Evaluating Streaming Strategies for Event Processing across Infrastructure Clouds

Radu Tudoran[¶], Kate Keahey [†], Pierre Riteau[§], Sergey Panitkin[‡] and Gabriel Antoniu[*]
[¶]*IRISA/ENS Cachan Rennes, France*
radu.tudoran@inria.fr
[*]*INRIA, Rennes Bretagne Atlantique Research Center, France*
gabriel.antoniu@inria.fr
[†]*Argonne National Laboratory, Argonne, IL, USA*
keahey@mcs.anl.gov
[§]*Computation Institute, University of Chicago, IL, USA*
priteau@uchicago.edu
[‡]*Brookhaven National Laboratory, Upton, NY, USA*
panitkin@bnl.gov

***Abstract*—Infrastructure clouds revolutionized the way in which we approach resource procurement by providing an easy way to lease compute and storage resources on short notice, for a short amount of time, and on a pay-as-you-go basis. This new opportunity, however, introduces new performance trade-offs. Making the right choices in leveraging different types of storage available in the cloud is particularly important for applications that depend on managing large amounts of data within and across clouds. An increasing number of such applications conform to a pattern in which data processing relies on streaming the data to a compute platform where a set of similar operations is repeatedly applied to independent chunks of data. This pattern is evident in virtual observatories such as the Ocean Observatory Initiative, in cases when new data is evaluated against existing features in geospatial computations or when experimental data is processed as a series of time events. In this paper, we propose two strategies for efficiently implementing such streaming in the cloud and evaluate them in the context of an ATLAS application processing experimental data. Our results show that choosing the right cloud configuration can improve overall application performance by as much as three times.**

I. Introduction

By providing a way to lease compute and storage resources on demand, for a short amount of time, and on a pay-as-you-go basis, infrastructure cloud computing revolutionized the way in which we approach resource procurement. This innovation is proving fundamental to the construction of systems capable of rapid scaling and high reliability. At the same time, however, as has been shown by multiple evaluation efforts [1], [2], [3], the performance trade-offs inherent in current virtualization technology mean that data-intensive applications are often not the best fit for infrastructure clouds and consequently cannot leverage the advantages of cloud computing.

This technological shortcoming is particularly impactful in the Big Data era. An increasing number of Big Data applications conform to a pattern in which data processing relies on streaming the data to a compute platform where a set of similar operations is repeatedly applied to independent chunks of data. This pattern is evident in virtual observatories such as the Ocean Observatory Initiative [4], in cases when new data is evaluated against existing features in geospatial computations (e.g., FluMapper[5]) or when experimental data is processed as a series of time events [6], [7]. These applications often need real-time response time, for example, to adaptively redeploy sensors in a virtual observatory or to support an experiment. Moreover, they need to scale rapidly to fluctuating request numbers as the detection of new phenomena generates more questions and thus necessitates more processing. This type of application clearly could benefit from on-demand resource availability provided by cloud computing, but they are often costly or hard to structure because of difficulties and inefficiencies in data management in the cloud.

In this paper, we propose two strategies for efficiently implementing streaming in the cloud for such applications. The first strategy seeks to overlap computation and communication by streaming data directly to the nodes where the computation takes place, in such a way that the rate of data streaming keeps pace with computation; the second strategy relies on first copying data to the cloud and then using it for computation. We evaluate these strategies in the context of an Atlas application application [8], [9] processing experimental data in terms of both performance and cost. Our results indicate the ways in which communication in the cloud can interfere in computation. We show that choosing the right cloud configuration can improve overall application performance by as much as three times and can significantly reduce the cost.

The paper is organized as follows. Section II describes the features of infrastructure clouds relevant to this paper, defines the type of applications and the interaction pattern addressed in the paper, and describes the two streaming strategies evaluated in the paper. Section III presents our experimental setup and describes the experiments we per-

formed to evaluate the two strategies. Section IV discusses their respective costs. In Section V we discuss related work. In Section VI we summarize our conclusions.

II. Background

A. Cloud Storage Basics

Infrastructure-as-a-Service (IaaS) clouds allow users to deploy virtual machines (VMs) *instances*, thus leasing computing capacity for a short amount of time usually paid for per hour. In addition to computing capability, cloud providers offer several types of storage with different availability, access, and performance characteristics, at different price. These types of storage are as follows.

- **Ephemeral storage** - is the local virtual disk attached to a deployed instance. Its storage capacity can reach up to many terabytes depending on the instance type, but it persists only for the lifetime of the instance and is thus subject to sudden loss when the instance fails unexpectedly. The storage is free in the sense that it comes with the cost of leasing an instance.
- **Persistent attached storage** - provides network-attached block-level storage volumes that can be attached to a running instance and exposed as a storage device within that instance. The lifetime of this type of storage is independent of the instances that mount it, i.e., it can be reused by many instances. Charges vary based on the size of volumes used and period of usage. Examples of persistent attached storage are Amazon Elastic Block Storage (EBS) [10] and Azure drives [11].
- **Cloud storage** - offers data storage as binary objects (blobs), usually in a two-level namespace structure (e.g., blobs and containers). This type of storage typically offers different levels of service, for example, in terms of access time or redundancy, at different prices. Users typically get charged based on the data size and on the storage time. Examples of cloud storage include Amazon Simple Storage Service (S3) [12], Amazon Glacier [13], Microsoft Azure BLOBs [14], [15], and Google Cloud Storage [16].

While cloud storage has been used in the past to provide a file system to support a running application [7], this type of storage is increasingly evolving to provide an archival capability. Therefore, in this paper we focus on the first two types of storage.

B. Streaming Applications

Increasingly more applications conform to a pattern in which an operation is applied to many independent chunks of data. This can happen because an operation is applied to data regularly produced by sensors in virtual observatories such as the Ocean Observatory initiative [4], when new data is evaluated against existing features in geospatial computations [17], [18] or when experimental data is processed as a series of time events. The data typically is streamed from the data source to a cluster or a cloud, where the computation takes place, either chunk by chunk or by bunching multiple chunks together, with the streaming constituting a significant component of the overall computation. Optimizing this scenario requires streaming to keep pace with computing. In this paper, we call such applications "streaming applications", and we investigate how they can best be mapped onto infrastructure cloud storage facilities.

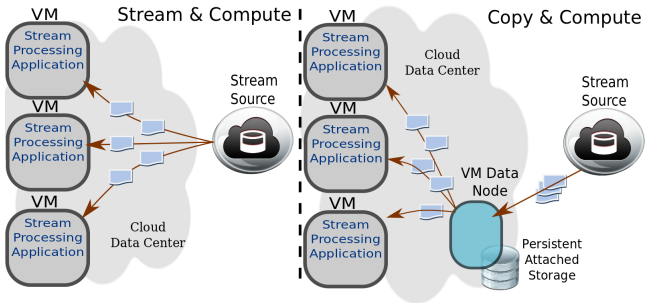

Figure 1. The two main cloud streaming scenarios: Stream&Compute (left) and Copy&Compute (right)

As an example of a streaming application we use a data analysis code from an ATLAS experiment at the Large Hadron Collider [8], [9]. The application performs data analysis searches in a channel where the Higgs decays into t anti-t quarks. The experimental data is collected as successive time events, one event corresponding to the aggregated readings from the ATLAS sensors at a given moment. Because the amount of collected data is on the order of tens of petabytes, efficient processing is of significant concern.

The application is written in the ROOT [19] version of C and is executed as a workflow using the ProofLite framework. The typical computation pattern consists of reading the job and data description, followed by successive reads of events that need to be processed. The data has been divided into independent chunks by annotating it with metadata, thus eliminating the need for communication between processes but adding to the bulk of data that must be streamed to each worker. The data-streaming technology used to run the application is CERN's xrootd client-server framework.

C. Streaming Scenarios

We consider two scenarios for managing data streaming on infrastructure cloud facilities. In each scenario we work with a data source and then deploy a set of compute instances (or compute VMs) on a remote cloud. Each compute instance is running a number of application workers that process the data. The architecture of these scenarios is presented in Figure 1.

- **Stream&Compute** – In this scenario data is streamed directly from the data source to the compute instances where it is ingested and processed by the worker processes. The events are processed in memory as they are streamed.

- **Copy&Compute** – In this scenario we allocate some persistent storage and an additional instance that mounts this storage in order to make it accessible to all compute instances. The streaming is split into two phases. First, we copy the data from the data source to this persistent attached storage. Second, once all data is available locally on the cloud, it is streamed locally to the compute instances so that the computation can begin.

The advantages of Stream&Compute are that (1) it provides better response time on a case by case basis where not all of the data chunks to be computed are available up front or need to be processed to yield a result; (2) it has the potential to overlap computation and communication, thus potentially shortening the time to compute a group of events; and (3) it uses storage provided with instances, thus making the computation potentially cheaper. At the same time, we note that at large scales, network saturation can slow the data-streaming rate to the point where it no longer keeps pace with computation potentially necessitating redistribution across different clouds.

The advantage of Copy&Compute is that it relies on persistent storage, thus leading to easier repair in cases where an instance is terminated unexpectedly: while any terminated computations will have to be rerun, the data will not have to be resent over wide-area network.

III. Experiments

The goal of the experiments is to assert the validity of our hypothesis and to determine the best options for processing stream data using the cloud. To this purpose, we analyze a series of experiments that compare the two main scenarios for using the cloud: Copy&Compute and Stream&Compute. We also discuss questions regarding scalability, bottleneck limits, the relation between the CPU and throughput, and the choice of instance types.

A. Experimental setup

The experiments presented here were run on FutureGrid [20], using Hotel (a University of Chicago cloud configured with Nimbus version 2.10.1 [21]) and Sierra (a San Diego Supercomputing Center cloud configured with OpenStack Grizzly [22]), as well as on the Azure commercial cloud [23]. To implement persistent attached storage we leveraged OpenStack Block Storage volumes for OpenStack and Azure drives [11] for Azure. All VMs used in the experiments run a CentOS 6.3 operating system. The Small instances used in FutureGrid offer 1 virtual CPU, 2 GB of memory, and 20 GB of local storage. In Azure the Small VM roles have 1 CPU, which is guaranteed to be mapped to an unshared physical CPU, 1.75 GB memory, 200 GB local storage, and 100 Mbps of bandwidth (note that while this is the advertised bandwidth, our experience is that the effective bandwidth is higher than 100 Mbps). Medium instances provide twice as many resources as do Small instances (2 virtual CPU, 4 GB memory, and 40 GB ephemeral disk); and, similarly, Large instances are 4 times larger than Small instances. The interconnection network between the Sierra and Hotel sites is a 10 Gbps network; the physical nodes are connected via a 1 Gbps Ethernet network.

The Hotel Nimbus cloud was used to host the data sources (i.e., the nodes that hold the data) for all experiments. The Sierra OpenStack cloud was used to host the compute nodes for all FutureGrid experiments. The Azure West US Data Center (located in California) was used to host the compute nodes for all Azure experiments.

The application we experimented with is composed of small data computation units of similar size, referred to as *events*. Small size differences appear because the events aggregate measurements of different aspects of a phenomenon, which may or may not be detected at a given point. The number of events processed is therefore a good measure of the progress of the application. We used the following metrics for our experiments.

- **Compute Rate** – this is the amount of events that are processed in a time unit; the metric unit is events/second.
- **Data Rate** – this is the amount of data that is read or acquired in a time unit; the metric unit is megabytes/second.

For the Stream&Compute case, we measured the compute rate by running the program with a set number of events and measuring on each compute node how long it took from start to finish of the computation. For the Copy&Compute case, the experiment involved two phases: a remote site copy followed by the computation on local data. For the first phase, we used the "time" command to measure how long a remote copy took using "scp". This was then added to the time taken by the application to complete processing over the set of events transferred. We measured the data rate by dividing the amount of input data by the total time to complete its processing. For the Copy&Compute case, this total time included both the remote data copy and running the application. We increased the input data size as we scaled up the number of instances (i.e., the amount of data processed with 32 VMs is equal to 32 times the amount of data for one VM).

For each measurement presented in the charts, 100 independent experiments were performed. The values shown represent the averages.

B. Copy&Compute vs. Stream&Compute

We first present average per VM compute rates and per VM data rates to application VMs deployed on FutureGrid small instances as well as Azure small instances. The average data rate per VM is computed by taking the total data rate for a given application VM configuration and dividing it by the number of VMs on which the application was

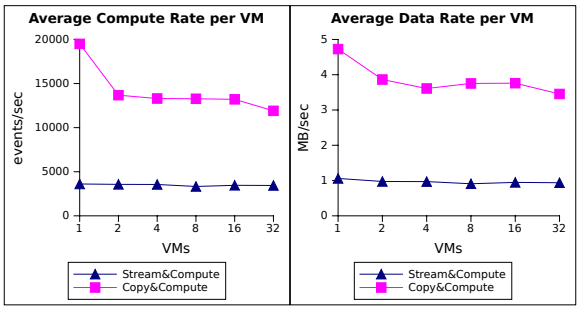

Figure 2. Comparing Copy&Compute and Stream&Compute scenarios in FutureGrid, considering average compute (left) and data (right) rates per VM

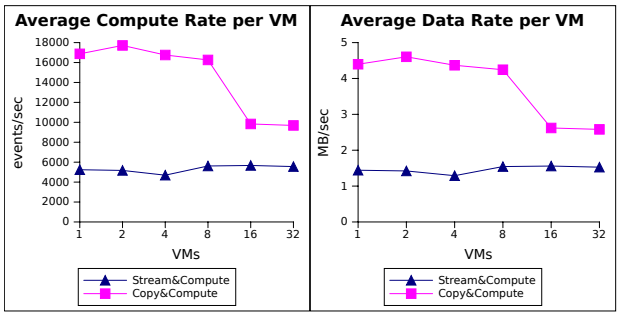

Figure 3. Comparing the Copy&Compute and Stream&Compute scenarios in Azure, considering average compute (left) and data (right) rates per VM

deployed. Figure 2 shows the results for FutureGrid and Figure 3 for Azure. In addition to the data shown in the figures, we note that the results for the Copy&Compute operation show a higher variability, having a coefficient of variation (i.e., standard deviation / mean) of ∼20%, than do the results for the Stream&Compute data, which has a coefficient of variation of ∼10%. The remote copy phase is mainly responsible for this high variability; the variability on local dissemination is very low.

We see that in both cases Copy&Compute outperforms Stream&Compute. The Copy&Compute strategy offers three to four times better performance than does Stream&Compute. This is contrary to our expectation; we expected the Stream&Compute method to be faster because of overlapping computation and communication. The overlapping of the computation with communication is available at the level of the framework, with no particular optimizations set up on our side. Hence, similar results are expected for any framework that provides such a behavior: acquiring the data for the next step of the computation while the available data for the current step is being processed.

In addition, we see a drop in per VM average data and compute rates for the Copy&Compute case on Azure once we reach 16 VMs. This reflects the bandwidth limitation of 100 Mbps to the Azure EBS node from which data is shared among the application nodes at a potential throughput of about 4 MB/s. As more nodes want to access the data on the EBS node at the same time the maximum bandwidth of this node is reached, and the resulting contention slows the data access. This situation could potentially be fixed by

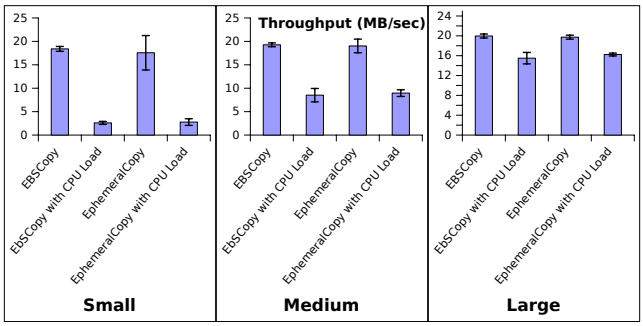

Figure 4. Asserting the data acquisition throughput with respect to the CPU load for three different instance types

either assigning the EBS node to a larger instance or by striping the access to the data over multiple nodes.

To understand why Copy&Compute outperforms Stream&Compute, we first compared data throughput of a VM using an ephemeral disk (as in the Stream&Compute scenario) and a VM using EBS (as in the Copy&Compute scenario) by copying large data files (0.5 GB) between Hotel and Sierra using the Unix "scp" command. The results, labeled EphemeralCopy and EBSCopy, respectively, are shown in Figure 4. We found that the throughput for small instances (the ones used in our experiments) in both cases was almost the same, so that the high throughput was not correlated with using EBS. However, when we induced a 100% CPU load using the Unix "stress" tool with 8 threads spinning over sqrt ("–cpu N") and spinning over the pair (memalloc/free) ("–vm N"), we saw the throughput diminish significantly – to roughly one-fifth of the initial throughput – but again roughly equally in the EBSCopy and EphemeralCopy case. In other words, the drop in throughput is correlated to processing because not enough CPU cycles are available both to do the processing and to manage the incoming data. Since the EBS node in Copy&Compute is dedicated to just handling the network traffic and the compute nodes in Stream&Compute are using the available cores for both computation and handling the network traffic, fewer cycles are available for handling the network traffic resulting in slower transfer.

The impact of CPU on I/O has been demonstrated before; in HPC systems [24], dedicating a core for handling the data has been shown to improve the overall performance and in [25], the authors report a significant impact on data throughput due to virtualization. TCP throughput degradations due to sharing the CPU between cloud VMs are discussed also in [26]. In [27], the authors identify the CPU as one of the bottlenecks that need to be overcome for data transfers.

To see how additional cores may affect this scenario, we repeated these experiments for different types of instances: Small (1 CPU), Medium (2 CPUs), and Large (4 CPUs), again shown in Figure 4. The results show that increasing the number of cores does alleviate the problem; however, since no cores are dedicated to processing data transfer (as is effectively the case in the EphemeralCopy scenario),

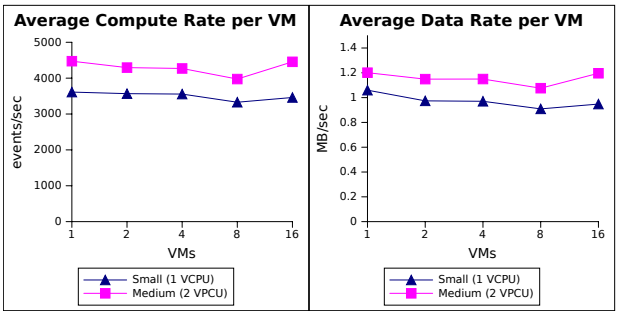

Figure 5. Remote streaming with 1 worker per VM for different type of instances. The performance metrics show the average compute rate per (left) and the data rate (right) per VM

the data transfer in the presence of CPU load is never as efficient as without it. To see what impact increasing the number of cores will have on the Stream&Copy, we ran an experiment in which we placed the application VMs in the Stream&Copy scenario on Medium (2 virtual CPUs) instance types and compared them with our previous results where the application VMs are deployed on FutureGrid Small (1 virtual CPU) instances. The results shown in Figure 5 present the data rate and the compute rate for the Stream&Compute scenario under those two types of instances.

As we expected, dedicating more cores to the computation improves the overall performance. This result is consistent with the experiment presented in Figure 4. The performance, however, improves only slightly, by ∼30% for the compute rate (e.g. going from 3,400 to 4,400 events/second for 16 VMs) and by ∼25% for the data rate (e.g. going from 0.95 to 1.2 MB/s for 16 VMs). As before, since no cores are actually dedicated to processing data transfer, the threads of the CPU-intensive program are able to claim a large share of attention from the CPU.

Second, we asked the question why streaming from a remote node to application VMs (as in Stream&Compute) is more impacted by the CPU activity than streaming from a local (i.e., EBS) node (as in Copy&Compute). To answer it, we used the "netem" tool to increase latency between the EBS node and application nodes of the Copy&Compute scenario to be equivalent to the remote latency used in the Steam&Compute scenario (i.e., 84 ms). The result shows that the performances (i.e., both data and compute rate) drop to the level of the Stream&Compute scenario, with the data rate decreasing from 7.5 MB/s to 1.2 MB/s. This shows that the difference in latency is responsible for the difference in performance; we believe it is tied to the data buffering mechanism in the Atlas application.

C. Scalability for streaming

The purpose of the next set of experiment was to probe the scalability barrier of the Stream&Compute scenario: we expected that for large enough numbers of application VMs the network between the data source and the application

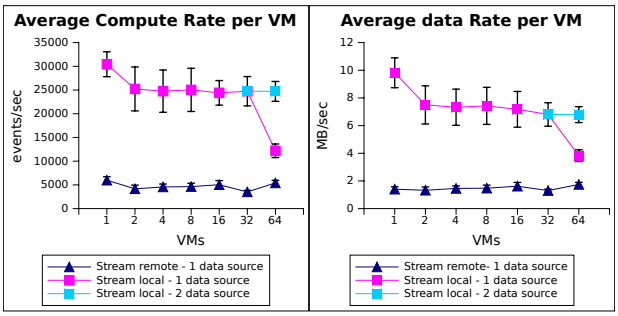

Figure 6. Scaling the compute VMs for local and remote stream processing. The performance metrics show the average compute rate per (left) and the data rate (right) per VM

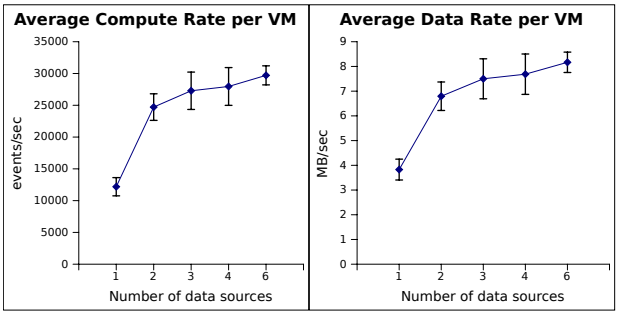

Figure 7. Scaling the data sources for local streaming, while keeping the number of compute VM fixed to 64. The performance metrics show the average compute rate per (left) and the data rate (right) per VM

VMs would become saturated and thus the data would start arriving at the VMs at a slower rate than it could be consumed, effectively "starving" the application and thus slowing it. For this experiment we placed the application nodes on the Hotel cloud since it has more nodes available for a larger-scale experiment. We then looked at interactions with two stream data sources: a remote source running on the Sierra cloud and a local one.

The results of the experiment are presented in Figure 6. We observe that working with the remote data source, we were unable to saturate the network for the scale we tested (up to 64 application VMs) and thus the per-VM performance remained stable. Since testbed limitations prevented us from scaling further than 64 application VMs, we instead placed the data source on the Hotel cloud (i.e. local to the application compute nodes); our intention was to see whether the higher available bandwidth between data source and application nodes would allow us to saturate the network sooner. This situation was in fact observed (Figure 6) when scaling to 64 VMs: the average compute and data rates drop by about half as the aggregate throughput at which the application VMs can receive exceeds the data sources ability to send. We hypothesized that this can be fixed by streaming from multiple data sources, and this proved to be the case, as can be again seen in Figure 6. We therefore conclude that it is possible to scale beyond this bottleneck by adjusting the number of data sources with respect to the aggregated data rates of the compute VMs. Similar results in non-cloud scenarios were also shown by [28].

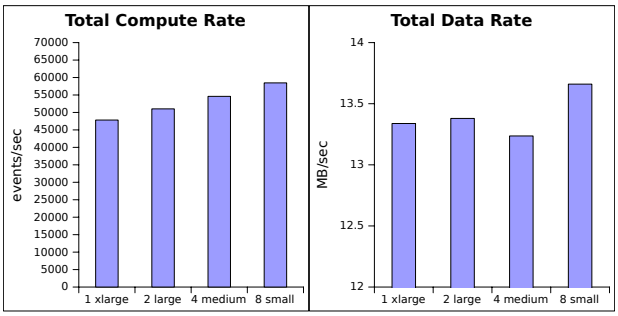

Figure 8. Comparing the performances of the compute rate (left) and data rate (right) that can be achieved for different combinations of instance types within the same budget (Small, Medium, Large, and xLarge VMs)

We further probed to see how much improvement can be obtained by adding data sources. In our next experiment we kept the number of application compute VMs fixed at 64 and increased the number of data sources. The results are shown in Figure 7. After an initial significant increase in performance when moving from 1 to 2 data sources, the performance gains are smaller when going beyond 2 data sources. This upper bound is due to the physical infrastructure, as we reach the limit of the cluster capabilities (i.e., network switches and Ethernet links). We recall that the nodes have 1 Gigabit Ethernet link, which means that the average aggregated throughput to the data sources, of about 100 MB/s to each, almost consumes it completely.

D. Choosing the VM Instance Types

The cloud cost model for cloud providers such as Amazon EC2 or Microsoft Azure links compute power of instances to their price. For example, Microsoft Azure starts from s base price charged for a Small instance with 1 CPU and doubles it as the instances become more powerful (Medium = 2x, Large=4x, xLarge=8x), consistently with the resources provided (CPUs, memory, ephemeral storage). Thus, buying 1 Extra-Large instance is intended to be roughly equivalent to buying 2 Large, 4 Medium, or 8 Small instances. We therefore investigated next whether this is indeed the case in the context of the Copy&Compute scenario.

The interesting question is whether it makes any difference from the performance point of view to choose one type of instances or another in a setup such as the Copy&Compute scenario. Figure 8 presents the results corresponding to the Copy&Compute performance obtained when using 8 Small, 4 Medium, 2 Large, or 1 xLarge OpenStack VMs. In each case, the computation consists of 7 processes computing the events and 1 process acquiring data from a remote data source, potentially executing within a different VM. The VMs are run in a multitenancy environment. The results show that the data rates for all four cases are similar but slightly better for Small VMs, indicating that scheduling between VMs on the same hypervisor provides better isolation for computation and communication than does scheduling functions within one VM. The computation rates improve steadily for smaller VMs, with a 20% performance improvement for the Small versus xLarge instances. The reason is performance degradation due to the memory and disk interference that appear between processes.

This observation indicates that, given the existing balance of inter- and intra-hypervisor scheduling policies, it is better to take, within the same budget, smaller VMs rather than bigger ones. Additionally, such a strategy can potentially harvest more physical resources, since the VMs can wind up distributed across multiple nodes.

IV. The Cost of Streaming

Another interesting aspect to evaluate is the cost of streaming to the cloud for the two streaming strategies. We modeled this cost using the following parameters:

- VM_{Cost}: the hourly cost for renting a VM instance;
- $N_{VMsComp}$: the number of VM instances used for computation;
- $N_{VMsData}$: the number of VM instances used for data acquisition (i.e., in the Copy&Compute scenario);
- $CompRate_{Total}$: the number of events processed per second in the system, for Stream&Compute (SC) or Copy&Compute (CC) scenarios;
- $TotalEvents$: the total number of events to be processed;
- $Size_{Ev}$: the total size of the events to be processed;
- $Storage_{Cost}$: the cost of storing the data on persistent storage for the duration of the computation (i.e., in the Copy&Compute scenario);
- GBH_{Cost}: the cost of storing 1 GB of data for 1 hour, which is computed as the $Storage_{Cost}$ divided by the number of hours in a 30-day period (the considered number of days in a month), i.e., 720 hours.

Using these parameters, we can express the cost for processing a given amount of events as the time needed to process them multiplied by the corresponding cost of the VM instances used.

$$Total_{Cost} = \frac{TotalEvents}{CompRate_{Total}} * (N_{VMsData} + N_{VMsComp}) * VM_{Cost} + Storage_{Cost} \quad (1)$$

The typical cost of instances similar to those we used in our computation is $0.09 per hour for Azure Small instances (as used in our experiments) and $0.06 for Amazon VM instances. The cost model for persistent attached storage typically considers long-term storage and is quoted in gigabytes per month but actually is calculated per day. For example, Azure charges $0.07 for 1 GB per month, whereas Amazon charges $0.095 per gigabyte per month.

As an illustration, in our particular use case, we need to consider 320 million events run on 32 compute nodes over 41 minutes for the Stream&Compute scenario and 15 minutes for the Copy&Compute scenario. The latter requires one extra VM instance for the data node and 5 GB of attached persistent storage additionally leased. Overall, we

reach a global cost of $1.33 for the Stream&Compute scenario compared with $0.48 for the Copy&Compute scenario, in other words, the Stream&Compute scenario is 2.77 times more expensive. Extrapolating to realistic application sizes that process billions of events for multiple hours (e.g. for 1 day) using much larger numbers of VM instances, this factor can have a substantial budget impact.

To express the price ratio between the two streaming options in the general case, we start by expending the $Storage_{Cost}$ from Equation 1 in terms of the price to store the events for the compute time interval: $\frac{TotalEvent}{CompRate_{Total}} * Size_{Ev} * GBH_{Cost}$. Hence, by expressing the price ratio between the cost of Stream&Compute and the cost of Copy&Compute, we get the following.

$$\frac{Total_{CostSC}}{Total_{CostCC}} = \frac{\frac{TotalEvents}{CompRate_{TotalSC}}}{\frac{TotalEvents}{CompRate_{TotalCC}}} * \frac{N_{VMsComp} * VM_{Cost}}{(N_{VMsData} + N_{VMsComp}) * VM_{Cost} + Size_{Ev} * GBH_{Cost}} \quad (2)$$

We note that the GBH_{Cost} is comparable with the VM_{Cost} for data sizes over 720 GB. The intuition is that the price to store 1 GB of data in the public clouds (e.g., Azure or Amazon) for 1 month (i.e. 720 hours) is comparable with the price of leasing one Small VM for an hour. Hence, the amount of data to be stored per hour which matches the VM cost is 720 GB. Thus, we can approximate the hourly cost to store the data as $\frac{Size_{Ev} * VM_{Cost}}{720}$. By applying this in Equation 2 we can approximate the price ratio between the two streaming options to the following.

$$\frac{Total_{CostSC}}{Total_{CostCC}} \approx \frac{CompRate_{TotalCC}}{CompRate_{TotalSC}} * \frac{N_{VMsComp}}{N_{VMsData} + N_{VMsComp} + \frac{Size_{Ev}}{720}} \quad (3)$$

Equation 3 can be used to approximate the price reduction obtained by the Copy&Compute strategy, which is proportional with the speedup over the Stream&Compute for the computation rate of the events. For sets of events up to 720 GB, 32 to 64 compute VMs, and up to 3–4 data nodes, the second part of the equation has values between 0.90 and 0.95. Hence, the price reduction is approximately 90%–95% times the compute speedup between the two methods. This approximation can be used also for the previously discussed cost scenario, in which the Stream&Compute scenario was shown to be 2.77 times more expensive. Doing so, we find that the price reduction is about 2.7 to 2.85 times, considering a speedup of 3, as per Section III-B.

V. Related Work

As cloud computing become a viable option for executing scientific applications in areas ranging from physics [7], to chemistry [29], to biology [30] and bioinformatics [18], understanding what cloud can provide becomes increasingly interesting. Multiple studies have explored various data management strategies using existing storage options for clouds [1], [2], [3], [31], [32], [33]. These general studies focus on scientific applications that process large, unstructured sets of static input data (i.e., data that is available in the cloud storage when the processing starts and remains unchanged during the computation). Consequently, the performance aspects considered in these studies focus on I/O performance with respect to VM instance types [1], data location [32], or the available storage options [2], [33]. To the best of our knowledge, however, no previous study has considered the case of *dynamic* sets of independent pieces of input data (e.g., data streamed from a large network of sensors to the cloud for processing).

A number of performance evaluations concerning data analysis in the clouds focus on the MapReduce [34] processing paradigm. Relevant observations for our work show the importance of an adequate ratio between compute nodes and data nodes, a ratio that depends on the type of the computation [25]. Additionally, the authors report significant performance penalties due to the virtualization layer. Other work related to our study was reported in [35], where the authors analyzed the streaming features of Hadoop [36] and reported the existence of an overhead due to streaming. We have made a similar observation regarding the scalability of stream processing. Since the out-of-the-box MapReduce paradigm is not appropriate for processing continuous streams of data, variations of the paradigm were proposed to support stream-based analysis [37], [38]. Unlike these studies, our performance evaluation does not focus on stream processing using MapReduce: instead, it considers stand-alone applications that process the stream data by using independent batch workers.

Despite the expected growth of BigData and on-line processing [39], most of today's analytics systems are limited to a single cluster or, more rarely, a single datacenter [36], [40], [41]. However, due to increasing data volumes to be processed, data starts to be distributed across multiple data centers (e.g. the data from CERN's ATLAS experiment [8]). Additionally, with the emergence of wide geographical-area observatories [42] new compute scenarios appear in which the data sources do not reside in the same geographical location with the compute nodes. Few systems, mostly derived from the MapReduce paradigm, tackle the problem of multisite or multicluster computation [18], [43], [44]. In the case of geographically distributed stream processing little knowledge exists about the best options for data management. The most relevant work for our study is [17], where the authors proposed their vision of a stream-processing system that could potentially reach high degrees of parallelism by means of a divide, conquer, and combine paradigm. Unlike that work, our study focuses on understanding how storage and compute options available on today's clouds can be best used to reach high performance under low costs when processing remote stream data.

VI. Conclusion

We presented and evaluated two streaming strategies in the cloud: Stream&Compute, where the data is streamed directly to the compute nodes, and Copy&Compute, where the data is first copied to the cloud and then used for computation. Our results show that, contrary to our intuition, Copy&Compute outperforms Stream&Compute by as much as three times because of the performance trade-offs present in current virtualization technology. This performance difference can have significant consequences in terms of cost, especially at higher scales.

Our evaluation highlights the impact that trade-offs in current virtualization technology can have on cloud computing scenarios. Our results also illustrate the need for hardware and virtualization technology that can provide a more controlled allocation of CPU to I/O processing.

Acknowledgment

This work was supported by Inria through the Data@Exascale Associate Team and by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This material is produced using the FutureGrid testbed supported in part by the National Science Foundation under Grant No. 0910812. Additional experiments were carried out using Windows Azure which was supported by the joint INRIA-Microsoft Research Center (Z-CloudFlow project).
We also want to thank the Nimbus team who provided helpful information about the Nimbus and FutureGrid platforms and helped us quickly resolve the issues we encountered.

References

[1] K. Yelick, S. Coghlan, B. Draney, and R. S. Canon, "The Magellan Report on Cloud Computing for Science," in *U.S. Depeartment of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR)*, 2011.

[2] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, "An evaluation of the cost and performance of scientific workflows on amazon ec2," *J. Grid Comput.*, vol. 10, no. 1, pp. 5–21, Mar. 2012.

[3] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *IEEE Communications Surveys and Tutorials*, vol. 13, no. 3, pp. 311–336, 2011.

[4] "Ocean Observatory Initiative," http://oceanobservatories.org/.

[5] A. Padmanabhan, S. Wang, G. Cao, M. Hwang, Y. Zhao, Z. Zhang, and Y. Gao, "Flumapper: an interactive cybergis environment for massive location-based social media data analysis," in *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, ser. XSEDE '13. New York, NY, USA: ACM, 2013, pp. 33:1–33:2.

[6] J. Balewski, J. Lauret, D. Olson, I. Sakrejda, D. Arkhipkin, J. Bresnahan, K. Keahey, J. Porter, J. Stevens, and M. Walker, "Offloading peak processing to virtual farm by STAR experiment at RHIC," *Journal of Physics: Conference Series*, 2012.

[7] K. R. Jackson, L. Ramakrishnan, K. J. Runge, and R. C. Thomas, "Seeking supernovae in the clouds: a performance study," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 421–429.

[8] "ATLAS," http://home.web.cern.ch/fr/about/experiments/atlas.

[9] "ATLAS Applications," https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/PhysicsAnalysisWorkBookRel16D3PDAnalysisExample.

[10] "Amazon EBS," http://aws.amazon.com/fr/ebs/.

[11] "Azure Drives," http://msdn.microsoft.com/en-us/library/windowsazure/jj156162.aspx.

[12] "Amazon S3," http://aws.amazon.com/s3/.

[13] "Amazon Glacier," http://aws.amazon.com/glacier/.

[14] "Microsoft Azure BLOBs," http://www.windowsazure.com/en-us/develop/net/how-to-guides/blob-storage/.

[15] B. e. a. Calder, "Windows azure storage: a highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 143–157. [Online]. Available: http://doi.acm.org/10.1145/2043556.2043571

[16] "Google Cloud Storage," https://cloud.google.com/products/cloud-storage.

[17] S. J. Kazemitabar, F. Banaei-Kashani, and D. McLeod, "Geostreaming in cloud," in *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on GeoStreaming*, ser. IWGS '11. New York, NY, USA: ACM, 2011, pp. 3–9. [Online]. Available: http://doi.acm.org/10.1145/2064959.2064962

[18] A. Costan, R. Tudoran, G. Antoniu, and G. Brasche, "TomusBlobs: Scalable Data-intensive Processing on Azure Clouds," *Journal of Concurrency and computation: practice and experience*, 2013.

[19] "ROOT Framework," http://root.cern.ch/drupal/.

[20] "FutureGrid," https://portal.futuregrid.org/.

[21] "Nimbus," http://nimbusproject.org/.

[22] "OpenStack," http://openstack.org/software/grizzly/.

[23] "Azure," http://www.windowsazure.com/en-us/.

[24] 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.

[25] E. Feller, L. Ramakrishnan, and C. Morin, "On the Performance and Energy Efficiency of Hadoop Deployment Models," in *The IEEE International Conference on Big Data 2013 (IEEE BigData 2013)*, Santa Clara, États-Unis, Oct. 2013, grid'5000 Grid'5000.

[26] S. Gamage, R. R. Kompella, D. Xu, and A. Kangarlou, "Protocol responsibility offloading to improve tcp throughput in virtualized environments," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, pp. 7:1–7:34, Aug. 2013. [Online]. Available: http://doi.acm.org/10.1145/2491463

[27] E. Yildirim and T. Kosar, "Network-aware end-to-end data throughput optimization," in *Proceedings of the first international workshop on Network-aware data management*, ser. NDM '11. New York, NY, USA: ACM, 2011, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/2110217.2110221

[28] G. Khanna, U. Catalyurek, T. Kurc, R. Kettimuthu, P. Sadayappan, and J. Saltz, "A dynamic scheduling approach for coordinated wide-area data transfers using gridftp," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008, pp. 1–12.

[29] J. CalA, H. Hiden, S. Woodman, and P. Watson, "Cloud computing for fast prediction of chemical activity," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1860–1869, Sep. 2013.

[30] L. Hodgkinson, J. Rosa, and E. A. Brewer, "Parallel software architecture for experimental workflows in computational biology on clouds," in *Proceedings of the 9th international conference on Parallel Processing and Applied Mathematics - Volume Part II*, ser. PPAM'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 281–291.

[31] N. Edwards, M. Watkins, M. Gates, A. Coles, E. Deliot, A. Edwards, A. Fischer, P. Goldsack, T. Hancock, D. McCabe, T. Reddin, J. Sullivan, P. Toft, and L. Wilcock, "High-speed storage nodes for the cloud," in *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing*, ser. UCC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 25–32. [Online]. Available: http://dx.doi.org/10.1109/UCC.2011.14

[32] R. Tudoran, A. Costan, and G. Antoniu, "Datasteward: Using dedicated compute nodes for scalable data management on public clouds," in *Proceedings of the 11th IEEE International Symposium on Parallel and Distributed Processing with Applications*, ser. ISPA '13. Washington, DC, USA: IEEE Computer Society, 2013.

[33] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, "Early observations on the performance of windows azure," *Sci. Program.*, vol. 19, no. 2-3, pp. 121–132, Apr. 2011.

[34] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[35] Z. Fadika, M. Govindaraju, R. Canon, and L. Ramakrishnan, "Evaluating hadoop for data-intensive scientific operations," in *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, ser. CLOUD '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 67–74.

[36] "Hadoop," http://hadoop.apache.org/.

[37] D. Alves, P. Bizarro, and P. Marques, "Flood: elastic streaming mapreduce," in *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '10. New York, NY, USA: ACM, 2010, pp. 113–114.

[38] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: fault-tolerant streaming computation at scale," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: ACM, 2013, pp. 423–438.

[39] U. Verner, A. Schuster, M. Silberstein, and A. Mendelson, "Scheduling processing of real-time data streams on heterogeneous multi-gpu systems," in *Proceedings of the 5th Annual International Systems and Storage Conference*, ser. SYSTOR '12. New York, NY, USA: ACM, 2012, pp. 8:1–8:12.

[40] Y. Simmhan, C. van Ingen, G. Subramanian, and J. Li, "Bridging the gap between desktop and the cloud for escience applications," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, ser. CLOUD '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 474–481. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2010.72

[41] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07. New York, NY, USA: ACM, 2007, pp. 59–72.

[42] A. Baptista, B. Howe, J. Freire, D. Maier, and C. T. Silva, "Scientific exploration in the era of ocean observatories," *Computing in Science and Engg.*, vol. 10, no. 3, pp. 53–58, May 2008.

[43] L. Wang, J. Tao, H. Marten, A. Streit, S. U. Khan, J. Kolodziej, and D. Chen, "Mapreduce across distributed clusters for data-intensive applications," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, ser. IPDPSW '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 2004–2011.

[44] Y. Luo and B. Plale, "Hierarchical mapreduce programming model and scheduling algorithms," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, ser. CCGRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 769–774.