This component of the application is programmed and deployed through the Templet Web system. The Everest platform [4] (the second component) is used to implement calculations based on the generated task flow. The Everest platform provides a REST API through which the orchestrator submits tasks for execution, receives information about the status of tasks, and receives results of execution. Finally, the third component is the Everest application. It is a small user program installed on the Everest platform. Everest platform completely manages the life cycle of these programs in the desktop grid, automatically performing necessary deployment using agent programs installed on compute nodes. Flexible configuration of the parameters for running Everest applications is accomplished via the web interface.

The architecture implements the separation between the policy (the first component) and the mechanism of computation (the second and the third component) for the EDG application. The architecture aims to fulfill the following requirements:

- The application should make the use of idle computers.

- The application should execute in a heterogeneous environment.
- The application should have a simple and rapid deployment.
- The application should support long term fault-tolerant computing.
- The application should manage a large number of interdependent tasks.

We have implemented this architecture to solve the problem of sorting a large data array divided into blocks. This is a new type of problem for the desktop grids, in particular, due to non-trivial ordering of subtasks in block merging. The subtasks form a complex dependency graph, if a number of blocks are large. This graph cannot be built manually. Instead, the graph is built at the rate of computation, at each event of completion of the block merge.

Block sorting algorithm. The statement of the sorting problem in the form of a sequential algorithm is simple:

```
for (int i = 0; i < N; i++) block_sort(i);
for (int i = 1; i < N; i++)
        for (int j = 0; j < I; j++) block_merge(j, i);.
```

But the problem is fairly representative, since sorting simulates operations associated with many application areas, including the detection of correlation between data elements, calculation of the frequency of data elements, and data reordering. Another advantage of the problem is the flexibility in changing its complexity (that depends on the parameter N of the sequential algorithm) and increasing the number of tasks (that is ~$O(N^2)$ of problem size) in experimental study.

Microservice structure of the sorting algorithm is shown in Figure 1.

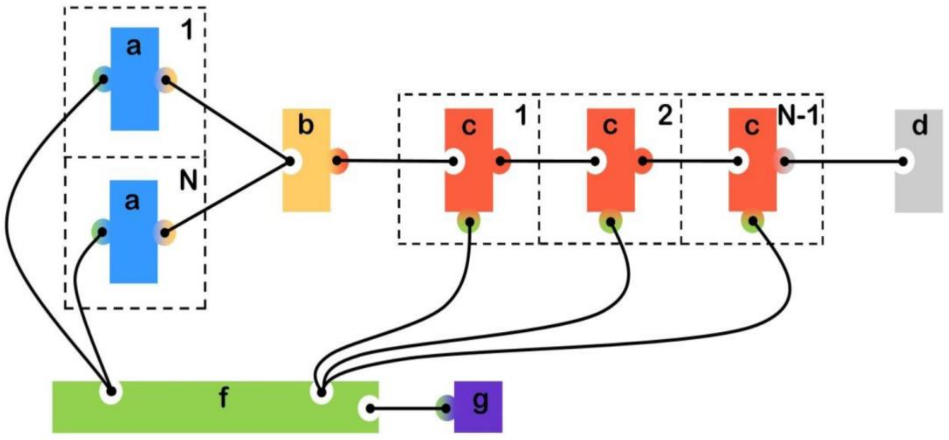

Figure 1. Microservice structure of the sorting algorithm

Microservices of type (a) perform the sorting of individual blocks. Microservice of type (b) starts the process of merging blocks at the end of sorting individual blocks. The management of merge operations is carried out with the help of microservices of type (c) as follows. Through a chain of microservices of type (c), a sequence of block numbers is transmitted from microservice of type (b) in ascending order (from 0 to N-1). If the block number gets into microservice of type (c) for the first time, this number is remembered. When the block number gets into microservice of type (c) for the second and subsequent times, the request is made to merge the block with one memorized early. Upon completion of the merge, this number is transmitted to the next (from left to right) microservice of type (c). Calculations are stopped when the flow of block numbers reaches microservice of type (d).

However, microservices of types (a) and (c) do not process the blocks by themselves. Instead, they request microservice of type (f) to start sorting or merging task. In turn, microservice of type (f) redirects the request to the Everest server via the REST API. Checking the status of the task is performed periodically at the request of microservice (g). If any of the executed tasks is completed, the microservice (f) notifies microservices of types (a) or (c) about it. Note that this structure provides the ability to perform multiple sorting tasks in parallel, as well as several block merging tasks.

3. Experimental study

According to the diagram in Figure 2, we have developed a distributed sorting application for the EDG environment and performed its testing. The orchestrator of the application (the first component) is implemented in the C++ programming language using the Templet markup preprocessor. We used the libcurl library to interact with the Everest server. The JSON library for modern C++ was used to form and parse commands in the JSON language when interacting with the Everest server. To perform sorting and merging, we have developed Everest applications in C++. The sorting and merging operations were performed using the standard C++ library. The compiled code for these applications was deployed on the Everest platform.

We used the following (see Figure 2) deployment of application components. The orchestrator was deployed on a dedicated virtual machine with Ubuntu operating system. The Vmware vSphere was used to control this virtual machine on the Sergey Korolev cluster of Samara University. The Everest server (https://everest.distcomp.org/apps) was used as the control node (the second component). Finally, to perform the sorting and merging tasks, we used desktop computers (Lenovo IdeaCentre Q190, Intel Celeron 1017U, RAM DDR3 4Гб, HDD 500GBб, MS Windows 7) in the local network of Samara University (the third component).

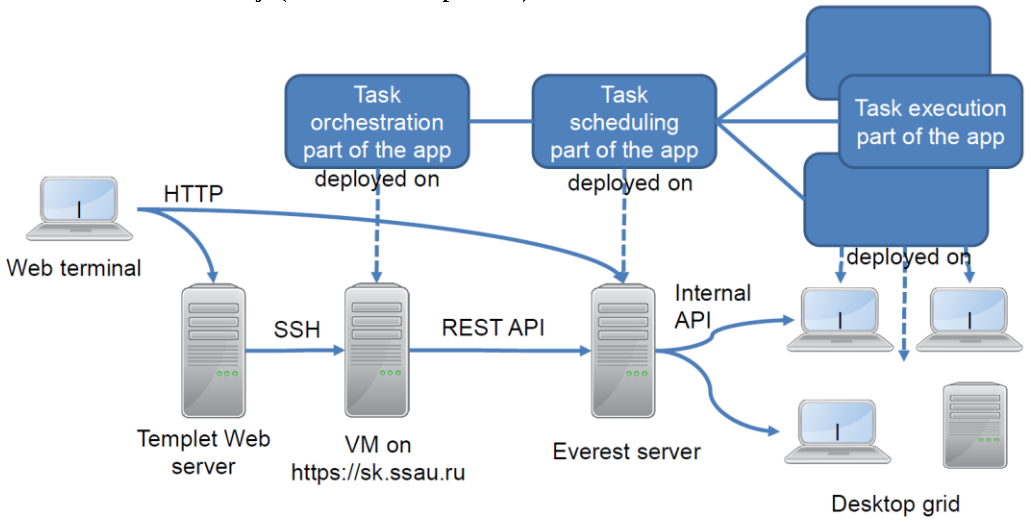

Figure 2. Deployment of application components

We conducted several experiments, which differ in the number of blocks of the sorted data array with a total size of 1,280 KB. The results are shown in Table 1.

Table 1. Load tests of block sorting application

Number of blocks	Block size, KB	Number of tasks	Execution time, s
2	640	3	31.18
4	320	10	83.44
8	160	36	315.08
16	80	136	1019.85
32	40	528	3755.36
64	20	2080	14580.80
128	10	8256	54076.76

The initial data array was splitted into separate files. Each file was uploaded to the Everest server, which managed the data transfer and execution of tasks on the desktop computers. When the sorting was completed, a special utility checked the correctness of the sorting.

4. Discussion

The results of the experimental study allowed us to verify the reliability of the developed application and the possibility of long (about 15 hours for 128 blocks) calculations. It can be seen that the size of the blocks significantly affects the speed of calculations. This is due to the fact that the amount of data transmitted through the control node (between the second and the third components) increases with the number of blocks. This problem can be solved by storing the data files within the corporate network and passing only links to these files inside the tasks. Also, the potential parallelism was not fully utilized in these experiments, because a small number of desktop computers has been used.

As a result, we have demonstrated that all the stated requirements for the application are met. We have ensured the simplicity of deployment using Everest platform and Templet Web. The fault-tolerance was provided by the internal mechanisms of the Everest platform and the VMware virtualization system. The use of microservices with actor execution semantics in the orchestrator design made it possible to form tasks dynamically depending on the results of previously launched tasks.

5. Acknowledgement

This work is partially supported by the Russian Foundation for Basic Research (grant #18-07-00956), and by the Ministry of Education and Science of the Russian Federation in the framework of the State Assignments program (Project No. 9.1616.2017/4.6).

6. Conclusion

We demonstrated the technology for enterprise desktop grids that enables dynamic generation of tasks during the computation. This technology extends the class of computing tasks that can be solved on the enterprise desktop grids. In the future, we plan to introduce implicit interaction between the Everest platform and Templet Web, which will allow researchers to code applications without the help of a system programmer.

References

[1] Ivashko E. E. Enterprise desktop grids // Program Systems: Theory and Applications, 2014, No. 1, p. 19.

[2] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. Workflows for E-Science: Scientific Workflows for Grids. Springer Publishing Company, Incorporated, 2014, pp. 1-523.

[3] I. Lazarev, O. Sukhoroslov. On Development of Workflow Management Service for Distributed Computations // Distributed Computing and Grid-Technologies in Science and Education: Proceedings of the 3rd Intern. Conf., 30 June - 4 July 2008, Dubna, Russia.

[4] Sukhoroslov O., Volkov S., Afanasiev A. A Web-Based Platform for Publication and Distributed Execution of Computing Applications // 14th International Symposium on Parallel and Distributed Computing (ISPDC). IEEE, 2015, pp. 175-184.

[5] Vostokin S.V., Artamonov Y.S., Tsaryov D.A. Templet Web: the use of volunteer computing approach in PaaS-style cloud // Open Engineering, 2018, Vol. 8(1), pp. 50-56.

[6] Hewitt C. What is computation? Actor model versus Turing's model //A Computable Universe: Understanding and Exploring Nature as Computation, 2013, pp. 159-185.

[7] Dragoni N. et al. Microservices: yesterday, today, and tomorrow //Present and Ulterior Software Engineering, Springer, Cham, 2017, pp. 195-216.

[8] Vostokin S.V. Templet: A markup language for concurrent actor-oriented programming // CEUR Workshop Proceedings, 2016, Vol. 1638, pp. 460-468.