Scalable Repositories for Virtual Clusters

Paolo Anedda, Simone Leo, Massimo Gaggero, and Gianluigi Zanetti

CRS4 Distributed Computing Group, Edificio 1, Polaris, Pula, Italy
{firstname.lastname}@crs4.it,
http://dc.crs4.it

Abstract. For a large class of scientific data analysis applications it is becoming important, due to the sheer size of datasets, to have the option to perform the analysis directly where the data are stored, rather than on remote computational clusters. A possible strategy is the use of virtual clusters, thus guaranteeing a high degree of isolation from the underlying physical computational structure, and a very compact initial description. Deploying, saving and restoring HPC dedicated virtual clusters introduces, however, a different class of requirements on the virtual machines managing infrastructure, in particular for what concerns storage I/O requirements, whose scalability boundaries are easily reached. Here we discuss an alternative approach based on a storage model that leverages the WORM (write once, read many) character of the data used by VM management to increase, in a scalable way, the aggregate data bandwidth available to virtual cluster level operations and provide preliminary results indicating that it is a viable solution.

1 Introduction

Current scientific data production technologies allow for the collection of huge datasets at a constantly decreasing price. Examples of research fields recently hit by what has been called the "data deluge" include geosciences with embedded networked sensing [1], life sciences with biomedical imaging [2] and high-throughput DNA sequencing [3]. Developing new technologies capable of properly managing these datasets constitutes a precondition for the efficient extraction of knowledge from the data and an interesting research problem in itself.

Applications for any nontrivial analysis of such datasets are usually parallel, and require large computing clusters to be effective. Due to their high installation and maintenance costs, large clusters are almost always shared between research groups with different, possibly conflicting software requirements, making their administration problematic. Moreover, when datasets reach this size, it is usually much more efficient to analyze them directly where they are stored, rather than moving them to a remote computational cluster.

A popular solution to this class of problems is OS-level virtualization, which allows to encapsulate applications, along with all their requirements down to the operating system, into virtual machines (VMs) thus guaranteeing a high degree

H.X. Lin et al. (Eds): Euro-Par 2009 Workshops, LNCS 6043, pp. 414–423, 2010.

of isolation and easy migration between different physical hosts. In particular, the paravirtualization approach, adopted by leading technologies like Xen [4,5], allows VMs to achieve extremely low performance overhead,[1] which makes them particularly attractive for running HPC applications. Since scientific data are typically analyzed by means of parallel applications, virtualization leads to computational entities which take the form of *virtual clusters* (VCs) [6], sets of VMs which are deployed and managed as single, self-consistent atomic entities.

A VM for data-driven applications like the ones cited above often needs to be described by a state (file system image and allocated memory) measuring several gigabytes. For example, a recently developed MapReduce [7] application [8] for the analysis of deep sequencing datasets employs VMs with 4 GB of RAM and more than 3 GB of disk space for local caching.

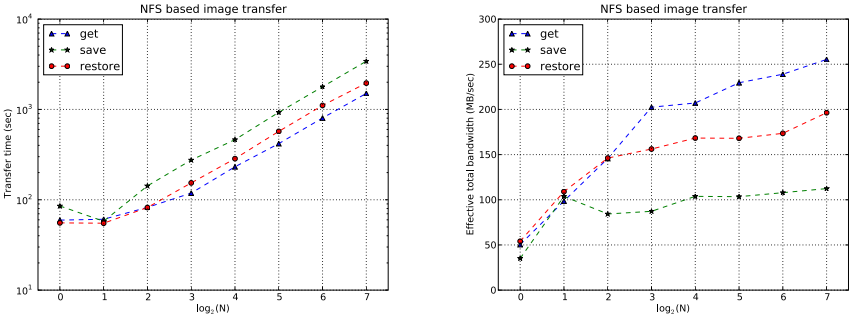

Fig. 1. Left: direct measurements of the total time taken to, respectively, *get* from a central NFS repository the same 3 GB image to N VM host nodes, *save* back to the repository N 3 GB images taken from the same N VM hosts and *restore* the N images from the repository to N VM hosts. Right: corresponding effective bandwidth defined as the total amount of data transferred divided by the total time taken by the transfer. The dashed lines are drawn only to guide the eye. The measurements were obtained using the setup described in section 4.1.

In this context, a virtual cluster's initial description is usually given in terms of N initially identical virtual machines, all of which are instantiated from the same disk image; subsequent events in a typical VC lifecycle include non-destructive save/restore and shutdown. Thus, while the initial cluster deployment involves transferring the same image to N VM hosts, other operations require the saving and, at restart, the restoring, of N different images from/to N different VM hosts. Fig. 1, on the left, shows direct measurements of the total time taken to, respectively, *get* from a central NFS repository the same 3 GB image to N VM host nodes, *save* back to the repository N 3 GB images taken from the same N VM hosts and *restore* the N images from the repository to N VM hosts. The right side of the same figure shows the corresponding effective bandwidth, defined as the total amount of data transferred divided by the total time taken

[1] `www.xen.org/about/paravirtualization.html`

by the transfer. The measurements were obtained using the setup described in section 4.1. All these processes are parallel, in the sense that all transfers to and from the VM hosts are started synchronously.

It is apparent from figure 1 that, while NFS is a perfectly adequate solution for small clusters, it will not scale as the number of nodes increases. Apart from the specifics of a given hardware setup, this is a direct consequence of having an external fixed storage system, whose bandwidth is independent from the computational cluster's size. On the other hand, NFS provides much more (e.g., supports arbitrary modification on files) than the simple streaming I/O required for handling VM images.

In this paper we discuss an alternative storage approach based on HDFS, the Hadoop[2] Distributed File System. Differently from general purpose, POSIX compliant, distributed file systems such as Lustre [9] and GPFS [10], HDFS is specialized to a model where files, once written, cannot be modified. This dramatically simplifies the management of distributed data coherence while guaranteeing high throughput for streaming applications. Although limited in general file system terms, HDFS is perfectly adequate to VC deployment and migration operations since the latter create what are, essentially, immutable data.

The main point here is that whatever the file system used, it should be able to use storage on the physical VM hosting cluster, and guarantee scalability. Our specific choice was HDFS – it could have been a similar storage system such as CloudStore[3] – because our main virtual cluster applications are Hadoop based and expect the hosting facility to export an HDFS file system.

Our preliminary results indicate that this is indeed a viable solution and that, at the cost of a moderate increase in the complexity of the VM hosts setup, it brings good scalability and dramatic performance improvements with respect to traditional, *out of the cluster* storage strategies.

The rest of the paper is organized as follows: section 2 discusses related work; in section 3 we briefly introduce HDFS; section 4 describes experimental setup and test results; finally, in section 5 we present our conclusions and plans for future work.

2 Related Work

Virtual clusters have been the subject of intense research activities in the past years. In [11], OS and network virtualization are used to partition machines into separate "virtual domains" in order to increase isolation between different organizational units and optimize resource utilization. In [6], a VC is defined as an aggregation of atomic virtual workspaces [12], implemented as sets of virtual machines. Maestro-VC [13] provides on-demand virtual clusters, implemented as sets of Xen VMs, which are deployed and managed as homogeneous entities. The problem of efficient and scalable VC installation is addressed in [14] by means

[2] `http://hadoop.apache.org`
[3] `http://kosmosfs.sourceforge.net`

of pipelined data transfer and automatic caching of frequently used VM images. In [15], VC deployment and configuration is discussed in an HPC context.

These works are mainly focused on extracting homogeneous, customized environments from a broader, possibly heterogeneous resource pool by means of automated installation frameworks. Although it shares similar goals, our work specifically focuses on deployment systems capable of supporting large images (in the order of several gigabytes) deployed on hundreds of cluster nodes.

In this work we use Hadoop HDFS as distributed VM image repository. Alternative solutions, often available in HPC cluster installations, include GPFS [10], PVFS [16] and Lustre [9]. However, given the WORM storage model that characterizes VC management, a specialized file system capable of harnessing it to maximize performance was the natural choice. Concurrent to our work, in an effort to support MapReduce on GPFS, IBM researchers have shown [17] how a standard cluster file system, after undergoing substantial modification, could achieve performances comparable to those of more specialized file system like HDFS, albeit with increased network traffic.

3 Technologies

The main technologies used in the work described here are HDFS and NFS. While the latter, being widely adopted, is well known, the former deserves a short introduction which we will present in the next section.

3.1 HDFS

Hadoop[4] is a popular open source implementation of MapReduce [7], a parallel programming framework initially developed by Google. Hadoop includes a distributed file system for application data storage called HDFS (Hadoop Distributed File System). HDFS has been specifically designed for very high scalability (thousands of nodes, hundreds of millions of files, tens of petabytes) and optimized for high throughput processes on very large datasets.

Its key features are:

- fault tolerance: data blocks are replicated according to a configurable replication factor so that the MapReduce engine can reassign failed tasks to other nodes;
- WORM (write once, read many) storage model: once a file is written, it can never be modified. This allows to maximize the aggregate bandwidth without resorting to complex synchronization mechanisms.

HDFS adopts a master/slave architecture: the master, called *namenode*, stores file system metadata and provides a namespace which allows groups of data blocks to be seen as ordinary files; the slaves, called *datanodes*, physically store data blocks and serve read/write requests from clients. By design, user data never flows through the namenode: when a client requests a file, the namenode

[4] `http://hadoop.apache.org`

simply replies with a set of block IDs and the addresses of the datanodes on which those blocks are stored; actual data transfer happens between the client(s) and the datanodes.

The usual HDFS application is to support the MapReduce computational framework, typically deployed over the same worker nodes as HDFS. Specifically, MapReduce, in its Map phase (where all data records are independent) uses information on data block location to optimize bandwidth to computation by scheduling tasks closest (in a network topology sense – e.g., on the same physical nodes or in the same rack) to where blocks are stored and minimize network traffic: by doing this it is possible, in principle, to reach an effective bandwidth that scales with the number of datanodes.

In the context of this paper, however, we are using HDFS in a different way. We are essentially interested only in reading and writing large numbers of VM image files, concurrently from multiple clients. Thus, while in its usual application HDFS clients independently read different data chunks corresponding to the different sections of the MapReduce input stream, in the VM repository case all clients read/write in sequence the data blocks that constitute each file. In this case parallelism is governed by a pipeline where each stage corresponds to a block and uses a group of datanodes to serve the operation. Therefore, even in the hypothetical case of an infinite size cluster, the depth of the pipeline controls the maximum bandwidth that can be achieved in a *get* operation where N clients concurrently download the same image file from HDFS. A simple model yields the following for the total time T needed to download from an HDFS cluster of S nodes to N clients a file of size $W = Bb_s$, where b_s is the HDFS block size and B the number of blocks, in the limit of $S \geq B$ and large N:

$$T = \frac{W}{b_w}\left(n_t + \frac{t_l}{\tau}\right)\left(1 + \frac{N - n_t R}{n_t RB}\right), \tag{1}$$

where b_w is the point-to-point bandwidth between a client and a datanode, n_t the number of server threads per datanode (in Hadoop, n_t defaults to 3), $\tau = b_s/b_w$ is the time needed to read a block of size b_s, t_l the latency involved in starting a block read, which ranges from tens to hundreds of milliseconds [18], while R is the HDFS block replication factor and by large N we mean $N \gg n_t R$. Eq. 1 predicts an essentially N-independent transfer time for $(N - n_t R) \ll n_t RB$. Accordingly, the effective bandwidth $E_{bw} = NW/T$ will initially grow linearly with N and then saturate at about RBb_w. Of course, this should be regarded as an indication of a general trend: the picture is more complicated for N smaller than $n_t R$ and large N behaviour is also controlled by the co-location of server and client processes and network effects. Differently from *get*, the many-to-many operations *save* and *restore* are in principle parallel since they involve N independent pipelines. However, their scalability is limited by finite size effects when the number of client and server datanodes becomes comparable with the hosting cluster size, especially because of competition for disk access.

4 Experimental Results

We have conducted a series of experiments to assess the feasibility of employing HDFS as a VC repository system by running various configurations on a medium size production cluster.

4.1 Setup

Image transport tests were conducted on a cluster of 384 HP BL460c blades equipped with two quad-core Intel E5440 (2.8 GHz) CPUs, 16 GB of RAM, two 250 GB SATA hard disks and two BCM5708S Gigabit Ethernet NICs. The

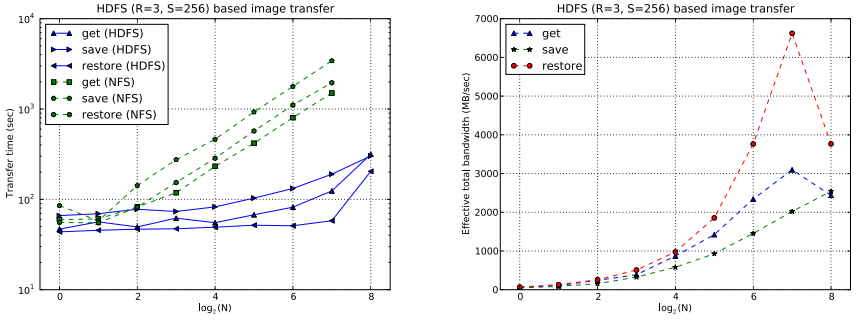

Fig. 2. Left: direct measurements of the total time taken to, respectively, *get* from a distributed HDFS repository the same 3 GB image to N VM host nodes, *save* back to the HDFS repository N 3 GB images taken from the same N VM host nodes, and *restore* the N images back from the repository to N VM hosts. As a matter of comparison, the figure also reports analogous measurements done using the NFS-based repository. HDFS was run with the default settings, i.e., block size equal to 64 MB and replication factor set to three. Right: corresponding effective bandwidth defined as the total amount of data transferred divided by the total time taken by the transfer. The lines are drawn only to guide the eye. The measurements were obtained using the setup described in section 4.1.

computing blades are contained, in groups of 16, in blade enclosures, in turn interconnected by 10GbE links to a central switch so that, albeit with potential latency hits, the cluster can support 1GbE wire speed interconnections between any arbitrary pair of blades.

The cluster is a shared production environment managed with Sun Grid Engine (SGE) [19]: to instantiate HDFS on it, we modified Hadoop On Demand[5] in order to use SGE (instead of its default, TORQUE) as its resource manager.

The external storage system used for NFS tests is a Sun StorageTek 5320 NAS Appliance which mounts 400 GB fiber channel and 1 TB SATA disks for a total capacity of 460 TB.

[5] `http://hadoop.apache.org/core/docs/current/hod_user_guide.html`

4.2 Measurements

In the following, we will use N to indicate the number of VC nodes and S to denote the number of nodes of the hosting physical cluster and thus of the HDFS file system. All scaling measurements are performed on the aforementioned production cluster that did not, at that time, support virtualization. On the other hand, here we are only considering image transport to/from the HDFS repository to the virtualization hosts and thus the reported measures are expected to be relevant to a full VC hosting setup. All measurements are the results of averages on multiple runs and were done while trying to minimize, as much as possible, the impact of external effects such as node sharing with other jobs and conflicts on network resources. The results can, therefore, be considered as best case data that, however, are expected to provide relevant general information on an actual VC hosting production configuration.

Fig. 2 shows, on the left, the results of direct measurements of the total time taken to, respectively, *get* from a distributed HDFS repository the same 3 GB image to N VM host nodes, *save* back to the HDFS repository N 3 GB images taken from the same N VM host nodes, and *restore* the N images back from the repository to N VM hosts. As a matter of comparison, the figure also reports analogous measurements done using the NFS based repository. HDFS was run with the default settings, i.e., block size equal to 64 MB and replication factor set to three. Unless specifically mentioned, we maintained these default settings for all the reported measurements. The simulated physical hosting cluster contained 256 machines, two of which were dedicated to the HDFS and MapReduce masters (respectively the namenode and the job tracker, with the latter unused), while the remaining 254 acted as HDFS slaves (datanodes). HDFS was configured to use only one of the two disks available on each node, while all per-host image read and write operations used the other one. As it can be seen from the figure, the two groups of curves start differing significantly for cluster sizes larger than four. The HDFS repository has a very weakly N-dependent behaviour up to VC clusters of size 32, after which competition between HDFS server and client processes starts to be relevant. It should be noted, however, that HDFS is still at least one order of magnitude better than NFS. On the right side of fig. 2 is shown the corresponding effective total bandwidth, defined as the total amount of data transferred divided by the total time taken by the transfer. As in the previous case, there is a finite size effect for large VC sizes.

Fig. 3 shows how the time taken by an image transfer for a given cluster size N depends on the size S of the physical hosting cluster (and thus of the HDFS repository). On the left is shown the case of the 1-to-N *get* operation. Transfer times seem to rapidly converge to their asymptotic, S-independent, values. Things are qualitatively different, as shown on the right side of fig. 3, for the N-to-N transport operation required by *restore*. Convergence is slower, probably due to competition for chunk reading on the datanodes.

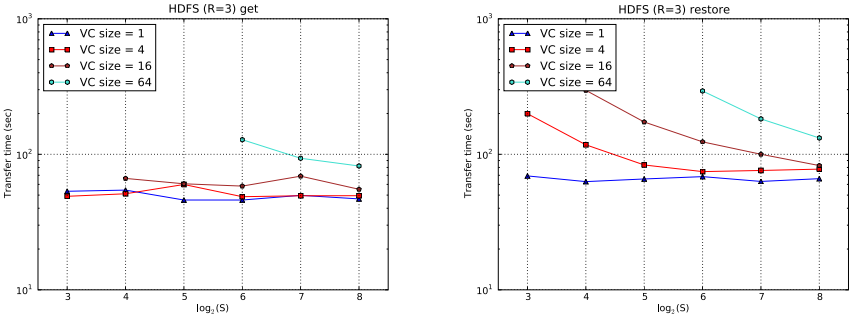

Fig. 3. Dependency of transfer time on the size of the physical hosting cluster S (and thus of the HDFS repository) for different VC size (N) values. Left: the 1-to-N *get* operation. Right: the N-to-N transport operation required by *restore*. Note the difference in convergence to the asymptotic values. The lines are drawn only to guide the eye.

Fig. 4 shows transfer timings – respectively *get* (left), *save* (center) and *restore* (right) – for a fixed physical hosting cluster size $S = 256$, as a function of VC size N for three different replication factors. Note how, as expected, the save operation is the most affected by R, especially as N becomes comparable to S.

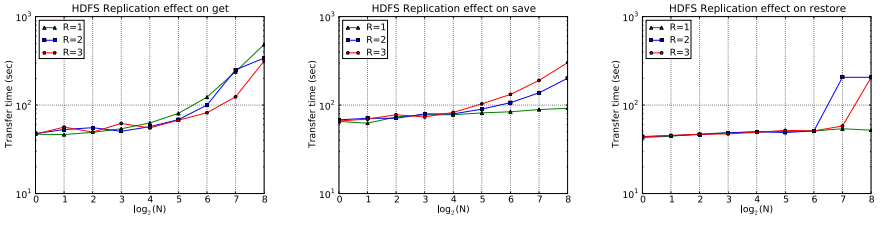

Fig. 4. *get* (left), *save* (center) and *restore* (right) timings for a fixed physical hosting cluster size $S = 256$, as a function of VC size N for three different replication factors. The lines are drawn only to guide the eye.

5 Conclusions and Future Work

We have shown that by using HDFS, a simple specialized distributed file system, it is possible, at the cost of a moderate increase in the complexity of the VM hosts setup, to provide – in a scalable way – the aggregate data bandwidth needed by HPC virtual cluster level management operations.

Measurements were obtained, as much as possible, while trying to minimize the impact of external effects such as node sharing with other jobs and conflicts on network resources. Although the results reported here can, therefore, be considered as best-scenario data, we expect them to provide relevant general information on actual VC hosting production configurations.

Future work will concentrate on analyzing the impact of the proposed solution on production virtual clusters. We also intend to compare HDFS with other distributed file systems (e.g., CloudStore) and to explore issues related to dynamical modifications in size and topology of the hosting physical cluster, both through modeling and experiments.

Acknowledgments. The work described here was partially supported by the Italian Ministry of Research under the CYBERSAR project. We would like to thank L. Leoni, C. Podda, M. Moro and M. Vocale for providing us with precious technical advice. We would also like to thank the anonymous reviewers for helpful comments and suggestions.

References

1. Borgman, C.L., Wallis, J.C., Mayernik, M.S., Pepe, A.: Drowning in data: digital library architecture to support scientific use of embedded sensor networks. In: 7th ACM/IEEE-CS joint conference on Digital libraries (2007)
2. Peng, H.: Bioimage informatics: a new area of engineering biology. Bioinformatics 24(17), 1827–1836 (2008)
3. Editorial: Prepare for the deluge. Nature Biotechnology 26(10), 1099 (2008)
4. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: 19th ACM Symposium on Operating Systems Principles (2003)
5. Chisnall, D.: The Definitive Guide to the Xen Hypervisor. Prentice-Hall, Englewood Cliffs (2007)
6. Foster, I., Freeman, T., Keahey, K., Scheftner, D., Sotomayor, B., Zhang, X.: Virtual clusters for grid communities. In: 6th IEEE International Symposium on Cluster Computing and the Grid (2006)
7. Dean, J., Ghemawat, S.: MapReduce: Simplified DataProcessing on Large Clusters. In: OSDI 2004: Sixth Symposium on Operating System Design and Implementation (2004)
8. Leo, S., Anedda, P., Gaggero, M., Zanetti, G.: Using virtual clusters to decouple computation and data management in high throughput analysis applications
9. Schwan, P.: Lustre: building a file system for 1000-node clusters. In: Proceedings of the 2003 Linux Symposium (2003)
10. Schmuck, F., Haskin, R.: GPFS: a shared-disk file system for large computing clusters. In: Proceedings of the First Conference on File and Storage Technologies (FAST), pp. 231–244 (2002)
11. Ruth, P., McGachey, P., Xu, D.: VioCluster: Virtualization for dynamic computational domains. IEEE International Cluster Computing (2005)
12. Keahey, K., Foster, I., Freeman, T., Zhang, X., Galron, D.: Virtual Workspaces in the Grid. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 421–431. Springer, Heidelberg (2005)
13. Kiyanclar, N., Koenig, G., Yurcik, W.: Maestro-VC: A paravirtualized execution environment for secure on-demand cluster computing. In: 6th IEEE International Symposium on Cluster Computing and the Grid Workshops (2006)
14. Nishimura, H., Maruyama, N., Matsuoka, S.: Virtual clusters on the fly – fast, scalable, and flexible installation. In: 7th IEEE International Symposium on Cluster Computing and the Grid (2007)

15. Begnum, K., Disney, M.: Scalable Deployment and Configuration of High-Performance Virtual Clusters. In: 3rd International Conference on Cluster and Grid Computing Systems (2006)
16. Carns, P., Ligon III, W., Ross, R., Thakur, R.: PVFS: a parallel file system for linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference (2000)
17. Ananthanarayanan, R., Gupta, K., Pandey, P., Pucha, H., Sarkar, P., Shah, M., Tewari, R.: Cloud analytics: do we really need to reinvent the storage stack? In: Workshop on Hot Topics in Cloud Computing (HotCloud '09) (2009)
18. Lin, J., Bahety, A., Konda, S., Mahindrakar, S.: Low-latency, high-throughput access to static global resources within the Hadoop framework. Technical Report HCIL-2009-01, University of Maryland, Human-Computer Interaction Lab. (2009)
19. Gentzsch, W.: Sun grid engine: Towards creating a compute power grid. In: First IEEE/ACM International Symposium on Cluster Computing and the Grid (2001)