EPJ Web of Conferences **68**, 00016 (2014)
DOI: 10.1051/epjconf/20146800016

Distributed Online Judge System for Interactive Theorem Provers

Takahisa Mizuno and Shin-ya Nishizaki[1]

[1]Department of Computer Science, Tokyo Institute of Technology, Ookayama, Meguro, Tokyo 152-8552, Japan

Abstract. In this paper, we propose a new software design of an online judge system for interactive theorem proving. The distinctive feature of this architecture is that our online judge system is distributed on the network and especially involves volunteer computing. In volunteers' computers, network bots (software robots) are executed and donate computational resources to the central host of the online judge system. Our proposed design improves fault tolerance and security. We gave an implementation to two different styles of interactive theorem prover, Coq and ACL2, and evaluated our proposed architecture. From the experiment on the implementation, we concluded that our architecture is efficient enough to be used practically.

1 Introduction

1.1 A New Style of Distributed Computation, Volunteer computing

Volunteer computing is a type of distributed computing in which computer owners ("*volunteers*") donate their computing resources to projects. Volunteers are typically members of the general public who own Internet-connected personal computers. The first volunteer computing project was the Great Internet Mersenne Prime Search (GIMPS), which was started in 1996. In 1999, the SETI@home project was launched, which received considerable media coverage and attracted several hundred thousand volunteers. In 2002, the Berkeley Open Infrastructure for Network Computing (BOINC) project was founded at the University of California, Berkeley Space Science Laboratory. It was originally developed to support the SETI@home project. Later it became useful as a platform for other distributed applications in various scientific areas. Volunteer computing systems must deal with problems related to correctness:

- We cannot predict the number of volunteers and the volunteers are essentially anonymous;
- Some volunteer computers are possibly overclocked and they occasionally do not work well and return incorrect results;
- Some volunteers can intentionally return incorrect results.

1.2 Interactive Theorem Provers

An interactive theorem prover is a software system which assists in developing formal proofs through human-machine collaboration. Formal systems, such as higher-order logic, and higher-order type systems, are used for describing formal proofs in interactive theorem provers. Interactive theorem provers provide automatic assistance for rigorous reasoning in such formal systems. The formal reasoning and proving process on the interactive theorem provers shed light on ambiguity in standard mathematics. In computer science, interactive theorem provers are applied to verification of mission-critical software. Several kinds of interactive theorem prover have been developed since the 1970s. LCF [1] was the first tactic/tactical based prover; Coq[2], the HOL prover [3], and Isabelle/HOL[4] are considered to be successors to LCF. In these provers, automated reasoning is provided by combining several kinds of simple reasoning step, called *tactics,* with control structures, called *tacticals.* The interactive theorem prover Coq is based on a higher-order type system with inductive definitions, Calculus of Inductive Constructions, which is powerful enough to describe definitions and proofs in mathematics and computer science. For example, an inductive definition of list concatenation is given in Coq as

```
Fixpoint app{A:Set}(xs ys: list A) :=
  match xs with
    | Nil => ys
    | x::xs' => x :: app xs' ys
  end.
```

where xs and ys are formal parameters of the concatenation function app, A is a type of element of the list, Nil is the empty list, and :: is the list constructor.

A proof of associativity of the list concatenation is written in Coq as follows.

```
Theorem app_assoc:
 forall {A:Set}(xs ys zs: listA),
```

[a] Corresponding author: author@email.org

```
app xs(app ys zs)=app(app x sys) zs).

intros.
induction xs.
simpl.
reflexivity.
simple.
rewrite lHxs.
reflexivity.
Qed.
```

Besides tactic/tactical based provers, there are other kinds of interactive prover. A remarkable prover is ACL2 [5], which is a part of the Boyer-Moore family of provers, developed R. S. Boyer, M. Kaufmann, and J. S. Moore. This prover is used in practical verification of software and hardware systems. For example, the correctness of FPU of AMD K5 is formally verified using ACL2[6].

In ACL2, the list reverse function is defined as

```
(defun rev (x)
 (if (endp x) nil
   (append(rev(cdr x))(list(car x)))))
```

For example, let us prove that (rev (rev (rev x))) is equal to (rev x). If we first prove a lemma:

```
(defthm rev-rev
    (implies (true-listp x)
       (equal (rev(rev x)) x)))
```

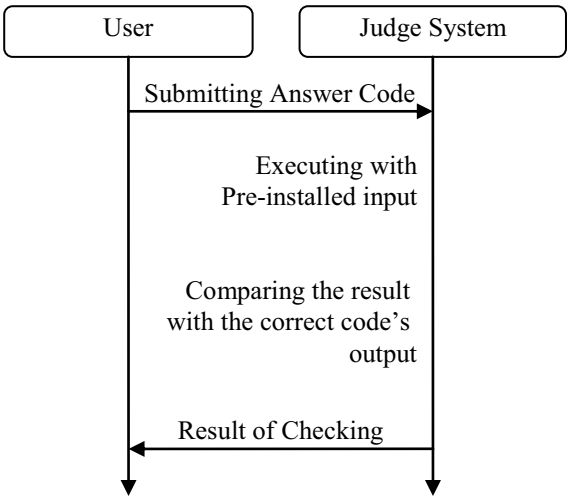
then the claim is automatically proved as

```
(defthm triple-rev
 (equal (rev(rev(rev x))) (rev x)))
```

1.3 Online Judge System

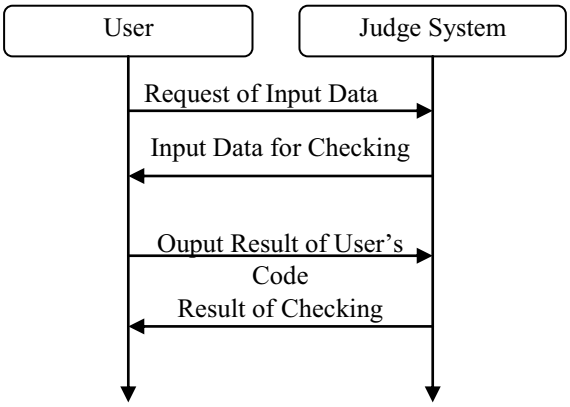
An *online judge system* poses problems such as questions in programming contests to users, receives their answers from the users, and checks the correctness of the submitted answers. Checking programs' correctness implies decision problem of programs' termination, and therefore, it is undecidable. Consequently, the program checking is restricted to an approximation of correctness checking, in which a submitted and the submitted programs are compared with the output data generated by the correct answer programs installed in advance. In the ACM International Collegiate Programming Contest (ACM-ICPC), a contest support system PC^2 [7] is used. The system PC^2 is also an online judge system which checks a submitted answer by comparing the results of pre-installed correct answer programs with pre-installed input data.

Figure 1. Server-Side Checking



Another remarkable example of a contest support system is that of Google Code Jam (https://code.google.com/codejam). The checking method of this system is different with that of PC^2, which is depicted in Figure 1. If the user completes an answer program code, then he/she requests input data to test his/her code. A checking system on the client side executes the answer program code with the received input data and obtains the result of the execution. The checking system sends back the output result to the checking server and the server verifies the result. We call this method "client side checking."

Figure 2. Client-Side Checking



From the viewpoint of security, the server-side checking is potentially vulnerable to denial-of-service attacks because a server has to execute unknown codes. On the other hand, the client-side checking forces clients to equip a compilation and execution environment for checking and it imposes a heavy burden on the developers of a judge system.

1.4 Research Purpose

In this paper, we propose a new design of an online judge system for interactive theorem proving in the style of

server-side checking. We study a method for reducing the vulnerability of server-side checking and computational load on the server, introducing volunteer computing.

In comparison with the existing proof-checkers, our proposed system has a merit that it can be used in the open distributed environment safely. The existing online judge systems such as PC^2 [7], PKU JudgeOnline, and Google Code Jam verify a submitted program by executing it with test data prepared in advance. Therefore, the submitted program possibly includes errors. On the other hand, our proposed system checks a submitted proof using theorem provers and the correctness of the checked proof is theoretically guaranteed.

2 Online Judge System Volunteer Computing

In this section, we propose a new design of an online judge system for interactive theorem proving. The following points concerning security of the online judge should be considered.

- Checking submitted proofs for existing interactive theorem provers has similar vulnerability to checking for contest programming. Because proof scripts in many interactive theorem provers can include program scripts, checking of proof scripts can derive executing of programs. The existing implementations of interactive theorem provers are not assumed to be used by anonymous users including malicious attackers on the internet. A malicious program code can carry out a buffer overflow attack on a server and hijacks the server. This kind of attack can be fended off by executing codes in a sandbox arranged in the server.
- A malicious contributor can submit a proof script which saps the computational resources of a checking server and makes the server stop. This kind of attack cannot be avoided by using a sandbox, since execution in the sandbox can exhaust the computational resources of the server.

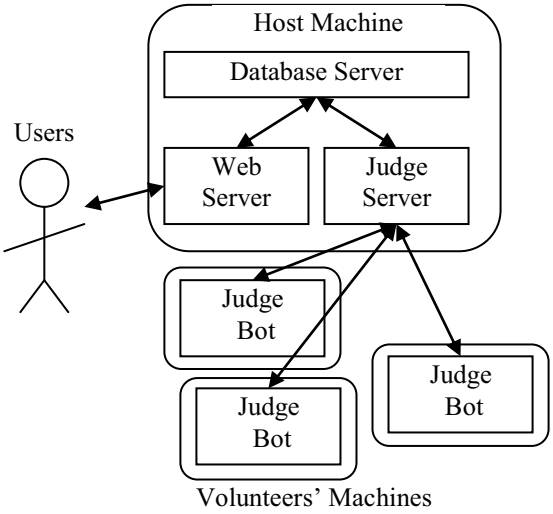
From the above, we know that checking of submitted proof scripts should be carried out on another machine than the main server. However, it is not easy to prepare machines for checking submitted proof scripts. We therefore introduce the idea of volunteer computing for processing possibly enormous demands of computational resources.

Our online judge system for interactive theorem proving consists of

- Web Server,
- Judge Controlling Server,
- Database Server
- Judge Bots.

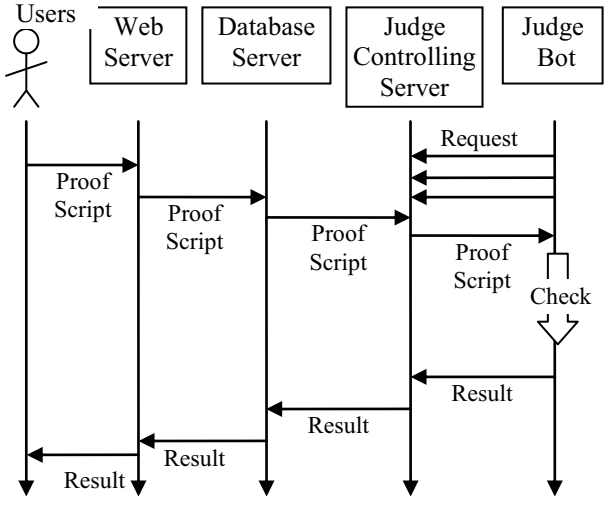
The web server provides a web-based interface to users. The judge bots are in charge of checking the correctness of submitted proof scripts. The judge controlling server supervises the judge bots. The database server mediates between the web server and the judge controlling server.

Figure 3. Overview of Our Online Judge System



The distinctive feature of our system is that each judge bot requests a proof script to be checked autonomously; in other words, each judge bot is an initiator of communication between the judge controlling server and the judge bot, which is the reason why we call them *bots*. Judge bots are *"subcontractors"* of the judge controlling system, but judge bots initiate the communication with the judge controlling server. This is important because it enables us to place judge bots inside volunteers' machines in private networks. Thanks to the private networks, if a judge bot were intruded and hijacked, its damage could be limited inside the private networks. Volunteers' machines in their private networks play a role of *sandbox.* In Figure 4, we show the procedures of our online judge system in Figure 4.

Figure 4. Procedures in Online Judge System



Though our system works normally with a single judge bot, a redundant configuration of multiple judge bots is

usually assumed, which improves the fault-tolerance of the system. The overview of communication between the judge controlling server and the judge bots can be described as follows.

1. The judge controlling server indicates a proof script submitted by a user, which is obtained via the database server.
2. Several judge bots that are disengaged enough to check a new proof script, pick it up from the judge controlling server.
3. The judge bots check the proof script using an interactive theorem prover and return the result to the judge controlling server.
4. The judge controlling server receives multiple results from some of the judge bots and compares them. If they are consistent, then it registers the result on the database server. If the judge controlling server finds some inconsistency, then it purges suspicious judge bots.

The redundant configuration of judge bots improves the reliability of the results. We assume that a judge bot is installed on a volunteer's computer and it could be malicious and unfaithful. By comparing multiple results from judge bots, we can find a false result from such a malicious judge bot.

Moreover, if an attacker submits a proof script including a malicious code, then the judge bots can be damaged but the host machine is not invaded.

3 Implementation and Evaluation

We implement the online judge system for interactive theorem proving following the design explained in the previous chapter.

We take up Coq and ACL2 as the interactive theorem provers which are supported in our judge system. We assume that users submitted proof scripts of either Coq or ACL2. Judge bots include these theorem provers in order to check the submitted proof scripts.

In our implementation, we use the following software:

- Python 2.7, by which we describe codes of the web server, the judge controlling server, and the judge bots;
- Django 1.3: a Python-based framework for web applications which is used for implementing the web server;
- SQLite 3.7.1.0: DBMS which is used in the database server;
- Coq 8.3: an LCF-style interactive theorem prover, which checks submitted proof scripts in the judge bots;
- ACL2 2.6.8: a Boyer-Moore-style interactive theorem prover, which also checks submitted proof scripts.

We evaluate our implementation with the following proof scripts (Table 1).

Table 1. Proof scripts for Evaluation

	Theorem	Prover	Lines
(1)	Modus Ponens	Coq	6
(2)	GCJJ R1 C	Coq	396
(3)	Triple Reverse	ACL2	11
(4)	Dupsp	ACL2	29

We want to know overhead of our design in comparison with a naïve implementation without using bots and with direct usage of interactive theorem provers.

Table 2. Result of Evaluation

	(1)	(2)	(3)	(4)
Our System	3.818	7.001	0.4	0.551
Naïve System	3.178	5.783	0.284	0.663
Direct Checking	0.605	5.375	0.162	0.380

(Seconds)

Our System: Our implementation of the online judge system
Naïve System: A naive implementation of an online judge system for interactive theorem provers. This executes interactive theorem provers directly, without using judge bots.
Direct Usage: Direct usage of interactive theorem provers.

From the result of Table 2, we know that the overhead of our software architecture proposed in this paper is not heavy: at worst, the overhead is less than double.

4 Conclusion

In this paper, we proposed new software architecture of an online judge system for interactive theorem proving. The distinctive feature of this architecture is that our online judge system is distributed on the network and especially involves volunteer computing. Several components of our system are possibly located in the various computers and they are connected with each other via the Internet. We gave an implementation to two different styles of interactive theorem prover, Coq and ACL2, and evaluated our proposed architecture. From the experiment on the implementation, we conclude that our architecture is efficient enough to be used practically.

In future, we would like to extend our online judge system to other kinds of interactive theorem provers. Moreover, collaboration with social network systems, such as Facebook and Google+ is also important and should be studied.

Acknowledgements

One of the authors, Takahisa Mizuno, is deeply thankful to the ACM ICPC 2012 Asian regional contest in Indonesia and its host university, BINUS university, which give him a chance to participate in the contest. The experience of the contest inspired this work.
This work was supported by Grants-in-Aid for Scientific Research (C) (24500009).

References

1. L. C. Paulson, *Logic and Computation: Interactive proof with Cambridge LCF,* Cambridge University Press (1987)
2. Y. Bertot and P. Castéran, *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions,* Springer, 2004.
3. M. J. C. Gordon and T. F. Melham (editors), *Introduction to HOL: A theorem proving environment for higher order logic*, Cambridge University Press (1993)
4. L. C. Paulson, *Isabelle: A Generic Theorem Prover,* Lecture Notes in Computer Science, Vol. 828, Springer-Verlag (1994)
5. M. Kaufmann, P. Manolios, J. S. Moore, *Computer-Aided Reasoning: An Approach,* Kluwer Academic (2000)
6. D. M. Russinoff, A Mechanically Checked Proof of Correctness of the AMD K5 Floating Point Square Root Microcode, *Formal Methods in System Design,* Vol. 14, Issue 1, pp 75—125, Kluwer Academic (1999)
7. PC2 home page,http://www.ecs.csus.edu/pc2/ (2013)
8. Google Code Jam Home Page, https://code.google.com/codejam (2013)
9. S. Nishizaki, Programs with Continuations and Linear Logic, *Science of Computer Programming,* Vol. 21, No. 2, pp. 165—190, Elsevier (1994)
10. S. Nishizaki, A Polymorphic Environment Calculus and Its Type-Inference Algorithm, *Higher-Order and Symbolic Computation,* Vol. 13, No. 3, pp 239—278, Kluwer (2000).
11. T. Sasajima and S. Nishizaki, Blog-based Distributed Computation, Proceedings of ICICA2012, *Lecture Notes in Computer Science,* Vol. 7273, pp. 461-467, Springer (2012)
12. T. Mizuno and S. Nishizaki, Analyzing Systems Dependent on Execution Speed with Model Checker, Proceedings of ICASCE 2012, Procedia Engineering, Vol. 50, pp. 544—554, Elsevier (2012)