Adapting Data-Intensive Workloads to Generic Allocation Policies in Cloud Infrastructures

Ioannis Kitsos*, Antonis Papaioannou*, Nikos Tsikoudis*, and Kostas Magoutis
Institute of Computer Science (ICS)
Foundation for Research and Technology Hellas (FORTH)
Heraklion GR-70013, Greece
{kitsos,papaioan,tsikudis,magoutis}@ics.forth.gr

***Abstract*—Resource allocation policies in public Clouds are today largely agnostic to requirements that distributed applications have from their underlying infrastructure. As a result, assumptions about data-center topology that are built-into distributed data-intensive applications are often violated, impacting performance and availability goals. In this paper we describe a management system that discovers a limited amount of information about Cloud allocation decisions –in particular VMs of the same user that are collocated on a physical machine— so that data-intensive applications can adapt to those decisions and achieve their goals. Our distributed discovery process is based on either application-level techniques (measurements) or a novel lightweight and privacy-preserving Cloud management API proposed in this paper. Using the distributed Hadoop file system as a case study we show that VM collocation in a Cloud setup occurs in commercial platforms and that our methodologies can handle its impact in an effective, practical, and scalable manner.**

Keywords: Cloud management; distributed data-intensive applications.

I. INTRODUCTION

Cloud computing [3] has become a dominant model of infrastructural services replacing or complementing traditional data centers and providing distributed applications with the resources that they need to serve Internet-scale workloads. The Cloud computing model is centered on the use of virtualization technologies (virtual machines (VMs), networks, and disks) to reap the benefits of statistical multiplexing on a shared infrastructure. However, these technologies also obstruct the details of the infrastructure from applications. While this is not an issue for some applications, others require knowledge of the details of the infrastructure for reasons of performance and availability.

Several distributed data-intensive applications [6][7][10] replicate data onto different system nodes for availability. To truly achieve availability though, nodes holding replicas of the same data block must have independent failure modes. To decide which nodes are thus suitable to hold a given set of replicas, an application must have knowledge about data center topology, which is often hidden from applications or provided only in coarse form. Data center applications often try to reverse-engineer data center structure by assuming a simple model of infrastructure (e.g., racks of servers interconnected via network switches) and attempt to derive deployment details (i.e., whether two VMs are placed on the same physical machine, same rack, or across racks) via network addressing information. Most current Cloud-infrastructure services however completely virtualize network resources (i.e., there is no discernible mapping between virtual network assignments and physical network structure), thus effectively hiding their resource allocation decisions from applications.

Cloud management systems today use generic allocation algorithms such as round-robin across servers or round-robin across racks with the intent to spread a user's VM allocation as much as possible across the infrastructure. Two VMs however can still end up being collocated because either the allocation policy may in some cases favor this (e.g., when using large core-count systems such as Intel's Single-Chip Cloud (SCC) [9]) or because of limited choices available to the allocation algorithm at different times. While collocation may be desirable for the high data throughput available between the VMs, it is in general undesirable when an application wants to decouple VMs with regards to physical machine or network failure for the purpose of achieving high availability.

One option to provide applications with awareness of Cloud resource-allocation policies is to place them inside the Cloud and offer them as (what is popularly called) a platform-as-a-service (or PaaS). This solution requires a close relationship with the Cloud infrastructure provider and is thus not an option for the average application developer. Another option is to extend Cloud resource-allocation policies with application awareness [12]. This solution ensures that application requirements will be taken into account to some extent when allocating resources. However deployment of the solution requires significant changes to current Cloud management systems and is thus hard to deploy. Finally, another approach proposed in this paper is to infer or be explicitly provided with a limited amount of information from the underlying Cloud. Adaptation mechanisms often built into distributed data-intensive applications (such as data migration) can leverage this information to adapt to a Cloud's generic allocation policies. We believe that this approach is simpler and thus easier to deploy compared to the other alternatives.

The two approaches we explore in this paper are: (a) the *black-box* approach, namely to discover VM collocation via the

* The authors are listed in alphabetical order.

978-1-4673-0269-2/12/$31.00 ©2012 IEEE

use of measurements between VMs [19] without help from the underlying Cloud management system; (b) the *gray-box* approach, namely to discover VM collocation via the use of a novel Cloud management API. We call this the gray-box approach because the Cloud exposes only limited information about resources associated with a given user. A drawback of the first approach is the sensitivity to the existence of other workloads in the infrastructure. The drawback of the second approach is the need for Cloud providers to support a new API. However, in this paper we demonstrate that such an API is simple to implement within Eucalyptus [17], an open-source variant of the Amazon Elastic Compute Cloud (EC2) [1]. To facilitate rapid adoption of our proposed API we plan to make our API implementation available as a patch to the Eucalyptus system.

Our contributions in this paper are:

- Two methodologies to perform distributed discovery of VM collocation information in a variety of Cloud environments. Adaptive distributed applications can use this information to achieve their performance and availability objectives.
- A novel, lightweight, privacy-preserving, easy to deploy Cloud management API that results into minimal overall discovery time.
- Evaluation of our methodologies on three commercial Cloud providers (Flexiant, Rackspace, Amazon) providing insight into their VM allocation characteristics, as well as our own experimental Eucalyptus setup.

Our evaluation of the black-box methodology has so far focused on bandwidth measurements. While we have considered alternative methods such as network diagnostics (traceroute) and response time (ping) on Cloud platforms, our experience with traceroute has so far been negative (no information returned) whereas response time results have been more promising. Due to space limitations we omit a full exposition of our results and focus on bandwidth in this paper.

Our results show that VM collocation occurs with non-negligible probability in commercial Cloud environments. We find that our black-box methodology provides accurate results with minimal impact to application performance. Increasing interference with other Cloud activity may affect our measurements, thus having an impact on the quality of our discovered results. As a countermeasure, the use of a novel Cloud API that provides same-user VM collocation information can improve results on all fronts. We demonstrate our work in the context of the Hadoop file system. Our methodologies however are more general and can apply to several other distributed data-intensive applications [4][7][11].

II. Related Work

The problem of mapping application requirements to Cloud resources has received significant attention in the past. One recently proposed approach is the topology-aware Cloud resource allocator (TARA) [12], a system that adopts a "what if" methodology to guide Cloud allocation decisions towards optimizing system or application-specific objective functions. TARA uses a prediction engine with a simulator to estimate the performance of a given resource allocation and a genetic algorithm to efficiently search the solution space. TARA differs from our work in that it is a solution targeted at the Cloud management system rather than at the application middleware.

Another approach aiming for improved performance in data-intensive applications such as Map-Reduce [6] in the face of heterogeneity of Cloud resources is the work of Zaharia et al. [24]. This work proposes a new scheduling algorithm (LATE) to speculatively re-execute tasks that appear to be running on slow Cloud servers. Our work differs from Zaharia et al. in that we propose a general solution that is not specific to Map-Reduce and we take into account data availability in addition to performance. Other approaches to adapting data-intensive application middleware to data center topology include Hadoop's rack-aware placement [20][21]. This approach assumes that network addressing reflects the structure of the underlying physical infrastructure, a condition that does not hold in Cloud infrastructures. The probability of machine failures in large-scale infrastructures and thus the need for availability mechanisms built into scalable middleware and applications has been documented in several field reports [5][22].

Measurement-based inference of Cloud properties has been applied in the past for reasoning about the possibility of privacy intrusion when an attacker manages to collocate a VM in the same physical machine as a victim VM [19]. This work used bandwidth, latency, and routing-based measurements to reason about VM collocation in the Amazon EC2 Cloud. Our work was influenced by their measurement methodology but differs in that we concentrate on collocation between VMs belonging to the same Cloud user. Wang and Ng [23] provided a thorough analysis of VM network performance at the Amazon EC2 Cloud, giving us important insight for interpreting our network bandwidth measurements across commercial Cloud providers. Apte et al. [2] determine dependencies between VMs, such as when a VM uses a service offered by another, with no knowledge of or modifications to applications. Their approach estimates auto-regressive models for the CPU usage of individual VMs and clusters them based on which models are similar. Their work is related to ours in its use of measurements to determine relationships between VMs but is not immediately applicable to the problem of discovering VM collocation.

In this paper we assume that data storage takes place at the local storage devices of participating VMs (an approach termed *instance-store* by Amazon EC2 [1]), a distributed storage model that is becoming increasingly popular [4][7][11][21] over the alternative of remotely-accessible storage, which is typically implemented over a storage-area network. The key benefits of instance-store are: (a) scalability in both capacity and bandwidth with the number of application nodes; (b) predictable performance; and (c) low cost (typically included in the cost of the VM); its main disadvantage is that a physical or virtual machine crash could take away the instance-store— it is therefore essential to replicate across system nodes. Note that VM shutdowns are preserving instance-store data. The key benefits of the remote-storage approach are storage consolidation and the ability to share storage across VMs. The

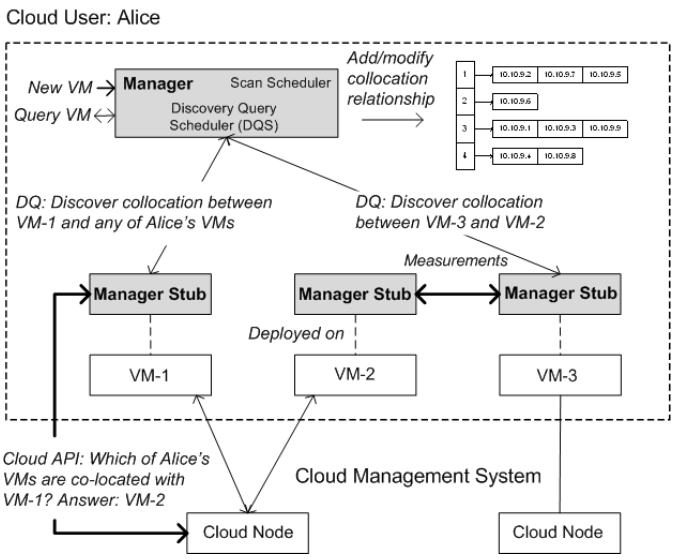

Figure 1: Management system design

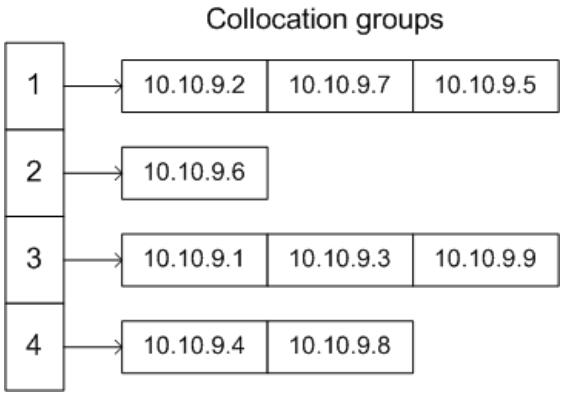

Figure 2: Representation of VM collocations of a Cloud user: 10.10.9.* are private IP addresses of VMs; 1-4 are identifiers of Cloud nodes hosting those VMs

drawbacks of the remote-storage approach are: (a) network bottleneck between the separate compute and storage clusters and (b) need (and significant challenge) to apply rate-control mechanisms [13][14] on the storage path by Cloud providers.

III. DESIGN

Our management system design is depicted in Figure 1. We assume that the system executes in the context of a specific non-privileged Cloud user (in this example, Alice) and consists of the following component types: The *manager*, a centralized component that orchestrates the operation of the system, and a set of distributed *manager stubs*. We chose the term stub rather than agent or sensor (which would better fit their functionality) because they are embeddable into distributed components belonging to another middleware system. We assume that each component (manager or stub) is deployed on a separate VM hosted by a specific physical machine (node) within the Cloud and that all VMs belonging to a Cloud user are hosting stubs.

The aim of the manager is to have up-to-date collocation information for all of Alice's VMs. It learns about the existence of application VMs either through an explicit call ("*New VM"* in Figure 1) or via starting a stub at that VM, which then registers with the manager. To maintain up-to-date collocation information the manager periodically instructs stubs to determine collocation relationships between themselves and other stubs according to a specific schedule. Each interaction between the manager and a stub for this purpose is called a *discovery query (DQ)*. The manager subsystem responsible for scheduling DQs is the *discovery query scheduler* (DQS). A sequence of DQs that refreshes knowledge about all stubs in the system is called a *scan* and results in a representation of collocation relationships depicted in Figure 2. Boxes numbered 1-4 represent anonymous Cloud (physical) nodes. Sequences of 10.10.9.* boxes in each row represent VMs that are believed to be collocated within that Cloud node (we refer to such VMs as a *collocation group*). The typical usage of this management system is to help adaptive data-intensive applications drive policies such as task/data assignment and migration.

An important parameter in our system is how often to repeat a scan, which depends on the approach chosen. We distinguish between the *black-box* and *gray-box* approaches:

A. Black-box approach

In the black-box approach the manager uses a simple round-robin policy to assign DQs to stubs. Upon receiving a DQ a stub performs pair-wise measurements (in sequence) between itself and the set of stubs specified by the manager. When the measurements have been completed, the stub responds to the manager with the VM collocation relationships found. The manager (DQS) ensures that only a single discovery query is executing at a time to avoid interference with the discovery process and to reduce the impact on the executing application. However, external activity originating from other Cloud users is beyond the system's control and may impact measurements[1]. Because of such interference the manager may discover that the information received from a DQ differs from the information it had previously discovered.

In performing a scan, the DQS chooses between two policies: I) trust collocation information found in a previous scan; or II) re-consider all collocation information. Assume that at the beginning of a scan there are N stubs in C collocation groups. In policy (I) the manager hands DQs to a representative stub from each collocation group and asks it to run measurements with representatives from each of the other collocation groups. The time to complete a scan is thus proportional to C^2. If we find a new collocation relationship between stubs x and y we merge their respective collocation groups. In policy (II) we consider any collocation information as potentially stale and thus schedule DQs with all stubs. For

[1] Note that this is not an issue in the gray-box approach.

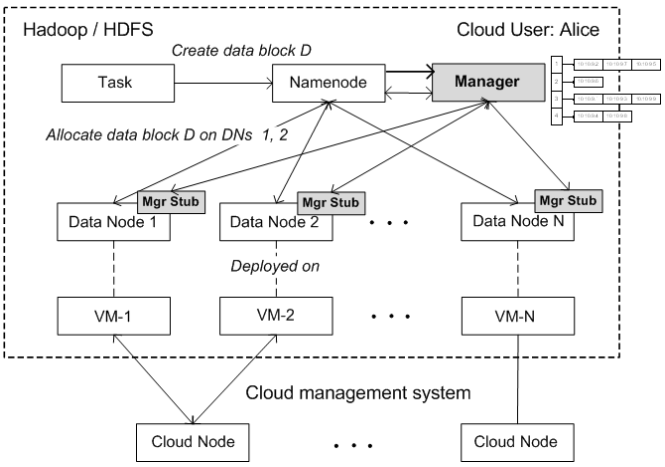

Figure 3: Integration of management system with HDFS

each stub we run measurements with representatives from each collocation group, including its own. The time to complete a scan in this case is thus proportional to *N***C*. We call the former an *incremental scan* and the latter a *full scan*. A special case of the full scan is the *initial scan*, where stubs are not known in advance but are registering with the manager as the scan progresses. In this case, DQS gives priority in scheduling DQs to stubs that have never previously been involved in a DQ.

Interference with other Cloud activity may cause the black-box approach to miss collocation relationships because of bandwidth or CPU sharing between VMs. The opposite however is not possible: A collocation relationship cannot be accidentally detected because of interference with other user workloads. Thus we believe that collocation relationships detected are generally accurate. Scans during periods of low Cloud workload will provide better quality results compared to scans over busy periods. Unfortunately prolonged periods of low load are rare in Cloud environments serving Internet-scale workloads around the clock. However brief periods of lower utilization are more common and as incremental scans coincide with them, we expect that the DQS will be correcting previous noise-induced errors over time (i.e., discover collocated VMs that were earlier determined to be non-collocated).

Incremental scans can achieve shorter DQ times by not re-considering past VM collocation information— this however may lead them to miss collocation changes that could have taken place via internal Cloud re-allocations such as VM migrations. Policy (II) can detect such changes but results into longer scan times. As Cloud re-allocations are rare and have typically long turnaround times we believe that the best approach is to use policy (I) in shorter time scales (e.g., hourly or daily) and policy (II) over longer time scales (e.g., weekly).

B. Gray-box approach

In the gray-box approach we use a specifically designed Cloud management API to provide answers to the query: "*Which of Alice's VMs are co-located with VM_x in the node that hosts VM_x*" (an example appears in Figure 1-left, where the manager poses this query to the Cloud via VM-1). This query does not expose to the requestor information about VMs belonging to other users; thus it cannot be used for launching information-leakage attacks such as described by Ristenpart et al. [19]. It also does not expose any other details about the underlying infrastructure. A possible abuse is a user repeatedly allocating and de-allocating VMs until achieving a desired degree of collocation to ensure high bandwidth across VMs. A Cloud provider can defend against such use by limiting the rate at which it provides answers to the query to say once per hour. Such limitations would not reduce the value of the query for our management system. The information returned by the Cloud is authoritative and thus repeated invocations will likely return the same results unless the Cloud management system modifies its internal allocations (an infrequent event).

In the gray-box approach, the manager can simultaneously issue DQs to all stubs since DQs do not interfere between them or with external Cloud activity. An alternative would be having the manager exercise the Cloud management API. While this is a feasible approach, we expect that the distributed version will result in better overlap of the queries and thus reduced response time. As the impact of gray-box scans to executing applications is expected to be minimal, such scans need not be distinguished into incremental or full for reasons of performance. The manager has thus significant more flexibility compared to black-box scans as to when to execute them, taking into account possible restrictions to frequency of use imposed by the Cloud provider.

IV. IMPLEMENTATION

Our implementation uses the Hadoop file system (HDFS) as a case study, a core component of the Hadoop Map-Reduce [6] framework. As shown in Figure 3, HDFS consists of a *name node* (NN) and a number of *data nodes* (DNs). The NN manages the metadata (file and directory namespace and mapping of file blocks to DNs) and is responsible for the block placement and replication policy. The DNs are the nodes where file blocks replicas are stored. The NN and each of the DNs are hosted by a separate VM on a Cloud setup. We assume that DNs use the local storage resources of their underlying VMs—specifically locally-attached disks. Hadoop *tasks* perform some computation on blocks of data files. The overall principles in the design of Hadoop/HDFS are representative of a number of scalable data-intensive applications using a distributed replicated storage backend [4][7][11].

A. Black-box approach

To implement our black-box approach we extended the HDFS NN and DN source code to embed our manager and stub functionality. Our extensions were focused on the HDFS data node protocol through which DNs communicate with the NN. Periodically (every three seconds) each DN reports to the NN its condition via a heartbeat message wrapped in a remote procedure call (RPC). Our first extension to the HDFS data node protocol was to piggyback discovery queries (DQs) to DNs along with its standard HDFS commands [21]. Each DQ carries a list of IP addresses of DNs with which the target DN stub must perform (sequentially) pair-wise bandwidth

measurements. In our implementation, each DN stub runs continuously a *netperf* [16] server to always be available to respond to a benchmark request by another DN stub. When the target DN stub completes its measurements it responds to the NN with the VM collocation relationships found (this is our second extension to the HDFS data node protocol).

The manager organizes stubs in collocation groups (Figure 2) using an array of hash tables grouping user-visible IP addresses of VMs in the form of strings. Each hash table corresponds to a distinct physical machine. This representation does not reveal any characteristics of the physical machine (which are in any case opaque to the Cloud user). The hash table representation allows rapid lookup of a VM into a collocation group. When the NN DQS communicates a list of IPs to a target DN stub, the size of that list is always proportional to the number of collocation groups since both black-box policies (I, II) choose representative VMs from collocation groups.

As stubs register with the manager when they are started or restarted, the manager will assign them a "pending" status as it schedules discovery tasks for them. The NN may or may not use the corresponding DNs for allocation. There are two possible policies for the NN here: Do not use a DN until its stub leaves "pending" state or use it while in "pending"; the former policy results in an unbalanced system (more replicas on nodes that register first), whereas the latter may result in replicas allocated on collocated DNs (another form of imbalance). These policies are relevant mostly when there is a continuous stream of allocate requests to the NN while the manager is processing stub registrations—namely, at boot time or right after crash recovery. In common-path operation we expect that there will be very few nodes in "pending" state. Even when imbalances occur, however a periodically-invoked re-balancer [21] can correct them by applying data-migration actions over time.

The DQS ensures that only a single DQ is scheduled across all DNs to avoid interference. To achieve this, it maintains a queue of stubs picking its next target to start a discovery query with from the tail of the queue. After receiving a response from it, the stub is moved to the head of the queue. In incremental scans, all stubs that are reported to be collocated with it are also placed at the head of the queue. A newly registered stub will be inserted at the tail of the queue. Consequently it is that stub that will be next chosen to perform a DQ when the currently running DQ completes. Other important parameters are how often to schedule incremental or full scans. We currently run an incremental scan every two hours and a nightly full scan.

Our final extension to HDFS was to alter its replica allocation algorithm to take into account DN collocation information. The standard HDFS replica allocation algorithm uses a rack-aware algorithm [20][21] for high availability. Our extension to it avoids picking collocated DNs for storing replicas of the same data block. If the number of physical machines available is smaller than the number of replicas per block (unusual in real-world HDFS setups), the NN places as many replicas as it can in different physical machines and then randomly places the remaining replicas across the remaining available DNs. When collocation is desirable (such as when a MapReduce task wants to have rapid read-access to a replica of a data block) our system can be used to provide that information to the task allocation algorithm.

B. Gray-box approach

We have implemented the Cloud management API described in Section III.B in the context of the Eucalyptus open-source Cloud management system [17]. Our implementation is currently a web service operating at the Eucalyptus *cloud controller* (CLC), a centralized component responsible for maintaining system and user metadata, managing and monitoring VM instances across *node controllers* (NC). NCs control VM activities, including the execution, inspection, and termination of VM instances through the use of a hypervisor.

Our web service exposes a SOAP API by which any DN/stub can find out other VMs belonging to the same user that are collocated on the same physical host. The web service is a privileged component of Eucalyptus. It discovers this information by querying internal CLC data structures or by parsing its operation logs. When a DN/stub contacts the web service, it sends only its own IP address. The web service is able to map the IP to the Cloud user or security group and thus only exposes information pertaining to the asking user. The service's response contains a list of IP addresses corresponding to VMs that are hosted by the same physical machine and are owned by the same user (security group) as the caller. The DN/stub then reports this IP list to the NN, which validates those IP addresses that are indeed used by the HDFS as DNs. Finally, the collocation-group information is updated.

A drawback of the existing web service is the fact that it is centralized and thus may lead to a load increase at the CLC when a large number of stubs are invoking it simultaneously. This situation could arise when the manager schedules a discovery query simultaneously across hundreds or thousands of stubs. Note that this not a unique issue with our management API but also applies to the Eucalyptus metadata query API accessible via http://169.254.169.254/latest/meta-data, which is also serviced at the CLC. One solution to this problem is to phase API invocations over time (possibly via random delays). Another solution would be to explore an alternative implementation of our web service that operates within each NC (since the information requested naturally exists there). The main technical challenge with this approach is the creation of a network path –opaque to the VM— between the VM and the host machine. We are currently exploring this implementation as part of ongoing research work while we believe that our current implementation is a robust and efficient solution for all practical purposes.

V. EVALUATION

We report results from three commercial public Cloud providers: Flexiant's Flexiscale [8], Rackspace [18], and Amazon EC2 [1]. We also report results from our own in-house Eucalyptus [17] experimental private Cloud. All VM instances allocated had similar specifications: single-core, 1GB RAM, and 20GB of disk space (except for Eucalyptus instances that had 512MB of RAM). The VM disk was an instance-store

Platform	Non-collocated (Mbps)	Collocated (Gbps)
Flexiscale	750	4.0
Rackspace	29.8	5.2
Amazon	550	-
Eucalyptus	850	1.5

Table 1: Inter-VM bandwidth measurements

Number of data nodes (VMs)	VMs/phys. node Avg.	Max	Number of physical nodes
5	1.9	2.0	2.7
10	1.7	3.0	6.0
15	2.2	3.7	7.0
30	2.8	9.0	10.5

Table 2: Flexiscale collocation results

(Section II) in all Cloud setups except Flexiscale, where the disk can only be remotely mounted [15]. While Flexiscale diverges from our assumption of using locally-attached disks, our results are valid and apply to the hypothetical case of using instance-stores on the same platform. The Eucalyptus Cloud is hosted on a 10-node cluster of dual-CPU AMD 244 Opteron servers with 2GB DRAM running Linux 2.6.18-238el.el5xen and interconnected via a 1Gbps Ethernet switch. Each node was provisioned with a dedicated logical volume comprising four 80GB SATA disks in a RAID-0 configuration. We used the xfs filesystem on this volume on all nodes. The nodes were under the control of the Eucalyptus (version 2.0.3) Cloud management system using the Xen hypervisor. We used Hadoop file system (HDFS) version 0.20.2. The measurement tool used in our black-box methodology in all experiments was the netperf [16] benchmark. We empirically determined that a 3-second run of netperf gives us a credible bandwidth estimate between any pair of nodes. We execute three such netperf runs and report the average bandwidth.

In our experiments, VMs are reported as collocated if the average bandwidth between them exceeds 1Gbps. This is a simple and accurate test based on the overwhelming dominance of 1Gbps Ethernet technology in the public Clouds we considered. However the more general assumption our system makes is not tied to a specific technology and is based on the observation that the memory bandwidth between collocated VMs is always greater than network bandwidth between non-collocated VMs. We believe this assumption holds true for the majority of commercial public or private Clouds. Even when considering Clouds using 10Gbit/s links at the node level, a VM will be capable of seeing only a fraction of that peak bandwidth, proportional to the share of system cores allocated to the VM. This is true even for 1Gbps links today, as previous studies [23] indicate and our results confirm. Additionally, several Cloud providers limit –through explicit control at the network level or at the hypervisor— the VM network bandwidth to a fraction of the peak bandwidth supported by the network technology. To the best of our knowledge, there are no such limitations for the memory bandwidth between collocated VMs. Finally, it is highly unlikely that commercial Cloud providers would choose to maintain compute nodes whose memory system would be so dated as to violate the above assumption. In the rare case that the assumption is violated, use of alternative metrics (such as response time) can help in improving the accuracy of our results. It is important to note that even when our system is incorrect (labeling two VMs as collocated when in fact they are not), this does not hurt applications: Our HDFS prototype will allocate replicas of a block to VMs that are reported collocated if it has no other choice.

Our bandwidth measurements across the Cloud platforms are summarized in Table 1. We observe that inter-VM bandwidth in public Clouds range from very low (~30Mbps at Rackspace, probably rate-controlled) to about 75% of hardware link speed at Flexiscale. Our Eucalyptus private Cloud setup provides better inter-VM bandwidth due to less interference across workloads. Collocated VMs can communicate at speeds far higher than the 1Gbps link speed (up to 5Gbps in Rackspace) in commercial Cloud platforms. Our lower results (1.5Gbps) in our Eucalyptus setup are due to previous-generation server hardware, unlikely to be used in modern Cloud setups. We were not able to measure collocated VM bandwidth in Amazon EC2 due to the relatively small size of our largest allocation (40 VMs) [23][24].

In what follows, we first show that VM collocation occurs with non-negligible probability in commercial Cloud setups and that if not taken into account it can cover up dependability issues when deploying data-intensive applications in the Cloud. We also show that our black-box approach can detect collocation in time proportional to system size and inversely proportional to the degree of collocation. Noise from other user activity however, can impact the quality of scan results. Repeated scans can take advantage of occasionally lower levels of noise improving the quality of results. We then show that our gray-box approach is a solution that is immune to noise and whose scan time is practically independent of system size. Finally, we show that black-box scans, while the more expensive of our two approaches, have no measurable impact on application performance, pointing to the non-intrusiveness of our system.

A. VM collocation in commercial public Clouds

To determine the degree of VM collocation on the Flexiant Cloud we perform successive allocations of 5, 10, 15, and 30 VMs and measure the average and maximum number of VMs per physical node, as well as the average number of physical nodes involved in an allocation. We perform three allocations for each size. Our results are summarized in Table 2. Our results show that there is a significant degree of VM collocation in this platform (up to 9 VMs per node in the 30 VM allocations).

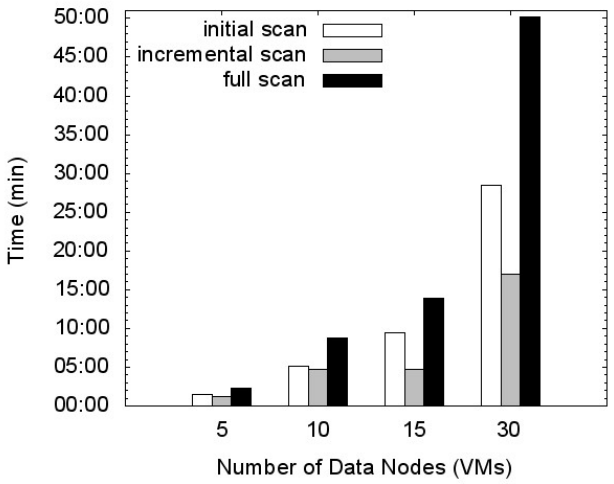

Figure 4: Duration of Flexiscale scans without noise

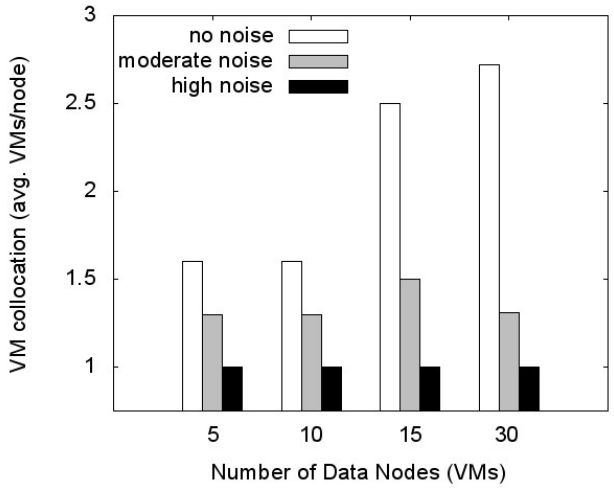

Figure 5: Quality of results in Flexiscale initial scan with varying noise level

To underline the problem created when an application is not aware of such collocation, we use the Hadoop dfsIO benchmark [21] to create 100 files of 200MB each (a total of 20GB) over HDFS deployed on two of the 10 VM allocations of Table 2. Each 64MB HDFS block is triply replicated on the 10VMs. After completion of the benchmark we examine the 500 blocks created on the VMs and find that on average 39% of the blocks have 2 out of 3 replicas stored on the same physical machine. Additionally, on average 2.6% of the blocks have all their replicas stored on the same physical machine, compromising data reliability if that physical machine experienced a failure. We repeat the dfsIO experiment on the larger Flexiscale allocation of 30 VMs and find that 33% of the blocks end up with 2 out of 3 replicas on the same physical machine. Furthermore 2% of them end up with all replicas on the same physical machine, exposing again the mismatch between the availability goals of the application and the quality of resources allocated within the Cloud.

Measurements on the Rackspace public Cloud show a variable degree of collocation: Specifically, one 40 VM allocation yielded two pairs of VMs collocated whereas all other VMs (36) were not. A second 40 VM allocation yielded VM instances broken down into 28 physical machines with an average of 1.3 and a maximum of 4 VMs allocated per physical node. We conclude that Rackspace is another Cloud provider with which a significant degree of VM collocation is possible at moderate allocation sizes. Measurements on the Amazon public Cloud on the other hand, showed no collocation for allocations of up to 40 VMs. This result is consistent with previous observations [24] where allocations of the order of 800 VMs had to be attempted before determining collocation (of up to 7 VMs per physical node). Our bandwidth measurements would have been sufficient to detect such collocations at the scale where they are likely to occur.

B. Scan duration and quality of results

In this section we show that the time to perform scans in the black-box approach is proportional to the overall system size and inversely proportional to the degree of VM collocation. We focus on the Flexiant platform as representative of a Cloud exhibiting a high degree of collocation. Figure 4 depicts the time to complete different scan types (initial, incremental, full) on allocations of 5, 10, 15, and 30 Flexiscale VMs. We observe that the time increases across all scan types for increasing system size. A full scan takes longer than an incremental or an initial scan across all system sizes. This is because a full scan (unlike an initial scan) knows all registered stubs from the beginning. Also, a full scan (unlike an incremental scan) cannot benefit from previously discovered collocation relationships. Note that when scaling the system from 10 to 15 VMs, incremental scan time is unaffected because the degree of collocation in these two cases happens to be the same. To better highlight the dependence of scan time with the degree of collocation, in Table 3 we depict the time to complete different scan types on allocations of 10 VMs with a degree of collocation (average number of VMs per physical node) ranging from 1.43 to 2. We observe that as the degree of VM collocation grows scan time decreases across all scan types.

Next we show that the quality of results (reported degree of VM collocation) varies with the level of noise that interferes with discovery queries. We emulate the impact of noise by injecting configurable levels of network traffic during an initial scan. To emulate different noise levels (moderate to high) we execute netperf tasks (unrelated to discovery) on stub

VMs/phys node (average)	Scan time (seconds)		
	Initial	Incremental	Full
1.43	361	394	615
1.66	304	285	529
2.00	292	268	461

Table 3: Scan time vs. degree of collocation with 10 VMs

nodes according to a certain schedule: each node talks in parallel with all other nodes (high level) or half the nodes talk among them in pairs (moderate level). Using the same 5, 10, 15, and 30 VM Flexiscale allocations as in the previous experiment we show (Figure 5) that noise reduces the number of reported collocation relationships under the black-box approach. However after removing the high level of noise, a full or incremental scan can discover all previously-missed VM collocation relationships. These scans take longer compared to the scans reported in Figure 4 (33% and 74% longer for full and incremental scans on 15 VMs and 52% and 48% for full and incremental scans on 30 VMs) due to the absence of collocation groups at the beginning of the scan that could be responsible for reducing scan time.

Finally, we show that the time to perform a scan is nearly constant in the gray-box approach compared to the black-box approach. In Figure 6 we report scan times (initial scan in the case of the black-box approach) from our Eucalyptus Cloud setup with 5, 10, and 15 VMs without noise. We observe that gray-box scans take between 4 and 12 seconds, whereas black-box scans range between 90 and 780 seconds. The difference is due to the fact that gray-box scans execute the Eucalyptus Cloud management API (practically a constant-time operation in this setup) to determine collocation relationships whereas black-box scans perform the full set of bandwidth measurements.

C. *Impact from/to application activity*

To determine the impact of our measurement-based approach to application activity we ran Hadoop dfsIO (configured to create 50 files of 300MB each) on 40VMs allocated on Rackspace, while the management system was simultaneously performing incremental black-box scans. Our results (average over three runs) show that there is no measurable impact on dfsIO performance, as expected due to the low-impact scheduling of discovery queries. At the same time we did not observe any impact on the quality of collocation results produced, evidence to the fact that our system is robust under low levels of noise.

VI. Discussion and Future work

Our initial attempts with measurement technologies beyond bandwidth, such with as response-time (ping), were promising and therefore we believe that there is room for eventually using them for strengthening VM collocation criteria or mining Cloud topology information. Our framework is certainly general enough to support other measurement methodologies that may fit and complement our approach. We further plan to extend our framework to cover other types of resource collocation in Cloud infrastructures, such as that of storage volumes within a shared network-accessible RAID array.

VII. Conclusions

In this paper we show that VM collocation takes place with non-negligible probability in commercial Clouds due to generic allocation policies that do not take application requirements into account. Especially in the case of data-intensive applications, VM collocation can create dependability issues exposing them to underlying failures. We also show that our black-box approach can detect collocation in time that is proportional to system size and inversely proportional to the degree of collocation. Noise from other user activity however, can impact the quality of scan results. Repeated scans can take advantage of occasionally lower levels of noise improving the quality of results. Our gray-box approach is a solution that is immune to noise and whose scan time is practically independent of system size. Finally, black-box scans, while the more expensive of our two approaches, have no measurable impact on application performance, pointing to the non-intrusiveness of our system.

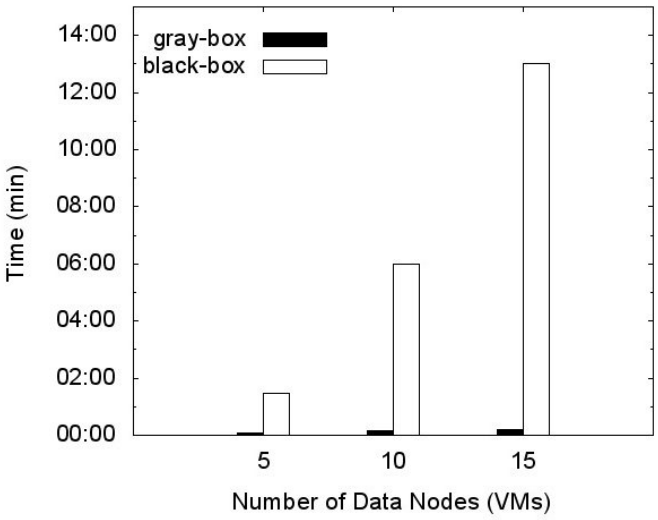

Figure 6: Comparison of gray- vs. black-box scan times

VIII. Acknowledgments

We thankfully acknowledge the support of the European ICT-FP7 program through the SCALEWORKS (MC IEF 237677) and CUMULONIMBO (STREP 257993) projects. We also thank the anonymous reviewers for their valuable comments and suggestions.

References

[1] Amazon Web Services EC2, http://aws.amazon.com/ec2/

[2] R. Apte, L. Hu, K. Schwan, A. Ghosh, "Look Who's Talking: Discovering Dependencies between Virtual Machines Using CPU Utilization", in Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing, Boston, MA, June 2010.

[3] M. Armburst, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, M. Zaharia, "A Berkeley View on Cloud Computing", in UC Berkeley EECS Technical Report EECS-2009-28, February 2009.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data", in Proceedings of Seventh Symposium on Operating System Design and Implementation (OSDI'06), Seattle, WA, November, 2006.

[5] J. Dean, "Design, Lessons, and Advice from Building Large Distributed Systems", Keynote talk at 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, Cornell, NY, 2009.

[6] J. Dean, S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, in Proceedings of Sixth Symposium on Operating System Design and Implementation (OSDI’04), San Francisco, CA, December 2004.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, “Dynamo: Amazon's Highly Available Key-value Store”, in Proceedings of 21st ACM Symposium on Operating Systems Principles, New York, NY, 2007.

[8] Flexiant Flexiscale Public Cloud, http://www.flexiant.com

[9] J. Howard *et al.*, “A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS”, in Proceedings of IEEE International Solid-State Circuits Conference, San Francisco, CA, February 2010.

[10] L. Koromilas, K. Magoutis, “CassMail: A Scalable, Highly-Available, and Rapidly Prototyped Email Service”, in Proceedings of 11th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2011), Reykjavik, Iceland, June 2011.

[11] A. Lakshman, P. Malik, “Cassandra - A Decentralized Structured Storage System”, in Proceedings of 3rd ACM SIGOPS Workshop on Large Scale Distributed Systems and Middleware, Cornell, NY, 2009.

[12] G. Lee, N. Tolia, P. Ranganathan, R. H. Katz, “Topology-aware resource allocation for data-intensive workloads”, in ACM SIGCOMM Computer Communication Review, Vol. 41, No. 1, January 2011.

[13] C. Lumb, A. Merchant, G. Alvarez, “Facade: Virtual Storage Devices with Performance Guarantees”, In Proceedings of USENIX Conference on File and Storage Technologies (FAST), March 2003.

[14] K. Magoutis, P. Sarkar, G. Shah, “OASIS: Self-tuning Storage for Applications”, in Proceedings of 23rd IEEE Conference on Mass Storage Systems and Technologies, College Park, MD, May 2006.

[15] G. Munasinghe, P. Anderson, “Flexiscale: Next Generation Data Centre Management”, in Proceedings of UK Unix-Users Group (UKUUG) Spring Conference, Birmingham Conservatoire, UK, 2008.

[16] Netperf, http://www.netperf.org/netperf/

[17] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, “The Eucalyptus open-source Cloud computing system”, in Proceedings of 9th IEEE CCGrid’09, Washington, DC, 2009.

[18] Rackspace, http://www.rackspace.com/cloud/

[19] T. Ristenpart, E. Tromer, H. Shacham, S. Savage, “Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds”, in Proceedings of CCS’09, Chicago, IL, November 2009.

[20] J. Shaffer, “A Storage Architecture for Data-Intensive Computing”, PhD Thesis, Rice University, Houston, TX, May 2010.

[21] K. Shvachko, H. Kuang, S. Radia, R. Chansler, “The Hadoop Distributed File System”, in Proceedings of IEEE Conference on Mass Storage Systems and Technologies (MSST), 2010.

[22] M. Theimer, "Some Lessons Learned from Running Amazon Web Services", Keynote talk at 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, Cornell, NY, 2009.

[23] G. Wang and T. S. E. Ng, “The Impact of Virtualization on Network Performance of Amazon EC2 Data Center”, in Proceedings of IEEE INFOCOM 2010, San Diego, CA, March 2010.

[24] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, I. Stoica, “Improving MapReduce Performance in Heterogeneous Environments”, in Proceedings of 8th USENIX Symposium on Operating Systems Design and Implementation, San Diego, CA, Dec. 2008.