Understanding and Minimizing Disk Contention Effects for Data-Intensive Processing in Virtualized Systems

Kassiano J. Matteussi, Claudio Fernando Resin Geyer
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil 91509–900
Email: {kjmatteussi, geyer}@inf.ufrgs.br

Miguel G. Xavier, Cesar A. F. De Rose
School of Computer Science
Pontifical Catholic University of Rio Grande do Sul
Porto Alegre, Brazil 90619–900
Email: miguel.xavier@acad.pucrs.br
and cesar.derose@pucrs.br

***Abstract*—Distributed computing systems (e.g., clouds) have been widely employed to support an expanding range of applications. As the scale of data generation grows in regards to volume, velocity and variety (3Vs of big data), data-intensive processing became essential to extract valuable information from complex datasets. In this scenario, the infrastructure needs to meet the scaling demand of applications and must use resource management techniques to avoid interference problems. Literature review mainly focuses on CPU and memory solutions to handle resource contention problems in data-intensive processing. Complementarily, this paper further analyses and proposes techniques to minimizes disk contention effects in order to improve application performance in virtualized systems - technology that drives the cloud computing environment. For this objective, we present a general-purpose resource management strategy that adjusts dynamically disk I/O utilization rates. Results showed that the proposed approach improves application's performance by up to 26%.**

Keywords*— *Big Data, data-intensive processing, virtualization, resource contention, interference, disk contention.

I. Introduction

The technological advancements (e.g., health care, scientific sensors, user-generated data, Internet, financial companies, and supply chain systems) over the past two decades have shaped the term "Big Data" [1]. This abundance of information has attracted a great deal of attention in recent years and led organizations to find out ways to handle this explosion of data sets in an efficient, reliable and cost-effective way.

As the popularity of large-scale data analysis increases, the emergence of data-processing frameworks and programming models beyond MapReduce (MR) grows as well. Initially, MR was a very popular programming model that composed the Hadoop MapReduce (HMR) being its most popular and widely deployed implementation. Subsequently, we can cite other popular alternatives like Storm, Spark and Flink consolidated a new ecosystem of data-processing frameworks.

Distributed computing systems such as private and public clouds are frequently employed to support an expanding range of applications in which resources are provisioned as Virtual Machines (VMs). VMs offer excellent containment and isolation properties while attending issues relating to simplifying software dependencies, and can be scaled elastically as demand fluctuates [1]. In the case of MR processing, applications are generally heterogeneous and require flexible solutions to support the intensive resource variations - different usage characteristics (e.g., data requests over the network, disk I/O, CPU and memory). Although cloud scenarios offer proper support, the task of obtaining application and infrastructure requirements is complex and sometimes incurs in higher costs than expected, mainly due to overestimated or underestimated resource planning.

In this context, our previous work shows that uncontrolled disk resource usage can generate difficulties for the Operational System (OS) I/O scheduler [2] [3]. This effects are even worse for MR applications, because they perform buffered I/O requests (random data access). This characteristic determines I/O speed and throughput for Hard Disk Drivers (HDD) - providing more throughput for sequential requests than random ones [4].

Moreover, if there are MR jobs that need more processing time (long jobs) or jobs that finish quickly (small jobs), there is a higher probability of interleaving phases, with map and reduce requesting I/O resources at the same time. This could lead to disk throughput beyond bandwidth limit, resulting in disk contention (resource interference problem). Even OS I/O schedulers such as cfq, noop and deadline, that maintain I/O utilization balanced, may not be sufficient or adequate to supply overall I/O operations for data-intensive processing.

The disk contention problem incurs in several anomalies such as inter-VM resource interference, unexpected completion time of applications, throughput variations andperformance degradation in applications and frameworks [5] [6]. In addition, MapReduce-based frameworks and resource management systems like YARN and Mesos only performs CPU and memory orchestration, assigning to the OS I/O scheduler the responsibility to manage disk resources. Addressing this scenario, several researches can be found in the literature, but they lack solutions to handle disk management in virtualized systems [6] [5] [4] [7].

Otherwise, resource allocation and management in clouds is an ongoing research concern [1] [8] [5] [9] [10]. In fact, it

978-1-5386-7879-4/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCS.2018.00144

is well known that improving resource allocation strategies results in optimization in performance and efficiency [1]. Although there is a large amount of research into resource allocation exists, most of the strategies do not consider disk and network problems. Complementarily, our previously research efforts [2] [3] indicate that is feasible to use container-based clouds for data-intensive processing. It is because traditional virtualized system models (full-virtualization, paravirtualization, for instance) became inefficient to support high-performance computing environments. As consequence, several well-solved efforts in resource management remains opened for container-based virtualization.

This article proposes techniques to minimize disk contention effects in order to improve applications' performance in virtualized systems. Thus, through a general-purpose resource management strategy, each VM will have the disk resource dynamically adjusted in order to control application I/O utilization rates. To demonstrate this in practice, our approach was coupled with a set of big data applications and we observe an overall performance improvement up to 26%.

This paper is structured as follows. Section II shows an overview of the MapReduce model, its implementations and virtualization models. A relative study of disk I/O characterization of Big Data applications is presented in Section III. Section IV presents a resource contention overview and the disk resource allocation challenges faced in this work. In Section V, we present the proposed strategy to control I/O utilization rates and also present the preliminary evaluation. The related work is presented in Section VI. Conclusions and future work are presented in Section VII.

II. Background

This section provides an overview of the MR model and its implementations. Although there are several big data implementations currently available (e.g., Storm, Spark and Flink), this work will focus on Hadoop - the most popular open-source implementation for Big Data analysis.

A. The MapReduce Model and Implementations

The MapReduce programming model is based on the *map* and *reduce* primitives, both provided by the programmer. First, the *map* function takes a single instance of data as input and produces a set of intermediate key-value pairs. Secondly, the intermediate data sets are automatically grouped over the keys. Then, the *reduce* function takes as input a single key and a list of all values generated by the *map* function for that key. Finally, this list of values is merged or combined to produce a set of typically smaller output data, also represented as key-value pairs.

Moreover, Hadoop MapReduce is an open-source version of the MapReduce model and the Hadoop Distributed File System (HDFS), a distributed file system that provides resilient, high-throughput access to application data [11]. Furthermore, there are currently two different versions of Hadoop (1.x and 2.x):

- In classic Hadoop, the jobs are assigned in map and reduce tasks to be processed in multiple waves, and afterwards the data is consolidated in the disk. In this case, the Resource Manager (RM) keeps track of available resources (CPU, memory and data location) on each node of the cluster.
- Hadoop YARN (Yet Another Resource Negotiator) divides resources into logical partitions (called "slots" and "containers" respectively) which are assigned to executing tasks [1]. Besides, YARN decomposes into fine-grained, loosely coupled parallel tasks, scheduling on task-level rather than on job-level, improving fairness and utilization of resources. YARN mainly focuses on data locality, CPU and memory requirements and schedules the job to run on the most optimum nodes in the cluster [12].

Nowadays, Hadoop implementations and frameworks neglect disk management. This overloads OS I/O schedulers and leads to performance degradation during data-intensive processing.

Especially in multi-tenant environments. It is common to face Hadoop resource management challenges. For instance, multiple Hadoop jobs can be scheduled and performed on the same shared clusters [13]. In this case, disk and network will be served by arrival order - First In, First Out (FIFO). Thus, if there are any jobs with strict Service Level Agreements (SLAs), they can sometimes suffer slowdown due to low priority jobs. Another common problem seen in multi-tenant Hadoop clusters is overestimated or underestimated resource planning. The former results in low resource utilization and the latter in delays because jobs cannot be scheduled due to resource starvation.

B. Virtualized Systems

Resource virtualization consists of using an intermediate software layer on top of an underlying system to provide abstractions of multiple virtual resources. In general, the virtualized resources are called virtual instances and can be seen as isolated execution contexts. The hypervisor-based virtualization, in its most common form (hosted virtualization), consists of a virtual machine monitor (hypervisor) on top of a host operating system (Dom0) that provides a full abstraction to virtual instances.

A lightweight alternative to the hypervisors is container-based virtualization, also known as OS-Level virtualization. This model partitions the physical resources of machines, creating multiple isolated user space instances on the same OS. Despite this, users in these instances have the illusion that they are working on their own independent subsystem of network, memory, and file system.

LXC [14] is today's most notable system based on containers for Linux. It has gained space because of its inclusion in the Linux kernel upstream (mainline source code) and has increasingly been used in a variety of platforms that high scalability with low-performance overheads. The system is a Linux application tool that allows users to create containers containing groups of processes that might access its isolated instance of the global resource. To support containers, the Linux kernel provides the capacity of resource isolation per process or group of processes. This feature is referred to as Kernel namespace.

Virtualization based on containers has been experienced by large-scale computing platforms such as clouds, HPC clusters, and IT-related clusters. Regarding HPC clusters, Hadoop has recently undergone changes in its built-in components to support containers with the emergence of YARN. We have also witnessed several advances in IT infrastructures aiming at deploying IT servers upon clusters of containers at a large scale with high resilience.

Next, we present a study about resource characterization of big data applications. The study can help users to avoid high expenditures in resource planning by reducing resource contention and interference in their applications.

III. Disk I/O Characterization of MapReduce Applications

Data-intensive processing is the key for big data analysis. In this scenario, the big data applications exhibit high demand for resources like memory, CPU and I/O [15] as well as they require efficient management to support large datasets.

This section aims to understand how MR applications manage the data processing flow internally. For this case, we observed some I/O characteristics such as data generation process, disk read/write bandwidth, data access patterns and application completion time. Complementarily, we generated resource usage profiles for CPU, memory, disk and network. Finally, we present some insights about disk-related problems.

In order to obtain these insights, we performed several experiments. Our hardware is comprised of a real cluster consisting of 4 identical servers, each equipped with 8 x86_64 cores at 2.27 GHz and 16GB of RAM. The servers were interconnected by an Ethernet switch with 1 Gbps links. In terms of software, all servers ran the data processing framework (e.g., Hadoop 1.1.2 for now, but we are going to use YARN shortly) installed on top of an Ubuntu Linux 14.04 LTS operating system. For Hadoop 1.x, each node was configured with 8 map task slots and 8 reduce task slots.

For our experiments, we used HiBench [16], a realistic and comprehensive benchmark suite based on the MapReduce programming model [17]. HiBench includes both synthetic micro-benchmarks, and real world applications such as Sort, Terasort, WordCount, Nutch, PageRank, Bayes, K-means and Hive. These models of applications are widely used for performance analysis, e-commerce, Web search engines, classification, clustering, image processing, textual recognition as well as they are popular in the current big data era. The applications are presented in Table I. Next, we will pinpoint some observations about obtained results.

1) Data generation process: At the first stage of application processing, HiBench utilizes specific data generation tools to create static data sets. These datasets are defined by the user and fully processed posteriorly by the applications, ensuring a real scenario for experimental labs. We present this behavior in Figure 1. Each application presents this stage at the beginning of the trace. The stage is a part of job a sequence and is called number 1. The resource utilization traces of Nuth Indexing and Page Rank were suppressed because they present memory and CPU-bound characteristics, both of which are not evaluated in this work.

TABLE I
Workload characteristics

Application	Load in	Load out	Time (s)
Sort	15,9Gb	15,1Gb	2225
Terasort	37,2Gb	37,2Gb	1265
WordCount	11,4Gb	1Mb	2286
Nutch	509Mb	208,4Mb	266
Pagerank	1,3Gb	476Mb	160
K-means	46,4Gb	57Gb	3038
Bayes	352Mb	1Mb	401
Hive	5Gb	1,7Gb	788

2) Data access pattern: The set of analyzed applications present in Figure 1 maintain the data-intensive I/O in HDFS continuously. In MR, the disk intensive operations occur on maps (multiples inputs and outputs), reducers (multiples inputs and outputs) and shuffle phases (multiples transfers and sort operations).

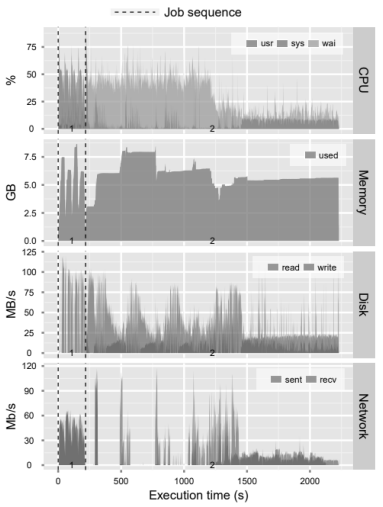
The number of map tasks within a MR job is driven by the number of data blocks in the input files. For example, considering a data block size of 128 MB, a MR job with an input of 10 TB will have 82K map tasks. Therefore, there are potentially more map tasks than task slots in a given cluster, which forces tasks to run in waves [18] [19] incurring high resource utilization rates. Moreover, some MR jobs demand more processing time creating an overlap between the execution of map tasks. This situation can lead to performance degradation due to resource contention, where multiple tasks require resources at the same time.

Furthermore, the amount of data generated by reducers can vary from application to application. Normally, reducers tend to generate significantly smaller outputs than the input data. However, as reported in work, MR workloads also perform data expansion (output data size $\gg$ input data size) or data transformation (output data size $\approx$ input data size). Finally, the output data of MR jobs are written to HDFS following the pipelined write procedure. The main difference, in this case, is that there are R reducers simultaneously writing to HDFS, instead of a single client application.

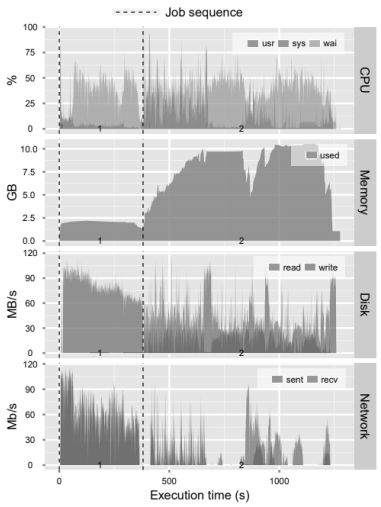
3) Resource utilization analysis: Figure 1 presents the resource utilization variations from multiple applications. Although, the presence of data access patterns in MR clusters is a well-documented phenomenon [20] [21], this work focuses in analyzing feasible solutions to group applications based on their disk resource patterns.

For instance, Sort 1(a) application performs more write operations and is more disk intensive in map phases than K-means 1(d). Next, K-means 1(d) is more disk intensive in the reduce phase. Then, imagine these applications running together into a cloud, their resource utilization matches, avoiding cluster underestimate scenario due to the well-used resource pool.

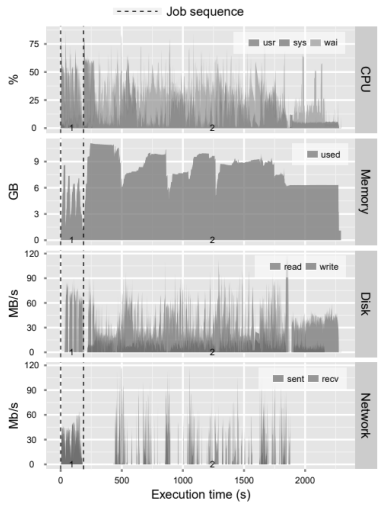
This approach can combine applications side-by-side until resource pooling guarantees isolation between them. Furthermore, it is possible to group applications by memory, CPU,



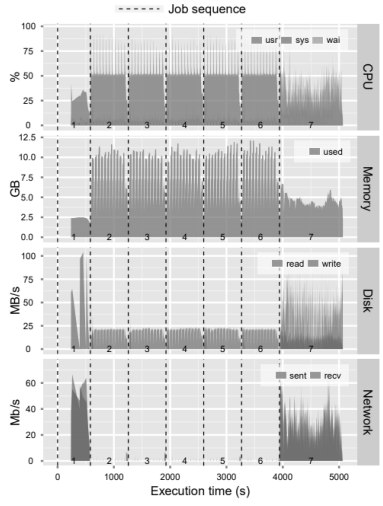
(a) Sort application, jobs: (1) random text writer, (2) sorter Sort



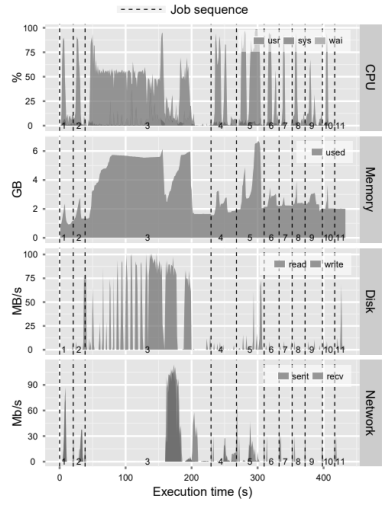
(b) Terasort application, jobs: (1) teraGen, (2) terasort



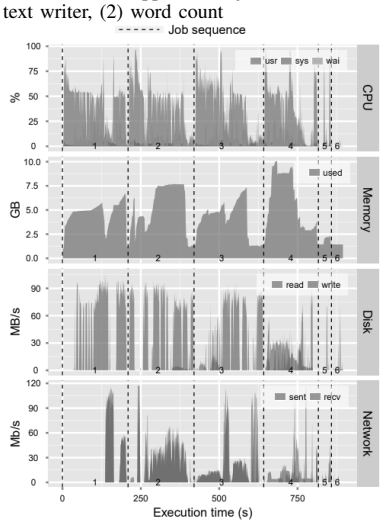
(c) Wordcount application, jobs: (1) random text writer, (2) word count



(d) K-means application, jobs: (1) datatools, (2-6) cluster iterations, (7) cluster classification



(e) Bayes applications, Jobs: (1) create bayes data, (2) document tokenizer, (3) generate collocations, (4) compute NGrams, (5) dictionary vectorizer, (6,9) partial vector merger, (7) vector TfIdf, (8) make partial vectors, (10,11) train index mapper-reducer



(f) Hive application, jobs: (1) create rankings, (2) create uservisits, (3) Job INSERT uservisits, (4-6) Job INSERT ranking stages

Fig. 1. Workload Characterization of MapReduce Applications): Sort, b) Terasort, c) Wordcount, d) K-means, e) Bayes, f) Hive. The heterogeneity of applications incurs multiple IO Utilization Patterns. Thus, the application traces analysis allows us a set of opportunities for optimization. For instance, we can observe several underestimated resource points in such a way that it is possible to adjust the resources in order to avoid underestimated resource planning and infrastructure expenditures.

network, disk or all resources. However, in this work, we focus only in disk I/O that incurs bottlenecks during data-intensive processing. Further, we characterize these applications. The metrics chosen for this are application completion time and resource utilization rates from CPU, memory, disk and network. Table II summarizes the classification.

Accordingly, the characterization motivated the possibility of improving disk resource distribution between big data applications. Meanwhile, the next section emphasizes disk isolation and resource allocation in order to make the general-purpose resource management strategy feasible, which adjusts

TABLE II
WORKLOAD CHARACTERISTICS

App Name	Figure	Type
Sort	1(a)	disk-intensive
Terasort	1(b)	disk-intensive
Wordcount	1(c)	disk-intensive
Hive	1(d)	disk/memory-intensive
Bayes	1(e)	CPU/memory-intensive
Kmeans	1(f)	CPU/memory-intensive

the disk I/O utilization rates statically.

IV. Disk I/O Contention and Resource Allocation

Uncontrolled access to shared resources can cause performance variations that lead applications to fail or run unsteadily. The friction generated by the competition to access RAM, disk storage, cache or internal busses is called resource contention. Many efforts have been made to alleviate the contention in the operating system level that range from better scheduling techniques in multi-core architecture [22] to dynamic addressing mapping to minimize memory contention. The steady growth of big data applications brought a concern about I/O contention and its impact in environments in which performance is crucial and SLA cannot be violated, such as clouds. I/O contention occurs when multiple virtual instances compete for a disk bandwidth portion in a scenario where the demand is higher than the available resource. To illustrate, the dispersion in Figure 2 presents a disturbed scenario in which two disk-intensive applications write to/read from a single disk while the bandwidth is not enough to supply the applications demands, making the performance fluctuate.

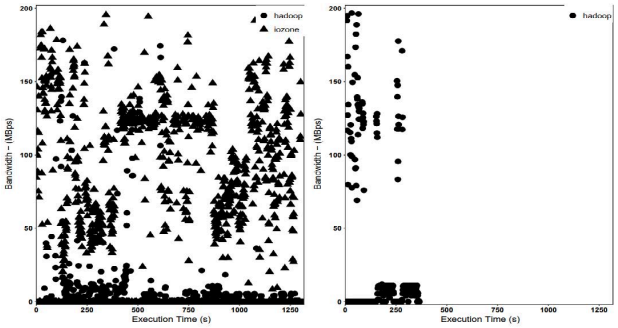


Fig. 2. Performance interference between two co-located applications due to disk contention

Operating system level I/O schedulers, such as CFQ, deadline and noop, detect resource utilization bottlenecks and attempt to divide block devices by reordering/prioritizing tasks in a fairly-balanced manner. As a result, the overhead is distributed equally across the consolidated instances, but it does not prevent their performance from varying unpredictably since the schedulers are unable to predict and make decisions based on workload characteristics.

On the other hand, performance interference may also be sourced from isolation issues in the virtualization layer, which occur when an application exceeds the amount of allocated resources. Even though a virtual instance receives a limited portion of resources, it does not prevent the resource utilization from leakage due to isolation flaws [23]. Hence, performance interference may be sourced from either resource contention or performance isolation issues. Data center administrators exaggerate the amount of allocated resources to sidestep performance interference, leading the cluster to low utilization and making it no longer resource efficient.

A. I/O Allocation in Container-based Systems

Container-based systems have become a tendency under HPC environments [3]. Promises of high performance and scalability have made such systems quite popular nowadays. A container consists of a group of processes that cannot communicate outside the box and have the impression that they are working in their own isolated system. Namespace is the kernel feature that implements the isolation layer and prevents multiple consolidated containers from seeing each other in the resource substrate (network, file-system, IPC, etc). From the resource restriction standpoint, containers leverage *Cgroup* subsystems to ensure resource guarantees and prevent resource utilization leakage.

The *Cgroup* subsystem "blkio" implements the block I/O controller, which is responsible for throttling per-process I/O bandwidth or disk policy based on the division of proportional weight time. In both cases, *blkio* works exclusively for unbuffered direct writes/reads and it does not restrict the I/O resource for MR applications at the current state. For this purpose, we patched the kernel to allow I/O throttling not just for direct writes, but also for buffered writes such as those carried out by MR-centric applications. In Figure 3, we illustrate the bandwidth consumption before and after the feature was installed.

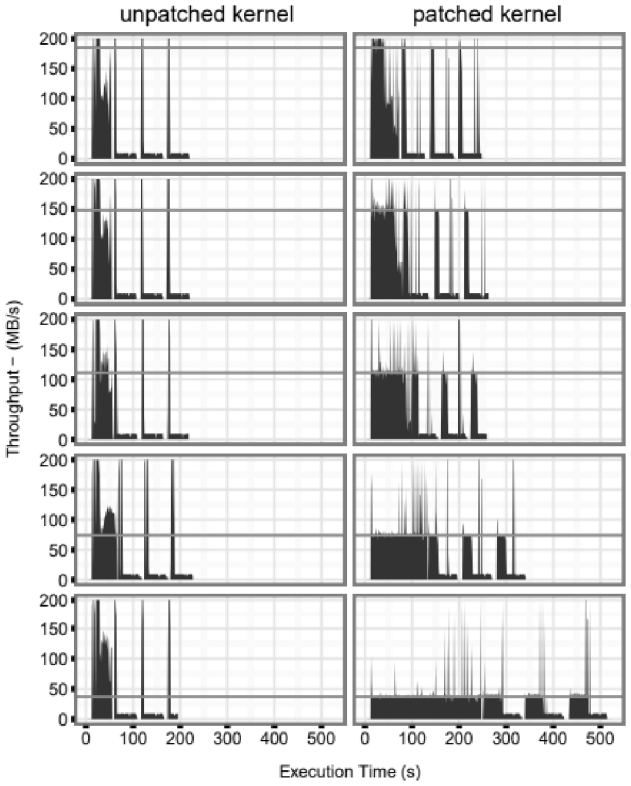


Fig. 3. I/O bandwidth restriction of buffered writes in patched and unpatched kernel

The evaluation of the *Cgroup* block controller reveals that, even with the feature, the I/O resource restriction does not always works properly and subjects to performance leakage. The performance isolation issue is clearly observed over all assessed limits.

V. Towards a Dynamic Disk Resource Allocation

In this section, we present a general-purpose resource management strategy to dynamically adjust the disk I/O utilization rates. The proposed strategy aims to improve disk resource allocation in order to minimize contention levels. We believe that an adequate I/O coordination of throughput based on disk bandwidth ensures the VM isolation. Thus, if interference is avoided, the application can run without noise in a shared cloud, for instance.

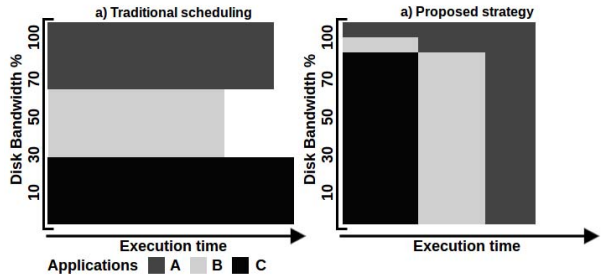


Fig. 4. Resource management comparison between a) current system level I/O schedulers like cfq, noop and deadline and, b) the proposed strategy

In figure 4, a) the scheduler reorders/prioritizes the disk resources (bandwidth) of each application in a fairly balanced manner. This approach does not ensure performance for the applications, but tries to alleviate starvation and interference problems. However, data-intensive workloads process large data sets and it leads the OS scheduler to incur bottleneck. It occurs because the resources consumption fluctuates depending on the processed workload and application type. In Figure 4, b) we present our strategy based on the Shortest Job First (SJF) algorithm [24]. This is a greedy-based algorithm that selects the process with the shortest execution time. Although SJF generates the starvation problem (well-documented problem), it is avoided in this work because we guarantee resources for all applications and always resize resources in real-time when one application finishes.

Our SJF version is a quite similar strategy; in this case, we always prioritize the disk resources for the lowest workload (in terms of size) - not taking into account the application type. Thus, we divided the disk resources between all applications, the highest slice of resources goes to the application i (the resource allocation is maintained until the application finishes) with the lowest workload and, the rest of resources goes to the subsequent applications $i+1$ to n. The main idea of this approach is to use the largest slice of resources to maximize the throughput and number of process per a period of time to the application i, leading it to higher performance gains. When the application i finishes, another i is chosen and the flow is again repeated.

In addition, the proposed strategy is designed to behave in a general-purpose manner in order to support multiple virtualization models. The throughput rates are instrumented dynamically using *Cgroup* block (blkio) for each VM in the host. As a result, we believe that the strategy minimizes the completion time of the application and guarantees small levels of interference in a shared environment, see Figure 4 a) e b). In the next section, we present our preliminary results.

A. Evaluation

This section presents the test scenario in which we apply the strategy proposed in this work, its evaluations and discussions. For all experiments, we compared the Linux I/O Schedulers (CFQ, noop, deadline) to demonstrate that by prioritizing applications dynamically it is possible to minimize disk contention and accelerate all co-located applications independent of the scheduler. The metric observed was makespan (MS), which represents the application completion time considering the beginning and end of a sequence of jobs or tasks. Measurements have a confidence interval of at least 95%.

B. Experimental Setup

Our hardware setup comprised two identical Dell PowerEdge M610 machines, with two Xeon Six-Core E5645 2.4GHz processors (totalizing 24 cores per node with Hyper-threading), 24GB of RAM and one 300GB 10K RPM SAS disk. The machines software stack was composed by Ubuntu Server 14.04.1 LTS, patched with a custom Kernel (version 3.3) to support buffered I/O writes performed by Hadoop[1]. We compiled the current kernels and discovered that until version 3.x buffered writes are not supported.

On top of OS, we employed the container-based virtualization (LXC). We also used MRv1 Hadoop stack (HDFS, MapReduce, etc.), replication factor three and 128MB block size. Map and reduce tasks were configured to 6 and 1 respectively. To monitor disk I/O resource utilization we used an instrumented version of IOtop[2]. For workloads, applications and data sets came from HiBench [16].

C. Preliminary Experiment

This experiment uses two identical nodes; each one has three containers with equivalent amounts of resources for CPU and memory. In addition, each container hosts just one MR Teragen application (provided by HiBench) with varied workloads (10, 20 and 30GB). Teragen was chosen due to disk-intensive behavior (read and write operations).

According to our strategy, disk write operations were dynamically limited in order to maintain the biggest resource slice to the container that comprises the lowest workload. In this case, the disk resource allocation were the following: Teragen 10GB receives 70%, Teragen 20GB 20% and Teragen 30GB 10%, where the presented rating values are fixed arbitrarily to help us to understand how the strategy works. This experiment allows us to observe that by concentrating disk resources in one application at a time, following the shortest job strategy, interference is minimized and helps to reduce the global makespan. In this scenario, when Teragen 10GB finishes, its disk resources are allocated to the container that hosts the application with the next lowest workload, and so on. The result can be observed in Figure 5 (bottom graphs labeled with restriction).

[1]Custom Kernel: git://git.kernel.org/pub/scm/linux/kernel/git/wfg/linux.git buffered-write-io-controller

[2]Custom Iotop: https://github.com/mvneves/iotop-cgroups

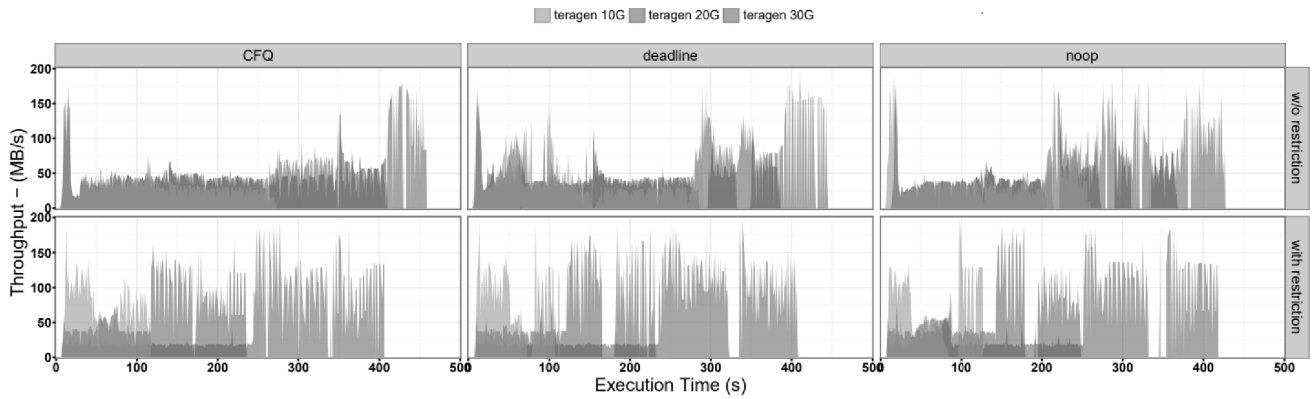


Fig. 5. Per-scheduler disk throughput analysis with and without the proposed strategy. The completion time of applications was affected by interference sourced from disk contention and isolation weaknesses

We can observe in Figure 5 that the proposed strategies ensure the isolation and reduce the application completion time for all OS schedulers. The improvement happens because the interference between applications is minimized, ensuring disk isolation. Sometimes the isolation fails (it can be observed clearly in Figure 5 at time 50' in CFQ and noop experiments - with restriction) and OS schedulers try to equalize the resources, reducing the throughput. In this scenario, the performance can be degraded and the interference can lead to contention because of OS schedulers difficulties to cope with concurrent disk operations that are intensive. A summarization of the improvements in makespan for each scheduler is presented in Table III.

TABLE III
MAKESPAN IMPROVEMENT BY SCHEDULER

State	CFQ	deadline	noop
w/o restriction (sec)	382	395	422
restriction (sec)	307	297	293
performance improvement (%)	19,6	24,8	30,56

The literature consideres noop the simplest OS scheduler. It does not perform sorting or any other optimization to minimize disk latency. The disk requests are just inserted into a queue to be processed later on, reducing the latency for disk intensive applications that perform random-access (buffered writes) in detriment of others. The Deadline I/O scheduler attempts to provide a guaranteed latency for all requests to avoid I/O starvation. However, in contention scenario, especially for data-intensive processing, this behavior can incur some interference, leading to low application performance. CFQ provides a fair resource sharing to coordinate I/O bandwidth for all the processes that request an I/O operation. Still, CFQ maintains the process which requests I/O (synchronous) in a queue. For asynchronous requests, the processes are batched together according to their process I/O priority. In this scenario, data-intensive processing suffers from excessive control, flooding the scheduler with multiple processes.

D. Real World Scenario

This experiment uses the same strategy as section V, in which disk write operations were dynamically limited in order to maintain the biggest resource slice to the container that executes the lowest workload. The applications and workloads were changed to represent a more realistic scenario. The set of chosen applications were Terasort (20GB), Sort (1.1GB) and K-means (3GB). The resource allocations were as follow 90% to the application i and 10% to the subsequent applications $i+1$ to n. The results are presented in Table IV.

We can see in Table V that the applications with mixed workloads respond well to the proposed disk allocation strategy. Although still significant, makespan gains are not as good as in our preliminary experiment. This is expected because in this experiment we use applications that have different workload patterns, which reduces disk interference, and consequently the potential benefits of our strategy.

TABLE IV
REAL WORLD SCENARIO: MAKESPAN IMPROVEMENT BY SCHEDULERS

State	CFQ	deadline	noop
w/o restriction (sec)	1614	1381	1525
restriction (sec)	1448	1131	1123
performance improvement (%)	10	18	26

Table V presents the detailed performance analysis by application. We can observe that Terasort has an increase in its execution time in all cases, which is expected in some cases since we are optimizing global makespan. This is the same effect that may happen with resource greedy applications in operating system scheduling. Moreover, in combination with our strategy, the obtained results indicated noop as the best scheduler for data-intensive processing.

VI. RELATED WORK

Many researches [8] [6] [5] [4] [7] [25] investigate performance interference and resource contention. These solutions perform adjustments in processing frameworks; use linear regression based on trace files to control the execution of running applications; propose new I/O schedulers; profile the applications' performance; throttle and coordinate slowest tasks and try to use slices of resources to manage only read requests - due to the kernel restrictions (solved in our approach); and test link aggregation and SDN strategies to improve communication channels. The researches presented

TABLE V
DETAILED PERFORMANCE ANALYSIS PER APPLICATION

Schedulers	CFQ			Noop			Deadline		
Applications	**K-means**	**Sort**	**Terasort**	**K-means**	**Sort**	**Terasort**	**K-means**	**Sort**	**Terasort**
w/o restriction (sec)	877	1252	2713	1417	1158	2000	1287	987	1546
Restriction (sec)	737	859	2749	676	564	2131	699	522	1871
Gain (%)	15,9	31,3	-1,6	52,2	51,2	-6	45,6	47,1	-20

the importance and the concern of investigating interference in virtualized environments.

However, these researches propose improvements that are tightly coupled with the target virtualization models, what restricts their applicability. Our work in the other hand, presents a general-purpose resource management strategy to minimize interference and optimize global makespan, which can be applied with any virtualization layer.

VII. CONCLUSION AND FUTURE WORK

In our effort to better understand and minimize disk contention effects for data-intensive processing in virtualized systems, we claim that a general-purpose resource management strategy to dynamically adjust disk I/O utilization rates is feasible and effective for this scenario. As presented in our preliminary evaluation, the strategy improves application performance by up to 26%. As future work, we intend to create a fine-grained dynamic model to resize resources on-demand. Furthermore, we intend to test the new features provided by kernel 4 and also measure the impact of Solid-State Drive (SSD) usage for data-intensive processing.

VIII. ACKNOWLEDGMENT

The authors would like to thank Dell Inc and the following Brazilian Agencies: FAPERGS Projects "GREEN-CLOUD - Computação em Cloud com Computação Sustentável" (#16/2551-0000 488-9) and "SmartSent" (#17/2551-0001 195-3), CAPES, CNPq and PROPESQ-UFRGS-Brasil for supporting this work.

REFERENCES

[1] T. Ryan and Y. C. Lee, "Multi-tier resource allocation for data-intensive computing," *Journal of Big Data Research*, vol. 2, no. 3, pp. 110–116, 2015.

[2] M. Xavier, I. De Oliveira, F. Rossi, R. Dos Passos, K. Matteussi, and C. De Rose, "A performance isolation analysis of disk-intensive workloads on container-based clouds," in *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, March 2015, pp. 253–260.

[3] M. G. Xavier, K. J. Matteussi, F. Lorenzo, and C. A. De Rose, "Understanding performance interference in multi-tenant cloud databases and web applications," in *Proceedings of the IEEE International Conference on Big Data (Big Data)*, Dec. 2016, pp. 2847–2852.

[4] K. R. Krish, B. Wadhwa, M. S. Iqbal, M. M. Rafique, and A. R. Butt, "On efficient hierarchical storage for big data processing," in *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 403–408.

[5] M. Malensek, S. L. Pallickara, and S. Pallickara, "Alleviation of disk I/O contention in virtualized settings for data-intensive computing," in *Proceedings of the 2nd IEEE/ACM International Symposium on Big Data Computing (BDC)*, Dec 2015, pp. 1–10.

[6] M. Siyuan, X.-H. Sun, and I. Raicu, "I/O throttling and coordination for mapreduce," Illinois Institute of Technology, Tech. Rep., 2012.

[7] P. Mishra, M. Mishra, and A. K. Somani, "Bulk I/O storage management for big data applications," in *Proceedings of the 24th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sept 2016, pp. 412–417.

[8] F. D. Rossi, G. D. C. Rodrigues, R. N. Calheiros, and M. D. S. Conterato, "Dynamic network bandwidth resizing for big data applications," in *Proceedings of the 13th IEEE International Conference on e-Science (e-Science)*, Oct 2017, pp. 423–431.

[9] B. Hou, F. Chen, Z. Ou, R. Wang, and M. Mesnier, "Understanding I/O performance behaviors of cloud storage from a client's perspective," *Journal of ACM Transactions on Storage (TOS)*, vol. 13, no. 2, pp. 16:1–16:36, June 2017.

[10] S. Amri, H. Hamdi, and Z. Brahmi, "Inter-VM interference in cloud environments: A survey," in *Proceedings of the 14th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, Oct. 2017, pp. 154–159.

[11] D. Borthakur, "The hadoop distributed file system: Architecture and design," Jan 2007. [Online]. Available: https://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.17/docs/hdfs_design.pdf

[12] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop YARN: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC)*, Oct. 2013, pp. 5:1–5:16.

[13] M. Pastorelli, A. Barbuzzi, D. Carra, M. Dell'Amico, and P. Michiardi, "Practical size-based scheduling for mapreduce workloads," *CoRR*, May 2013. [Online]. Available: https://arxiv.org/abs/1302.2749

[14] "Linux Containers," 2018. [Online]. Available: https://linuxcontainers.org/

[15] F. Pan, Y. Yue, J. Xiong, and D. Hao, "I/O characterization of big data workloads in data centers," in *Journal of Big Data Benchmarks, Performance Optimization, and Emerging Hardware (BPOE)*, 2014, pp. 85–97.

[16] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench Benchmark Suite: Characterization of The MapReduce-Based Data Analysis," in *Proceedings of the 26th IEEE International Conference on Data Engineering Workshops (ICDEW)*, March 2010, pp. 41–51.

[17] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[18] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proceedings of 8th the USENIX Conference on Operating systems design and Implementation (OSDI)*, Dec. 2008, pp. 29–42.

[19] M. V. Neves, "Application-aware software-defined networking to accelerate mapreduce applications," Ph.D. dissertation, Pontifícia Universidade Católica do Rio Grande do Sul, 2015.

[20] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: Coping with skewed content popularity in mapreduce clusters," in *Proceedings of the 6th Conference on Computer Systems (EuroSys)*, Apr. 2011, pp. 287–300.

[21] C. L. Abad, Y. Lu, and R. H. Campbell, "DARE: Adaptive data replication for efficient cluster scheduling," in *IEEE International Conference on Cluster Computing (CLUSTER)*, Sept. 2011, pp. 159–168.

[22] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proceedings of the 15th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, vol. 38, no. 1, 2010, pp. 129–142.

[23] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Proceedings of the 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, March 2013, pp. 233–240.

[24] E. W. Davis and J. H. Patterson, "A comparison of heuristic and optimum solutions in resource-constrained project scheduling," *Management Science*, vol. 21, no. 8, pp. 944–955, 1975.

[25] C. Rista, D. Griebler, C. A. F. Maron, and L. G. Fernandes, "Improving the network performance of a container-based cloud environment for hadoop systems," in *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS)*, July 2017, pp. 619–626.