

Development and debugging of MPI programs on the environmental research computing portal in the mining region

Andrey Vlasenko^{1,1}, *Igor Sotnikov*¹, *Eugeniya Prokopenko*², and *Anton Demidov*³

¹Kemerovo State University, 650000 Kemerovo, Russia

²Kuzbass State Technical State University, 65000 28 Vesennya st., Kemerovo, Russia

³ RMIT University, Melbourne, Australia, 3024

Abstract. The article is devoted to the development of automated debugging software for parallel programs used in the analysis of the impact of mining on the ecology of the region using the MPI communication interface. The system combines the approaches of static analysis of the source code and automated control of correctness at runtime. Each approach implements a separate component of the system. An innovative idea embedded in the system is to determine the necessary checks in the patterns of erroneous behavior that the user can control by modifying the templates from the proposed set, adding their own or deleting ones that are not required. A system for describing patterns is described and several examples of formalizing errors are given. Some of the examples of errors are suitable for static analysis, while others are suitable for automated verification. The architecture and the general scheme of the automated debugging system on a computing cluster are presented. Another software project described in the article is an engineering computing portal. The portal is an expandable environment for a comprehensive study of the environmental problems of the region, integrating various services united by high-performance computing.

1 Introduction

In the modern scientific world, the accuracy of numerical models of the studied objects and phenomena continues to increase, which causes an avalanche-like increase in computations in HPC problems.

The creation of an information portal that combines information on mining and their further use into a single model is extremely important for the Kuzbass. The region produces more than 250 million tons of coal, 20 million tons of iron ore. Accordingly, the total emissions into the atmosphere exceed 1 million tons, into the water basin – 500 thousand tons [1].

In the mining region, in which enterprises that extract, enrich and process coal, iron ore, gold (Kuzbass, Western Siberia, Russia), it is important to create a single computing portal

¹ Corresponding author: vlasenko_a@list.ru

that links together all the scattered information about the environmental impact. This information comes from each mine, mine, processing plant, metallurgical and chemical plant. The analysis of this information will allow creating a computer model of the environmental impact of the entire industrial complex of Kuzbass, identifying the most dangerous sources of pollution and putting together administrative, economic and engineering environmental protection measures.

In turn, the accuracy of data processing directly determines the quality of decisions to improve the environmental situation, and increases the requirements for developers of the portal software complex [2].

It is the reason for the active growth in the scale of high-performance computing. This dynamic can be easily traced by regularly updated rating lists of supercomputers, for example, the Top500 world list and the Russian Top 50. The increase in HPC tasks is reflected in the increase in the number of threads and processes of the corresponding parallel programs.

Despite the fact that many researchers use off-the-shelf computing packages (OpenFOAM, ANSYS products, FlowVision, etc.), a significant share in the HPC segment is occupied by proprietary programs. This fact, in particular, is associated with the development of new numerical methods, the search for optimal software technologies for a specific task, etc. Various implementations of the MPI standard continue to remain the most common programming tools for supercomputers. One of the main problems with MPI programming is debugging. Even in the case of sequential programs, debugging sometimes takes up to 2/3 of the total development time. And when compiling parallel code, the developer may encounter many additional semantic errors caused by the interaction of asynchronous processes and the use of MPI technology. Clause 3 describes some of these errors, and a more detailed look at their classification can be found in [3, 4]. The observed tendency to increase the size of the tasks being solved only aggravates the situation with debugging.

To detect logical errors in parallel programs, such dialog debuggers as Rogue Wave TotalView [5] and Allinea Distributed Debugging Tool (DDT) [6] are traditionally used. However, as the number of processes in programs increases, the usefulness of dialog debuggers decreases. After all, to observe the behavior of each of one hundred processes, executing the program in steps, is an incomparably more difficult task than the behavior of ten. In practice, users debug the task on “small dimensions” (this term refers to the number of processes / threads, the number of iterations of loops, the dimensions of arrays, etc.), and then upload it to the target computing resource in full scale. But at the same time, new errors may appear that did not arise on “small dimensions”. In this case, dialog debuggers are almost useless.

The debugging system described in the article is able to function as a standalone software product. However, work is currently underway to integrate the system into the engineering computing portal [7, 8]. The purpose, structure of the portal, as well as the integration scheme of the debugging system are given below.

2 Materials and Methods

To detect logical errors in parallel programs, such dialog debuggers as Rogue Wave TotalView [11] and Allinea Distributed Debugging Tool (DDT) [9] are traditionally used. However, as the number of processes in programs increases, the usefulness of dialog debuggers decreases. After all, to observe the behavior of each of one hundred processes, executing the program in steps, is an incomparably more difficult task than the behavior of ten. In practice, users debug the task on “small dimensions” (this term refers to the number of processes / threads, the number of iterations of loops, the dimensions of arrays, etc.), and

then upload it to the target computing resource in full scale. But at the same time, new errors may appear that did not arise on “small dimensions”. In this case, dialog debuggers are almost useless.

The debugging system described in the article is able to function as a standalone software product. However, work is currently underway to integrate the system into the engineering computing portal [7, 8]. The purpose, structure of the portal, as well as the integration scheme of the debugging system are given below.

When constructing a system for automated debugging of MPI programs, the authors find it most productive to use a combination of several approaches, since various methods are suitable for searching for logical errors of various types. Suppose that the program contains deadlock, caused by the fact that two processes launched blocking communication functions, of which they must be participants together. Fig. 1 shows a fragment of such a program where the zero process calls the function of sending a message to the first (MPI_Send), and the first, among others, calls the group operation MPI_Gather. In this case, the zero process must also be a member of MPI_Gather.

The standard way to obtain information from the running processes of an MPI application is to use the profiling interface described in the MPI standard. This interface is used by almost all tools of automated correctness control, both developing projects and completed ones. Among them are Umpire, Marmot, MUST [9], Intel Trace Analyzer and Collector, which is the successor to the Intel Message Checker project [6]. The basic principle of using a profiling interface is as follows. The developer of the tool creates a library of MPI functions that is statically linked to the user program. While the user application is running, calls to MPI functions are intercepted by this library. Thus, the tool receives all the information about which process at which moment called the function and with what actual parameters. The tool processes the received data at its discretion, and in order for the action that the user expects to call this function to be performed, it is necessary to call the corresponding PMPI function at any place in the handler with exactly the same arguments as the original function.

For these reasons, when constructing our own automated debugging system, 2 methods were used: static analysis and automated correctness control.

3 Results and Discussion

The automated debugging system has a distributed architecture and is designed to work on a computing cluster (Fig. 1). On the head node (it is assumed that it is the compilation node), the source code of the user program is processed by the preprocessor, which is the first component of the system. The preprocessor introduces some additional operators and performs static analysis. To do this, from the presented set of templates (the path to the template directory is one of the input parameters of the preprocessor), he selects suitable for static analysis, which is determined by the number of processes installed in the unit, and the presence in the template text of certain keywords, which include, in particular, pseudo-function "SIZE". The preprocessor parses these templates and passes the user code, during which checks are performed for the occurrence of template situations. Information about the matches found is recorded in the resulting error file, where the name of the template is entered (field “Name”); names of source code files where an error was detected, with line numbers; The text of the lines of code that caused the error.

At the start of its work, the analyzer server parses text template files, filling out a list of templates with global errors and determining those MPI functions that need to be profiled based on the contents of the templates. The server analyzer passes the list of these functions to MPI processes. It also transfers those of the disassembled templates where the situation is described, for the determination of which information from only one process is sufficient

(local errors). During their work, processes call MPI functions, which are intercepted by the profiling library. If the called function is included in any patterns for local errors, then the service flow enters information about the parameters of this operation into special structures and performs an analysis. If a function is included in at least one of the templates describing a global error, then its parameters and code line number are passed to the analyzer server. That server stream, to which this MPI process is assigned, parses the received line, enters data into special program structures and performs analysis for compliance with the current situation patterns with global errors.

Computing services are provided directly through the portal using the CompServiceGen component. In addition to services, users and experts interact through the portal, cooperate with specialists, and search among ready-made solutions. Administrators are provided with such functions as managing portal users, registering and assigning rights to access computing resources and services.

To conduct their own computational experiments on high-performance resources, the Onlide subsystem is used. This subsystem is a development environment with which you can create your own code, compile and run it on available high-performance resources.

Computing services, subsystems, as well as external client systems interact with the portal API, which is displayed in the Portal API component. The following functions are implemented through the portal API:

1) User authorization (Auth API). Data about users, as well as about user groups, computing resources and software is stored in an LDAP database (LDAP DB).

2) Management of user file storage (Storage API). The repository contains input files and calculation results.

3) Interaction with software on computing resources (Software Manager API). The Software Manager API provides the following functions:

- start and a stop of implementation of programs on remote computing resources in the form of the identified tasks;
- tracking of a condition of the started tasks (in turn, it is carried out, it is complete);
- obtaining information on the used computing resources (the RAM, loading of the CCP) and about the started tasks (date and time of start of a task, date and time of start of the program, date and time of end of a task, restriction of resources).

The execution of the above functions is delegated to a specific Software Proxy implementation, depending on the computing resource. There are currently two versions of its implementation.

In the first version, the interaction is performed by setting up an SSH connection with a remote computing resource and running a special Python script to be placed on that resource. This script is responsible for running the specified program and tracking the resources used during its operation.

In the second release, the interaction is performed by establishing a TCP connection with the remote agent being deployed to the desired resource. The agent starts once and processes startup requests. This version is implemented primarily for Windows computing resources, due to the problem of running the SSH server on this operating system (OS).

The calculation services are described in the business process simulation language BPMN. In Fig 1, the BPMN Engine is a business process execution environment. For software BPM, this environment provides its own set of APIs (Engine APIs). The CompServiceGen component is responsible for generating a Web interface to start and track the status of the business process.

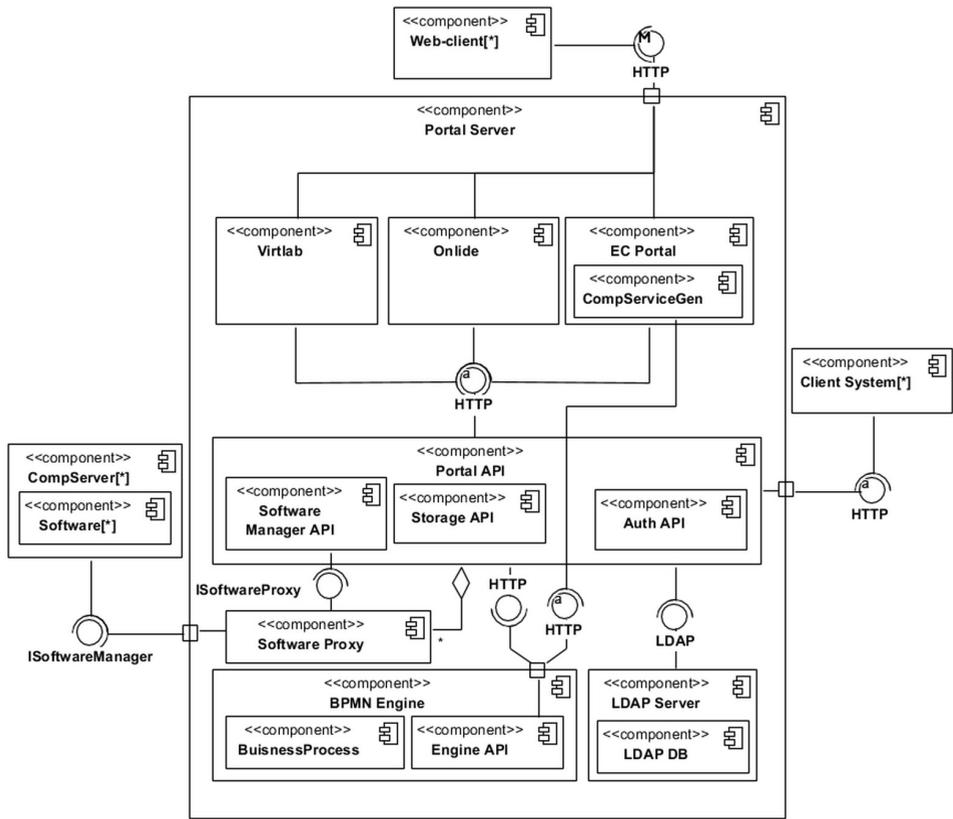


Fig. 1. Logical structure of the TPI in the form of a component diagram (UML2).

Camunda is selected as the execution environment for business processes. Camunda supports most of the elements that make up the business process, as specified in BPMN 2.0. In particular, they include an element that sends HTTP requests (for example, API requests) and an element that allows you to call another business process (sub-process). Based on the first element, two sub-processes are described.

The first sub-process is responsible for communicating with the user's file store. The repository contains input files and results of service execution (video files, images, model 3D, etc.). As input parameters, the element takes the path to the file or directory and the action to perform on it (create, delete, move).

The second sub-process is responsible for interacting with software on compute servers, namely, running and checking status - the program is running or has already been completed.

4 Conclusion

Currently, environmental protection in a region with intensive mining is impossible without the creation of computer models of the impact of industry on the environment. These models should be presented in a single information portal, and information should be processed with maximum accuracy.

The system presented in this paper is intended to help researchers developing MPI applications in such a time-consuming and fairly routine part as debugging. The obvious advantage of the system is that the user does not need any preliminary work to obtain the result. All you have to do is specify multiple parameters and one command to run your program under system control. At the end of the system operation, the user has a ready list of logical errors with indication of code lines that caused the found errors. The automated debugging system has been tested on application parallel applications. In particular, it was used in writing the tutorial. With the help of the system, the authors of the manual found errors in the sorting programs of the array using the method of chet-odd permutation and block multiplication of the matrix by the vector. The current integration of the system into the engineering computing portal will further simplify the researcher 's interaction with the system and increase the number of users.

References

1. M. Tyulenev, S. Markov, E. Makridin, Y. Lesin, V. Gogolin, E3S Web of Conferences, **105**, 02022 (2019)
2. A. Medvedev, I. Kislyakov, Ye. Prokopenko, M. Semenkina, K. Brester, E3S Web Conf., **105**, 03020 (2019)
3. J. Desouza, B. Kuhn, B. Supinski, Proceedings of the second international workshop on Software engineering for high performance computing system applications, **1**, 02101 (2005)
4. A.M. Gudov, S.Y. Zavozkin, I.Y. Sotnikov, IOP Conference Series: Earth and Environmental Science, **115**, 1 (2018)
5. A. Gudov, V. Perminov, Y. Filatov, L.H. Un, S. Zavozkin, I. Grigorieva, I. Sotnikov, CEUR Workshop Proceedings, **1**, 61-73 (2017)
6. J. Protze, T. Hilbrich, M. Schulz, B.R. de Supinski, 43rd International Conference on Parallel Processing Workshops, **1**, 206-215 (2014)
7. S. Siegel, Proceedings of the 14th European PVM/MPI Users' Group Meeting, **1**, 13-14 (2007)
8. A.Yu. Vlasenko, A.M. Gudov, Journal of Computer and Systems Sciences International, **56:4**, 708-720 (2017)
9. L. Williams, *Upgrade Adds Muscle to Debugger* (Oak Ridge National Laboratories, Oak Ridge, 2010)