A PEER TO PEER PLATFORM USING SANDBOXING

Fabien HANTZ
Hervé GUYENNET
LIFC: Laboratoire d'Informatique de l'Université de Franche-Comté
FRE CNRS 2661
16 Route de Gray, 25030 BESANÇON CEDEX, FRANCE
E-mail: {hantz,guyennet}@lifc.univ-fcomte.fr

Keywords—**P2P-Computing, DAG, sandboxing**

Abstract—**HiPoP is a P2P platform of computing specialized in the execution of applications as DAG (Directed Acyclic Graph). This platform works in an heterogeneous environment and relies on the portability of the JAVA language. We secure resources lent by volunteer by the use of HDS (HiPoP Dynamic Sandbox). Moreover the communication protocol of our platform fight against the attack "Man In The Middle".**

I. INTRODUCTION

To secure a system is not an easy thing to do. In client/server applications it is possible to secure communications by using the public certificate of the server. Thus, clients can authenticate servers, communications can be encrypted and messages stay confidential, messages are signed to check their integrity and the extensibility of a such system can be obtained using credential.

In P2P system, there is no such procedure to secure a system because nobody can have confidence in nobody. The use of one certificate by users is to avoid because the signing of the request certificate is not scalable. But above all, signing a request certificate aof a user which just provide an email address doesn't directle take a great interest.

A big number of platforms exists but are not used because of their difficulties to install, administrate, use,... Thus, we can't forced volunteer to follow a tedious and long procedure to secure its system. All must be simple and transparent as much as possible. The main goal is to permit user to share their resources in a secure wy in order to increase their power computation and to be able to perform application they can't do locally on their own PC.
HiPoP is not a classical P2P-platform because it is submitted to a set of precedance constraints due to its specialization to carry out DAG. To secure it is then again more difficult. This article presents the solutions we use to offer a minimal security solution for user resources.

In this article, we firstly describe DAG that we use to perform parallel applications. We carry on with the descriptions of HiPoP architectures and HiPoP functionning. Thus we presents the sandbox concept and how we made to adapt it, in a dynamic way, to our configuration. We then detail the protocol modifications we made to protect against the attack "Man-In-The-Middle". The next section deals with the problem of confidentiality of data and applications. Finally, we announce one of our short-term prospect and we conclude.

II. DAG

As we saied it, HiPoP is a P2P platform specialised to perform applications as DAG. Its functionning is dependant of the DAG to perform. It's why we formaly detail a DAG.
A DAG (Gerasoulis & Yang 1993), (Chen & Maheswaran 2002), (Suter et al. 2003) is a data structure used to represent the dependences among the tasks of an application. Let $G = (J, E)$ be a DAG which models an application, where $J = J_i : \{i = 1, ..., N\}$ is a set of N nodes, jobs or tasks and $E = E_{i,j}$ is a set of edges. $Pred(J_i)$ denotes the set of immediate predecessors of the task J_i and $Succ(J_i)$ is the set of immediate successors of the task J_i. $Pred(J_i) = \emptyset$ means that J_i is an entry task because it has no predecessor. In the same way, $Succ(J_i) = \emptyset$ means that J_i is an exit task because it has no successor. An edge in a DAG corresponds to dependences in terms of communications or precedence constraints. Typically, one notes by the precedence relation $J_i \rightarrow J_{i'}$ the fact that $j_{i'}$ can't begin as long as J_i isn't entirely finished. Generally speaking, a task is ready to be computed when the set of its predecessors have finished their execution. Each task is a node in the DAG and each node can have unspecified number of parents as long as there is no loop. **Fig.** 4 gives an example of DAG.

III. HIPOP ARCHITECTURES

Our goal is to build a light platform, simple of use and easy to administrate. The different architectures of HiPoP and its functioning are described in this section.

A. *Software architecture*

The HiPoP platform is composed of 5 elements:
• RP (Resource Provider): The goal of the Resource Provider is to register and to provide references of resources. Sometimes we also name it referencor because it just gives some references. Each RD registers on the RP to be available to perform calculation for other RD.
• Resultd: Resultd is the application which gets results of calculations submitted by the client. We also name it results

collector.

- RD: RD is the peer-to-peer application itself. It manages relations with other peers. It can be client or worker for the other peers. This dual role is sometimes called servent.
- hipop_submit: hipop_submit is the application used to submit a DAG to the platform.
- dagdesigner: dagdesigner is a tool that builds graphically DAG to perform. This tool is not essential because DAG can be built manually or generated by a third generator using HiPoP libraries.

B. *Hardware architecture*

Contrary to platforms based on client-server architecture, HiPoP doesn't have to take care of the set of machines/resources of the server side. Indeed, except the RP which is a dedicated machine, the rest of the platform is composed of machines of volunteer which are applicants or providers of power calculation. A resource is a machine connected to the Internet having computation power (CPU, HD, free memory space).

C. *P2P discussion*

We saied a system is a P2P system when it is fully decentrallized. With this definition, a few of system can really be qualified of P2P. HiPoP is part of them because of the Resource Provider. We choose to use a centralized Resource Provider for several reasons :

- We don't want to contact resources which are ever use
- For security reasons we need to the minimum a trusted machine
- The actions of the Resource Provider are very limited thus it is not necessary to distribut it.

HiPoP s a P2P system in the sens where there is no central scheduler. Each peer schedule its successors. The load of the system is then balanced enough.

IV. FUNCTIONNING OF HIPOP

A. *Dedicated resources*

Resources implied in the platform are resources lent by users. These resources are, in general, never used at 100% so we can employ them for personal or professional use and at the same time imply them in the platform. Nevertheless, the trouble caused by the execution of tasks in user's machines can urge him to unsubscribe to the platform. So it is imperative not to overload the user's machine. In order to do so, we submit only one task by resource. We say that a resource is dedicated to the execution of one task. Although that doesn't ensure a weaker use of the processor, to dedicate a resource to the execution of only one task restricts the memory occupation in decreasing the number of instance of JVM launched and avoid to machines equipped with a weak memory capacity to "swap" and to be hampered.

Say that the machine M_j executes only one task at the same moment means that at any moment t, the $\sum_i \delta_i^j(t) \leq 1$ inequality is verified. In other words, at the most a dash $i^+ \in [0,n]/\delta_{i^+}^j(t) = 1$. Consequently, if a resource disappears, the whole execution of the DAG is stopped. It is therefore imperative to implement a fault tolerance procedure. This point is approached in section IX.

B. *The resource deamon*

The peer-to-peer (Shirky 2000), (Crowcroft & Pratt 2002), (Chawathe et al. 2003) architecture of HiPoP leans on the ResourceDeamon application which is the main element of the platform. RD almost manages all the communications of the platform as well as the execution of the tasks of the DAG. As we have already noticed previously, resources are dedicated to the execution of one task. Thus, a RD executes only one task at the same time. The functioning of the ResourceDeamon is detailed in this subsection.

Actions carried out by J_k, a task located in an unspecified position of a DAG $\mathcal{G}$:

- $\forall d(i,j) \in D/j = J_k$ makes the download of $d(i,j)$
- performs the task J_k itself
- $\forall d(i,j) \in D/i = J_k$: asks a resource to the Resource Provider for the execution of J_j. The resource M_j obtained is like $M\prime = M - M_j$, $M\prime$ being the new set of resources
- $\forall d(i,j) \in D/i = J_k$: makes the delivery of $d(i,j)$

C. *Particular case*

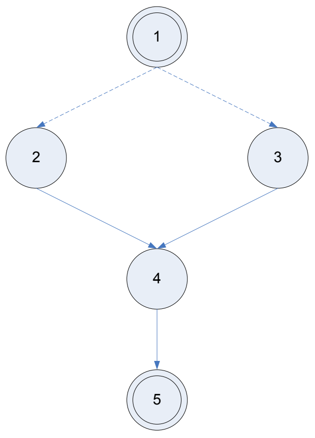

Fig. 1. **Synchronisation**

In the DAG of **Fig.** 1, J_1 and J_5 are respectively the entry and exit tasks. These tasks which do not perform any calculation are always taken in charge by the user's machine which has submitted the DAG. The entry task, besides being in possession of the entry data of the DAG, partially performs the role of coordinator between the tasks of the DAG. Partially, because it is far from being a central scheduler which makes all decisions. It only intervenes in particular cases such as the one shown in **Fig.** 1. Indeed J_4 is contained at once in the successors of the tasks J_2 and J_3. In this situation, only one task must ask a resource to the Resource Provider to perform J_4. That is precisely here that J_1 intervenes because it will be able to say to J_2 and J_3 which task can request a resource.

When the Resource Provider sends the reference of one resource to a resource which requests one, this reference is removed from the list of available resources. To monopolize the resource which has to perform J_4 the shortest possible time, the task which ends last its execution is elected to make the request.

D. *State of the RD*

The RD can take different states during the execution of a DAG, according to its execution step of the task and its role : client or server. **Fig. 2** shows these states.

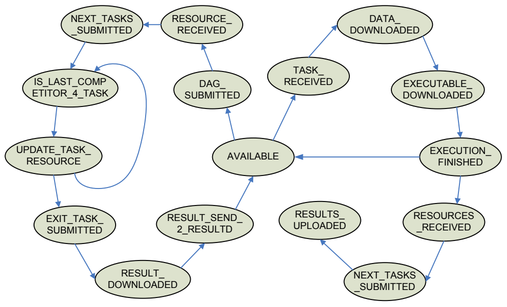

Fig. 2. **State of the RD**

The departure state is the AVAILABLE ONE, when a peer is registred to the Resource Provider. To the left part of the figure, we have the states that a RD can take when a DAG has been submitted to him. On the opposite side, we can see the states that a RD can take when a task has been submitted to him by another RD. A RD becomes available before to go through the RESULTS_UPLOADED state, this is for the case where the number of availaible resources is lower than the number of tasks to carry out.

V. SANDBOX

In compute security, a sandbox is a security mechanism for safely running programs owning to untrusted users. All the permissions like network access, the inspection of host systems or the reading from input devices are disallowed or heavily restricted. The sandbox prevents the scratch space on disk and memory by a hard controll of the guest programs running in a resource. A lot of kinds of sandboxes exists like applets, jails, emulation.

VI. HIPOP DYNAMIC SANDBOX

In HiPoP, peers are either clients or servers. If they are servers, they perform tasks submitted by client peers. These peers can be trusted peers or peers which disguise themselves as trusted peers. Thus, it is dangerous to allow anyone to execute anything on its own machine. Very few users would be ready to provide its resources with such a big risk. The concept of sandboxing (Oaks 2001), (Network (1997) 1997), (Gong et al. (1997) 1997) answers this problem by analyzing and by limiting the actions of an application. Each time an application requires particular rights (read, write, connect...) the sandbox authorizes or not these actions according to a certain number of privileges allocated to the execution of these applications. Many systems already use the concept of sandboxing but many do it in a static way. I.e. they allocate a list of privileges/permissions/restrictions which remains unchanged from the start to the end of the execution of their system. Even though the needs for the applications evolve in time. One could for example allows connection of a peer at a moment and refuses it later. This is this kind of dynamic allocation we put in place using the JAVA sandboxing. We set some default permissions VI-A which are completed by dynamic permissions VI-B during the execution of a DAG.

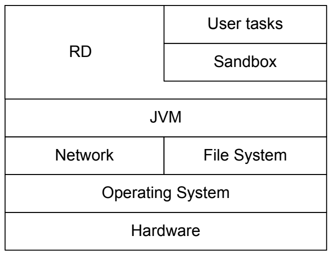

Fig. 3. **HiPoP Dynamic Sandbox representation**

Fig. 3 shows that user tasks have to go through the sandbox to be allowed to achieve the network or file system layers.

A. *Default permissions*

By default, the permissions given to a Resource Daemon is very limited. It can :

- connect to the Resource Provider to register itself or to obtain the references of other Resource Deamon
- be contacted by the Resource Provider which verifies that it is joinable from the outside
- be contacted by the local hipop_sumbit application to perform a DAG
- contact the local Resultd application to send it a result

All other permissions are disallowed. Moreover, users can't launch the RD in overloading the permissions using a ".policy" file. The permissions are managed in the source code and it is not possible to use another SecurityManager. The reading of system informations are unauthorized too.

B. *Dynamic permissions*

B.1 *Network Access*

During the execution of a DAG, peers need to contact and to be contacted by other peers. This subsection presents which peer are allowed to contact and to be contacted to which peers and how its permissions are given.
When a peer M_1 finishes the execution of a task J_1 ("EXECUTION_FINISHED" state), it asks some resources to the Resource Provider with the intention of submitting them the tasks in $SUCC(J_1)$. In order to accept these submissions, the Resource Provider informs chosen resources that they will have to accept the connections from M_1. In this case, affected resources have not to be open to all the other peer but just to M_1. When all the successors of M_1 will have downloaded the results of M_1, they will remove this permission and M_1 will not be able to contact them again.
RD_CLIENT must accept connections from all the resources used in the DAG because of the synchronisation process. RD_CLIENT knows all the resources implies in the DAG it carries out because when a resource obtains resource references from the RP to perform its successors, it communicates them to the RD_Client. **Fig 4** details the sequence of

messages need in order that $M4$ accept a connection from $M1$ to perform a task.

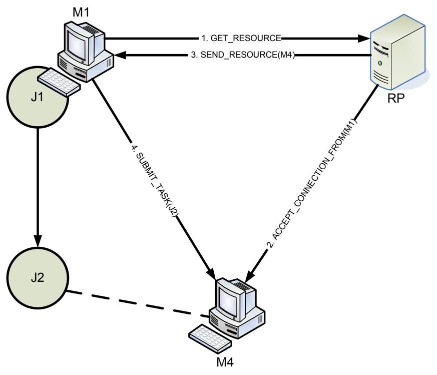

Fig. 4. **Messages sequence for netwok access**

B.2 *File Permissions*

A resource can only read files contained in a tempory directory identified by the task it executes. All other reading is forbidden. For instance, it is not permitted to read the ”/etc/passwd” file.

VII. MAN IN THE MIDDLE ATTACK

When a peer registred itself to the Resource Provider, it is not added to the list of available resources without carrying out checks. Indeed, the Resource Provider try to contact it. If the connection is successful, it means that it is the good peer and not a peer which uses its IP address for the add request. On the contrary, either the peer is an untrusted peer, either the peer is misconfigured. In that last case, a message is addressed explaining to the user how setting its RD.

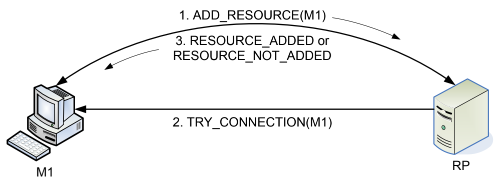

Fig. 5. **Registration of a RD on the RP**

Fig. 5 shows the registration stage of a RD. The answer numbered 3 is made in the same connection as the request numbered 1. It’s why it is possible for the RP to tell $M1$ the result of the connection test (request 2) from the outside.

VIII. CONFIDENTIALITY

A. *Application confidentiality*

The application that a peer receives to perform is not received in the form of file. It is direcly load from network to memory using our HipopJarLoader class. Then it is not impossible but very more difficult for somebody to get the .jar or .class file from memory and to uninterpret it using a JAVA decompilator. In add, a .jar or a .class application is just a part of a DAG. Without the others tasks, a task of a DAG is needless.

B. *Data confidentiality*

Contrary to the applications, data are downloaded and written on the hard drive. It is then easy to steal data files. But firstly, these data files are only a few part of the data of the DAG, secondly, even if a user read these files, he has to understand them and thirdly, users do not chose the task they will perform. Then this kind of attack is a short sniffing one.

IX. FUTURE WORKS

We are currently adding a host certificate to the Resource Provider in order that peers can check its authenticity and don’t register on another machine that the real Resource Provider and be exploited to do something else.

X. CONCLUSION

HiPoP is a securized P2P platform of computing specialized in the execution of applications as DAG. Its goal is not to be the most performant system but to offer the possibility to users to share their resources in a secure way. Thus, users can carry out calculations they can’t do before. The use of HiPoP is very simple and they can use it without having administrative privileges, without having to create a specfic user with specific rights, without having to install a database,... In a lot of cases, the simplicity and the performances of HiPoP suit users.

REFERENCES

Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N. & Shenker, S. (2003), Making gnutella-like p2p systems scalable., *in* ‘SIGCOMM’, pp. 407–418.

Chen, H. & Maheswaran, M. (2002), ‘Distributed dynamic scheduling of composite tasks’.

Crowcroft, J. & Pratt, I. (2002), ‘Peer to peer: Peering into the future’, *Lecture Notes in Computer Science* **2497**, 1–19.

Gerasoulis, A. & Yang, T. (1993), ‘On the granularity and clustering of directed acyclic task graphs’, *IEEE Transactions on Parallel and Distributed Systems* **4**(6), 686–701.

Gong, L., Mueller, M., Prafullchandra, H. & Schemers, R. (1997), ‘Going beyond the sandbox: An overview of the new security architecture in the java development kit 1.2’, *USENIX Symposium on Internet Technologies and Systems* pp. 103–112.

Network, S. D. (1997), ‘Secure computing with java: Now and the future’.

Oaks, S. (2001), *JAVA Security. Second Edition*, O’ Reilly.

Shirky, C. (2000), ‘What is p2p... and what isn’t?’.

Suter, F., Casanova, H., Desprez, F. & Boudet, V. (2003), From heterogeneous task scheduling to heterogeneous mixed data and task parallel scheduling, Technical Report RR2003-52, ENS Lyon, (LIP).