Building Secure Resources to Ensure Safe Computations in Distributed and Potentially Corrupted Environments

Sebastien Varrette[1]*, Jean-Louis Roch[2], Guillaume Duc[3], and Ronan Keryell[3]

[1] University of Luxembourg, LACS Laboratory, Luxembourg
[2] MOAIS/SAFESCALE Project, CNRS–INRIA, LIG Laboratory, Grenoble, France
[3] HPCAS/SAFESCALE group, Computer Science Laboratory, Institut TÉLÉCOM/TÉLÉCOM Bretagne, Plouzané, France

Abstract. Security and fault-tolerance is a big issue for intensive parallel computing in pervasive environments with hardware errors or malicious acts that may alter the result. In [1, 2] is presented a novel, robust and secure architecture able to offer intensive parallel computing in environments where resources may be corrupted. Some efficient result-checking mechanisms are used to certify the results of an execution. The architecture is based on a limited number of safe resources that host the checkpoint server (used to store the graph) and the verifiers able to securely re-execute piece of tasks in a trusted way.
We extend this approach in the case we have also some secure processors to build a trusted check-pointing infrastructure and to replace some untrusted nodes, avoiding some re-execution. Our approach is illustrated on a medical application and some experimental results are presented.

1 Introduction

Nowadays, intensive parallel applications require often global computing platforms composed of scattered resources that are interconnected through the Internet. Such platforms, called *grids* [3], are more and more used since the 90's.

Grid environments raise various concerns in terms of security and data privacy. In particular: users and resources should be authenticated; only authorized users should be allowed to access and to use only the resources of the grid allocated for their own purpose; communications should be ciphered to ensure privacy but also integrity; data should be securely stored; the system should remain operative even in case of failure of some grid components or disconnections which are relatively frequent events. In other words, the integrity of the execution should be guaranteed; the resources of the grid should be protected from malicious code; the infrastructure should not fail even if some parts of the grid are under control of a pirate or, worse, an wicked system administrator, since, on quite large grids, it is no longer possible to know neither to trust every remote system administrator or computer owner.

* Also affiliated with MOAIS.

Other constraints could intervene depending on the type of grid. In this article, we describe a computing platform able to address these issues. More precisely, the approach as been presented in [1, 2] and relies on dataflow graph to represent a parallel execution in a portable way. Using this representation, fault-tolerance mechanisms and efficient result-checking algorithms for heterogeneous multithreaded applications have been proposed. Those functionalities assume the availability of a strongly secured area among the resources of the computing platform. Such resources host the checkpoint server (which stores the graph) and verifiers which are used to re-execute some tasks in a trusted way in order to figure out with a given detection probability if a task did not returned the expected result.

In this paper, we focus on the effective construction of strongly secured resources in the context of the SAFESCALE project. Our talk is mainly oriented toward open-source solutions in computing grids based on Unix or Linux operating systems which constitute major actors in this field. While this issue is generally treated with software solutions, we combine both software and hardware (secure processors) approaches to build strongly secured resources.

In the following, the computing platform of the SAFESCALE project is described in section 2. This infrastructure is able ensure the computation resilience despite crash faults and malware attacks that lead to computation alterations. In section 3, we expound the construction of the secured resources that are essential for both result-checking and fault-tolerance mechanisms. Section 4 details an application deployed on the proposed architecture.

2 A computing platform ensuring computation resilience

Resilience in grid execution is a prerequisite that should be embedded in the application: at this scale, component failures, disconnections or results modifications are unfortunately part of operations, and applications have to deal directly with repeated failures during program runs.

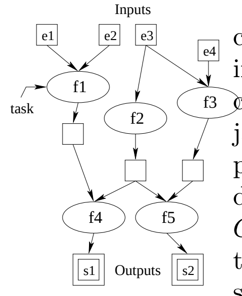

Fig. 1. A data-flow graph with 5 tasks.

In [1, 2], the authors present a robust and secure architecture able to deal with intensive parallel computing in environments where resources could be corrupted. The corruption could be caused by DDoS attacks, virus or trojan horses, as expounded in the precedent section. The proposed approach uses a portable representation of the distributed execution: a bipartite Direct Acyclic Graph $G = (\mathcal{V}, \mathcal{E})$. The first class of vertices is associated to the tasks (in the sequential scheduling sense) whereas the second one represents the parameters of the tasks (either inputs or outputs according to the direction of the edge). Such a graph is illustrated in Figure 1.

Using this representation, portable fault-tolerance mechanisms for heterogeneous multithreaded applications have been proposed [4]. Since we have a clean separation of the applications in tasks that communicate only through their defined inputs and outputs with no other side effect, each one can be run again in case of trouble. Furthermore, efficient result-checking mechanisms exploiting the graph have been developed [1] and are able to certify the behaviour and the results of an execution.

Both approaches are conducted with a low overhead that only required the existence of a checkpoint server deployed on a set of strongly safe resources. This server stores the dataflow graph of the execution provided by the *Kernel for Adaptive, Asynchronous Parallel and Interactive* — KAAPI — application programming interface. KAAPI is a C++ library that allows to program and execute multithreaded computations with dataflow synchronization between threads. In addition, result-checking algorithms require the deployment of verifiers that could securely re-execute some tasks in a trusted way. The proposed approach aims at limiting the number of re-executions and therefore the number of verifiers calls.

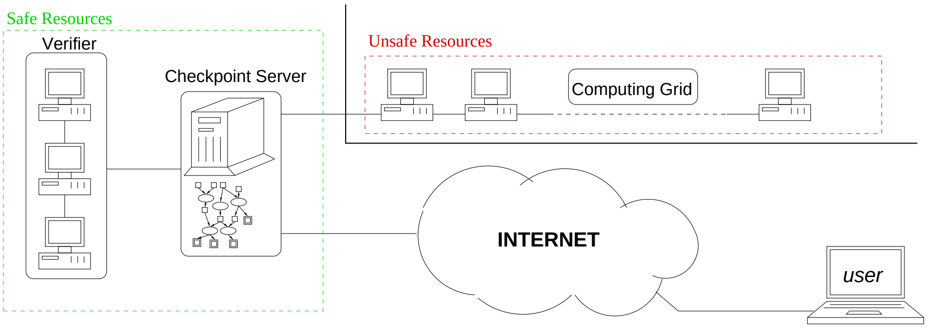

Fig. 2. *Resources hierarchy and mandatory components for portable fault-tolerance and error-checking algorithms.*

This leads to the infrastructure presented in Figure 2 in which the resources have been divided in two classes:

1. a limited number of strongly safe resources that host the checkpoint server and the verifiers;
2. the other resources, mentioned as “unsafe”, which constitute the real computing grid and which are scattered among the different institutions (for example, the different hospitals involved in the experiment described in section 4).

It remains to detail the effective construction of the safe resources used in the proposed architecture. This is one of the contributions of this article and the purpose of the next section.

3 Building strongly secured resources

Building safe resources is one of the most important challenge of system administrators. The solution used generally combines various software solutions to make the system more robust.

3.1 Software Components

Even if the software components are secure, there is an asymmetry in the trust for all the previous techniques: the aim is to protect the computing infrastructure from the program execution and there is no way for the users to have a certified and protected execution.

Software obfuscation is a software-only partial answer to software protection and data protection [5]. The code is transformed at the source or binary level to render more complex its readability and also change the data coding. Of course, often this slows down the execution and it is not impossible for someone very motivated to reverse-engineer the obfuscating process since the code is available for execution and can be exercised at will.

Another way is to transform an algorithm that produces data from input data into an isomorphic one that acts on ciphered input and produces ciphered output. It is also possible to add some authentication mechanisms in it to certify the computation done. For some simple algorithms there exist such efficient isomorphic algorithms but, unfortunately, this seducing approach has an intractable complexity for real life programs [6]. Since basic computations (as, for example, floating point operations that are highly optimized in modern processors), are transformed in elementary operations executed to emulate some enciphered circuits, the expected efficiency on grids is no longer possible.

There is still a big issue with a full software approach: the operating system on the computing nodes have the full control of the hardware and the software running on the nodes. It is very useful to build very complex and powerful computing environment but it may be very dangerous too for the (foreign) users of these computing nodes.

A computing node can easily discard a foreign process with all its data if it looks malicious or exploits too many resources. In addition, even if the operating system environment would be bug free, a foreign process needs to be completely confident in the local software environment: this one can discard the process, the data, read the program and the data, modify both, execute the program step-by-step, and so on. If DoS are unavoidable (the local administrator could decide to switch off the power supply of the computer anyway), computers should have a mechanism to avoid this excess of power used in a malicious way or to signal a running process that a kind of DoS occurred.

In the real life, operating systems and distributed computing environment are huge pieces of software and it is unavoidable to have many bugs in them. Thus, adding pieces of hardware to protect some processes from other parts running out of their rails is quite interesting.

3.2 Hardware Components

During the last few years, several hardware architectures [7–9] have been proposed to provide computer applications with a secure computing environment. These architectures use memory encryption and memory integrity checking to guarantee that an attacker cannot disturb the operation of a secure process, or can only obtain as little information as possible about the code or the data manipulated by this process even with some external physical attacks.

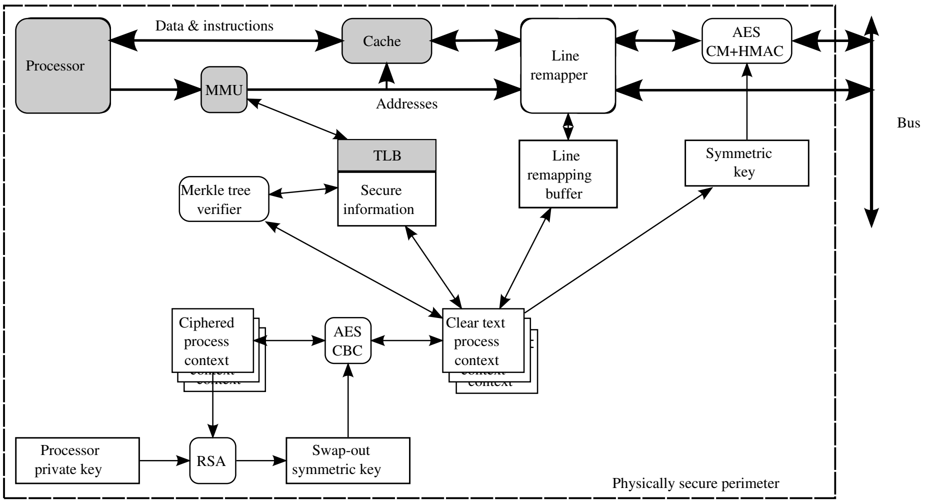

Fig. 3. *Simplified* CryptoPage *secure high-performance processor.*

Some secrets can also leak out through the address bus of the processor (an attacker can monitor the control flow graph of a running program and infer algorithms or ciphering keys for example), some approaches try to cipher the address bus more [10] or less [11]. Recently, we have proposed a processor architecture that combines opaque secure mode execution with efficient memory encryption and verification, resistant to replay-attack, with dynamic random remapping of cache lines in memory page to hide memory usage and to avoid address tracing [12]. The simplified architecture is represented on Figure 3. The white boxes are the ones we added to a plain processor architecture (the gray boxes). Rounded boxes are computing or function elements and the squared boxes are some storage elements.

A process cannot be stolen or tapped since its execution context is enciphered with the public key of a given target processor. The confidentiality of the code and the data is guaranty by a cipher (using a random session key) between the internal cache and the memory. A data Hide-like address remapper [10] is added to shuffle data each time a memory page is read into the processor. At this page level,an efficient mechanism is also added to avoid a replay attack (when an attacker replay an old data written and authenticated by the processor).

We implemented this architecture in the SimpleScalar simulator to have some quantitative results. The performance on the SPEC CPUInt2000 running on CyptoPage shows only an average slow-down of around 3% [12] with address and data ciphering and lazy verification through speculative insecure execution.

Of course, a foreign process needs to ultimately trust the manufacturer of the processor which could have some wire-tapping or key-escrow features in it, or more probably hardware bugs too.

3.3 Secure grid

Some grids with some tamper-proof processors such as Java Smart Card [13] have already been built but are not powerful enough to be general purpose. In our globally secure grid environment, high performance tamper-proof hardware processors can be used to securely run some parts of the computation if we have enough of those trusted elements.

By using the same session key between 2 processors, we can migrate one running process to another processor, that is useful in distributed computing for load-balancing or fault-tolerance, without compromising the security.

If we cannot trust some parts of the nodes, i.e. we do not have enough trusted elements, we need to probabilistically run again (as seen in section 2) some parts of the computation on some secure nodes that are the trusted element (the verifiers). If there is no trusted nodes at all, a grid user must choose to run the verification parts of the remote computation on her/his own nodes she/he is confident of.

4 An application using strongly secured resources in SAFESCALE

To validate the presented infrastructure, the following security-demanding healthcare application is considered:

- a given picture A is compared to a set $\mathcal{S}$ of pictures stored in a distributed database;
- based on metadata information, some images $X \in \mathcal{S}$ are extracted and compared to A; the result of this comparison is a score $s_A(X)$ that measures the correlation between pictures A and X;
- finally, the sorted results are brought to the end users.

Among concrete instances of such a generic application is the RAGTIME software that uses medical image comparison within PACS (Picture Archiving and Communication Systems) to detect rare and hard to predict diseases [2].

Due to numerical uncertainties brought by score computations, relevant results are obtained if several images in $\mathcal{S}$ are identified as matching picture A. Also, such an application may directly takes benefit from the computational power of a global computing infrastructure:

- the number of pairwise comparison scores to compute is huge and scores may easily be computed in parallel;
- for a given user that submits a picture A interactively to the system, the system usage is irregular, from high when it submits a picture to zero when the user has no picture to score. Thus, federating resources from several users in a single grid, positively contributes to increase the system throughput for any users;
- the application tolerates few errors, which are expensive to prevent in a grid context. Indeed, the major result is to find several images in $\mathcal{S}$ that are correlated to A, and thus bring together information on A. In this context, the fact that only few scores have been fake (let say one or two) does not affect the result. The critical point is here to ensure that almost all computations have been correctly performed.

To illustrate the latter point, we consider the case where the user wants to compute statistics on the top 5% best related picture to A in a huge database. To achieve this, the top 10% scores computed on the grid are returned to the user. Then if it is certified that the probability of an attack that may have fakes more than 5% of the results is negligible, then the user is sure that among the top 10% resulting scores on the grid, at least half of the results are undoubtedly in the top 10%. The, recomputing the top 10% on secure resources may be a way of preventing from a massive attack on such an application.

This generic application has been ported on top of the KAAPI environment on in Grid'5000 infrastructure (a French grid with 5000 nodes France-wide). In order to ensure that all metadata have been correctly analyzed, the selection of the images in $\mathcal{S}$ that have to be scored against A is carried out on a secure infrastructure C_S (that may also be a grid); C_S also manages access to a safe checkpoint server where are stored both description of task to be performed (i.e. path names of both pictures to be scored) and, later, result of the comparison (which is associated to the two pathnames that identify the task). Then, all scores to be computed are send to a global computing infrastructure C_U. Afterwards, on G_C a few number N_{ϵ_q} of scores are recomputed and verified. If no result forgery is detected, then the result of the comparison is correct with high probability.

4.1 Distribution of scores computation based on work-stealing

Since the number of scores to be computed on the grid is huge, having a centralized allocation strategy (when a processor becomes idle, it contacts the master processor to obtain a new computation task) introduces contention and inefficiency. Instead, a distributed solution based on KAAPI work-stealing has been developed. When a processor becomes idle, it picks a victim at random and steals about half of is computation queue. Such a solution ensure high performances on a global architecture where processor speed may vary. Indeed, let $\Pi(t)$ be the instantaneous computation speed (i.e the number of unitary operations per time unit) and $W_A(X)$ be the number of unit operations to compute to score picture X against A; then the time T_p to carried out the whole computation on

G_U is, with high probability, lesser than $\frac{\sum_{X \in S} W_A(X)}{\Pi(t)} + \mathcal{O}(W_A(X) + \log n)$ which is merely optimal [14, 15].

4.2 Ensuring resource resilience

Since the checkpoint is safely stored on G_C, the checkpoint/restart of the application is directly managed by KAAPI which accounts for completed tasks registration.

5 Conclusion

Even if security in grid infrastructures is a major research area for a decade, the fact that resources that compose the grid cannot be fully trusted prevents a wide acceptance of such systems as a cheaper computation platform for high-valued applications. The corruption either comes from hardware issues (such as network disconnection) or from malicious act (using malwares and software vulnerabilities) in order to alter the computation and consequently its result.

In our SAFESCALE platform, we ensure a correct and safe computation by combining efficient result-checking and fault-tolerance algorithms with a limited number of strongly secure resources. The result-checking algorithms relies on stochastic verifications by running again on trusted verifiers, some parts of the global computation chosen by studying the data-flow graph of the application. Since verifications can also be tampered, these tasks need to be run on different computers in different entities to have forgeries likely detected. But by using a limited number of strongly secure hardware resources, the sensible and time-consuming task of verification can be executed on some remote tamper-proof nodes with no fear about any attacker changing these results. According to the number of trusted resources available in the grid, our method needs more or less redundant verifications.

This architecture has been validated on a medical application consisting in similarity computations between medical images and we are now working on automatic parallelization of application into the KAAPI model, based on the PIPS source-to-source compiler [16]

Acknowledgments

We would like to thanks all the members of the SAFESCALE project. This work is supported by the ANR (French National Research Agency) project SAFESCALE-BGPR ANR-05-SSIA-005 and the French Department of Defense.

References

1. Krings, A., Roch, J.L., Jafar, S., Varrette, S.: A Probabilistic Approach for Task and Result Certification of Large-scale Distributed Applications in Hostile Environments. In Verlag, S., ed.: Proceedings of the European Grid Conference (EGC2005).

LNCS 3470, Amsterdam, Netherlands, LNCS, Springer Verlag (February 14–16 2005)

2. Varrette, S., Roch, J.L., Montagnat, J., Seitz, L., Pierson, J.M., Leprévost, F.: Safe Distributed Architecture for Image-based Computer Assisted Diagnosis. In: IEEE 1st International Workshop on Health Pervasive Systems (HPS'06), Lyon, France (june 2006)
3. Foster, I., Kesselman, C.: Globus: A metacomputing infrastructure toolkit. International J. of Supercomputer Applications and High Performance Computing **11**(2) (Summer 1997) 115–128
4. Jafar, S., Varrette, S., Roch, J.L.: Using Data-Flow Analysis for Resilence and Result Checking in Peer to Peer Computations. In IEEE, ed.: IEEE DEXA'2004 - Workshop GLOBE'04: Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems, Zaragoza, Spain (September 2004) 512–516
5. Collberg, C.S., Thomborson, C.: Watermarking, tamper-proofing, and obfuscation - tools for software protection. In: IEEE Transactions on Software Engineering. Volume 28. (August 2002) 735–746
6. Loureiro, S., Bussard, L., Roudier, Y.: Extending tamper-proof hardware security to untrusted execution environments. In: CARDIS. (2002) 111–124
7. Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J., Horowitz, M.: Architectural support for copy and tamper resistant software. In: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX). (October 2000) 168–177
8. Suh, G.E., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: AEGIS: Architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the 17th International Conference on Supercomputing (ICS'03). (June 2003) 160–171
9. Keryell, R.: Cryptopage-1 : vers la fin du piratage informatique ? In: Symposium d'Architecture (SympA'6), Besançon, France (June 2000) 35–44
10. Zhuang, X., Zhang, T., Pande, S.: HIDE: an infrastructure for efficiently protecting information leakage on the address bus. In: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XI), ACM Press (October 2004) 72–84
11. Duc, G., Keryell, R., Lauradoux, C.: CRYPTOPAGE : Support matériel pour cryptoprocessus. Technique et Science Informatiques **24** (2005) 667–701
12. Duc, G., Keryell, R.: CRYPTOPAGE: an efficient secure architecture with memory encryption, integrity and information leakage protection. In: Proceedings of the 22th Annual Computer Security Applications Conference (ACSAC'06), IEEE Computer Society (December 2006)
13. Chaumette, S., Grange, P., Sauveron, D., Vignéras, P.: Computing with java cards. In: International Conference on Computer, Communication and Control Technologies (CCCT'03), Orlando, FL, USA (July 2003)
14. Bender, M.A., Rabin, M.O.: Online scheduling of parallel programs on heterogeneous systems with applications to cilk. Theory Comput. Syst. **35**(3) (2002) 289–304
15. Roch, J.L., Traore, D., Bernard, J.: On-line adaptive parallel prefix computation. In Springer-Verlag, L..h., ed.: EUROPAR'2006, Dresden, Germany (August 2006) 843–850
16. Ancourt, C., Coelho, F., Creusillet, B., Keryell, R.: How to add a new phase in PIPS: the case of dead code elimination. In: Proceedings of the Sixth Workshop on Compilers for Parallel Computers (CPC'96), Aachen, Germany (December 1996) 19–30