DataSteward: Using Dedicated Compute Nodes for Scalable Data Management on Public Clouds

Radu Tudoran, Alexandru Costan, Gabriel Antoniu

DataSteward: Using Dedicated Compute Nodes for Scalable Data Management on Public Clouds

Radu Tudoran*, Alexandru Costan†, Gabriel Antoniu*

*INRIA Rennes - Bretagne Atlantique, France
{radu.tudoran, gabriel.antoniu}@inria.fr

†IRISA / INSA Rennes, France
alexandru.costan@irisa.fr

***Abstract*—A large spectrum of scientific applications, some generating data volumes exceeding petabytes, are currently being ported on clouds to build on their inherent elasticity and scalability. One of the critical needs in order to deal with this "data deluge" is an efficient, scalable and reliable storage. However, the storage services proposed by cloud providers suffer from high latencies, trading performance for availability. One alternative is to federate the local virtual disks on the compute nodes into a globally shared storage used for large intermediate or checkpoint data. This collocated storage supports a high throughput but it can be very intrusive and subject to failures that can stop the host node and degrade the application performance. To deal with these limitations we propose DataSteward, a data management system that provides a higher degree of reliability while remaining non-intrusive through the use of dedicated compute nodes. DataSteward harnesses the storage space of a set of dedicated VMs, selected using a topology-aware clustering algorithm, and has a lifetime dependent on the deployment lifetime. To capitalize on this separation, we introduce a set of scientific data processing services on top of the storage layer, that can overlap with the executing applications. We performed extensive experimentations on hundreds of cores in the Azure cloud: compared to state-of-the-art node selection algorithms, we show up to a 20% higher throughput, which improves the overall performance of a real life scientific application up to 45%.**

I. Introduction

With big-data processing and analysis dominating the usage of cloud systems today, the need for services able to support applications that generate intense storage workloads increases. Commercial clouds feature a large variety of specialized storage systems, targeting binary large objects. However, these object stores are often optimized for high-availability rather than high performance, not to mention that they incur costs proportional to the I/O space and bandwidth utilization. One alternative is to rely on the local storage available to the virtual machine (VM) instances where the application is running. In a typical cloud deployment, this local storage is abundant, in the order of several hundreds of GBs, available at no extra operational costs. Building on data locality, these storage solutions collocated on the application nodes achieve I/O throughputs up to an order of magnitude higher [1] compared to the remote cloud storage.

Despite its evident advantages, collocated storage has some important issues. First, it relies on commodity hardware that is prone to failures. An outage of the local storage system would make its host compute node inaccessible, effectively leading to loss of data and application failure. Second, the software stack that enables the aggregation of the local storage can become rather intrusive and impact on the application's perceived performance.

We propose to *decouple storage and computation* through the use of *dedicated compute nodes* for storage, as a means to address the previous issues. This separation allows applications to efficiently access data without interfering with the underlying compute resources while the data locality is preserved as the storage nodes are selected within the same cloud deployment. The idea of using dedicated infrastructures for performance optimization is currently explored in HPC environments. Dedicated nodes or I/O cores on each multicore SMP node are leveraged to efficiently perform asynchronous data processing and I/O, in order to avoid resource contention and minimize I/O variability [2]. However, porting this idea to public clouds in an efficient fashion is challenging, if we consider their multi-tenancy model, the consequent variability and the use of unreliable commodity components.

In this paper we introduce DataSteward, a system that federates the local disks of the dedicated VMs into a globally-shared data store in a scalable and efficient fashion. The dedicated nodes are chosen using a clustering-based algorithm that enables a *topology-aware selection*. With DataSteward, applications can sustain a *high I/O data access throughput*, as with collocated storage, but with *less overhead* and *higher reliability through isolation*. This approach allows to extensively use *in-memory storage*, as opposed to collocated solutions which only rely on virtual disks. To capitalize on this separation, we provide a set of higher-level data-centric services, that can *overlap with the computation* and reduce the application runtime. Compression, encryption, anonymization, geographical replication or broadcast are examples of *data processing-as-a-service* features that could exploit a dedicated infrastructure and serve cloud applications as a "data steward".

Our contributions can be summarized as follows:

- We present an overview of the data storage options in public clouds. (Section II-A)

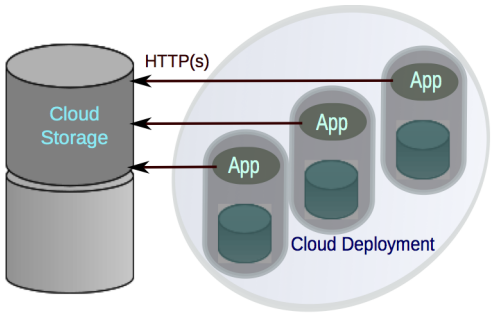

Figure 1. The default interaction between applications running in the compute nodes and the data hosted by the cloud storage services.

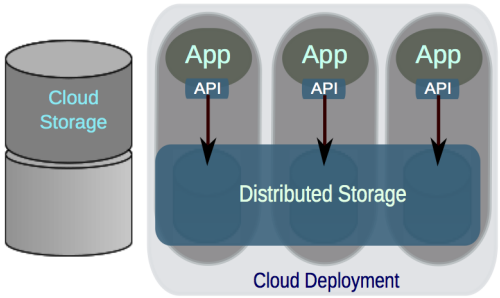

Figure 2. A typical deployment of a distributed file system (DFS) that leverages the virtual disks of the compute nodes.

- We introduce DataSteward, a data management system that leverages dedicated compute nodes to efficiently address the reliability and intrusiveness issues of colocated cloud storage. We show how to implement this approach in practice in the Azure cloud. (Section II)
- We propose a clustering algorithm for dedicated nodes selection, that also gives an intuition of the cloud internal details. (Section III)
- We evaluate DataSteward experimentally on hundreds of cores of the Azure cloud, both with synthetic benchmarks and a real-life application. These experiments demonstrate up to a 45% speedup when compared to state of the art. (Section IV)

II. A STORAGE SERVICE ON DEDICATED COMPUTE NODES

A. Approaches to data storage on clouds

Currently, the options for sharing data are the cloud object stores, the distributed file-systems deployed on the compute nodes or some hybrid approaches between these two. Each one is geared for various types of data and maximizes a different (typically conflicting) set of constraints. For instance, storing data within the deployment increases the throughput but has an ephemeral lifetime, while using the cloud storage provides persistence at the price of high latencies.

1) Cloud store services: The standard cloud offering for sharing application data consists of storage services accessed by compute nodes via simple, data-centric interfaces. Under the hood, these are distributed storage systems running complex protocols to ensure that data is *always available* and *durable*, even when machines, disks and networks fail [3]. This makes them good candidates for persistently storing *input/output data*.

Most of these services, however, focus on data storage primarily and support other functionalities essentially as a "side effect" of their storage capability. They are accessed through high-latency REST interfaces both by cloud users and by VMs, making them inadequate for data with high update rates. Typically, they are not concerned by achieving high throughput, nor by potential optimizations, let alone offer the ability to support different data services (e.g. geographically distributed transfers, placement etc.) Our work aims to specifically address these issues.

2) Colocating data and computation: An alternative to the cloud store services consists of leveraging the locally-available storage on compute nodes to share application-level data. This idea is one of the founding principles for some scalable distributed file systems, which physically integrate the local disks of the nodes where the application runs. Exploiting *data locality*, this approach provides *high throughput* at potentially *no additional cost*.

The dynamic and ephemeral environment in which VMs and services are continuously being created and destroyed opens the avenue for such deployment specific storage solutions. They build on two observations: 1) compute nodes usually do not store persistent state, though they can store soft session state and be "sticky" with respect to individual user sessions; 2) in most cloud deployments, the disks locally attached to the compute nodes are not exploited to their full potential. This creates an opportunity to aggregate the storage space and I/O bandwidth of the nodes running the application, in order to build a low-cost data-store.

In order to support data persistency, some hybrid approaches combine colocated storage with the cloud stores. For a given request to data, they favor access to the local globally shared storage first, if possible; if not, the operation translates into access to the remote storage through the network. Such architectures usually assume high-performance communication between computation nodes and the cloud store service. This assumption does not hold on current clouds, which exhibit much larger latencies between the compute and the storage resources.

Clearly, storing the data and running the applications on the same VMs avoids a level of indirection. However we think such close coupling is quite limiting. On the one hand, it can become intrusive for the application when handling intensive I/Os; on the other hand, it is error-prone since a failure on the storage node affects the whole computation.

B. Design principles

Our proposal relies on four key design principles:

Dedicated compute nodes for storing data. This approach preserves the data proximity within the deployment and increases the application reliability through isolation. Keeping data management within the same compute infrastructures (i.e. same racks, switches) preserves the cluster

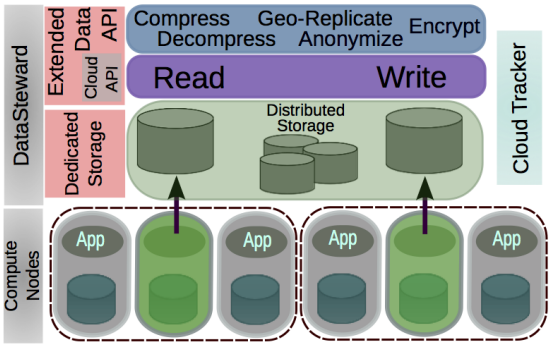

Figure 3. DataSteward architecture with dedicated compute nodes for data storage and a set of additional data processing functions.

bandwidth as the intermediate routing layers are minimized.

Topology awareness. This design needs some mechanism to ensure that the dedicated storage servers are located as ”close” as possible (in terms of bandwidth and latency) to the rest of the computing nodes for efficiency, while not being dependent on such tight coupling.

No modification of the cloud middleware. Since the storage system is deployed inside the VMs, the cloud middleware is not altered. Previous works on aggregating the physical disks attached to the compute nodes [4] impose modifications to the cloud infrastructure, so they only worked with open source cloud kits. Our solution is suitable for both public and private clouds, as no additional privileges are required.

Overlapping computation with data processing. With this dedicated approach, we are able to propose a set of data processing services (e.g. compression, encryption), hosted on the dedicated nodes and supporting scientific applications. In a typical cloud deployment, users are responsible for implementing these higher level functions on data, which translates in stalling the computation for such processing. We offload this overhead to the dedicated nodes and provide a data processing toolkit and an easily extensible API.

C. Architecture

Our proposal relies on a layered architecture, built around 3 components, presented in Figure 3.

The Cloud Tracker has the role of selecting the dedicated nodes through a 4 steps process executed at the VM’s booting time. First, a leader election algorithm is run, using the VM IDs. Second, each VM makes an evaluation of the network links and reports the results back to the leader, using a queue based communication system. Third, the leader runs the clustering algorithm described in Section III to select the most fitted nodes for storage (throughput wise). Finally, these nodes are broadcasted into the deployment.

The Distributed Storage is the data management system deployed in the dedicated nodes, that federates the local disks of the VMs. Users can select the distributed storage system of their choice.Additionally, the local memory is aggregated into an *in-memory storage*, used for storing, caching and buffering data. The Distributed Storage can dynamically scale up/down, dedicating new nodes when faced with data usage bursts or releasing some of them.

The Data Processing Services are a set of data handling operations, provided by DataSteward and targeting scientific applications. Examples include: compression, geographical replication, anonymization, etc. Their role is explained in Section V.

D. Implementation

The **Cloud Tracker** assesses the network capacities using the iperf tool [5]: it starts an iperf server on each VM, that measures the throughput of the links with all other VMs. Cloud Trackers communicate with each other to send their measurements results, using a queueing system. The elected leader runs the clustering algorithm on the collected network data and advertises the results, using the same queue-based system. We rely on the visibility timeout of the queues to guarantee that a result will not be lost and will be eventually used by the VM. The messages read from the queues are not deleted, but instead hidden until an explicit delete is received after a successful processing. If no such confirmation arrives, the message will become visible again in the queue, after a predefined timeout. With this approach, the dedicated nodes selection is protected from unexpected node crashes.

The **Distributed Storage** can be implemented on top of any storage system that may be deployed in a cloud. We have chosen BlobSeer [6], motivated by several factors. First, BlobSeer enables a scalable aggregation of the virtual disks of the dedicated nodes, for storing binary large objects (BLOBs) with minimal overhead. Second, its versioning-oriented design enables an efficient lock-free access to data. Finally, it uses a decentralized metadata schema which enables high throughput under heavy concurrency.

BlobSeer consists of multiple entities which are distributed across the dedicated nodes within our implementation. *Data providers* store the chunked data in-memory or on the VM disks; new providers may be elastically added or removed. *The provider manager* keeps information about the storage capacity and schedules data placement, according to a load balancing strategy. *Metadata providers* implement a DHT and store the information about the data blocks that make up a version. *The version manager* assigns and publishes version numbers, providing serialization and atomicity properties.

The **Data Processing Services** are exposed to applications through an API providing an abstract set of operations (currently available in C#), independent of the distributed storage chosen by the user. The API extends a set of cloud-based interfaces, which make the data access transparent to applications. The data processing services are loaded dynamically, from the default modules or from libraries provided by the users. Two such services are:

- *Geographical replication.*Data movements are time and resource consuming and it is inefficient for applications

Algorithm 1 Initialization of the clustering algorithm.

Input
$Nodes = \{node_1..node_N\}$ ▷ the set of compute nodes
$Client_{Clusters}[] = \{List_1..List_{NrOfDataServers}\}$ ▷ the set clients grouped in clusters
Output
$Servers = \{server_1..server_{NrOfDataServers}\}$ ▷ the set of data servers - the cluster centroids
$clients\_per\_server = N/NrOfDataServers$;
▷ Initialize the centroids randomly
for $i \leftarrow 0, NrOfDataServers$ **do**
  Servers $\leftarrow node \in Nodes$ (random selected)
end for

to stall their execution in order to perform such operations. DataSteward provides an alternative, as the applications can simply check-out their results to the dedicated nodes (through a low latency data transfer, within the deployment). Then, DataSteward performs the time consuming geographical replication, while the application continues interrupted.

- *Data compression.* Typically, the parallelization of scientific applications in multiple tasks leads to the output of multiple results. Before storing them persistently, one can decrease the incurred costs through compression. Grouping together these results on the dedicated nodes, we are able to achieve higher compression rates, than with results compressed independently on source nodes.

III. Zoom on the dedicated nodes selection

As the selection of the storage VMs can significantly impact application performance, we believe that the topology and utilization of the cloud need to be carefully considered to come up with an optimized allocation policy. Since cloud users do not have fine-grained visibility into or control over the underlying infrastructure, they can only rely on some application-level optimization. In our case, the storage nodes are selected based on the (discovered) topology, such that the aggregated throughput from the application nodes to the dedicated storage nodes is maximized. To get an intuition of the cloud topology, we opted for a *clustering algorithm*, observing that the cloud providers tend to distribute the compute nodes in different fault domains (i.e. behind multiple switches). Hence, we aim to discover these clusters based on the proximity that exists between the nodes in a fault domain. To this end, we fitted the clustering algorithm with adequate hypotheses for centroid selection and nodes assignment to clusters, in order to maximize the data throughput. Finally, the selection of the dedicated nodes is done based on the discovered clusters.

Clustering algorithms are widely used in various domains ranging from data mining to machine learning or bioinformatics. Their strength lies in the adequate hypotheses

Algorithm 2 Clustering-based dedicated nodes selection

procedure DEDICATENODES($NrOfDataServers, N$)
  repeat
    ▷ Phase 1: Assign nodes to clusters
    for $i \leftarrow 0, N$ **do**
      if i $\notin$ Servers **then**
        for $j \leftarrow 0, NrOfDataServers$ **do**
          if $throughput[Servers[j], i] > max$ && $Client[j].Count < clients\_per\_server$ **then**
            update_max();
          end if
        end for
        Client[maxserver].Add(i)
      end if
    end for
    ▷ Phase 2: Centroid Selection
    for $i \leftarrow 0, NrOfDataServers$ **do**
      for all $j \in Client[i]$ **do**
        if $j.variability < ADMITED\_STD$ and $j.bandwidth > ADMITED\_THR$ **then**
          for all $k \in Client[i]$ **do**
            server_thr $+=$ throughput[j,k]
          end for all
          update_max();
        end if
      end for all
      if $Servers[i] \neq maxserver$ **then**
        Servers[i] $\leftarrow$ maxserver
        changed $\leftarrow$ true
      end if
    end for
  until changed $\neq$ true
end procedure

used for creating the clusters (i.e. assigning an element to a cluster) and for representing them (i.e. selecting the centroids). In our scenario, the clusters are formed from compute nodes, and a cluster is represented by the node to be dedicated for storage (the centroid), with one dedicated node per cluster. The assignment of a node to a cluster is done based on the throughput to the data node that represents a cluster and by balancing the nodes between the data nodes:

$$cluster = \underset{i \in Servers}{\arg\max} \underbrace{\text{Max}\, throughput[i,j]}_{|Client[i]| < clients\_per\_server} \quad (1)$$

After creating the clusters, we update the centroids. With our hypothesis, we select the node that maximizes the throughput to all VMs in the cluster:

$$maxserver = \underset{j \in Client[i]}{\arg\max} \sum_{k \in Client[i]} throughput[j,k] \quad (2)$$

In Algorithm 1 we introduce the data structures used to represent the problem (the set of compute nodes, the set of dedicated nodes) and we initialize the centroids randomly. Algorithm 2 describes the 2 phases of the clustering process.

In the first phase, the compute nodes are assigned to the clusters based on the throughput towards the dedicated node and considering the theoretical load (the number of potential clients for a server). This phase implements Equation 1. The second step, consists in updating the centroids. We select the node which would provide the highest cumulated throughput per cluster, according to Equation 2. More, the nodes with poor QoS (low throughput or high variability) filtered.

IV. Experimental evaluation

This section evaluates the benefits of our proposal in synthetic settings and in the context of scientific applications.

A. Experimental setup

The experiments were carried out on the Azure cloud. We used compute nodes (Azure Worker Roles) and the cloud object storage service (Azure BLOBs) from two geographically distributed data-centers (North Europe and West US). The experimental setup consists of 50 up to 100 Medium Size VMs, each having 2 physical CPUs, 3.5 GB of memory, a local storage of 340 GB and an expected available bandwidth of 200 Mbps. The duration of an experiment ranges from several hours (for measuring the throughput) to several days (for measuring the network variability).

B. Assessing the cloud network variability

The network variability greatly impacts on the node selection. We want to avoid costly data movements in case of a reconfiguration of the participating nodes, due to the performance degradation of the underlying network. We are therefore interested in assessing the network stability in time, besides the (few) QoS guarantees of the cloud provider. The throughput measurements in the next 2 sections were done using iperf: we sent once every 5 minutes approximately 300 MB of data (with a TCP window size of 8 KB) between each VM; we re-executed this experiment in different days of the week during a 3 months period.

In Figure 4a, we show the variation of the network performance for 50 VMs, during a 24 hour interval. We notice that the average throughput *between all VMs* is stable and consistent with the QoS guarantees advertised by Azure (i.e. 200 Mbps for a Medium VM). A more detailed evaluation of the cloud jitter supporting these results was done in our previous work [1]. Building on this low variability of the Azure cloud, we are able to select the dedicated nodes once, at deployment start time and avoid further costly reconfigurations.

Figure 4b presents the average throughput from *each VM* to all others. Most VMs exhibit a high sustainable throughput, while a small fraction (3 out of 50) deliver a poor performance, due to multi-tenancy (e.g. VM with the ID 15). Luckily, such VMs can be easily identified and avoided, based on the initial measurements done by the Cloud Tracker. A compute node is then selected as a dedicated node only if its delivered throughput exceeds a specific threshold (currently set at 95%) with respect to the theoretical QoS and the tolerated network variability. This translates into the filtering step of the potential centroids of the clustering algorithm.

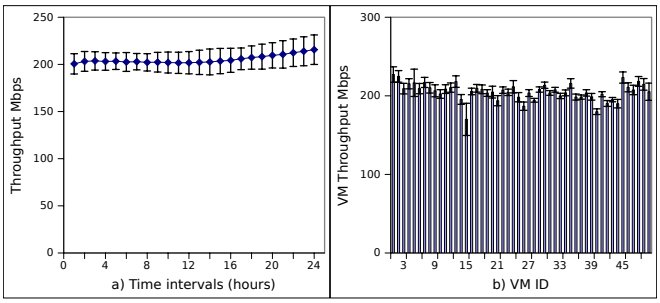

Figure 4. Evaluation of the network performance: a) The average throughput between all the compute nodes during 24h. b) The average throughput between a compute node (VM) and the other compute nodes.

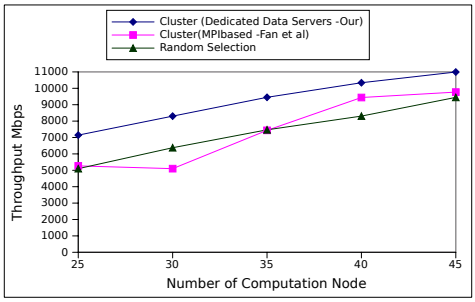

Figure 5. The cumulative throughput of the compute nodes when accessing the storage nodes, with varying number of data servers and compute nodes.

C. Clustering algorithm evaluation

In the next series of experiments, we compare our nodes selection strategy with the state of the art clustering algorithm introduced by Fan et al [7], using as distance the latency between nodes, and with a random selection, that doesn't consider the topology. Figure 5 shows the cumulated throughput from the compute nodes to the dedicated ones. The total number of nodes is fixed (50), and the number of dedicated nodes is decreased as the number of application nodes is increased. The average throughput *per node* is decreasing, when having less dedicated data nodes (i.e. the throughput increase ratio is inferior to the number of clients increase ratio). However, our approach is still able to handle this increasing number of concurrent clients, showing up to 20% higher *cumulative throughput*, compared to the other selection strategies.

Next, we assess how our decision of running the selection algorithm once, at deployment time, impacts on the overall system performance. We compare the aggregated throughput for a 24 hour interval when *the node selection is repeated every hour* with the achieved throughput when this selection is *fixed, done only once*, at the beginning of the experiment. The measured throughput is similar provided the network is relatively stable, so we can avoid node reconfiguration, which would incur expensive data movements. In both scenarios, our clustering algorithm outperforms the other

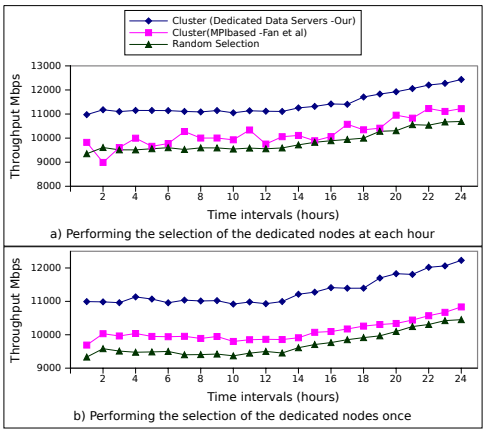

Figure 6. The cumulative throughput of the compute nodes when accessing the storage nodes, selected every hour (a) and selected once (b).

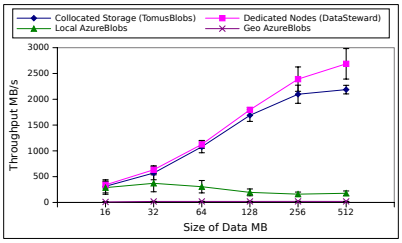

Figure 7. The cumulative read throughput with 50 concurrent clients.

strategies with more than 10%, capitalizing on its topology-awareness. Finally, we note that by filtering the problematic VMs (with a high variability), the cumulative throughput remains stable.

D. Data storage evaluation

Our next experiments evaluate the performance of the DataSteward storage layer in 3 different scenarios. We compare our approach with a storage system collocated on the application nodes (TomusBlobs), the local cloud storage (Local AzureBlobs) and a geographically remote storage from another data center (Geo AzureBlobs). TomusBlobs and DataSteward were each deployed on 50 compute nodes.

Scenario 1: Multiple reads / writes. We consider 50 concurrent clients that read and write increasing data sizes (ranging from 16 to 512 MB) from the memory to the storage system. For TomusBlobs, the clients are collocated on the compute nodes (1 client / node), while for DataSteward the clients run on 50 distinct machines. The cumulative throughput for the concurrent reads and writes is presented in Figures 7 and 8, respectively. Clearly, data locality has a major impact on the achieved throughput: storing data remotely on both types of dedicated cloud storage (AzureBlobs), the overall performance drops considerably.

A more interesting observation is made when zooming on the DataSteward and TomusBlobs performance. One might expect that, due to data locality, the collocated storage delivers slightly better performances than our approach, which moves the data to the dedicated nodes. However, our measurements show that for smaller amounts of data sent by the clients to the network, the performance of the

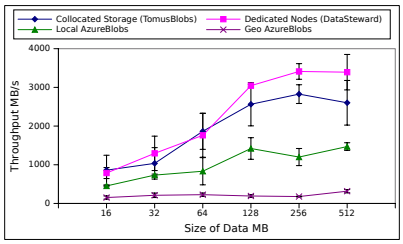

Figure 8. The cumulative write throughput with 50 concurrent clients.

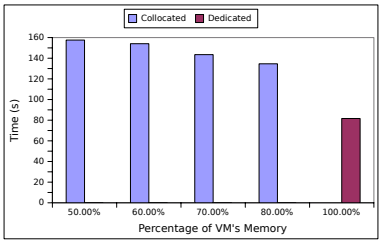

Figure 9. The execution time of the Kabsch-based application on a fixed set of input data. The percentage of the VM memory used by the application is increased, when using the collocated storage. The final bar represents the application execution time using all VM memory when the storage is moved to dedicated nodes

two solutions is similar, while for increasing data sizes our solution outperforms the collocated storage. This is due to two reasons. One the one hand, for both solutions the distributed storage splits data into chunks scattered across the federated virtual disks; so even with the collocated storage not all data written by a client remains in the same VM. Moreover, the throughput is determined both by the network bandwidth and by the CPU's capability to handle the incoming data, which is better exploited by our approach that separates computation from data. On the other hand, when the data size increases beyond 200 MB, a poor management of the network links between the VMs leads to a faster saturation of the network. Clustering the nodes with higher throughput, DataSteward is able to outperform the collocated storage by more than 10%.

Scenario 2: Memory usage for a scientific application. An important feature brought to cloud applications by our proposal is a better usage of their VMs resources (CPU, memory and disks). To evaluate this, we used a bio-informatics application running the Kabsch algorithm [8], which computes the optional rotation matrix that minimizes the root mean squared deviation between two sets of data. This application was chosen as it is representative for many scientific computations from fields like statistical analysis or cheminformatics for molecular structures comparison.

We executed the application with increasing amounts of memory available for the actual computation; the remaining memory was assigned to the collocated storage - TomusBlobs. We compared these execution times with the case in which all the VM memory is used for computation and the storage lies on dedicated nodes - DataSteward (Figure 9). With our approach, the application can harness all its local computing resources and its makespan (computation and data handling time) is reduced by half.

Scenario 3: Data processing services for a scientific application. Finally, we evaluate the two data processing services provided by the DataSteward top layer: data compression and geographical replication. We consider the data computed by the Kabsch-based application (having a size of approximately 100 MB), which is compressed and geographically replicated from Europe to the United States.

The execution times depicted in Figure 10 sum up the times to write the data to the storage system, compress it and transfer it to the remote data centers. With DataSteward, these operations are executed separately, non-intrusively, at the price of an extra transfer within the deployment, from the application nodes to the storage (the read, depicted in green). Nevertheless, the overall execution time is reduced by 15% as it is more efficient to compress all the aggregated data at one place than doing it independently on small chunks in the application nodes. Building on this grouping of data, we are also able to obtain up to 20% higher compression rates, by exploiting the similarities between the results obtained from multiple application nodes.

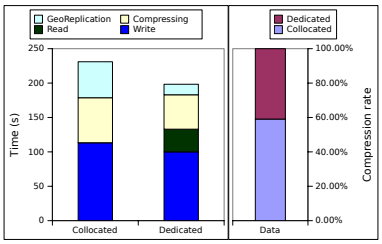

Figure 10. The time spent to compress and geogrpahically replicate 100 MB of data from the Kabsch-based application on the application nodes and with DataSteward (left). Buffering data from multiple nodes before compression and exploiting results similarities we are able to increase the compression rate (right).

V. Discussion

The motivation to use a dedicated infrastructure inside the deployment was to enable a set of *data services* that deliver the power and versatility of the cloud to users. The idea is to *exploit the compute capabilities* of the storage nodes to deploy data specific services, that otherwise couldn't be run on the application nodes The section introduces the services that could leverage such a dedicated storage infrastructure.

- **Cache for the persistent storage.** Its role would be to periodically backup into the cloud store service the data from the dedicated storage nodes. This approach complements DataSteward with persistency capabilities, following closely the structure of the physical storage hierarchy: machine memory, local and network disks, persistent storage.
- **Geographically-distributed data management.** Being able to effectively harness multiple geographically distributed data centers and the high speed networks interconnecting them has become critical for wide-area data replication as well as for federated clouds ("sky computing" [9]). A data transfer scheduling and optimization system hosted on dedicated cloud nodes will mitigate the large-scale end-to-end data movement bottleneck over wide-area networks.
- **Scientific data processing toolkit.** Scientific applications typically require additional processing of their input/output data, in order to make the results exploitable. For large datasets, these manipulations are time and resource consuming; using dedicated nodes, such processing can overlap with the main computation and significantly decrease the time to solution.
- **Cloud introspection as a service.** Building on the clustering scheme presented in Section III and the measurements done in Section IV-B, we can design an introspection service that could reveal information about the cloud internals to the interested applications. The ability to collect large numbers of latency, bandwidth and throughput estimates without actively transferring data provides applications an inexpensive way to infer the cloud's internal details.

The goal is to provide a software stack on top of the storage nodes, following a *data processing-as-a-service* paradigm. This "data steward" will be able, on one hand, to optimize the storage management and the end-to-end performance for a diverse set of data-intensive applications, and on the other hand, to prepare raw data issued/needed by experiments into a science-ready form used by scientists.

VI. Related work

A series of recent works focus on the idea of *leveraging the locally-available storage* on compute nodes to efficiently store and share application-level data. We introduced this approach in our previous work: TomusBlobs [10], a concurrency-optimized cloud storage system that federates the virtual disks into a globally-shared object store. GFarm [4] follows a similar approach but its file system runs on the host OS: this requires to specifically modify and extend the cloud middleware. Whereas this solution can be used with open-source IaaS clouds and on private infrastructures, our approach is different: the storage system can be deployed by the user within the VMs without modifying the cloud middleware and can thus be used on any cloud infrastructures, including commercial ones. Both these solutions rely on collocating storage and computation; in our current work, we advocate decoupling them through the use of dedicated storage nodes for increased reliability and less I/O variability.

A number of *scavenged storage systems* [11], [12], [13] aggregate idle storage space from collaborating nodes to provide a shared data store. Their basic premise is the availability of a large amount of idle disk space on nodes that are online for the vast majority of the time. Similarly to our approach, these highly configurable storage systems that harness node resources (although through collocation) are configured and deployed by users and have a lifetime

dependent on the deployment lifetime. However, they only rely on underutilized nodes and not dedicated ones.

The relationship between application and physical hardware is dynamic and unknown prior to the deployment. This has led some providers to introduce special classes of *attached storage volumes* such as the Amazon Elastic Block Store [14] and Azure Drives [15], which can be linked to virtual machines hosting storage intensive workloads. They give the possibility to mount the remote cloud storage directly as a virtual drive, which allows an easier access to the object storage. As our system, these volumes are placed in the same deployment (availability zone) with the user's instances but in order to share data one needs to create a file system on top of them.

One of our key objectives was to design an efficient scheme for *selecting the dedicated storage nodes*. Based on the cloud nodes' QoS performance, a number of selection and schedule strategies have been proposed in the recent literature. The major approaches can be divided into three categories. *Random* methods select nodes randomly and have been extensively employed in works like [16]. *Ranking* strategies rate the cloud nodes based on their QoS and select the best ones [17], [18]. The basic idea of these strategies is to cluster the cloud nodes that have a good communication performance together to deploy an application. We use a similar approach to cluster all the nodes within a deployment and then choose "leaders" from each cluster (best connected with all nodes within), which will make up the global dedicated storage nodes. In contrast to existing clustering solutions, we first discover the communication topology and the potential virtual network bottlenecks by pre-execution a series of measurements; then, we consider both the resource and topology properties of a node in a unified way to select the most suited ones. Different from existing works that only take into consideration the resource (e.g., CPU, bandwidth, or both) of a node while neglecting its topology properties, our work fills this gap.

VII. Conclusions

Many scientific applications leverage the VM local storage in order to achieve higher throughput and better scalability while reducing costs, however doing so raises intrusiveness and fault tolerance issues. In this paper we present DataSteward, a data management system that provides a higher degree of reliability while remaining non-intrusive through the use of dedicated nodes. DataSteward aggregates the storage space of a set of dedicated VMs, selected using a topology-aware clustering algorithm. The storage layer is augmented with a set of scientific data processing services that can overlap with the running applications. We demonstrated the benefits of our approach through extensive experiments performed on the Azure cloud. Compared to traditional node selection algorithms, we show up to 20% higher throughput, which improves the overall performance of a real life scientific applications up to 45%.

Encouraged by these results, we plan to complement this user perspective on the use of dedicated storage nodes, with a cloud provider's insight. In particular, we are interested by the means through which the public clouds can support and optimize such deployments: for instance providing an informed node selection scheme as a service or some special VM, fitted with extra types of storage (memory, SSD, disks).

Acknowledgment

This work was supported by the joint INRIA-Microsoft Research Center. The experiments presented in this paper were carried using the Azure Cloud infrastructure provided by Microsoft in the framework of the A-Brain project. The authors would like to thank the Azure teams from Microsoft Research ATLE for their valuable input and feedback.

References

[1] R. Tudoran, A. Costan, G. Antoniu, and L. Bougé, "A Performance Evaluation of Azure and Nimbus Clouds for Scientific Applications," in *CloudCP 2012 – 2nd International Workshop on Cloud Computing Platforms, Held in conjunction with the ACM SIGOPS Eurosys 12 conference*, Bern, Switzerland, 2012.

[2] M. Dorier, G. Antoniu, F. Cappello, M. Snir, and L. Orf, "Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O," in *CLUSTER - IEEE International Conference on Cluster Computing*, 2012.

[3] B. Calder and et al, "Windows azure storage: a highly available cloud storage service with strong consistency," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, 2011.

[4] K. H. O. Tatebe and N. Soda, "Gfarm grid file system," in *New Generation Computing: 257-275*.

[5] "Iperf," http://iperf.fr/.

[6] B. Nicolae, G. Antoniu, L. Bougé, D. Moise, and A. Carpen-Amarie, "BlobSeer: Next Generation Data Management for Large Scale Infrastructures," *Journal of Parallel and Distributed Computing*, 2010.

[7] P. Fan, Z. Chen, J. Wang, Z. Zheng, and M. R. Lyu, "Topology-aware deployment of scientific applications in cloud computing," in *IEEE CLOUD*, 2012.

[8] W. Kabsch, "A solution for the best rotation to relate two sets of vectors," in *Acta Crystallographica A , 32:922923*, 1976.

[9] K. Keahey, M. Tsugawa, A. Matsunaga, and J. A. Fortes, "Sky Computing," *Cloud Computing*, 2009.

[10] A. Costan, R. Tudoran1, G. Antoniu, and G. Brasche, "TomusBlobs: Scalable Data-intensive Processing on Azure Clouds," *Concurrency and Computattion: Practice and Experience*, 2013.

[11] A. Gharaibeh and M. Ripeanu, "Exploring data reliability tradeoffs in replicated storage systems," in *18th ACM international symposium on high performance distributed computing*, 2009.

[12] S. Al-Kiswany, A. Gharaibeh, and M. Ripeanu, "The case for a versatile storage system," *SIGOPS Oper. Syst. Rev.*, 2010.

[13] "dCache," http://www.dcache.org/.

[14] "Amazon Elastic Block Store," http://aws.amazon.com/ebs/.

[15] B. Calder and A. Edwards, "WINDOWS AZURE DRIVE," 2010, http://go.microsoft.com/?linkid=9710117.

[16] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, 2004.

[17] K. Budati, J. Sonnek, A. Chandra, and J. Weissman, "Ridge: combining reliability and performance in open grid platforms," in *Proceedings of the 16th international symposium on High performance distributed computing*, 2007.

[18] B. K. Alunkal, "Grid eigen trust a framework for computing reputation in grids," Ph.D. dissertation, Illinois Institute of Technology, 2003.