One step further in large-scale evaluations: the V-DS environment

Benjamin Quétier, Mathieu Jan, Franck Cappello

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

One step further in large-scale evaluations: the V-DS environment

Benjamin Quétier, Mathieu Jan, Franck Cappello

N° 9999

December 2007

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--9999--FR+ENG

One step further in large-scale evaluations: the V-DS environment

Benjamin Quétier*, Mathieu Jan*, Franck Cappello*

Thème NUM — Systèmes numériques
Projets Grand-Large

Rapport de recherche n° 9999 — December 2007 — 29 pages

Abstract: Validating current and next generation of distributed systems targeting large-scale infrastructures is a complex task. Several methodologies are possible. However, experimental evaluations on real testbeds are unavoidable in the life-cycle of a distributed middleware prototype. In particular, performing such real experiments in a rigorous way requires to benchmark developed prototypes at larger and larger scales. Fulfilling this requirement is an open challenge but is mandatory in order to fully observe and understand the behaviors of distributed systems. In this work, we present our Virtualization environment for large-scale Distributed Systems, called V-DS, as an answer for reaching scales steps further than current experimental evaluation scales. Then, through the use of V-DS, we experimentally benchmark the scalability of three P2P libraries by reaching scales commonly used for evaluations through simulations. Notably, our results on the Grid'5000 research testbed show that only one of the P2P library can optimally build overlays, whatever the size of the overlay is.

Key-words: performance evaluation, large scale, virtualization, peer-to-peer.

* LRI/INRIA Futurs, University of Paris South, 91405 Orsay, France, {Benjamin.Quetier}{Mathieu.Jan}{Franck.Cappello}@lri.fr

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

Une étape supplémentaire dans les évaluations à large échelle : l'environnement V-DS

Résumé : Valider les systèmes distribués actuels et futurs visant les infrastructures à grande échelle est une tâche complexe. Plusieurs méthodologies sont possibles. Toutefois, des évaluations pratiques sur des plate-formes réelles sont inévitables dans le cycle de vie d'un intergiciel distribué. En particulier, réaliser de telles expériences de manière rigoureuse nécessite d'évaluer les prototypes développés à des échelles de plus en plus grandes. Remplir cette exigence est un défi mais est nécessaire pour pleinement observer et comprendre les comportements des systèmes distribués. Dans ce papier, nous présentons notre environnement de virtualisation pour les systèmes distribués à grande échelle, appelé V-DS, comme une réponse à ce besoin d'atteindre des échelles plus larges que celles actuellement utilisées pour des expériences réelles. Puis, à travers l'utilisation de V-DS, nous évaluons de manière expérimentale le passage à l'échelle de trois bibliothèques pair-à-pair (P2P), en atteignant des échelles classiquement utilisées pour des évaluations via des simulations. Nos résultats sur la plate-forme de recherche Grid'5000 montrent notamment que seulement une des trois bibliothèques P2P étudiées peut construire de manière optimale un réseau logique, quelque soit la taille de ce dernier.

Mots-clés : évaluation de performance, large échelle, virtualisation, pair-à-pair.

1 Introduction

Current and next generation of distributed systems, for instance content delivery on P2P networks [31], aim to target large-scale distributed infrastructures such as Internet or grids. Validating systems at these scales is a complex task. It requires to verify and benchmark proposed algorithms, as well as use produced outputs as inputs to drive software developments. For the evaluation process, several methodologies are possible. When the issues to investigate can be reduced to a simple distributed algorithm, the study of the system behavior can be done rather analytically or through simulations. However, when numerous parameters must be considered and complex interactions occur between resources, modeling becomes impractical because of the difficulty to capture and extract factors influencing the distributed systems behaviors. In addition, a survey of, for instance, P2P simulators [27] have shown that: 1) half of the simulators are inactive projects, 2) most of them are lacking of documentations for using them as well as an easy process to collect new statistics and 3) some of them wrongly implement P2P protocols. Furthermore, theoretical evaluations are certainly necessary, but they clearly are only a preliminary step. To fully observe and therefore understand the behavior of a proposed distributed system, *experimental evaluations* on real testbeds are unavoidable. In addition, such evaluations are mandatory to validate (or not!) proposed models of these distributed systems, as well as to elaborate new models.

However, performing real experiments of distributed systems in a rigorous way requires to benchmark developed prototypes at larger and larger scales in order, for instance, to evaluate the scalability of proposed algorithms. As of today and to the best of our knowledge, no testbeds provide the mandatory full-set of tools to perform experiments at large scale and in a rigorous way, i.e. in a controlled way. Consequently, very few practical evaluations of for instance P2P algorithms at large-scale are reported, a field that should benefit from the use of such experiments. For example, [8] and [2] provide such experimental evaluations, but by using 425 peers over 150 nodes of the PlanetLab testbed [7] and 580 peers over 580 nodes of the Grid'5000 testbed [6]. While such scales are clearly beyond current scales for practical evaluations, they are still one magnitude behind scales commonly used for evaluations through simulations. Furthermore, when larger scales are reached through emulation, no clear guarantees are given, for instance, on the fair sharing of CPU between running peers [21]. In addition, while being extremely useful to the distributed community, the use of PlanetLab as a testbed for software evaluations opens the question of reproducible and accurate results. This testbed indeed lacks of control over the experimental conditions. Furthermore, these experimental conditions might not be the appropriate one for, for instance, the evaluation of a P2P system [10].

Therefore, to enable an experimentally-driven approach for the design of next generation of large scale distributed systems, developing appropriate evaluation tools is an open

challenge. To address this issue, in this work we answer two questions. The first question we address is: **What are the requirements such tools should observe to perform real experiments at scales of ten thousands entities in a controlled way?** Before answering this question, Section 2 presents some related work in this area. Then in Section 3, we present our Virtualization environment for large-scale Distributed Systems, called V-DS, as a partial answer to the previous question. It allows reaching scales steps further with existing experimental tools. The second question we answer, as a case study to show the strength of our environment, is: **What is the convergence time of the routing structures of three P2P DHT-based algorithms?** Therefore, in Section 4, we describe three P2P libraries, namely Chimera [18], Khashmir [29] and i3/Chord [4], which implements Tapestry [32], Kademlia [19] and Chord [28] algorithms respectively. We then formally define this notion of convergence. Finally, thanks to V-DS, we report in Section 5 our results for this metric.

2 Related work

Recently, the number of large scale distributed infrastructures has grown. However, these infrastructures usually fall either into the category of production infrastructures, such as EGEE [9] or DEISA [20], or in the category of research infrastructures, such as PlanetLab. To our knowledge, only one of the testbeds in the latter category, namely Grid'5000 [6], meet the mandatory requirement for performing reproducible and accurate experiments: full control of the environment.

For instance, PlanetLab [7] is a good example of testbed lacking of means to control experimental conditions. Nodes are indeed connected through Internet, and a low software reconfiguration is possible. Therefore, it is difficult to mimic different hardware infrastructure components and topologies, since PlanetLab depends on a specific set of real life conditions. Consequently, it may be difficult to generalize to other environments results obtained on PlanetLab, as pointed out by [10]. Grid'5000 consists of 9 sites geographically distributed in France and aims at gathering a total of 5,000 CPUs in the near future. Grid'5000 [6] is an example of testbed which allows experiments in a controlled and precisely set environment. Grid'5000 indeed provides a way to reconfigure the full software stack between the hardware and the user on all processors. However, much work remains to be carried out for injecting or saving, in an accurate and automatic manner, experimental conditions in order to reproduce experiments. Finally, Emulab [30] is an experimental environment that aims to integrate simulated, emulated and live networks into a common framework, configured and controlled in a consistent manner for repeated research. However, this project focuses only on the full reconfiguration of the network stack.

Software environments for enabling large-scale evaluations most closely related to ours are [5] and [21]. [5] is an example of integrated environment for performing large-scale experiments, via emulation, of P2P protocols inside a cluster. The proposed environment allows to deploy and run 1,000,000 peers, but at the price of changes in the source codes of tested prototypes and support only Java based applications. Besides, this work concentrates on evaluating the overheads of the framework itself and not on demonstrating the strength of it by, for instance, evaluating P2P protocols at large scales. In addition, the project provides a basic and specific API suited for P2P systems only. P2PLab [21] is another environment for performing P2P experiments at large-scale in a (network) controlled environment, through the use of Dummynet [25]. However and as for the previously mentioned project [5], it relies on the operating system scheduler to run several peers per physical node, leading to CPU time unfairness.

3 V-DS: enabling large-scale evaluations

3.1 General requirements

A current trend is to push an experimentally-driven approach for evaluating large-scale distributed systems and services, in addition to analytical and simulation evaluations [1]. Consequently, appropriate environments for performing such experiments should be designed and developed. Such environments should at least meet the following three properties.

3.1.1 Scalable

an environment for experimental evaluations should provide a means to reach large-scales, up to ten thousands of entities or more. Host virtualization technologies arise as a solution for folding a distributed system, made of l entities, on a set of m physical nodes, with $l \gg m$. In [23], we have shown that the Xen [3] virtualization technology can easily incorporate a large number of VMs with a negligible overhead for the physical machine.

3.1.2 Accurate

furthermore, the environment for large-scale experimental evaluations should manage virtualized resources in an accurate way. It should be transparent, for entities of a distributed system, that they are running in a physical system (cluster(s) or grid(s)) smaller than the virtualized system. For instance, fair CPU sharing should be enforced and each entity should be isolated from the others. Another example is the linearity performance of virtualization tools. The performance degradation of every virtual machine should evolve linearly with

the increase of the number of virtual machines. Again in [23], results show that Xen can perfectly fulfill this requirement. In addition, the use of experimental condition injectors is required to benchmark distributed system under realistic use cases. Finally, benchmarking environments should provide reproducible experimental conditions through deep control of the experiment configuration, execution and measurements. Consequently, it should be possible to capture the experimental conditions from real platforms monitoring for enabling reproducible experiments.

3.1.3 Configurable

finally, the targeted environment should provide a custom and optimized view of the physical infrastructure used. Distributed systems requirements may indeed be different from a system, as well as from an experiment, to another. For instance, it should be possible to support different operating systems, in addition to different version of the same operating system. Besides, the environment for large-scale evaluations should also be open to the use of various tools for controlling experiments, such as fault [11] or workload injectors [13] as well a deployment tools [16].

3.2 General architecture

V-DS is our proposal of such an environment for enabling large-scale experiments of distributed systems. V-DS virtualizes distributed systems entities, at both operating and network level, by providing each virtual node its proper and confined operating system and execution environment. Thus, V-DS virtualizes and schedules a full software environment for every distributed system node. It allows to fold a distributed system in a accurate way and up to 100 times larger than the experimental infrastructure [23], typically a cluster. This folding ratio is a key point for allowing large-scale experiments.

For the operating system virtualization, V-DS is based on the Xen [3] virtualization tool in its version 3.0.4. Our previous work [23] has indeed demonstrate better results for Xen compared to other virtualization technologies. In addition, Xen get some interesting network configuration capabilities which is central in the injection of network topologies. V-DS mainly consists of 2,000 lines of C codes for generating VMs configurations as well as for deploying and launching them. Section 3.4 describes in more details the use of V-DS.

Figure 1 shows the general architecture of V-DS. Thanks to Xen, m physical nodes (noted $PM - m$ and also called host domain is the Xen terminology), typically a cluster, run each n virtual nodes (noted VM-m-n with $n = 1..100$ and also called guest domains). Each virtual node is configured with two different IP addresses, one being part of the administration network of V-DS and the other being part of the so called “experiments” network.

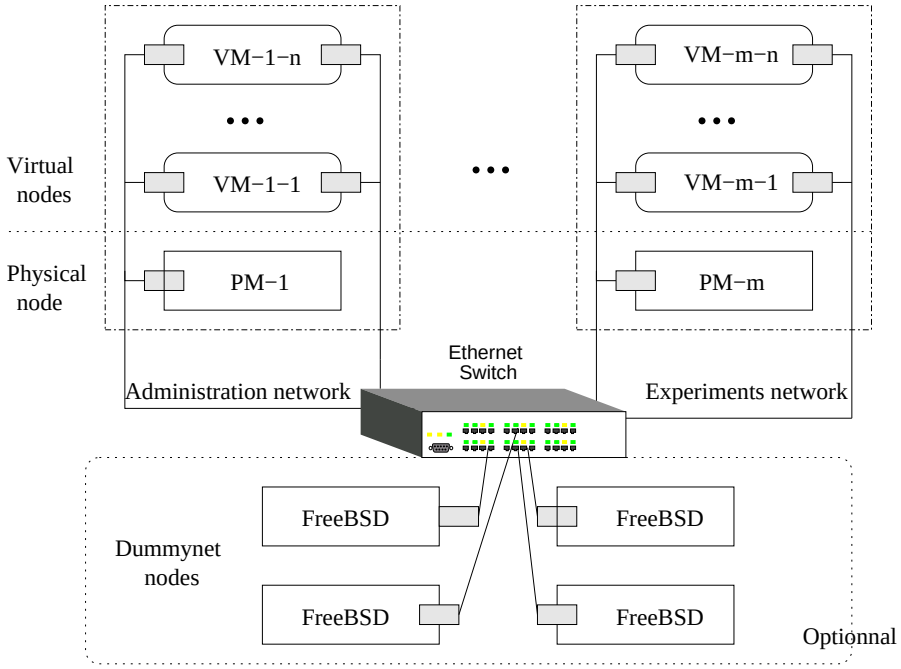

Figure 1: Overview of the architecture of V-DS.

This design choice allow to decouple the network seen by distributed systems under test from the network used for interacting with V-DS itself. Note that physical nodes are only part of the administration network. Finally, V-DS can optionally leverage the use of a various number k of FreeBSD nodes configured with Dummynet [25] (called Dummynet nodes on Figure 1. Dummynet allows to emulate various network conditions, at link level, by for instance introducing delays, limiting the bandwidth, dropping packets, etc. The choice of the number k of FreeBSD nodes is currently let to the user ($k = 4$ on Figure 1). When Dummynet nodes are used, virtual nodes routing tables are configured such that every packet sent goes through at least one Dummynet node to apply user configured network conditions. Every FreeBSD node manage the network traffic of m/k physical nodes or more precisely of $n * m/k$ virtual nodes.

Note that for all communications with physical nodes (for instance in the launch VMs step), V-DS relies on the use of `Taktuk` [12] for optimized `ssh` connections.

3.3 Requirements for using V-DS

In order to be able to run V-DS, a user needs to have a Linux kernel patched with Xen support for the host domain and all guest domains. In addition, various packages, such as for instance `gcc` and `python` are required for running Xen. The user also need to keep in mind that resources of a physical node are shared amongst all VMs running on top of the host domain.

3.3.1 CPU

the Xen hypervisor evenly allocates the CPU time between all VMs.

3.3.2 Memory

every VM needs a minimal amount of memory to run correctly. For instance, a kernel and some basic services or daemons like `ssh` needs approximatively 16MB. Let M_{vm} be the memory specified, by the user, for a VM and n the number of VMs on the host domain. In addition to its default memory requirement, the footprint memory of the host domain increases with n. Let M_{pm} be this fixed amount of memory that the host domain requires (using Xen $M_{pm} = 260$MB) and $M_{pm_{vm}}$ be the increased memory due to the management of a VM. Consequently, when the user sets n and M_{vm}, the following formula should be observed: $M_{pm} + n * (M_{pm_{vm}} + M_{vm}) < M_{phy}$, where M_{phy} is the amount of physical memory of the node (for instance 2GB). A Typical value for M_{vm} is 32MB, meaning that the memory footprint of the distributed system entity should be less or equal to 16MB.

3.3.3 Disk

each VMs runs its own file system (FS). We provide a basic FS of 500MB which contain a minimal *Debian etch* of 250MB, with some standard libraries and the `gcc` compiler. Therefore, users have around 250MB of free space on each VM to add the code of distributed system under test, as well as required input data and generated results and/or log files. With the actual size of hard-disks (typically 80GB at least), this FS size allows to run more than 100 VMs on every host domain.

3.3.4 Network

let C_1 be the network throughput between a classical node and the switch, C_2 the network throughput between a FreeBSD node and the switch, U_{rate} and D_{rate} respectively the maximum upload and download rate of all VMs with $U_{rate} = D_{rate}$, $\#_{vm}$ the number of VMs per physical node, $\#_{phy}$ the number of physical nodes and $\#_{bsd}$ the number of FreeBSD nodes. The following two constraints: $(U_{rate} + D_{rate}) * \#_{vm} < C_1$ and $(U_{rate} + D_{rate}) * \#_{vm} * \#_{phy}/\#_{bsd} < C_2$ must be met by the user. Note that the user has the full control on all variables, except C_1 and C_2.

3.4 Overview of use

Figure 2 shows the different steps of the workflow for performing experiments through V-DS. V-DS comes with a default environment which provides the required libraries and software for evaluating a C/C++ based distributed system. As a first step, a user that wishes to evaluate the distributed system A may however need to customize this environment. Typically, the dependencies A in terms of libraries of, A itself and the benchmark codes need to be added in this default environment. Then, when the environment is properly configured a user can launch virtual nodes through the use of V-DS scripts. At this step, a user set the targeted scale of the experiments by specifying the number n of virtual nodes per physical node, and by giving a list of m physical nodes. The third step is optional and is taken care by V-DS scripts. It allows to configure Dummynet nodes by currently introducing network delays and limit the available bandwidth to the entities of the distributed system. Finally, the user can configure the distributed system A under test and launch a benchmark code of A. For performing this step, the user can rely on various deployment tools, possibly specific to a given distributed system (see section 3.1.3 for examples).

Note that V-DS has already been use in its previous version for testing the behavior of some MPI NAS benchmark [22], and the Condor [17] batch scheduler.

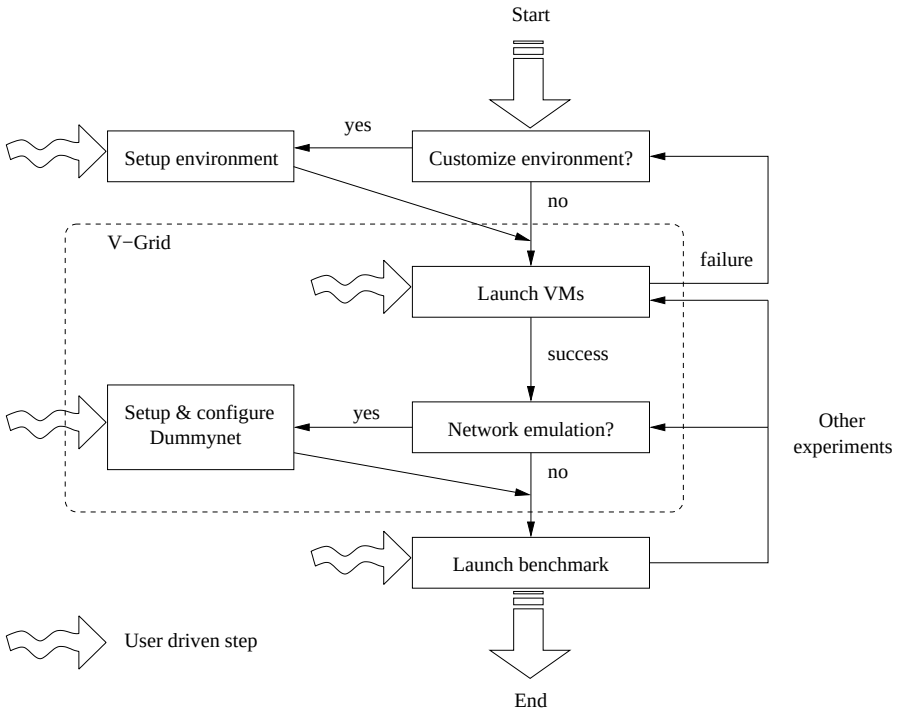

Figure 2: V-DS workflow for performing large-scale experiments.

3.5 Evaluation

As a first microbenchmark, we check if the behavior of Xen in its version 3.0.4 still meet our requirements, as in [23] in which Xen 2.0 was used. The evaluation results demonstrate that there is no change in the resources behavior in term of linearity, overhead and fairness. Note that we do not give details on this evaluation because 1) of size limitation and 2) the results do not provide novelty compared to [23].

Then, we benchmark the deployment time of the V-DS platform for various number of physical and virtual nodes. The deployment of a VD-S platform can be split into 6 procedures. The first one is named *Pre configuration*, which create some basic files like hosts resolution. The next one is the *File system (FS) copies* procedure which replicates the original FS for each VM on a physical node. Then, *Configuration* creates the configuration file of each VM. *Virtual machines (VM)'s launch* launches all VMs, while *Wait* wait for all the VMs to complete the boot. Finally, the *Network configuration* procedure configures networks used by each VM of the deployed V-DS platform, such as routing tables for instance, as well as the FreeBSD nodes. For all these steps, Figure 3 shows the deployment time of a V-DS platform where $\#_{phy} = 50$, $\#_{bsd} = 10$ and $\#_{vm} = 5, 10, 20$ and 40 (thus for a maximum number of 2000 VMs). Figure 3 demonstrates that each procedure takes a time proportional with $\#_{vm}$. Besides, half of the deployment time is taken by the *FS copies* procedure. Note that since amount of time taken by the *Pre configuration* procedure is a few seconds, it is not shown of the Figure 3.

We also benchmarked the amount of time required to deploy a V-DS platform where $\#_{vm} = 10$, $\#_{bsd} = 5$ and $\#_{phys} = 5, 50$ and 150. Results show an overhead of 20% for the total deployment time of a V-DS platform with $\#_{phys} = 150$, compared to a platform with $\#_{phys} = 5$. The higher number of `ssh` connections (even if TakTuk scale in a logarithmic scale) and the bigger routing tables, needed to handle more physical nodes, explain this overhead. We have also vary $\#_{bsd}$ from 5 to 25. Results show a negligible overhead, less than 2%.

4 Case study

In order to demonstrate the evaluation possibilities offered by V-DS, we evaluate the behavior of three P2P libraries, namely Chimera [18], Khashmir [29] and i3/Chord [4]. It is important to notice that these software are run under V-DS without any change in their source code. Thus experiments evaluate the exact same piece of software than one would run on a real P2P platform.

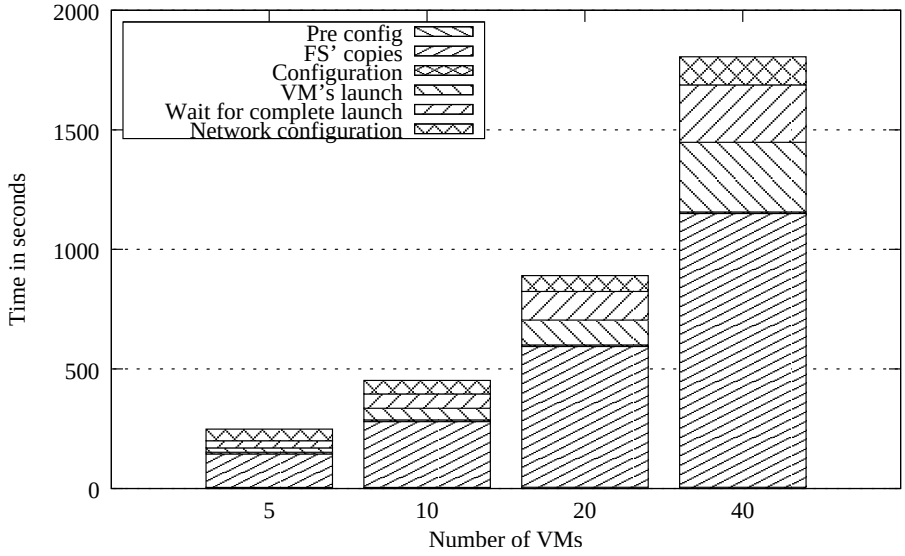

Figure 3: Evolution of the deployment time in function of the #virtual machines

4.1 Overview of DHTs

Note that for all DHTs, we define n as the size of the overlay. In addition, the lookup complexity of all DHTs is in $O(log\ n)$.

4.1.1 Chimera

is a P2P library designed to provide to applications a light-weight implementation of prefix-based routing protocols, such as Tapestry [32]. Within Chimera, each peer is uniquely assigned an identifier in the range $[0, 2^{160})$, of a so called key space. For its routing algorithms, Chimera therefore uses on every peer two structures: 1) a leaf set L of $2k$ peers immediately preceding and following in the key space and 2) a prefix-matching routing table R, where the i entry of the l row (noted $R[l][i]$) is a peer[1] whose identifier matches l digits of the local peer identifier and whose $(l+1)$ digit is i.

In the latest version of Chimera (1.20), L is periodically updated as shown by Algorithm 1. We note L_P the leaf set of peer P. At each step of the algorithm, the liveness of all peers included in L_P is checked, and if needed removed from L_P (lines 3 to 7).

[1]We discard topology-based optimization for selecting peers in R.

Algorithm 1 Pseudo-code of the periodic algorithm used to maintain the leaf set L_P of a Chimera peer P.

```
1: count = 0
2: repeat
3:    for peer ∈ L_P do
4:       if ping(peer) ! = success then
5:          remove peer from L
6:       end if
7:    end for
8:    if count == 2 then
9:       count = 0
10:      for peer ∈ L_P do
11:         send L_P to peer
12:      end for
13:   else
14:      count + +
15:   end if
16:   wait for LEAFSET_CHECK_PERIOD
17: until peer stopped
```

Then, one third of the time, L_P is being shared with all peers of L_P (lines 6 to 9 of Algorithm 1). In response, contacted peers return their leaf set[2]. In each received leaf set, closer neighbors in the key space than the current one are added to either the right or the left leaf set of the peer. This is how a peer P of Chimera learn about new peers logically around him. The elapsed time between two iterations of the algorithm is controlled by the `LEAFSET_CHECK_PERIOD` constant (set by default to 20 seconds). Finally, note that Chimera does not implement any algorithm for the maintenance of R.

4.1.2 Khashmir

is a DHT based on Kademlia algorithms [19] and its XOR metric for defining the distance between two peers. Within Khashmir, each peer is uniquely assigned an identifier in the range $[0, 2^{160})$. For its routing algorithms, a Khashmir peer uses a list of peers for each i digit of its identifier (with $i = 0..39$). Such lists are called `k-buckets` in the Kademlia terminology, where k specify the maximum size of the lists (for Khashmir $k = 8$). The ith list l of peer P contains peers whose identifier matches the i digits of P identifier. These lists implement a least-recently seen peer eviction policy, if the tested peer does not respond to a ping request.

The `k-buckets` of Khashmir are periodically updated (Algorithm not shown due to space constraint). At each step of the algorithm, a `k-bucket` is requested to be updated, if it has not been updated since the value of the `BUCKET_INTERVAL` constant. The default value of this constant is 1 hour. To update a `k-bucket` a new identifier is randomly generated in this bucket and looked up. Note that the `findNode` function generates by default four concurrent requests. The elapsed time between two iterations of the algorithm is controlled by the `CHECKPOINT_INTERVAL` constant (set by default to 15 minutes). Finally, note that a `k-bucket` can also be updated when new nodes are discovered, when for instance forwarding requests.

4.1.3 i3/Chord

is a C implementation of the Chord P2P DHT algorithms [28]. Within i3/Chord, each peer is uniquely assigned an identifier in the range $[0, 2^{160})$. For its routing algorithms, i3/Chord relies on two structures: 1) a list S of k peers immediately following in the key space and 2) a table F of (at most) l peers, called the finger table. The i entry of F contains the identity of the first know node that succeeds the local peer by at least 2^{i-1} on the key space. The values of k and l are set by default to 7 and 160 respectively in i3/Chord. Finally, note that

[2] We fi xed the Chimera function responsible for inserting a peer in a leaf set.

a i3/Chord peer uses a list P of m peers immediately preceding in the key space (with m set to 2 by default). The goal of this structure is to simplify join and leave mechanisms.

In i3/Chord (0.3), a stabilization procedure periodically updates S, F and P, as shown by Algorithm 2. A each step of the algorithm, a peer from the union of F and P is tested,

Algorithm 2 Pseudo-code of the periodic algorithm used to maintain the routing structures of i3/Chord peer P.

```
1:  repeat
2:     ping(round_robin(F ∪ P))
3:     succ = first_suc()
4:     pred_succ = stabilize(succ)
5:     if pred_suc not in F then
6:        insert(pred_succ, S)
7:        notify(pred_succ)
8:     else
9:        update(pred_succ, S)
10:    end if
11:    if (idx % PERIOD_PING_PRED == 0) then
12:       pred = first_pred(P)
13:       ping(pred)
14:    end if
15:    ping(ask_succ(round_robin(S)))
16:    ping(ask_succ(round_robin(P)))
17:    idx++
18:    wait for (STABILIZE_PERIOD)
19: until peer is stopped
```

in a round-robin manner (line 2). Then, the first successor $succ$ of P is asked to return its predecessor, noted $pred\_succ$, through the call of the `stabilize` function (lines 3 and 4). If $pred\_succ$ is not known from P, it is used as a new successor and $pred\_succ$ is notified of the presence of P (lines 5 to 7). This is how P learn from a new successor and $pred\_succ$ of a new predecessor. Every `PERIOD_PING_PRED` period, the first predecessor is tested to check its liveness and its replacement is requested if needed (lines 11 to 13). Finally, a peer from S, as well as from P, is asked in a round-robin manner to return its successor. These successors are tested before being included, if needed, in either S or P of the peer (lines 15 and 16). Note that the elapsed time between two iterations of the algorithm is controlled by the `STABILIZE_PERIOD` constant (line 19), set by default to 1 second (!) in i3/Chord.

4.2 Related work

Let us remind the reader that the goal of our evaluation is two-fold: 1) answer a specific research question related to the performance of P2P DHTs under real size conditions and at large-scale, but more importantly 2) to show the strength of the V-DS approach for large-scale evaluations of distributed systems. We therefore provide only a short overview of related work in the area of our targeted case study for V-DS, in order to motivate this evaluation.

Papers introducing DHTs such as Tapestry [32] or Chord [28] have evaluated the benefits of this approach. Since then, benchmarks have focused on evaluating the performance of routing strategies in terms of number of hops, under various conditions and for alternative routing policies. To summarize, a main targeted goal is to evaluate the trade-off between the number of required hops to perform a lookup and the size of the routing structures. Evaluations have been mainly performed through theoretical analyses and simulations, allowing to reach scales up to 100,000 peers [26]. Few works, even more recent ones have focused on evaluating P2P protocols through real experiments, but still at limited scales. To our knowledge, only [2, 8, 24] and [21] have performed such kind of work.

To summarize, P2P DHTs has been the subject of various and extensive evaluations, such as routing efficiency and resilience to churn. However, such performance evaluation have been mainly performed either at: large-scale but through simulations or using real experiments but still a limited scales (up to hundreds of peers). While simulations are clearly useful, the Network Simulator (`ns-2`) experiences have for instance shown that it can sometimes lead to unfounded confidence in evaluated protocols [10]. The goal of this case study for V-DS is therefore to evaluate for a given metric the performance of P2P DHTs, by reaching scales classically reached through the use of simulations, that is towards 10,000 peers.

4.3 Benchmark

4.3.1 Motivations and goals

to the best of our knowledge, few works report the required time for a various large number of peers to fully and correctly build their routing structures. Note that during this amount of time, lookups either fail in the worst case or are delayed. In the latter case, the complexity is indeed increased to $O(n)$ instead of $O(log\ n)$. Routing structures maintenances are therefore a priority for achieving optimal lookups. Precisely, the goal of our benchmark is to measure this required amount of time, that we note t_c, for a various number of peers n to build a new overlay. This benchmark simulates a flash-crowd arrival of peers in an overlay, and enable the analysis of how of the overlay react to such conditions. The outcome of

this benchmark will be a precise answer about the availability of the most efficient routing process for the given set of tested DHTs. Such an answer is useful for P2P users as well as P2P developers. Note that for Chimera the observed routing structure is restricted L, as no maintenance is implemented for R. For Khashmir, `k-buckets` are monitored. Finally, we restrict the monitoring of i3/Chord routing structures to S and P.

4.3.2 Notions and notations

we define the previously introduced variable t_c as the convergence time of a peer P, that we note t_c^P. To be more general, t_c should be defined as the difference between the current convergence time and the end of the last convergence period of a peer. We also define the convergence time of a DHT (noted DHT_{CT}) as the maximum of all t_C^P across a given overlay. In addition, to monitor and analyze the evolution of routing structures, we introduce the notion of distance that we note $d(t)$, as it depends on the time. We define $d(t)$ as the difference, in terms of IDs, between the content of the routing structures at time t and the expected routing structures of a stabilized overlay (i.e. routing structures with optimal values allowing $O(log\ n)$ lookups). Mathematically, $d(t)$ on peer P is defined as:

$$d_P(t) = \sum_{i=1}^{r} \sum_{j=1}^{z} abs(R_P^i(t)[j] - R_{theo}^i[j]) \quad (1)$$

where r is the number of routing structures used by a DHT and z is the number of entries in the considered routing structure. For instance, r is equals to 2 and 1 for respectively Chimera and Khashmir, while j is equals to the variable k for Chimera and Khashmir. However, k is equal to 4 in Chimera, whereas it is equal to 8 in Khashmir (see Sections 4.1.1 and 4.1.2). Note that R_{theo} defines a routing structure with optimal values, according to its algorithmic goal for this structure, while $R_P^i(t)$ is the content of the i routing structure of peer P. More generally, peer P is said to have converged when $d_P(t) = 0$. Note that a peer can converge at time t_1 while at time t_2 it may not have, even though $t_1 < t_2$. The same applies for DHT_{CT}. Finally, we note t_{end} the end time of an experiment.

To illustrate these notions and notations, let us take a simple example. Let us assume that $n = 10$ and that these 10 peers are simultaneously started in order to build an Chimera overlay. Peer IDs are numbered from 1 to 10. Let us remind the reader that for Chimera, r is equals to 2. The routing structures are indeed a left and a right leaf sets, both of size k. In this example, k is equals to 4. Then, the expected left leaf set of the peer 2 is $(8, 9, 10, 1)$ whereas its expected right leaf set is $(3, 4, 5, 6)$. Let us further assume, that the left and right leaf sets of peer 2 at time t_1 are respectively $(7, 9, 10, 1)$ and $(3, 5, 6, 7)$. Therefore, the distance of P

at time t_1 is defined as follows[3]: $d_P(t_1) = abs(8-7)+abs(4-5)+abs(5-6)+abs(6-7) = 4$. Therefore in this example, at time t_1 peer 2 has not yet converged. However, when $t == t_c^2$ its left leaf set will be equal to $(8, 9, 10, 1)$ while its right leaf set will be equal to $(3, 4, 5, 6)$, leading to a value of 0 for d.

4.3.3 Benchmarks description

for benchmarking purpose, we have instrumented each DHT. Any modification of the content of their routing structures is logged. These generated logs are analyzed off-line in order to compute the t_c of each peer, if it has converged, as well as various other statistics. Note that the t_c we measure is a Xen domain virtual time, that is the execution time only when the virtual node hosting the process is running. Our benchmark iteratively starts n peers in order to build a single overlay. Note that for $n <= 100$, peers are started before the first run of the observed algorithm. From these n peers, one is used as a bootstrap peer for the remaining $n - 1$ peers. Then, we monitor the evolution of routing structures for typically 2 hours. No churn conditions are injected while the experiment takes place, i.e. we assume a static overlay throughout the remainder of the experiment. We perform this evaluation for various values of n using V-DS ($n = 10, 100, 1000, 4369$).

5 Results and discussion

5.1 Benchmark conditions

For all our benchmarks, we use the largest cluster of the Grid'5000 testbed, namely Grid eXplorer, which is made of 342 nodes. Nodes used for the experiments mainly consist of machines using dual 2.0 GHz AMD Opteron, outfitted with 2 GB of RAM each, and running a 2.6 version Linux kernel; the hardware network layer used is a Giga Ethernet (1 Gb/s) network (therefore $C_1 = C_2 = 1\ Gb/s$, see Section 3.3). When n is equal to 10, 50 and 100 we use 1 virtual node (VM) per host. When n is equals to 1000, we use 20 VMs per physical node, therefore using 5 and 50 physical nodes. Finally, when $n = 4369$ we use 200 physical nodes and 50 VMs per physical node. Benchmarks were executed using Chimera version 1.20, Khashmir from its CVS directly and i3/Chord version 0.3. Chimera and i3/Chord were compiled using `gcc` 4.0 with the `-O2` level of optimization, as Chimera and i3/Chord are C based implementations. Khashmir is implemented in Python.

Note that we generate peers ID from either 0 (Chimera and i3/Chord cases) or 1 (Khashmir) up to the overlay size. This eases the reproducibility of our benchmarks, as well as the

[3] For the sake of clarity we omit terms leading to a value of 0.

computation of the expecting routing structures of peers. The value of the Chimera constant `LEAFSET_CHECK_PERIOD` is set to 20 seconds. We configure the Khashmir constants `STALENESS` and `CHECKPOINT_INTERVAL` to 20 and 60 seconds respectively. Finally, we set the i3/Chord constant `STABILIZE_PERIOD` to 20 seconds. We understand that these settings might not cover the full usage of these systems. However, such an evaluation is outside the scope of this paper.

5.2 Experimental results

Note that due to space limit, we generally focus on one particular value of n. However, all experiments follow the same pattern, unless explicitly indicated.

5.2.1 Chimera

when n is equals to 10, 50, 100, and 1000 no DHT_{CT} can be computed[4]. For $n = 10..100$, no peers of a given experiment have indeed converged, despite $t_{end} = 2$ hours. The only exception is for the bootstrap peer of the overlay: peer 0 converges when n is equals to 10 and 100, but surprisely not when $n = 50$. Figure 4 shows value of d across the key space at time t_{end}. The value of d increases with IDs, which is explained by the iterative launch process of peers (see Section 4.3.3). Figure 5 shows the evolution of the routing structures of Chimera for respectively the bootstrap peer (peer 0, upper part) and peer 99 (bottom part). The y-axis is the key space of the overlay, while a line in the graph indicates the presence of the associated peer ID in the routing structures of the local peer. The upper part of this Figure shows that the bootstrap peer quickly converges ($t_c = 8.5$ seconds) by reducing the value of d for its left leaf set (k immediately following peers, see Section 4.1.1). The bottom part of Figure 5 explains why other peers within Chimera do not converge. For instance, peer 99 indeed keeps throughout the experiment peer 39 in its routing structures, even though peer 39 is not part of its expected the routing structures. Note that only the first seconds of the experiment are shown, as no modification is observed latter on.

5.2.2 Khashmir

similarly to Chimera, when n is equals to 50 and 100, no DHT_{CT} can be computed[4]. However and contrary to Chimera, a majority of peers did converged: 55% and 66% for n equals 100 and 50 respectively. Note that when peers converge, t_c is around 100 seconds. The pourcentage of peers that converge increases as the size of the benchmark overlay decreases: when $n = 10$ all peers converge. DHT_{CT} is then equals to 394 seconds and the

[4]Therefore, experiments with $n >= 1000$ were not performed.

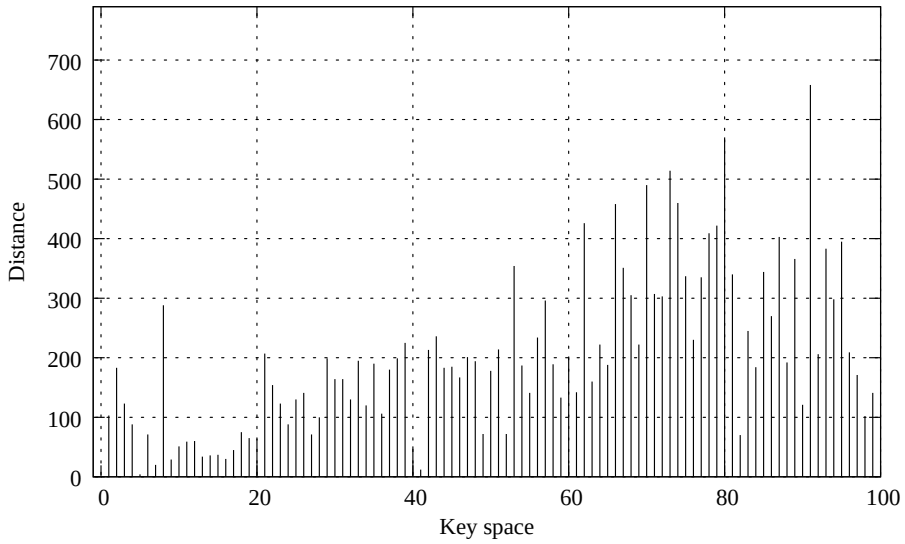

Figure 4: Value of d across the key space (down) at time t_{end} and for a Chimera overlay of $n = 100$.

average value of all t_c is 267$\pm$150 seconds. Figure 6 shows the value of d across the key space, at time t_{end}. Mainly 2 values of d exist for this experiment: 16 and 24. The upper part of Figure 7 shows the evolution of the routing structures (`k-buckets`) of peer 66, a peer that did converge. Repeatedly small gaps for each line of this graph can be observed. This indicates that routing structures of Khashmir are unstable: same IDs are constantly added and removed. Note that the size of the gaps between the availability of IDs varies, but with no link with the value of n. The bottom part of Figure 7 shows the consequences of this instability on d ($n = 100$). Its value constantly changes and does not remain equals to 0. Note that when n increases from 10 to 100, d sometimes constantly remains strictly higher than 0, after it has first reach 0. Therefore, peer 66 for instance converged only once and for a short period of time, as shown by the bottom part of Figure 7. This instability also explains why other peers do not converge even once. In their case, this instability is too high to allow them to reach, even once, a value of 0 for d.

5.2.3 i3/Chord

contrary to Chimera and Khashmir, a DHT_{CT} value can be defined for i3/Chord overlays when $n = 10, 50, 100, 1000$. DHT_{CT} is equals to 300, 1080, 2280 and 534 seconds,

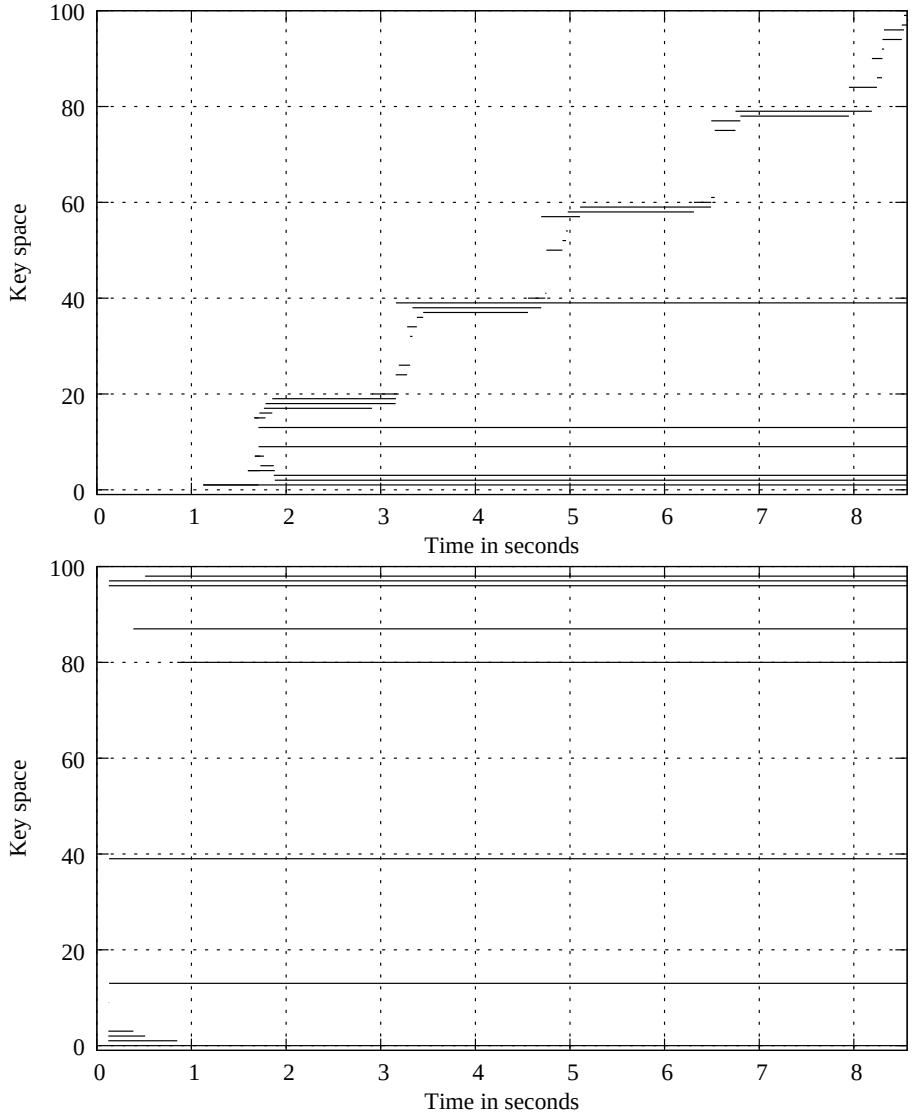

Figure 5: Evolution of the routing structures of peer 0 (up) and peer 99 (bottom), for a Chimera overlay where $n = 100$.

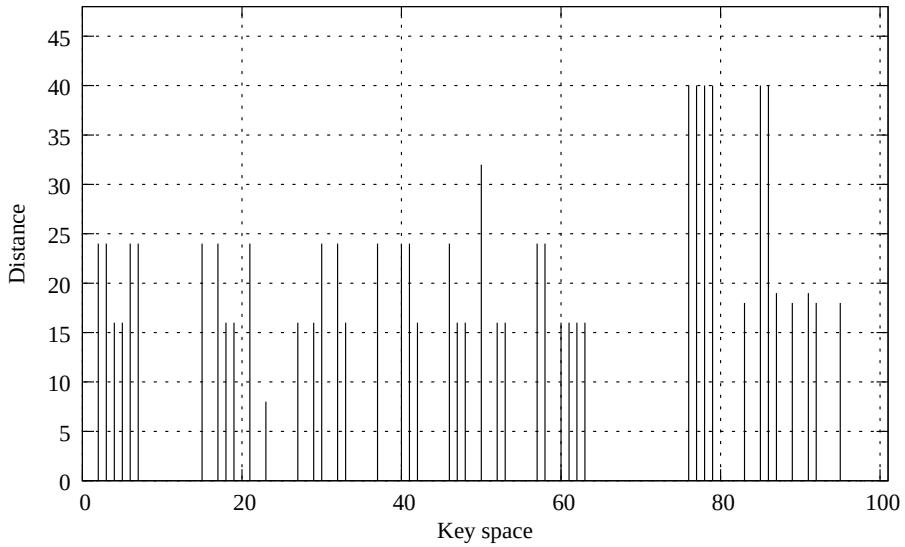

Figure 6: Value of d across the key space (down) at time t_{end} and for a Khashmir overlay of $n = 100$.

while the average convergence time of peers is equals to 172±47, 747±203 1438±395 and 32.59±19 seconds (for $n = 10$, 50, 100 and 1000). Note that lower values when $n = 1000$ are explained by the fact that 20 peers are running within a given physical host. In addition, our analysis shows that once a peer has converged, its distance value remains equals to 0 till t_{end}. Figure 8 shows value of t_c across the key space at time t_{end}. The value of t_c across the key space is of the same order, except for the first two peers. These peers indeed need to cover the whole key space to fill P, as best shown by the next Figure. Figure 9 indeed shows the evolution of the routing structures of peer 1, which is the first peer of the overlay after the bootstrap peer. We can observe that S of peer 1 quickly converges to its expected and optimal value (between 2 and 8). However, P also converges but through small jumps in key space up to its missing ID for achieving optimal values for P: 99. Note that other peers converge more quickly as they do not need to cover the whole key space for finding optimal values for P. Lines at regular intervals throughout the key space on Figure 9 shows that F is filled as expected (see Section 4.1.3). However, note that for some peers and when $n = 1000$, these lines are dashed lines therefore indicating that the content of F is constantly changing. This instability of F does not exist when $n = 100$. In addition, this

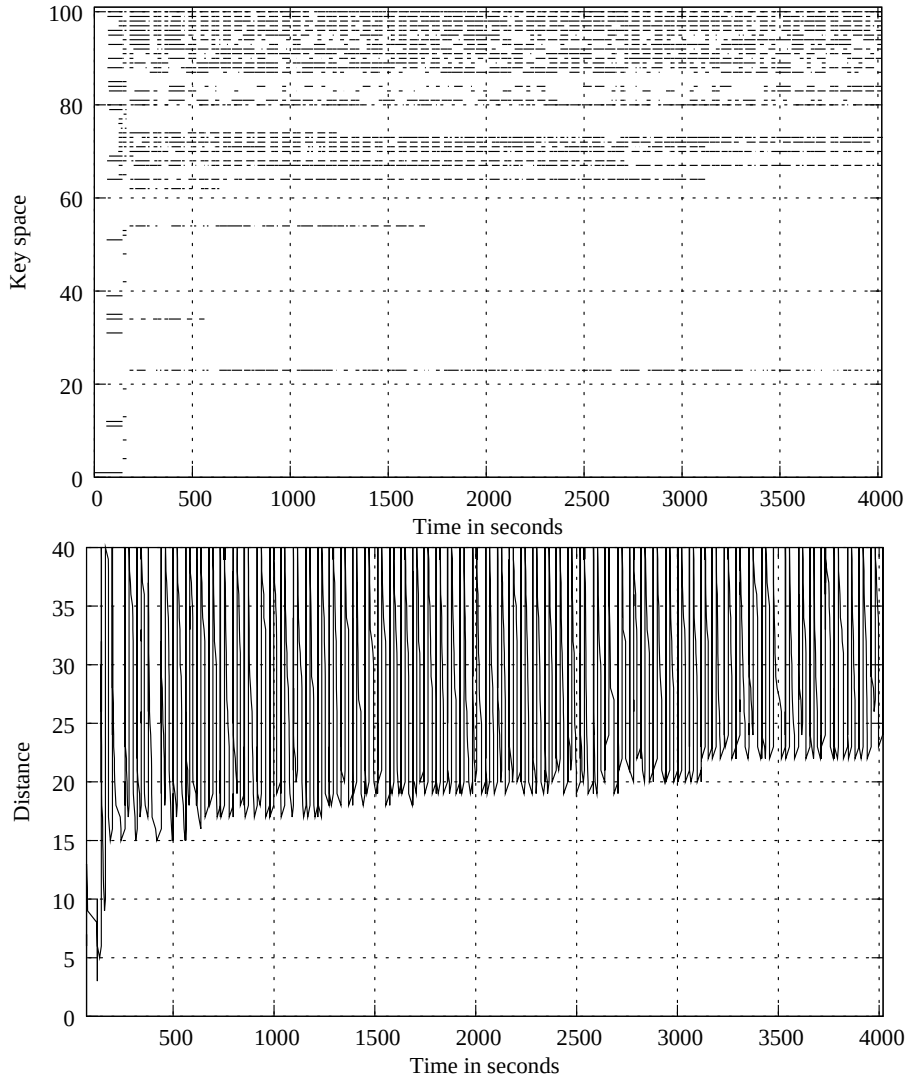

Figure 7: Evolution of the routing structures (up) and of the distance d of the Khashmir peer 66 ($n = 100$).

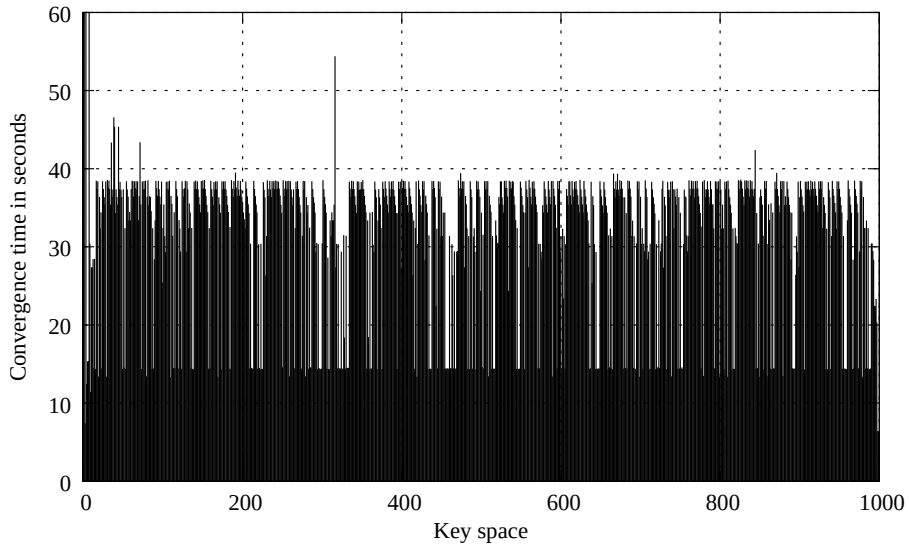

Figure 8: Value of t_c across the key space (bottom), at time t_{end} and for an i3/Chord overlay of $n = 1000$.

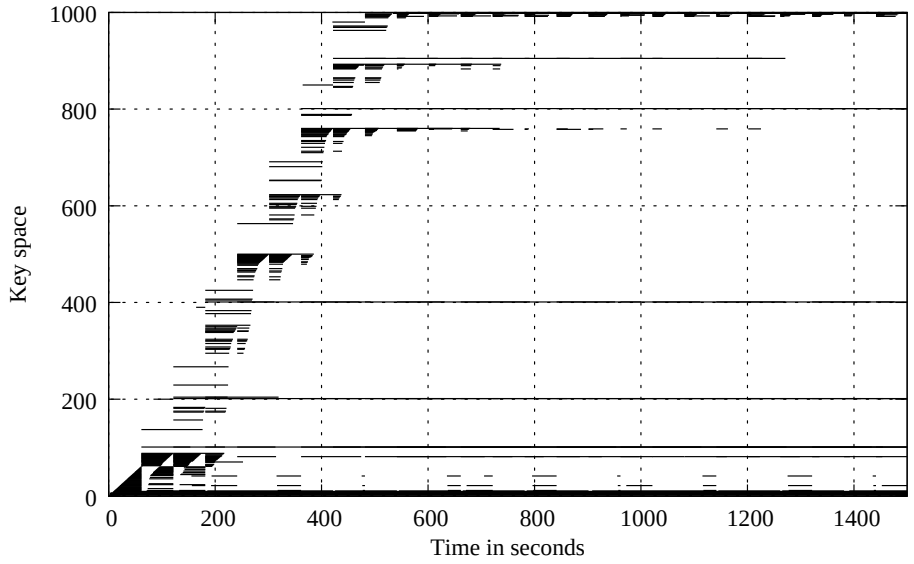

Figure 9: Evolution of the routing structures of the i3/Chord peer 1 ($n = 1000$).

figure also shows that F is periodically cleared in order to reduce the number of stored peers within F. Results for $n = 4369$ show a similar behavior.

5.3 Discussion

Chimera and Khashmir are not able to optimally build routing structures. Therefore, users can either experience increasing delays (the complexity can increase up to $O(n)$ instead of $O(log\ n)$) or even failures for their lookup requests. In both cases, it is most likely due to bugs in the implementations. For Chimera, we have for instance observed the failure of many `RPCs` (Remote Procedure Calls) during experiments. These `RPCs` are used by peers to communicate. For $n = 100$ but on a single physical node, such time outs have not been observed: all peers were able to successfully convergence as well as to keep the value of d equals to 0. The instability of routing structures of Khashmir has not been explained so far. For both implementations, V-DS does not introduce a timing overhead as we use one VM per host. i3/Chord peers are able to converge and keeps their value of d equals to 0. Therefore, i3/Chord users can benefits of the most efficient routing strategy for their requests, even for large overlays. However, starting from $n = 1000$, the content of F is unstable. Further investigations are needed to understand the potential impact (if not none) of the performance lookups of i3/Chord.

6 Conclusion

Validating current and next generation of distributed systems targeting large-scale infrastructures is a complex task. Currently performed evaluations are mainly done through mathematical analysis or simulations and lack of large-scale experiments. Very few environments indeed provide a mean for performing such kind of experiments by folding entities of a large scale distributed system on small set of physical nodes, typically a cluster. In this work, we have described our proposal for enabling large-scale experiments and validate it through an evaluation three P2P systems at large scales.

First, we have listed three properties that such an environment should observe: scalability, accuracy and means to configure the distributed system environment under. Second, we have demonstrated the power and usefulness of our environment by performing large-scale evaluations of three P2P DHTs. VD-S has indeed allow us to reach scales close to commonly used scales for simulations, towards 10,000 peers. We mainly focus on the performance of building and maintaining P2P overlays based on: 1) Chimera [18] which implements a Tapestry-like routing [32], 2) Khashmir [29] which implements a Kademlia-like routing [19] and 3) i3/Chord [4] which implements the Chord routing algorithm [28].

Notably, our results show that at large-scale only the i3/Chord is able to optimally build overlays, whatever the size of the overlay is. i3/Chord can therefore offer best performance to users for lookups (in $O(log\ n)$), while other implementations may either introduce delays or even failures in lookups.

As future work, we will investigate the impact on the routing structures of a flash-crowd arrival of various number peers on already stabilized overlays. Obviously, this benchmark is only possible with i3/Chord. Then, we would like to investigate the use at large-scale of the Mace [15] environment for building distributed systems and its model checker, in order to be more independent from implementation details. The goal would be to evaluate algorithms in addition to their implementations. Finally, we also plan to analyze the impact of various churn conditions, through the weaving of V-DS and FAIL [11] as well as the use of various availability traces, such as [14]. In this area of churn study, we will pay a special attention to compare obtained results with result already published from simulators.

Acknowledgments

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 `https://www.grid5000.fr`).

References

[1] 7th Framework Programm. New Paradigms and Experimental Facilities. `http://cordis.europa.eu/fp7/ict/fire/`, 2007.

[2] Gabriel Antoniu, Loic Cudennec, Mike Duigou, and Mathieu Jan. Performance scalability of the JXTA P2P framework. In *Proc. 21st IEEE International Parallel & Distributed Processing Symposium (IPDPS 2007)*, page 108, Long Beach, CA, USA, March 2007.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfi eld. Xen and the art of virtualization. In *Proc. of the 19th ACM symposium on Operating systems principles (SOSP'03)*, pages 164–177. ACM Press, 2003.

[4] UC Berkeley. i3/Chord. `http://i3.cs.berkeley.edu/`, 2006.

[5] Erik Buchmann and Klemens Bohm. How to Run Experiments with Large Peer-to-Peer Data Structures. 01:27b, 2004.

[6] Franck Cappello, Eddy Caron, Michel Dayde, Frederic Desprez, Emmanuel Jeannot, Yvon Jegou, Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst,

Pascale Primet, and Olivier Richard. Grid'5000: A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. In *Proc. of the 6th IEEE/ACM Int. Workshop on Grid Computing (Grid '05)*, pages 99–106, Seattle, Washington, USA, nov 2005.

[7] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, July 2003.

[8] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a DHT for Low Latency and High Throughput. In *Proc. of the 1st Symp. on Networked Systems Design and Implementation (NSDI 2004)*, pages 85–98, San Francisco, CA, USA, March 2004.

[9] EGEE Team. LCG. `http://lcg.web.cern.ch/`, 2004.

[10] Andreas Haeberlen, Alan Mislove, Ansley Post, and Peter Druschel. Fallacies in evaluating decentralized systems. In *Proc. of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, Santa Barbara, CA, USA, February 2006.

[11] William Hoarau and Sébastien Tixeuil. A language-driven tool for fault injection in distributed applications. In *Proc. of the IEEE/ACM Workshop GRID 2005*, Seattle, USA, November 2005.

[12] Guillaume Huard and Cyrille Martin. TakTuk. `http://taktuk.gforge.inria.fr/`, 2005.

[13] Alexandru Iosup and Dick H. J. Epema. GrenchMark: A Framework for Analyzing, Testing, and Comparing Grids. In *6th IEEE Int. Symp. on Cluster Computing and the Grid (CCGrid 2006)*, pages 313–320, Singapore, May 2006. IEEE Computer Society.

[14] Alexandru Iosup, Mathieu Jan, Ozan Sonmez, and Dick H. J. Epema. On the dynamic resource availability in grids. In *8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, pages 26–33, Austin, TX, USA, September 2007. IEEE.

[15] Charles Edwin Killian, James W. Anderson, Ryan Braud, Ranjit Jhala, and Amin M. Vahdat. Mace: language support for building distributed systems. In *Proc. of the 2007 ACM SIGPLAN conference on Programming language design and implementation (PLDI'07)*, pages 179–188. ACM, June 2007.

[16] Sébastien Lacour, Christian Pérez, and Thierry Priol. Generic application description model: Toward automatic deployment of applications on computational grids. In *6th IEEE/ACM International Workshop on Grid Computing (Grid2005)*, Seattle, WA, USA, November 2005. Springer.

[17] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proc. of the 8th Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 104–111, San Jose, CA, USA, June 1998.

[18] Krishna Puttaswamy Matthew Allen, Rama Alebooeyh and Ben Zhao. Chimera. `http://current.cs.ucsb.edu/projects/chimera/`, 2005.

[19] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of 1st Int. Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65, Cambridge, MA, USA, 2002. Springer.

[20] Ralph Niederberger. DEISA: Motivations, Strategies, Technologies. In *Proc. of the Int. Supercomputer Conference (ISC '04)*, Heidelberg, Germany, June 2004.

[21] Lucas Nussbaum and Olivier Richard. Lightweight emulation to study peer-to-peer systems. *Concurrency and Computation: Practice and Experience - Special Issue on HotP2P 06*, 2007.

[22] Benjamin Quétier, Thomas Hérault, and Frand Cappello. Virtual parallel machines throught virtualization: impact on mpi exections (poster abstract). In *Proc. of the 14th European PVM/MPI Users's Group meeting*, number 4757 in Lecture Notes in Computer Science, pages 381–384, Paris, France, 2007. Springer.

[23] Benjamin Quétier, Vincent Neri, and Franck Cappello. Scalability Comparison of Four Host Virtualization Tools. *Journal of Grid Computing*, 5:83–98, 2006.

[24] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *Proc. of the USENIX Annual Technical Conference 2004 on USENIX Annual Technical Conference*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[25] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[26] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. of the 18th IFIP/ACM Int. Conference on Distributed Systems Platforms (Middleware 2001)*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–250, Heidelberg, Germany, November 2001. Springer.

[27] Barnaby Livingston Stephen Naicken, Anirban Basu and Sethalat Rodhetbhai. A survey of peer-to-peer network simulators. In *Proc. of the 7th Annual Postgraduate Symposium (PGNet 2006)*, Liverpool, UK, June 2006.

[28] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the ACM 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM 2001)*, pages 149–160, San Diego, CA, August 2001.

[29] Aaron Swartz. Khashmir. `http://khashmir.sourceforge.net/`, 2004.

[30] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the 5th Symp. on Operating Systems Design and Implementation (OSDI)*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[31] Chuan Wu, Baochun Li, and Shuqiao Zhao. Magellan: Charting Large-Scale Peer-to-Peer Live Streaming Topologies. In *Proc. of the 27th Int. Conference on Distributed Computing Systems (ICDCS)*, page 62, Toronto, Ontario, Canada, june 2007. IEEE Computer Society.

[32] Ben Yanbin Zhao, John Kubiatowicz, and Anthony Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

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 scientifi que
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 0249-6399