Abstract—While the rapidly increasing number of users and applications running on Desktop Grid (DG) systems demonstrates its inherent potential, current DG implementations follow the traditional master-worker paradigm and DG middlewares do not cooperate. To extend the DG architecture, we propose a novel system, called BonjourGrid, capable of (1) creating, for each user, a specific execution environment in a decentralized fashion and (2) contrarily to classical DG, of orchestrating multiple and various instances of Desktop Grid middlewares. This will enable us to construct, on demand, specific execution environments (combinations of XtremWeb, Condor, Boinc middlewares). BonjourGrid is a software which aims to link a discovery service based on publish/subscribe protocol with the upper layer of a Desktop Grid middleware bridging the gap to meta-grid. Our experimental evaluation proves that BonjourGrid is robust and able to orchestrate more than 400 instances of XtremWeb middleware in a concurrent fashion on a 1000 host cluster. This experiment demonstrates the concept of BonjourGrid as well as its potential and shows that, comparing to a classical Desktop Grid with one central master, BonjourGrid suffers from an acceptable overhead that can be explained.

Index Terms—Desktop Grid, self-organization, Decentralized management, Publish/Subscribe, Collaborative computing.

I. INTRODUCTION

Desktop Grids have been successfully used to address large applications with significant computational requirements, including searching for extraterrestrial intelligence (SETI@Home [18]), global climate prediction (Climatprediction.net [16]), and cosmic rays study (XtremWeb [5]). While the success of these applications demonstrates the potential of Desktop Grid, existing systems are often centralized and suffer from their reliance on an administrative staff who guarantees that the master will remain up for as long as the computation is running. Moreover, although, the crash of the master is, in practice, infrequent and replication techniques can resolve this problem should it occur, we still believe in the need to decentralized approaches.

We base our belief on the fact that all institutions, for a variety of reasons (budget, strategic goals, etc.) will not necessarily guarantee the same high levels of equipments robustness. Hence, the community as a whole should be able to offer a higher level of reliability. Thus, we believe in the importance of collaborative, decentralized, and robust solutions.

The goal of this paper is to present BonjourGrid, a novel approach which enables the establishment of a specific execution environment for each user. Indeed, BonjourGrid constructs, dynamically and in a decentralized fashion, a Computing Element (CE — a CE is a set of workers managed by one master, or an instance of a local Desktop Grid middleware) when a user needs to run an application. BonjourGrid orchestrates multiple instances of a CE in a decentralized manner. Our approach helps by avoiding a reliance on a unique static central element in the whole system, since we use a dynamic temporary central element for each running application. Furthermore, it is important to note that BonjourGrid is not only a decentralized approach to the orchestration and coordination of local DG (Desktop Grids), but it is also a system which enables, contrarily to classical DG, the construction of an on-demand specific execution environment (based on any combination of XtremWeb, Boinc, Condor middlewares), in a decentralized, dynamic and autonomous fashion. As referenced in many other works [7], [8], this kind of environment is called Institutional Desktop Grid or Enterprise Desktop Grid (e.g., in the same institution).

The remainder of this paper is organized as follows. Section II presents an overview of the BonjourGrid system and outlines its principal components. Section III gives a detailed description of the design of BonjourGrid and the interaction between user, services, and resources. Section IV illustrates the implementation phase. Section V shows the experimental setup and analyzes the measurements obtained. In sections VI and VII, we discuss related works, summarize our contributions, and offer suggestions for future work.
II. Overview of BonjourGrid System

The principle of the proposed approach is to create, dynamically and in a decentralized way, a specific execution environment for each user to execute any type of applications without any system administrator intervention. An environment do not affect another one if it fails.

Let us give more details. Each user, behind a desktop machine in his office, can submit an application. It is important to note that BonjourGrid can handle bag of tasks and distributed applications with precedences between tasks. BonjourGrid deploys a master (coordinator), locally on the user machine, and requests for participants (workers). Negotiations to select them should now take place. Using a publish/subscribe infrastructure, each machine publishes its state (idle, worker or master) when changes occur as well as information about its load or its use cost, in order to provide useful metrics for the selection of participants. Under these assumptions, the master can select a subset of workers nodes according to a strategy that could balance the “power” of the node and the “price” of its use. The master and the set of selected workers build the Computing Element (CE) which will execute, manage and control the user application. When a CE finishes the application, its master becomes free, returns in idle state and releases all workers to return also to the idle state. When no application is submitted, all machines are in the idle state.

The key idea of BonjourGrid is to rely on existing Institutional Desktop Grid middlewares, and to orchestrate and coordinate multiple instances, i.e multiple CEs, through a publish/subscribe system (see figure 1). Each CE will be owned by the user who has started the master on his machine. Then this CE is responsible for the execution of one or many applications for the same user. As shown in figure 1, in the user level, a user A (resp. B) deploys his application on his machine and the execution seems to be local. Level 1 (middleware layer) shows that, actually, a CE with 4 (resp. 5) workers has been dynamically created, specifically for the user A (resp. B). Level 0 shows that all machines are interconnected and under the availability of any user.

To realize this approach, we compose BonjourGrid in three fundamental parts: a) A fully decentralized resources discovery layer, based on Bonjour protocol [1]; b) A CE, using DG middlewares such as XtremWeb (XW), Condor or Boinc, which executes and manages the various tasks of applications; c) A fully decentralized protocol of coordination between a) and b) to manage and control all resources, services and CEs. In the current work, we use XW as a computing system. Future works will deal with other systems such as BOINC and Condor, where will intend to take into account the security issue (for instance, applications signature for BOINC).

In a previous work [2], we tackled the following questions: Can a system based on publish/subscribe protocols be “powerful”? Can it be scalable? What is the response time to publish a service? What is the discovery time of a service? We carried out experiments on various protocols such as Bonjour protocol, on the GdX node of Grid5000 [15] platform using more than 300 machines (AMD Opterons connected with a 1Gb/s network). Measures show that Bonjour is reliable and very powerful in resources discovery. Indeed, Bonjour discovers more than 300 services published simultaneously in less than 2 seconds with 0% of loss. These evaluations support and justify our choice to use Bonjour as a basis for the discovery protocol in BonjourGrid system. Moreover, we believe that using an existing and tested protocol for the industry (Bonjour is available on Macintosh machines), to evaluate its usability in desktop grid environment is, in itself, of great value and very useful for the community.

III. Service oriented architecture for building a computing element

Each machine in BonjourGrid can have one of the three states (Idle, Worker or Coordinator). Each state is associated to a service: IdleService for idle machine, WorkerService for worker and CoordinatorService for coordinator. When a machine changes its state, it publishes the corresponding service to notify its new state after having deactivated the old one.

A. From idle to coordinator state

In the remainder, we assume that we have to coordinate XW instances. We suppose that each user keeps his machine connected to the desktop grid during the execution of his application in order to avoid managing the fault-tolerant problem between coordinators. The fault tolerance issue will be studied in a future work. A user submits his application through the local user interface of BonjourGrid (installed on his machine). The user machine changes, now, its state into the coordinator state in order to initiate the construction phase of a new CE. If this machine is a worker for another application,
BonjourGrid waits until it finishes the task in progress then launches the coordinator in order to not degrade the machine performance. This is a preliminary choice; the next version will support the migration of a job to another idle machine, if there is no idle machine, BonjourGrid informs the coordinator to put the job in break until it finds an idle machine or a worker finishes its task, so that the user will not wait for a long time, especially, if his machine runs a long job. The coordinator starts by running a discovery program (or Browser) on idle machines. The discovered machines are recorded in an IdleMachineDict dictionary. From this data structure, the coordinator selects machines which fit the application requirements and creates a new MyWorkersDict dictionary. The coordinator continues the research, i.e., its browser remains listening on idle machines, until the size of MyWorkerDict reaches the number of required machines (i.e., the size of the CE). Thereafter, the coordinator checks if selected workers accept to work for it. Indeed, the coordinator publishes for each selected worker, a new "RequestWorker" service using the "Worker Name" as service type. We note that the browser program listens on service type and not on service names. The confirmation of participation of the worker "A" to the "C" coordinator relies on the fact that "A" publishes a new "MyConfirmation" service putting as service type the name of the "C" coordinator. Only the coordinator "C" runs a browser program on services of type. Hence, if the browser discovers new registration of a service of type, it means that there is a new idle machine which has accepted to work for it. If confirmations number does not reach the size of the CE, the coordinator launches again the browser on idle machines, but now with the number of missing machines only.

It is possible that more than one coordinator select the same worker "A" and publish a service "RequestWorker" with the same service type (PTR: WorkerA)\(^1\). Thus, there may be contention in the access to the worker "A". BonjourGrid provides efficient and simple feedbacks on that. Indeed, an idle machine "A" confirms its participation to the first service "RequestWorker" discovered (i.e., the first coordinator), knowing that the protocol Bonjour guarantees that the browser does not discover more than one service at the same time.

In the middleware layer, BonjourGrid establishes the connection between XW-Coordinator and XW-Workers that have accepted to participate. Indeed, when an idle machine publishes the CoordinatorService, it launches also the XW-Coordinator. This XW-Coordinator remains listening to new connections of XW-workers. In the same way, when an idle machine browses the "RequestWorker" of a coordinator and publishes the service "MyConfirmation", it starts also the XW-Worker with the IP address of the coordinator (the address is stored in an attribute of the "RequestWorker" service). Since the first connection of an XW-Worker, the XW-Coordinator submits the first task of the application and remains ready to other connections from other XW-workers. If the coordinator does not find available idle machines, the application finishes its execution with only connected XW-Workers before reaching the required size of the CE. When the application finishes its execution, the coordinator deactivates its CoordinatorService service and stops XW-Coordinator. It returns into its initial state by publishing the IdleService service again. Figure 2 illustrates the steps of the protocol from idle state to coordinator state.

\(\text{Fig. 2. Steps to transform an idle machine into a coordinator}\)

\(\text{B. From idle to worker state}\)

Figure 3 presents states diagram to describe the steps taken by a host to change its state from idle to worker. Indeed, when a host joins BonjourGrid system, it is recorded with the initial Idle state. This host remains listening to notifications of coordinators which need its participation. When the machine discovers the participation request, it answers to this request by publishing a "MyConfirmation"
service as described above. Thereafter, the machine stops the browser, removes the IdleService service and publishes the WorkerService one to announce its new state: it is no longer free, now it becomes a worker. The worker runs the XW-Worker program with the IP address of the coordinator. The worker remains in possession of the coordinator while this one is alive. It runs a browser in order to listen to new events from its coordinator, especially its "death". When the coordinator is stopped, the worker removes the WorkerService and stops the XW-Worker process. Finally, the machine turns over to the initial state by publishing the IdleService again. In this way, the coordinator does not need to contact its workers to release them.

![Diagram](image.png)

**Fig. 3. Steps to transform an idle machine into a worker**

### IV. Implementation

BonjourGrid is entirely implemented using Python. We have used Bonjour-py package that offers a python interface to interact with Bonjour protocol. When a machine joins our system, information about its characteristics (i.e. MHZ, CPU, RAM, Hostname, IP address, average load) are automatically collected and stored in a Python dictionary. These information are useful for the selection of the suitable machines that match the application requirements. For instance, one would like to select workers according to the CPU metric only; another one would like to select workers that have been being idle for a long time. In our system, we can easily plug a new policy based on other selection criteria.

#### A. Running the XW services

To run or deactivate an XW service (coordinator or worker), the environment should be already installed. The installation procedure of a XW-Coordinator, in particular, is not currently simple enough, so we have improved it. We would not like to make user spend his time in configuring and installing the various files and modules necessary to the XW installation. An installation procedure consists in installing a MySQL server, Java Development Kit, creating a specific database for XW, making several directories and configuring system files. Consequently, we set up an automatic installation of all the necessary packages, as well as the part of configuration for the worker and the coordinator. Thus, our system checks if all the packages are installed, if not, the installation is done in an automatic way and without the intervention of the user. For the first use, the user has to launch a script with the super-user rights, to install the necessary modules. All the rest is realized in a transparent and automatic way. Such facilities were not included in the current distribution of XW.

#### B. Workload model

Evaluating a system with a set of specific applications does not necessarily demonstrate its real behavior. In the same way, using the arrival pattern of the tasks applying the Poisson's law does not reflect the reality of arrival models. Thus, in this work, we use the workload model [9] proposed by Feitelson which make it possible to generate a workload model very close to the reality. Indeed, we need a workload in which we can specify the system size (number of machines), the period of arrival time of jobs, the maximum number of parallel tasks per application and the maximum runtime of a task (see table I for an example of output results). Feitelson mentions that this workload is generated for rigid jobs. These are jobs that specify the number of processors they need, and run for a certain time using this number of processors. This matches our requirements well, since we need to construct several CE with different sizes. Then, an application is a set of parallel tasks. But, in order to detect the completion time of an application, we have inserted a fictitious task (a "gather" task) which starts its execution only after having received all the (fictitious) results of the preceding tasks. An application with \( k \) tasks means that \( k - 1 \) tasks are parallel and the \( k^{th} \) task is the gather task. The tasks of a given application, except the gather, make the same sleep time according to the runtime mentioned in the workload model, thereafter, they send a fictitious file to the gather.
TABLE I
A PART OF THE WORKLOAD OUTPUT

<table>
<thead>
<tr>
<th>Application ID</th>
<th>Arrival Time (in sec.)</th>
<th>Run Time (in sec.)</th>
<th>Number of // tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>11</td>
<td>128</td>
</tr>
<tr>
<td>3</td>
<td>69</td>
<td>13</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>75</td>
<td>59</td>
<td>93</td>
</tr>
<tr>
<td>5</td>
<td>78</td>
<td>127</td>
<td>160</td>
</tr>
</tbody>
</table>

C. A Generator of applications

We have developed a generator of applications in Python which takes as parameter the workload model and provides a set of applications (one application for each entry in the workload). Indeed, for each entry in the workload (see table I), the generator uses the configuration presented in the entry to create a compressed file containing the XML description of the dataflow graph. This file is used by XW to describe what are the parallel tasks and the different precedences between application tasks.

V. EXPERIMENTATION AND VALIDATION

We developed a program which takes as parameter 1) the workload model, 2) the path of the applications and 3) the list of machines which will be used as coordinators or workers. All machines are initialized to idle state. The program submits the applications following the arrival pattern times given in the workload. A submission of an application consists in selecting from the list, the first free machine on which a coordinator will be launched to start the CE construction, as well as the execution of the application. Each machine which changes its state from idle towards worker or coordinator will be locked; a temporary file /tmp/occupied is created to mention that it is occupied during the turnaround time of the application. The coordinator is deactivated automatically when the application is finished. The workers are released when the coordinator is deactivated. For the analysis of the obtained results, it is important to note that, especially for this experimentation, the database of the coordinator is erased each time it is deactivated, because the same machine may be used as a coordinator for another application, what may cause an overlapping in the detection of the application end. It is an implementation choice made for XW.

We carried out our first experiments on the Grid5000 platform using 128 machines (AMD opterons) on the local Orsay's node. With the application generator, we created 100 applications with different parallel tasks numbers. It varies from an application to another one from 2 to 128 tasks. We generated 2110 tasks for the 100 applications. The run time of a task varies from 1 to 150 seconds according to the workload model. The arrival pattern is set to a period of 3 hours.

This configuration is applied for BonjourGrid, which dynamically creates a CE for each application. The same configuration is used for the centralized XW model which uses only one central coordinator and 127 static workers to carry out the 100 applications. In XW, if an application has N parallel tasks and if there is only k free workers (k < N), the coordinator starts the execution using k workers, and waits for new coming or released workers. The same principle is applied in BonjourGrid. Then, the coordinator may finish all application tasks with k workers k < N if there is no sufficient free machines. This detail is important to understand measurements analysis.

Concerning the turnaround time of applications, we measure the difference between the date of ending of an application and its submission one. The overhead generated by BonjourGrid includes the waiting time to find a free machine (coordinator) plus the building time of a CE (connection of workers to the coordinator) plus the time to erase the database of the coordinator when it is released. On figure 4, the average overhead is about 2 min by application. The peaks in the curves are explained by the saturation of the system (i.e there is no more free machines). We notice that each peak has at least one previous application which uses more than 120 workers. For some applications (e.g. 51, 63, 68), BonjourGrid gives better turnaround times. This is explained by the fact that in XW: 1) The coordinator is overloaded and needs more time to allocate machines and to affect tasks; 2) It is possible that the master looses connections to the workers since it uses the same port to receive all heart beat signals emitted by them each 30 seconds to mark their presence. In contrast with XW, using BonjourGrid, a coordinator is responsible for only one application at the same time and listens only to the workers which carry out its application; 3) The coordinator assigns a task to a worker if this one sends a WorkRequest message (this concept is called a Back-off effect, it is useful to bypass the firewall). The worker sends this signal each 30 seconds. It is possible that the submission date is just one second after this request, and then the master is obliged to wait another WorkRequest signal to submit the application if there is not any free worker. This is a waste of time especially as the back-off time of XW (30 seconds) does not reflect the reality of existing Institutional Desktop Grid systems. It is too small, a real back-off is around 5 minutes at least. In contrast with XW, in BonjourGrid, the submission initiates the CE, i.e the coordinator is started with 0 workers but has already the list of tasks in its queue, thus, from the first connection, each worker catches a task.

We have realized a second experiment aiming to confirm our previous analysis. Indeed, we have added 20 machines (15%) to the 128 ones of the previous experiment. We would like to relax the saturation of the system by increasing the number of free machines. Figure 5 shows that the difference in end times, comparing to the curve of figure 4, decreased considerably. In the curve of figure 5, there is just one peak (because there is three previous applications which used more then 128 workers) with an
attenuation of 140 seconds. This improvement is explained by 1) there is more chance to find a free machine to launch a coordinator, hence, we reduce this difference between the application submission date and its CE building start time, and 2) because, as mentioned before, the back-off effect can delay the application tasks submission in XW, but not in BonjourGrid. This improvement confirms that the first setup with only 128 machines and 2110 tasks is a very stressful scenario for BonjourGrid.

Figure 5 confirms that if there is sufficient free machines, BonjourGrid gives an acceptable overhead. Yet, it is important to make more experiments over a large number of machines to check if BonjourGrid scale well or not, i.e, can it orchestrate a great number of CEs? In that respect, we performed out two other experiments using the virtual system Vgrid [17]. Vgrid provides a mechanism to create several virtual machines on the same host, then, the different virtual machines communicate through a virtual hub created on each physical machine (or real machine denoted by RM in the following). Indeed, when we mention that we have 500 virtual machines that means: X VM * Y RM; for instance 10 VM * 50 RM or 5 VM * 100 RM. We have used Vgrid to reach 1000 hosts.

Figure 6 illustrates that the use of virtual machines did not give good results. For that reason, we have picked out the execution of the 50 applications already executed on 128 real machines (RM) in order to observe the impact of virtual machines (VM) use. We point out that virtual machines lacked sufficient RAM to manage many opened sockets at the same time and to allocate necessary memory for the Java virtual machine. This explains the fact that: the increase of machines number from 128 real machines to 500 virtual ones (4 VM * 125 RM) did not enhance turnaround times. Indeed, in most cases, times reached on many virtual machines are worst than times obtained on less real ones, for both BonjourGrid and XW. Although, the use of VM had an impact on turnaround time, we intend to go on in experiments, since this will allow us to test whether BonjourGrid can manage a big number of CEs over a large grid composed of hundreds of nodes.

To this end, we have performed out an experimentation with 405 applications over 1000 virtual machines. We have launched 4 virtual machines per real machine (4 VM * 250 RM). The number of parallel tasks per application
varies from 2 to 128. The majority of applications has more than 32 parallel tasks and 20% have more than 60 parallel tasks. From figure 7, we deduce that BonjourGrid performs better around the 380th application. In fact, the XW-Coordinator becomes overloaded after this number of submitted applications and then it takes more time to update its database, to get results from workers and to submit new tasks. Moreover, it is important to mention that for XW evaluation, we have chosen to run a specific virtual machine with 1500 MB of RAM for the coordinator. However, the XW-coordinator does not succeed to exceed 500 connections of workers. This limitation is caused by the low performance of virtual machines. In contrast with XW, for BonjourGrid evaluation, we kept the same configuration for all virtual machines (300 MB per virtual machine) and then the coordinator in BonjourGrid does not have the same performance than the XW-coordinator. Thus, performance of virtual machines does not give the correct turnaround times of applications.

We remind that these two last experiments are performed out to show that BonjourGrid can scale and manage more than 400 CE with different sizes. The main features of BonjourGrid are the self-organization and the decentralization. The performance of BonjourGrid depends on the behavior of the computing system (i.e XW in the current experiment). In that respect, we plan to make other experiments with another computing systems such as BOINC or Condor to check the performance of BonjourGrid in that context.

VI. RELATED WORK

OurGrid [10] system avoids the centralized server by creating the notion of the home machine from which applications are submitted; the existence of several home machines reduces the impact of failures at the same time. Moreover, OurGrid provides an accounting model to assure a fair resources sharing in order to attract nodes to join the system. However, the originality of BonjourGrid comparing to OurGrid is that it supports distributed applications with precedence between tasks (since XW-CH handles this type of applications), while OurGrid supports only Bag-of-Tasks (BOT) applications (BOT applications are independent divisible tasks). WaveGrid [3] is a P2P middleware which uses a timezone-aware overlay network to indicate when hosts have a large block of idle time. This system reinforces the idea of BonjourGrid concept since changing from a set of workers to another one depending on the time zone (Wave Grid) is analogous to the principle of creating a CE from an application to another one in BonjourGrid, and depending on users requirements.

Approaches based on publish/subscribe systems to coordinate or decentralize desktop grid infrastructures are not very numerous according to our knowledge. The project which could resemble a little bit to our project is the Xgrid project [14]. In Xgrid system, each agent (or worker) makes itself available to a single controller. It receives computational tasks and returns the results of these computations to the controller. Hence, the whole architecture relies on a single and static component which is the controller. Moreover, Xgrid runs only on MacOS systems. In contrast with Xgrid, in BonjourGrid, the coordinator is not static and is created in a dynamic way. Furthermore, BonjourGrid is more generic since it is possible to graft it on any computing system (XW, BOINC, Condor) while Xgrid has its own computing system.

Let us note that our system BonjourGrid has been already used as a decentralized scheduler in [4]. It is essential to affirm that the approach design, the details about the discovery protocol and the BonjourGrid scalability study are described, in this paper, for the first time.

VII. CONCLUSION AND FUTURE WORK

In this work, we have proposed BonjourGrid, a novel approach designed to orchestrate multi-instances of computing elements in a decentralized manner. BonjourGrid dynamically creates a specific environment for the user who initiates applications without any need for a system administrator because BonjourGrid is fully autonomous
and decentralized. We have conducted several experiments to demonstrate the functionality of BonjourGrid and analyze the overhead induced by the decentralization. We conclude that BonjourGrid is extremely robust and well capable of managing more than 400 CEs in an interval of 3 hours, over 1000 nodes in the Grid5000 testbed.

Several issues must be taken into account in future work. The first is the fault-tolerant of the coordinator. Indeed, it is important to continue the execution of the application when the coordinator (user machine) fails (one instance would be its disconnection). The second issue is the reservation of participants: in the current version, BonjourGrid allocates available resources for a user without any reservation rules. Thus, if a user demands all the available machines for a long time, BonjourGrid grants them to that user. The third issue is in going up from a small to a wide area network. While the current version works only in a local network infrastructure, it is important to bypass this constraint. Grafting the new package of Bonjour, Wide Area Bonjour from Apple, seems to be a good solution to resolve this problem.

ACKNOWLEDGMENT

The authors would like to thank Benjamin Quetier for his technical assistance with Vgrid system used to conduct our experiments and Jean-Luc Gaudiot for reading this paper and for his valuable comments.

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.

This work is supported by a grant of Regional Council of Ile-de-France under the SETCI program.

REFERENCES