THE DESIGN OF A MULTI-AGENT FRAMEWORK FOR DISTRIBUTED HYPER-LINKING-BASED THEOREM PROVING

Chih-Hung Wu
Department of Information Management
Shu-Te University
Kaohsiung County, Taiwan 824
E-mail: johnw@mail.stu.edu.tw

Shie-Jue Lee
Department of Electrical Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan 804
E-mail: leesj@ee.nsysu.edu.tw

Abstract

A multi-agent framework for distributed theorem proving of the hyper-linking method is proposed in this paper. Distributed theorem proving at the clause-level can be achieved by collaborating multi-agents. In this framework, each computer collaborates on proving theorems without stopping the jobs currently running. The hyper-linking proof procedure is wrapped as software agents, which can be partially performed for a specific task. Each computer, referred to as proof house, involved in this framework can work solely or collaborate with other registered proof houses to complete the proof of the given problems. Agents for monitoring the traffic of environment, the performance and liveness of each proof houses, updating and sharing proof experiences, and tuning parameters of proof houses are designed. The architecture and the design concepts are addressed.

KEYWORDS: theorem proving, hyper-linking, first-order logic, intelligent agents, distributed problem solving, collaborated work.

1 Introduction

For past decades, scientists have been interested in automated deduction systems which have applications in expert systems, planning, common sense reasoning, deductive databases, hardware/software verification, etc. However, finding the proof of a hard problem may require tens of hours of execution on workstations due to the huge search space. This motivates many researchers to exploit parallel or distributed computation to produce high-performance deduction systems.

Many high performance parallel deduction systems have been proposed and implemented, such as those discussed in Section 6. Parallelizing an existing system can be studied from the machine level, or fine-grain, by executing machine instructions in parallel, to the operational level, or coarse-grain, by executing subprocesses of the system in parallel. The experimental results show that much computational effort in finer-grain parallelism is futile and not directive. Also, implementation of finer-grain parallelism is costly since there usually need (1) expensive hardwares such as computer systems with a number of processing units and high-speed memory modules and (2) a lot of human-hours for developing and testing parallel algorithms and (3) efficient and usually complicated communication protocols for sharing data. Attention has been shifted to coarser-grain parallelism or distributed processing [3].

Distributed processing is a kind of coarse-grain parallelism, which usually performs on several connected, homogeneous or heterogeneous, computer systems. In the Internet or intranet, there are a number of in-expensive computers, e.g., PC, connected and available for communication and computation. With standard connection protocols like TCP/IP and ease-to-use interfaces like browsers, the Internet provides a flexible and inexpensive platform for distributed computing and so for distributed theorem proving. Though distributed theorem proving on the Internet is one intuitive solution, we still have to carefully address the following issues.

1. Deductive systems are usually computational intensive. When they are distributed processed, the among of communication may degrade the performance of the whole system. The grainity to be distributed processed should be carefully designed.

2. Different deductive systems apply different directive parameters which may be set by heuristics or users' experience. Such parameters need to be carefully adjusted.

3. All computing machines should fast exchange information in a standard format, without devoting too many efforts in processing communication.

4. Deductive problem solving is goal-oriented, no matter it is done in centralized or in distributed.

An clearly known fact is that the reliability of the Internet is less than well-designed and tested multi-processor computers. Directly design and implementing a distributed framework for deduction system on the Internet may face the same problems we mentioned in finer-grain parallelism, especially there are several heterogeneous computing machines. In doing so, a peer-to-peer communication protocol among the member processors should be supported.

Software agents are computer programs capable of flexible, autonomous actions. In their most complex form, agents may persist over time, are capable of timely internal context-dependent reaction to sensed events, plan and initiate unique series of actions to achieve stated goals, and communicate with other agents (or people) toward those ends. Fortunately, with the advantages of software agents, distributed theorem proving can be achieved through the collaboration of multi-agents. Recently, Internet-based

collaboration of multi-agents is an interesting research topics and has been receiving a lot of attentions. In this paper, we propose a multi-agent framework for distributed theorem proving. We demonstrate our idea using the hyper-linking proof procedure(HLPP) [14] which is a refutational deduction method in first-order logic. In this framework, distributed theorem proving is achieved in clause-level by the collaboration of multi-agents. Agents solve the given problem and exchange information for sharing data and adjusting directive parameters. The design of the framework and the role of each agent are described. The interaction and the collaboration of agents are discussed. A simple, KQML-based [1], scheme has been designed for exchange of information among agents in a peer-to-peer style. The advantages of use of this framework include lower cost and higher expandability for distributed theorem proving, usefulness for solving large and complex problems, being easily to be applied to any other theorem proving techniques, and the ability of sharing and learning proving experience.

The rest of the paper is organized as follows. Section 2 gives a brief description of HLPP. Section 3 discusses the work to be collaborated for distributed theorem proving of HLPP. In Section 4, the architecture of the multi-agent framework is presented. The design of each agent is described in Section 5. Related researches are discussed in Section 6 followed by a summary of this study in Section 7.

2 The Hyper-Linking Theorem Proving

The hyper-linking proof procedure(HLPP) was proposed to eliminate the duplication of clauses occurring in many other deduction methods and proved to be efficient [14] compared with widely known systems such as OTTER [18], *sprfn* [19], and PTTP [22]. HLPP is a refutational clause-form deduction method, consisting of the following two processes: hyper-instance generation (HIG) and propositional satisfiability test (PSAT).

Suppose R is a set of clauses in first-order logic. HIG produces hyper-instances by performing unification between literals of clauses in R to instantiate these clauses. If $C = L_1 \vee \ldots \vee L_m$ is a clause in R, and $M_1, \ldots, M_m$ are literals from the clauses in R, with variables renamed to avoid conflicts, and Θ is a most general unifier such that $L_i\Theta$ and $M_i\Theta$ are complementary for all i, $1 \leq i \leq m$, then $C\Theta$ is called a *hyper-instance* of C. The set $\{(L_1, M_1), \ldots, (L_m, M_m)\}$ is called a *hyper-link*. We call C a *nucleus* and M_i the *electrons* of C. We denote by $H(R)$ the set of hyper-instances of all the clauses in R and by $Gr(R)$ the ground set of R where all variables in R are replaced by the same constant symbol \$. Let S_0 be the set S of input clauses and S_i be $S_{i-1} \cup H(S_{i-1})$, $i \geq 1$. HIG takes clauses in S_i as nuclei and generates all hyper-instances, $H(S_i)$. Then PSAT tries to decide the satisfiability of $Gr(S_i \cup H(S_i))$. If $Gr(S_i \cup H(S_i))$ is unsatisfiable, then S is unsatisfiable and we are done; if all clauses in $H(S_i)$ are contained in S_i, i.e., $S_i = S_i \cup H(S_i)$, then S is satisfiable and we are also done. Otherwise, one more round of HIG and PSAT is performed on S_{i+1}. HLPP is a saturation-based method, i.e., no clauses in S_{i+1} can be considered as a nucleus before all the hyper-instances of S_i have been generated.

The efficiency of the proof procedure can be improved by filtering hyper-instances through unit-simplification which is analogous to the operations in the Davis-Putnam procedure [8]. Suppose $\{L\}$ is a unit clause retained in the database and C is a generated hyper-instance and M is a literal of C. The following two simplification rules are applied to C:

1. Unit literal deletion. If M is an instance of the negation of L, then C is simplified to $C - \{M\}$.

2. Unit subsumption. If M is an instance of L, then C can be deleted from the database.

Here we call L a *unit-literal*. C is said to be *simplifiable* if M can be removed from C. If all the literals of C are removed, C becomes an empty clause which indicates logical unsatisfiability and HLPP may stop and the proof is done. The HLPP method can be described by the algorithms in Figure 1. Note that PSAT can be any propositional calculus decision procedure. The one we used was the Davis-Putnam procedure with dependency-directed backtracking.

Example 1 Let S be a set containing the following clauses:

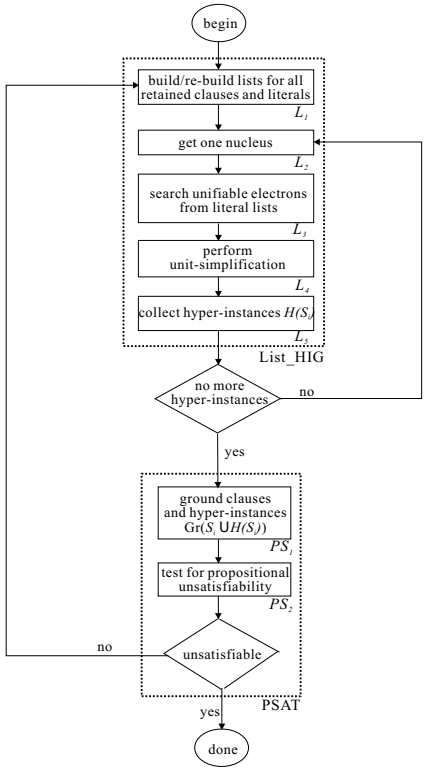
- $C_1 = \{p(X), \neg q(X), \neg s(Y, X)\}$
- $C_2 = \{p(X), \neg q(X), r(X, Y)\}$
- $C_3 = \{\neg p(X), q(X)\}$
- $C_4 = \{\neg r(X, b), s(b, Y)\}$
- $C_5 = \{p(a)\}$

From S, we have a list of negative literals $L_N = \{\neg q(X), \neg s(Y, X), \neg p(X), \neg r(X, b)\}$ and a list of positive literals $L_P = \{p(X), p(a), r(X, Y), q(X), s(b, Y)\}$ where $p(a)$ is a unit-literal. C_i, C_2, ..., C_5 are considered as nuclei individually. In each nucleus, every positive literal tries to find unifiable, complementary literals in L_N and each negative literal tries on L_P. Finally, we obtain 6 hyper-instances, after unit simplification, in the first round of hyper-linking, i.e., $H(S) = \{\{p(X), \neg q(X), \neg s(b, X)\}, \{p(X), \neg q(X), r(X, b)\}, \{\neg r(X, b), s(b, Y)\}, \{\neg p(X), q(X)\}, \{q(a)\}, \{p(a)\}\}$. Now we check in PSAT if $Gr(S \cup H(S)) = \{\{p(\$), \neg q(\$), \neg s(\$, \$)\}, \{p(\$), \neg q(\$), r(\$, \$)\}, \{\neg p(\$), q(\$)\}, \{\neg r(\$, b), s(b, \$)\}, \{p(a)\}, \{p(\$), \neg q(\$), \neg s(b, \$)\}, \{p(\$), \neg q(\$), r(\$, b)\}, \{q(a)\}\}$ is unsatisfiable. Since $Gr(S \cup H(S))$ is satisfiable and $S \neq S \cup H(S)$, another round of hyper-linking should be performed. □

Two different data structures have been developed for HLPP; one is list-based presented in [14] and another is network-based in [15]. Throughout this paper, the list-based version of HLPP is considered for its simplicity.

Procedure HLPP(S) /⋆ Hyper-Linking-Proof-Procedure ⋆/
$R \leftarrow S$;
while PSAT(R)=TRUE
    if R = HIG(R) /⋆ No more new hyper-instances ⋆/
    then return and print "S is satisfiable";
    else $R \leftarrow$ HIG(R);
return and print "S is unsatisfiable";
end HLPP
Procedure PSAT(R) /⋆ Propositional SAtisfiability Test ⋆/
if $Gr(R)$ is satisfiable *then* return TRUE;
                else return FALSE;
end PSAT
Procedure HIG(R) /⋆ Hyper-Instance Generation ⋆/
for each clause C in R
    find all hyper-instances of C;
    for each hyper-instance I of C;
        perform unit-simplification on I;
return $\{R \cup H(R)\}$;
end HIG

(a) Algorithms of HLPP



(b) Processing flow of HLPP

Figure 1: Algorithms of HLPP

3 The Computational Model

Parallelism of HLPP has been studied and presented in [27]. A three-workstations environment was developed for simulating a multi-processor computer with shared-memory. The analytical results show that the parallelism of HLPP can be achieved in different schemes, i.e., process-level, clause-level, literal-level, and flow-level, each of which associates different hardware architecture and synchronization problems and benefits different speedups. The purpose of this paper is to design, not just to simulate, a practical framework on the Internet for distributed theorem proving of HLPP. First, according to [27]'s classifications, we consider the degree of parallelism or distribution as follows.

- Process-level parallelism – simultaneously performing HIG(list-based or network-based version) and PSAT on two processing elements(PEs).
- Clause-level parallelism – performing PSAT on one PE and HIG(list-based version) on m-PEs each of which takes care of a partition of clauses. If the number of hyper-instances generated by each nucleus is almost the equal, the more PEs, the more speedups can be gained.
- Literal-level and Flow-level parallelism – performing PSAT on one PE and HIG(network-based version) by simultaneously exploring m-paths of the network with m-literals on m-PEs.

The maximum speedup can obtain in the process-level scheme is only 2. Also, when implemented on the Internet, the process-level scheme may result in many available but idle machines. Since theorem proving is computational intensive, the literal-level and flow-level schemes may not be adapted for massive access to the shared network structure. The contributions of these two schemes in distributed theorem proving may not be significant. The clause-level scheme may be a viable computational model.

However, directly and blindly distributing m clauses to m PEs on the Internet may not also gain a good result. Let's consider the experimental results of clause-level parallelism obtained in [27]. One of the most important things in this scheme is the handle of the synchronization problem. Suppose that n nuclei are assigned to n-PEs for performing HIG. Table 1 shows the status of each nucleus in generating hyper-instance. This table tells us that no all nucleus generate hyper-instances; the number of hyper-instances generated by each nucleus is not equal; some of them even generate no hyper-instances. Such a fact can be recognized by lower Avg. and higher SD. The computational environment we selected is the Internet or intranet on which the hosts are less reliable and of heavier communication traffic, compared with the well-designed and tested multi-processor computers. Since theorem proving is computational intensive, most efforts should be done on HIG and PSAT and less on communication or synchronization. These conclude that sophisticated dispatches of clauses is needed to gain a better performance of distributed HLPP in clause-level. In this paper, we use the techniques of software agents to take care of the tasks needed, such as monitoring, supervising, negotiation, and dispatching, for collaboratively completing a proof in HLPP.

Table 1: Values of $I_{/C}$ in each round (data obtained from [27])

#rth	Prob.	$I^H_{/C}$			$I^C_{/C}$			$I^I_{/C}$			#rth	Prob.	$I^H_{/C}$			$I^C_{/C}$			$I^I_{/C}$		
		Avg.	SD.	CV.	Avg.	SD.	CV.	Avg.	SD.	CV.			Avg.	SD.	CV.	Avg.	SD.	CV.	Avg.	SD.	CV.
1	ls112	0.1	0.3	0.3	0.0	0.0	-	0.3	0.8	0.4	1	tptp2	1.4	4.7	0.3	0.1	0.5	0.2	0.5	1.9	0.3
2		5.7	12.1	0.5	0.3	1.1	0.3	0.0	0.0	-	2		5.1	15.6	0.3	2.4	4.9	0.5	3.8	11.3	0.3
3		3.5	19.4	0.2	4.8	14.3	0.3	19.2	147.4	0.1	3		5.6	39.1	0.1	63.8	189.0	0.3	99.0	438.2	0.2
	Avg.	3.1	10.6	0.3	1.7	5.1	0.3	6.5	49.4	0.3		Avg.	4.0	19.8	0.3	22.1	64.8	0.3	34.4	150.5	0.3
1	zebra	0.3	1.1	0.3	0.0	0.0	-	3.5	23.6	0.1	1	wos31	0.7	2.4	0.3	0.5	2.4	0.2	0.9	2.5	0.4
2		10.1	68.9	0.1	3.0	21.7	0.1	0.3	2.4	0.1	2		6.5	20.5	0.3	2.6	8.0	0.3	69.4	329.8	0.2
3		0.1	1.8	0.1	2.2	33.7	0.1	0.5	11.7	0.0	3		7.4	45.1	0.2	102.1	720.4	0.1	138.5	1339.8	0.1
	Avg.	3.5	23.9	0.2	1.7	18.5	0.1	1.4	12.6	0.1		Avg.	4.9	22.7	0.3	35.1	243.6	0.2	69.6	557.4	0.2
1	example	0.4	0.5	0.8	0.0	0.0	-	0.0	0.0	-	1	ex4t1	0.9	0.9	1.0	0.6	0.7	0.9	2.0	2.9	0.7
2		0.9	1.1	0.8	0.6	0.7	0.9	0.0	0.0	-	2		1.7	1.9	0.9	6.0	3.9	1.5	7.0	7.8	0.9
3		4.2	5.3	0.8	4.2	3.9	1.1	14.1	17.7	0.8	3		1.3	1.6	0.8	18.2	9.6	1.9	29.8	43.6	0.7
4		2.4	7.3	0.3	33.1	70.6	0.5	28.5	76.0	0.4	4		1.2	2.0	0.6	116.1	149.0	0.8	63.8	81.0	0.8
	Avg.	2.0	3.6	0.7	9.5	18.8	0.8	10.7	23.4	0.6		Avg.	1.3	1.6	0.8	35.2	40.8	1.3	25.7	33.8	0.8
1	schubert	0.5	0.7	0.7	0.3	0.5	0.6	0.0	0.0	-	1	jobs	0.2	0.8	0.3	0.0	0.0	-	0.0	0.0	-
2		2.0	9.7	0.2	0.8	0.9	0.9	0.2	0.7	0.3	2		1.9	7.8	0.2	0.2	0.8	0.3	1.2	6.0	0.2
3		6.0	38.3	0.2	4.4	11.0	0.4	25.6	150.0	0.2	3		0.6	2.2	0.3	11.3	90.2	0.1	2.9	24.4	0.1
4		2.8	17.2	0.2	44.9	186.3	0.2	47.5	281.4	0.2	4		0.3	1.8	0.2	9.4	91.7	0.1	0.1	2.0	0.1
		-	-	-	-	-	-	-	-	-	5		0.1	0.8	0.1	6.5	64.4	0.1	1.4	17.4	0.1
	Avg.	2.8	16.5	0.3	12.6	49.7	0.5	18.3	108.0	0.2		Avg.	0.6	2.7	0.2	5.5	49.4	0.1	1.1	10.0	0.1

Abbrec.: $I_{/C}$: the number of hyper-instances generated from a clause in a round of hyper-linking, $I^H_{/C}$: the number of distinct hyper-instances, $I^C_{/C}$: the number of hyper-instances that are duplicates of existing clauses, and $I^I_{/C}$: the total number of hyper-instances having been generated. Avg.: average, SD.: standard derivation, and CV.=Avg./SD.: coefficient variance.

4 The Multi-Agent Framework

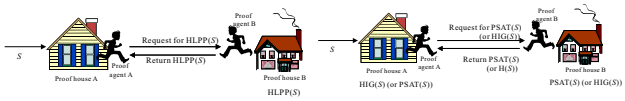
4.1 The Architecture

The main design concepts of this framework are as follows. Suppose P is a problem to be proved in first-order logic and S is the set of clauses corresponding to P and S is input to a host A. The system A can perform HLPP on S and solve P solely. However, let us consider the following cases.

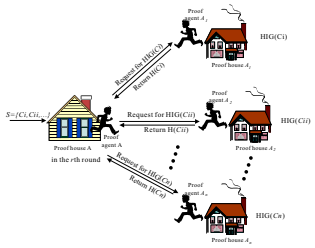
- If there already are many tasks to be served in A or the performance of A will be degraded badly by solving P, it may be good to pass the proof of P to some other hosts whose performance is better than A.
- If A accepts the assignment of P but, after some rounds of HLPP, it seems that P is a very hard problem which may take much longer time to complete or the performance of A is degraded by P, it may be needed to distribute some tasks of HLPP such as HIG or PSAT or unit-simplification from A to other hosts.
- Similarly, in the ith round of HLPP, if there are a lot of clauses, C_1, C_2, ..., retained in the database and many of them produce a considerable among of hyper-instances, it may be helpful to share the load of HIG(C_j), $j \geq 1$, with some other hosts.

However, hosts on the Internet may be in different operational platforms and may be turn on or shutdown arbitrarily. The performance of these hosts is not always predictable all the time. Tracing the progress of each host and monitoring the performance of the system are two issues to be seriously considered. The framework should designate the following improvements.

- Recognizing how many available hosts are there on the network.
- Deciding the types of collaboration to perform.
- Adjusting the clauses/literals to be forwarded to other hosts.



(a) A bypasses HLPP(S) to B (b) HIG(S) on A and PSAT(S) on B



(c) HIG(C_i) on A_i, A_{ii}, A_n

Figure 2: The types of collaboration among agents

- Monitoring the traffic and predicting the turnaround time of each host.
- Communicating for exchanging or sharing data, knowledge, or even proof experiences among hosts in a peer-to-peer style.

With the techniques of software agents, HLPP can be wrapped as agent applications and the improvements mentioned above can be implemented as agents. Collaboration of multi-agents may be a solution to distributed theorem proving in clause-level, serving as a flexible and convenient framework. We call each host as a *proof house* and the agent resided in a proof house a *proof agent*. The types of collaboration among the proof agents are illustrated in Figure 2. Figure 3 presents the high-level architecture of our idea. The main components are as follows.

- Proof Houses. The proof house are computer systems which like to perform distributed HLPP. The core program of a proof house is HLPP whose

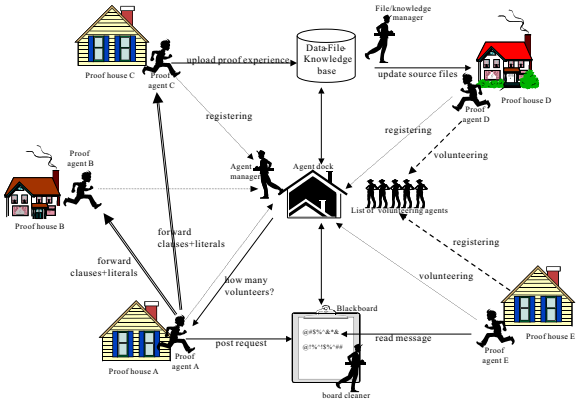


Figure 3: The high-level architecture of the system

tasks, such as HIG, PSAT, unit-simplification, etc., can be invoked individually by the users or agents.

- Proof Agents. In each proof house there resides a proof agent which invokes HLPP and dispatches clauses/literals and communicates with other agents and monitors the performance of the proof house.

- An Agent Dock. The agent dock is the pool gathering available proof agents. Before starting collaboration, a proof house has to register in to the agent dock. An *agent manager* resides in the agent dock, which names the agents, associates the agent's name and IP, monitors the liveness of the registered agents, and lists the volunteering agents for a specific request proof houses, etc.

- A Blackboard. The blackboard is an open place where agents post requests for help and answers corresponding to the requests. Such requests include to perform HIG or PSAT on a specific set of clauses, to simplify a set of clauses, to find a unit clause, etc. Proof agents read the posts on the blackboard; and if they estimate that the request can be served, they volunteer to the agent dock. A request on the blackboard has three status, i.e., new, served, and out-of-date, indicating requests that are newly post by a proof agent, under serving or processing, and not satisfied by any agents after a long time, respectively. A *board cleaner* takes care of listing volunteers and the clean work.

- A File/Knowledge Server. This server performs two tasks, updating/installing the latest version of HLPP and sharing proof knowledge/experience to proof houses. Since proof houses are of different platforms, e.g., Unix, Windows 95, MacOS, or Linux, the needs for the source codes of HLPP are platform-dependent. Also, knowledge or information like the check-in period, set-of-support strategies, the order of tries of unification in HIG, etc., are set as default parameters and can be reset according to the suggestion of agents. Proof agents can forwards their experiences and the agent, *file/knowledge manager*, summarizes and "pushes" the results togethered with the latest version of HLPP periodically.

4.2 The Interactions

Each computer can work solely or can collaborate with other registered provers to prove the given problems. When working together, provers have to decide how many and which clauses to be sent outward and to expect the returning messages from the other provers. Suppose a proof house A accepts a problem P to be proved. The interactions among agents are described as follows.

- Initiating. A converses P into a set of clauses S as the sequential version of HLPP does.

  If the proof agent A recognizes that the performance of A is good for solving S, it sets up a timer and invokes the HLPP procedure.

  If not, the proof agent can either refuse to accept the proof of P or call for help on the blackboard for performing HLPP on S.

- Performance Tuning. Suppose A accepts the proof of P and performs HLPP on S. The proof agent monitors the progress of HLPP. If the operation of HLPP exceeds a predefined among of time but still not obtain a proof result, the proof agent on the blackboard a request for help.

  If the time spent in HIG and PSAT are significantly different, the request is to submit a time-consuming task to a proof house with better performance.

  If HIG takes a very long time in each round, the request is to distribute the clauses/literals for HIG on n proof houses.

- Volunteering. Periodically, the proof agent living in a well-performed proof house visit the blackboard and picks up the posted requests. If a request is acceptable, it checks in to the agent dock and claims that the proof house it resided is available for collaboration. The agent dock queues the names of such volunteering agents.

- Collaboration. After the round of HLPP, in which the agent posted a request is finished, the proof agent communicates with the agent dock and requests for m available proof houses. The set of clauses is then divided into $m+1$ disjointed subsets m of which are sent to the m proof houses together with the set of literals.

- Feeding Back. When a proof house finishes the assignment, the proof agent sends the result, e.g., new hyper-instances, back to the requester. The proof agent initiates the request then waits for assembling the feedback information and completes the proof.

- Experience Sharing. In solving the given problems solely or performing HIG collaboratively on the given clauses, information such as

  patterns of clauses usually generating a lot of duplicated or redundant hyper-instances,

  patterns of clauses for the PSAT module to complete the proof, or

  patterns of problems which usually can be solved efficiently with backward-/forward-support strategies and a specific sliding-priority, etc., may be helpful to future collaboration. Such information can guide or adjust the directive parameters

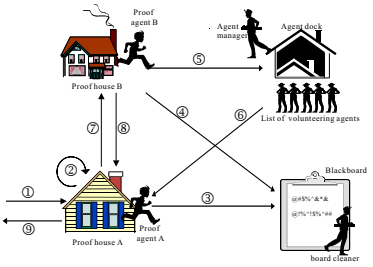


Figure 4: Collaboration of agents for distributed theorem proving

of the prover and improve the performance. These information are forwarded to the knowledge server wherein they are summarized and analyzed.

Figure 4 presents the above interactions. Note that, the above interactions may occur multiple times when proving P. Also, a volunteering agent can continue its assigned jobs, e.g., text processing, ftpping, web browsing, etc., and takes the collaboration of theorem proving as one of regular jobs. In other words, we are stealing available computation time from each member computers.

5 The Design of Agents

Like many other mobile software agents, the agents in our work have different capabilities of computing or proving, knowledge for reaction, and methods for communication. The agents introduced in this paper include proof agents, agent manager, board cleaner, file/knowledge manager. The follows discuss the issues of design of the agents.

5.1 The Life of Agents

Basically, each agent is hosted in a computer system connected to the Internet or intranet via TCP/IP. Each agent is associated with a name, an IP address, and a knowledge base. System agents, i.e., the agent manager, the board cleaner, and the file/knowledge manager, are named by default using *name_IP* pair such as, `AgentManager_210.71.14.33`, `BoardCleaner_210.71.14.36`, and `FileKnowManager_210.71.14.66`, respectively. Proof agents residing in the proof houses are named by the agent manager when they register in to the agent dock. The naming scheme of proof agents uses the tuple of *name_time* such as `john22_24316367`, where *time* is given according to the clock of the agent dock. Since the time proof agents registered in are different, the agent manager can give to each proof agent a unique name.

The agent dock associates the name and the IP number where the agent is from. The agent manager periodically visits the registered proof agents to make sure they are alive. When a proof house is turned off or the network is disconnected, the agent manager unregisters the agent and releases the name after waiting for a fixed period of time. Also, if the consigned task is not finished, the agent manager sends message to the caller to re-consign another agent. Some waiting lists are designated to the agent dock, which are lists of available agents volunteering to different requests posted on the blackboard. The caller can then access to the waiting lists and get the name and IP and forward associated data(clauses and literals) to the volunteers via direct ftp. The board cleaner watches the messages posted on the blackboard and traces their status of being served. The file/knowledge manager knows from the agent manager the platforms of proof agents and updates the source code of HLPP. Proof agents can upload their proof experiences and data to the server. The file/knowldge manager periodically analyzes the uploaded data using statistics or simple data-mining techniques and forwards the summarized data to the agents registered in to the agent dock.

System agents keep alive and are running permanently after installation and the IP number can be changed only by re-installing the whole framework. However, in doing this, all te working-on proof agents may lost the the links for communication to other proof agents. The user can turn down a proof house through the user interface with or without notifying the agent manager. Proof houses re-connecting to the agent dock using the same IP may receive a different name.

5.2 Working Knowledge

Agents work in an events-conditions-actions cycle. Each agent is working with a knowledge base containing pre-defined, essential responses and actions for collaboration or communication. Different agents work with different knowledge bases. Currently, the knowledge bases are mainly constructed by IF-THEN rules. Below we list part of the rule-based working knowledge associated with agents.

- IF the running time of HLPP exceeds the default value, THEN post a request for help on the blackboard.
- IF the performance of the host machine is fine AND there is a request on the blackboard, THEN volunteer for the service.
- IF the request is to generate hyper-instances for a set S of clauses AND there are n volunteers ready for service, THEN partition S into n disjointed sets, S_1, S_2, ..., S_n, AND pass S_i to the ith volunteer together with the set of literals retained in the database.
- IF the request is to perform HIG on a set S of clauses and a list L of literals, THEN call the HIG procedure on S and L AND wait for the host to return H(S) to the caller.
- IF the request is to perform PSAT on a set S of clauses, THEN call the PSAT procedure on S AND wait for the host to return PSAT(S) to the caller.
- IF the problem has been solved, THEN inform all of the volunteers to terminate the consigned task and to discard all the registered data.

5.3 Communication

System agents and proof agents communicate with each other using KQML(Knowledge Query and Manipulation Language) which was defined under the DARPA-sponsored Knowledge Sharing Effort [1]. Basically,

KQML provides a way to structure the messages in a layered architecture with the functionality of message communication. Between the layers are the primitives, or performatives, with which agents can exchange meaningful message. The KQML performatives are assertive and directive and used to perform actions like "tell", "evaluate", "subscribe", or change agents' states, etc.

6 Related Work

PARTHENON [5] is a parallel resolution theorem prover implemented on multiprocessors with shared memories. PARTHENON uses the model elimination procedure and explores alternative branches of the proof tree. Fine-grain parallelism is supported by parallel OR-search. The ROO theorem prover [17] is a parallel version of OTTER, which is implemented on a shared-memory architecture on which several clauses are activated in parallel. SPTHEO [24] is a parallel version of a sequential theorem prover, SETHEO, by combining parallel OR-search and AND-search. SETHEO is a top-down prover based on the calculus of connection tableaux which generalizes weak model elimination. Proofs are found by a consecutively bounded depth-first iterative deepening search with backtracking. DADO [23] is a tree-structured parallel architecture in which the processing elements form a complete binary tree. Programs running on DADO are first partitioned into 16-32 groups each of which is executed in a processing element. DARES [7] is a parallel version of a resolution-based method working on a distributed network. A number of processes, called agents, each of which sequentially executes resolution on part of the clause set, work concurrently. The search space is partitioned into parts according to some heuristics. Each agent works on a partition of the search space and communicates with other agents by message passing. Different theorem proving methods may utilize different parallel strategies to make use of their distinct characteristics [3]. Therefore, it is hard to compare the performance among different parallel theorem proving systems. Readers may refer to [4, 11, 12] for a broader discussion. Agent-based collaboration on the Internet has been receiving great attentions [6, 13, 10, 28, 26, 25]. Many agent-based applications have been developed and presented. Successful applications include BIG [16] for resource-bounded search, DNX [9] for intelligent registration for domain names, NetChaser [21] for personal mobility of using computer terminals, Nomad [20] for auction on electronic commerce, XPECT [2] for electronic commerce framework, etc., and many others applied to CAD, information retrieval, personal assistant. However, rare of them are for deductive systems.

7 Concluding Remarks

We have presented a multi-agent framework for distributed theorem proving of the hyper-linking proof procedure. The advantages of this framework include the following.

- Any trusted computer system can contribute to distributed theorem proving by volunteering to be a proof house. This means that the number of proof houses can be extended considerably. Compared to the multi-processor systems, the cost of adding a processing element in our work is much smaller. Also, the time spent on developing parallel theorem provers is significantly deduced.
- This framework is very useful for solving large and complex problems which usually need considerably long running time.
- The concepts and techniques proposed in this paper can be applied to any other theorem provers.
- The use of agents simplifies the complexity of communication among the contributed components since the proof agents talk in a peer-to-peer style, ignoring the details of the physical communication protocols.

We conclude with the following remarks. In this research, each proof house can prove problems solely and request for collaboration when it "feels" it is needed. As stated in [7], theorem proving is computationally intensive and the processing elements should be loosely coupled, for contributing more time to computation than to communication. However, the reliability of the Internet is not as high as multi-processor systems. The unexpected delays and turn-around time are the risks of using this environment. To overcome these, each proof agent can set up it knowledge base or learn the change of environment and dynamically determine the related parameters. Also, the agent dock monitors the liveness and performance of agents and always provides reliable proof houses for any request. Moreover, the knowledge manager can inform the user or the proof agent to tune the related parameters.

Furthermore, the performance and quality of distributed inference could be improved by mining the knowledge base for obtaining a better combination of parameters. Load-balance is an important issue for parallel or distributed processing. Using more accurate evaluating measures to estimate the potential behavior of the underlying problem might help, in the sense that agents can collaborate on complex tasks. Moreover, exploring better heuristics for tuning parameters is worth advanced studies. Currently, in the proposed environment, HLPP is performed round by round. Continuously performing HIG and PSAT on the generated hyper-instances is another computational model, which will be our next direction. The concept of TEAMWORK can also be applied to agent-based collaboration. In this case, each proof house can run theorem provers with different directive parameters, or even run with different theorem proving techniques. A research project has been working on this topic.

References

[1] The KQML specifications in 1993, 1993. http://java.stanford.edu/KQMLSyntaxBNF.html.

[2] J. Andreoli, F. Pacull, and R. Pareschi. Xpect: a framework for electronic commerce. *IEEE Internet Computing*, 1(4):40 –48, July-Aug. 1997.

[3] M.P. Bonacina and J. Hsiang. Parallelization of deduction strategies: An analytical study. *Journal of Automated Reasoning*, 13:1–33, 1994.

[4] M.P. Bonacina and W. McCune. Distributed theorem proving by peers. In *Proceedings of the 12th International Conference on Automated Deduction*, pages 841–845, 1994.

[5] S. Bose, E.M. Clarke, D.E. Long, and S. Michaylon. PARTHENON: A parallel theorem prover for non-Horn clauses. *Journal of Automated Reasoning*, 8(2):153–182, April 1992.

[6] F.W. Bruns and H. Gathmann. Auto-erecting agents for a collaborative learning environment. In *Proceedings of the 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 287 –288, 1999.

[7] S.E. Conry, D.J. MacIntosh, and R.A. Meyer. DARES: A distributed automated resolution system. In *Proceedings the 11th National Conference on Aritificial Intelligence*, pages 78–85, 1990.

[8] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[9] Lassaâd Gannoun, Julien Francioli, Stanislav Chachkov, Fréderic Schutz, Jarle Huaas, and Jürgen Harms. Domain name exchange: A mobile-agent based shared registery system. *IEEE Internet Computing*, 4(2):59–64, March/April 2000.

[10] J.N. Hansoty, M. Vouk, and S.F. Wu. Lava: secure delegation of mobile applets: design, implementation and applications. In *Proceedings of the Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1997.*, pages 242 –247, 1997.

[11] H. Hong. Special issue on parallel symbolic computation: Foreword of the guest editor. *Journal Symbolic Computation*, 21(4):377–376, 1996.

[12] K. Konrad. HOT: A concurrent automated theorem prover based on higher-order tableaux. In *Lecture Notes in Computer Science*, volume 1479, pages 245–262. Springer, 1998.

[13] Andrew P. Kosoresow and Gail E. Kaiser. Using agents to enable collaborative work. *IEEE Internet Computing*, pages 85–87, July/Aug. 1998.

[14] S.-J. Lee and D. Plaisted. Eliminating duplication with hyper-linking strategy. *Journal of Automated Reasoning*, 9(1):25–42, 1992.

[15] S.-J. Lee and C.-H. Wu. Improving the efficiency of a hyper-linking based theorem prover by incremental evaluation with network structures. *Journal of Automated Reasoning*, 12:359–388, 1994.

[16] Victor Lesser, Bryan Horling, Anita Raja, Xiaoqin Zhang, and thomas Wager. Resource-bounded searches in an information marketplace. *IEEE Internet Computing*, 4(2):49–58, March/April 2000.

[17] E.L. Lusk and W.W. McCune. Experiments with ROO: A parallel automated deduction system. *Parallelization in Inference Systems*, 590:139–162, 1992.

[18] W. W. McCune. *OTTER 3.0 Users' Guide*. Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, January 1994.

[19] D. Plaisted. Non-Horn clause logic programming without contrapositives. *Journal of Automated Reasoning*, 4:287–325, 1988.

[20] T. Sandholm and Q. Huai. Nomad: mobile agent system for an internet-based auction house. *IEEE Internet Computing*, 4(2):80 –86, March-April 2000.

[21] Antonella Di Stefano and Corrado Santoro. NetChaser: Agent support for personal mobility. *IEEE Internet Computing*, 4(2):74–79, March/April 2000.

[22] M.E. Stickel. A Prolog technology theorem prover: Implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.

[23] S.J. Stolfo and D.E. Shaw. Architecture and application of DADO: A large-scale parallel computer for artificial intelligence. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1983.

[24] C.B. Suttner. SPTHEO: A parallel theorem prover. *Journal of Automated Reasoning*, 18:253–258, 1997.

[25] P. Tarau, V. Dahl, and K. De Bosschere. A logic programming infrastructure for remote execution, mobile code and agents. In *Proceedings of the Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 106 –111, 1997.

[26] A. Wallis, Z. Haag, and R. Foley. A multi-agent framework for distributed collaborative design. In *Proceedings of the Seventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1998. (WET ICE '98).*, pages 282 –287, 1998.

[27] C.-H. Wu and S.-J. Lee. Parallelization of a hyper-linking based theorem prover, accepted by and to appear. *Journal of Automated Reasoning*, 2000.

[28] Jerome Yen, Alan Chung, Heron Ho, Birgitta Tam, Rocky Lau, Michael Chua, and Kai Hwang. Collaborative and scalable financial analysis with multi-agent technology. In *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences*, 1999.