2011 IEEE International Parallel & Distributed Processing Symposium

Evaluating Adaptive Compression to Mitigate the Effects of Shared I/O in Clouds

Matthias Hovestadt, Odej Kao, Andreas Kliem and Daniel Warneke

Technische Universität Berlin
Email: {firstname}.{lastname}@tu-berlin.de

***Abstract*—IaaS clouds have become a promising platform for scalable distributed systems in recent years. However, while the virtualization techniques of such clouds are key to the cloud's elasticity, they also result in a reduced and less predictable I/O performance compared to traditional HPC setups. Besides the regular performance degradation of virtualized I/O itself, it is also the potential loss of I/O bandwidth through co-located virtual machines that imposes considerable obstacles for porting data-intensive applications to that platform.**

In this paper we examine adaptive compression schemes as a means to mitigate the negative effects of shared I/O in IaaS clouds. We discuss the decision models of existing schemes and analyze their applicability in virtualized environments. Based on an evaluation using XEN, KVM, and Amazon EC2, we found that most decision metrics (like CPU utilization and I/O bandwidth) are displayed inaccurately inside virtual machines and can lead to unreasonable levels of compression. As a remedy, we present a new adaptive compression scheme for virtualized environments which solely considers the application data rate. Without requiring any calibration or training phase our adaptive compression scheme can improve the I/O throughput of virtual machines significantly as shown through experimental evaluation.

***Keywords*-Cloud Computing, Adaptive Compression, I/O Performance**

I. Introduction

In recent years, Infrastructure-as-a-Service (IaaS) clouds have gained a lot of attention as a flexible and inexpensive platform for large-scale distributed systems. Based on virtualization techniques like XEN [1] or KVM [2], customers of such cloud systems can allocate large sets of virtual machines and use them on a short-term pay-as-you-go basis. The virtual machines themselves run somewhere inside the cloud operator's data center without the customer knowing any details about the underlying physical IT infrastructure. Prominent examples of such IaaS clouds are Amazon EC2 [3] and the Rackspace Cloud [4].

While the virtualization software used in today's IaaS clouds is crucial for rapid resource provisioning, it also imposes some serious challenges compared to traditional high-performance computing (HPC) setups. One of these challenges, in particular for data-intensive applications, is the reduced and less predictable I/O performance of cloud systems.

Although recent developments in the area of virtualization have led to significant improvements in I/O performance, preliminary scientific evaluations of commercial IaaS clouds [5]–[7] indicate that the throughput in practice is still considerably lower than in unvirtualized environments. From the cloud customer's perspective, the physical environment of his virtual machine is opaque, not giving him any insight into technical details. However, the customer can usually assume one of the following reasons for the degraded I/O performance:

On the one hand, virtualized I/O is known to cause CPU overhead [8], [9]. In scenarios with high I/O load, it therefore may be the CPU resources allocated to the virtual machine which limit the data throughput. On the other hand, several virtual machines may be co-located on the same physical host and in fact share the I/O resources of the host system. As a result, the I/O workload induced by one virtual machine can negatively affect the I/O performance of a co-located virtual machine and lead to unpredictable performance fluctuations.

A variety of projects is currently working to improve the performance and fairness of shared virtualized I/O (e.g. [10]–[12]. However, since these proposals require modifications to either the operating system kernel or the hypervisor, users of commercial clouds cannot benefit from those until their cloud providers consider them mature enough to be adopted.

For this reason this paper presents an infrastructure agnostic approach to mitigate the effects of shared I/O in clouds which can be applied by the cloud customers without assistance of the cloud providers, namely *adaptive online compression*.

The idea of adaptive online compression is to improve the I/O throughput by continuously choosing between different compression levels and applying them dynamically to the outgoing data stream. The compression level is selected by a decision model which constantly estimates the performance gain based on system metrics like the current CPU load, available I/O bandwidth, or the compressibility of the data.

Although several adaptive online compression schemes have been introduced in recent years ([13]–[16]), it is unclear if they can be applied in virtualized environments for the following two reasons:

First, most of the existing adaptive compression schemes require a training phase in order to calibrate their decision model. During that phase an unloaded system with stable I/O characteristics is assumed. In a cloud, where information on the physical IT infrastructure and co-located virtual machines is not available, this assumption does not necessarily hold. Second, the decision models of existing adaptive compression schemes rely on the displayed system metrics of the operating system, like the current CPU utilization or available I/O bandwidth. However, the accuracy of these systems metrics in virtualized environments has not been studied so far.

This paper addresses the applicability of adaptive compression schemes in virtualized environments like IaaS clouds. In particular, it has two main contributions:

- We provide an analysis of the accuracy of the displayed system metrics which are the basis for the decision models of existing adaptive compression schemes using different types of virtualization technologies and clouds. Our analysis covers the popular open source virtualization techniques XEN and KVM as well as some baseline tests on Amazon EC2. We found that even in situations with no co-located virtual machines the CPU utilization displayed inside the virtual machine is highly inaccurate under high I/O load throughout all studied virtualized techniques. Moreover, caching effects can seriously hamper estimations of the available I/O bandwidth.
- As a result of the system metrics' poor accuracy we propose a new decision model for adaptive compression schemes in clouds. Unlike existing decision models, our model does not require a training phase. In addition, our mechanism takes the achievable application data rate, i.e. the data rate experienced by the application before compressing the data, as foundation for the decision process. Thus, the decision on compression levels solely relies on directly measurable factors instead of system metrics provided by a (virtualized) operating system.

The rest of this paper is structured as follows: In Section II we present the initial analysis of the accuracy of system metrics. Motivated by our findings, we describe our new decision model for adaptive compression in Section III including pseudo code as well as aspects on the implementation in our data processing framework Nephele [17]. In Section IV we analyze the performance of our adaptive compression scheme for TCP streams using different load scenarios. Section V discusses related work and is followed by Section VI which concludes the paper.

II. Accuracy of Decision Metrics

Most adaptive compression schemes (e.g. [13], [15], [16]) consider the available system resources when deciding on the compression level to apply on the outgoing data. Therefore, these systems have to rely on the accuracy of the system metrics like the CPU utilization and I/O bandwidth provided by the operating system. The obtained data is fed into the respective scheme's decision model to continuously assess the potential gain of compression under the current circumstances.

While this approach seems reasonable for physical, fully-controlled computers, it is unclear whether the accuracy of these system metrics, which are displayed inside virtualized machines, is high enough to provide a meaningful basis of decision-making.

To clarify this initial question, we conducted a series of small I/O experiments on our local Eucalyptus-based cloud [18] and measured the accuracy of the CPU utilization and I/O throughput based on the popular open source virtualization techniques XEN [1] and KVM [2]. A detailed description of the setup is provided in the appendix.

A. Accuracy of CPU Utilization

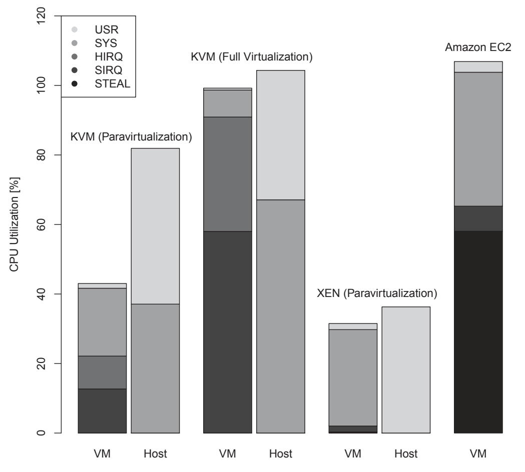
Our first experiments aimed at determining the accuracy of the CPU utilization as displayed inside the virtual machines during I/O intensive operations. We created a set of small auxiliary programs to generate network and file I/O load. Then we contrasted the displayed CPU utilization inside the virtual machine with the actual CPU utilization as reported by the host system while the respective programs were running.

In order to monitor the CPU utilization inside the virtual machines we continuously queried the Linux system interface `/proc/stat` at an interval of one second. On the host system our monitoring scheme was dependent on the virtualization layer we used for the respective experiment. For KVM-based experiments we first determined the process ID of the corresponding `qemu` process, afterwards traced the CPU utilization for that process using the `/proc/<process ID>/stat` interface, again at a sample interval of one second. For XEN-based experiments we used the management tool `xentop` to observe the CPU utilization that was accounted to the monitored `domU` from the perspective of the `dom0`.

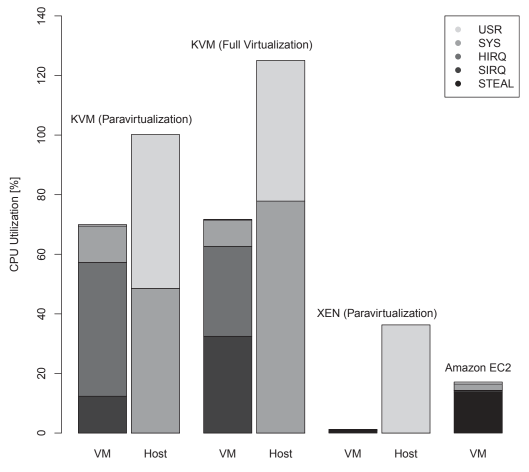
Figure 1 illustrates the results of our tests pertaining to the accuracy of the CPU utilization. Each of the four plots shows the average CPU utilization during one particular type of I/O operation (network send and receive, file write and read) as reported by the operating system of the virtual machine and the host system. The average has been calculated from at least 120 individual samples and is split into the fraction of time the CPU spent processing into user (USR) or kernel mode (SYS), serving hardware (HIRQ) or software interrupts (SIRQ). In case of XEN-based virtualization, STEAL denotes the amount of CPU time that the hypervisor has allocated to tasks other than the observed virtual machine.

For all four types of I/O operations we also analyzed the accuracy of the displayed CPU utilization using different kinds of virtualization techniques. KVM (paravirtualization) refers to experiments with KVM-based virtual machines which used the `virtio` device drivers inside the virtual machines. In contrast to that, KVM (full virtualization) marks experiments with KVM-based virtual machines and unmodified device drivers. For comparison, we also added the CPU utilization as displayed by virtual machines on Amazon EC2. For these experiments, however, we were unable to observe the CPU utilization as reported by the host system.

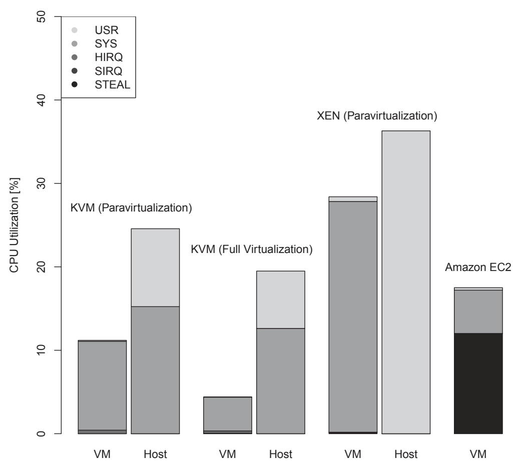
During all the experiments involving network transfer we used a TCP connection and made sure that the opposite part of the connection was an unvirtualized machine which was at least as fast as the observed virtual machine. Hence, any potential performance bottleneck during these experiments was either induced by the network or the virtual machine itself. For all experiments including file I/O we used raw I/O API



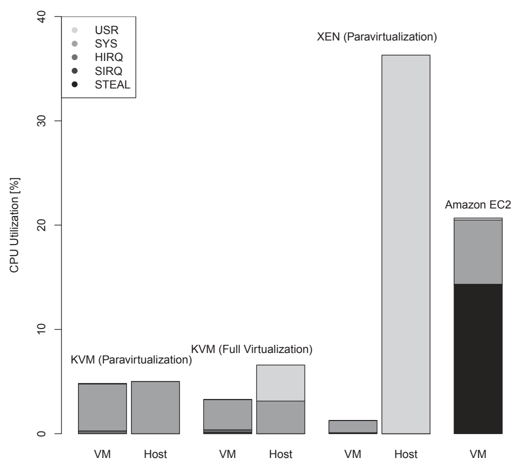
(a) Network I/O (send operation)



(b) Network I/O (receive operation)



(c) File I/O (write operation)



(d) File I/O (read operation)

Fig. 1: Accuracy of displayed CPU utilization inside virtual machines during I/O intensive operations.

to avoid caching effects inside the virtual machine as far as possible.

In sum, our experiments revealed a large discrepancy between the CPU utilization displayed inside the virtual machine and the one reported on the host system. More importantly, this discrepancy is not specific to a particular type of I/O operation or virtualization technique. It can be found across all considered I/O operations and virtualization techniques. While for some I/O operations the discrepancy in the reported CPU utilization is rather small (e.g. network send operation using KVM (full virt.) or XEN), for others (e.g. network send operation using KVM (paravirt.) or file read operation using XEN) the gap can grow up to a factor of 15.

B. Accuracy of I/O Bandwidth

Our second series of experiments was targeted at the accuracy of the I/O bandwidth as reported by the virtual machine, the second major performance characteristic considered by most existing adaptive compression schemes. Therefore, we modified our set of auxiliary programs to record timestamps after every 20 MB of generated or consumed I/O data, respectively. With the help of these timestamps we then calculated the I/O data rate as it appeared from within the virtual machine. In total, each auxiliary program either produced or consumed 50 GB of data.

Figure 2 illustrates the distribution of the reported application layer throughput in the course of our network experiments from the perspective of the sender's virtual machine. Again, we display the results for the different types of virtualization environments. As a baseline, we also conducted the experiments on the native host systems.

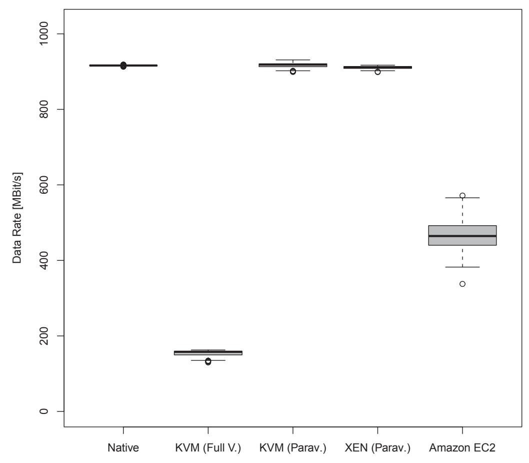


Fig. 2: Distribution of network I/O throughput as observed within the sending virtual machine.

For all experiments conducted on our local Eucalyptus-based cloud, the fluctuations of network throughput only increased marginally compared to those we measured on the native host system. On Amazon EC2, however, we experienced heavy throughput variations in the range of tens or even hundreds of MBit/s. This confirms the findings of previous evaluations [6], which report that the TCP/UDP throughput on Amazon EC2 can fluctuate rapidly between 1 GBit/s and zero, even at a time scale of tens of milliseconds.

Figure 3 depicts the throughput variance we experienced for writing data to the virtual machine's disk. The measurements conducted on KVM-based virtual machines (both full and paravirt.) as well as on Amazon EC2 showed a throughput fluctuation which is comparable to the one of our native, unvirtualized baseline system. However, with the XEN-based virtual machines, which we instantiated on our local cloud, we witnessed significant caching effects. Due to these caching effects the data rate inside the virtual machine occasionally appeared to be exceedingly high. In fact, the data was only buffered inside the host system's main memory. Periodically, when the host system decided to actually flush the buffered data to disk, the data rate displayed inside the virtual machine dropped to a few MB/s. As a result of these caching effects, the average data throughput for the XEN-based experiments also spuriously appears to be higher compared to the experiments on the native or KVM-based system in the plot. However, after having written the 50 GB to the virtual machine's disk, large portions of the data had not actually been written to the physical hard drive, but still remained inside the host system's main memory.

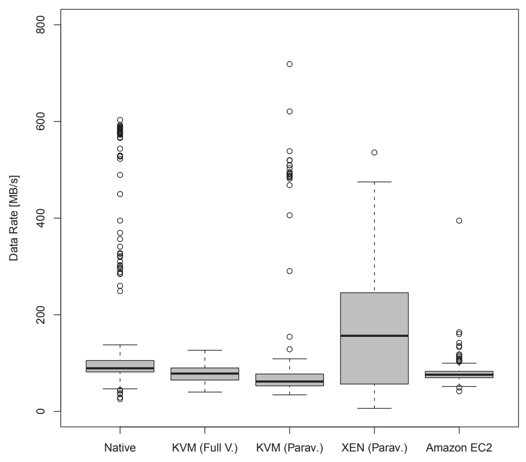


Fig. 3: Distribution of file I/O throughput (write) as observed within the virtual machine.

C. Discussion

As a result of our initial experiments, we can conclude that the virtualization layer used in today's IaaS clouds can have a serious impact on the accuracy of the system metrics that are displayed inside a virtual machine. In nearly all of our experiments the displayed CPU utilization did not accurately reflect the CPU overhead that was caused by the I/O operations. Therefore, the virtual machine often considered the CPU to be (nearly) idle when in fact it consumed a significant amount of processing time according to the host system. Moreover, we found that virtualization can increase the variance in data throughput for both network and file I/O. The variations for file I/O can be so severe that data streams of several GB must be observed before a meaningful mean throughput can be calculated.

This inaccuracy of both types of system metrics (CPU utilization and I/O throughput) is a shortcoming for all kinds of decision algorithms that have to trade off CPU against I/O performance. With regard to adaptive data compression the inaccurate display of CPU utilization inside the virtual machine can lead to false estimations about the compression times on the one hand. On the other hand, the fluctuating I/O throughput makes it very hard to provide accurate bandwidth predictions on a short-term basis.

Moreover, the accuracy of the system metrics depends heavily on the virtualization technique (XEN vs. KVM), the type of virtualization (full vs. paravirt.) as well as underlying IT infrastructure. E.g., our throughput experiments on Amazon EC2 showed completely different results than the ones we conducted on our local Eucalyptus-based cloud although both systems had a comparable kernel installed.

III. Adaptive Compression Algorithm

Based on our findings from the previous section we devised a new decision model for adaptive data compression in virtualized cloud environments. In order to reflect the particular characteristics of such environments, the new decision model has been built along to the following design goals:

- **No training phase:** The decision model must not require any offline calibration or training phase. An offline training phase must typically be performed in a verifiably unloaded system. In a commercial IaaS cloud, however, a virtual machine may spuriously appear to be idle although the host system is heavily stressed by co-located virtual machines. Moreover, given that the performance characteristics of two virtual machines of the same type can differ significantly in some cloud systems [7], the training phase must potentially be re-executed after the instantiation of each machine. In sum, these training periods can account for considerable amounts of processing time and, thus, also increase the processing cost.
- **No decision based on CPU resources:** As the display of available CPU resources under high I/O load in virtual machines is likely to be skewed, the new decision model must not rely on it.
- **Embrace throughput fluctuations:** Unlike existing compression schemes, which try to adapt the compression level to the outgoing data stream on the granularity of KB, our decision model shall focus on a granularity level of MB in order to allow for the possible throughput fluctuations we highlighted in Section II.

As a result of these design goals, we present a new decision model which dynamically adapts the compression level as a response to changes in the application data rate, i.e. the data rate that is experienced by the application before compressing the data. Although the application data rate at a particular compression level also involves aspects like CPU utilization, available I/O bandwidth or the compressibility of the data itself, it is only indirectly influenced by those. Therefore, our approach does not have to rely on the possibly inaccurate displays of those metrics inside the virtual machine.

In the following we will explain our decision model and highlight its implementation as part of our data processing framework Nephele [17].

A. Decision Algorithm

Similar to existing approaches we assume our adaptive compression module to be placed between the application and the respective I/O layer. Instead of passing the data right to the I/O layer it is first intercepted by the adaptive compression module which, if considered beneficial, compresses the data according to a specific compression level. As already demonstrated by existing schemes (e.g. [16]), the entire adaptive compression/decompression logic can be encapsulated in a higher-level communication library and therefore becomes completely transparent to the application.

Following the idea of previous publications (e.g. [14]–[16]), we also assume that our adaptive compression algorithm can choose between a fixed set of n compression levels. Each compression level thereby refers to a specific compression algorithm which is applied at the respective level. The individual compression levels must be ordered by their respective time/compression ratio. Compression level 0 stands for no compression. A variety of compression algorithms also offers parameters to influence their time/compression ratio. Therefore, it is conceivable to use the same compression algorithm at multiple levels but with different parameters.

Our adaptive compression scheme reconsiders the decision which compression level is to be applied every t seconds. Based on the amount of application data which has been received from the application, (possibly) compressed, and passed to the I/O layer during that time span, we calculate the application data rate for these last t seconds. In case of network I/O the application data rate also includes the decompression time at the receiver because of the network's flow control mechanisms. The concrete decision algorithm to determine the compression level for the next t seconds is shown in Algorithm 1. The algorithm uses a series of auxiliary variables which are explained in Table I.

Variable	Meaning
ccl	The compression level that is currently applied to the outgoing data stream. Initially, the variable is set to 0 (no compression).
ncl	The next compression level that shall be applied to the data based on the algorithm's decision.
c	A simple counter variable which stores how often the decision algorithm has been called since the last change of compression level. The counter is initialized with 0.
inc	A Boolean variable which indicates if the compression has been increased at its previous change. Initially, the variable is set to $TRUE$.
bck	An array which stores the backoff values for the individual compression levels. Initially, all fields inside the array are set to 0.
cdr	The average application data rate which has been determined for the last t seconds using the compression level ccl.
pdr	The average application data rate which has been determined for the t seconds before the last t seconds. On the first call of the decision algorithm, pdr is set to cdr.

TABLE I: Explanation of the decision algorithm's variables.

As already mentioned, our decision model adapts the compression level in response to changes in the observed application data rate. This is also reflected in the structure of our algorithm which distinguishes three major cases:

In the first case (lines 4-14) the application data rate during the last t seconds cdr (compressed with compression level ccl) does not differ from the data rate of the previous t second time span pdr. In order to cope with fluctuations in the data rate we introduce the parameter α. The parameter α defines in what range cdr may differ from pdr before our algorithm actually responds to the change. Small values of α allow our algorithm to detect the best compression level even if the performance gains between the respective compression algorithms are rather small. However, they also make the decision algorithm more prone to incorrect decisions because variations in the application data rate can also result from variations in the throughput of the underlying I/O system (e.g. the TCP connection). During our experiments we found 0.2 to be a reasonable value for α.

Since our decision model cannot rely on any previous knowledge from an offline training phase, it optimistically switches to the next higher or lower compression level occasionally to see how the application data rate is affected. However, a fundamental aspect of our algorithm is that these switches occur less often for compression levels which have continuously led to improvements in the data rate. We achieve this behavior through an exponential backoff scheme (line 6). The decision to increase or decrease the compression level as part of such an optimistic switch depends on the variable inc. The variable inc indicates if the last change of compression level has been an increase or a decrease. Note that inc is usually updated outside of the displayed algorithm depending on the input parameter ccl and the return value ncl.

Algorithm 1 GetNextCompressionLevel(cdr, pdr, ccl)

```
1:  $d \leftarrow (cdr - pdr)$
2:  $c \leftarrow c + 1$
3:  $ncl \leftarrow ccl$
4:  if $|d| \leq \alpha \times pdr$ then
5:      {No change in application data rate}
6:      if $c \geq 2^{bck[ccl]}$ then
7:          {Backoff over, try another compression level}
8:          if $inc = TRUE$ then
9:              $ncl \leftarrow ncl + 1$
10:         else
11:             $ncl \leftarrow ncl - 1$
12:         end if
13:         $c \leftarrow 0$
14:     end if
15: else if $d > 0$ then
16:     {Application data rate has improved}
17:     $bck[ccl] \leftarrow bck[ccl] + 1$
18:     $c \leftarrow 0$
19: else
20:     {Application data rate has decreased}
21:     $bck[ccl] \leftarrow 0$
22:     if $inc = TRUE$ then
23:         $ncl \leftarrow ncl - 1$
24:     else
25:         $ncl \leftarrow ncl + 1$
26:     end if
27:     $c \leftarrow 0$
28: end if
29: return $ncl$
```

The second major case our algorithm has to handle is an improvement of the application data rate (lines 15-18). In this case our algorithm increments the backoff value of the current compression level $bck[ccl]$ by one. Thus, the algorithm will less often try out other compression levels from the current compression level given that no change in the data rate occurs.

The third and final case addresses a degradation of the application data rate (lines 19-27). In this case the algorithm reverts the last compression level change (lines 22-26). Moreover, it sets the backoff value for the compression level with which it experienced the degradation ($bck[ccl]$) back to 0. Hence, optimistic switches to other compression levels again become more frequent for that compression level in the future.

Although our algorithm can make wrong decisions with respect to the chosen compression level, it can always react to degradations of the application data rate immediately (i.e. after t seconds) and revert the wrong decision. Good decisions are rewarded with increased backoff values. This ensures that any unnecessary probing of other compression levels decreases exponentially over time.

B. Implementation

We integrated the new adaptive compression scheme into Nephele, our framework for massively parallel data processing [17]. Nephele executes data flow programs which are expressed as directed acyclic graphs (DAGs) on large sets of shared-nothing servers, e.g. IaaS clouds. Thereby each vertex of the DAG represents a task of the overall processing job. Tasks can exchange data through communication channels which are modeled as the edges of the job DAG.

Currently, Nephele supports three different types of communication channels: file, TCP network, and in-memory channels. For our initial prototype we integrated our adaptive compression scheme into Nephele's file and network channels. The implementation is completely transparent to the tasks, so there is no modification required to their program code.

Our implementation features four different compression levels. As in the description of the decision algorithm, compression level 0 again represents no data compression. Given the comparably high bandwidth of the available I/O interfaces, we have chosen the compression algorithms for the remaining three levels with regard to their compression speed. At compression level 1 (LIGHT) we use the QuickLZ compression library [19] which is highlighted by its fast compression speed. QuickLZ is also used for compression level 2 (MEDIUM), but with a setting which favors a better compressed size over compression speed. For compression level 2 (HEAVY) we use the compression library LZMA [20]. Although LZMA is known to be significantly slower than QuickLZ, it generally offers a better compression ratio which might pay off if the available I/O bandwidth is low enough.

For performance reasons Nephele internally buffers data that is written to its file or network channel in memory blocks of at most 128 KB size before passing it the respective I/O layer. Each of these blocks is passed independently to the library assigned to the currently chosen compression library. This means each block contains all the information to be decompressed by the receiver, including meta information about compression algorithm and the compression dictionary.

IV. Evaluation

After having motivated and described our new adaptive compression scheme, we now want to evaluate its performance through a series of experiments. All of these experiments were conducted on our local Eucalyptus-based cloud using KVM-based virtual machines with paravirtualized I/O devices. The concrete setup of the virtual machines and the host systems corresponds to the one described in the appendix.

As a result of the tremendous caching effects for file I/O we observed in Section II, we focus on network I/O only.

A. Adaptivity

Our first series of experiments aims at demonstrating the ability of our adaptive compression approach to determine a suitable compression level for a given type of data and I/O bandwidth. We created a simple Nephele job which consists of two tasks (sender and receiver task) connected by a TCP network channel. The sender and receiver task were concurrently executed on two distinct virtual machine. Each virtual machine ran on a separate host system.

In order to evaluate the impact of different compressibilities on our approach, we conducted our experiments with three distinct files. The first two files were chosen from the Canterbury Corpus [21], a well-known compression benchmark. As a file with a high compressibility (HIGH) we chose the file `ptt5` from the benchmark which common compression libraries can compress down to 10-15% of its original size. As a representative for a file with moderate compressability (MODERATE) we chose the file `alice29.txt` from the corpus. Its compression ratio is about 30-50% depending on the algorithm used. Since the Canterbury Corpus does not offer files with a notably poor compressibility, we chose a standard JPG image of about 250 KB (refered to as `image.jpg` or LOW) as the third file for our experiments. Its compression ratio ranged between 90-95%.

In some of the experiments we co-located additional virtual machines on the same physical hosts in order to realistically assess the effects of shared I/O on our adaptive compression scheme. Each co-located virtual machine on the sender's host system thereby established a separate TCP connection to another virtual machine co-located on the receiver's host system and transmitted data as fast as possible.

In all the experiments the sender task repeatedly wrote the respective test files (either `ptt5`, `alice29.txt`, or `image.jpg`) to the network channel until a total data volume of 50 GB was generated and consumed by the receiver. During all the experiments t was set to 2 seconds and α to 0.2.

Table II summarizes the results of our experiments. The table shows the average completion times of the sample job for the different data compressibilities and the number of concurrent TCP connections which have been established by the co-located virtual machines. The numbers in brackets denote the standard deviation. For comparison, the table also includes the average completion times when the compression level was chosen statically before the execution and was not determined by our adaptive compression scheme at runtime. The numbers written in bold type mark the fastest execution.

As indicated in Table II, the compression levels chosen by our adaptive compression scheme (DYNAMIC) led to average completion times which were at most 22% worse than the fastest average completion times with statically set compression levels. In many cases the completion times achieved with our adaptive scheme were between the first and second fastest completion times with statically set compression levels. Given that our algorithm has to perform some initial probing to determine the best compression levels, the results in these cases can be considered ideal.

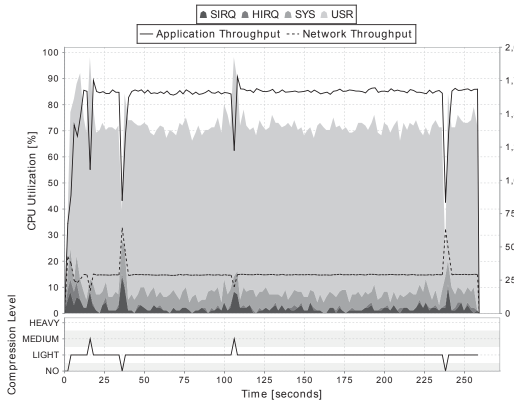


Fig. 4: Performance of our adaptive compression scheme with highly compressible data (HIGH) and no background traffic.

Figure 4 depicts the compression decisions our adaptive scheme made in such an ideal case, i.e. the highly compressible `ptt5` file (HIGH) with no concurrent TCP connections. The figure shows the sender's CPU utilization, application throughput, network throughput as well as the chosen compression levels over time. Due to the large differences in the compression/time ratios of the respective compression libraries, the decision algorithm can quickly determine the compression level LIGHT (QuickLZ, best compression speed) to result in the best overall application data rate. The figure also illustrates how the backoff mechanism we integrated in our decision algorithm reduces optimistic switches to other compression levels exponentially.

In case the performance differences between the respective compression levels are less distinctive, our decision algorithm may spuriously consider changes in the application data rate as fluctuations and continue the probing process. Figure 5 illustrates such a case for the experiment with the poorly compressible `image.jpg` file (LOW) and two concurrent TCP connections. Lowering the value of α can help to counteract this behavior, however, it also increases the risk of wrong compression decisions due to regular fluctuations in the available TCP throughput.

B. Changing Data Compressibility

In the second experiment we evaluate how our adaptive compression scheme responds to severe changes in the data compressibility. Therefore, we reused the sample job from the previous adaptivity experiments and switched between the highly compressible file `ptt5` (HIGH) and the already compressed image file `image.jpg` (LOW) every 10 GB. Again,

Compression Level	No concurrent TCP connection			One concurrent TCP connection		
	HIGH Mean (SD)	MODERATE Mean (SD)	LOW Mean (SD)	HIGH Mean (SD)	MODERATE Mean (SD)	LOW Mean (SD)
NO	569 (3)	**567** (7)	**566** (3)	908 (6)	896 (6)	**903** (6)
LIGHT	**252** (3)	629 (2)	688 (3)	**258** (3)	**624** (7)	927 (8)
MEDIUM	347 (6)	795 (5)	1095 (8)	367 (3)	840 (5)	1241 (42)
HEAVY	1881 (23)	5760 (25)	9011 (30)	1974 (24)	5979 (34)	9326 (30)
DYNAMIC	265 (4)	635 (4)	602 (3)	273 (3)	648 (16)	920 (13)

Compression Level	Two concurrent TCP connections			Three concurrent TCP connections		
	HIGH Mean (SD)	MODERATE Mean (SD)	LOW Mean (SD)	HIGH Mean (SD)	MODERATE Mean (SD)	LOW Mean (SD)
NO	1393 (75)	1292 (67)	**1313** (39)	1642 (70)	1584 (120)	1638 (70)
LIGHT	**312** (14)	**756** (23)	1440 (87)	**358** (10)	1027 (65)	**1555** (17)
MEDIUM	378 (10)	896 (38)	1481 (27)	397 (3)	**953** (55)	1829 (100)
HEAVY	1985 (26)	6130 (31)	9597 (45)	1994 (21)	6218 (34)	9278 (49)
DYNAMIC	363 (22)	920 (18)	1452 (40)	411 (35)	1075 (37)	1865 (114)

TABLE II: Average completion times of the sample job using different statically chosen compression levels (NO, LIGHT, MEDIUM, HEAVY) as well as our adaptive approach (DYNAMIC). The completion times are subdivided by the compressibility of the data (HIGH, MODERATE, LOW) and the number of concurrent TCP connections.

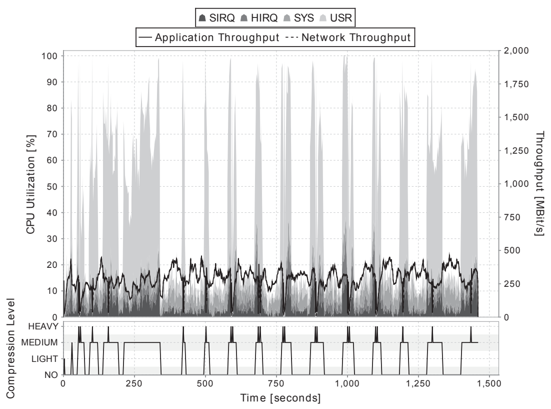


Fig. 5: Performance of our adaptive compression scheme with hardly compressible data (LOW) and two concurrent TCP connections.

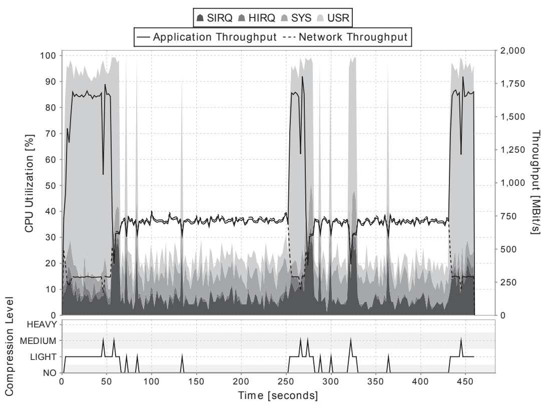


Fig. 6: Responsiveness to changes in data compressibility.

50 GB of data were generated in total for this experiment. During the experiment, no background traffic was present.

The results of the experiment are depicted in Figure 6. Apart from some minor shortcomings our decision algorithm detected the changes in the data compressibility correctly and switched the compression level accordingly. Large backoff values for compression level 0 (no compression), which arose during the transmission of `image.jpg`, can lead to relatively late optimistic switches to a higher compression level. The reason for this behavior is that without compression the application data rate is not affected by the compressibility of the data. However, the opposite case is detected immediately by our algorithm.

V. Related Work

The performance characteristics of cloud systems or virtualized systems in general have recently gained a lot of attention.

In [22] Menon et al. analyze XEN's performance overhead for network I/O virtualization through their Xenoprof toolkit. Similar to our work, the authors motivate their toolkit with the performance anomalies they experienced during their work with virtual machines. However, instead of mitigating the effects on a user level, the paper focuses on a lower level evaluation and proposes several modifications for future optimization efforts within the virtualization layer itself.

Cherkasova and Gardner [8] propose a monitoring framework for XEN and provide a performance study to quantify the CPU overhead for I/O intensive workloads. In their main evaluation the authors focus on analyzing the increase of CPU utilization depending on different levels of network traffic. The accuracy of the CPU utilization as observed inside the virtual machines is not part of their work.

Wood et al. [23] present a regression-based approach to estimate the additional CPU load an application will cause in a virtualized environment in comparison to a native one.

In order to calibrate their estimation model, the authors rely on a set of microbenchmarks which must be executed on the virtual machine as well as on the host system. Thus, their approach requires access to the host system which is typically not granted in commercial IaaS clouds.

In [24] Tickoo et al. also address the problem of modeling virtual machine performance. Their paper puts much emphasis on highlighting performance degradations that result from shared CPU caches or memory bandwidth. Based on a virtualization benchmark the authors describe a performance modeling approach to capture such effects.

Recently, the performance characteristics of Amazon EC2 have gained the interest of various research groups:

Walker [5] evaluated Amazon EC2 as a platform for high-performance scientific computing. He demonstrated that the execution speed of classic MPI applications can be significantly slower on Amazon's virtual machines compared to native cluster setups with comparable hardware characteristics.

Wang et al. [6] analyzed the impact of virtualization on EC2's network performance. Based on a set of benchmarks the authors conclude that network performance in Amazon's cloud is subject to drastic fluctuations. In particular, they highlight that TCP/UDP throughput can vary between 1 GBit/s and zero at a time granularity of tens of milliseconds. This effect is accounted to CPU sharing among several virtual machines.

Schad et al. [7] studied Amazon EC2 with respect to the variances in CPU, I/O and network performance. For all three of these resources the authors observed considerable performance fluctuations across different runs of their microbenchmarks. Their analysis revealed that virtual machines of the same type (with the same described hardware characteristics) may be hosted on different generations of host systems. Since the end user in general cannot influence the target host system for his virtual machines, the authors conclude that performance predictability on Amazon EC2 is currently hard to achieve.

In the area of adaptive compression schemes, several approaches already exist:

Motgi and Mukherjee [13] introduced their network conscious text compression system (NCTCSys) in order to reduce the transmission times of HTML streams generated by a web server, e-mail text messages or large text files transmitted by FTP. Similar to our work, NCTCSys is capable of switching between different compression algorithms. The compression algorithm is chosen by evaluating a set of parameters (e.g. network bandwidth, server load, number of clients connected), which are gained from sensor modules.

Krintz and Sucu [16] presented a more general approach applicable to various kinds of input data. Their decision model includes CPU utilization and network bandwidth as well as data obtained from an offline training phase. By exploiting a linear relationship between compression ratios of different compression algorithms, the decision model can quickly compare the estimated performance of the different compression algorithms used without testing them online. Avoiding the usage of unsuitable compression levels seems to be a great benefit, however, it is difficult to guarantee the accuracy of parameters gained from offline training as mentioned in Section III. In order to measure the benefit gained by compressing a data packet with a specific compression level, a key challenge is to make accurate predictions of the available bandwidth to forecast the transmission time needed. That is why the Network Weather Service [25] is used. The NWS is a distributed system consisting of several modules to monitor and forecast network and resource performance.

Wiseman et al. [15] also implemented an adaptive compression system for various kinds of data. They chose the best fitting algorithm by observing each algorithm's compression time as well as the speed with which compressed blocks are accepted by the receiver. A disadvantage of their decision method is the usage of several hard-coded parameters, which need a short sampling phase with unloaded I/O and CPU to fit input data different from the ones used in their evaluation.

A system which does not rely on measurements of resource performance was presented in [14]. Its main idea is to split the process of sending a data package into a compression thread, a sending thread, and a FIFO queue in the middle. The decision to raise or lower the compression level depends on the size of the FIFO queue. If the size is decreasing (resp. increasing) the compression level is lowered (resp. raised). A disadvantage of this system is the assumption that a higher compression level will lead to higher compression ratio, which is not always true, e.g., when the data is not compressible. Moreover, the system does not consider that using higher compression levels increases the compression time.

VI. Conclusion

In this paper we evaluated adaptive online compression as a means to mitigate the effects of shared I/O in IaaS clouds. Motivated by the decision models of existing adaptive compression schemes, we first analyzed the accuracy of the system metrics like CPU utilization or I/O bandwidth as displayed inside virtual machines of various types. Since we found these metrics to be potentially inaccurate, we proposed a new adaptive compression scheme which only relies on the application data rate and does not require any calibration or training phase. In extensive network experiments our new adaptive scheme yielded job completion times which were at most 22% worse than the fastest completion times with statically set compression levels and improved the overall application throughput up to a factor of 4.

In general, we think adaptive online compression is a valuable building block to improve the efficiency of distributed applications which rely heavily on the network, especially since no assistance of the cloud provider is required to apply it. For file I/O we found the aggressive caching mechanisms of some virtualization technologies to be a major obstacle which we intend to address for future work.

Appendix

All the experiments presented in this paper have been conducted on our local IaaS cloud. Each physical compute node of the cloud had the following configuration:

CPU	Two Intel Xeon 2.66 GHz CPUs (model E5430)
RAM	32 GB
Hard Disk	Seagate Barracuda ES.2 SATA 500 GB (formatted with ext3 file system)
Network	Intel Corporation 80003ES2LAN 1 GB/s Ethernet (connected to local 1 GB/s switch)
OS	Gentoo Linux
Kernel	2.6.34-xen-r4 for XEN-based experiments, 2.6.32-gentoo-r7 otherwise

In order to provision both XEN and KVM-based virtual machines on the physical nodes we used the software Eucalyptus [18] (version 1.6). The virtual machines used during all our experiments had the following characteristics:

CPU	1 CPU core
RAM	2 GB
Hard Disk	60 GB, device driver `scsi` for KVM (full virt.), `virtio_blk` for KVM (paravirt.), and `xenblk` for XEN, formatted with ext2 file system
Network	Bridged network, device driver `e1000` for KVM (full virt.), `virtio_net` for KVM (paravirt.), and `xennet` for XEN
OS	Ubuntu Linux 9.10 (Karmic Koala)
Kernel	2.6.31-22-server for XEN-based virtual machines, 2.6.32-gentoo-r7 otherwise

In order to avoid any side effect during our experiments, we only ran one virtual machine per physical host and shut down all unnecessary background services.

To put the experimental results which we gained on our local Eucalyptus-based cloud into perspective with a commercial cloud system, we also conducted some baseline tests on Amazon EC2. For these tests we instantiated two virtual machines of type "m1.small" in Amazon's US East (Virginia) data center. Both virtual machines were created inside the same availability zone and same security group. To reduce the probability of obtaining two virtual machines which are co-located on the same physical host, we destroyed and reinstantiated the virtual machines several times between different runs of our experiments. For all our experiments on Amazon EC2 we used the virtual machine image "Basic 32-bit Amazon Linux AMI 1.0" with the identifier "ami-08728661". The virtual machines reported a Linux kernel of version 2.6.34.7-56.40.amzn1.i686. For all experiments which involved file I/O we used the ephemeral storage partition of the virtual machines with an ext2 file system.

REFERENCES

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[2] Avi Kivity. kvm: the Linux virtual machine monitor. In *OLS '07: The 2007 Ottawa Linux Symposium*, pages 225–230, July 2007.

[3] Amazon Web Services LLC. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/, 2010.

[4] Rackspace US, Inc. The Rackspace Cloud. http://www.rackspacecloud.com/, 2010.

[5] Edward Walker. Benchmarking Amazon EC2 for high-performance scientific computing. *USENIX; login: magazine*, 33(5):18–23, October 2008.

[6] Guohui Wang and T.S. Eugene Ng. The impact of virtualization on network performance of Amazon EC2 data center. In *INFOCOM, 2010 Proceedings IEEE*, pages 1 –9, mar. 2010.

[7] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *PVLDB*, 3(1):460–471, 2010.

[8] Ludmila Cherkasova and Rob Gardner. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 24–24, Berkeley, CA, USA, 2005. USENIX Association.

[9] Jose Renato Santos, Yoshio Turner, G. Janakiraman, and Ian Pratt. Bridging the gap between software and hardware techniques for I/O virtualization. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.

[10] Seetharami R. Seelam and Patricia J. Teller. Virtual I/O scheduler: a scheduler of schedulers for performance virtualization. In *Proceedings of the 3rd international conference on Virtual execution environments*, VEE '07, pages 105–115, New York, NY, USA, 2007. ACM.

[11] Hiroshi Yamada and Kenji Kono. Foxytechnique: tricking operating system policies with a virtual machine monitor. In *Proceedings of the 3rd international conference on Virtual execution environments*, VEE '07, pages 55–64, New York, NY, USA, 2007. ACM.

[12] Mukil Kesavan, Ada Gavrilovska, and Karsten Schwan. Differential virtual time (dvt): rethinking I/O service differentiation for virtual machines. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 27–38, New York, NY, USA, 2010. ACM.

[13] Nitin Motgi and Amar Mukherjee. Network conscious text compression system (NCTCSys). In *ITCC '01: Proceedings of the International Conference on Information Technology: Coding and Computing*, page 440, Washington, DC, USA, 2001. IEEE Computer Society.

[14] Emmanuel Jeannot, Björn Knutsson, and Mats Björkman. Adaptive online data compression. In *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pages 379 – 388, 2002.

[15] yair Wiseman, , Karsten Schwan, and Patrick Widener. Efficient end to end data exchange using configurable compression. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 228 – 235, 2004.

[16] Chandra Krintz and Sezgin Sucu. Adaptive on-the-fly compression. *IEEE Trans. Parallel Distrib. Syst.*, 17(1):15–24, 2006.

[17] Daniel Warneke and Odej Kao. Nephele: Efficient parallel data processing in the cloud. In *MTAGS '09: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, pages 1–10, New York, NY, USA, 2009. ACM.

[18] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus open-source cloud-computing system. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.

[19] QuickLZ. Fast compression library for C, C# and Java. http://www.quicklz.com/.

[20] Igor Pavlow. LZMA SDK (software Development Kit). http://www.7-zip.org/sdk.html.

[21] Ross Arnold and Tim Bell. A corpus for the evaluation of lossless compression algorithms. In *Data Compression Conference, 1997. DCC '97. Proceedings*, pages 201 –210, March 1997.

[22] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Willy Zwaenepoel. Diagnosing performance overheads in the Xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, VEE '05, pages 13–23, New York, NY, USA, 2005. ACM.

[23] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and modeling resource usage of virtualized applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 366–387, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[24] Omesh Tickoo, Ravi Iyer, Ramesh Illikkal, and Don Newell. Modeling virtual machine performance: challenges and approaches. *SIGMETRICS Perform. Eval. Rev.*, 37:55–60, January 2010.

[25] Rich Wolski. Experiences with predicting resource performance online in computational grid settings. *SIGMETRICS Perform. Eval. Rev.*, 30(4):41–49, 2003.