EURECOM
Department of Networks and Security
Campus SohiaTech
CS 50193
06904 Sophia-Antipolis cedex
FRANCE

Research Report RR-13-287

A Measurement Study of Data-intensive Network Traffic Patterns in a Private Cloud

November 28th, 2013

Daniele Venzano and Pietro Michiardi

Tel : (+33) 4 93 00 81 00
Fax : (+33) 4 93 00 82 00
Email : {daniele.venzano,pietro.michiardi}@eurecom.fr

[1]EURECOM's research is partially supported by its industrial members: BMW Group Research & Technology, IABG, Monaco Telecom, Orange, SAP, SFR, ST Microelectronics, Swisscom, Symantec.

A Measurement Study of Data-intensive Network Traffic Patterns in a Private Cloud

Daniele Venzano and Pietro Michiardi

Abstract

In this work we investigate the impact of virtualization on the raw network performance attainable by "data-intensive" applications deployed in a private cloud. To this end we developed a new software tool, called OSMeF, to take repeatable measurements on our OpenStack-based platform. We also discuss the implications of our measurement results toward informed deployments of distributed applications such as Hadoop.

Index Terms

Big data, OpenStack, measurement, network, virtualization, cloud computing

Contents

List of Figures

1 Introduction

Nowadays, cloud computing represents a key enabler for the rapid provisioning of resources from public or private data-centers to deploy a wide range of applications. To provide flexible and cost-effective resource sharing among users, most cloud service providers rely on machine virtualization, that support multiple instances of virtual machines (VMs) on the same physical server. VM instances share physical processors and I/O interfaces with other instances: as such, it is important to understand the impact of virtualization on computation and communication performance of cloud services.

In this work, we study the performance – from the networking perspective – that private cloud deployments expose to a specific class of applications, namely data-intensive scalable computing frameworks such as Hadoop [1]. In this context, virtualization brings many benefits, including ease of operation, high availability, elasticity and multi-tenancy [2]. However, only few studies attempted to measure rigorously the raw performance achievable by I/O intensive applications running on virtual compute clusters (section 2). Thus, our goal is to cast light on the *bulk transfer capacity* attainable by a range of traffic patterns derived from a typical Hadoop operation.

By using our OpenStack-based [3] private cloud deployment, detailed in section 3, and by leveraging a software tool we built (the OpenStack Measurement Framework, or OSMeF), described in section 4, we studied the behaviour of virtual and physical networks, along with the loopback mechanism that is heavily used by distributed applications like Hadoop. Our measurement results can be used to inform the proper configuration and tuning of OpenStack and Hadoop deployments, including: *i)* the selection of appropriate VM "flavors" to run Hadoop components, *ii)* the importance of VM placement across available physical servers, which largely determines the attainable network throughput of data transfers and *iii)* the need for novel mechanisms to support fast cross-tenant communications.

2 Related work

Understanding the impact of virtualization on network performance has attracted several studies [4, 5] in the past, albeit oriented toward public cloud providers. These works use a black-box approach in their study, as it is difficult (if not impossible in some cases) to determine the measurement conditions (e.g. interference from other users) in their experiments, not to mention knowledge of the physical servers and networks the cloud operator uses for the service. Instead, in our approach, the platform used for the experiments is completely under our control and hardware and software configurations are known and measurable.

Other works focus especially on studying network performance from the hypervisor point of view: for example, studies [6] and [7] examine TCP and HTTP throughput, respectively, in Xen virtual machines. In our work, instead, we use KVM, the most commonly used hypervisor in conjunction with OpenStack. Moreover we also study

end-to-end throughput across the whole virtual network setup by OpenStack. A network performance measurement campaign is made in [7], with HTTP servers running in Xen VMs on a single host and physical clients generating requests.

From the application perspective, some recent works analyze the behavior of Hadoop when deployed in virtual machines. In [8], Han et al. present a model of Hadoop to examine how performance could be affected by VM characteristics and placement in a cluster. However, that work focuses on application-level performance metrics only, neglecting network-level ones. Similarly, the work in [9] studies the performance of Hadoop deployed in an Amazon EC2 cluster, albeit the focus is on the design of a more efficient scheduling mechanism to optimize application-level metrics.

Finally, recent works [10, 11] recognize the challenge of designing efficient virtual networks, and propose software approaches to increase virtual network performance.

3 Experimental set-up

We conduct our measurement study on a cluster of dedicated machines, configured as a *private cloud*, using the OpenStack cloud operating system [3].

In this section we briefly outline the design of OpenStack, concentrating on its networking component, `quantum`,[1] and provide details on our platform, both from the hardware and software points of view.

3.1 OpenStack

OpenStack provides a range of management functionalities needed by a cloud provider, from user and tenant management (`keystone`) to interfacing with hypervisors (`nova-compute`), virtual networking (`quantum`), block (`cinder`) and object storage (`swift`). All these software components are implemented in a distributed and fault-tolerant way, with each service storing a minimal amount of state in a series of off-the-shelf database systems and communicating with each other through a queueing system based on the AMQP (Advanced Message Queuing Protocol).

As an illustrative example, we outline the procedure to create and instantiate a virtual machine on one node of a cluster. Users issue requests for new VMs using either a Web-based console (similar in nature to that available for Amazon Web Services [12]) or one of the available APIs. An OpenStack *master node*, receives such requests through `nova-api`, that passes it on to a scheduling component called `nova-scheduler`. The scheduler applies a set of system-wide policies to find a suitable compute node where the VM can be run on. At this point, the scheduler instructs the `nova-compute` process running on the selected node to negotiate with the hypervisor the instantiation of a new VM. The VM image is provided by a service (namely `glance`) that ships the VM file to the compute node, while all VM network interfaces are configured by the `quantum` component.

[1] This component is going to be renamed to `neutron` starting from the next release of OpenStack.

Configuring and tuning OpenStack is complex: the large number of parameters that govern system behavior, the variety of hypervisor technologies (XEN, KVM or others), the different flavors of storage systems (various filesystem and logical volume management combinations), the number of alternatives to implement network switching (GRE tunnels, dynamic VLANs) all play a crucial role in determining the overall system performance.

Next, we provide details on the configuration of the platform we use for our measurements.

3.2 Our Platform

Our cluster uses a heterogeneous set of physical machines: we have two *master* nodes running on a dual quad-core Xeon L5320 server clocked at 1.86GHz, with 16GB of RAM, two 1TB hardware RAID5 volumes, and two 1Gbps network interfaces; *worker* nodes execute on six dual exa-core Xeon E5-2650L (with hyperthreading enabled) servers clocked at 1.8GHz, with 128GB of RAM, ten 1TB disks (configured as JBOD, Just a Bunch of Disks) and four 1Gb/s network cards. In this work we use a single network interface per host, as splitting traffic or using bonding would have added complexity to the system that, instead, we strive to keep as simple as possible, since in this work we are interested only in baseline performance.

The hardware and network configuration closely resembles the one suggested by commercial private cloud providers, such as Rackspace [13]. In particular storage is distributed on the master and compute hosts and is not concentrated on a separate storage network.

Each machine in the cluster runs the same Linux distribution, a Ubuntu 12.04.2 LTS, updated with the most recent patches. All energy saving settings in the BIOS are disabled, since they cause severe performance penalties. We use the KVM hypervisor, with `virtio` and `vhost_net` acceleration modules enabled. Virtualization support in the CPUs is enabled (VMX) and KVM uses it automatically. The hypervisor is configured by Nova to use LVM for VM storage. VMs use the unmodified Ubuntu 13.10 image from the Ubuntu Cloud archives.

We use the *Grizzly* release of OpenStack, which is installed via the Ubuntu cloud repository. One of the master nodes runs the OpenStack management services: the web-based dashboard console, `cinder`, `glance`, `keystone`, and `quantum` (including the server, layer 3 services, OpenVSwitch and DHCP agents). Worker nodes are configured as compute-only nodes, and they host all the VMs created by our tenants and users. Currently, we configure `quantum` to use GRE tunnels over a physical network that interconnects all nodes of our cluster.

We implemented the most common setup, where `quantum` is configured to use the OpenVSwitch [14] (OVS) plugin to provide connectivity between VMs. OVS is a software switch implementation that materializes as a virtual switch spanning across multiple physical hosts. In our configuration, `quantum` creates a single OVS switch for all VMs, using VLAN tagging to separate traffic from different tenants.

To provide connectivity between tenants and the external network, our virtual network is configured according to the *provider router with private networks* use-case described in the OpenStack documentation [15]. Thus, each tenant has its own IP subnet, and exchange traffic between each other and the Internet using a single virtual router connected to the subnets of each tenant from one side and to the external network on the other side. The `quantum` virtual router is implemented as network namespace on the master node, where a number of NAT and routing rules provide interconnection, external access and floating IPs allocated to the VMs.

These settings are the result of a tedious trial and error process that lasted several months. The OpenStack installation manuals only cover the basics to setup a system that is (mostly) operational, but far from being optimized for performance. Other parameters are buried in bug reports, OpenStack *blueprints* (informal feature proposals to the community) and mailing list archives.

4 Measurement methodology

In this section we describe and motivate the methodology used to perform the measurements presented in section 5. To guarantee reproducible and accurate results we implemented OSMeF, a measurement framework described in section 4.3, that automates deployment and execution of tools required to establish a variety of traffic patterns, and the collection, parsing and representation of measurements output. Finally, in section 4.4 we describe the performance metrics we considered for our results.

4.1 Motivation and background

We consider data-intensive scalable computing applications, and in particular we focus on the Hadoop framework [1]. Hence, we are interested in studying the network performance of traffic patterns that mimic[2] those generated by a typical data analysis task. In addition, we consider several "flavors" of Hadoop deployments, that exploit the characteristics of virtualization in the context of multi-tenant, shared clouds. Next, we briefly describe the basic principles of Hadoop MapReduce.

MapReduce, popularized by Google [16] and by Hadoop [1], is both a programming model and an execution framework. In MapReduce, a data analysis job consists of three phases and accepts as input a dataset, appropriately partitioned and stored in a *distributed file system*. In the first phase, called MAP, a user-defined function is applied in parallel to input partitions to produce intermediate data stored on the local file system of each machine of the cluster; intermediate data is sorted and partitioned when written to disk. Then, a REDUCE phase begins. It comprises a SHUFFLE phase, where intermediate data is pulled by the *reducers*: data from multiple mappers is sorted and aggregated to produce output data.

[2]Note that, in this work, we do not execute Hadoop in our cluster, hence we do not measure communication performance at the application level. Rather, we generate traffic with system-level tools, according to the communication patterns of Hadoop.

When a *single* job is submitted to an Hadoop cluster, the Hadoop scheduler assigns a number of MAP tasks equal to the number of partitions of the input data. The scheduler tries to assign MAP tasks to "slots" available on machines in which the underlying storage layer holds the input intended to be processed, an important concept called *data locality*. Essentially, when reading input data and writing output data, local communications are to be preferred over remote ones, since the network is generally assumed to be the bottleneck. REDUCE tasks are scheduled once intermediate data, output from mappers, is available. Overall, the performance of an individual job is determined by the slowest task: asymmetries due to an uneven amount of data to process or to network bottlenecks may create "stragglers" that should be carefully handled by the application.

As a concluding remark, it is important to notice that Hadoop is made of a number of components that run as daemons: *master* nodes execute daemons (e.g. the scheduler) that orchestrate several workers, each in charge of *data* (namely, a DATANODE of the distributed file system) and *compute* (namely, a TASKTRACKER of MapReduce) operations. For the principle of data locality outlined above, data and compute components are collocated on the same cluster machine. All components communicate through a simple REST-ful API: in particular, co-located components exchange messages through the loopback interface, whereas components on different hosts, use network interfaces.

4.2 Traffic patterns

We are now ready to describe the traffic patterns we consider in this work. They stem from two main points we are considering: characterization of the distributed applications we support in our cluster and how they are affected by virtualization.

Application-level characteristics. The nature of the communications between Hadoop components discussed above leads us to focus on the performance achieved by both "regular" network and loopback interfaces, the former being used mainly during the SHUFFLE phase of MapReduce and the latter used mainly for data I/O operations. As such, in section 5 we measure the performance of the *localhost* – both at the hypervisor and VM level – and of the *cluster network* – between hypervisors and VMs.

Recall the data-intensive nature of the applications we consider: therefore, we study the behavior of *long-lived connections* (in all our experiments we consider 6 minutes long data transfers) that are typical in Hadoop. For example, in the SHUFFLE phase, it is common to transfer several GBytes of data among cluster machines. Moreover, from a measurement point of view, long-lived connections have the advantage of reducing the variance [17] caused by load spikes in network and CPU usage in the cluster.

Finally, we consider the effects of a number of *parallel communications* taking place concurrently: this is another typical traffic pattern of Hadoop, whereby each component could establish a large number of connections to exchange data.[3] Note that

[3]For example, during the SHUFFLE phase, a TASKTRACKER serves up to 40 connections, and establishes up to 5 connections to pull intermediate data.

it is also important to study whether bandwidth allocation among competing parallel connections is fair, as uneven performance may contribute to the creation of "stragglers" and impact the overall application-level performance of a data analysis job.

Virtualization effects. Deploying data-intensive applications in a cluster of virtual machines involves a wide range of possible architectural choices. As we alluded above, in a typical "bare-metal" Hadoop architecture, *compute* and *data* components are co-located, in light of the data locality principle. Similarly, with respect to the traffic patterns, we consider a virtual Hadoop cluster in which individual VMs host both components.

In addition, we also study more elaborate setups that stem from recent efforts [2] to define reference Hadoop architectures for multi-tenant, shared clusters of virtualized resources. Essentially, we consider traffic patterns that arise when *compute* and *data* components live in different VMs and possibly in different tenants. In our experiments, we therefore study network performance under a variety of VM-to-VM communication patterns: in doing so, we revisit the notion of data locality and define a new distance metric that accounts for *VM placement* choices. For example, we consider communications to be local even when they involve distinct VMs, as long as they are instantiated on the same physical host.

Finally, we also study the effects on network performance of a variety of VM "flavors": namely, we focus on the number of virtual CPUs available to a VM. Indeed, the CPU plays a crucial role in determining the performance of some software network components we use in our platform (including the loopback interface and the virtual switches). As such, we use VMs with 1 up to 16 virtual CPUs.

Scenarios not covered in this study. There is a number of traffic patterns that we deliberately omit from this work, for the sake of space. Essentially, applications such as Hadoop involve one-to-many, many-to-one and many-to-many communication patterns between VMs. In addition, several configuration parameters of Hadoop dictate how and when this kind of traffic pattern arise: we are currently investigating such scenarios, and we will present our measurement results in an extension of this work.

4.3 OSMeF

The OpenStack Measurement Framework (OSMeF) provides a way to perform a large number of measurements in an automatic and reproducible way. It is implemented in Python and uses a JSON output format.

OSMeF implements the OpenStack and Quantum APIs, which are used to instantiate and delete VMs and virtual interfaces as required by the specific measurement scenario being performed. Essentially, OSMeF reads a measurement configuration file, performs VM placement according to the specification (including a proper selection of VM flavor), instruments all VMs with additional software components required to perform a specific measurement and finally runs the necessary tools, parsing and aggregating their outputs to create a compact JSON summary for each experiment.

In addition, before any experiment is executed, OSMeF gathers a number of statistics from each end point operating system, which are required to verify that mea-

surements are performed under known system conditions. These statistics include a time-stamp, the current load on each machine (both physical and virtual), network utilization, kernel versions and many others. Such experiment "meta-data" is appended to the JSON output of each measurement campaign.

Currently, OSMeF uses *nuttcp* [18] to perform the actual network measurements.[4] *Nuttcp* is a well-known tool (originated from *ttcp* [19]) that produces a number of useful statistics (in addition to raw throughput figures) gathered during a TCP transfer of a specified length. For each experiment, OSMeF configures *nuttcp* according to the measurement configuration file, including the number of parallel connections and their duration.

Nuttcp is the end product of a number of evolution steps that started in 1984 with *ttcp* [19], a tool developed to compare different TCP stacks to help make an informed decision of which stack to place in the first BSD Unix release. *IPerf* is another well-known tool of the same family.

All the results presented in section 5 were produced by OSMeF.

4.4 Performance metrics

The main performance metric we measure in our experiments is related to network throughput: indeed, the very nature of the data-intensive applications we consider in this work is geared toward moving and ingesting large volumes of data, rather than guaranteeing low-latency access to small records. In addition to throughput, we also consider a metric related to the fairness of bandwidth allocation across competing, parallel connections. Next, we describe our metrics in detail:

- **Bulk Transfer Bandwidth (BTC).** BTC is defined as the rate that a transmitting entity implementing a standard congestion control mechanism can attain over a given network path [20]. Throughout this paper, we measure BTC as the average throughput generated during 6-minute TCP transfers between two physical or virtual network interfaces. Note that our experimental platform uses the standard Linux TCP stack.

- **Jain's fairness index (J).** Jain's fairness index [21] is a well-known fairness metric used to establish how equally a network resource is being shared. When J is equal to or approximately 1, then all connections are treated equally, in that they receive roughly the same amount of bandwidth. Instead, if J is less than 1, some connections are mistreated with respect to others, which creates an asymmetry that may severely impact application-level performance (see section 4.2).

The metrics described above are enriched by two important elements:

- **Distance.** We borrow the definition of distance between communicating entities from that used in Hadoop [22] to take informed decisions on the placement of

[4] We also used *IPerf* as an alternative to *nuttcp*, and preferred the latter for its verbosity.

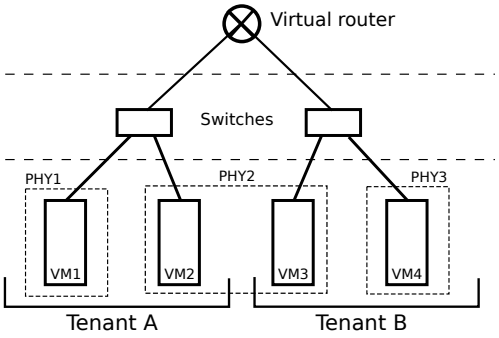

Figure 1: The distance between two VMs is equal to the number of edges data must traverse in this tree. Distance between VM1 and VM2 is 2 and distance between VM2 and VM3 is 4, even if they reside on the same physical host.

data and compute tasks. From the logical point of view, the network is represented as a tree and the distance between two nodes is the sum of their distances to their closest common ancestor. As can be seen in figure 1, this definition takes into account the virtual topology created by OpenStack.

- **CPU load.** Our definition of CPU load refers to the percentage of time spent running by a receiving or sending UNIX process. Indeed, CPU speed and memory bus bandwidth are important factors that limit the throughput achieved by a local communication. This metric is measured by *nuttcp* and reported by OSMeF on a per-connection basis.

5 Results

We now present our results in terms of the metrics defined in section 4.4, that we obtained with more than 250 OSMeF runs, using various combinations of the traffic patterns discussed in section 4.2. We consider the following scenarios:

- **Physical loopback** performance of an individual host;
- **Virtual loopback** performance of an individual VM, with flavors in the range of 1, 4, 8 and 16 virtual CPUs;
- **Host-to-Host** performance, which provides a "physical" baseline to study and compare the behavior of the virtual networks;
- **VM-to-VM** performance[5] in the following placements (the same convention is used in the remainder of the paper):
  1. Same host and same tenant;

[5]In this work we present results for a 1 virtual CPU VM, since results for different VM flavors are qualitatively similar.

2. Different host and same tenant;
3. Same host and different tenant;
4. Different host and different tenant.

In addition to the scenarios presented above, we also focus on specific characteristics of our cluster configuration, and on the OpenStack/KVM implementation of virtual networking: our goal is to characterize, in a fine-grained way, the "path" taken by a packet from its source to the destination, and pinpoint potential bottlenecks in the system. Precisely, we perform the following additional set of measurements:

- **VM-to-Host** and **Host-to-VM**: these traffic patterns allow to measure the capacity available through the virtualization layer, that is between a network interface inside the VM and the corresponding TAP interface managed by the hypervisor;
- **Host-to-Host** through a GRE tunnel. GRE is a IP-over-IP tunneling technique used by OVS to instantiate a single switch spanning multiple physical hosts.

All results are produced by averaging three (often five or more) OSMeF runs of the same scenario, varying the number of parallel connections between end-points from 1 to 50, with each connection having a fixed duration of 6 minutes. We also report the standard deviation for each scenario we consider, along with additional statistics on CPU utilization for both end-points of a connection. Note that, in our experiments, OSMeF is the only active user in the cluster: we thus eliminate interference due to multiple applications and background traffic in the system, a necessary condition in obtaining baseline results.

5.1 Loopback Performance

Armed with the motivations discussed in section 4.2, we now focus on the behavior of the loopback interface. We use OSMeF to establish a variable number of parallel, concurrent connections with both end processes (client and server) running in the same host or VM, using the loopback interface to exchange data.

Figure 2 illustrates the **aggregate** BTC we measure for both the physical and virtual interfaces, computed as the sum of the individual BTC each connection achieves in our measurements. In particular, for each individual connection, we compute the average across 5 distinct measurement runs (we report the standard deviation in Table 1). Figure 2 reports the aggregate BTC as a function of the number of parallel connections, and each line in the figure corresponds to a different VM flavor.

We observe that both the physical and virtual loopback interfaces share similar characteristics: the aggregate BTC increases as more connections are established, up to a plateau, at which BTC saturates. Clearly, the loopback behavior is dominated by the (complex) interplay between several components: the (physical or virtual) CPU and the number of cores, the speed at which data can be copied in RAM and the operating system scheduler. In particular:

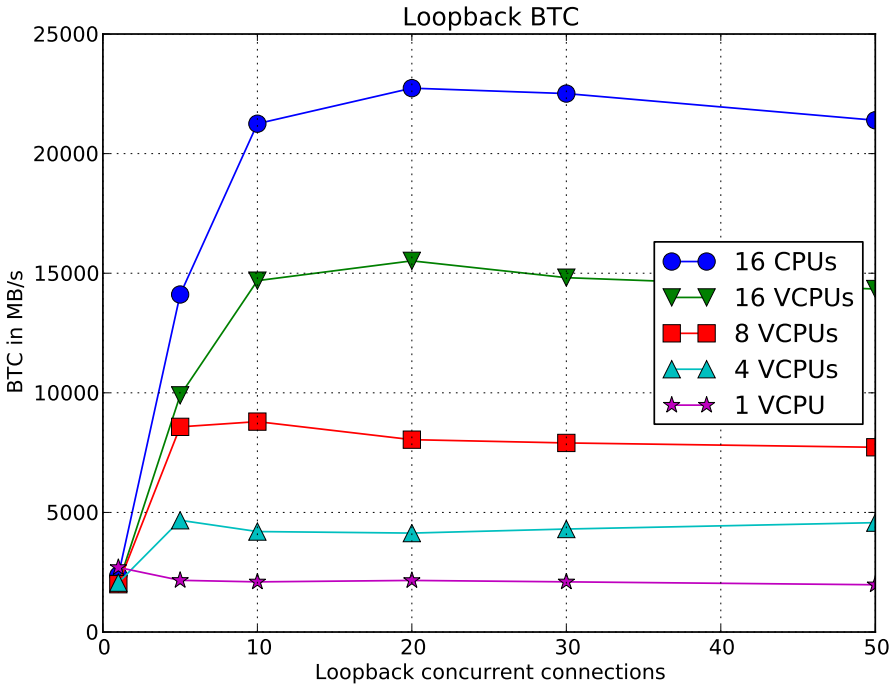

Figure 2: Loopback BTC comparison between a physical host with 16 cores (32 with hyperthreading), and VMs with 1 to 16 virtual cores.

- **physical loopback** performance is clearly dominated by the CPU load, as it can be seen in Table 1. For a single connection (Table 1a), OSMeF reports a single core supporting nearly 100% of load; for 30 parallel connections (Table 1b) CPU load is lower (recall that hosts have two exacore CPUs with hyper-threading enabled) but non-negligible;

- **virtual loopback** performance is more complex to analyze; the number of parallel connections that "saturate" BTC is proportional to the number of virtual cores, with the exception of the 1 VCPU slope, which does not cope well with parallel connections. In particular BTC doubles by doubling the number of VCPUs available, but, comparing the 16 virtual CPUs throughput with the 16 physical CPUs one, we see a 40% penalty due to virtualization. With additional measures, we observe that, despite the negligible VCPU load visible in Table 1b, the hypervisor CPU is saturated, with VM processes occupying more than 90% of CPU time.

While loopback performance is quite obviously dominated by the CPU load, it is important to note that BTC has a point of maximum, where the number of connections maximises the amount of resources available. After this point available bandwidth flattens or even starts to decrease.

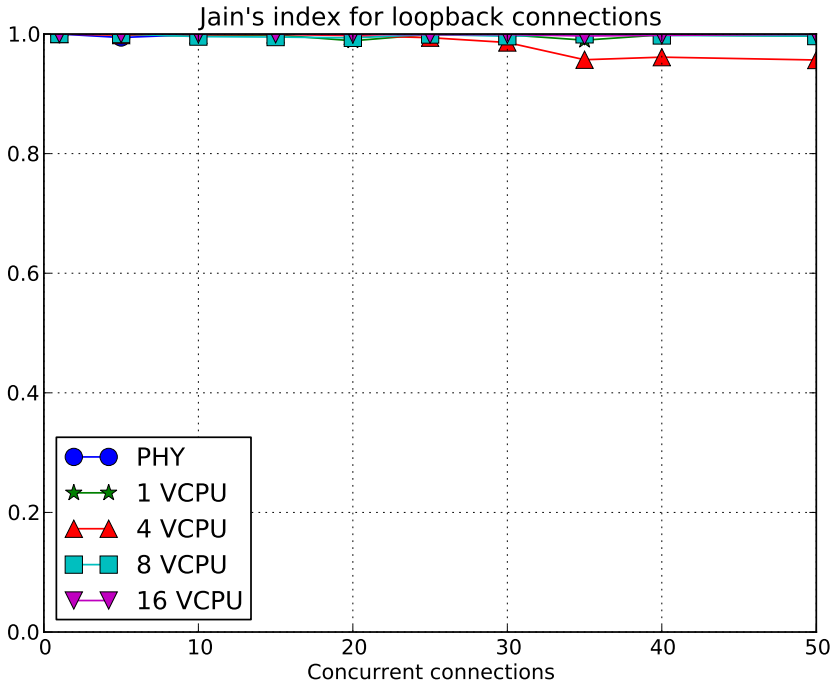

Figure 3: Jain's index for an increasing number of connections made on the loopback interface. Please note the values on the y axis, all results are within the top 5% of the fairness index.

Figure 3 illustrates, for each case discussed above, the Jain's fairness index as a function of the number of parallel connections established by OSMeF, and for different VM flavors. We remark that, for long-lived communications, each connection receives a fair share of the available bandwidth to transfer data, with values very close to 1, independently of the number of connections and across different VM flavors.

In summary, the analysis of the loopback behavior allows to draw a number of conclusions. As anticipated, many modern distributed systems[6], and in particular Hadoop, use the loopback interface for their operation. Our results indicate that:

- It is important to properly chose a VM flavor that supports and scales well with the number of parallel connections required by the application;
- Configuring and tuning data-intensive applications, such as Hadoop, requires an in-depth knowledge of the impact of virtualization to avoid saturation (by limiting the number of parallel tasks that establish connections through the loopback interface) and to ensure that available capacity is not left unused;

[6]Including OpenStack itself. Indeed, in our platform, a number of OpenStack modules instantiated on the *master* node communicate through the loopback interface.

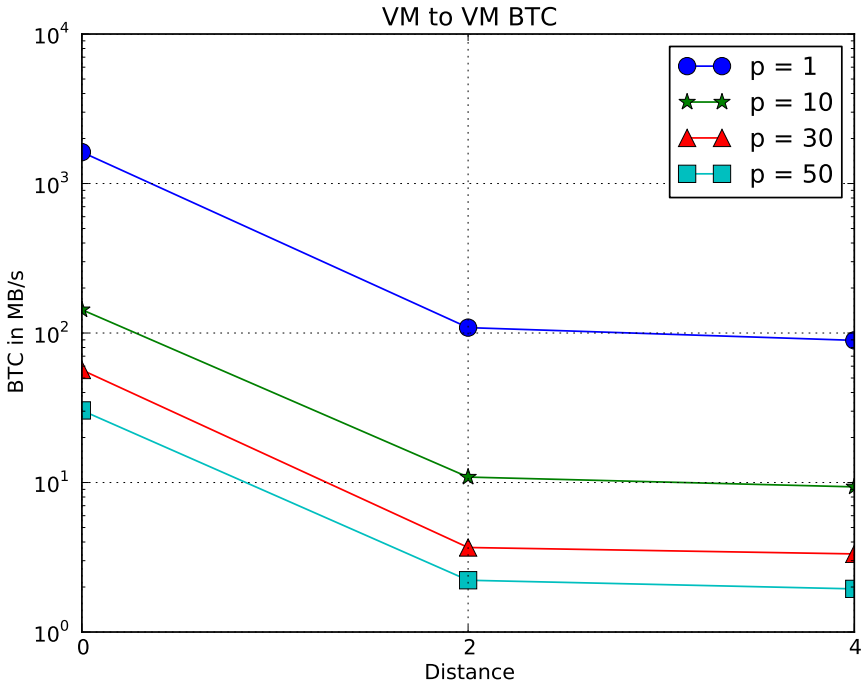

Figure 4: BTC behaviour of VM to VM communication with increasing distances and number of parallel connections (p). For distance 4 we chose to plot scenario 3 (same host, different tenant). Please note the log scale on the y axis.

- For specific traffic patterns, such as the SHUFFLE phase of Hadoop MapReduce, it is important to understand and quantify asymmetries (and hence, potential "stragglers") due to the use of the loopback interface – whose behavior is heavily influenced by CPU load – with respect to other network interfaces.

5.2 VM-to-VM performance

We now study the behavior of VM-to-VM traffic patterns: for this, OSMeF establishes a variable number of connections between different VMs, varying their placement, each hosting one end (client or server) of the connection. In addition to VM measurements, we use OSMeF to perform a series of measurements that involve the physical host to upper-bound the BTC for VM-to-VM communications. This is a summary of our results (refer to the beginning of this section for placement descriptions):

- VM-to-VM 1: in this case, the BTC is upper-bounded by the minimum of the available capacity between the VM and the underlying hypervisor, corresponding to VM-to-Host and Host-to-VM traffic patterns;

- VM-to-VM 2: the capacity of the physical network is the bottleneck in this case, with BTC upper-bounded by the Host-to-Host traffic patterns;

- VM-to-VM 3 and 4: given the platform configuration we use (suggested in [15]), the BTC for both traffic patterns we study is upper bounded by the physical network, even when VMs run in the same physical host.

Complete results, with CPU usage statistics and standard deviations for all scenarios are available in table 1.

Next, we study the behavior of VM-to-VM communications as a function of the distance we define in section 4.4. Figure 4 illustrates the **per-connection** BTC, computed as the average BTC each connections achieves in 5 consecutive runs of our measurements, where each line is representative of a measurement campaign with a different number of parallel connections. In summary, we observe that:

- Distance 0: the physical network is not involved in this case, since all communications are established within the same physical host. As anticipated above, the bottleneck that determines the overall performance of this traffic pattern is related to VM-to-Host and Host-to-VM communications, which suffer from overheads due to network virtualization;

- Distance 2: in this case, the physical network is the bottleneck. Virtualization overheads are low, as the loss in BTC for this traffic pattern is roughly 4% with respect to the physical upper-bound. Additionally, we remark that VM-to-VM communications at distance 2 achieve a BTC that is one order of magnitude lower than what can be obtained at distance 0;

- Distance 4: also in this case the physical network constitutes the bottleneck for the attainable BTC, even for connections between VMs in the same physical host. We note a further drop in performance, as compared to distance 2, due to routing overheads to move data between different tenants.

Overall, the results we show in Figure 4 indicate that the total network capacity is shared consistently among competing connections: a single connections uses virtually all of the available network capacity, whereas on average, for example, 10 connections receive roughly 1/10-th of the total capacity.

Figure 5 reports the Jain's fairness index for VM-to-VM traffic patterns, as a function of the number of connections. Our results indicate that each (long-lived) connection receives a fair share of the available capacity. It is interesting to notice that this is not the case for Host-to-Host communications: in this case, the hypervisor operating system does not distribute evenly the network capacity among competing flows, a result that is corroborated also by some recent works [23]. In addition, our results indicate a 2% performance loss of Host-to-Host communication through a GRE tunnel, which imposes a 15% overhead in the CPU utilization.

In summary, our results cast light on the impact of network and system virtualization for the applications we consider in our work. **VM placement** plays a crucial

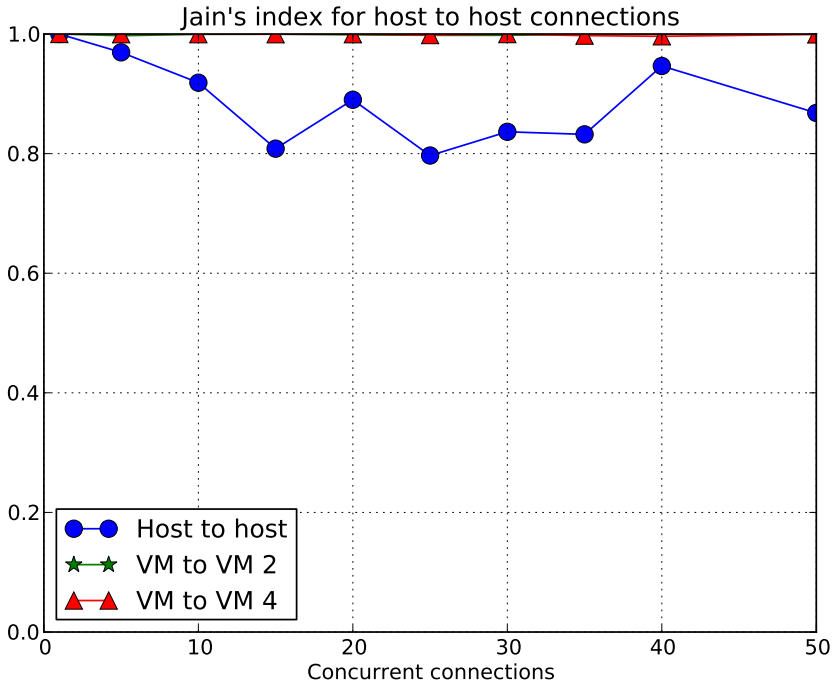

Figure 5: Jain's index for an increasing number of connections made between different physical hosts.

role in determining application-level performance. For SHUFFLE-like traffic patterns, sub-optimal VM placement might contribute to the creation of "stragglers" due to the inherent asymmetry of the BTC attainable between VMs, depending on their distance.

Our results can thus be used to inform VM placement strategies to cope with application requirements. Moreover, we observe that the architecture suggested in [2], whereby data-intensive applications are deployed by separating across different tenants *data* and *compute* layers, may suffer from a severe performance degradation. The cluster configuration we used in our experiments – which follows OpenStack guidelines – is inappropriate for inter-tenant communications that occur on the same physical host. It is thus necessary to devise new mechanisms that exploit **host-locality** to improve network performance and avoid unnecessary traffic routing.

6 Conclusions and future work

Understanding the consequences of machine and, more generally, cluster virtualization on communication performance of cloud applications is fundamental for their correct operation. This is particularly true for data-intensive scalable computing,

Table 1: Global overview. BTC is measured in MB/s and the CPU value is a percentage of occupation. The last four results are variations on the placement of VMs, in this order: same host and same tenant, different host and same tenant, same host and different tenant, different host and different tenant. Standard deviation is shown in parenthesis)

(a) Results for 1 connection. Average and standard deviation calculated over at least 5 samples.

		BTC	CPU %	
Measure	Distance	Avg	Avg RX	Avg TX
PHY loopback	-	2379.36 (77.04)	89.5 (1)	98 (0.8)
VM loopback	-	2596.02 (248.07)	66 (1.3)	42 (23.3)
VM to host	-	1313.57 (100.16)	43.2 (2.4)	99 (0)
host to VM	-	1678.52 (11.64)	92.6 (2.2)	87.2 (5.3)
host to host	2	112.23 (0)	5.4 (0.8)	5.0 (0)
host to host GRE	2	110.36 (0)	17.8 (0.7)	20.33 (1.7)
VM to VM 1	0	1589.43 (155.97)	80.67 (5.9)	97.2 (2)
VM to VM 2	2	108.66 (0.01)	20.9 (0.3)	3.9 (0.9)
VM to VM 3	4	89.37 (0.74)	20 (0)	3.5 (0.8)
VM to VM 4	4	94.72 (0.48)	19 (0)	3.2 (0.4)

(b) Results for 30 connections. Average and standard deviation calculated by first averaging all samples for each connection and then aggregating the connections.

		BTC		CPU %	
Measure	Distance	Sum	Avg	Avg RX	Avg TX
PHY loopback	-	22496.8	749.89 (5.5)	60.8 (0.5)	42 (0.7)
VM loopback	-	4013.3	133.78 (4.64)	4.6 (0.2)	5.2 (0.3)
VM to host	-	1082.87	36.1 (0.37)	1 (0)	3 (0)
host to VM	-	1552.75	51.76 (1.24)	3 (0)	2 (0)
host to host	2	112.33	3.74 (1.55)	0.6 (0)	0 (0)
host to host GRE	2	110.4	3.68 (1.90)	1.4 (1)	0.2 (0.6)
VM to VM 1	0	1642.17	54.74 (2.14)	2.4 (0.4)	3 (0.2)
VM to VM 2	2	110.7	3.69 (0.16)	0 (0.2)	0 (0)
VM to VM 3	4	98	3.27 (0.33)	0.7 (0.3)	0 (0)
VM to VM 4	4	95.74	3.19 (0.13)	0.7 (0.3)	0 (0)

where I/O performance plays a crucial role in determining the overall rate at which data analysis tasks can proceed.

In this work, we described our measurement results and their implications in light of an appropriate approach to deploy and tune data-intensive applications. Essentially, this work helps to inform the design of mechanisms to perform VM placement on physical servers, and to appropriately tune communication parameters of Hadoop-like applications.

Currently, we are extending the traffic patterns supported by our measurement tool (for example to account for n-way communications), and we plan to run a range of new measurement campaigns to understand and quantify the impact of interference caused by background traffic and multiple, coexisting applications running in our platform. In addition, we will complement our study by considering the impact of virtualization on disk I/O performance.

References

[1] Apache, "Hadoop." [Online]. Available: http://hadoop.apache.org/

[2] VMWare, "Hadoop virtualization extensions on VMware vSphere 5." [Online]. Available: http://www.vmware.com/files/pdf/Hadoop-Virtualization-Extensions-on-VMware-vSphere-5.pdf

[3] OpenStack Foundation, "OpenStack." [Online]. Available: http://openstack.org

[4] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *Proc. of INFOCOM'10*. IEEE Press, pp. 1163–1171.

[5] E. Walker, "Benchmarking amazon ec2 for high-performance scientific computing," *Usenix Login*, vol. 33, no. 5, pp. 18–23, 2008.

[6] P. Apparao *et al.*, "Characterization of network processing overheads in xen," in *Proc. of VTDC '06*, Washington, DC, USA, 2006.

[7] Y. Mei *et al.*, "Performance measurements and analysis of network i/o applications in virtualized cloud," in *Proc. of CLOUD 2010*. IEEE Press, 2010, pp. 59–66.

[8] J. Han, H. Makino, and M. Ishii, "Design and performance evaluation for hadoop clusters on virtualized environment," in *Proc. of ICOIN '13*. IEEE, 2013, pp. 244–249.

[9] M. Zaharia *et al.*, "Improving mapreduce performance in heterogeneous environments," in *Proc. of OSDI '08*, Berkeley, CA, USA, 2008.

[10] L. Rizzo and G. Lettieri, "Vale, a switched ethernet for virtual machines," in *Proc. of ACM CoNEXT '12*, 2012, pp. 61–72.

[11] D. Crisan, R. Birke, G. Cressier, C. Minkenberg, and M. Gusat, "Got loss? get zovn!" IBM Research, Tech. Rep. RZ 3840, March 2013.

[12] Amazon, "Web services console." [Online]. Available: http://console.aws.amazon.com/

[13] Rackspace, "Rackspace private cloud installation manual." [Online]. Available: http://www.rackspace.com/knowledge_center/article/rackspace-private-cloud-installation-prerequisites-and-concepts

[14] "Open vswitch." [Online]. Available: http://openvswitch.org/

[15] OpenStack Foundation, "Openstack networking administration guide," grizzly 2013.1. [Online]. Available: http://docs.openstack.org/grizzly/openstack-network/admin/content/use_cases_single_router.html

[16] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. of OSDI '04*, 2004, pp. 107–113.

[17] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput," *SIGCOMM Comput. Commun. Rev.*, no. 4, 2002.

[18] "Nuttcp." [Online]. Available: http://www.nuttcp.net

[19] R. Prosad *et al.*, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE Network*, no. 6, 2003.

[20] M. Mathis and M. Allman, "A framework for defining empirical bulk transfer capacity metrics," in *Proc. of 51st Internet Engineering Task Force*. IETF, 2001.

[21] R. Jain, D.-M. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *CoRR*, vol. cs.NI/9809099, 1998.

[22] T. White, *Hadoop The Definitive Guide*, 2nd ed. O'Reilly and Yahoo! Press, October 2010.

[23] G. Urvoy-Keller, D. M. L. Pacheco, and H. S. Ha, "Networking in a virtualized environment: the TCP case," in *Proc. of CloudNet'13*. IEEE.