Study of the event log method to organize fault-tolerant and self-balancing calculations in a hybrid environment

Исследование метода журнала событий для организации отказоустойчивых и самобалансирующихся вычислений в гибридной среде

S.V. Vostokin

С.В. Востокин

Samara National Research University, Samara, Russia

Самарский университет, Самара, Россия

The article presents an experimental study of an application for hybrid environments that uses a new method for synchronizing the states of its distributed components by sending special jobs to an event log node. The method differs from traditional grid applications, where a single control node creates many autonomous jobs to run on the grid. It allows the use of a programmer-friendly SPMD model. It also successfully addresses fault tolerance and load balancing issues, providing speedup with little overhead.

В статье представлено экспериментальное исследование приложения для гибридных сред, в котором применен новый метод синхронизации состояний его распределенных компонентов через отправку специальных заданий на узел журнала событий. Метод отличается от традиционных грид-приложений, где один узел управления создает множество автономных заданий для выполнения в гриде. Он позволяет использовать удобную для программиста модель SPMD. Также он успешно решает проблемы отказоустойчивости и балансировки нагрузки, обеспечивая ускорение с небольшими накладными расходами.

INTRODUCTION

Due to the growth in the amount of calculations in the field of big data processing and artificial intelligence, as well as in solving traditional problems of numerical modeling, there is a need for programs that can be deployed and run in hybrid environments consisting of an arbitrary set of network computing resources. Good examples of such resources are (a) volunteer computers, as in the BOINC platform [1] or other voluntary distributed computing projects; (b) idle corporate computers that are potentially available over the network to solve production problems; (c) idle computing nodes of high performance supercomputer or cluster systems [2, 3]; (d) free or low cost spot virtual machines from cloud providers [4].

The use of hybrid environments allows you to reduce the cost and achieve high performance. But the key issues when programming applications for hybrid environments are fault tolerance and load balancing. The specificity of the hybrid environment leads to the fact that the application itself, and not its computing environment, should solve the issues. For example, we cannot put on hardware virtualization in the cloud. The application component in a hybrid environment needs to quickly start computing on the newly connected resource. At the same time, a sudden or planned shutdown of a resource with an application component deployed on it should not lead to the failure of the entire application. As a result, a special application design is required.

The article presents an experimental study of the design in a hybrid environment. We will limit ourselves to performing many independent tasks to demonstrate the applicability of the method. First, we discuss the traditional many-task application and its alternative architecture under study, emphasizing the advantages of the latter. Second, we present a model problem and the results of simulation and load testing. Finally, we draw conclusions on the experiments and further work.

COMPUTING ARCHITECTURE

Figure (see Fig. 1) shows the traditional architecture of a distributed many-task application. In the figure, Everest cloud platform [5] is used as a middleware for the distributed application. In traditional application design, one dedicated node orchestrates computation on the other worker nodes. Worker nodes have Everest platform agents to communicate with the platform. The orchestrator node communicates with the platform by submitting jobs via the REST protocol.

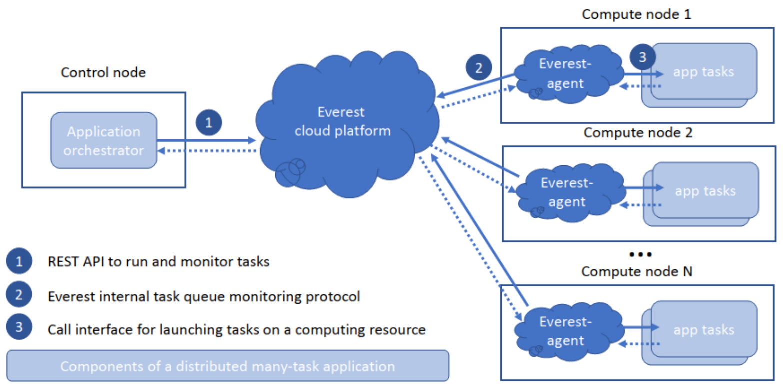
Control node
Application orchestrator
Everest cloud platform
Compute node 1
Compute node 2
Compute node N
Everest-agent
app tasks
1 REST API to run and monitor tasks
2 Everest internal task queue monitoring protocol
3 Call interface for launching tasks on a computing resource
Components of a distributed many-task application

Fig. 1. Traditional distributed application architecture (based on Everest middleware)

We propose to invert the computing architecture (see Fig. 1), which gives us the new architecture (see Fig. 2).

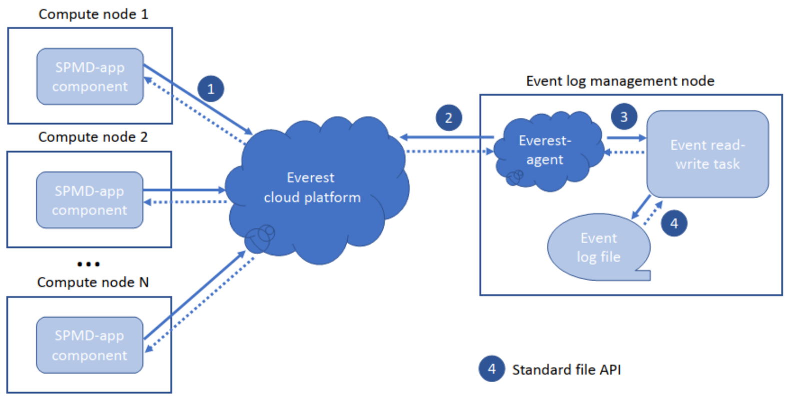

Fig. 2. Experimental architecture with event log (based on Everest middleware)

Application components reside on separate nodes. Each component executes the same code, moreover, it maintains the same state of computation as other components, implementing the SPMD programming model. Components periodically interact through the Everest platform using the REST protocol. The interaction is based on submitting a special job to Everest platform. The essence of the job is to write a local state change to the event log as an input and receive state changes from other nodes as an output. The job executes on the event log management node (see Fig.2). Only this node must have the Everest agent installed. The generic events read/write task component is used to access the event log file.

The computing architecture shown in Fig. 2 has the following advantages. From the point of view of computing configuration, this makes it convenient to quickly deploy the components of the SPMD application on non-dedicated computing resources, since no installation of a resource agent is required. In addition, the architecture is fault tolerant and allows application code to be migrated in real time. This is achieved by copying the event log file from one event log management node to another. In this case, the SPMD application component can be started or stopped on any node at any time of execution. The application components follow the pilot-job model [6].

SAMPLE APPLICATION AND EXPERIMENTAL RESULTS

The problem statement is the following. We have an arbitrary number of independent tasks. In the experiments it varies from 10 to 50 with a step of 10. The tasks must be solved in 10 processes or computing nodes. For each process, an individual random order is defined in which it will solve tasks. The calculation time of one task is determined. In experiments, it changes from 10 to 50 seconds with a step of 10 seconds.

The implementation guarantees the following conditions: (a) each task is processed by at least one process; (b) all processes agree on a common order in which task solutions will be obtained using the event log.

During the experiment, an assessment was made of the speedup of the calculation on 10 processes in the absence of communication overheads in simulation and in the presence of communication overheads in load testing. The results of the experiment are presented in Table 1.

Table 1. Experiment results: speedup when using 10 worker processes

Duration of task, sec	**Number of tasks, pcs.**				
	10	**20**	**30**	**40**	**50**
in the absence of communication overheads in simulation					
10-50	3.33333	4.0	5.0	5.71429	5.55556
in the presence of communication overheads in load testing					
10	1.23553	2.02957	3.34938	2.84712	3.19064
20	2.23945	3.44074	3.73296	4.97036	5.42627
30	3.0209	3.62881	4.55871	5.20853	5.09511
40	3.04433	3.71611	4.66365	5.29505	5.21234
50	3.1375	3.77058	4.72567	5.40605	5.28163

The simulation experiment is implemented using the Templet SDK [7]. Programming language is C++ with the Cling interpreter. Development environment is JupyterLab notebook, cloud deployment is done on Binder [8]. The code for load testing is implemented in C++ using GCC compiler. Middleware is the Everest platform of IITP RAS [5]. The libcurl implements communication with the platform.

The deployment was carried out on a virtual machine in OVHcloud using the Binder service. The event log process and computing processes were deployed on the same virtual machine to create the potentially largest load on the communication system.

The experiments [9] show that the method of calculations based on the event log allows you to successfully solve the problem of fault tolerance, load balancing, and provides speedup of calculations. But the method has some limitations. This is an excess amount of computation, although it is not significant given the availability and low cost of computing resources in a hybrid cloud.

CONCLUSIONS

We have presented the event log method and experimental study of the corresponding sample application, which shows its applicability for computing in a hybrid environment. In future work, we plan to adapt the method for applications with a dynamically generated set of dependent tasks and eliminate speedup limitations in the current implementation.

REFERENCES

1. *Anderson D.P.* BOINC: a platform for volunteer computing // Journal of Grid Computing. – 2020. – V. 18, no.1. – P. 99-122.
2. *De K. et al.* Integration of PanDA workload management system with Titan supercomputer at OLCF // Journal of Physics: Conference Series. – IOP Publishing, 2015. – V. 664, no. 9. – P. 092020.
3. *Korenkov V.V., Kondratyev A.O., Bondyakov A.S.* Interaction technology of jAliEn client and ALICE central services // Modern Information Technologies and IT-Education. – 2019. – V. 15, no.3. – P. 602-610.
4. *Varshney P., Simmhan Y.* AutoBoT: Resilient and cost-effective scheduling of a bag of tasks on spot VMs // IEEE Transactions on Parallel and Distributed Systems. – 2018. – V. 30, no.7. – P. 1512-1527.
5. *Volkov S., Sukhoroslov O.* Simplifying the use of clouds for scientific computing with Everest // Procedia computer science. – 2017. – V. 119. – P. 112-120.
6. *Turilli M., Santcroos M., Jha S.* A comprehensive perspective on pilot-job systems // ACM Computing Surveys (CSUR). – 2018. – V. 51, no.2. – P. 1-32.
7. *Vostokin S., Popov S., Sukhoroslov O.* Experience in organizing flexible access to remote computing resources from JupyterLab environment using technologies of Everest and Templet projects //CEUR Workshop Proceedings. – 2021. – V. 3041. – P. 558-561.
8. *Ragan-Kelley B. et al.* Binder 2.0-Reproducible, interactive, sharable environments for science at scale // Proceedings of the 17th python in science conference. – F. Akici, D. Lippa, D. Niederhut, and M. Pacer, eds., 2018. – P. 113-120.
9. Sample application and experimental results [Electronic resource]. Available at: https://github.com/the-templet-project/templet/tree/master/samples/blchsym (accessed 2.09.2023).