Toward Third Generation Internet Desktop Grids

Ala Rezmerita, Vincent Neri, Franck Cappello

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

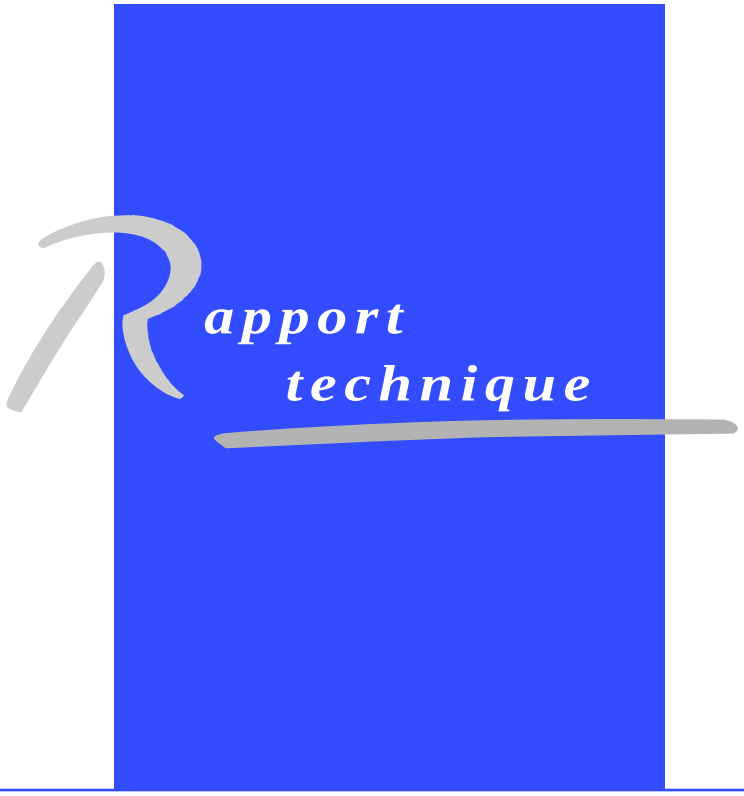
Toward Third Generation Internet Desktop Grids

Ala Rezmerita — Vincent Neri — Franck Cappello

N° 7002

Mai 2007

Thème NUM



ISSN 0249-0803 ISRN INRIA/RT--7002--FR+ENG

Toward Third Generation Internet Desktop Grids

Ala Rezmerita , Vincent Neri , Franck Cappello

Thème NUM — Systèmes numériques
Projets GRAND - LARGE

Rapport technique n° 7002 — Mai 2007 — 25 pages

Résumé : Les projets comme SETI@home et Folding@home ont popularisé le calcul de grille de PCs sur Internet (IDG). La première génération de ces projets a eu des millions de participants mais a été dédiée à une application spécifique. BOINC, United Device et XtremWeb appartiennent à une deuxième génération des plateformes IDG. Leur architecture a été conçue pour s'adapter à beaucoup d'applications, mais elle possède des inconvénients comme une sécurité limitée et une architecture centralisée.

Dans cet article, nous présentons une nouvelle conception pour les grilles de PCs sur Internet, suivant une approche en couches. La nouvelle architecture établit un réseau de recouvrement, donnant aux noeuds participants les possibilités de communication directes. Partant de cette base les principaux mécanismes d'IDG peuvent être mis en application en utilisant les outils de grappe existants et des logiciels spécifiques supplémentaires. Comme preuve de ce concept, nous exécutons une application bioinformatique (Blast) sur la troisième génération IDG, basé sur un service de connectivité (PVC), un ordonnanceur de tâches existant (Condor), un service de transfert de données (Bittorent) et un mécanisme de certification de résultat.

Mots-clés : grilles de PCs, troisième génération, intergiciel de grille, réseau virtuel, partage de ressources, pare-feu, connectivité, DSL

Unité de recherche INRIA Futurs
Parc Club Orsay Université, ZAC des Vignes,
4, rue Jacques Monod, 91893 ORSAY Cedex (France)
Téléphone : +33 1 72 92 59 00 —Télécopie : +33 1 60 19 66 08

Toward Third Generation Internet Desktop Grids

Abstract: Projects like SETI@home and Folding@home have popularized Internet Desktop Grid (IDG) computing. The first generation of IDG projects scalled to millions of particiapatings but was dedicated to a specific application. BOINC, United Device and XtremWeb belong to a second generation of IDG platforms. Their architecture was designed to accommodate many applications but has drawbacks like limited security and a centralized architecture.

In this paper we present a new design for Internet Desktop Grid, following a layered approach. The new architecture establishes an overlay network, giving the participating nodes direct communication capabilities. From that basis many key mechanisms of IDG can be implemented using existing cluster tools and extra IDG specificic software. As a proof of concept, we run a bioinformatic application on a third generation IDG, based on a connectivity service (PVC), an existing job scheduler (Condor), a high performance data transport service (Bittorent) and a custom result certification mechanism.

Key-words: third generation, desktop grid, grid middleware, virtual network, Instant Grid, resource sharing, firewall, connectivity, DSL

1 Introduction

Internet Desktop Grid platforms have demonstrated exceptional performance. Scaling to millions of participants, systems like SETI@home exhibit an aggregated performance of Hundreds of Teraflops. Due to the poor communication performance and the high instability of the nodes, only applications featuring trivial parallelism and executing huge sets of independent tasks can be executed efficiently on such systems. However despite this limitation, many scientists in laboratory and engineers in industry are interested by Internet Desktop Grid infrastructure. Moreover, a lot of applications in science and engineering actually exhibit trivial parallelism and can be executed on these systems (Monte-Carlo computation, virtual screening, sequence comparison).

The first generation of IDG systems was designated around a specific application. This is the case for SETI@home, Distributed.net, Folding@home and others. Because of their specific design, these platforms were unable to run other applications. For example, the protocol used to communicate the parameters and results were not necessarily identical in the objective of optimizing the transport performance according to their respective size. Design decisions linked to the specificities of an application may obviously not comply with the specificities of other applications. Thanks to the popularity of Internet Desktop Grids, it becomes essential for Desktop Grid designers and developers to imagine a novel architectures being able to execute many different applications. The second generation of Desktop Grids was represented by general-purpose platforms like BOINC, XtremWeb, Entropia, and United Devices. On these platforms, a single client submits the tasks of several applications to a server sending them to workers. While this design still holds, it has several strong limitations making it subject to improvements.

The first limitation is their monolithic design. For designers and developers, it becomes more and more complicated to add improvements or simply new features requested by users. In addition, such design shows the Desktop Grid systems to the users as an unmodifiable black box. For example, users are not able to associate data management systems to the Internet Desktop Grid, nor they can use their preferred batch scheduler to manage the jobs. A second drawback is the system asymmetry. Only one machine manages the client requests. May this machine fail and the whole Desktop Grid System is down or works in a highly degraded mode. A third severe limitation for some of these platforms (BOINC for example) is the limited security. Users are neither authenticated nor participating machines.

From these three major limitations, we have designed a novel Internet Desktop Grid architecture that could be called "a third generation" Internet Desktop Grid architecture. This architecture features the essential mechanisms of the previous Desktop Grid systems but these mechanisms are all customizable by the users. Thanks to its modular design, users may run their favorite job and data management systems. They can also run tools originally developed for Clusters.

This paper presents the design and the development of this third generation of Internet Desktop Grid system. We recall the essential features of IDG and present a layered architecture. We give details on the connectivity layer and describe our prototype. As an example of the design modularity, the last part of the paper presents performance evaluation for a

bioinformatics application of a third generation IDG, based on a connectivity service (PVC), a third party job schedulers (Condor), a high performance data transport service (Bittorent) and a custom result certification mechanism.

2 Essentials of Desktop Grids Architecture

Internet Desktop Grid systems share several essential mechanisms. The current monolithic implementations tend to hide or mix these mechanisms together. These mechanisms should be considered separately if we want to propose a novel architecture. This section deconstructs" popular Internet Desktop Grids (BOINC [3], XtremWeb [4]) , exhibiting there essential features. Note that all mechanisms described in this section have a dedicated implementation in most of IDG implementations, which is not the case in clusters and Grids where users and administrators prefer to use standard and well-known software.

Batch scheduler Among the mechanisms shared with clusters and Grid, every Internet Desktop Grids uses a batch scheduler. The batch scheduler receives all users job requests and manages their execution on the IDG resources. In addition to accepting jobs from the users, IDG batch schedulers generally allocate jobs according to the resource features : OS, CPU Instruction set, available libraries, etc. The job manager can be associated with some sophisticated dataflow and workflow engines like DAGMan for Condor or YML in XtremWeb[5].

Data transport Another mechanism required in IDGs and Grids is the data transport service. Most of IDG implementations transport the job parameters and results along with the job requests. Only recently, BOINC and XtremWeb have proposed to use data repositories and transfer only data references along with the job requests. The most used data transport protocol in IDG is FTP. Other data transport protocols have been tested successfully. For example XtremWeb [2] uses Bittorrent to transport and replicate large parameter data sets.

Data storage management Some IDGs and Grids projects have also tested the interest of using caches for parameter and result data sets [1], with the objective of reducing the communication traffic induced by the transfer of data sets when these data sets will be reused by the same resource. Behind these research efforts is the more general notion of data storage management. Ideally, IDG would rely on a file system to handle, store and access data. Such a file system would use all concepts associated with sophisticated file systems : replication, file catalogues, transparent distributed storage and resource discovery, caches, etc. However, none of the existing IDG implementation uses this kind of file system, essentially because of the lack of direct communication between resources and the complexity associated with the implementation of such file systems.

Most of the other mechanisms are specific to IDG and reflect some of their essential distinguishing characteristics : resource volatility, loosely managed resources, untrusted results,

volunteer based participation, limited performance and security degradation for resource owner.

Resource protection Currently, very few IDG systems provide a mechanism to protect the resources from attacks. The resource owners accept this situation because they trust the users and administrators of the IDG platforms. However, this situation is obviously not acceptable in general and the IDG administrators should provide the resource owners a mechanism to protect their resource. In the past two systems were proposed : virtual machines executing bytecodes like JAVA and .NET and sandbox for binary codes. Currently the most promising approach is the virtual machine technologies (XEN, VMWARE, KVM, etc.). These technologies allow confining the execution of the IDG application into a virtual machine which has, in principle, only very few connections with the rest of the system (the other virtual machines).

Connectivity A major problem to solve in every IDG is the connectivity of the computing resources to the rest of the system. Most of IDG resources are located in different administration domains and protected by NATs and Firewalls. Inbound direct communications with these resources is generally not possible. Moreover direct communications between resources themselves are not possible. Thus every IDG system uses a mechanism to deal with NATs and Firewall to establish the connectivity between the resources and the rest of the system. Most of existing IDG use reverse protocols where only outbound connections are used by the resources. But this is not enough to solve every situation, and in some cases, relays have to be used to establish the connectivity.

Volatility tolerance A key mechanism in IDG is the volatility tolerance. Every IDG platform uses resources that are not fully manageable by the IDG administrator. A very common situation is the resource owner disconnecting its resource from the IDG without prior notice. Volatility tolerance systems are also used to cope with the unavoidable failures appearing in IDG : resource hardware of software failures, networking Failures. Like cycle stealing, volatility tolerance is associated with the batch scheduler. When the IDG systems suspect the disconnection or the failure of a resource, they reschedule the job allocated on the suspected resource on another available resource (or their put the job into the waiting queue).

Cycle Stealing One of the main mechanisms associated with IDG, from the beginning of their history, is the cycle stealing system. Cycle stealing essentially limits dynamically the resource utilization to a level given by the resource owner, when the owner uses its resource. While this mechanism was mandatory for the deployment and acceptation of IDG, it is not clear any more that cycle stealing is still highly desired in the context of Mutlicore processors. For example, resource owner may prefer to allocate statically a given number of cores to IDG, while keeping the rest for its own usage.

Result certification Since unknown participants host the resources, the user has a limited trust on the results returned by the resources. To manage this situation, IDG systems use replication and spot-checking to distinguish between correct and incorrect results and to highlight resources returning incorrect results.

Incentive for participation Since the resource owner is not the main user and since, in many cases, he does not have an access to the results of the computation, IDG users have to develop incentive mechanisms to attack resource owners. Most of existing IDG projects offer a system of credits, allowing the ranking of participants, based on their contribution in term of CPU time.

One of the main drawbacks of the monolithic approach in the design of IDG is the limited flexibility and adaptability. There are many ways of implementing every specific mechanism of IDG and many algorithms to deal with result certification, participation incentive, scheduling, volatility tolerance, etc. Users and administrators of different IDGs may have different needs, requirements and preferences. A monolithic design cannot match all existing requirements, needs and preferences.

3 A third generation of Desktop Grid architecture

Our proposal for a third generation IDG intents to cope with the flexibility and adaptability limitations of the first and second generations IDG. In addition, the proposed architecture aims at reusing existing cluster and Grid software compliant with the low communication performance of IDG. For these mechanisms, it is questionable to design and implement novel software. If we take the example of a batch scheduler, there are several existing software that have demonstrated to be extremely reliable and have the experience of years of utilization (Condor, Sun Grid Engine, Torque, LSF). The development effort of a dedicated stable scheduler with a sophisticated interface similar to the existing ones would be considerable. Moreover, most of the extra features implemented in a novel would probably be implementable in the existing ones. In addition to mechanisms shared with Cluster and Grid, there are some specific mechanisms that clearly distinguish Desktop Grids. The objective of our proposal for a third generation is to concentrate the development and research effort on these specific mechanisms.

We propose to follow a layered architecture, as shown in Figure 1. Most of the components of the layers are already available as software in the context of cluster or Grid computing. The missing parts concern some of the specificities of IDG : the connectivity layer, the result certification and participation incentive mechanisms. In this paper we focus on the connectivity layer.

The connectivity layer, described in the next section is the core mechanism enabling the deployment of an IDG. It uses sophisticated component to establish an overlay network at IP level requiring minimum configuration for its installation. The connectivity layer also provides strong security by requiring the authentication of resources before their connection to the overlay network.

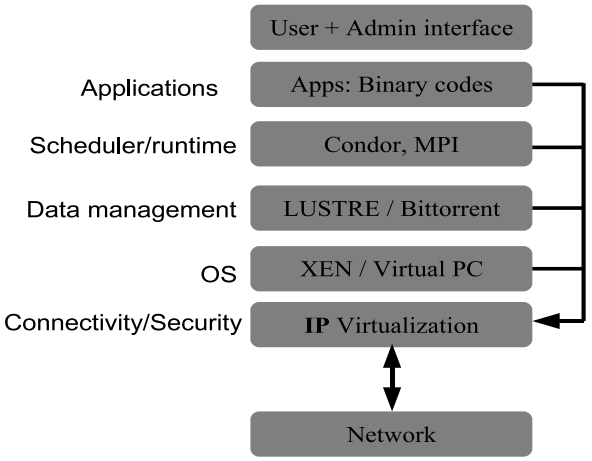


FIG. 1 – A layered software architecture for Internet Desktop Grids

One of the key benefits of the proposed approach is to reuse existing software to implement IDG mechanisms. All existing software using IP as the communication protocol can run without modification on top of the connectivity layer, providing that their performance are compliant with the ones of IDG resources. Well known batch schedulers like Condor, SUN Grid Engine, Torque and others can also be used without modification on top of the connectivity layer. Also, data transport and management software like Bittorrent and LUSTRE (distributed file system) can be executed on top of the connectivity layer. In the evaluation section we run Condor in conjunction with Bittorrent on top of the connectivity layer to highlight the flexibility of the proposed architecture.

From this basis, all IDG specific mechanisms presented in the previous section can be implemented from existing software or specific ones. In term of resource protection, virtual machine technologies such as XEN, KVM and VMWARE can run in conjunction with the connectivity layer. This allows establishing a virtual cluster from virtual machines and a virtual network, fully decoupled of the other resources utilization.

Volatility tolerance is a standard feature of popular batch schedulers. They use failure detectors in conjunction with checkpoint and migration to restart failed jobs (jobs executed on failed resources or on unreachable resource). Cycle stealing is also a standard feature in many batch schedulers (Condor, Torque, LSF). Thus, if this feature is required in an IDG deployment, the administrator can configure one of these batch schedulers to run jobs under the cycle stealing policy. Result certification is fundamentally based on mechanisms like resource selection, job replication, voting and result checking. Approaches like spot-checking and voting from results of replicated jobs can be implemented over popular batch schedulers. In the evaluation section, we evaluate the performance of these techniques implemented as script controlling Condor job submissions. Incentive for participation can also be implemen-

ted on top of a batch scheduler, rewarding the resources with credits when jobs are executed successfully and correctly (result certification).

Even thought, there is still a large space for research in result certification, incentive for participation and scheduling in IDG, we believe that most of the research and development effort could be concentrated on improving these techniques or existing software, instead of designing and implementing novel job schedulers, data transport and management systems and resource protection mechanisms.

Several other projects like VNET, Viocluster and ViNe [7] share some common goals with our proposal. They do not consider Desktop Grids as their targets. VNET is a proxy scheme working at user level and using the Layer Two Tunneling Protocol (L2TP). Virtuoso [9], a virtual private network tool, extends VNET by implementing a virtual local area network over a wide area using tunneling, for virtual machines in Grids. The In-VIGO [11] grid computing system uses virtualization technologies to create dynamic pools of virtual resources. VioCluster [12] logically gather machines between virtual domains, allowing a cluster to dynamically grow and shrink based on resource demand. Network virtualization in VioCluster is made by a hybrid version of VIOLIN [13] which gives to a machine the ability to connect to the private network. Cluster-On-Demand(COD) [14] was inspired by Oceano [15] . Its main difference is its dynamic resource management between multiple clusters by reinstalling the base OS on resources. The VNET [16] is a virtual private network tool implementing a virtual local area network over a wide area, for virtual machines in Grids.

WOW [6] is the closest project to our proposal. It clearly aims at providing a virtual cluster over a set of resources belonging to different administration domains in the context of IDG. WOW mainly focuses on self-organization and P2P routing while in our case, we mainly address the issues related to the connectivity layer including virtual networking, direct connection establishment and security. We also show in this paper how essential mechanisms of IDG can be rebuilt from the connectivity layer and existing software.

4 A First Prototype

A key part of the third generation of Desktop Grids is the connectivity layer. In this section we present the latest organization of Private Virtual Cluster (PVC), a generic framework dynamically turning a set of resources belonging to different administration domains into a virtual cluster. A previous paper about PVC [17] presented in details the organization of the modular framework and its connectivity component. In this section, we recall the main features of PVC and describe in detail the new version of the interposition component.

PVC connects firewall-protected nodes without any intervention of domain administrators and without breaking the security rules of the hosting domains. PVC is a distributed system working as a daemon process (peer) running on every participating host connected to a brokering service. The role of every local peer daemon is to establish a secure direct connection between the local peer and the other participating peers, subsequently leaving the connection control to the application. The role of the brokering service is to help establishing these connections by collecting and advertising the peer connection requests and

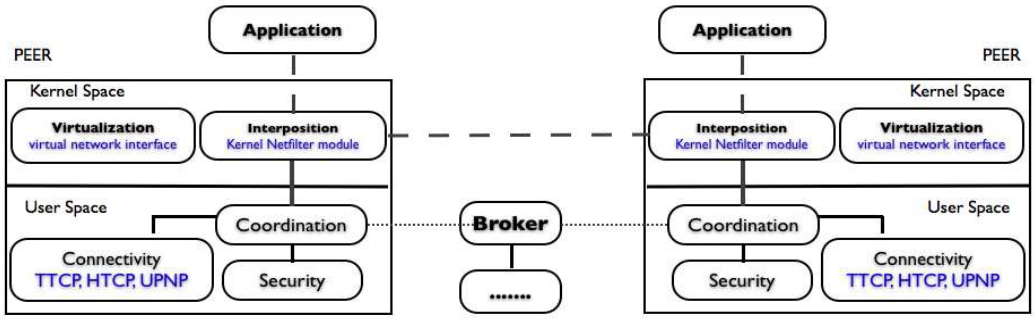


FIG. 2 – PVC architecture

tunneling some communications between peers when direct connections are not established, translating network addresses from virtual to real and transporting security negotiation messages. The brokering service is implemented as a set of replicated nodes, connected to the Internet and accepts inbound communications from PVC peers.

Figure 2 presents the modular architecture of PVC. The peer daemon encapsulates five modules for : 1) operation coordination, 2) communication interposition, 3) network virtualization for the application, 4) security checking and 5) peer-to-peer direct connection establishment. The modular architecture offers the possibility to extend and adapt every module to fit with the target environment.

All daemon modules are coordinated locally by the coordinator, which also participates in the global coordination of aPVC deployment. The coordinator runs a workflow through the four other modules to establish the direct connection between the local peer and distant ones. The coordinator also exchanges messages with the brokering service to implement the global coordination.

The interposition module intercepts the application connection requests and transfers them to the coordination module. It may be implemented in various ways (network calls overloading, virtual network interface) offering high adaptability to the system configuration. The intercepted requests are routed on a virtual network simulated by the PVC virtualization module, which features its own IP range and domain name service. In this virtual network, the PVC security module checks the respect of pre-existing security policies and authenticates the virtual cluster participants. Different security standards and specific methods may be adapted to the PVC architecture (SSL certificates, standard security challenges, etc.). The connectivity module transparently helps the cluster application to establish direct connections between virtual cluster nodes (peers). Like all the other modules, a variety of techniques can be used depending on participant's host configuration as well as its local network environment (firewall, NAT). Standard (UPnP) and original mechanisms

(Traversing-TCP, TCP Hole Punching) may be used to establish direct connection between peers.

4.1 Interposition Module

The goal of this module is to intercept applications connection requests. It plays a very important role as the performances of the whole system highly depend on the used method. We have investigated the solutions that do not require the cluster applications modification or recompilation and that are completely transparent for the applications using PVC.

In our first prototype we have implemented 3 interposition methods 1) library interposition, 2) the use of universal TUN/TAP driver and 3) a kernel Netfilter module. Depending on the system configuration regarding the operating system, user rights and installed modules, the PVC user is able to specify the method that fits better within his system. PVC assures the connectivity between heterogeneous cluster participants regardless of the different chosen interposition techniques.

4.1.1 Library Interposition

Basically used for tuning performance, collecting runtime statistics, or debugging application, the library interposition [18] consists in creating a dynamic library that is preloaded during the execution of application. The environment variables LD LIBRARY PATH and LD PRELOAD are used to load the new library before any other shared library.

In order to use this method we overloaded standard C library network system calls like : bind, connect, listen, accept, etc. The library is preloaded during the execution of application. This way, the application uses custom network (connect, listen, bind, etc) functions, which are implementing the different phases of the connection establishment protocol and this without any code source modification or recompiling.

The main advantage of this method is that it does not require super-user privileges. However this technique works only with dynamically linked applications and does not work with suid binaries. The use of this technique in PVC introduces some overhead on the connection initiation but once the connection established the application recovers full control on it with no further implication from PVC. The performance analyses of this method were presented in [17].

4.1.2 Universal TUN/TAP Driver

Universal TUN/TAP driver [19] is a largely used solution for the implementations of VPN [10, 22, 23] and virtual machine networking [25]. TAP simulates an Ethernet device and is usually used to create a Network Bridge. TUN simulates a network layer device and is used with Routing. The IP packets sent by the operating system via a TUN/TAP device are delivered to a user-space program that attaches itself to the device. A user-space application may also pass packets to a TUN/TAP device. In this case, TUN/TAP device delivers (or "injects") these packets into the system network stack, thus emulating their

reception from an external source. When PVC is started, a virtual network device is created and all communication between the applications is routed via this virtual device. When the PVC peer intercepts a connection request, a secure direct connection is established with the distant peer. All network traffic between the two peers is tunneled using this connection.

This method requires special rigths only for the network device configuration. As all the concerned IP packets are transferred from kernel space to user space the performances of this method are not very good. However, the fact that all the traffic is tunneled within one connection between PVC peers makes this method suitable when no traversing technique can be applied and when there is an open incoming port on one of the two peers trying to get connected. The performance evaluation of PVC using this method is presented in the Performance Evaluation section.

4.1.3 Netfilter module

Netfilter [27] is a framework inside the kernel that allows a module to observe and modify packets as they pass through the IP stack. This packet observation and filtering mechanism was introduced during the 2.3 kernel development. In this case, the PVC interposition module is composed from two parts : a new kernel module and a peer daemon in user space. The communication between the two parts is performed using IOCTLs via /dev. The kernel module intercepts the application connection request and informs the PVC on the connection characteristics (IP, port).

PVC performs the connection establishment protocol and gives to the kernel module some masquerading rules. Using this rules the kernel module will perform the changes in each outgoing/ingoing packet. Once the end of the connection is detected the masquerading rule is erased. The performance evaluation of this technique will be presented in the Performance Evaluation section.

4.2 Domain Virtualization

PVC allows the execution of applications without any modification. The applications generally use the socket model as interface with the communication network. Following this constraint PVC uses a domain virtualization at IP level. The virtualization layer establishes an IP domain over resources belonging to different administration domains having public or private (possibly conflicting) addresses. Like in a VPN, an overlay network featuring virtual IP addresses is built on top of the actual network.

To avoid the conflict between real and virtual networks used by the resource, we use a specific IP class defined by a RFC [21] for experimental purposes (class E ranging between 240.0.0.1 and 255.255.255.254). The use of these IP addresses guaranties that no real machine uses them (such addresses are actually not routed on the Internet). A virtual DNS, configurable by the PVC members, is associated with this experimental IP class.

4.3 Security Policy in PVC

The main objective of the security mechanism is to fulfill the security policy of every local domain and enforce a crossdomain security policy. Two security levels are implemented : 1) local to the administration domain and 2) between domains. The local system administrator can configure the intra and inter-domain security policies and also define the global policy. When a connection is requested by the application, the local peer first checks that it can accept inbound and outbound communications with other peers outside the administration domain, according to the local policy. Then, it checks that IP addresses of external peers and the ports to be used are granted by the global policy.

Without a strong access control mechanism, someone may take advantage of the brokering service and pretend to take part of the virtual cluster. To avoid this, every virtual cluster has a master peer (a peer managed by the virtual cluster administrator) implementing the global security policy. Only the master peer can dynamically register new hosts. Before opening the connection, every peer checks that the other peer belongs to the same virtual cluster. The cross authentication is performed using master information and crossing the brokering service. A key point in the design is that the security protocol does not need to trust the brokering service. This protocol ensures that : 1) only the participating hosts of a cluster can be connected to each other and 2) only trusted connections are returned to the cluster application.

In the current implementation, each host connected to PVC has its own private and public key. Every participant to a virtual cluster knows the public key of its master before connecting to PVC infrastructure. The master peer registers the participation of a new peer, asking the brokering service to store its public key previously encoded with the master's private key. During the establishment of the connection, both peers obtain the other side's public key from the brokering service and decode the received message with the master's public key. This mechanism ensures that only the master registers other participants on the brokering service.

The peer's mutual authentication consist in a classical security challenge-response : the client generates a cryptographically random string M, encrypts it with server public key and sends it to the server ; the server decrypts the message with its private key, encrypts the obtained value using its private key and returns the result, Es(M), to the client ; the client decrypts Es(M) using the server's public key, obtaining Ds(Es(M)) ; if that value is equal to the original M, the client is satisfied of the server's identity. Similarly, the server picks a random string L, encrypts it and sends it to the client, which returns Ec(L) to the server. The server checks that Dc(Ec(L)) equals L and it thereby satisfied with the client's identity. The security is implemented using OpenSSL Crypto library.

4.4 Inter-domain connectivity techniques

The objective of PVC is to give the participating nodes direct communication capabilities. To achieve this, the connectivity module hosts several connection protocols. In the

current implementation, we have integrated three techniques. In this section, we present the integration of these techniques.

4.4.1 Integration of a Firewall configuration protocol

In the last two years, UPnP project [29] became very popular. The principal vendors of domestic network devices incorporated UPnP in their routers. Using UPnP, a PVC peer can communicate with the router and can open the ports for direct connections.

To guarantee the safety of the local area network, the port forwarding rules are erased from the router when they are no longer needed. We also use a distributed architecture to detect node failure and handle firewall rule deletion by the mean of monitoring daemons that are periodically checking the rules present on the firewall and detect eventual faulty machines.

4.4.2 TCP hole punching

Widely used for various network-based applications, TCP Hole Punching [28] allows connection establishment between two hosts behind NATs in different administration domains. Both clients establish a connection with the broker that observes the public addresses (given by NAT) and private addresses of the clients and shares this information between the peers. After this exchange, the clients try to connect to each other's NAT devices directly on the translated ports. If NAT devices use the previously created translation states then a direct connection is possible. The advantage of using this method is that it does not require special privileges or specific network topology information. However, this technique does not work with all type of NATs as their behavior is not standardized.

4.4.3 Traversing-TCP

Traversing TCP is derived from the TCP protocol and it works with firewalls that are not running stateful packet inspection. It essentially consists in 1) transporting, using the PVC Broker, the initiating TCP packet (SYN) blocked by the firewalls or NAT on the server side and 2) injecting the packet in the server IP stack.

Figure 3 presents in details the TTCP technique. Plain lines show the packets corresponding to the TCP standard. Dashed lines correspond to specific Traversing-TCP messages. A TTCP connection behaves as follows : **Definitions :** Server node : S, Broker : B, Client node : C, initializing packet : SYN

1. The peer on S connects to B and waits for new connection demand ;
2. C sends SYN to S. It opens the Firewall of C but it is stopped by the Firewall of S ;
3. The peer on C sends the SYN packet information to B. B forwards it to the peer on S ;
4. The peer on S injects the SYN packet to the IP stack on S ;

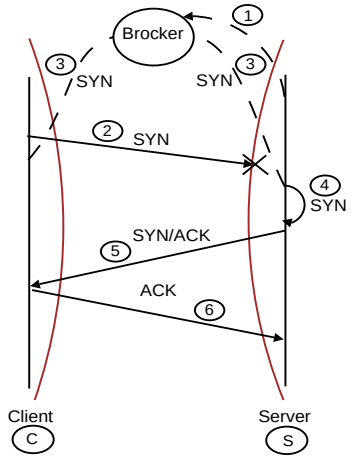


FIG. 3 – Traversing-TCP

5. To this SYN packet S replies with a SYN/ACK packet. The SYN/ACK packet opens Firewall on S and is accepted by the Firewall on C (previously opened) ;
6. The initialization of the TCP connection ends with an ACK packet from C to S : the TCP connection is established.

TTCP works under the following device configurations : 1) The firewall must authorize the outgoing packets and must accept all packets from established connections ; 2) Following the [30] classification, TTCP should work with all NAT's, except symmetric NAT (which maps a port to a quadruplet : the internal host-port and external host-port).

Note that RST packets sent as rejection notification are also captured by a PVC client peer and not forwarded to the client IP stack. Following our experience with the DSL-Lab platform (cf. the evaluation section) and related work [31], these requirements fit many professional and domestic configurations.

After connection establishment, the communication can continue following classical TCP operations. The communication between the two peers is direct, bypassing the broker and ensuring high communication performance.

5 Performance evaluation

To demonstrate our design modularity this section presents the performance evaluation of the new version of connectivity service (PVC). To highlight the flexibility of the proposed architecture we run the batch scheduler Condor in conjunction with Bittorrent on top of the connectivity layer. We also present the performance evaluation of Condor flocking mechanism

that is an interesting point for a third generation IDG and a custom result certification mechanism.

5.1 Experimental Protocol

All the experiments presented in this paper were performed on the experimental Grid'5000 platform. We used homogeneous clusters with AMD Opteron 248 (2.2 GHz/1MB L2 cache) dual-processors running at 2GHz. All the experiments were made on the 216-node cluster at Orsay. The nodes were interconnected by a Gigabit Ethernet switch. One major feature of the Grid'5000 project is the ability of the user to boot his own environment (operating system, distribution, libraries, etc.) on all the computing nodes booked for his experiments. We used this feature to run all of our experiments in an homogeneous environment. All the nodes were booted under Linux 2.6.14.4. To simulate the firewall we used Linux Netfilter Iptables [35]. We have used the 6.8.1 Condor Version and we configured its network communication to TCP only.

5.2 PVC

To demonstrate the minimal overhead of the new version of PVC for TCP communications, we run two types of tests. The first one evaluates the overhead of PVC for establishing a connection. In the second one, we used NetPIPE [32] to compare network performance with/without PVC, using the two new interposition methods.

5.2.1 Connection overhead

The connection establishment overhead highly depends on the interposition method. The overhead for the library interposition technique was presented in [17]. Using this method, the mean value for the connection establishement overhead is about 60ms. When using Universal TUN/TAP device, the connection overhead exists only for the first connection between two peers (all the communications between the applications running on these hosts are routed on a single network connection). This tunnel uses a timeout and it closes after a period of non-communication, being re-opened for the next connection requests. The mean value for connection overhead using this method is about 80ms. In the case of the Netfilter interposition method, the overhead is larger due to the kernel/user space context switch with a mean value of approximately 0.5s by connection request.

The overhead encompasses the costs of connection attempt interception and the communication with the broker. As it intervenes at most one time for every end-to-end communication, it still remains reasonable in the context of distributed applications.

5.2.2 Bandwidth and Latency overhead

Figure 4 presents the bandwidth (in Mbps) of PVC (with Universal TUN/TAP device and a kernel module) and the reference (without PVC), using NetPIPE on nodes interconnected

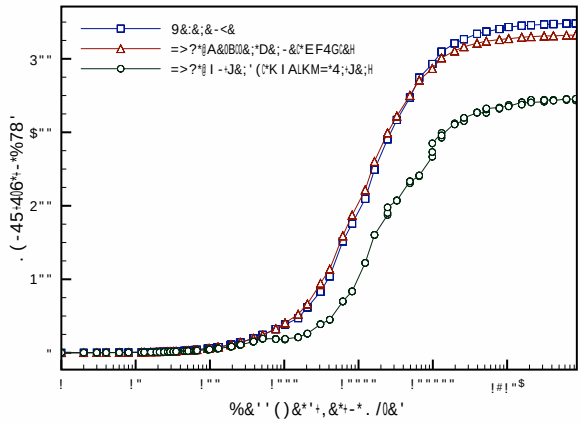


FIG. 4 – Bandwidth measured by NetPIPE without PVC (Reference) and with PVC on Gigabit Ethernet network.

by a Gigabit Ethernet switch. Figure demonstrates that for PVC with kernel module and the reference, until 64kB message size, the network rates computed by NetPIPE are statistically similar. From this size, the stream of IP packets modifyed by PVC becomes larger, making PVC overhead to become more significant (the maximum overhead value representing 3.5% of the total communication bandwidth). The figure also shows that PVC, using Universal TUN/TAP module has a very large overhead (30%). This is due to kernel/userspace context switching and IP encapsulation.

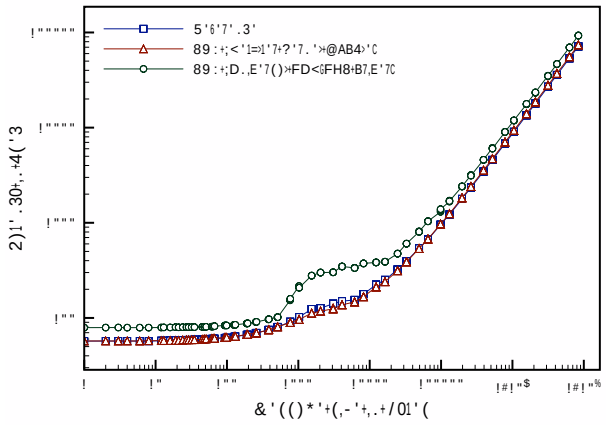


FIG. 5 – Latency measured by NetPIPE without PVC (Reference) and with PVC on Gigabit Ethernet network.

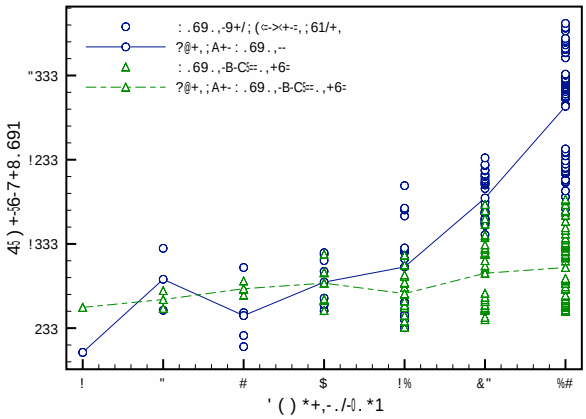


FIG. 6 – Completion Time for the distribution of a 3.2GB file to a varying number of jobs. The vertical axe shows the time in second, the horizontal axe shows the number of submitted jobs.

Figure 5 presents the latency (in usec) of PVC (with firewall) and the reference (without PVC), using NetPIPE on nodes interconnected by a Gigabit Ethernet switch. The obtained results are similar to the bandwidth overhead. Using the Netfilter module, the PVC latency overhead is 3.5% of the total communication latency and for Universal TUN/TAP module, the latency overhead is about 30% of the total communication latency. Thus, the netfilter method is the prefered one compared to TUN/TAP but in some situations, this interposition approach cannot be used, as explained in the previous section.

5.3 Condor

To demonstrate our design modularity we execute, on top of the connectivity layer, the Condor [8] workload management system. This batch scheduler provides a job querying mechanism, scheduling policy, priority scheme, resource monitoring and resource management.

5.3.1 Data Transport

To highlight the flexibility of the proposed architecture, we executed Condor in conjunction with Bittorrent and compared its performance with the Condor default data transfer protocol.

We considered a bioinformatic application where a DNA database has to be transmitted to every participating nodes prior to the job execution. The file corresponds to the BLAST application DNA database (3.2 GB). As a first transmission method, a synthetic job was

created using the Condor default data transport protocol. As a second transmission method, we implemted another synthetic job using the Bittorrent protocol. In order to use Bittorrent, a torrent file is automatically created and a tracker is launched before submitting the job. Afterwards, we specify, in the job submission file, that Condor must copy the torrent file to all the nodes and launch the Bittorrent client on each of them.

The experiment was performed on a Condor pool of 64 machines with 1 to 64 submitted jobs. The Figure 6 presents the results of this experiment. These results demonstrate that the Condor default transfer protocol exhibits transfer times growing proportionally with the number of scheduled jobs (as the data is transferred for every submitted job). In contrary, Bittorrent maintains the transfer times rather constant. Therefore, Bittorrent demonstrates much better performances, especially when the number of submitted jobs is growing (starting with 16 jobs) and this improvement should be even more significant in the context of Desktop Grids using public networks with a much lower bandwidth [39].

5.3.2 Condor flocking

For the new generation of Desktop Grids, an interesting point of Condor system is the mechanism for sharing resources among Condor pools [33, 34], called flocking. Using this technique, a Condor pool is able to accept job requests forwarded from a remote pool. The main drawback of the method consists in its static configuration : to use flocking, the Condor pools must be manually configured.

The purpose of our experiments was to compare the job throughput achieved by flocking among several pools with the one obtained by a single pool containing the same number of machines. To make the measurements, 64 hosts were configured to create various configurations of Condor Pools. These configurations are shown in Figure 7.

In order to observe their effect on the job scheduling we used the BLAST (Basic Local Alignment Search Tool) application, originated from the National Center for Biotechnology Information (NCBI). This application compares a query sequence with a database of sequences, and identifies library sequences that resemble the query sequence above a certain threshold. In our experiments, we used the blastp program that compares an amino acid query sequence against a protein sequence database. The ten input DNA sequences used were taken from the GenBank database and the DNA databases were taken from The National Center for Biotechnology Information.

For the purpose of our experiments, we have created 64 sequences of 10 BLAST job submissions, each one with a duration between 40 and 160 seconds, with an average of 70 seconds. For the cases of one Condor pool (Config. C) and 64 Condor pools (Config. B) the 64 job sequences were merged into a single queue and were issued simultaneously. For the case of 8 Condors pools, each one containing 8 machines, the sequences were equally spitted between the pools. When the flocking was enabled, they were issued simultaneously on each pool.

Table 1 shows the job waiting times, in the queue for the configuration A presented in Figure7. In this configuration, we did not use the flocking mechanism. The same number of jobs sequences (8) was submitted in each pool (a sequence represents 10 BLAST jobs).

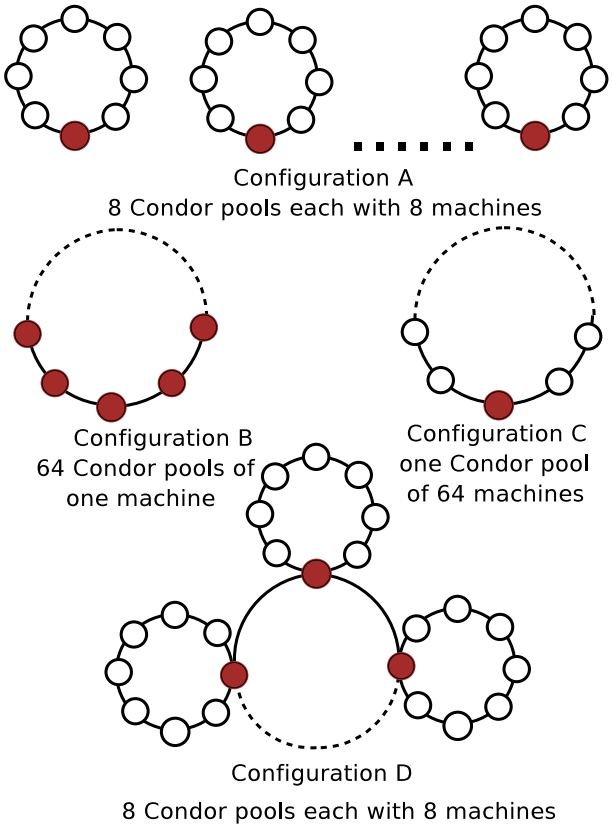


FIG. 7 – The configurations of Condor pools used for making performance measurements. The red nodes are the central managers and the white nodes are the compute-machines. The ring between the central managers represents the flocking mechanism.

TAB. 1 – Wait times for jobs in queue for the configuration A (without flocking). All jobs were equally spitted between the pools

Pool	mean	min	max	stdev
I	0 :02 :41	0 :00 :02	0 :05 :28	0 :01 :36
II	0 :02 :43	0 :00 :02	0 :05 :27	0 :01 :35
III	0 :02 :41	0 :00 :03	0 :05 :40	0 :01 :37
IV	0 :02 :50	0 :01 :00	0 :05 :12	0 :01 :14
V	0 :02 :48	0 :00 :02	0 :05 :42	0 :01 :42
VI	0 :02 :33	0 :00 :42	0 :04 :53	0 :01 :17
VII	0 :02 :47	0 :00 :41	0 :05 :10	0 :01 :19
VIII	0 :02 :47	0 :01 :00	0 :04 :46	0 :01 :07
Average	0 :02 :44	0 :00 :26	0 :05 :17	0 :01 :26

TAB. 2 – Wait times for jobs in queue for the configuration D (with flocking). All jobs were equally spitted between the pools

Pool	mean	min	max	stdev
I	0 :02 :49	0 :00 :02	0 :05 :49	0 :01 :43
II	0 :01 :34	0 :00 :02	0 :03 :23	0 :00 :55
III	0 :33 :36	0 :00 :17	0 :48 :27	0 :15 :05
IV	0 :02 :08	0 :00 :02	0 :04 :21	0 :01 :14
V	0 :01 :44	0 :00 :02	0 :03 :24	0 :00 :57
VI	0 :02 :32	0 :01 :21	0 :04 :00	0 :00 :37
VII	0 :01 :43	0 :01 :08	0 :02 :17	0 :00 :16
VIII	0 :01 :22	0 :00 :37	0 :02 :20	0 :00 :30
Average	0 :05 :56	0 :00 :26	0 :09 :15	0 :02 :40

Subsequently, we determine how the jobs waiting times are changing when we enable the flocking mechanism between the centrals managers. Table 2 shows the job waiting times for the Configuration D. As for the Configuration A, we have submitted the same number of jobs in each pool. The job submission was issued, in order, from pool I to pool VIII. We have observed that, in the pool III, several compute-machines were rejecting the submitted jobs. When the pool II becomed idle, the central manager of pool III flocked 35% of the jobs in pool II.

The average job waiting time in the Condor wait queue, before being scheduled, in the case of configuration A, was only 2 :44 minutes. The average waiting time for the Configuration D, with flocking, was 5 :56 minutes. However, if the flocking mechanism were not enabled in configuration D, the average job waiting time would have been larger as the flocked jobs would have had to wait to be executed in pool III. We may conclude that the flocking helps the overloaded pools to distribute some of their jobs to idle pools, improving the job throughput.

TAB. 3 – Wait times for jobs in queue for the configuration B. The jobs are submited from 8 machines.

	mean	min	max	stdev	jobs executed remotely
I	0 :13 :31	0 :00 :11	0 :20 :11	0 :05 :00	76%
II	0 :27 :51	0 :00 :03	0 :41 :31	0 :11 :47	50%
III	0 :33 :29	0 :00 :02	0 :48 :16	0 :15 :06	40%
IV	0 :36 :59	0 :00 :02	1 :00 :46	0 :19 :26	26%
V	0 :27 :35	0 :00 :03	0 :40 :37	0 :11 :47	49%
VI	0 :45 :58	0 :00 :02	1 :20 :14	0 :26 :07	15%
VII	0 :41 :09	0 :00 :02	1 :16 :09	0 :23 :47	9%
VIII	0 :42 :22	0 :00 :03	1 :14 :33	0 :24 :07	28%
Average	0 :33 :37	0 :00 :03	0 :55 :17	0 :17 :08	37%

TAB. 4 – Wait times for jobs in queue. All numbers are in format hh :mm :ss.

	mean	min	max	stdev	jobs executed remotely
Pool of 64 machines (Conf. C)	0 :07 :50	0 :00 :03	0 :15 :22	0 :04 :08	0%
8 pools of 8 machines (Conf. D)	0 :16 :33	0 :00 :02	0 :28 :25	0 :07 :48	61%
64 pools of 1 machine (Conf. B)	0 :27 :45	0 :00 :02	0 :41 :25	0 :10 :17	94%

Nevertheless, the improvement is not as significant as it could be for long duration jobs. In the paper [20] the authors demonstrate that introducing the flocking between distributed pools brings a significant improvement in job throughput, especially for long execution time jobs.

We have further tried to determine how the waiting time changes when every machine is configured to act as its own central-manager and to be able to flock jobs to all other 63 machines (Configuration B). The same amount of jobs was submitted from 8 machines. Table 3 shows the job waiting times and the remotely executed jobs percentage. The average amount of time the jobs have to wait in the Condor wait queue was 33 :37 minutes. The average percentage of the jobs that were executed remotely was 37%.

In order to show the flocking mechanism efficiency, we have merged the 64 job sequences and submitted them from a single machine, in the three different configurations of Condor pools. Table 4 shows the results for configurations C, D and B (waiting time and remote

execution). Submitting jobs from a single machine, in the Configurations B, leads to an improvement of the average waiting time, compared to the previous experiment setting where jobs were submitted from several machines. This improvement is due to the fact that only one manager takes the decisions whereas in the previous tests the various Condor managers had to negotiate the available resources.

We may also observe that the most efficient solution for scheduling jobs is a large pool with a central manager. However, in the case of Desktop Grids, merging the machines across different administrative domains is not desirable because of the fault tolerance concern.

In summary, these experiments show that, with PVC, the user is able to organize his cluster in various configurations, permitting him in the Desktop Grids context, to choose the optimal one, in terms of performance and fault tolerance.

5.3.3 Result certification

The important aspect of this new generation of desktop grids is the result certification : the computing process needs to be protected from erroneous results returned by possible malicious or faulting participants. Most cluster batch schedulers, including Condor, do not provide such mechanisms because the used resources and the network connecting them are placed under the control of the owner of the computation.

Two of traditional techniques used in for the results certification are the majority voting and spot-checking [37]. The majority voting technique validates a submitted job by requiring that a majority of results for the job, from different resources, have the same values. We have been able to experiment the majority voting in Condor by implementing a simple 20 lines bash script used to replicate the submitted jobs and to collect and compare their results. When a job is submited, it is replicated by a factor of m + 1 (where m is the maximum number of saboteurs). Once the results have been recovered they are compared and if differences are found than the jobs are re-submitted with a replication factor of 2m + 1, excluding the resources used in the first phase. This time, the obtained responses will contain a majority of results with the same value, therefore permitting to choose the correct one. The spot-checking consists in randomly validating the trustworthiness of participants by submitting them a job with a known pre-computed result. The resources that returned erroneous results are blacklisted as un-trustable and banned from further job submissions. The spot-checking was also adapted to Condor using a script that submits a checking job for every 10 jobs submitted by the user. For every invalid checking job result, the concerned resource is added to the blacklist and no longer used for the job submissions.

We tested the script with Condor in an environment of 70 machines where 10% of them were randomly choused to act as saboteurs : they always returned an erroneous result. The goal of the test was to determinate the number of executed jobs till the detection of all saboteurs. The results of this test are presented in Figure 8. As result, for this case study, the necessary amount of jobs to detect all the saboteurs was 201. In the light of the the facility of applying the described techniques to a cluster application lead us to the conclusion that the result certification can be efficiently integrated. The implementation is generic and the

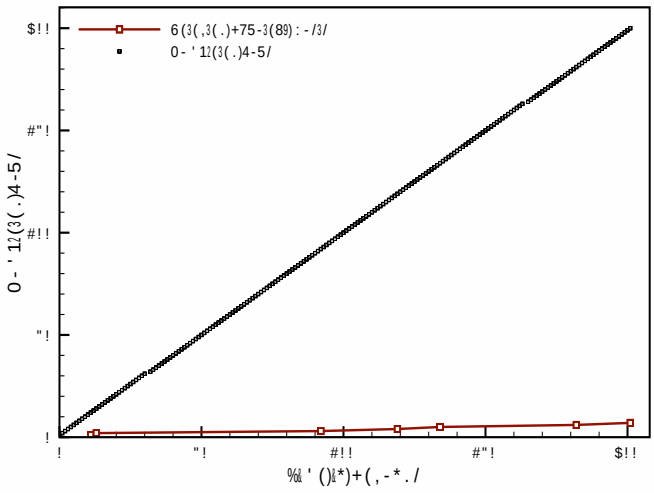


FIG. 8 – Detection of malicious hosts with spot-checking

only part that is job specific is the results comparison, thus the reliability can be easily ameliorated by using more complex techniques.

6 Conclusion

This paper has presented the concept of a third generation Internet Desktop Grid, based on a layered, modular and open architecture. The main principles of this architecture are : 1) estalish transparently and securely a virtual network layer over resources belonging to different adminstration domain, 2) use existing cluster and Grid tools to implement complex mechanisms like batch scheduler and data transport and management systems, 3) developp extra software only for the specificities of Internet Desktop Grids.

Following these principles, we have presented in details the connectivity layer, establishing the virtual network. The connectivity layer is implemented using PVC, a modular framework where several interposition and cross domain connection techniques can be used and added. To demonstrate the feasability of the approach, we have run several experiments for a bioinformatic applications, over a third generation IDG, based on PVC, the Condor batch scheduler, the Bittorrent communication protocol and a specific result certification code.

The evaluation highlights the low overhead of the connectivity layer, the interest of using several Condor job managers by flocking and the performance comparison of several result certification approaches. Overall, we believe that this novel architecture of Internet Desktop

Grid will allow researchers focusing on the improvement of existing tools (batch schedulers, data management systems, etc.) and the design of novel approaches for the features specific to IDG.

7 Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see www.grid5000.fr).

Références

[1] Xiaosong Ma, Vincent W. Freeh, Tao Yang, Sudharshan S. Vazhkudai, Tyler A. Simon and Stephen L. Scott. Coupling prefix caching and collective downloads for remote dataset access. *In Proceedings of the 20th annual international conference on Supercomputing(ICS'06)*, pp 229-238, Queensland, Australia, 2006.

[2] Baohua Wei, G. Fedak and F. Cappello. Scheduling independent tasks sharing large data distributed with BitTorrent. *In Proceedings of the 6th IEEE/ACM International Workshop on Grid (Grid'2005)*, Seattle, USA, November, 2005

[3] David P. Anderson. BOINC : A System for Public-Resource Computing and Storage. *In Proceedings of the 5th IEEE/ACM International Workshop on Grid(Grid'2004)*, Pittsburgh, USA, 2004

[4] Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Herault, Frdric Magniette, Vincent Nri and Oleg Lodygensky. Computing on large-scale distributed systems : XtremWeb architecture, programming models, security, tests and convergence with grid. *In Journal of Future Generation Computer Systems*, Elsevier, Volume 21, Issue 3, Pages 417-437, March 2005

[5] O. Delannoy, N. Emad and S. Petiton. Workflow Global Computing with YML. *In Proceedings of the 7th IEEE/ACM International Conference on Grid Computing (Grid'2006)*, Barcelona, Spain, September, 2006.

[6] Ganguly, Arijit , Abhishek Agrawal, P. Oscar Boykin and Renato Figueiredo. WOW : Self-Organizing Wide Area Overlay Networks of Virtual Workstations. *In Proceedings of the 15th Conference on High Performance Distributed Computing (HPDC'2006)*, IEEE press, Paris, France, 2006

[7] M. Tsugawa and J.A.B Fortes. A virtual network (ViNe) architecture for grid computing. *In Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, IEEE press, Rhodes Islland, Greece, 25-29 April 2006.

[8] Michael Litzkow, Miron Livny, and Matt Mutka. Condor - A hunter of idle workstations. *In Proceedings 8th International Conference on Distributed Computing Systems (ICDCS 1988)*, pages 104111, San Jose, CA, 1988.

[9] SHOYKHE T, A. , LANGE , J . , AND DI NDA, P. Virtuoso : A System For Virtual Machine Marketplaces. Tech. Rep. NWU-CS-04-39, Northwestern University, Jul. 2004.

[10] Maxim Krasnyansky. Virtual tunnels over tcp/ip networks. http ://vtun.sourceforge.net/

[11] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. K rsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. From Virtualized Resources to Virtual Computing Grids : The In-VIGO System. *FGCS special issue on Complex Problem-Solving Environments for Grid Computing* , vol. 21, no. 6, April 2005.

[12] P. Ruth, P. McGachey, X. Jiang, and D. Xu. VioCluster : Virtualization for dynamic computational domains. *IEEE IC on Cluster Computing (Cluster 2005)*, 2005.

[13] X. Jiang and D. Xu. Violin : Virtual internetworking on overlay infrastructure. Technical report, Purdue University, 2003.

[14] J. Chase et all. Dynamic virtual clusters in a grid site manager. *The 12th International Symposium on High Performance Distributed Computing)*, 2003.

[15] K. Appleby et all. Oceano-SLA based management of a computing utility. *In Proceedings 7th IFIP/IEEE International Symposium on Integrated Network Management*, 2001.

[16] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B. Palter. Layer two tunneling protocol l2tp. USA, August 1999. RFC Editor.

[17] Ala Rezmerita, Tangui Morlier, Vincent Neri, Franck Cappello. Private Virtual Cluster : infrastructure and protocol for Instant Grids. *In Proceedings of the Int. Euro-Par Conf. on Parallel Processing (Euro-Par 2006)*, LNCS 4128, pp.393-404, Dresden, Germany, 2006.

[18] Timothy W. Curry. Profiling and Tracing Dynamic Library Usage Via Interposition. *USENIX Summer*, pp.267-278, 1994.

[19] Maxim Krasnyansky. Virtual Point-to-Point(TUN) and Ethernet(TAP) devices. Virtual tunnels over tcp/ip networks. http ://vtun.sourceforge.net/tun/

[20] Ali Raza Butt, Rongmei Zhang, and Y. Charlie Hu. A self-organizing flock of Condors. *Journal of Parallel and Distributed Computing*, 66(1), pp. 145-161, 2006.

[21] J. Reynolds and J. Postel. Rfc 1340 - assigned numbers. USA, 1992. RFC Editor.

[22] An Open Source SSL VPN Solution by James Yonan. http ://openvpn.net/

[23] HTTP Tunneling Interface. http ://htun.runslinux.net/

[24] Virtual Tunnels over TCP/IP networks. http ://vtun.sourceforge.net/

[25] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. *In Proceedings of the USENIX 2005 Annual Technical Conference*, FREENIX Track, pages 4146, 2005.

[26] Cooperative Linux : Cooperative Virtual Machine. http ://www.colinux.org/

[27] Rusty Russell and Harald Welte. Linux Netfilter Hacking HOWTO. http ://cvs.netfilter.org/cgi-bin/cvsweb/netfilter/documentation/HOWTO/

[28] Bryan Ford, Pyda Srisuresh and Dan Kegel. Peer-to-peer communication across NATs. *USENIX Annual Technical Conference*, 2005.

[29] Universal Plug and Play Forum.http ://www.upnp.org/standardizeddcps/

[30] C. Huitema J. Rosenberg, J.Weinberger and R.Mahy. Rfc 3489 - STUN - simple traversal of UDP through NATs. USA, March 2003. RFC Editor.

[31] Avishai Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, volume 37, number 6, pages 62-67, 2004.

[32] A Network Protocol Independent Performance Evaluator. http ://www.scl.ameslab.gov/netpipe/

[33] X. Evers, J. F. C. M. de Jongh, R. Boontje, D. H. J. Epema, and R. van Dantzig. Condor flocking : load sharing between pools of workstations. TechnicalReport DUT-TWI-93-104, DelftUniversity of Technology, Department of Technical Mathematics and Informatics, Delft, The Netherlands, 1993.

[34] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid, in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, *Grid Computing : Making The Global Infrastructure a Reality*, John Wiley, 2002

[35] Netfilter/iptables project homepage. http ://www.netfilter.org/

[36] Tatusova TA, Madden TL. BLAST 2 sequences, a new tool for comparing protein and nucleotide sequences. *FEMS Microbiol Lett 1999 ; 174 : 24750.*

[37] Luis F. G. Sarmenta. Sabotage-Tolerance Mechanisms for Volunteer Computing Systems. *ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, May 15-18, 2001.

[38] B. Cohen. Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer Systems*, 2003

[39] B. Wei and G Fedak and F. Cappello. Collaborative Data Distribution with BitTorrent for Computational Desktop Grids. *International Symposium on Parallel and Distributed Computing (ISPDC 2005)*, pp. 250-257, IEEE Computer Society, 2005.

Table des matières

Unité de recherche INRIA Futurs
Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803