J Supercomput (2007) 41: 119–131
DOI 10.1007/s11227-006-0029-9

Trustworthy remote compiling services for grid-based scientific applications

Yaohang Li · Daniel Chen · Xiaohong Yuan

Published online: 2 March 2007

Abstract Grid computing, which is characterized by large-scale sharing and collaboration of dynamic resources, is becoming an emerging computing platform on a global scale for data-intensive and computation-intensive scientific application. However, the complications of large-scale scientific computations and simulations harnessing massive computing resources are compounded by extensive heterogeneity in environments arising from "the Grid." Scientists and engineers lack an intuitive grid-based compilation tool, which has contributed to the difficulty of exploiting these diverse resources and developing their applications on the grid. While manual configuration of various toolkits simplifying the end-to-end completion of a job is adequate for a computational grid with a limited number of nodes, the compilation procedure becomes inefficient for a computational grid with an increasing number of heterogeneous computational service providers. On the other hand, a global-scale computational grid is a potentially untrustworthy computing environment. How to take advantage of the potentially untrustworthy grid resources to provide trustworthy computational services for large-scale scientific applications is another critical issue.

In this article, a remote compiling service for a heterogeneous computational grid is developed. In addition to running compilation tasks, the remote compiling service provides security enforcement and validation facilities, including intermediate value checking, secure source program submission, restricted compilation, and binary inspection, to support trustworthy compilation and execution of grid-based scientific applications. Overall, it is expected that our remote compiling services on the grid

Y. Li (✉) · D. Chen · X. Yuan
Department of Computer Science, North Carolina A&T State University, Greensboro, NC 27411, USA
e-mail: yaohang@ncat.edu

D. Chen
e-mail: dtchen@ncat.edu

X. Yuan
e-mail: xhyuan@ncat.edu

can tackle the heterogeneity problem of the grid and provide a secure, trustworthy, reliable, and state-of-the-art mechanism to develop grid-aware scientific applications.

Keywords Grid computing · Remote execution · Security · Trustworthiness

1 Introduction

Grid computing [13, 17] is an emerging technique in parallel computing for distributed resource sharing and problem solving on a global scale for data-intensive or computation-intensive applications. It focuses on dynamic and heterogeneous cooperation of "virtual organizations," innovative applications, and high performance [15]. The participation of scientists and experts in multiple disciplines also permits the implementation of scientific tasks, such as analyzing raw data streams, performing large-scale computations, designing cutting-edge technological products, collaborating on interdisciplinary research, and implementing a complicated scientific computing process [24].

Computational grids, in particular, are sophisticated conglomerates of a number of computational services that are analogous to public utilities: a scientist or engineer "flips a switch" by submitting a computational job description and the job is processed just as electricity flows on demand via the electrical grid. Many applications for scientific research, such as SETI@home [32], distributed.net [8], and folding@home [10], have demonstrated the utilization of the grid computing facilities with rudimentary success. Nowadays, the larger Grid-based research projects include the U.S. Department of Energy's Grid Portal Development Kit [30], CactusCode [18], the San Diego Supercomputing Center's Biology WorkBench [34], NEESit's earthquake research [29], the European E-Science Grid [9], and the Grid Projects and Deployment System [19]. Despite the attractive characteristics of grid computing, successfully exploiting the grid techniques for computational applications depends on overcoming a number of challenges stemming from properties of a computational grid such as dynamism, cross-domain physicality, heterogeneity, lack of intrinsic trustworthiness, reliability, and performance [13, 15].

In particular, the problem of heterogeneity in a computational grid is paramount because scientists and engineers have to face a daunting array of system architecture and operating environment choices on the computational grid. A well-known fact is that the executable binaries for one architecture/operating system combination cannot in general run on another node with a different architecture or a different operating system. When presented with such choices, often the decision in preference to particular platform stems from historical and administrative biases. Ideally, users of the computational grid should not be required to understand ramifications of system architectures and operating environments.

A more serious problem is, a computational grid is a potentially untrustworthy computing environment [33]. Many grid computing projects mentioned above have recorded various misbehaviors of grid computational service providers, malicious grid users, and malfunctioned grid applications. Indeed, untrustworthy grid components post significant threats to the correctness of scientific computations being carried out on the grid. How untrustworthy computational service providers can be composed to perform trustworthy computations becomes a grand challenge in grid computing community.

In [35], a remote compiler has been developed to produce executable binaries for a submitted task to run on heterogeneous machines in a Condor pool [25]. To support convenient usage of a large-scale computational grid with heterogeneous components, in this paper, we extend the functionalities of the remote compiler for Condor and present the development of a remote compiling service on the grid. First of all, the remote compiling service is a grid service that conveniently allows a naive grid user to produce reliable executable binary files on various platforms with various configurations, which can later be deployed to run on the computational grid. Moreover, our remote compiling grid service employs novel validation mechanisms to enforce trustworthiness of grid-based applications at compilation level. Overall, the remote compiling service is expected to provide a trustworthy, reliable, and state-of-the-art compilation support to grid-aware scientific applications development.

The remainder of this paper is organized as follows. We illustrate the architecture and the working paradigm of remote compiling service on the grid in Sects. 2 and 3, respectively. In Sect. 4, we discuss the trustworthiness issues and our implementation of trustworthiness enforcement mechanisms in the remote compiling grid services. Finally, Sect. 5 summarizes our conclusions and future research directions.

2 Architecture of remote compiling grid service

The remote compiling grid service is designed on top of the common grid services [14] provided by the Globus toolkit, [11, 16] and supplies facilities and services for scientific applications on the computational grid. The common grid services include GRAM (Globus Resource Allocation Manager), GIS (Grid Information Service), GSI (Grid Security Infrastructure), and GridFTP. GRAM is used to submit compilation tasks to distributed remote compiling service providers and to manage the execution of these tasks. GIS provides information services, viz., the discovery of the properties of remote compiling service providers as well as their underlying local compiler configurations. GSI offers security services such as authentication, encryption, and decryption for running compilation tasks on the grid. GridFTP provides a uniform interface for transporting source packages, executable binary files, and compilation logs before and after compilation. Via the elementary grid services provided by Globus, the remote compiling service can invoke local compilers, e.g., gcc/g++, f77/f90/f95, MPI and PVM compilers, and Condor compiler [26], for different operating system and hardware platforms on the grid, execute the remote compilation task, and produce executable binaries. Moreover, popular scientific computing libraries, such as LAPACK [21], NAG [28], SPRNG [27], GSL [20], and others, are available for linking in the remote compiling process. Furthermore, the remote compiling service provides trustworthiness enforcement mechanisms, including intermediate value checking, secure source program submission, restricted compilation, and binary inspection, to support trustworthy execution of large-scale scientific applications on a computational grid. Figure 1 illustrates the overall system architecture of the remote compiling grid service.

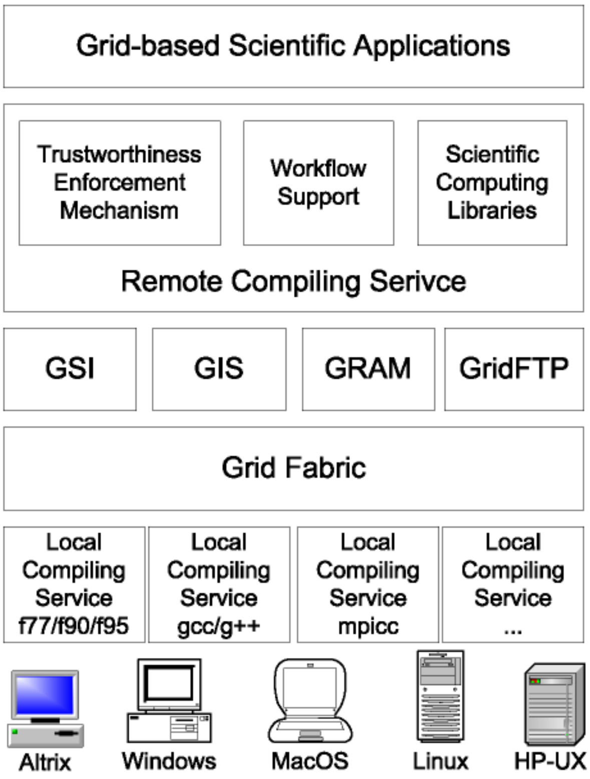

Grid-based Scientific Applications
Trustworthiness Enforcement Mechanism
Workflow Support
Scientific Computing Libraries
Remote Compiling Serivce
GSI
GIS
GRAM
GridFTP
Grid Fabric
Local Compiling Service f77/f90/f95
Local Compiling Service gcc/g++
Local Compiling Service mpicc
Local Compiling Service ...
Altrix
Windows
MacOS
Linux
HP-UX

Fig. 1 System architecture of remote compiling grid service

3 Working paradigm

The life cycle of grid-based applications generally includes phrases of software development, compilation on heterogeneous platforms, distributed execution, and distributed result collection. The remote compiling service on the grid provides facilities of compiling on heterogeneous platforms. In addition to the compiling operations, the remote compiling service focuses on providing trustworthy computing support throughout the life cycle of grid-based applications.

Figure 2 depicts the working paradigm of the remote compiling service on the grid. Logically, virtual organizations are constituted by grid service providers with homogeneous platforms. A remote compiling service is running for each virtual organization to receive and carry out remote compiling tasks. The process of remote compiling on the grid is initiated by a user who submits a compilation package. According to the compilation description specified by the user, the compilation task server searches a remote compiling service provider via GIS whose underlying system configuration can meet the user's requirement. An example of the compilation task description is shown in Fig. 3. If a matching remote compiling service provider is found, the compilation package is then transported to it via GridFTP. To enforce security, the source file can be encrypted using GSI and can be sent to only an authenticated grid service provider. The remote compiling service then performs a security check on the submitted program, compiles the source program according to the com-

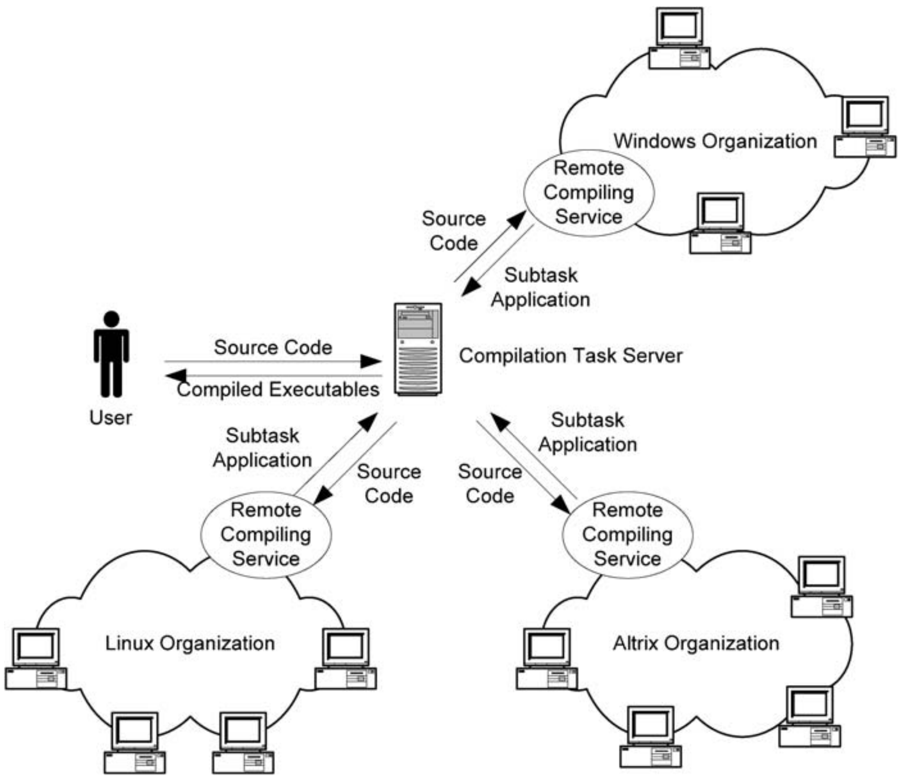


Fig. 2 Working diagram of remote compiling service on the grid

Fig. 3 Remote compiling task specifications

Remote compiling task specifications	
TaskName	= "Monte Carlo Integration"
Compiler	= g++
SourceFile	= http://abner.ncat.edu/mcint.cpp
SourceFile	= http://abner.ncat.edu/random.cpp
FLAGS	= -O3–wall
LIBS	= -lm
Arch	= INTEL
OpSys	= LINUX
OutputFile	= mcintlinux.out
Log	= mcintlinux.log
Encription	= yes

pilation commands specified in the compilation package, links with the necessary scientific computing libraries, and generates the executable binaries if the compilation is successful. At the same time, intermediate value checking mechanisms may also be incorporated. After compilation, the executable binaries as well as the compilation log file for the compilation process are then transported back via GridFTP again. After receiving the executable binaries, the compilation task server validates the compilation logs and binary files and then makes them available to the user. The

```
<WorkFlow id = “mainworkflow”>
  <SubWorkFlow id = “CompilationOnLinux”, order = 1>
    <Organization id = “linuxorg”> </Organization>
<DataTransfer> mci.cpp </DataTransfer>
    <DataTransfer> random.cpp </DataTransfer>
<Operation>
 g++ -O3 -wall -lm mci.cpp random.cpp -o mcintlinux.out
</Operation>
<DataTransfer> mcint.linuxout </DataTransfer>
<DataTransfer> compileinfo.linux </DataTransfer>
  </SubWorkFlow>
  <SubWorkFlow id = “CompilationOnAIX”, order = 1>
    <Organization id = “aixorg”> </Organization>
<DataTransfer> mci.cpp </DataTransfer>
    <DataTransfer> random.cpp </DataTransfer>
  <Operation>
  mlc -O3 -wall -lm mci.cpp random.cpp -o mcintaix.out
  </Operation>
<DataTransfer> mcint.aix.out </DataTransfer>
<DataTransfer> compileinfo.aix </DataTransfer>
  </SubWorkFlow>
<SubWorkFlow id = “CompilationOnAltrix”, order = 1>
    <Organization id = “altrixorg”> </Organization>
<DataTransfer> mci.cpp </DataTransfer>
    <DataTransfer> random.cpp </DataTransfer>
<Operation>
c++ -O3 -wall -lm mci.cpp random.cpp -o mcintaltrix.
out
</Operation>
<DataTransfer> mcintaltrix.out </DataTransfer>
<DataTransfer> compileinfo.altrix </DataTransfer>
  </SubWorkFlow>
  <IntermediateWorkFlow id =”validationworkflow”>
    <DataTransfer></DataTransfer>
<Operation>
Validate the executable binaries and compilation
information
</Operation>
    <DataTransfer></DataTransfer>
  </IntermediateWorkFlow>
  <SubWorkFlow id = “usernotification”, order = 2>
    <Organization id = “compilationtaskserver”>
</Organization>
<DataTransfer> mcintlinux.out </DataTransfer>
<DataTransfer> mcintaix.out </DataTransfer>
<DataTransfer> mcintaltrix.out </DataTransfer>
<Operation>
  Notify users
</Operation>
    </SubWorkFlow>
</WorkFlow>
```

Fig. 4 Example workflow of the remote compiling process on the grid

user can later build computational tasks based on these executable binaries and deploy them to computational service providers with different platforms available in the computational grid.

The remote compiling process on the grid can be efficiently represented by a workflow [1, 5]. Figure 4 shows an XML pseudo-workflow describing the process of remotely compiling a C++ program on Linux, AIX, and Altrix platforms. The whole remote compiling process is described as a workflow. Within this workflow, several compilation subworkflows are defined. Each of them specifies the operations of a compilation task carried out on a remote compiling service provider in the virtual organization with homogeneous platforms. The fact that the order numbers of these compilation subworkflows are identical indicates that they can be carried out in parallel. The intermediate workflow mediates the compilation subworkflows and describes the validation operations of the compilation results. Finally, the user notification subworkflow describes the process of notifying the user of the compilation results.

4 Trustworthiness enforcement mechanism

4.1 Trustworthiness issues of remote compiling service on the grid

The main function of the remote compiling service on the grid is to execute the compilation tasks on different computing platforms to produce platform-dependent binaries. However, in reality, a computational grid is a potentially untrustworthy computing environment. Many Internet-wide grid-computing projects, such as SETI@home [2] and folding@home [10], have reported malicious behaviors of some grid service providers, e.g., some service providers don't faithfully execute their assigned tasks. On the other hand, misbehaviors of application program due to software bugs, inappropriate program parameters, and misconfigurations may harm a volunteer computational service provider and eventually discourage a volunteer participant to participate in the computational grid. Therefore, in addition to complete the compilation tasks, the remote compiling grid service should also address the trustworthiness issue posed by the potentially untrustworthy grid computing environment.

The grid application user, the remote compiling service provider, and the computational service providers are three main categories of players in the remote compiling process. The remote execution of the compilation task has to provide trustworthiness enforcement mechanisms to them. Typically, the following trustworthiness requirements need to be satisfied in the remote compiling process on the grid:

(1) To a grid application using the remote compiling service on the grid, the correct and faithful execution of the compilation task is vitally important. Many grid-based scientific computing applications are sensitive to each computational task carried out on various computational service providers. An erroneous result of a computational task will most likely lead to the corruption of the whole grid-based scientific computation. Security enforcement mechanisms must be employed to ensure the trustworthiness of the compilation results.
(2) Another concern of grid application is that the user may also desire to keep their computations, including input parameters, output results, and programs, secure so that nobody can steal their data or program. Generally speaking, keeping the computation secure from its remote computational service provider is a difficult, likely impossible, problem [4]. However, the remote compiling service provider should at least be able to keep the source code of grid applications safe.
(3) To a remote compiling service provider, the remote compiling service should only execute compilation-related commands and nothing else. All unauthorized commands should be prohibited.
(4) Similarly, to a computational service provider on the grid, the execution of the assigned subtask should not harm the system. Some potentially dangerous system calls, such as *fork*(), *exec*(), etc., should be prevented to execute on a remote computational service provider.

4.2 Security and trustworthiness solutions in remote compiling service on the grid

4.2.1 Partial result validation via intermediate value checking

From grid-based application point of view, the correctness of the overall computation depends on all partial results generated from subtasks running on the distributed computational service providers. Therefore, the trustworthiness of each subtask computation, i.e., the partial results obtained are, in fact, due to the computation requested, is of prime importance. In some distributed systems, rudimentary accountability measures are often employed when the list of service providers is generally static and all are known to each other by hostname or address. Misbehavior on anyone's part leads to a permanent bad reputation. However, in a complicated grid computing environment when central control of resources is lack, the traditional mechanism for ensuring proper behavior can no longer provide the same level of protection. It is usually difficult to uniquely and permanently identify service providers and their operators. As a result, partial result validation mechanisms must be employed to ensure the trustworthy execution of each subtask.

One effective approach to check the validity of a subtask computation is to validate intermediate values within the computation on a grid computational service provider. Intermediate values are quantities generated within the execution of the subtask. To the node that runs the subtask, these values will be unknown until the subtask is actually executed and reaches a specific point within the program. On the other hand, to the clever application owner, certain intermediate values are either pre-known and secret or are very easy to generate. Therefore, by comparing the intermediate values and the pre-known values, one can control whether the subtask is actually faithfully carried out or not.

In [22, 23], Li and Mascagni proposed to use the inherent predetermined pseudorandom numbers for intermediate value checking in grid-based Monte Carlo applications. Monte Carlo applications consume pseudorandom numbers, which are generated deterministically from a pseudorandom number generator. If this pseudorandom number generator has a cheap algorithm for computing arbitrarily within the period, the random numbers are perfect candidates to be these cleverly chosen intermediate values.

The strategy of using pseudorandom numbers as intermediate value checking in grid-based Monte Carlo applications is based on the leapfrog property of their underlying pseudorandom number generators. Some pseudorandom number generators, such as Linear Congruential Generators (LCG) or Lagged Fibonacci Generators (LFG), exhibit the fast leap-ahead (leapfrog) property, which enables us to easily and economically jump ahead in the sequence. Using the fast leap-ahead algorithm, one can transform a seed at a particular point in a pseudorandom number generator's cycle to a new point n steps away in $O(log_2 n)$ "operations," where one "operation" is the cost of generating a single random number [7]. Figure 5 shows a pseudoran-

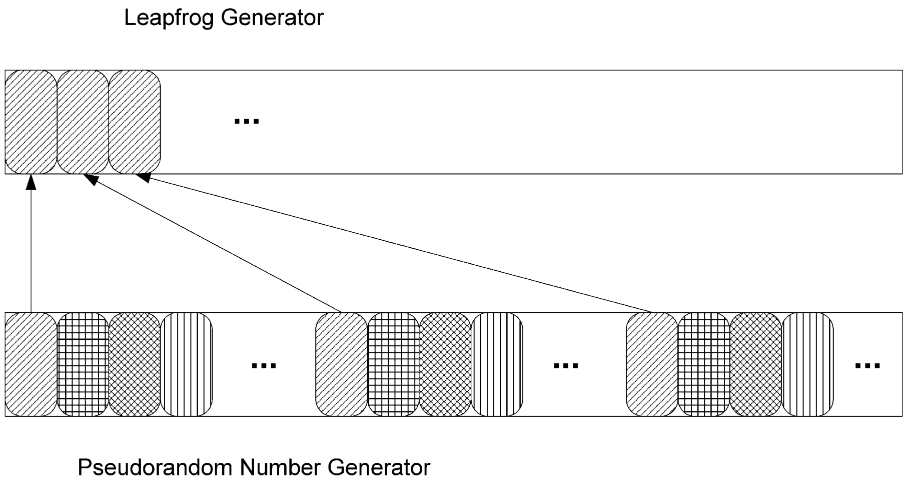


Fig. 5 Pseudorandom number generator and its leapfrog generator

dom number sequence and a leapfrog generator that can quickly leap ahead in this sequence. The fast leap-ahead (leapfrog) property of pseudorandom number generators is widely used to spread a serial pseudorandom number sequence across parallel processors. However, to enforce the trustworthiness of the execution of a subtask, the fast leap-ahead technique with intermediate value checking is employed to economically regenerate any selected pseudorandom numbers used in a Monte Carlo subtask. These selected pseudorandom numbers are used as the intermediate values for further validation. During the execution of a Monte Carlo subtask on a computational service provider, the values of the current pseudorandom number after every N pseudorandom numbers are generated and saved as intermediate values. Thus, a record of the Nth, $2N$th, ..., kNth random numbers used in the subtask computation are produced. When a subtask is complete, the verification service obtains this record and then re-computes the Nth, $2N$th, ..., kNth random numbers using a leapfrog generator. A mismatch indicates problems during the execution of the subtask.

The approach of using pseudorandom numbers with leapfrog property as intermediate value checking can be easily extended to a general scientific computing application where random numbers are not necessary needed. Unlike business programs, a long-run scientific computing program is usually composed of loops with a large number of iterations. Thus, to produce a random number digest, a programmer can smartly “inject” codes of producing certain number of pseudorandom numbers into each iteration and periodically record one random number. These pseudorandom numbers can be generated in several CPU cycles and should not cause a serious overhead to the performance of the scientific application. In the intermediate value checking process, the random number digest is quickly reproduced and compared with the one generated on a computational service provider. A mismatch indicates some problems. Figure 6 shows the matching procedure of pseudorandom numbers in a subtask and the one regenerated by leapfrog in the intermediate value checking process.

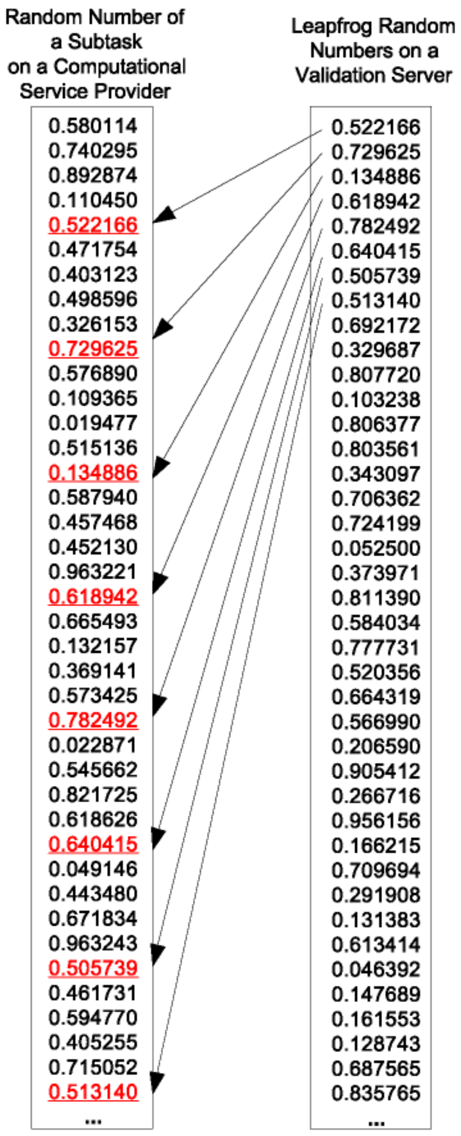
The remote compiling service provides leapfrog pseudorandom number generator library to support intermediate value checking. In our implementation, we employed the Scalable Parallel Random Number Generator (SPRNG) library [7], which can generate up to $2^{78,000} - 1$ independent pseudorandom sequences based on parameterizing the random number generator. The remote compiling service links a SPRNG generator to submitted program with randomized parameters. Also, it provides a corresponding leapfrog generator, which can be used to produce the random number digest, to the grid application user for intermediate value checking after the task is executed on a computational service provider.

4.2.2 Secure source program submission

To keep the source code of the grid application secure, most important of all, we must guarantee that the program to be compiled can only be safely submitted to a trustworthy remote compiling service provider. The following security mechanisms are employed:

(1) End-to-end encryption based on GSI is used to supply a basic level of trustworthiness in data communication in the program submission process.

Fig. 6 Match of pseudorandom numbers in a subtask and the ones regenerated by leapfrog



(2) The compilation task server keeps track of all trusted remote compiling service providers within each virtual organization in its security certificate repositories [31], which are recommended to the grid application users.

(3) Authentication of submitting a compiling task to a remote compiling service provider is enforced by the GSI proxy certificates [12].

(4) The remote compiling service removes the submitted source program right after compilation to minimize the exposure of the source program.

4.2.3 *Restricted compilation*

Preventing a malicious user from carrying out unauthorized operations on the remote compiling service provider is relatively simple. In our implementation of the remote compiling service, we keep a list of valid compiler names. When the requested compiler name is received by the remote compiling service, it is checked against this list—a compilation command not on the list will be refused execution. Moreover, the compilation task is carried out as a user with limited system resource access privileges, such as “nobody” in UNIX.

4.2.4 *Binary inspection*

Sandbox [6] is one of the common techniques to prevent a remote program from harming the computational service providers. A sandboxed environment intercepts important system calls to gain complete control of usage of system resources, such as the file system, I/O devices, registry, etc. The sandbox prevents applications from maliciously or accidentally causing harm to the resources in a computational service provider, since they cannot modify the file system or the system registry.

In our implementation of the remote compiling service, we intend to make the sandbox safer by providing binary inspection. Tools for inspecting the variable and function table in the executable binary file are integrated in the remote compiling service. Within the program checking, some potentially dangerous system calls, such as *fork*(), *exec*(), *ssh*(), *signal*(), etc., are spotted in the remote compiling process. Programs with these system calls are prevented from executing on a remote computational service provider.

5 Conclusions and future research directions

A computational grid generically comprises of large number of heterogeneous computational resources. For legacy computational programs in science and engineering to take advantage of these heterogeneous computational resources, executable binaries on various platforms on the grid must be obtained. In this paper, we present our implementation of a generic remote compiling service on the grid. The remote compiling service is a grid service that allows a grid user to produce executable binaries on various platforms, which can later be deployed to execute on the computational grid. Security enforcement and validation mechanisms, including intermediate value checking, secure source program submission, restricted compilation, and binary inspection, are employed to enforce the trustworthiness and quality assurance of the remote compilation and execution process.

Currently, our remote compiling service on the grid only supports simple compilation commands. In the future, we plan to develop a generic tool for the remote compiling service on the grid, which will be similar to the GNU *automake* and *autoconfig* [3] mechanism and will provide trustworthy grid-based compilation support for development of grid-aware scientific applications.

Acknowledgement This work is partially supported by the “Building an NCA&T Campus Grid Project” of the University of North Carolina General Administration and the NC-HPC Project of the University of North Carolina Office of the President.

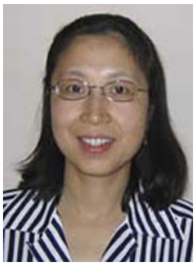
References

1. Allen R (2001) Workflow: an introduction. Workflow Handbook 2001, Workflow Management Coalition
2. Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D (2002) SETI@home: an experiment in public-resource computing. Commun ACM 45(11):56–61
3. Automake and Autoconfig (2005) http://www.gnu.org/software/automake/
4. Beck M, Dongarra J, Eijkhout V, Langston M, Moore T, Plank J (2003) Scalable, trustworthy network computing using untrusted intermediaries: a position paper. In: DOE/NSF workshop on new directions in cyber-security in large-scale networks: development obstacles, 2003
5. Cao J, Jarvis SA, Saini S, Nudd GR (2003) GridFlow: workflow management for grid computing. In: Proceedings of 3rd international symposium on cluster computing and the grid, CCGRID'03, 2003
6. Chien A, Calder B, Elbert S, Bhatia K (2003) Entropia: architecture and performance of an enterprise desktop grid system. J Parallel Distrib Comput 63:597–610
7. De' ak (1990) Uniform random number generators for parallel computers. Parallel Comput 15:155–164
8. Distributed.net website (2005) http://www.distributed.net
9. EGEE (2004) http://public.eu-egee.org/
10. Folding@home Distributed Computing (2003) http://folding.stanford.edu
11. Foster I, Kesselman C (1997) Globus: a metacomputing infrastructure toolkit. Int J Supercomput Appl 11(2):115–128
12. Foster I, Kesselman C, Tsudik G, Tuecke S (1998) A security architecture for computational grids. In: ACM conference on computers and security, 1998, pp 83–91
13. Foster I, Kesselman C, Tuecke S (2001) The anatomy of the grid. Int J High Perform Comput Appl 15(3):200–222
14. Foster I, Kesselman C, Nick JM, Tuecke S (2002) Grid services for distributed integration. Comput 35(6):37–46
15. Foster I, Kesselman C, Nick JM, Tuecke S (2003) The physiology of the grid. Wiley series in communications networking and distributed systems
16. Globus website (2005) http://www.globus.org
17. Goble C, Roure DD (2003) The Grid: an application of the semantic Web. In: Grid computing: making the global infrastructure a reality, pp 437–470
18. Goodale T, Allen G, Lanfermann G, Masso J, Radke T, Seidel E, Shalf J (2004) The cactus framework and toolkit: design and applications. In: Vector and parallel processing—VECPAR '2002, 5th international conference, 2004
19. GPDS. http://www.gpds.org/
20. GSL (2005) http://www.gnu.org/software/gsl/
21. LAPACK (2005) http://www.netlib.org/lapack/
22. Li Y, Mascagni M (2002) Grid-based Monte Carlo applications. In: GRID2002, grid computing third international workshop/conference. Lecture notes in computer science, vol 2536, Baltimore, 2002, pp 13–24
23. Li Y, Mascagni M (2003) Analysis of large-scale grid-based Monte Carlo applications. Int J High Perform Comput Appl 17(4):369–382
24. Li Y, Mascagni M (2004) E-science on the grid: toward a dynamic E-science automation with XML and workflow techniques. In: Proceedings of the 8th world multi-conference on systemics, cybernetics, and informatics, SCI'04, Orlando, Florida, 2004
25. Litzkow M, Livny M, Mutka M (1988) Condor—a hunter of idle workstations. In: Proceedings of the 8th international conference of distributed computing systems, June 1988, pp 104–111
26. Thain D, Livny M (2001) Multiple bypass: interposition agents for distributed computing. J Clust Comput 4:39–47
27. Mascagni M, Srinivasan A (2000) Algorithm 806: SPRNG: a scalable library for pseudorandom number generation. ACM Trans Math Softw 26:436–461
28. NAG (2005) http://www.nag.co.uk/
29. NEESit (2004) http://it.nees.org/
30. Novotny J (2000) The grid protal development kit. Concurr: Pract Experience 00:1–7
31. Novotny J, Tuecke S, Welch V (2001) An online credential repository for the grid: MyProxy, High Performance Distributed Computing (HPDC)
32. SETI@home (2002) SETI@home: the search for extraterrestrial intelligence. http://setiathome.ssl.berkeley.edu

33. Taesombut N, Chien A (2004) Distributed Virtual Computer (DVC): simplifying the development of high performance grid applications. In: Proceedings of the workshop on grids and advanced networks (GAN '04), Chicago, Illinois, held in conjunction with the IEEE cluster computing and the grid (CCGrid2004) conference, April 2004
34. WorkBench (2004) http://workbench.sdsc.edu/
35. Zhou M (2000) A scientific computing tool for parallel Monte Carlo in a distributed environment. PhD Dissertation, Univ of Southern Mississippi, 2000

Yaohang Li received his B.S. in Computer Science from South China University of Technology in 1997 and M.S. and Ph.D. degree from Department of Computer Science, Florida State University in 2000 and 2003, respectively. After graduation, he worked as a research associate in the Computer Science and Mathematics Division at Oak Ridge National Laboratory, TN. His research interest is in Grid Computing, Computational Biology, and Monte Carlo Methods. Now he is an assistant professor in Computer Science at North Carolina A&T State University.

Daniel Chen received his B.S. in Mathematical Sciences with a concentration in Computer Science from the University of North Carolina at Chapel Hill and his Master's degree in Computer Science from North Carolina A&T State University. He is currently an Adjunct Assistant Professor in Computer Science at North Carolina A&T State University.



Xiaohong Yuan received her B.S. in Electrical Engineering from Huazhong University of Science and Technology in 1992, Ph.D. degree from Institute of Automation, Chinese Academy of Sciences in 1997 and Ph.D. degree from Department of Computer Science, Florida Atlantic University in 2000. After graduation, she has worked as an assistant and associate professor in the Department of Computer Science at North Carolina A&T State University. Her research interest includes Information Security, Software Engineering, and Visualization Tools for Computer Science Education.