Deploying Large-Scale Data Sets on-Demand in the Cloud: Treats and Tricks on Data Distribution.

Luis M. Vaquero[1], *Member, IEEE*
Antonio Celorio[1]
Felix Cuadrado[2], *Member, IEEE*
Ruben Cuevas[3]

[1]Hewlett-Packard Laboratories.
Security and Cloud Lab. Bristol, BS34 8QZ, UK.
[2]Queen Mary University of London.
School of Electronic Engineering and Computer Science. London, E1 4NS, UK.
[3]Universidad Carlos III of Madrid
Department of Telematics. Leganes, 28911, Spain.

Public clouds have democratised the access to analytics for virtually any institution in the world. Virtual Machines (VMs) can be provisioned on demand, and be used to crunch data after uploading into the VMs. While this task is trivial for a few tens of VMs, it becomes increasingly complex and time consuming when the scale grows to hundreds or thousands of VMs crunching tens or hundreds of TB. Moreover, the elapsed time comes at a price: the cost of provisioning VMs in the cloud and keeping them waiting to load the data. In this paper we present a big data provisioning service that incorporates hierarchical and peer-to-peer data distribution techniques to speed-up data loading into the VMs used for data processing. The system dynamically mutates the sources of the data for the VMs to speed-up data loading. We tested this solution with 1000 VMs and 100 TB of data, reducing time by at least 30 % over current state of the art techniques. This dynamic topology mechanism is tightly coupled with classic declarative machine configuration techniques (the system takes a single high-level declarative configuration file and configures both software and data loading). Together, these two techniques simplify the deployment of big data in the cloud for end users who may not be experts in infrastructure management.

Index Terms—**Large-scale data transfer, flash crowd, big data, BitTorrent, p2p overlay, provisioning, big data distribution**

I. Introduction

PROCESSING large data sets has become crucial in research and business environments. Practitioners demand tools to quickly process increasingly larger amounts of data and businesses demand new solutions for data warehousing and business intelligence.

Big data processing engines have experienced a huge growth. One of the main challenges associated with processing large data sets is the vast infrastructure required to store and process the data. Coping with the forecast peak workloads would demand large up-front investments in infrastructure. Cloud computing presents the possibility of having a large-scale on demand infrastructure that accommodates to varying workloads.

Traditionally, the main technique for data crunching was to move the data to the computational nodes, which were shared [1]. The scale of today's datasets have reverted this trend, and led to move the computation to the location where data are stored [2]. This strategy is followed by popular MapReduce [3] implementations (e.g. Hadoop).

These systems assume that data is available at the machines that will process it, as data is stored in a distributed file system such as GFS [4], or HDFS. This situation is no longer true for big data deployments on the cloud. Newly provisioned virtual machines (VMs) need to contain the data that will be processed.

Loughran et al. exposed how different flavours of big data processing applications could easily be provisioned in the cloud using automated deployment and configuration tools. In this approach, the computing software and data are deployed on-the-fly over the provisioned VMs. For instance, deploying a Hadoop cluster on demand requires installing Hadoop, configuring and populating the Hadoop Distributed File System (HDFS) with the data to be crunched [5]. In contrast, on a physical cluster Hadoop is already predeployed and the data has been populated beforehand.

The task of populating freshly deployed VMs with large chunks of data may look trivial for small data sets, where a simple sequential copy from a central repository to the newly deployed VMs works well [6]. As data size grows (and therefore the number of VMs to process it), data transfer becomes one key performance bottleneck even within the same data centre. Populating the newly deployed VMs may take prohibitively long (1 TB of data sequentially loaded to 100 VMs on a 10 Gb ethernet network may take 1500 min[1]). Sometimes this time may be 3-4 orders of magnitude longer than the time it takes to make the actual computations (tens of minutes to crunch data that takes days to load into the VMs).

Manuscript received March 1, 2014; revised . Corresponding author: LM Vaquero (email: luis DOT vaquero AT hp.com).

[1]This number can be seen as a minimum boundary, since the performance of VMs can significantly vary due to their lack of I/O isolation, which makes data transfers susceptible to the interference of "noisy neighbours"

Big Data exposed as a Service (BDaaS) on top of an Infrastructure as a Service (IaaS) cloud is, thus, a complex environment where data distribution plays a crucial role in the overall performance.

In this paper we propose a solution based on combining hierarchical and Peer to Peer (P2P) data distribution techniques for significantly reducing the system setup time on an on-demand cloud. Our technique couples dynamic topology to speed-up transfer times with software configuration management tools to ease the quality of experience for the end users. As a result, we significantly decrease the setup time (VM creation, software configuration and VM population with data) of virtual clusters for data processing in the cloud.

The paper is structured as follows. Section II presents the main steps required to enable BDaaS in the cloud. These steps are enforced by an architecture (see Section III) whose implementation is shown in Section IV. Section V presents an evaluation of the proposed method and implementation. Our method and system are compared to other relevant works in Section VI. Finally, Sections VII and VIII discuss future work and present the main conclusions of this work.

II. Big Data as a Service in the Cloud

IaaS cloud providers offer computation and storage resources to third parties [7]. These capabilities are exposed by means of an imperative API that allows customers to deploy VMs based on predefined virtual images, as well as persistent storage services. In addition to the fundamental functionality of providing computing and storage as a service, providers offer an increasing catalogue of services (e.g. authentication, and key management), although these do not provide functional building blocks for setting up a generic Big Data Processing infrastructure. Some attempts have been done at setting up Hadoop in the cloud (see [5] and OpenStack Sahara).

In this section we examine the initial provision of a big data service (e.g. MapReduce or a large scale graph analytics engine) on top of the API exposed by the IaaS provider. Assuming we have predefined VM images containing the required software, we still need to configure the distributed processing platform nodes and provide each node with data for processing. This "data loading" process is often overlooked by most research papers, but it would be essential for a fair comparison on the results obtained in different cloud infrastructures.

The sequence of tasks needed to prepare a big data job for parallel analysis on a set of recently deployed VMs is as follows:

- *Partitioning*: the data set is split and assigned to the workers, so that data processing can occur in parallel.
- *Data distribution*: data is distributed to the VM where it is going to be processed.
- *Application configuration*: the VMs have the big data applications correctly configured
- *Load data in memory*: in some computing models, during job preparation, the data must be loaded from the hard disk to RAM.

We discuss each task in the following subsections. We present design alternatives for the tasks, discussing the pros and cons of each design. Discussion is grounded on experimental results and assessment of the manageability of the different alternatives. These preliminary experiments lead the way for us to propose an architecture for offering BDaaS on top of a virtualised infrastructure.

A. Data Partitioning

The key for most big data systems is scaling horizontally (or scale out), in the hope that adding more resources (VMs) will reduce the overall execution time. Therefore, partitioning the data and assigning partitions to each VM is the first task of the big data job configuration.

Placing the right data on each VM is an NP-hard problem [8], with different constraints depending on the specific computation model. There are numerous heuristics in the literature that try to find partitioning schemas that can both: 1) keep partitions balanced (so that all the VMs crunch an equivalent portion of the data set) and 2) reduce the number of messages exchanged between VMs to synchronise their state.

Depending on the selected partitioning heuristic, data partitioning can be a very time consuming task (see [9] for instance).

B. Data Distribution

Deploying small data sets across a few servers (or VMs) is a trivial task. However, deploying TBs of data across hundreds or thousands of servers becomes substantially more challenging from a performance perspective. Substantial amounts of data must be transferred to each newly created VM[2].

Data distribution approaches should focus on reducing the overall transfer time. Reducing the total amount of data transferred around the local network reduces the impact of deploying such a service on the network. However, the cost factor from renting idle VMs waiting for the data transfer is the dominating element in this analysis[3]. It must also be considered that a data centre environment provides no guarantees of available traffic, which can render any prediction-based strategies useless. In this subsection we analyse the different approaches for performing data distribution among the deployed VMs.

a) Centralised Approach: As a first naive approach, we let all the VMs download the required dataset from a central repository (Figure 1A on page 3). VMs contain an initialisation script that connects them to the central repository so that they can get the required data after boot (this approach is used by most academic efforts [6][4]).

Chaining sequential transfers from a central location will render prohibitively high transfer times for the deployment of BDaaS. The bandwidth to the central server can easily

[2]Data transfers are typically made to the local disk of the worker VMs. Some authors propose using iSCSI Storage Area Networks as copy-on-write logical volumes attached to all the VMs, like for instance [10]. This moves the heavy load traffic from the local network to the Storage Area Network (SAN), but the bandwidth limitation would be similar.

[3]We assume the data has already been loaded to the cloud and is available from one of the many storage systems available in the cloud, like blob stores, and databases.

[4]See also: http://thegraphsblog.wordpress.com/running-mizan-on-ec2/

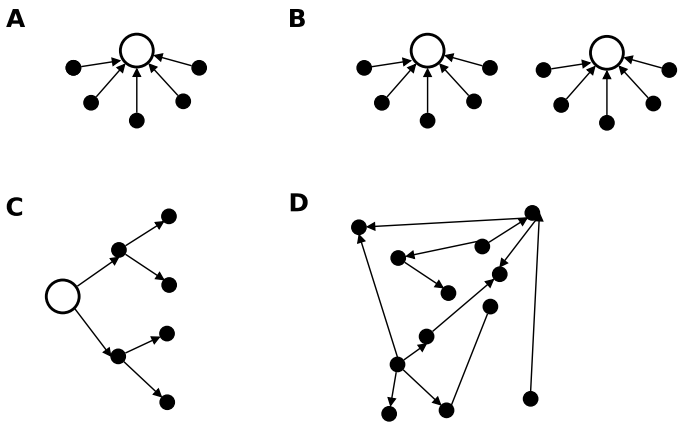

Fig. 1. Data distribution alternatives

TABLE I
NETWORK PERFORMANCE WITH ONE SERVER AND ONE CLIENT

	10Gbit Ethernet	Infiniband
1 TB	31 min	4 min
10 TB	3 h	47 min
100 TB	28 h	6.8 h

become a bottleneck for the whole transfer (see Table I). Also, if the transfers are requested in parallel, the central server will drop connections (due to throttling or denial of service effect) and get a "flash crowd effect" caused by thousands of VMs requesting blocks of data. We need smarter approaches to data distribution.

b) Semi-Centralised Approach: In order to alleviate the effect of all clients accessing simultaneously the same server (flash crowd effect), and also potentially reduce the stress on the networking infrastructure, it would be possible to shard the dataset across different machines in the data centre (Figure 1B).

A perfectly orchestrated download of the shards (so that the VMs do not get the same shard at the same time) would reduce the figures on Table I by M, where M is the number of shards.

This approach presents limitations when the data sets change over time (which is the case for most companies). It is very difficult to foresee the bias with which data sets may grow. For instance, one could shard personal data based on the initial letter of the family name, but family names do not have a uniform distribution. Even if we accounted for the name distribution bias, it may still be the case that more customers whose first initial is 'E' join our services. In that case, we would need to re-shard the 'E' shard again. Semi-centralised solutions often require re-replicating or re-sharding, making things hard to track and maintain in the long run[5].

c) Hierarchical Approach: Semi-centralised approaches are hard to maintain, specially if new data are continuously added; centralised approaches do not scale well once you get past a few hundred VMs (in our experiments we observed that the server containing the data starts dropping connections and overall throughput decreases by 2-3 orders of magnitude).

A next logical step would be to benefit from the knowledge IaaS providers have on the underlying network topology of the data centre (Figure 1C). Building a relay tree where VMs get data not from the original store, but from their parent node in the hierarchy, which ideally is in the same rack. This way N VMs will access the central server to fetch data, and as soon as some blocks are downloaded by these N VMs, they will provide the blocks to N additional VMs (ideally in their same racks), and so on. This way we also confine most of the traffic within top of the rack switches and avoid more utilised routers. The VMs need to be finely configured to download the data from the right location at the right time (see more on the section on configuration below[6]).

Some P2P streaming overlays like PPLive or Sopcast are based on hierarchical multitrees (a node belongs into several trees), which may be used to implement this approach. In practice their multi-tree nature has shown to evolve towards a mesh-like topology similar to P2P approaches [11], [12].

d) P2P Approach: The downside of the hierarchical approach is that it provides no fault tolerance during the transfer. If one of the VM deployments fails or the VM gets stuck after the transfers have been initiated, it is not easy to recover from failure and reschedule transfers (all the branches from the failing point need to be re-created and transfers re-started). Failure of one of the upstream leaves in the hierarchy dries the flow of data to the nodes that were supposed to be fed from there. This also implies more synchronisation is required.

To deal with this issue, we adopted an approach that also takes advantage of the fact that the data centre environment presents low-latency access to VMs, no NAT or Firewall issues, and no ISP traffic shaping to deliver a P2P (BitTorrent) delivery approach for big data in the data centre (Figure 1D).

Also, since having thousands of VMs connecting to a single repository will result in throttling mechanisms being activated or the server dropping connections, we employ an adaptive Bittorrent topology that evolves as block transfers get completed.

C. Application Configuration

Before the data transfer has started, the VMs need to be configured with the required software. For instance, we need a system that takes care of installing and configuring the right data transfer tools (e.g. BitTorrent) and in parallel gets the Big Data application installed on the VMs, so that by the time the data transfer is done, the application is ready to be started in an automated manner requiring no user intervention.

D. In Memory Data Loading

As mentioned above, some big data frameworks require loading the data from the local disk into the memory of the VM for processing. Depending on the type of application, these transfers from disk to memory can be really bulky (e.g.

[5]http://gigaom.com/2011/07/07/facebook-trapped-in-mysql-fate-worse-than-death/

[6]For now, it suffices to say that we need to create a mechanism for "rack awareness" on the VMs so that they find each other in the rack and organise that branch of the tree autonomously from VMs on other racks.

TABLE II
SUSTAINED RATES FOR DISK WRITE AND READ OPERATIONS IN MB/S

	HDD	SSD
Sustained read rate	160	409
Sustained write rate	107	321

100 TB deployed over 1000 VMs is 0.1 TB per VM if the whole data set is to be processed in memory). Disk read performance can be critical for large datasets.

The rates shown in Table II may be misleading. VMs run a virtualised memory hierarchy that includes virtual disks. This hampers the actual performance obtained on real disks. The exact sustained rate that can be achieved on a VM is hard to obtain. The I/O is shared across VMs and there is a physical limit that cannot be exceeded. If our VM happens to be co-located with a "noisy neighbour" making extensive usage of I/O operations (see [13] for an example on the importance of noisy neighbours), then our read/write rates are going to be nowhere near those in Table II.

Loading 1 TB of data from HDD into the memory of a physical machine can take hours. In order to reduce this, some well-known public cloud vendors offer VMs with SSDs, which reduce the time it takes to read the file from disk and load it in memory.

On the other hand, a quick look at some of the descriptions of the available VM sizes in public cloud vendors reveal one appalling fact: large disks are difficult to come by (only a few instance types allow for more than 1 TB storage space) at about $3 per hour (meaning that we will pay for hours while files are being transferred and loaded in memory before we can actually do any productive work).

The following section dives deeper into the design principles that drove the system we implemented. It is structured in different subsections/functions, each of them highlighting a relevant design aspect.

III. DESIGN OF THE PROPOSED SOLUTION

A. User Input Processing

The user provides the parameters to size the virtual infrastructure, e.g. number and size of slave nodes, and the cloud provider to use. In addition to the topology of the virtual cluster, input files and output folder locations are captured. Thirdly, software configuration parameters, i.e. total number of machines, local directories where the data will be located, data file names, etc. are captured. Finally, the cloud provider credentials, i.e. username/password, token, and the like. All these user provided configuration parameters are further referred to as configuration parameters. This is the only user interaction needed, the rest of the process is automatically done by the proposed framework.

B. Centralised Partitioning

Data partitioning splits the data input into multiple chunks (in principle as many as VMs). Partitioning can occur at two points in the provisioning of the big data service. It could

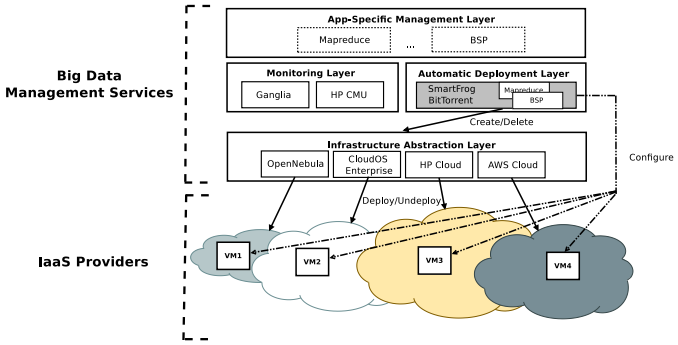

Fig. 2. Main building blocks of the big data provisioning service. The main innovation is the interaction between SF and BT, highlighted as a grey box, to create a data loading overlay with a topology that dynamically evolves as required.

be done initially at a central location with a "partitioning" service. A second alternative would be copying the whole data set to all the VMs involved in its processing and let the local software pick the right data. There are parallel strategies for partitioning the data (see [14] for a recent example) that can take advantage of this approach. However, that requires the leased VMs to have higher storage capacity, and also imposes more stress on the data distribution task.

We have created a centralised partitioning service that runs on top of where the original copy of the data is stored (e.g. a logical volume on a SAN, attached to a partitioning VM). When a new big data instantiation request arrives, the partitioning VM is spawned and configured to partition the data, determining a map of blocks to be transferred to each new VM. The central partitioner is an integral part of the App-specific management layer (see Figure 2).

The partitioner enables dynamic loading of different partitioning strategies. For instance, for very large datasets a "binary" partitioning of the file may be enough, while more specialised processing (e.g. large scale graphs [15] use the modulus of the integer ID of the vertex to decide where it needs to be allocated). At this stage, data is partitioned logically (no actual movement of data is initially done, but an index is kept that limits the beginning of a partition and end of the previous one).

Using a central partitioner reduces the amount of data transferred across the network from M (where M is the size of the file containing the data set) to M/N. The same applies to the local disk requirements of the computing VMs. Our partitioner creates an overlap between partitions (see Figure 3 on page 5). The overlapped part, which belongs to other partitions, is ignored by the data set loader on each of the workers, but it is used to seed other peers which may need it.

The higher the overlap, the higher the consumption of bandwidth and the lower the transfer time (assuming the actual maximum capacity of the underlying network fabric has not been reached). On the other hand, if the overlap is not big enough (like in the example of Figure 3), some chunks of the file will only be stored in one of the initial relay nodes accessing the repository.

Since a full overlap is not possible, we assumed some

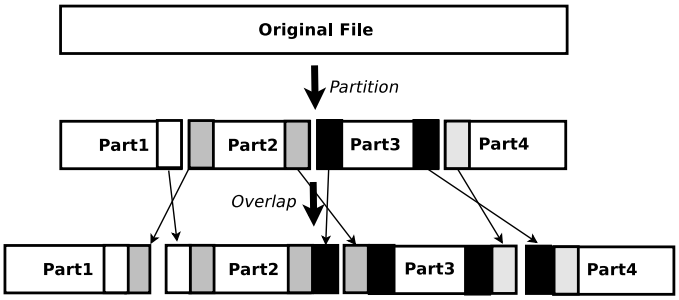

Fig. 3. Chunk overlap to reduce overhead in relays.

chunks would only be located at the main repository. Access to these chunks is randomised to reduce the likelihood of a flash crowd effect.

C. Data Distribution

The logical partitions defined in the first step need to be distributed among the VMs that will process the data. Data distribution is a critical task; distribution should maximise transfer rates and minimize information redundancy. We support this function of our architecture with two components. The first component is a torrent controller, a lightweight httpd server that hosts an announce path onto which BitTorrent clients update their state. The second component is a central data repository which has the files to be deployed onto all the VMs.

The central partitioner decides how the original file is split across machines (see above). After getting the global partition index from the partitioner, each partition of the files is placed into a directory and a new torrent is created per directory; this directory is then compressed and packaged and transformed into a .torrent file (small sized file with basic hash information about the compressed directory). The BitTorrent tracker keeps track of which .torrent files are currently being distributed.

Our configuration management tool sets a number of VMs to play the role of initial relays (the number is dynamically computed as $log(N)^3$ where N is the total number of VMs; this result was empirically determined to be the most favourable, see results below). Initial relays connect to the data repository and relay data to the rest of the peers. All other machines will get blank information from the tracker until the initial relays get some blocks of data, in which case peers can connect to their assigned relay. Initial relays prevent the initial avalanche that would happen if all peer VMs accessed the repository at once while initialising the download (also known as flash crowd).

The peers will receive the files and distribute the chunks amongst themselves. Once a peer is done downloading its assigned chunk, it will continue seeding for a configurable amount of time (which depends on the size of the download) to prevent a hotspot effect on its initial relay.

Since the number of initial relays is much smaller than the total number of VMs, the relays get data from one or more partitions to serve as initial relay for several peers. Once peers start getting blocks and seeding them to other peers, relays are automatically reconfigured by our configuration management systems as normal peers. If the partitioning algorithm leaves some chunks without replication and no relay gets data, the peers can randomly connect to the central repository once they do not have any other chunk left to download.

More formally we could express this as a Markov chain model ([16], [17], [18]), which has as state variables the populations of initial relays and "normal" peers in the system, behaving as follows:

Peers arrive as a Poisson process of rate λ to their assigned relay, stay in a leecher queue until completing the download, and then in a relay queue for an exponential time of parameter γ; the mean upload time for the file being $1/\mu$ which leads to a continuous time Markov chain.

We assume the mean upload time for a block is $1/\mu$, this is capped by the 10 MB/s limit we impose to avoid crashing the network. In our case λ is controlled by our configuration management software so that the rate of arrival of peers is controlled at the beginning. The tracker has been modified to filter out responses from peers during the initial stages. Also the 10 MB/s cap is indirectly limiting the value of λ (not just initially, but throughout the whole transfer).

The number of initial relays accessing the repository is constrained and controlled to prevent the "flash crowd effect"; all other nodes are "capped leechers" as they are not allowed to establish any connection until their assigned initial relay has got some blocks from the initial repository. Thus, during the load of the initial blocks we let $x(t)$ denote the number of capped leechers at time t, and $y(t)$ the number of relays. Therefore, the total upload capacity in the system in blocks per second becomes:

$\Omega_{up} = \mu(y + x)$; since $x(0) = 0$ and $y(0) = 0$ since the initial relays are not uploading anything, it turns out that $\Omega_{up} = 0$

Let Ω_{down} denote the total download rate of the system, in blocks per second:

$\Omega_{down} = \mu log(N)^3$ where N is the total number of VMs.

After this, there is a transient period where initial relays start to supply peers with data blocks and peers that finish downloading some blocks start to seed themselves (leechers do not leave the system, which is specially true for these large data sets they need to download):

$$dx/dt = \lambda - \Omega_{down}(x, y)$$

$$dy/dt = \Omega_{down}(x, y) - \gamma(y)$$

where $\Omega_{down}(x, y) = min\{\Omega_{up}, cx\}$ states that Ω_{down} is constrained (only) by either the available upload capacity or the maximum download rate per peer, defined as c.

After this transient period, the system reaches a dynamic equilibrium characterised by: a total upload capacity in the system in blocks per second becomes:

$$\Omega_{up} = \mu(y + x);$$

since $x(t) = M$ and $y(0) = 0$ since all VMs have turned into seeders now, it turns out that

$$\Omega_{up} = \mu M.$$

It is easily checked that at an equilibrium of dx/dt and dy/dt the number of seeders must be

$$y^* = \lambda/\gamma.$$

In our case the number of leechers will tend to none after the initial setup period, independently of the value of λ.

D. Automatic Deployment

This function performs the actual deployment of the virtual infrastructure, installation, and configuration of the software installed in the VMs (in our specific case it is our VMs and the peers for distributing the partitioned file).

Firstly, the specified number of VMs are created and started within the selected cloud provider. Our architecture includes an Infrastructure Provider Abstraction Layer (see Figure 2 on page 4) to abstract the specifics of individual providers, presenting a common high level interface to them all.

VMs are configured by the boot volume attached to them. In the case of MapReduce in the cloud, most public cloud providers use a preloaded boot volume which contains a pre-configured MapReduce implementation for slave and master nodes, i.e. a static provisioning of the MapReduce service. As pointed out by Loughran et al. [5], this method is not well suited for public clouds since: 1) it requires manual maintenance of each image (patches, updates, mounting, etc.); 2) missing application configuration parameters or parameters that do not fit the default values need to be manually set by the user; 3) there is no runtime model with the current status of the infrastructure and services.

Our proposal is sharing a boot volume across all VMs. The volume contains a clean OS installation with a Configuration Management Tool (CMT) and our modified BitTorrent installed as daemons. There is a myriad of alternative CMT tools for the shared boot volume. Puppet [19], Chef [20] and CFEngine3 [21] are CMT solutions originally designed for client-server distributed environments. Although they do a good job at configuring software artifacts, they do not cope well with dynamic relationships between components in cloud environments, where virtual infrastructure is created as part of the provisioning of services [5].

SmartFrog (SF) [22][7] is a pure peer-to-peer architecture with no central point of failure that enables fault tolerant deployment and dynamic reconfiguration to change infrastructure and services at runtime. Another crucial element is that SF uses late binding configuration information to configure services. This information becomes available after a prerequired step has been completed but it is not readily available at the beginning of the deployment. This is specially important in cloud environments where users usually do not have control over resource names, e.g. IP address.

The VMs start the SF daemons during boot time, see Figure 2 on page 4. After the VMs have started, the SF daemon will dynamically provision the required services, following the received software installation and configuration instructions [23]. The deployment layer plays the role of an orchestrator making sure the following actions occur when needed:

- *SF setup*: SF downloads the application configuration file to some VMs and initiates the P2P distribution on a software distribution overlay (thin lines in Figure 4 on page 9). The VMs operating as masters (e.g. VM1 in Figure 4) get the configuration file (user provided data plus specific partitioning information) and start seeding it to all other VMs in a P2P manner (#1 in Figure 4).
- *Creation of the software distribution overlay*: SF downloads required software from service catalogues (see Figure 4), which are repositories with all needed configuration files and software (#2 in Figure 4). These repositories store the big data software packages to be copied and executed on the VMs and the configuration templates filled with the configuration parameters and copied into the VM (see [23] for more details). The software configuration files and binaries are also distributed in a P2P manner. This approach speeds up the distribution and prevents "flash crowd calls" against the repositories when hundreds or thousands of VMs are deployed. Some optimisations like reduced timeouts, deactivate encryption, and keep files in memory for the whole process have been made to get the maximum throughput in the transfers of the software bundles (which are in the order of a few hundred MB).
- *Creation of the data distribution overlay*: Once the download of our modified BitTorrent from our git repositories has finished, SF configures BitTorrent with the parts of the data set that need to be processed by that specific host (as specified by the central partitioner). Then, BitTorrent is started by SF and the data transfers get started for the relays (#4 in Figure 4) and later on by the peers (#5). Remember that depending on the level of overlapping, more of the servers will have to go to the central data repositories, but this access is randomised so as to minimise the number of concurrent calls and transfers from a single central location. The overall bandwidth for concurrent transfers was capped at 10 MB/s to avoid outages in the data centre due to too many connections exhausting the bandwidth. Note that many of these VMs may be in the same rack or even in the same physical host. However, big data VMs will have very high RAM requirements, which minimises the possibility of having too many of them co-located on the same physical host.
- *Software downloads* take much less time than data downloads (our distribution is about a few hundred MB), see Tables III and IV. Thus, in parallel with the download of the dataset, SF configures our installed software (see Section IV below) from an automatically generated configuration file. This file is created by the Automatic

[7] http://www.smartfrog.org/

Deployment Layer using the configuration parameters taken from the user. In this scenario one of the VMs is randomly chosen to be the master and the others become slaves of the application.

- Upon completion of all the transfers, the modified BitTorrent signals back to SF and SF initiates all the required big data processes on the machine.

E. Monitoring

The Monitoring function tracks the progress of the application so that SF can make any corrective action to keep the desired state of the resources. In these big data applications the desired state is "running", so SF will check if some VMs have failed and automatically spawn new ones if needed.

Monitoring information can either be periodically pulled from the components or pushed to it. Different plugins can be used to provide additional information depending on the specific software stack and cloud provider.

F. Infrastructure Abstraction

The Infrastructure Provider Abstraction function in Figure 2 (page 4) homogenises the APIs exposed by the different cloud infrastructure providers, presenting a high level view across different public and private infrastructure providers. We adopted OpenStack's APIs as *de facto* standard to be used by our Automatic Deployment Layer and built adapters for all others. Additional providers can be added by supplying new Infrastructure Provider mapper implementations.

IV. Implementation Details

Parts of the proposed architecture have been implemented as open source software and released publicly under LGPL[8].

The application consists of two different components: A RESTful web service offering the functionalities of the App-Specific Management layer and a front-end web-based application.

Classic big data applications involve using the MapReduce abstraction for crunching different data sources (e.g. log files). Hadoop is one of the most popular frameworks for this type of processing. Graph analytics are also a popular application that crunches large amounts of data. There are many systems for processing very large scale graphs, like Pegasus, Pregel and others [24], [15], [25], [26]. Most of them are based on Valiant's Bulk Synchronous Parallel (BSP) model, consisting on processors with fast local memory connected via a computer network [27]. A BSP computation proceeds in a series of iterations (also known as supersteps). We tested our tool in Hadoop scenarios (see [5] for more details), as well as with a BSP graph processing system [26].

We loaded a graph with at least 600 million edges and 100 million nodes, each of which performs more than 100 differential equations that simulate the behaviour of the human heart (see [26] for details).

The boot volume is composed of a clean installation of Ubuntu 13.10 and SF v3.17. Additional components that SF relies on have been added to the image. For example, apt-get to install necessary Linux packages required by BSP and Hadoop.

Furthermore the SF, BSP and Hadoop components have been pre-installed[9], which provides the necessary classes used to map configuration parameters to the configuration file format expected by the required application (BSP or Hadoop in our case) and to control individual services (e.g. start, stop. etc.). Both Hadoop and BSP have been used in the evaluation of our architecture, although the key factor in performance is the size of the files and the number of VMs involved.

To test the virtualisation abstraction capabilities of our architecture, we developed a number of different cloud connectors, namely HP Helion, HP Cells, VMWare, Mock, OpenNebula and Amazon EC2.

TABLE III
Distribution time of the software to the VMs via SmartFrog.

	100 VMs	1000 VMs
Centralised distribution	403 s	1400 s
SF distribution	15 s	18 s

TABLE IV
Distribution time for the data sets to the VMs via our modified BitTorrent client. NA is assigned for transfers longer than 24h, which were automatically cut off by SF and marked for optimisation and analysis. NS means there is no space available on the VM disk to attempt the transfer.

	100 VMs		1000 VMs	
	Centralised	BitTorrent	Centralised	BitTorrent
1 TB dataset	1010 min	21 min	NA	14 min
100 TB dataset	NS	NS	NA	5 h

V. Evaluation of our Design

A. Results of Our Data Distribution Methodology

Table III shows the measured time for software transfers using our proposed methodology based on SF. The results show the performance limitations of the central distribution approach (caused by factors such as the previously described "flash crowd effect"), with substantially longer times, and linear increase of the distribution time with the number of machines. With our SF methodology, the low distribution time values (with small increase with ten times more machines), make installation and configuration time the main factor.

Data transfer proved to be more demanding for the big data applications deployed in our system. As can be observed in Table IV, a central distribution from a single data repository into a large number of VMs rapidly becomes infeasible. Using this naive strategy may result in more time taken to distribute data than doing the processing job that extracts value from the data. Our modified BitTorrent technique distributes the data across the VMs much more efficiently, with our experiments going up to 100 TB of data distributed over 1000 VMs.

[8]http: //smartfrog.svn.sourceforge.net/viewvc/smartfrog/trunk/core/

[9]The BSP service is a targz'ed file built on a periodic basis by our continuous integration system. The same applies to our modified version of BitTorrent, which is a slightly changed version of BitTornado, https://github.com/effigies/BitTornado.

In order to establish a fairer comparison with a centralised approach we also tested a "centralised sharded" variant, where we also used a central partitioner (instead of distributing the whole file everywhere). We tested this approach for 1 TB on 100 VMs, with network bandwidth limited to 10MB/s (as in the case of BitTorrent). The distribution time in this experiment was improved by a factor of 5, although our P2P distribution approach completed 10 times quicker.

Figure 4 on page 9 shows (dashed lines) how some peers play the role of seeder nodes connecting to the data repositories (circled 5), while others play the role of pure peers, contacting only other peers or seeders. This behaviour is enabled by embedding our configuration management (SF) technology in the architecture.

Initially our CMT restricts tracker access to only the relay nodes, hence having an initial unchoke value k around one. The value gradually increases as more peers become leechers or seeders and the relays stop performing their role to become normal seeders. At that point we get $k = 4$, which is the default for most BitTorrent client implementations

Figure 5 on page 9 shows how the topology of the distribution network changes from a hierarchical (tree-like) structure to a P2P one. The initial tree-like infrastructure avoids the "flash crowd effect" of hundreds/thousands of VMs accessing a single central repository at once, and the evolution to a more decentralised P2P structure maximises throughput. After 4 minutes the system adopts the P2P organisation shown on the right hand side panel on the Figure. Peers coordinate with the configuration management system the cases where "exclusive" chunk is needed (no other peer has the chunk). In these cases the peer needs to directly go to the central repository (as can be observed, there are always a few connections to this repository). Central repository access is randomised by the configuration management system upon starting BitTorrent.

In our experiments we found this approach to be about $\simeq$40 times faster than a purely centralised approach (90th percentile on more than 100 deployments of 100 VMs working on 1TB), depending on the dataset and the deployment of VMs done by the cloud scheduler.

In Section II above, we described several alternatives to implement more sophisticated data distribution techniques (not just a centralised approach). We compared our approach with hierarchical and pure P2P options (Sharded approaches were ruled out due to their complex maintainability and evolution).

Hierarchical techniques perform worse than our approach (see Figure 6A on page 10), mainly due to the fact that the level of data redundancy is very high (initially 2 machines get 5% of the file, 50TB in our example). This is made even worse in the light of failures in the propagation of the file in the hierarchy: if the failure happens early (at the beginning of the transfer, see Figure 6B, the throughput is reduced even further. This is less relevant if the failure occurs late in the transfer process (as the latest VMs handles only a minor fraction of the original file).

Deploying the data in a HDFS-like fashion (data nodes get blocks from HDFS client on a central storage) somewhat resembles this hierarchical approach (only it is more efficient since there are not many layers involved and it benefits from rack awareness). Transferring data to 3 HDFS datanodes resulted in very poor transfer rates (around 5 MB/s, see in Listing 1 the results of a run of the standard Hadoop HDFS benchmarking tool, TestDFSIO); this is substantially slower than the same setup with our overlapped partitioner and modified BitTorrent. Other authors report throughput around 50 MB/s for more optimised settings [28], which would still be 2-3 times slower than our approach. These results are not surprising, as HDFS is not optimised for speed of transfer. It would be interesting to know if HDFS could be modified so add coordination among the data nodes, so that they collaborate to get the first block from each other, instead of all of them accessing the client.

Listing 1. Capture of the output of the TestDFSIO Hadoop benchmark

```
TestDFSIO ----- :
write Date & time: Fri Dec 13 2013
Number of files: 1000
Total MBytes processed: 1000000
Throughput mb/sec: 4.989
Average IO rate mb/sec: 5.185
IO rate std deviation: 0.960
Test exec time sec: 1113.53
-----
TestDFSIO ----- :
read Date & time: Fri Dec 13 2013
Number of files: 1000
Total MBytes processed: 1000000
Throughput mb/sec: 11.349
Average IO rate mb/sec: 22.341
IO rate std deviation: 119.231
Test exec time sec: 544.842
```

We also compared our approach to a traditional P2P transfer mechanism. We chose Azureus, a BitTorrent client, for its capability of handling nodes that are overloaded with requests. Azureus uses a Distributed Hash Table (DHT) that stores values for a key in 10 alternate locations when the read rate exceeds 30 reads per minute or when the current node hits a global capacity limit of 64,000 keys or 4MB of storage. It also limits any one key to at most 4kB of storage or no more than 512 values, corresponding to a swarm with 512 members. Crosby et al. observe fewer than 4 application messages per second [29], but in our scenario we can have thousands of peers connecting to a single machine, triggering replication of values to 10 alternate locations.

At the start of our scenario all DHT clients try to connect to the central repository. However, 60% of the established connections do not achieve any data transfer, since they are immediately terminated with a "too busy" message. As in the initial stages there are not 10 alternate locations for the requested content the rejected peers are unable to retrieve the requested data. These results in an increased transfer time of 19% for 100 VMs and 1TB and 33% for transfers of 100 TB to 1000 VMs compared to our hierarchical technique.

As can be seen in Figure 6, the P2P client is more resilient to failure and performs better than a hierarchical approach.

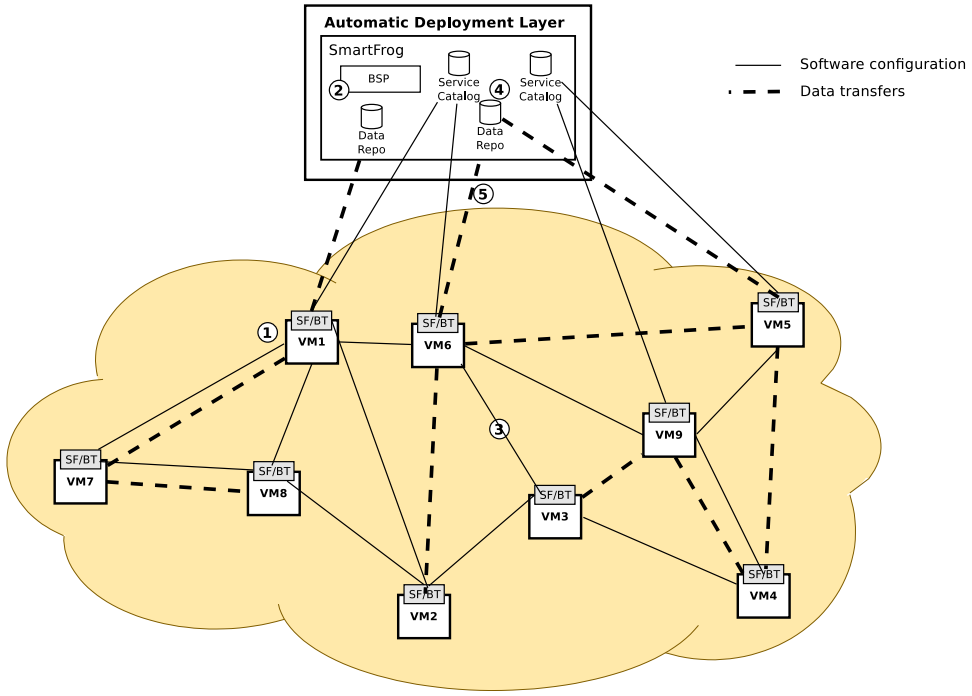

Fig. 4. P2P interactions between our SF/modified BitTorrent daemons and the Automatic Deployment Layer.

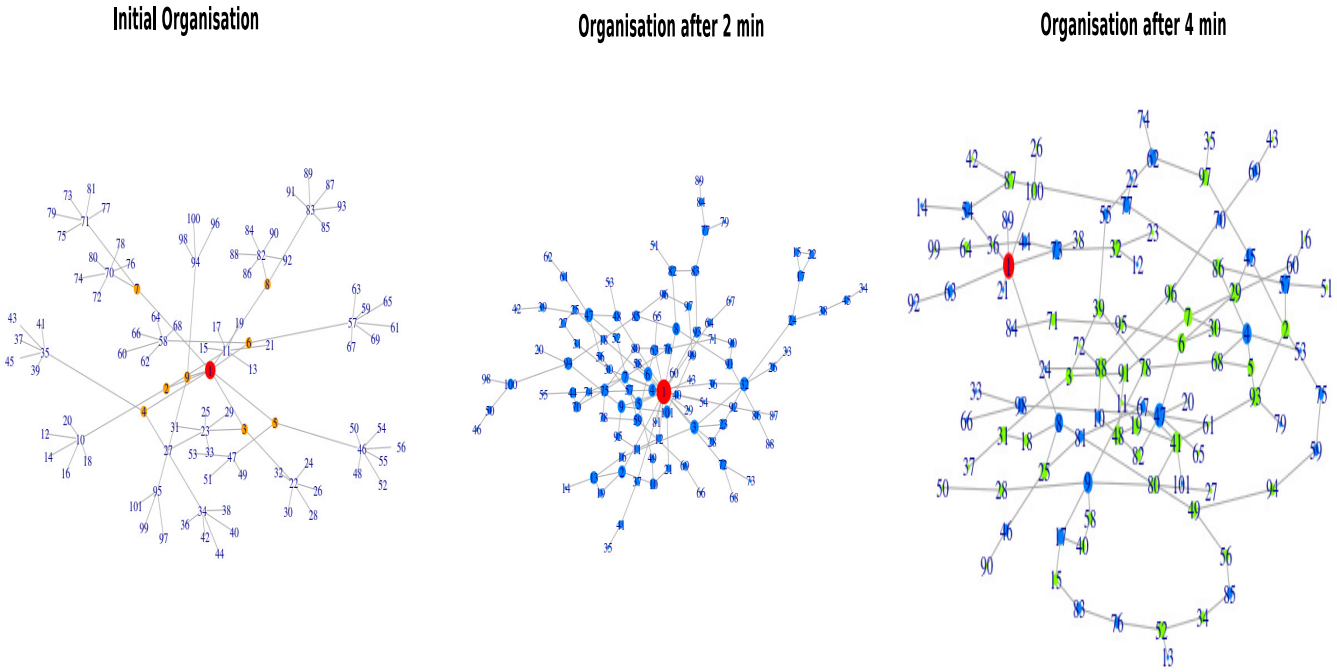

Fig. 5. Snapshots of the distribution process of 1TB into 100 VMs with 30% overlap. The red vertex is the central repository, and orange vertices illustrate the initial seeder peers. Green dots are peers with less than 20% of their chunks transferred, and blue vertices have downloaded between 20 % and 40 % of their assigned file.

B. Effect of Partition Overlapping

As can be seen in Table V, a low overlap shares the limitations of a centralised approach; too many VMs connect to the central repository at the same time and it starts dropping connections, resulting in reduced throughput. As the level of overlapping is increased, more P2P connections are established and throughput is increased. Distribution time decreases up to $\simeq$45 % overlap, but the situation degrades as we increase overlap further.

Our empirical tests yielded recommended values of 20-30% overlap for getting the maximum speed up, although the specific values will depend on the adopted scheduling heuristics (see discussion below).

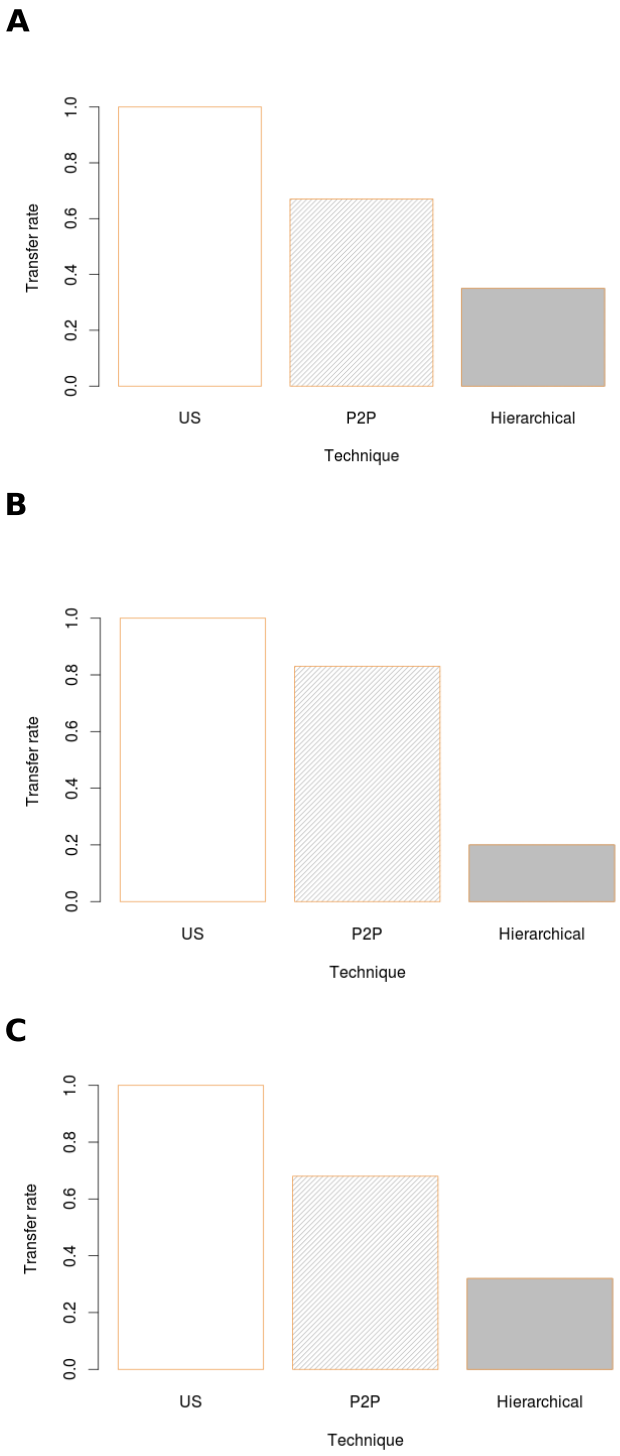

Fig. 6. Performance Comparison of Different Techniques for Loading Large Datasets on VMs on-demand in normal conditions (A), during an early failure (B) and during a late failure (C) of one of the machines. Data represent the 90th percentile of $n >= 20$ repeated measurements. Hierarchical data are based on a x2 expansion factor (each VMs serves two VMs downstream).

C. *Effects of the Number of Relay VMs*

The transfer rate increases as more relays are placed in the system up to a point where it starts a steady decline due to bandwidth limitations and additional synchronisation overhead (see Figure 7 on page 11).

TABLE V
DISTRIBUTION SPEEDUP FOR VARYING OVERLAP LEVELS. MEASURED AS THE DISTRIBUTION TIME FOR A CENTRALISED APPROACH DIVIDED BY THE TIME TAKEN BY OUR APPROACH. FIGURES ARE GIVEN FOR THE 90TH PERCENTILE OF MORE THAN 50 EXPERIMENTS.

% of overlap	Speedup
10	5
30	98
50	12

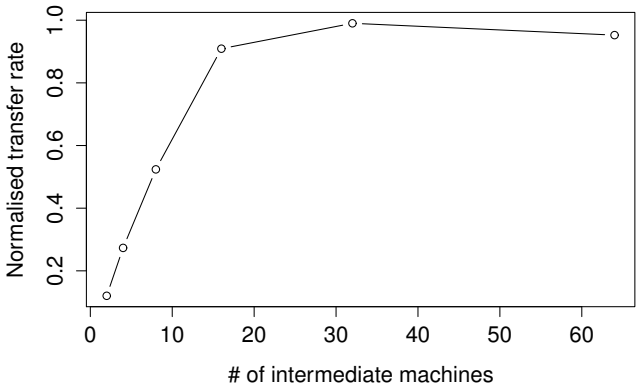

Fig. 7. Effect of varying the initial number of intermediate servers for 1000 VMs and 100 TB transfers.

D. *Detecting Poor Connectivity*

A standard metric for measuring the global connectivity of the peers is the second eigenvalue λ_2 of the stochastic normalisation of the adjacency matrix of the graph the peers form during the transfer, values over 0.85 are typically considered a sign of poor connectivity [30]. During the first phase (where relays are in action), our system presents poor connectivity ($\lambda_2 \simeq 0.9$ during the first 2-3 min).

The nodes arrange in the initial stages in clusters of nodes with few inter-cluster connections (as shown on the left hand side of Figure 5 on page 9). This transient poor connectivity limits the bandwidth consumed by thousands of nodes simultaneously connecting to a single data source. After this initial transient period (which typically lasts only for a few minutes (percentile 99.5th is 5 min) the λ_2 value decreases to a final average value of $\simeq 0.66$.

The fact that the topology is tree-shaped at the beginning and evolves towards a P2P-like one as time goes by explains why connectivity is poor. There is just one path between any two nodes in different trees. Thus, this connectivity value characterises the evolution in the graph topology as new chunks of the file are made available by non relay peers. The λ_2 value is correlated (0.912 ± 0.02) with the number of trees in the topology at any given point in time [10]. As soon as new data chunks are made available, peers start getting their data from other (not relay) sources and the connectivity changes very rapidly. Thus, the number of trees decreases exponentially

[10]Simple recursive graph traversing algorithms can be applied here.

with a time constant $\tau \simeq 3.5min$.

VI. Related Work

Data mining systems have typically run on distributed clusters and grids, which assumed the processors are scarce resources, and should hence be shared. When processors become available, the data is moved there and the data crunching process runs until all the results have been obtained [1].

This works well in practice for small data sets, but it soon becomes intractable for larger data sets where an unacceptable part of the time can be spent in moving data around.

In contrast, recent approaches claim data should be fixed and computation should be moved wherever the data are. The Google File System [4] and its open source counter part, the Hadoop Distributed File System (HDFS)[28] support this model. Also based on this model, Grossman and Gu [2] explained the design and implementation of a high performance cloud specifically designed to archive and mine large distributed data sets, highlighting the advantages of IaaS clouds for processing large data sets.

None of these proposals consider the case where the infrastructure, the data and the software running on those data need to be deployed on-demand. Other proposals such as Hadoop on Demand (HOD) offer client-side alternatives for deploying Hadoop without considering the potential bottlenecks of large-scale data distribution.

There are also some systems that have the goal of allowing cluster sharing between different applications, improving cluster utilisation and avoiding per-framework data replication like Mesos [31] and YARN (Yet-Another-Resource-Negotiator) [32]. We believe these could be deployed together with our approach, as we already support multiple data partitioning strategies, which would be suitable for different types of applications.

Structured P2P organises peers into an overlay network and offers DHT functionality so that data is associated with keys and each peer is responsible for a subset of the keys [33], [34], [35]. In hierarchical DHTs, peers are organised in groups with autonomous intra-group overlay network and lookup services organised in a top-level overlay network. To find the peer that is responsible for a key, the top-level overlay first determines the group responsible for the key; this group then uses its intra-group overlay to determine the specific peer that is responsible for the key [36]. None of them proposed a dynamic topology (initial tree/structure that dynamically evolves into a P2P topology as more chunks have been transferred), neither do they deal with dynamic deployment of application/data into VMs in the cloud (the application domain is usually restricted to routing overlays for lookup services). Our technique also couples this dynamic topology to software configuration management tools to ease the quality of experience for the end user.

VII. Discussion

Most public cloud vendors do not charge extra fees for intra-cloud transfers, but the cost to upload the data to the public cloud can be significant. In our work we assumed the data set had been pre-uploaded to the public cloud and the population of VM with the data occurs only within the cloud provider's network (no extra charge per new deployment). In case users want to upload their data from their own facilities, they should also attempt to balance the time it takes to upload data to the cloud (possibly over highly asymmetric links) with the amount of money charged by the cloud provider for uploading data into the cloud. The associated costs for downloading the results back to the user's facilities might require similar analysis.

Existing big data processing platforms make assumptions on latency and bandwidth that might not hold in a cross data centre environment, severely hurting performance[37]. We have not explored cross data centre configurations of our data distribution approach. We suspect some of the optimisations (low time out times, fewer retrials, agnosticism regarding ISP traffic policies, and the like) will not apply in this context and new challenges will be presented. Some authors have successfully tested this idea [38], and we believe some engineering optimisation work can make across-data-center transfers more suitable for on demand big data service instantiation than it is today.

While our approach scales well to a few thousand VMs, it could be argued that tracker-less approaches are more scalable. That would be a fair statement, but we know of few systems that require more than ten thousand machines today. In addition, our experiments with a DHT client show that the flash crowd effect can be too large when 1000 VMs are trying to get chunks of data from a single node. In the future, we would like to improve the interaction between SF and a trackerless BitTorrent deployment by, for instance, modifying well known protocols to filter unauthorised peers out [39].

We have shown the performance of our system with reasonably big data sets (TBs) and large number of VMs (100 and 1000). We did not include results at a smaller scale as they would be less relevant in the context of big data processing. On the other hand, scaling up to 10000 VMs and 1 PB would be impractical, as it would take too long to be usable for on-demand data analytic tasks in the cloud (further advances in data centre networking, e.g. optical connectivity to the rack, are required). The limits on data size can also be circumvented using complementary techniques. For instance, many data analytics applications work on text files and very good compression rates are easy to obtain (5:1 is not at all uncommon, even more in structured logs). Thus, a 100 TB file could be reduced to roughly 20 TB, which still would not fit in our 100 VMs and would take a prohibitive amount of time unless proper pre-partitioning and wiser transfers were scheduled.

Highly consolidated physical infrastructures imply VMs share room with many other neighbours. Some of these tenants may be less desirable room mates, specially those that make heavy use of the shared IO resources. Thus, VM allocation can cause significant variability on the obtained results. Smarter scheduling policies taking big data applications into account are needed. The academic community has proposed several alternatives for this type of smarter scheduling (see [40], [41] for instance), but few of these works have been tested in large scale environments, which makes transfer into products more

complicated.

Object stores such as OpenStack's Swift could be used as a first hop (instead of having our initial relay nodes). This is conceptually analogous to our approach. Assuming the object store nodes would play the role of our relay nodes, then our partitioner would need to upload the initial torrents to separate buckets in the object store (as it happens now with the relay VMs).

Debugging problems across logs distributed in thousands of nodes is a labour intensive problem. Being able to re-run a trace of all the events that occurred that lead to the failure/problem is essential to be able to reproduce the problem and fix it. Having smarter schedulers may pay off (see two paragraphs above), but not at the expense of obscuring devops-level understanding.

Uplink utilisation is critical for the performance of BitTorrent clients in classic scenarios. Every 10 seconds a node "unchokes" the $k = 4$ default nodes which have provided it with the highest download rates during the previous 20 sec. Each of the unchoked nodes is thus given an equal probability of pulling the missing chunks of data. Some works have reported 90 % utilisation of the uplink except for the initial period where utilisation is very low. Some authors have tried to use new protocols to improve the utilisation of the uplink [42]. Our technique is not constrained by limited asymmetric DSL uplink connectivity and therefore, uplink capacity is not our main concern. We limit the maximum transfer rates, though, so that we ensure peers do not consume all the network capacity in our data transfers.

Our approach works well in practice partly thanks to the enforced coordination at the initial stages. Network coding promises optimal utilisation of the available bandwidth [43]. Future work aims to explore the introduction of network coding mechanisms as another tool for the partitioner to schedule transfers of data to the right VMs.

VIII. Conclusion

Provisioning thousands of VMs with datasets to be crunched by their big data applications is a non trivial problem. A big data provisioning service has been presented that incorporates hierarchical and peer-to-peer data distribution techniques to speed up data loading into the VMs used for data processing. The method is based on a modified BitTorrent client that is dynamically configured by the software provisioning modules. Peers are initially configured in a tree topology, where a subset of VMs play the role of relay nodes (preventing flash crowd effects on the permanent data storage). As soon as some data chunks start to be ready in the leaves of the tree, the topology evolves to a classic P2P mesh shape. Our implementation and evaluation with hundreds of TB and thousands of VMs show this is an effective method for speeding up dynamic provision of big data applications in the cloud, obtaining improvements on transfer times around 30 % over current state of the art techniques. This may represent significant savings in the price paid by users of public clouds. At the same time, our system keeps a low entry barrier for users who may not be experts in infrastructure management (they deal with a single high-level declarative configuration file and the system takes care of configuring software and data loading).

Acknowledgment

The authors would like to thank Omer Rana from Cardiff University and Andrzej Goscinski from Deakin University for their valuable comments on the manuscript. We would also like to thank the anonymous reviewers for their great suggestions and encouragement during the review process.

References

[1] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[2] R. Grossman and Y. Gu, "Data mining using high performance data clouds: Experimental studies using sector and sphere," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '08. New York, NY, USA: ACM, 2008, pp. 920–927. [Online]. Available: http://doi.acm.org/10.1145/1401890.1402000

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 29–43. [Online]. Available: http://doi.acm.org/10.1145/945445.945450

[5] S. Loughran, J. Alcaraz Calero, A. Farrell, J. Kirschnick, and J. Guijarro, "Dynamic cloud deployment of a mapreduce architecture," *Internet Computing, IEEE*, vol. 16, no. 6, pp. 40–50, Nov 2012.

[6] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis, "Mizan: A system for dynamic load balancing in large-scale graph processing," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 169–182. [Online]. Available: http://doi.acm.org/10.1145/2465351.2465369

[7] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1496091.1496100

[8] K. Andreev and H. Räcke, "Balanced graph partitioning," in *Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '04. New York, NY, USA: ACM, 2004, pp. 120–124. [Online]. Available: http://doi.acm.org/10.1145/1007912.1007931

[9] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson, "From "think like a vertex" to "think like a graph"," *PVLDB*, vol. 7, no. 3, pp. 193–204, 2013.

[10] A. Coles, E. Deliot, A. Edwards, A. Fischer, P. Goldsack, J. Guijarro, R. Hawkes, J. Kirschnick, S. Loughran, P. Murray, and L. Wilcock, "Cells: A self-hosting virtual infrastructure service," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, ser. UCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 57–64. [Online]. Available: http://dx.doi.org/10.1109/UCC.2012.17

[11] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt, "Measurement and modeling of a large-scale overlay for multimedia streaming," in *The Fourth International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness & Workshops*, ser. QSHINE '07. New York, NY, USA: ACM, 2007, pp. 3:1–3:7. [Online]. Available: http://doi.acm.org/10.1145/1577222.1577227

[12] Q. Ye and C. Chen, "A study on topology model and data contribution strategy of pplive," in *Proceedings of the 2010 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, ser. CYBERC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 301–304. [Online]. Available: http://dx.doi.org/10.1109/CyberC.2010.61

[13] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 199–212. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653687

[14] Z. Zeng, B. Wu, and H. Wang, "A parallel graph partitioning algorithm to speed up the large-scale distributed graph mining," in *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, ser. BigMine '12. New York, NY, USA: ACM, 2012, pp. 61–68. [Online]. Available: http://doi.acm.org/10.1145/2351316.2351325

[15] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 135–146. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807184

[16] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '04. New York, NY, USA: ACM, 2004, pp. 367–378. [Online]. Available: http://doi.acm.org/10.1145/1015467.1015508

[17] A. Ferragut, F. Kozynski, and F. Paganini, "Dynamics of content propagation in bittorrent-like p2p file exchange systems," in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, Dec 2011, pp. 3116–3121.

[18] X. Yang and G. de Veciana, "Service capacity of peer to peer networks," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4, March 2004, pp. 2242–2252 vol.4.

[19] J. Turnbull, *Pulling strings with Puppet : configuration management made easy*, ser. FirstPress. Berkeley, CA: Apress New York, NY, 2007. [Online]. Available: http://opac.inria.fr/record=b1125928

[20] A. Jacob and E. Zygmuntowicz. (2009) Infrastructure in the cloud era. [Online]. Available: http://velocityconf.com/velocity2009/public/schedule/detail/8324

[21] M. Burgess, "Knowledge management and promises," in *AIMS*, 2009, pp. 95–107.

[22] P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, and P. Toft, "The smartfrog configuration management framework," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, pp. 16–25, Jan. 2009. [Online]. Available: http://doi.acm.org/10.1145/1496909.1496915

[23] J. Kirschnick, J. M. A. Calero, L. Wilcock, and N. Edwards, "Toward an architecture for the automated provisioning of cloud services," *Comm. Mag.*, vol. 48, no. 12, pp. 124–131, Dec. 2010. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2010.5673082

[24] U. Kang, C. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Data Mining, 2009. ICDM '09. Ninth IEEE International Conference on*, Dec 2009, pp. 229–238.

[25] B. Shao, H. Wang, and Y. Xiao, "Managing and mining large graphs: Systems and implementations," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 589–592. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213907

[26] L. M. Vaquero, F. Cuadrado, and M. Ripeanu, "Systems for near real-time analysis of large-scale dynamic graphs," in *Submitted for publication*, ser. xxx'14, 2014.

[27] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, Aug. 1990. [Online]. Available: http://doi.acm.org/10.1145/79173.79181

[28] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/MSST.2010.5496972

[29] S. A. Crosby and D. S. Wallach, "An analysis of bittorrents two kademlia-based dhts," 2007.

[30] C. Gkantsidis, M. Mihail, and E. Zegura, "Spectral analysis of internet topologies," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1, March 2003, pp. 364–374 vol.1.

[31] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 22–22. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[32] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633

[33] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, ser. Middleware '01. London, UK, UK: Springer-Verlag, 2001, pp. 329–350. [Online]. Available: http://dl.acm.org/citation.cfm?id=646591.697650

[34] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, Feb. 2003. [Online]. Available: http://dx.doi.org/10.1109/TNET.2002.808407

[35] K. Xu, M. Song, X. Zhang, and J. Song, "A cloud computing platform based on p2p," in *IT in Medicine Education, 2009. ITIME '09. IEEE International Symposium on*, vol. 1, Aug 2009, pp. 427–432.

[36] L. Garces-Erice, E. Biersack, K. W. Ross, P. A. Felber, and G.Urvoy-Keller, "Hierarchical peer-to-peer systems," ser. Euro-Par 2003 Parallel Processing Lecture Notes in Computer Science, vol. 2790, 2003, pp. 1230–1239.

[37] A. Mandal, Y. Xin, I. Baldine, P. Ruth, C. Heerman, J. Chase, V. Orlikowski, and A. Yumerefendi, "Provisioning and evaluating multi-domain networked clouds for hadoop-based applications," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, Nov 2011, pp. 690–697.

[38] R. Cuevas, N. Laoutais, X. Yang, G. Siganos, and P. Rodriguez, "Bittorrent locality and transit traffic reduction: When, why and at what cost?" *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013.

[39] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 53–65. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687801

[40] A. Stage and T. Setzer, "Network-aware migration control and scheduling of differentiated virtual machine workloads," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, ser. CLOUD '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 9–14. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2009.5071527

[41] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, "Remedy: Network-aware steady state vm management for data centers," in *Proceedings of the 11th International IFIP TC 6 Conference on Networking - Volume Part I*, ser. IFIP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 190–204.

[42] N. Laoutaris, D. Carra, and P. Michiardi, "Uplink allocation beyond choke/unchoke: Or how to divide and conquer best," in *Proceedings of the 2008 ACM CoNEXT Conference*, ser. CoNEXT '08. New York, NY, USA: ACM, 2008, pp. 18:1–18:12. [Online]. Available: http://doi.acm.org/10.1145/1544012.1544030

[43] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive view of a live network coding p2p system," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 177–188. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177104

Luis M. Vaquero Luis holds a BSc in electronics, MSc in electrical engineering and PhD in telematics from the University of Valladolid (Spain). He also holds a MSc in Pharmacology and a PhD in Medicine from the University Complutense (Spain). After graduating he spent some time in a few U.S.-based research institutions and then working in building the private and public cloud of the third largest telecom in the world (Telefonica). He then joined HP Labs, where he is a senior researcher on large scale distributed systems and patent coordinator. As a part time activity he was adjunct assistant professor at URJC in Madrid Spain and visiting senior lecturer at Queen Mary University of London in the UK.

Antonio Celorio Antonio Celorio holds a MSc in Telecommunications Engineering from Universidad Politcnica de Madrid (Spain). He previously worked for HP Labs but has since joined the Center for Open Middleware in Madrid, a joint technology center by Universidad Politcnica de Madrid and the Santander Group

Felix Cuadrado Felix received a MEng in telecommunications engineering in 2005 and a PhD in telematics in 2009 from Universidad Politecnica de Madrid (Spain). He received a prize for the best PhD thesis in 2010 by the Spanish Association of Telecommunication Engineers. Since 2011 Felix is Assistant Professor at Queen Mary University of London, where he works on large-scale distributed systems, autonomic computing, Internet Application analytics, and software-defined networking.

Ruben Cuevas Ruben obtained his MSc and PhD in Telematics Engineering at University Carlos III of Madrid (Spain) in 2007 and 2010, respectively. He is currently Assistant Professor at the Telematics Engineer- ing Department at University Carlos III of Madrid. He has been research intern at Telefonica Research Lab and Courtesy Assistant Professor at University of Oregon in 2008 and 2012, respectively. He has been involved in several national and international research projects in the area of content distribution and green networking. Moreover, he is co-author of more than 30 papers in prestigious international journals and conferences such as IEEE/ACM TON, IEEE TPDS, ACM CoNEXT, IEEE Infocom or IEEE P2P. His main research interests include Content Distribution, P2P Networks, Online Social Networks and Internet Measurements.