Implementation of frequency analysis of Twitter microblogging in a hybrid cloud based on the Binder, Everest platform and the Samara University virtual desktop service

Sergey Vostokin
Samara National Research University
Samara, Russia
easts@mail.ru

Irina Bobyleva
Samara National Research University;
Joint Stock Company Space Rocket Center Progress
Samara, Russia
ikazakova90@gmail.com

***Abstract*—The paper proposes the architecture of a distributed data processing application in a hybrid cloud environment. The application was studied using a distributed algorithm for determining the frequency of words in messages of English-language microblogs published on Twitter. The possibility of aggregating computing resources using many-task computing technology for data processing in hybrid cloud environments is shown. This architecture has proven to be technically simple and efficient in terms of performance.**

Keywords—parallel computing model, algorithmic skeleton, parallel algorithm, actor model, workflow, many-task computing

I. Introduction

Advances in artificial intelligence and data processing technologies have contributed to the intensive development of tools for building scientific applications. One of the popular tools in this area is the JupyterLab integrated development environment [1]. An important functional feature of the JupyterLab is the ability to perform calculations on a specially configured computer remotely. The user can interact with the JupyterLab environment on any other computer over the Internet in a web browser. Cloud services, in particular the Binder service [2] used in this study, magnify this feature. The Binder service allows you to automatically prepare the necessary configuration of the application with the JupyterLab environment as a docker image [3] and run the application on a free virtual machine, for example, in the Google Cloud. Thus, all control over the application is carried out completely through a web browser, therefore there is no need for a specially configured physical computer.

This method of working with the application is convenient for demonstration purposes, teaching students, solving computationally simple tasks, and collaborative research. However, with ease of implementation, the application is limited in computing resources. Obviously, when deploying the application in free computing environments, the user should expect a minimum quota for memory and processor time. Also the user is limited in the possibility of networking and other rights on the free virtual machine. These limitations make it difficult or even impossible to organize high-performance data processing in the way standard for JupyterLab.

The article discusses the approach to overcome the described limitations. The essence of the approach is that the calculations are not performed on the machine where JupyterLab is installed. Instead, only control part of the many-task application is installed together with JupiterLab. The control part of the application communicate with the auxiliary cloud platform Everest [4] and uses its computing part deployed to other cloud systems. The computing part includes the required number of virtual machines. This solves the problem of obtaining necessary processor resources and memory.

In the research (1) we propose the distributed application architecture and deployment method; (2) we studied the possible speedup of calculations for a given application while solving an actual problem from the field of data processing.

II. Method for frequency analysis in a hybrid cloud environment

As a model problem for assessing the effectiveness of the proposed distributed application architecture, the problem of calculating the frequency of words in a text array was chosen. The source text array for analysis was taken from a weekly dump of all messages transmitted through the Twitter microblogging service. We chose the frequency analysis problem, since it has many practical applications, for example, for market analysis [5] or clustering [6]. On the other hand, this problem is important in our study to assess the data volume that can be processed in a reasonable time. It is also important to estimate typical processing and sending times of text data over the network, which affects the efficiency of calculations.

The specifics of computing in a hybrid cloud is that application components are limited in the possibilities of interaction: (1) they communicate not directly, but only through some intermediate service; (2) they are connected by communication channels with medium bandwidth and/or latency. These features require the organization of data processing in a special way.

The adequate models for organizing computations in a hybrid cloud is a task model of computations [7]. According to the task model of computations, calculations are understood as the periodic launch of non-interacting tasks. The reason for replenishing the set of tasks launched at a certain moment of observation may be the completion of another task.

Our implementation of the frequency analysis algorithm is based on launching two types of tasks from the control part of the application. The task of the first type receives a Twitter dump in the form of a JSON file, extracts English text messages from it, splits them into words, and forms an alphabetically ordered list of words with their local frequency for the given dump. This list is stored in a file. A task of the second type uses two files built by tasks of the first type. It combines the files into one ordered list of words and frequencies, then splits this list in half. Further, the content of the first file is replaced by the first half, and the contents of the

second file is replaced by the second half of the ordered list. Note that the common method of merging files in one file also solves the problem of frequency analysis, but the size of the resulting files is constantly increasing during the processing. Partitioning of files is necessary, since tasks should operate on not large files in order to facilitate file transfer over the network.

It is easy to notice that the periodic application of the second type task to individually ordered files (built by tasks of the first type) will order the entire file array. For example, if the total number of files in array is N, then procedure

for all i from 1 to N-1 execute
for all j from 0 to j < i perform
the 2d type task for file j and file i
end of i loop

orders the file array. The work of the control part of the application consists in the parallel execution of this procedure. In the experiments we applied a small optimization to reduce the diameter of the task dependencies graph and speedup the calculations.

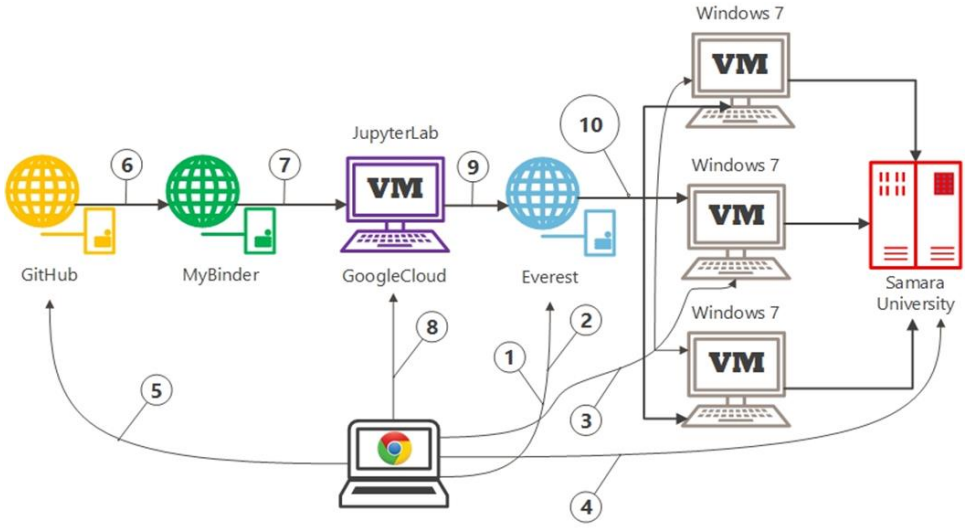

Fig. 1. Application deployment and execution diagram.

Parallelization of the task invocation procedure can be performed as explained in [8]. The idea of the procedure is to simulate the behavior of a computing part of the application on the JupyterLab virtual machine using the master-worker actor system. The master actor decomposes the problem into tasks of the first and second type and distributes the tasks to workers. The worker actors accept tasks, pass completion message back to the master and request more work. But instead of performing tasks on JupyterLab machines, the worker actors send tasks for the initial processing of Twitter dump files (task of the first type) and pairwise consolidation of received files (task of the second type) to the Everest server. The actor system was implemented in C ++ programming language using the Templet parallel computing system [9].

Optimization is based on the observation that the task dependencies graph for the parallel version of the procedure is asymmetric. Although we cannot change the total number of vertices in the task graph (it is equal to N(N-1)/2), we can make it symmetrical with a smaller diameter. To do this, we use the recursive procedure for listing the pairwise merging tasks instead of the original iterative procedure. The detailed descriptions of the parallel algorithm for data processing and its optimization can be found in [10].

III. Deployment and Operation of the Frequency Analysis Application

The main feature of the proposed distributed application architecture is the optimization for computing in hybrid cloud environment. Such a computing environment is built on the basis of free computing resources available in public and academic cloud systems. In addition, the application is fully deployed and launched through a web browser.

Let's take a look at the steps to deploy and run the application shown in Figure 1. The steps explain the implementation of the main architectural features.

Step 1. Registration of computing resources of the application on the Everest platform. Obtaining access tokens for agent programs through the web interface of the Everest platform.

Step 2. Installing application components for the primary processing of Twitter dump files (a task of the first type) and pairwise merging of the resulting files (a task of the second type). This installation is performed through the web interface of the Everest platform.

Step 3. Running Windows 7 virtual machines in the corporate cloud of Samara University. Installing agent programs on them using the access tokens obtained in Step 1. Verifying the activity of agent programs through the web interface on the Everest platform.

Step 4. Uploading the data set as Twitter text files in JSON format to a file server in the corporate cloud of Samara University. This upload can be performed through one of the virtual machines that you started in Step 3.

Step 5. Launching the control part of the application (so called the application orchestrator) from the GitHub code repository via the web interface.

Step 6. Automatically access the Binder service (after completing Step 5) to build a docker container with an application orchestrator running in the JupyterLab environment.

Step 7. Deploy the docker container (from Step 6) in the Google Cloud. Returns the link to the web interface of the

application orchestrator to the web terminal of the application user.

Step 8. Launch the application orchestrator by the user via the web interface obtained in Step 7. Start automatic data processing.

Step 9. The application orchestrator sends commands to launch next tasks to the Everest platform server and polls the status of previously launched tasks.

Step 10. The Everest platform server distributes tasks for execution to free virtual machines through resource agent programs (installed in Step 3). The calculation ends when N tasks of the first type and N(N-1)/2 tasks of the second type are started and completed, where N is the number of files in the information array.

Note, if a series of experiments to determine word frequencies is performed, then the computing part of the application (Steps 1-4) is set up once. For the second and subsequent manual launches, only Step 8 is performed.

The initial launch of JupyterLab is fully automatic. The sequence of steps, starting from step 5 and ending with step 7, is initiated by the user of the application when he / she presses a special button in the graphical interface of GitHub. All basic actions, including resolving application dependencies, building an image in the form of a Docker container, searching for a free virtual machine in public clouds (currently Google Cloud, OVH, GESIS laptops and Turing Institute clouds) are fully automated by MyBinder platform. The developer only needs to follow the special format for the git repository of the application orchestrator.

The restriction on the method of deploying the computing part of the application in the test implementation (Steps 1-4) is access to the shared file system from all the virtual machines deployed in Step 3. However, it is technically possible to transfer files directly from the application orchestrator through the Everest server to worker virtual machines. Also one can use the IPFS distributed file system client program for file transfer [11]. These methods were not considered in this study.

IV. Performance measurement of the frequency analysis application

The purpose of the experiments was to verify the functionality of the application and to confirm the effectiveness of calculations using the proposed distributed application architecture. This architecture is appropriate to consider effective, if the calculations are completed in a reasonable time and the use of several virtual machines in the computing part of the application leads to faster calculations. The results of the experiment are shown in Table 1.

For the experiments, we used a fragment of a 5.88 GB data array collected in [5]. The array consisted of 10 files. The size of the input JSON files ranged from 524 MB to 849 MB. The result of processing was stored in an array of 10 text files with a total size of 1.83 MB. 148885 words were found (including word forms and neologisms). The size of the resulting files ranged from 158 KB to 223 KB. The resulting files consisted of entries in the format:

<word 1>
<total number of repetitions of a word 1 in 10 files>
<CR>

<word 2>
<total number of repetitions of a word 2 in 10 files>
<CR>

<word M>
<total number of repetitions of a word M in 10 files>
<CR>.

TABLE 1. THE SPEEDUP OF DATA PROCESSING ON A HYBRID CLOUD COMPUTING ENVIRONMENT

Number of files, N	Sequential processing time, s	Parallel processing time, s	Speedup estimated by the three measurements
2	79.226	53.7776	1.29
	68.4562	59.9446	
	79.0123	63.2141	
3	145.436	85.1115	1.54
	114.838	94.4716	
	140.661	86.5152	
4	213.356	106.638	1.90
	183.547	110.084	
	212.875	104.538	
5	304.123	140.8	2.14
	245.916	140.235	
	347.438	140.065	
6	468.526	171.275	2.38
	330.048	175.948	
	458.023	180.496	
7	562.377	198.964	2.73
	409.327	192.46	
	599.395	185.208	
8	670.223	204.349	2.69
	484.539	220.411	
	586.413	223.25	
9	736.98	247.328	3.20
	763.388	239.363	
	845.347	246.543	
10	889.71	273.798	3.40
	936.003	276.348	
	983.331	274.694	

The entries were sorted through the entire set of 10 files (words on ‘a’ were stored in the first file, and words in ‘z’ were stored in the last file).

We carried out a series of experiments. The number of processed files and the number of virtual machines on which the computing part of the application was run changed from 2 to 10. The observed time for completing tasks of the first type in all experiments varied from about 24 to 36 seconds, and the time for completing tasks of the second type ranged from about 6 to 26 seconds. The maximum 3.4x speedup (relative to the version with one virtual machine in the computing part of the application) was achieved when processing 10 files on

10 virtual machines. The sequential processing of the data array on one virtual machine took 983 seconds in the worst case (~16 minutes). The parallel processing using 10 virtual machines took about 270 seconds (~4.5 minutes). The absolute reduction in processing time was approximately 11.5 minutes. Thus, the studied distributed application architecture can be considered effective.

V. Conclusion

The paper proposes the architecture of a distributed data processing application for computing in a hybrid cloud environment built as a combination of free and academic cloud services. In a computational experiment to determine the frequency of words in Twitter messages, the possibility of speeding up the calculations was demonstrated, which proves the effectiveness of proposed architecture.

The practical advantage of the architecture is the possibility of high-performance data processing using only a web browser. This feature is great for demos, training, and collaborative research.

References

[1] B. Granger, C. Colbert and I. Rose, "JupyterLab: The next generation jupyter frontend," JupyterCon, 2017.

[2] P. Jupyter, M. Bussonnier, J. Forde, J. Freeman, B. Granger, T. Head, C. Holdgraf, K. Kelley, G. Nalvarte, A. Osheroff, M. Pacer, Y. Panda, F. Perez, B. Ragan-Kelley and C. Willing, "Binder 2.0 - Reproducible, interactive, sharable environments for science at scale," Proceedings of the 17th python in science conference, vol. 113, pp. 120, 2018.

[3] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," Linux Journal, no. 239, 2014.

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

[5] D.A. Vorobiev and V.G. Litvinov, "Automated system for forecasting the behavior of the foreign exchange market using analysis of the emotional coloring of messages in social networks," Advanced Information Technologies (PIT), pp. 416-419, 2018.

[6] I.A. Rycarev, D.V. Kirsh and A.V. Kupriyanov, "Clustering of media content from social networks using bigdata technology," Computer Optics, vol. 42, no. 5, pp. 921-927, 2018. DOI: 10.18287/2412-6179-2018-42-5- 921-927.

[7] P. Thoman, K. Dichev, T. Heller, R. Iakymchuk, X. Aguilar, K. Hasanov, P. Gschwandtner, P. Lemarinier, S. Markidis, H. Jordan, T. Fahringer, K. Katrinis, E. Laure and D. S. Nikolopoulos, "A taxonomy of task-based parallel programming technologies for high-performance computing," The Journal of Supercomputing, vol. 74, no. 4, pp. 1422-1434, 2018.

[8] S.V. Vostokin, O.V. Sukhoroslov, I.V. Bobyleva and S.N. Popov, "Implementing computations with dynamic task dependencies in the desktop grid environment using Everest and Templet Web," CEUR Workshop Proceedings, vol. 2267, pp. 271-275, 2018.

[9] S.V. Vostokin, "The Templet parallel computing system: Specification, implementation, applications," Procedia Engineering, vol. 201, pp. 684-689, 2017.

[10] S.V. Vostokin and I.V. Bobyleva, "Asynchronous round-robin tournament algorithms for many-task data processing applications," International Journal of Open Information Technologies, vol. 8, no. 4, pp. 45-53, 2020.

[11] J. Benet, "Ipfs-content addressed, versioned, p2p file system," arXiv preprint: 1407.3561, 2014.