J Supercomput (2016) 72:1271–1290
DOI 10.1007/s11227-015-1564-z

A self-organized volunteer Cloud for e-Science

Walid Saad[1] · **Heithem Abbes**[2] · **Christophe Cérin**[2] · **Mohamed Jemni**[1]

Published online: 16 November 2015

Abstract Nowadays, the adoption of Cloud Computing platforms and Service Computing technologies are almost natural for the different e-Science communities. Cost benefits for data-intensive applications, ease of access, rich and varied offers for services are examples of positive returns by users. However, beyond this favorable welcome for the technology, some research problems remain and are still challenging. In this paper, we focus on the problems of automatically deploying IaaS for computing and for data management, using the SlapOS Cloud. The core of the system is a distributed protocol for orchestrating data and compute nodes. Using this interaction scheme, users are able to deploy, without any system administrator intervention, a PaaS inside the IaaS basically a Desktop Grid middleware. The aim of this paper is to demonstrate that the Desktop Grid and Cloud paradigms may merge and may be widely used by non-experts in the different areas of e-Science. We propose a fully self-organized volunteer Cloud for researchers where they can carry out e-Science experiments and process large amounts of data in a coherent way.

✉ Walid Saad
walid.saadd@gmail.com; walid.saad@utic.rnu.tn

Heithem Abbes
heithem.abbes@lipn.univ-paris13.fr

Christophe Cérin
christophe.cerin@lipn.univ-paris13.fr

Mohamed Jemni
mohamed.jemni@fst.rnu.tn

1 LaTICE, Université de Tunis, ESSTT 5, Av. Taha Hussein B.P. 56, Bab Mnara, Tunis, Tunisia

2 LIPN/CNRS UMR 7030, Université de Paris 13, 99 avenue Jean-Baptiste Clément, 93430 Villetaneuse, France

Keywords e-Science applications as a service · Cloud Computing · Volunteer Cloud · Desktop Grid · Emerging platform technologies · Self-configuration of applications

1 Introduction

Scientific computing in the Cloud is now a tendency for researchers. Thanks to the elasticity and high availability of resources, researchers are requested to prepare their applications and they no longer have to worry about resources for experimentation, failure or maintenance. In September 2013, Ian Foster and XSEDE Cloud Integration Investigation Team[1] published a study [16] where results indicate that the Cloud is the only immediate alternative for a researcher when academic institutions do not have sufficient in-house resources required to compute and store data. With cloud computing services, the scientific world does not need to invest in supercomputers, since the design and construction of such large-scale systems are extremely complex and very expensive. Thus, the classic functionalities of conventional grid computing will become accessible as a service and at lower cost.

Usually, the main applications and programming models that can be deployed on the Cloud are presented as Cloud-hosted SaaS. It reduces time to science by hiding platform complexities. Cloud services offer a solution for intensive computing and the management of large amounts of data. It provided some examples of applications and programming models that are good candidates to run in the Cloud:

- Big Data management: Archiving, analysis and sharing tools for structured data (SQL database) and unstructured data (MapReduce [5], Hadoop [28] and BlobSeer clusters [18]). Globus Online SaaS [9] is used to move massive amounts of data without requiring custom end-to-end systems.
- High-throughput scientific workflows: On-demand SaaS for data mining, BLAST searches, Monte Carlo simulations, MATLAB and R execution environments.
- Academic labs and teaching tools: Interactive Cloud-hosted tools that help researchers and educators to learn how to write HPC and big data applications.

Our contribution in this paper is to provide to scientific researchers a self-configurable volunteer Cloud on-demand. We provide e-Science users a wide access to any kind of HPC middleware (BOINC [4], Condor [6]) and to automate the full integration process into a single framework for big data applications.

This paper is a synthesis of several years of related works, which justifies high number of self-bibliographic citations. The experimental work considers the performance of the deployment sub-systems involving different SaaS and IaaS, it is also an experience in building *system-of-systems* in a methodic manner. Figure 1 shows our vision on merging Desktop Grid and Cloud technologies. On the top left side, we have the BonjourGrid [20,21,23] series of works and on the top right side our works on the SlapOS Cloud [22,25]. When we combine the two systems, we get the possibility to manage e-Science and Big Data applications as we can see in the bottom of Fig. 1. The

[1] https://www.xsede.org/.

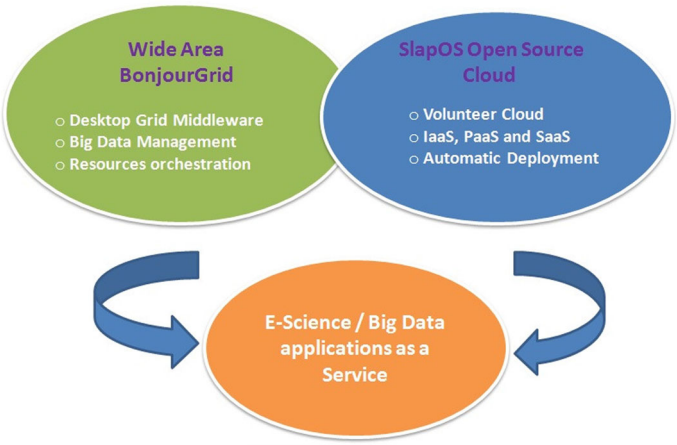

Fig. 1 Motivations: combining Desktop Grid and Cloud technologies

remainder of this paper is organized as follows. In Sect. 2, we outline related works by describing the background of high performance computing on Cloud environment. Section 3 presents the main concepts of our approach and the implementation of BonjourGrid into SlapOS. In Sect. 4, to demonstrate the effect of the implementation, we introduce experimental results. Finally, in Sect. 5, we summarize the contributions of the paper and we investigate future works.

2 Related work

In this section, we present scientific computing on Cloud. Furthermore, we exhibit our research area and previous work, in the context of Desktop Grid and Cloud.

2.1 Scientific computing on Clouds

In recent years, the rapid growth in big data and Cloud technologies have contributed to the emergence of high performance computing as a service (HPCaaS and HTCaaS). Research laboratories and enterprises are migrated to Cloud for offering computing and storage resources on demand according to the pay-as-you-use model. Given the diversity of applications and the growing need for computing, scientists have used a variety of public and private Cloud providers. The XSEDE survey report [16] shows that 58 % of researchers use the Amazon AWS Cloud followed by FutureGrid [10] (33 %), Windows Azure (30 %), Red Cloud, Google Cloud Platform, Globus Online [9] and Grid5000 testbed. Other Cloud providers (13 %) are mainly CloudSigma, Nimbix Accelerated Compute Cloud, Open Science Data Cloud [12], Open Science Grid [13], And Penguin On-Demand HPC Cloud Service (POD).[2]

[2] http://www.penguincomputing.com/.

Among the solutions we would like to give some details about Penguin, Open Science Data Cloud and FutureGrid to put them into perspective.

Penguin Computing on Demand (POD) delivers, for researchers who lack infrastructures or cluster management expertise, customized HPC solutions based on TORQUE scheduler [7] for large-scale simulations and Hadoop [28] clusters for big data application.

The Open Science Data Cloud (OSDC) [12] provides scientific researchers a set of services to store, share and analyze large- scale datasets. Researchers can deploy and share customized virtual machines with all tools for their research needs.

The FutureGrid [10] experimental platform allows scientists to develop and validate their associated research on optimization, integration and programming models for both Grid and Cloud Computing systems. FutureGrid provides a variety of resources based on three kinds of IaaS (Infrastructures as a Service) Nimbus, Eucalyptus and OpenStack. On top of this infrastructure, the platform offers researchers a range of middleware as a service such as UNICORE, Pegasus [14], ScaleMP and GenesisII.

2.2 Hybrid infrastructures for data-intensive science

The emergence of data-intensive application has prompted scientists around the world to investigate new challenges of the implementation of HPC platforms. With Cloud, the classical computing systems (Desktop and Data Grids) should be revisited. To enable big data processing and management on heterogeneous platforms, we can combine clouds and desktop grids providing hybrid infrastructures.

The work presented by Fedak et al. in [5,24] proposes an approach which aims to involve the behavior of Map-Reduce-based applications [27], to achieve scalable, concurrency-optimized, fault-tolerant data-intensive processing on clouds and desktop grids infrastructures. Their implementation combines, into a unified system, two different approaches, BlobSeer [17,18] and Bitdew [8] frameworks. Blobseer is a distributed data-store systems that is optimized for large unstructured blobs on Cloud infrastructures. Bitdew is suitable for large-scale data-intensive management and it can be grafted into Desktop Grid middleware.

While the common area is building hybrid infrastructures by combining Desktop Grid and Cloud technologies, our research direction is fundamentally different from these works. In fact, we attempt to provide e-Science users and researchers with fully volunteer infrastructures as a Service equipped with multiple Desktop Grid and data-intensive tools as a Service (SaaS) for performing experiments.

2.3 Our previous work

2.3.1 BonjourGrid meta-middleware

The BonjourGrid system [1,2] allows the creation of, dynamically and in a decentralized way, a specific execution environment (Condor [6], BOINC [4]) for each user to execute any type of applications without any system administrator intervention. Each user, behind a desktop machine in his office, can submit an application. Bon-

jourGrid deploys a master (coordinator), locally on the user machine, and requests for participants (workers).

Workers are selected according to specific criteria. The master and the set of selected workers build the Computing Element (CE) which will execute and manage the user application. When the execution of an application of a CE terminates, its master becomes free, returns in the idle state and it releases all workers before returning to the idle state too. Then, the nodes can participate to other projects. The coordination between masters is guaranteed using the publish/subscribe paradigm. There is no direct communication between masters, each one focuses on its applications and manages its workers (that actually execute the applications tasks).

The basic issue of BonjourGrid is going up to a wide area network. The first version works only in a local network infrastructure because of Bonjour implementation [26], so it is important to bypass this constraint. Grafting a new wide area package may be a good solution to resolve this problem.

2.3.2 Evolution of BonjourGrid

Since its first release in 2010, the BonjourGrid meta-middleware has never stopped evolving. There is a lot of development that have been added such as data management support and Cloud technologies convergence.

Data-intensive applications support In a recent work [20], we proposed a self-configurable Desktop Grids (DGs) platform on demand. We attempted to extend DGs middleware with the orchestration of different data management middleware and computing systems to deploy data-intensive applications. Basically, our research focuses on an extension of the BonjourGrid [1] meta-middleware for supporting data-intensive BoT/DAG (Bag of Tasks/Direct Acyclic Graph) applications by exploiting existing data management protocols and Grid middleware. However, since resources in a Desktop Grid are typically accessed through wide area network links, the bottleneck comes with the bandwidth limitation. One way to solve the problem is to build architectures that are able to mask (in part) the bandwidth limitation. In [21], we propose a decentralized approach for data prefetching for BoT and DAG applications to mask the bandwidth limitation and to overcome the disadvantages of the master–worker paradigm.

We note that our proposals systems presented in [20,21] have been validated experimentally using data-intensive jobs on Grid5000 [11] testbed.

Merging Desktop Grid and Cloud technologies Grid users want to manage data intensive in the same way as they perform other tasks on their computer. However, the traditional use of Desktop Grid requires to acquire a solid knowledge in the development of high performance computing (HPC), and big data management tools. Thus, we consider that the most important challenge is to imagine automatic data management tools that are able to mask the installation and configuration difficulties of data management software. In [22], we investigated works in the data management field and we made them "easy to use" for Grid users. We estimate that it is useful for them to manage data via the implementation of two Software as a Service tools for data

management. The first service provides a mean for users to transfer data from their sites to the computation or simulation sites. The second service will be used to share data in widely distributed environment.
Certainly, Desktop Grid is among the success stories during last years using volunteers nodes participating into projects. However, with the emergence of Cloud Computing, three questions can be posed (1) where to take resources? (2) how to coordinate the resources? and (3) how to manage big data transparently from end users? Our assumption is that Desktop Grid will continue to survive if we are able to transform the old-fashioned client/server architecture to new web-oriented architecture to deliver services on demand.

To answer the first two questions, in [23] we revisited and extended the coordination protocol of BonjourGrid. We dealt with the coordination of resources over internet using the publish-subscribe paradigm. We have focused on the usefulness of modeling and formal verification of such a specific paradigm, by studying the BonjourGrid system, which is dedicated to the management of multiple instances of Desktop Grid middleware that includes different computing elements and different data managers. We have proposed a formal modeling using colored Petri nets. We have also implemented, with Redis [19], a wide area new release of BonjourGrid in which multiple computing systems and data management frameworks are orchestrated in a transparent and decentralized manner.

Many experiments have been conducted attempting to evaluate the performance of systems presented in [22,23], using the BLAST[3] genomic application. BLAST is widely used in almost every large molecular biology laboratories for sequence alignment analysis.

Our contribution, in this paper is the integration of Wide Area BonjourGrid [23] in the SlapOS Cloud [25]. We aim to provide to scientific researchers a self-configurable volunteer Cloud on-demand. The usage scenario of a user is as follows: on the web portal (the web interface of SlapOS), the user selects the BonjourGrid system among the catalog of applications. Then, he chooses Desktop Grid middleware for computing and data management. The BonjourGrid system is deployed and calls for the deployment of middleware for computing and data management. In fact, we deploy an IaaS based on volunteer SaaS tools hence the name *self-configurable volunteer Cloud on-demand*.

2.4 SlapOS Cloud

2.4.1 Overview

Simple Language for Accounting and Provisioning Operating System (SlapOS [25]) is an open source-distributed operating system and it provides an environment for automating the deployment of applications, while including accounting and billing services through the ERP5 platform. Based on the idea that "Everything is process", SlapOS combines Grid computing, in particular the concepts inherited from BonjourGrid [1,2] and the techniques inherited from the field of ERP to manage, through the

[3] http://blast.ncbi.nlm.nih.gov/Blast.cgi.

SlapGrid daemon, IaaS, PaaS and SaaS Cloud services. The SlapOS strengths are the compatibility with any operating system, in particular GNU Linux, the compatibility with all software technologies, and support for several infrastructures (IaaS). More than 500 different recipes are available for consumer applications such as LAMP (Linux Apache MySQL PHP).

SlapOS is a distributed Cloud Computing system designed by Paris 13 University[4] and Nexedi. Marketed in Europe, Japan and China, it comprises deployment functions and automatic orchestration used in production by SANEF, Mitsubishi and Airbus Defence.

SlapOS architecture is composed of two types of components: SlapOS Master and SlapOS nodes. The Master tells to the SlapOS node which is the software that must be installed and also which instance of a specific software will be deployed. It acts as a centralized directory for all SlapOS nodes and it knows the location where software is located and all software that are installed [25].

We can view the topology of classical Clouds infrastructures as data centers to which are connected user machines. In these architectures, the computations are centered on a subset of machines (the data centers) among the possible ones. SlapOS [25] considers an alternative view of Clouds where both users machines and data centers are used for servicing requests. We refer to these Clouds as *volunteer Clouds*.

2.4.2 How to join SlapOS?

SlapOS is a voluntary Cloud, which mean that each person can potentially add its own server into the Cloud. The participation is simple and many platforms are supported by SlapOS. If a volunteer wants to join the SlapOS community, he has to:

1. Register on a SlapOS Master.
2. Install SlapOS Node on the node. For a full system installation, we can use the SlapOS image disk and to install SlapOS Node on existing operating system.
3. Add a virtual server on the Master and link it to the physical server by configuring the Node installed on the physical server. To simplify this process, the command *slapos node register COMPUTER_NAME* is implemented allowing to connect a physical server to a virtual server in SlapOS.
4. Select and install applications, from the list of available applications on the Master, that will be allowed to be deployed on the node.

At this moment, the Master knows that the node is a potential target to deploy instances of applications. The number of instances that can be run on the node depends on the capacity and the configuration of SlapOS on the server. If the maximum number of instances is reached, then the Master may no longer deploy instances on the node until one or more partitions become free. Note also that the volunteer must specify, every time he deploys a service, the server to be used by SlapOS Master, to avoid that the service be deployed anywhere in the Cloud.

To make our applications available on the SlapOS Master, it is necessary to integrate them to SlapOS. The integration of applications to SlapOS goes through the writing

[4] https://slapos.cloud.univ-paris13.fr.

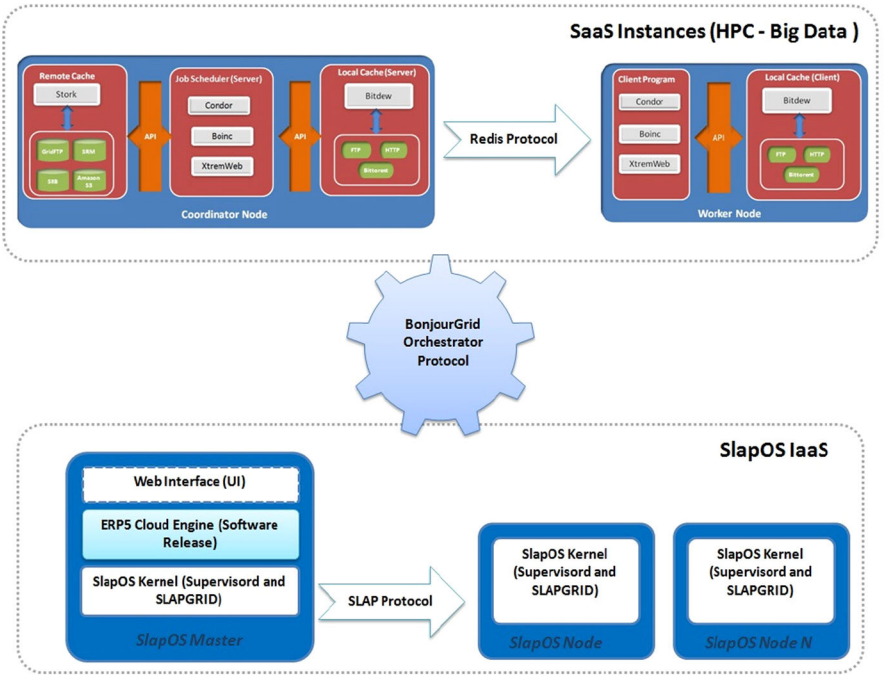

Fig. 2 Volunteer Cloud orchestration with BonjourGrid on the top of SlapOS

of Buildout profiles, consisting mainly of the file *software.cfg* which will then make reference to all other required files.

3 Volunteer Cloud orchestration with BonjourGrid

3.1 Our proposal architecture overview

A volunteer can join the BonjourGrid overlay network in an automatic way, we mean without the help of a system administrator. We use the SlapOS deployment system to make this task as simple as possible. We use also the Software Release and Software Instance concepts as explained previously. We suppose that BOINC, Condor, data managers and BonjourGrid (both masters and workers modules) are available as a single Software Release.[5,6] Hence, our implementations are based on (see Fig. 2):

– SlapOS IaaS: with only a "one-click" install, instantiate, configure a volunteer Cloud (data managers and Desktop Grid middleware) and deploy it over the Internet.

[5] http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/heads/grid-computing:/software/bonjourgrid/software.cfg.

[6] http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/heads/grid-computing:/software/bonjourgrid-client/software.cfg.

– BonjourGrid orchestrator protocol: it manages and coordinates multiple instances of HPC middleware (Boinc, Condor) and multiple data managers (Stork [15], Bitdew [8]).

We present below all steps that allow volunteer to participate in SlapOS community and exploit Desktop Grid services. He now has to do the following tasks:

1. *slapos-connect(Login, Password)*: an end user must connect to his space "My space" through a Web interface by giving login and password parameters.
2. *request-bonjourgrid-software(SlapOS_Node_Name, Software_Release_Name):* in the "My Servers" section, the user selects his server by choosing the computer ID in computer field, click on "Install New Software" button and select BonjourGrid (Master or Worker) and click on "Add Software". If the user wants to deploy the instance on a no dedicated node, he can keep the computer field empty and SlapOS would automatically choose an available computer.
3. *download-bonjourgrid-software(BonjourGrid_Software_Release_URL):* This will compile and install the BonjourGrid package (middleware and data managers). Once installation is completed, the user is allowed to make a deployment of BonjourGrid instances.
4. *request-instance-parameters(Slap_Parameters_List):* Through the "My Services" section, the user can choose the different parameters (which kind of middleware and the number of workers) of the instance or keep the default parameters, the computer on which he wants the instance be deployed.
5. *deploy-instance(Slap_Parameters_List):* At this stage, the BonjourGrid instance is being deployed. Using default parameters, the deployed instance contains the necessary packages for running DesktopGrid middleware (BOINC or Condor) and data managers (Stork and Bitdew).
6. *submit-application(submit_job_file, submit_data_file, coordinator_node, stork_server):* Once the instance is deployed, the user can submit application and monitor jobs.

3.2 BonjourGrid orchestrator protocol

Wide Area BonjourGrid is based on Redis publish-subscribe system [19,23], in which multiple computing systems and data management frameworks are orchestrated in a transparent and decentralized manner. Redis [19,23] includes a publish-subscribe object that subscribes to channels and listens for new messages. Once a publish- subscribe instance is created, clients can subscribe to one or more channels using the SUBSCRIBE (CHANNEL-NAME) command. Next, clients post (publish) a message to the given channel through the PUBLISH (CHANNEL-NAME, MESSAGE) command. Messages sent by clients to the channel will be pushed by Redis Server to all the subscribed clients. The following paragraphs provide more information on the resources coordination protocols of BonjourGrid.

In Wide Area BonjourGrid, when a machine joins the network, it takes the *idle* state. During the execution of an application, a machine changes status to become a *worker* or a *coordinator*. To build a computing element constituted by one coordinator and n workers, the resource coordination protocol is based on the following steps:

1. The coordinator machine starts the deployment process by executing the *Browse-IdleService('idle')* service, which allows to subscribe to *idle* channel;
2. Each machine that wants to participate in computation publishes the *idle* service by running *Publish-IdleService('idle')* function;
3. Once the *idle* services are discovered by the coordinator host, the *Request-Worker('WorkerName')* service will be published for each machine, *WorkerName* is the name of the Redis channel. If the idle machine confirms to participate in the computation, it must publish the *Confirm-Request('CoordName')* service, *Coord-Name* is the name of the coordinator Host. By doing so, machine changes state from *idle* to *worker*;
4. Repeat 1, 2 and 3 steps until building a computing element composed of n machines;
5. The coordinator starts the execution of the user application. To mark the end of the execution, worker must subscribe to the *AppIsCompleted* channel by performing the *Browse-Application('AppIsCompleted')* service;
6. One application is completed, the coordinator publishes the *Publish-Application ('AppIsCompleted')* service. At this stage, all machines will be released and the coordination protocol returns again to the first step.

For further details, the Algorithms 1 and 2 illustrate how coordinator and worker can work, respectively.

Given that BonjourGrid protocols will be established among wide area network, an important challenge is the consistency and synchronization between different published events. This was achieved by adding a specific methods to decrease the probability of losing a publication (resp subscription) events. For example, in the worker Algorithm 2 (line 12), the *os.waitpid(pid, 0)* function enables worker to wait for ending the parent process (*BrowseCoordinator()* function). This means that now the worker received from coordinator a request for participation in computation. As well, in line 21, the *os.waitpid(pid, 0)* function shows that a client program (Condor or BOINC) will be started if and only if the worker confirmed its participation in computation.

3.3 Desktop Grid middleware as a service

In this section, we review in detail the integration of BonjourGrid into the SlapOS Cloud. Basically, we are going to focus on all stages of the implementation and present in detail the deployment process.

3.3.1 The BOINC use case

BOINC is a platform for distributed computing. To use it, we create a project and then provide one or more applications, data files and configuration files to execute these applications on this platform. It is therefore necessary to create a *stack* that we have called BOINC and that will allow installing, configuring an execution environment for BOINC applications in SlapOS. Thus, the creation of a project and the integration of one or more applications to run is simply reduced to the definition of parameters

Algorithm 1: Wide Area BonjourGrid: Coordinator Node

Input: RedisServer, Wrapper, hostname, NbreOfWorkers
Output: NbreOfFetchedWorkers

```
1: data ← GenerateMachineFeatures();
2: redis ← ConnectToRedis(RedisServer);
3: IdleChanel ← (redis.pubsub()).subscribe('idle');
4: CoordinatorChanel ← (redis.pubsub()).subscribe(Hostname);
5: pid = os.fork();
6: if pid = 0 then
7:    print "Starting the coordinator (Condor Or BOINC)...";
8:    StartCoordinatorScheduler(Wrapper);
9: else
10:   NbreOfFetchedWorkers ← SearchWorkers(NbreOfWorkers, data,
    hostname, redis, IdleChanel, CoordinatorChanel);
11:   if NbreWorkers = NbreOfFetchedWorkers then
12:      Lucky: All requested machines are workers for your Master;
13:   else
14:      Unlucky: NbreWorkers − NbreOfFetchedWorkers, machines was reserved by another
    Master;
15:   end if
16:   while true do
17:      if CheckEndApplication() then
18:         Lucky: Application is finished. I will tell my workers to disconnect;
19:         PublishApplication('AppIsCompleted');
20:         exit();
21:      else
22:         time.sleep(5);
23:      end if
24:   end while
25: end if
```

allowing to customize our project and download useful files. BOINC is divided into two separate applications: BOINC server[7] and BOINC client.[8] To deploy BOINC Server, we have added to *slapos.cookbook* two recipes:

- The recipe *slapos.cookbook:boinc* that allows to deploy an empty BOINC project and is used directly in the stack.
- The recipe *slapos.cookbook:boinc-app* that allows to deploy an application in an existing BOINC instance. It is therefore possible to deploy multiple applications for the same project, by calling successively the recipe *slapos.cookbook:boinc-app* for different applications.

Also, we have added to the *slapos.cookbook* recipe an entry *slapos.cookbook:boinc-client* that allows to deploy an instance of a BOINC Client.

[7] http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/heads/grid-computing:/software/boinc/software.cfg.

[8] http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/heads/grid-computing:/software/boinc-client/software.cfg.

Algorithm 2: Wide Area BonjourGrid: Worker Node

Input: RedisServer, Wrapper, hostname
Output: MyCoordinator

```
1: data ← GenerateMachineFeatures();
2: redis ← ConnectToRedis(RedisServer);
3: IdleChanel ← (redis.pubsub()).subscribe('idle');
4: CoordinatorChanel ← (redis.pubsub()).subscribe(Hostname);
5: pid = os.fork();
6: if pid = 0 then
7:    print "Search master for working..."
8:    MyCoordinator ← BrowseCoordinator(hostname);
9: else
10:    print "STATUS=IDLE I'm Idle and I'm waiting for new calls";
11:    PublishIdleService(hostname, data,"idle");
12:    os.waitpid(pid, 0);
13:    print "I discovered a Coordinator Node, continue..."
14:    Print "The new master " MyCoordinator " calls me to work for him";
15:    pid = os.fork();
16:    if pid = 0 then
17:       print "Send confirmation to the master..."
18:       ConfirmRequest("idle", hostname, data);
19:       print "STATUS=WORKER I will work for this Master " MyCoordinator;
20:    else
21:       os.waitpid(pid, 0);
22:       pid = os.fork();
23:       if pid = 0 then
24:          print "Starting the worker (Condor Or BOINC)...";
25:          StartWorker(Wrapper);
26:       else
27:          print "I'm Waiting until application execution will be finished...";
28:          BrowseApplication('AppIsCompleted');
29:          print "Application is completed and " MyCoordinator " has gone";
30:          print "Stopping the worker...";
31:       end if
32:    end if
33: end if
```

3.3.2 The Condor use case

In this section, we explain the integration of Condor into SlapOS and the main differences with BOINC integration. Condor installation can operate in three modes: *manager*, *execute* and *submit*. We have defined two types of instance: The Condor Master and the Condor Worker. The condor master represents a coordinator node it allows to manage and submit jobs, condor worker represents a compute node. So to have our condor master instance, we specify the types manage, submit and for the worker instance we specify the type execute. We implemented a new component called *condor* which install binaries allowing to execute both master and worker, depending on whether we specify a master or worker instance.

We also have a software release called *condor* which allows to install condor with all dependencies and, according to the parameters provided, deploys a condor master or condor worker instance. This software release also allows to submit jobs, the sub-

mission is accepted if and only if the instance is deployed as a master. The component contains the Buildout profile used to install binaries for condor and all required dependency. The software release contains profiles for installing templates and necessary files for the instance configuration, it mainly consists of two files: *software.cfg*[9] and *instance-condor.cfg*.[10]

- The *software.cfg*, used during the compilation will allow to run the condor component profile and preparing all the elements needed to deploy a condor instance.
- The file *instance-condor.cfg* is used for the deployment of a condor instance. In this file, the recipe *slapos.cookbook:condor* is used to deploy a master or a worker instance. The submission of a job is done by the *slapos.cookbook:condor.submit recipe*.

4 Experimental results

The experiments have been performed on Grid5000 testbed using 55 machines of the Lyon and Luxembourg sites' machines. We set two Debian Linux Distribution images of SlapOS as already mentioned and depicted in Fig. 2. In the first image, we deploy SlapOS Master (SlapOS Kernel, Supervisor, SlapGrid and ERP 5 Engine). The second image contains the kernel of SlapOS Node (SlapOS Kernel, Supervisor and SlapGrid), and it will be used to deploy, on demand BonjourGrid SaaS instances. In this section, we present the different steps for configuring and deploying BonjourGrid on Grid5000. Next, we explain the experimentation scenario, present results and interpretations.

4.1 Deployment of BonjourGrid orchestrator on Grid5000

To integrate SlapOS in Grid5000, we met several technical difficulties, the most important are IPv6 configuration and Grid5000 access restriction. SlapOS IaaS deployment steps on Grid5000 are already presented in this work [22].

We now focus on how to easily deploy BonjourGrid in Grid5000 through SlapOS. The user has only to prepare parameters and applications to run and then request BonjourGrid services using SlapOS command lines. Because SlapOS is optimized for high performances, we are able to run about 20 competing BonjourGrid services on the same Grid5000 node. This helps to reach about 1000 BonjourGrid services if we deploy 50 Grid5000 nodes with Slapos image.

As shown in the Sect. 3.1, to deploy BonjourGrid software in SlapOS, the user needs to complete the following steps using the command line or through the Web interface:

- Supply the software: this helps to prepare BonjourGrid components on the node.
- Request instances: once the software and recipes are installed, SlapOS can now request instances into the node's partitions.

[9] http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/heads/grid-computing:/software/condor/software.cfg.

[10] http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/heads/grid-computing:/software/condor/instance-condor.cfg.

SlapOS supplies software by compiling it from sources (if necessary or upon request) using Buildout. As the compiling step can take a while, we decided to simplify this step by providing a pre-built application for Grid5000. This helps us to reduce significantly the software supplying time and to focus on the deployment of instances. For example, the next scripts will create a BonjourGrid computing element with one coordinator and 10 workers instances. To do this, the user must provide the software URL (`BONJOURGRID_SR_URL`) and the slapOS node ID (`COMPUTER_ID`) as parameters. The user can request instances through the Web interface of the SlapOS master.

```
BONJOURGRID_SR_URL = "http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/
heads/grid-computing:/software/bonjourgrid/software.cfg"
COMPUTER_ID = $(cat /etc/opt/slapos/slapos.cfg | egrep computer_id.*= |
 awk '{print $3}')
MIDDLEWARE = boinc | condor
WORKERS = 10
APPLICATION = boinc-app-list | condor-app-list
slapos supply $BONJOURGRID_SR_URL $COMPUTER_ID
slapos request "My First BonjourGrid Coordinator Instance"
"$BONJOURGRID_SR_UR"  --node "computer_guid=$COMPUTER_ID"
--parameters "worker-number=$WORKERS" "project-type=$MIDDLEWARE"
"application=$APPLICATION"
```

The next command will request a BonjourGrid worker instance and will attach it to the BonjourGrid Coordinator instance.

```
BONJOURGRID_SR_URL= "http://git.erp5.org/gitweb/slapos.git/blob_plain/refs/
heads/grid-computing:/software/bonjourgrid-client/software.cfg"
COMPUTER_ID = $(cat /etc/opt/slapos/slapos.cfg | egrep computer_id.*= |
 awk '{print $3}')
REDIS_SERVER = ipv6_address
PROJECT_URL = boinc_project_url | condor_manager_address
slapos supply $BONJOURGRID_SR_URL $COMPUTER_ID
slapos request "My First BonjourGrid Worker Instance"
"$BONJOURGRID_SR_UR"  --node "computer_guid=$COMPUTER_ID"
 --parameters "redis-ip=$REDIS_SERVER" "project-url=$PROJECT_URL"
```

We have the possibility to run again the above command to get the service status. If the status is `Started`, this command also returns the connection parameters defined in the deployment recipe.

4.2 Usage scenario

Users may connect to the same SlapOS master and may demand a hundred of computing element using BonjourGrid instances. The SlapOS master is deployed in Lyon site and SlapOS nodes are distributed over 6 clusters located in the Lyon (sagittaire, orion, hercule, taurus) and Luxembourg (petitprince, granduc) sites. For each SlapOS node, BonjourGrid deploys one computing element: a coordinator (BOINC server,

Table 1 Workload used for the experimentation

Testbed information	Workload information
Number of SlapOS Nodes: 47 machines of Lyon Site Debian Wheezy 86_64 3.2.0-4-amd64 (clusters: sagittaire 2 CPU and 2 GB RAM, orion, hercule, taurus)	Number of BonjourGrid coordinator per node: 1 instance (2 computers partitions)
8 machines of Luxembourg Site Debian Wheezy 86_64 3.2.0-4-amd64 (clusters: petitprince, granduc)	Number of BonjourGrid worker instance per node: 18 instances (18 computers partitions)
One SlapOS Master : deployed on Lyon Site	Total number of instances: $(1 + 18) * 55 = 1045$ distributed over 1100 computers partitions
Total number of computer partition per Node: 20	
Total number of computer partition: $55 * 20 = 1100$	

MariaDB, Redis server and BonjourGrid protocol) with a set of workers (BOINC client and BonjourGrid protocol). This architecture is more general than the one used in [22] because the number of clusters is larger (wide area network) and we offer IaaS rather than SaaS, making the scenario realistic.

Thus, we decided to measure the time required to attach SlapOS nodes to the master and the time needed to create coordinators and workers instances as a function of the number of SlapOS nodes. Table 1 presents the workload and the system information of the test. We used 55 SlapOS nodes (20 computers partitions per node) to deploy 1045 instances of BonjourGrid (19 instances per node). Algorithms 3, 4 and 5 show all the experimental steps.

Algorithm 3: Scenarios used for the Grid5000 experiments (Part 1)

Input: Node List File (Lyon and Luxembourg sites)
Output: Times to register SlapOS Nodes
1: Deploy SlapOS Master Image: Kadeploy3 -e slapos-vifib..
2: *nodes* ← 55;
3: **do in parallel**
4: **for** *i* ← 1 **to** *nodes* **do**
5: Deploy SlapOS Node Image: Kadeploy3 -e slapos-bonjourgrid-cloud..;
6: Register and configure SlapOS Nodes with 20 computer partitions: slapos node register..;
7: **end for**
8: **end parallel**

4.3 Results analysis

4.3.1 SlapOS nodes registration time

We compute how long it takes on average for new node to register with a master. We note that the master is running in Lyon site and it is configured by default to register

Algorithm 4: Scenarios used for the Grid5000 experiments (Part 2)

Input: Node List File (Lyon and Luxembourg sites)
Output: Times create BonjourGrid coordinators instances
1: $nodes \leftarrow 55$;
2: **do in parallel**
3: **for** $i \leftarrow 1$ **to** $nodes$ **do**
4: Configure BonjourGrid Coordinator software release : slapos supply bonjourgrid-coordinator..;
5: Request one BonjourGrid Coordinator instance: slapos request "Coordinator instance i"..;
6: **end for**
7: **end parallel**

Algorithm 5: Scenarios used for the Grid5000 experiments (Part 3)

Input: Node List File (Lyon and Luxembourg sites)
Output: Times to create BonjourGrid workers instances
1: $nodes \leftarrow 55$;
2: $workers \leftarrow 18$;
3: **do in parallel**
4: **for** $i \leftarrow 1$ **to** $nodes$ **do**
5: Configure BonjourGrid Worker software release : slapos supply bonjourgrid-worker..;
6: **for** $i \leftarrow 1$ **to** $workers$ **do**
7: Request one BonjourGrid instance: slapos request "Worker instance i"..;
8: **end for**
9: **end for**
10: **end parallel**

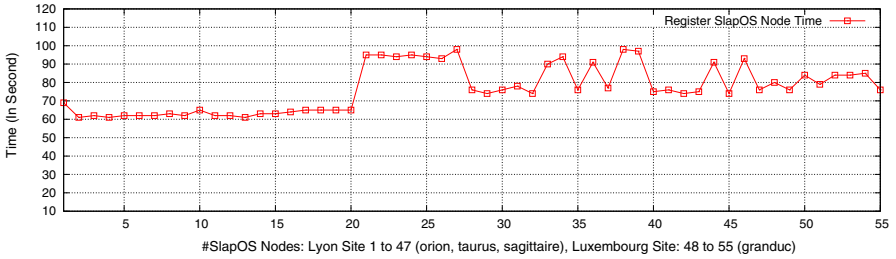

Fig. 3 Elapsed time to attach SlapOS Nodes (Lyon and Luxembourg sites) to the SlapOS Master running on Lyon site

one node at a time. Node executes *slapos node register* command to retrieve key and certificate from the master and to create 20 computer partitions. Results in Fig. 3 show that the register time ranged between [61..100] s. Indeed, the measures depend on hardware characteristics of each cluster. For instance, Orion, Taurus and Hercules clusters (Lyon Site, Nodes 1–20 with 12 CPUs and 32 GB of memory) give the best recording time (with a stable average value equal to 62 s). In contrast, Sagittaire cluster (Nodes 21–47 with 2 CPUs and 2 GB of memory) presents the highest record values. As seen in curves (Fig. 3), the recording time increases considerably (ex. Node 21) and is not linear. Otherwise, the Granduc cluster (Luxembourg Site with 8 CPUs and 32 GB of memory), nodes 48 to 55 provide more stable measurements relative to Sagittaire despite the long distance between the two sites.

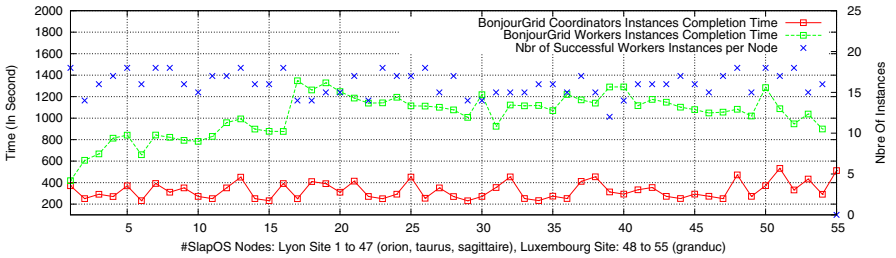

Fig. 4 Performance results of BonjourGrid with SlapOS over Grid5000

4.3.2 BonjourGrid completion time

We calculate, as discussed in Sect. 4.1 and in Algorithms 4 and 5, the time required to create BonjourGrid coordinators and worker instances:

- Coordinators instances: The total completion time for the BonjourGrid coordinator instances is shown in Fig. 4 (red curves). All instances are launched simultaneously and completed successfully. The average amount of time taken for the instances to complete is equal to 300 s. This time illustrates the real time for the initialization of different daemons on BonjourGrid coordinator, orchestration protocol (Redis server), computing middleware (BOINC server, MariaDB server) and data management protocols (Stork and Bitdew caches).
- Workers instances: In this use case, we have requested 990 workers instances (18 instances per node). The number of instances sent to the SlapOS Master is the main parameter impacting the total completion time. Results are presented in Fig. 4. For each node, the deployment of workers instances (Green curve) is calculated as the difference between the submission time of the first instance and the finish time of the last instance (number 18). We measured total time in the range of 418 and 1348 s. We observe that the turnaround time increases as the number of concurrent requests (respectively, the nodes number) increases. This is explained by the fact that in SlapOS the master is unreachable and needs more time to deploy instances and to receive subsequent requests. To explain this behavior, the Y2 axis (Blue curves of Fig. 4) gives the number of correctly deployed instances per node. Results show that 834 requests (85 %) have been completed of a total of 990. Also, 38.5 % of failed instances are in the Luxembourg site (e.g., no instances were deployed on the machine number 55). Here, we believe that the SlapOs Master looses connections to the nodes since it uses the same TCP port to receive all heart beat signals emitted by nodes: every 5 min, the node asks the list of software that must be installed and requests the list of computer partitions to reconfigure.

5 Conclusion and future works

In this paper, we have demonstrated how to cloudify the Desktop Grid paradigm as presented in [23], into the SlapOS Cloud infrastructure to form a IaaS. We introduced the new volunteer Cloud concept that helps scientist to deploy easily their distributed

environment as a service for e-Sciences requirements. The users can now request and install automatically Desktop Grid middleware (BOINC and Condor) and data management tools such as Stork and Bitdew without any intervention of a system administrator. Our contribution is on the way we manage PaaS. From a user point view, we offer the same functionalities of classical PaaS including Hadoop, Spark and integration services (AWS) but in an autonomous and configurable fashions. For example, the pair BOINC and Stork may play the role of Hadoop and BonjourGrid orchestrator as the role of AWS.

Also, through a large-scale experimentation, we have evaluated our volunteer Cloud by creating 1100 BonjourGrid services using more than 50 machines of Grid5000. We noticed that the SlapOS master may be overloaded when the number of nodes connected to the Cloud increases at the very initial steps of the node attachment. Certainly, this overload does not depend on the number of instances because SlapGrid runs in loop to initiate failed requests. Thus, further experimental investigations are needed to estimate communication and traffic bottlenecks between nodes at these times. At this time, we are not convinced that a simple adjustment of the value of a watchdog in SlapOS or BonjourGrid will solve the problem of scaling with SlapOS and Desktop Grids. We think that the reason is more profound and related to the ways we capture, modeling and effectively solving with relevant software interfaces the problems of the deployment of *systems-of-systems*.

In [3], authors introduce their work towards migration and adoption of cloud computing technologies at Université Sorbonne Paris Cité (USPC), a major french consortium of universities, higher education and research institutes. The paper is divided into two parts. In the first part, the paper provides with a methodology to identify the business processes of the members of the IDV Life Imaging USPC project as well as the tools and volume of data they currently use. In the second part of the paper, authors explain how the IDV members migrate to cloud computing by integrating the identified tools into virtual machines. USPC is currently finalizing the investments for an OpenNebula cloud infrastructure with 1.2PB of storage capacity and 1000 virtual machines for an inaugural opening session in November 2015.

The USPC context is an opportunity for practical usage of our system. We plan to integrate our work in the IDV project where the needs for collecting and analyzing images from different hospitals is a major issue. The goal is to facilitate the deposit, by medical doctors or engineers, of images in different formats and to allow people to use annotation tools on images. Since the work in [3] has identified the business processes of people, it will be more easy now to tailor our work to match the needs of the IDV users of the cloud. From a scientific point of view, at this occasion, we are also planning a coupling of our work with the Active-Data framework [24] to capture the data life-cycle in a better way than today.

References

1. Abbes H, Cérin C, Jemni M (2009) Bonjourgrid: orchestration of multi-instances of grid middlewares on institutional desktop grids. In: 23rd IEEE international symposium on parallel and distributed processing, IPDPS, pp 1–8

2. Abbes H, Cérin C, Jemni M, Saad W (2010) Fault tolerance based on the publish-subscribe paradigm for the bonjourgrid middleware. In: Proceedings of the 11th IEEE/ACM international conference on grid computing GRID, pp 57–64
3. Abidi L, Cérin C, Geldwerth-Feniger D, Lafaille M (2015) Cloud computing for e-sciences at université sorbonne paris cité. In: Cloudway workshop, service oriented and cloud computing—4th European conference, ESOCC 2015, Taormina, Italy, 15–17 September 2015, Proceedings of Springer CCIS series. vol 567
4. Anderson DP (2004) BOINC: a system for public-resource computing and storage. In: 5th international workshop on grid computing GRID, pp 4–10
5. Antoniu G, Costan A, Bigot J, Desprez F, Fedak G, Gault S, Pérez C, Simonet A, Tang B, Blanchet C, Terreux R, Bougé L, Briant F, Cappello F, Keahey K, Nicolae B, Suter F (2013) Scalable data management for map-reduce-based data-intensive applications: a view for cloud and hybrid infrastructures. IJCC 2(2/3):150–170
6. Butt AR, Zhang R, Hu YC (2006) A self-organizing flock of condors. J Parallel Distrib Comput 66(1):145–161
7. Chlumsky V, Klusácek D, Ruda M (2012) The extension of torque scheduler allowing the use of planning and optimization in grids. Comput Sci (AGH) 13(2):5–20
8. Fedak G, He H, Cappello F (2009) BitDew: a data management and distribution service with multi-protocol file transfer and metadata abstraction. J Netw Comput Appl 32(5):961–975
9. Foster I (2011) Globus online: accelerating and democratizing science through cloud-based services. IEEE Internet Comput 15(3):70–73
10. Fox G, von Laszewski G, Diaz J, Keahey K, Fortes J, Figueiredo R, Smallen S, Smith W, Grimshaw A (2013) Futuregrid: a reconfigurable testbed for cloud, hpc, and grid computing. In: Contemporary high performance computing: from petascale toward exascale, CRC computational science. Chapman & Hall
11. Grid'5000 (2015) Large-scale and Grid experimental testbed. http://www.grid5000.fr. Accessed 30 June 2015
12. Grossman RL, Greenway M, Heath AP, Powell R, Suarez RD, Wells W, White KP, Atkinson MP, Klampanos IA, Alvarez HL, Harvey C, Mambretti J (2012) The design of a community science cloud: the open science data cloud perspective. In: SC companion: high performance computing. Networking storage and analysis, Salt Lake City, UT, USA, pp 1051–1057
13. Juve G, Rynge M, Deelman E, Vöckler J, Berriman GB (2013) Comparing futuregrid, amazon ec2, and open science grid for scientific workflows. Comput Sci Eng 15(4):20–29
14. Kang U, Chau DH, Faloutsos C (2012) Pegasus: Mining billion-scale graphs in the cloud. In: 2012 IEEE international conference on acoustics, speech and signal processing, ICASSP 2012, Kyoto, Japan, 25–30 March 2012, pp 5341–5344
15. Kosar T, Livny M (2004) Stork: making data placement a first class citizen in the grid. In: 24th international conference on distributed computing systems ICDCS, pp 342–349
16. Lifka D, Foster I, Mehringer S, Parashar M, Redfern P, Stewart C, Tuecke S (2013) XSEDE cloud survey report. Technical report, XSEDE Cloud Integration Investigation Team. http://www.cac.cornell.edu/technologies/XSEDECloudSurveyReport.pdf. Accessed 22 Oct 2014
17. Nicolae B (2010) BlobSeer: towards efficient data storage management for large-scale, distributed systems. Theses, Université Rennes1
18. Nicolae B, Antoniu G, Bougé L, Moise D, Carpen-Amarie A (2011) Blobseer: next-generation data management for large scale infrastructures. J Parallel Distrib Comput 71(2):169–184
19. Redis Protocol (2015) Open source (BSD licensed) publish/subscribe and in-memory data structure store. http://redis.io/. Accessed 15 Feb 2015
20. Saad W, Abbes H, Cérin C, Jemni M (2012) A self-configurable desktop grid system on-demand. In: Proceedings of the 2012 seventh international conference on P2P, parallel, grid, cloud and internet computing 3PGCIC, pp 196–203
21. Saad W, Abbes H, Cérin C, Jemni M (2013) A data prefetching model for desktop grids and the condor use case. In: Proceedings of the 2013 12th IEEE international conference on trust, security and privacy in computing and communications TRUSTCOM '13, pp 1065–1072
22. Saad W, Abbes H, Cérin C, Jemni M (2014a) Designing and implementing a cloud-hosted saas for data movement and sharing with slapos. Int J Big Data Intell IJBDI 1(1/2):18–35

23. Saad W, Abidi L, Abbes H, Cérin C, Jemni M (2014b) Wide area bonjourgrid as a data desktop grid: Modeling and implementation on top of redis. In: 26th IEEE international symposium on computer architecture and high performance computing, SBAC-PAD, pp 286–293
24. Simonet A, Chard K, Fedak G, Foster IT (2015) Using active data to provide smart data surveillance to e-science users. In: 23rd Euromicro international conference on parallel, distributed, and Network-Based Processing, PDP 2015, Turku, Finland, 4–6 March 2015, pp 269–273
25. Smets-Solanes JP, Cérin C, Courteaud R (2011) Slapos: A multi-purpose distributed cloud operating system based on an erp billing model. In: IEEE international conference on services computing SCC. IEEE, pp 765–766
26. Steinberg D, Cheshire S (2005) Zero configuration networking: the definitive guide, 1st edn. O'Reilly Media, Inc., Sebastopol
27. Tang B, Moca M, Chevalier S, He H, Fedak G (2010) Towards mapreduce for desktop grid computing. In: 3PGCIC 2010, international conference on P2P, parallel, grid, cloud and internet computing, 4–6 Nov 2010. Fukuoka Institute of Technology, Fukuoka, pp 193–200
28. White T (2015) Hadoop–the definitive guide: storage and analysis at internet scale, 4th edn. O'Reilly, Sebastopol (revised and updated)