# .NET Library for Seamless Remote Execution of Supercomputing Software

Alexander Tsidaev[1,2]

[1] Bulashevich Institute of Geophysics, Yekaterinburg, Russia
[2] Ural Federal University, Yekaterinburg, Russia
pdc@tsidaev.com

**Abstract.** Supercomputers are usually a Linux machines, that requires user to have additional skills in Linux. And while high-level programming for Linux and Windows is quite similar, all the infrastructure and tools are different. This paper presents a way to hide Linux interaction level from user by introducing the new library for .NET framework that handles SSH communication internally. Library has ability to run single-CPU/GPU processes as well as multicore/multi-GPU. Slurm task scheduler is supported for parallelized programs. Library also contains helper classes that allows user to retrieve source code from remote location (git and Subversion are supported) and compile program from sources (make is supported).

**Keywords:** asynchronous library · parallel computing · SLURM · remote supercomputing

## 1 Introduction

Supercomputer calculations are not exotic anymore. With growth of data, which should be processed during science research, there is a need to use supercomputers for programmers of any skill and background. And there is a problem. Supercomputers usually (if not always) works under Linux operation system, while most widespread operation system for desktop computers is Microsoft Windows. Thus, there is a big amount of programmers, who can develop software under OS Windows, but are not experienced Linux developers (or even users).

On the other hand, even Linux developers may experience difficulties working with remote supercomputer system using Secure Shell (SSH, the most common way to access Linux computers via network). This procedure includes many routine operations such as transmission of program source code to remote computer, compilation of source code, transmission of data to process and, at last, obtaining the results of calculations.

There are many different approaches to make easier remote access to supercomputers: REST API, Grid access or Web interface. This techniques are described in [1], where first two methods were acknowledged to be hard for user. This is because REST API or Grid control software are complex usually and,

thus, big additional efforts are needed to learn them. Akimova et al. in [2] develops Web interface method and presents a system that allows execution of precompiled programs. However, this solution does not allow any changes in used programs. While this is not a problem for regular user, for a developer this creates a barrier for developer who wants to modify algorithms.

So, there is a need in small tiny wrapper that should minimize count of required manual operations over SSH during supercomputer usage, but still allow developer to modify executed programs. In this paper author presents new library developed for .NET programming platform, which has these abilities. Library is written with full support of asynchronous features of .NET platform and C# language and allows easy convenient access to remote Linux system.

### 1.1 Library Structure

Fig. 1 presents class structure of the library. Base class is Remote Executor, which contains all functions for SSH communication and program execution.

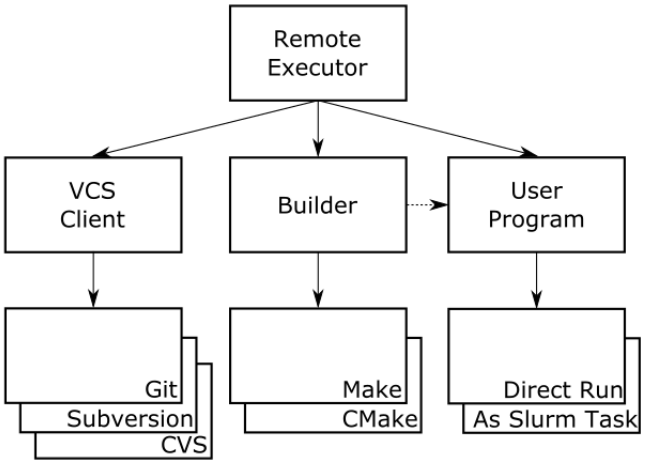


**Fig. 1.** Class structure of the library

Three types of classes are inherited from the base class. First one is VCS (version control system) Client classes, which are intended to get source code for use program from remote repository. Currently three providers are implemented: Git, Subversion and CVS. Password-based authentication schemes are supported.

Second group is Builder classes. This is an abstraction for a set of tools that can translate source code into executable. Make and CMake are supported.

Classes in third group are needed to execute user programs. There are two ways of execution. First is direct run, that could be used if target computer has

no ability to perform time-separation of tasks of different users or if executed program does not use any shared resource. Second way is execution using Slurm workload manager by sending program to the execution queue.

## 2 Usage

SSH communication is implemented in a fully asynchronous way by using async/await paradigm of .NET 4.0. All of the operations can be executed in parallel.

All SSH communication level is encapsulated in *RemoteExecutor* class. Usage example is provided below.

```
var exec = new RemoteExecutor("server.com", "user", "password");
var git = new GitClient(exec);
var path = await git.Instantiate("https://github.com/my/program");

var compiled = await new MakeClient(exec).Make(path);

if (compiled)
  Task.WaitAll(exec.Run("(cd program; ./program)"));
else
  Console.WriteLine("Compilation error");
```

This program first logs into Secure Shell server with provided credentials. Then, git repository with needed program is cloned from external location. After that, a compilation is executed with *make* command. And if the compilation succeeds, resulting program is executed.

As it can be seen from the example above, in this case there is no parallel execution. All created tasks are awaited before the execution continues. This is related to sequential character of operations: *make* cannot be executed until all source code is available, and program cannot be run until it compiled.

However, for more independent tasks the parallelization can be obtained in the same simple way:

```
var cmd1 = exec.Run("for i in `seq 1 10`; do echo $i; sleep 1; done");
var cmd2 = exec.Run("for i in `seq 1 5`; do echo $i; sleep 2; done");

var results = await Task.WhenAll(cmd1, cmd2);
```

Each command is simple bash loop, which takes 10 seconds to be executed fully. Both loops will be run simultaneously and program will still finish in 10 seconds. Result of each *exec.Run* operation is an instance of *CommandOutput* structure that contains return value, standard output and standard error streams (*stdout* and *stderr*). No mixing of output is performed even when commands are run in parallel because new SSH stream is created for any new command.

## 3 Execution of Parallelized Programs

The main goal of creation of this library was to provide an easy way for seamless execution of software, which had been written not just for Linux platform, but for Linux-based supercomputers. This includes programs working via MPI technological stack or using CUDA/OpenCL libraries for GPU computations. Such supercomputer systems in most cases are designed to be used by many users

simultaneously and, thus, contain some software for program execution scheduling. Library, which is described in this paper, contains support for one of most widespread schedulers, Slurm [3].

Class for execution of parallel program with Slurm scheduler is called *ParallelCodeExecutor*. Execution itself is performed with *Execute* method, which signature is

```
public async Task<CommandOutput> Execute(
  int    nodesNumber,
  int    gpuNumber,
  string partitionName,
  string program,
  params ProgramArgument[] arguments)
```

Here *nodesNumber* is number of CPU nodes that should be used by this execution, *gpuNumber* is the number of GPU devices, requested on each node, *partitionName* is a name of cluster partition, where program should be executed, *program* and *arguments* are executed program and its arguments.

Each element of *arguments* can be either fixed file name on remote system or local file, which will be uploaded automatically. *ProgramArgument* encapsulates this logic.

Return value is the *CommandOutput* structure with information about return code and *stdout*/*stderr*, the same for non-parallel executor. All streams redirection is handled internally and no user attention is needed to these details.

## 4 Conclusion

The presented software library covers full spectrum of tasks that exists in execution of custom supercomputing programs: source code obtaining, compilation and execution. Library helps developer to speed up the development process. Novice users are isolated from SSH layer, which may be not well-known for them. Experienced users can obtain development acceleration by automating routing operations.

Two main tasks can be solved using this library:

1. Reduce manual operations count during development process
2. Allow users to execute parallel programs in the same way as non-parallel. All logic related to execution queue, job identification and task completeness is encapsulated inside library

As the result, it is convenient to use described technique for iterative processes. If program to be executed should be run in a loop, each iteration of which depends on results of the previous one, then this process includes many manual operation such as file copy or command execution. Usage of the library can significantly reduce overall time needed to obtain the final result.

Library is distributed “as is” under LGPL 3.0 license. Source code can be obtained from https://github.com/atsidaev/parallexec. Since the structure can easily be extended by additional blocks, author welcomes everyone interested to submit patches and pull-requests.

## References

1. Dubenskaya, J., Kryukov, A., Demichev, A.: Some approaches to organizing of remote access to supercomputer resources. CEUR Workshop Proceedings Volume 1482: 1st Russian Conference on Supercomputing Days, 712–719 (2015)
2. Akimova, E.N., Misilov, V. E., Skurydina, A. F., Martyshko, M. P.: Specialized Web Portal for Solving Problems on Multiprocessor Computing Systems. CEUR Workshop Proceedings Volume 1513: Proceedings of the 1st Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists, 123–129 (2015)
3. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: Simple Linux Utility for Resource Management. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Volume 2862, 44–60 (2003)