SUPPORTING EFFICIENT EXECUTION OF MANY-TASK APPLICATIONS WITH EVEREST

O.V. Sukhoroslov

Centre for Distributed Computing, Institute for Information Transmission Problems, Bolshoy Karetny per. 19 build.1, Moscow, 127051, Russia

E-mail: sukhoroslov@iitp.ru

Distributed computing systems are widely used for execution of loosely coupled many-task applications. There are two important classes of such applications. Bag-of-tasks applications, e.g., parameter sweeps or Monte Carlo simulations, represent a set of independent tasks. Workflows, which are used for automation of complex computational and data processing pipelines, consist of multiple tasks with control or data dependencies. The paper discusses the common problems related to the efficient execution of such applications on distributed computing resources and the relevant solutions implemented within the Everest platform.

Keywords: distributed computing, many-task applications, bag-of-tasks, workflow, Everest platform

1. Many-task applications

Many-task applications are loosely-coupled parallel applications consisting of potentially large number of computational tasks which can be executed more or less independently. There are two important classes of such applications widely used in science and technology. *Bag-of-tasks* or embarrassingly parallel applications have no data or control dependencies between the tasks. The typical examples are parameter sweeps, Monte Carlo simulations, image rendering. *Workflows* consist of multiple tasks with control or data dependencies between them. Such applications, which are commonly represented as directed acyclic graphs (DAG), are used for automation of computational and data processing pipelines consisting of multiple steps. While the workflow steps are usually executed independently, there is another special class of applications which require some form of cooperation between the running tasks. For example, the distributed implementation of the branch-and-bound method require the exchange of incumbent values between the tasks concurrently processing different subtrees.

While the many-task applications are naturally suited for execution on distributed computing resources, there exists a number of challenges related to the efficient execution of such applications such as management of a large number of tasks, accounting for dependencies between tasks, implementing coordination and data exchange, use of multiple independent computing resources, task scheduling, accounting for local resource policies and dealing with failures. The rest of the paper provides an overview of solutions to these problems implemented within the Everest platform.

2. Everest platform

Everest [1-3] is a web-based distributed computing platform which provides users with tools to publish and share computing applications as web services. The platform also manages the execution of applications on remote computing resources. Everest implements the PaaS model by providing its functionality via remote web and REST interfaces. A single instance of the platform can be accessed by many users in order to create, run and share applications with each other.

Instead of using a dedicated computing infrastructure, Everest performs the execution of applications on external resources attached by users. The platform supports integration with standalone servers, clusters, grid infrastructures, desktop grids and clouds [3-5]. The developed agent runs on a resource and acts as a mediator between it and Everest enabling the platform to submit and manage tasks on the resource. The agent performs routine actions related to staging of input files, submitting a task, monitoring a task state and collecting the task results.

Applications running on Everest follow a common model which natively supports many-task applications. An application has a number of *inputs* that constitute a valid request to the application and a number of *outputs* that constitute a result of computation corresponding to some request. Upon each request Everest creates a new *job* consisting of one or more *tasks* generated by the application from the job inputs. The tasks are executed by the platform on computing resources specified by a user. The dependencies between the tasks are currently managed internally by the applications. The results of completed tasks are passed to the application and are used to produce the job outputs or new tasks if needed. The job is completed when all its tasks are completed. The described model is generic enough to support all classes of many-task applications described in Section 1.

Everest implements a number of tools and mechanisms for execution of different types of many-task applications described below.

3. Execution of bag-of-tasks applications

Everest includes a general-purpose service for execution of bag-of-tasks applications called ParameterSweep [6-7]. Parameter sweep applications (PSA) require a large amount of computing resources in order to run a large number of similar computations across different combinations of parameter values. At the core of the service is a declarative format for describing a parameter sweep experiment, which contains parameter definitions and other directives that together define rules for

generation of parameter sweep tasks and processing of their results by the service. This approach aims to solve a common problem faced by researchers trying to use general-purpose distributed computing tools for running PSAs. Namely, a user have to implement custom programs for generating individual tasks comprising PSA and processing their results. The use of declarative description enables users to minimize or completely avoid such programming work, thus increasing the productivity and accessibility of the developed service. While originally aimed at PSAs, the service can be used for running arbitrary bag-of-tasks applications.

The service implements generation of tasks according to the provided description. To support the execution of applications with a large number of tasks, the tasks are generated dynamically based on the amount of available resource slots. Since tasks often share common input files, the input data transfer is optimized by caching of common input files by the Everest agent.

4. Execution of workflows

The most natural way of running workflows with Everest is by means of composition of existing Everest applications. In this case, the steps of workflow correspond to invocations of individual applications. To support automation of repetitive tasks, application composition and integration, Everest implements a REST API. It can be used to access Everest applications from any programming language that can speak HTTP protocol and parse JSON format. However REST API is too low level for most of users, so it is convenient to have ready-to-use client libraries built on top of it. For this purpose a Python client library [8] was implemented.

The Python client library implements the nonblocking semantics similar to the dataflow paradigm, which makes it simple to describe workflows without requiring a user to implement the boilerplate code dealing with waiting for tasks and passing data between them. This approach also implicitly supports parallel execution of independent tasks. Finally, the use of a general-purpose programming language instead of a declarative workflow description language provides maximum flexibility by enabling users to embed any additional processing logic.

The workflow driver implemented in Python using the client library can be executed on the user machine or published on Everest as a new application. In the latter case, the platform automatically passes the credentials for accessing the required applications on behalf of the user submitted the workflow to the driver. This enables reuse and sharing of workflows as applications among Everest users.

While the described approach is generally flexible and convenient, it does not allow passing the complete task graph to the platform to enable scheduling optimizations. Since the dependencies between the tasks are managed externally by the driver program, only the tasks ready to run are dynamically submitted to Everest. To overcome this limitation, an experimental general-purpose service for execution of workflows similar to the ParameterSweep service described in Section 3 is being developed. The service will execute a workflow described in YAML format inside a job encapsulating the corresponding DAG and an internal task scheduler.

5. Raw job mechanism

In some cases, it is desirable to flexibly manage application tasks during the application execution, for example to dynamically add new tasks based on the results of already completed tasks. The ParameterSweep service (Section 3) supports only a static set of tasks defined before the execution. The Python client library (Section 4) supports dynamic tasks but only on the level of invocations of separate Everest applications. To provide a general-purpose solution for such cases, a low-level mechanism has been implemented in Everest. This mechanism allows a platform client to submit the so called raw job and then manage the execution of tasks within this job via a WebSocket interface. This interface is similar to the one provided by the Everest agent and allows the client to send commands (e.g., submit or cancel a task) and receive notifications (e.g., change of a task state). Initially the raw job contains no tasks, and the client can upload input files, submit new tasks to the job, download task results and so on. The raw job should be explicitly cancelled by the client when the computations are completed.

6. Task coordination

While in classical bags-of-tasks and workflows the tasks are executed independently, there is a special class of applications which require some form of cooperation between the running tasks. A typical example is the distributed implementation of the branch-and-bound method for solving optimization problems where the tasks are concurrently processing different subtrees [9]. For efficiency reasons, it is crucial to exchange the locally found incumbent values between these tasks.

To support such class of applications, a lightweight coordination mechanism is implemented in Everest. This mechanism is based on the notion of shared variables similar to tuple spaces and blackboard models. It allows application tasks to write and read values of arbitrary named variables by exchanging messages with Everest. The values of variables are stored in the platform and represent an analogue of shared memory accessible to all tasks within a job. When the value of a variable is updated by some task, the new value is automatically forwarded by Everest to all other tasks. Thus the implemented mechanism can also be used as a lightweight publish-subscribe service. The implementation details can be found in [9].

7. Task scheduling

The application tasks are executed by Everest on computing resources specified by user. The efficiency of application execution, i.e. the execution time, critically depends on the methods used for task scheduling [10]. Everest implements a two-level scheduling mechanism that allows to plug-in different task scheduling algorithms. The platform-level job scheduler, which is periodically invoked with information about current jobs and resource states, fairly distributes the available resource slots among the jobs. Each job encapsulates an application-level task scheduler, which is invoked to select the tasks for running on provided resources. The default task scheduler implements a simple algorithm, which greedily assigns unscheduled tasks to available resources in decreasing order of their performance. If there are some jobs with unscheduled tasks left in the end of the scheduling cycle, the job scheduler also initiates the allocation of additional slots for on-demand resources, such as grid and cloud.

The described two-level mechanism allows the use of specific task schedulers for different types of many-task applications. Two experimental schedulers are developed for bags-of-tasks and workflows, which are based on MaxMin and DLS algorithms. These algorithms can produce more efficient schedules than the default scheduler, but require the estimates of task execution and data transfer times. Currently, these estimates are computed based on the information about already completed tasks (waiting time, execution time, data transfer time) which is collected by the platform.

8. Miscellaneous

The other features that are essential for efficient execution of many-task applications include accounting for local resource policies and automatic recovery of failed tasks.

The limit on the maximum number of jobs per user imposed by an HPC cluster administrators may not allow to fully utilize the resource when running a single cluster job per Everest task. An advanced adapter for Slurm cluster manager has been developed which allows to solve this problem by packing multiple Everest tasks in a single cluster job.

When dealing with failed tasks, the platform distinguishes between the critical (non-zero exit code) and recoverable (resource is disconnected, data download/upload error) faults. In the latter case, the task is automatically retried multiple times, and the resources with many failures are blacklisted. The application developer can optionally enable task retries after a critical error. To account for temporary network failures, which can happen between Everest and resources, the tasks running on the disconnected resource are not rescheduled immediately to avoid wasting compute time.

9. Acknowledgement

The work was supported by the Russian Foundation for Basic Research (RFBR), grant #18-07-00956 "The development of methods and tools for distributed computing on the base of Everest platform".

10. Conclusion

The paper presented ready-to-use solutions for efficient execution of many-task applications, such as bag-of-tasks and workflows, in distributed computing environments implemented within the Everest platform. A distinctive feature of the presented approach is the availability of these solutions on the principles of cloud computing and PaaS model. The public instance of the platform [1] is available online for all interested users. Future work will focus on improving and extending the presented functionality, including the workflow execution service and application-level schedulers.

References

[1] Everest. http://everest.distcomp.org/

[2] 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.

[3] Sergey Smirnov, Oleg Sukhoroslov, and Sergey Volkov. Integration and Combined Use of Distributed Computing Resources with Everest // Procedia Computer Science, Volume 101, 2016, pp. 359-368.

[4] Sukhoroslov O. Integration of Everest platform with BOINC-based desktop grids // In: E. Ivashko, A. Rumyantsev (eds.): Proceedings of the Third International Conference BOINC:FAST 2017, Petrozavodsk, Russia, August 28 - September 01, 2017, pp. 102-107, http://ceur-ws.org/Vol-1973/paper13.pdf.

[5] Volkov S., Sukhoroslov O. Simplifying the Use of Clouds for Scientific Computing with Everest // Procedia Computer Science. Volume 119, 2017, pp. 112–120.

[6] ParameterSweep. https://everest.distcomp.org/apps/sol/psweep

[7] Volkov S., Sukhoroslov O. A Generic Web Service for Running Parameter Sweep Experiments in Distributed Computing Environment. Procedia Computer Science, Volume 66, 2015, pp. 477-486.

[8] Everest Python API. https://gitlab.com/everest/python-api

[9] Voloshinov V., Smirnov S., Sukhoroslov O. Implementation and Use of Coarse-grained Parallel Branch-and-bound in Everest Distributed Environment // Procedia Computer Science. Volume 108, 2017, pp. 1532-1541.

[10] Nazarenko A., Sukhoroslov O. An Experimental Study of Workflow Scheduling Algorithms for Heterogeneous Systems. In: Malyshkin V. (eds) Parallel Computing Technologies. PaCT 2017. Lecture Notes in Computer Science, vol 10421. Springer, Cham, 2017, pp. 327-341.