Exploiting Spatial Locality to Improve Disk Efficiency in Virtualized Environments

Xiao Ling *, Shadi Ibrahim†, Hai Jin*, Song Wu*, Songqiao Tao*

*Cluster and Grid Computing Lab
Services Computing Technology and System Lab
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, 430074, China
{hjin, wusong}@hust.edu.cn

†INRIA Rennes - Bretagne Atlantique
Rennes, France
shadi.ibrahim@inria.fr

***Abstract*—Virtualization has become a prominent tool in data centers and is extensively leveraged in cloud environments: it enables multiple *virtual machines* (VMs) — with multiple operating systems and applications — to run within a physical server. However, virtualization introduces the challenging issue of preserving the high disk utilization (*i.e.*, reducing the seek delay and rotation overhead) when allocating disk resources to VMs. Exploiting spatial locality, a key technique for improving disk utilization and performance, faces additional challenges in the virtualized cloud because of the transparency feature of virtualization (hypervisors do not have the information about the access patterns of applications running within each VM). To this end, this paper contributes a novel disk I/O scheduling framework, named *Pregather*, to improve disk I/O efficiency through exposure and exploitation of the special spatial locality in the virtualized environment (regional and sub-regional spatial locality corresponds to the virtual disk space and applications' access patterns, respectively), thereby improving the performance of disk-intensive applications without harming the transparency feature of virtualization (without a *priori* knowledge of the applications' access patterns). The key idea behind *Pregather* is to implement an intelligent model to predict the access regularity of sub-regional spatial locality for each VM. We implement the *Pregather* disk scheduling framework and perform extensive experiments that involve multiple simultaneous applications of both synthetic benchmarks and a MapReduce application on Xen-based platforms. Our experiments demonstrate the accuracy of our prediction model and indicate that *Pregather* results in the high disk spatial locality and a significant improvement in disk throughput and application performance.**

Keywords*—*virtualization; disk-intensive; I/O scheduling; spatial locality; efficiency

I. Introduction

Virtualization has become a prominent tool in data centers and is extensively leveraged in cloud environments: it enables multiple *virtual machines* (VMs) — with multiple operating systems and applications — to run within a physical server. For example, Amazon web services [1] rely on the Xen virtualization hypervisor to provide the VM-based infrastructure as a service (IaaS) solution, which enables users to lease and customize their environments in order to run their applications. Virtualization however imposes new challenges in the scheduling and the allocation of system resources. With the volume of data growing rapidly and applications becoming more disk-intensive [2][3], allocating disk resources efficiently while preserving a high disk throughput becomes of key importance in virtualized environments.

Exploiting spatial locality is an important technique for improving disk efficiency and performance (*i.e.*, high spatial locality results in a significant reduction in seek delay and rotation overhead, which leads to high disk efficiency and low power consumption). For example, traditional file systems often allocate the accessed data of a process as contiguous blocks if possible, so disk scheduling can easily exploit spatial locality. Unlike traditional environments, in a virtualized environment achieving high spatial locality is a challenging task, due to the *transparency feature of virtualization* which causes semantic isolation between the hypervisor and guest VMs. As a result, the block I/O layer lacks a global view of the I/O access patterns of processes [4]. The lack of coordination between file systems in both the hypervisor and VMs reduces the efficiency of exploiting disk locality in virtualized environments. Moreover, another use has recently emerged: sharing an infrastructure between multiple applications with different disk I/O characteristics (*mixed applications*). A VM may therefore encapsulate more than one application, which in turn increases the complexity and irregularity of the I/O behavior of the VM (*e.g.*, multiple applications, including file-editing and media streaming, run in a virtual desktop).

Research effort has been directed toward improving disk efficiency through exploiting spatial locality in virtualized environments. These efforts use either *invasive mode scheduling* (*i.e.*, select the disk pair schedulers within both the hypervisor and the VM according to the applications' access patterns [5][6]), or *non-invasive mode scheduling* (*i.e.*, schedule the I/O requests while treating a VM as a black box [7][8]). However, the aforementioned solutions target similar types of applications (mainly read-dominated applications), and cannot be applied when a VM encapsulates mixed applications [4][9]. Moreover, they come at the cost of violating the transparency feature of virtualization. This paper follows this line of research and contributes to the goal of improving the performance of disk-intensive applications in virtualized environments by enabling efficient disk utilization while preserving the transparency feature of virtualization and ensuring the high throughput for complex I/O workloads, including write-dominated applications or mixed applications. In particular, we aim at detecting and exploiting spatial locality in virtualized environments. In this paper we make the following three contributions to achieve this goal:

1) We investigate the spatial locality of disk data accesses in virtualized environments. By tracing the I/O

requests of VMs with mixed applications, we observe that the disk data accesses are grouped into *regions*, bounded by the virtual disk sizes, and within each region the disk data accesses are grouped into *sub-regions*, which correspond to the applications' access patterns.

2) We introduce an intelligent prediction model that uses a temporal access-density clustering algorithm to analyze the data access of a VM running mixed applications. Our model can predict the distribution of sub-regions with spatial locality within each region (*i.e.*, detect the sub-regional spatial locality for each VM) and the arrival times of future requests accessing these sub-regions.
3) We propose *Pregather*, a disk scheduling framework with a spatial-locality-aware heuristic algorithm in the hypervisor for exploiting the *special* spatial locality (*i.e.*, the regional and sub-regional spatial locality) to reduce disk seek and rotational overhead in the virtualized environment: *Pregather* does that — thanks to our prediction model — *without any prior knowledge* of the access patterns of the applications running within each VM.

We build the *Pregather* scheduling framework in Xen. Our evaluations, using both synthetic benchmarks and a MapReduce application (distributed sort), demonstrate the accuracy of the proposed intelligent prediction model and the throughput benefits from our heuristic scheduling algorithm: *Pregather* achieves high disk spatial locality, and thus improves the disk utilization and the applications' performance. For example, in contrast to the default Xen disk I/O scheduler -*Completely Fair Queuing* (CFQ), *Pregather* achieves throughput performance improvement by a factor of 1.6x when mixed applications are running within a VM.

The rest of the paper is organized as follows. Section II discusses the related work. Section III zooms on the disk access patterns in virtualized environments with mixed applications. Section IV discusses our prediction model and section V describes *Pregather* along with the spatial-locality-aware heuristic algorithm. Section VI details the performance evaluation. Finally, we conclude the paper in section VII.

II. Related Work

Ever since the advent of virtualization technology, a huge number of studies have been dedicated to improving the performance of disk-intensive applications in virtualized environments [4–10]. On one hand, some of these solutions use *invasive mode* scheduling to manage I/O requests [5][6][10]. They introduce an additional hypervisor-to-VM interface to achieve better coordination between the disk scheduler within both the hypervisor and VMs. However, these solutions can only be applied when VMs are running the same type of application. Moreover, they require the hypervisor to be aware of the applications running within VMs, which harms the transparency feature of virtualization.

On the other hand, some solutions use *non-invasive mode* scheduling to manage the I/O requests in virtualized environments without harming the transparency feature of virtualization [4][7–9]. *Streaming scheduling* (SS) [9] turns any work-conserving disk scheduler into a non-work-conserving one based only on the request's own locality, thus reducing the disk seek time. SS essentially examines the existence of a stream by analyzing the characteristics of requests with relative spatial locality. Antfarm [4] enables the hypervisor to track the creation and exits of processes in VMs to infer the information of processes. The process information can help the disk scheduler at the hypervisor level to check the existence of read streams and map requests at the right read stream. However, both SS and Antfarm can only infer read streams, and cannot be applied for write applications or when mixed applications are running within a VM.

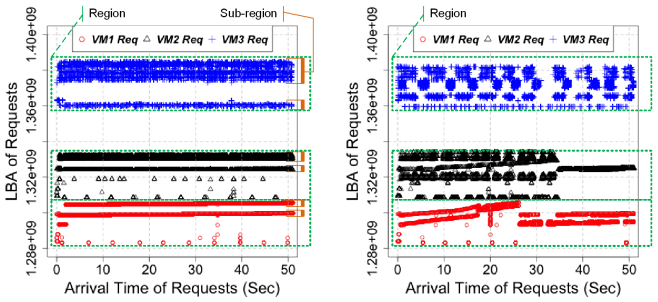
To the best of our knowledge, analyzing the regularity of data accesses of VMs has thus far been performed *only when a specific (one) application* is running within a VM (such as online transaction processing, mail server or file migration) [11–14]. Moreover, exploring the benefits of predicting the access locality and regularity of processes to exploit the disk spatial locality (*e.g.*, by facilitating I/O perfecting) has so far been discussed only in *non-virtualized environments* [15–17]. These studies cannot be applied in virtualized environments, because unlike general processes, the access patterns of VM processes are more complicated and frequently changing. Our own work focuses on investigating and exploiting the disk access locality and regularity in *virtualized environments* when *mixed applications* are running within each VM. As far as we know, *we are the first to explore the benefits of the locality and regularity of data accesses to improve the disk efficiency in the presence of mixed applications while preserving the transparency feature of virtualization*.

III. Zoom on Disk Access Patterns in Virtualized Environments

In this section, we seek to obtain an overview of understanding the disk spatial locality in virtualized environments. Ideally, we would like to get a rough idea of the disk access patterns in virtualized environments, and the impact of both virtualization features and mixed applications on the access patterns.

Experimental setup. We conduct our experiment in one physical node, equipped with four quad-core 2.40GHz Xeon processor, 22GB of memory and one dedicated SATA disk of 1TB, running CentOS5 with kernel 2.6.18. All results described in this section are obtained using Xen version 4.0.1 with Blktap AIO driver [18]. Three guest VMs are deployed within this physical machine. The guest VM is configured with two virtual CPUs, 1GB memory and 12 GB virtual disk (the virtual disk is mapped to a default file-backed image). We set the disk scheduler of VMs to the default Noop[1]in order to send requests as early as possible, and set the disk scheduler of the hypervisor to the default CFQ. As stated earlier, we want to study the disk access patterns when mixed applications are running, thus running mixed I/O workloads. In the experiments, two sequential access applications run on VM1, a sequential access application and a random access application run on VM2, and two random access applications run on VM3. As shown in Table I, we use *sysbench* [19] to generate these workloads with different disk access patterns. In

[1]Noop has recently been used as the default VM scheduler because any (re)ordering at the VM level will be counterproductive as I/O requests from several VMs will be managed (dispatched) to the disk device according to the disk scheduler at the hypervisor level. Thus it is better to simply push the requests as early as possible, and therefore save CPU cycles and avoid any conflict that could occur between the I/O schedulers at the hypervisor and the VM levels.



(a) Read-dominated applications: *VMs with only sequential applications exhibit stronger sub-regional spatial locality (i.e., sub-regions have smaller ranges) than VMs with mixed or random applications.*

(b) Write-dominated applications: *Sub-regional spatial locality can hardly be noticed for write applications, especially for random applications.*

Fig. 1: **Disk access patterns of requests in virtualized environments**: *The LBAs of the requests from the same VMs can be grouped into regions.*

TABLE I: **The description of the workloads generated by *sysbench***

Workload	Description
Sequential Read (*SR*)	16 threads sequentially read 128 files, with total size of 1 GB
Random Read (*RR*)	16 threads randomly read 128 files, with total size of 1 GB
Sequential Write (*SW*)	16 threads sequentially write 128 files, with total size of 1 GB
Random Write (*RW*)	16 threads randomly write 128 files, with total size of 1 GB

the first experiment (read-dominated applications), VM1 runs two *SR* applications, VM2 runs one *SR* and one *RR* application and finally VM3 runs two *RR* applications. In the second experiment (write-dominated applications), VM1 runs two *SW* applications, VM2 runs one *SW* and one *RW* application and VM3 runs two *RW* applications. We use the *blktrace* tool [20] in order to track the logical block addresses (LBAs) and arrival times of I/O requests from VMs.

Results. Fig. 1 shows the change in the LBAs of arriving requests for the three guest VMs in the read-dominated and write-dominated experiments, respectively.

Our *first observation* is that the LBAs of requests from the same VMs can be grouped into *regions* which are bounded by the range of disk space occupied by a guest VM image. The ranges [1.2832×10^9-1.3057×10^9], [1.3086×10^9-1.33×10^9] and [1.3594×10^9-1.384×10^9] represent the regions of VM1, VM2 and VM3, respectively, for both read-dominated and write-dominated applications. This is because the hypervisor file system assigns contiguous disk blocks to the VM image in order to improve disk efficiency. Accordingly, the virtualized environment has regional spatial locality across VMs.

Our *second observation* is that looking at the disk accesses within each VM, LBAs and arrival times of requests divide the accessed disk region of each VM into several sub-regions over time. These sub-regions differ in their ranges and frequencies of access. The reason is that the file system of a VM also assigns contiguous virtual blocks to the applications without being aware of physical disk information, and accordingly all VM image formats (*e.g.*, RAW, Qcow [21], Fvd [22]) try to map LBAs of these virtual blocks into contiguous offsets in the VM image. As these blocks belong to the same VM (the hypervisor treats a VM as a process), the file system of the hypervisor accordingly allocates these blocks together in the physical disk. Furthermore, because different applications access their own data sub-regions, requests from the same application have sub-regional spatial locality, especially from applications with sequential access.

As shown in Fig. 1, in contrast to VM1, both VM2 and VM3 do not have a clear sub-regional spatial locality (*i.e.*, the number and the range of observed sub-regions are higher), because of the random access of data sets exhibited by random applications. For example, for read-dominated applications (Fig. 1(a)) the requests from VM1 are mainly concentrated into two sub-regions with small ranges: around 1.29×10^9 with a maximum distance of $\mp 0.005 \times 10^9$, and around 1.305×10^9 with a maximum distance of $\mp 0.005 \times 10^9$. Meanwhile, the distribution of the requests from VM3 is concentrated in two sub-regions: one with small range, around 1.36×10^9 with a maximum distance of $\mp 0.005 \times 10^9$, and the second with larger range, around 1.379×10^9 with a maximum distance of $\mp 0.05 \times 10^9$. For write-dominated applications, the distribution of sub-regions and their range are more diverse. Moreover, the access frequencies of these sub-regions are often changing. Fig. 1(b) shows that the distribution of sub-regional spatial locality of VM2 during 0 to 30s is different to that after 30s. This is due to the transparency feature of virtualization which leads to a non-deterministic allocation of write data set, in addition to the impact of the VM caches.

Fig. 1 also shows that some data sub-regions are accessed with low frequency at the beginning of each region, thus not having spatial locality. This can be explained due to: (1) the writing of log files — the journal process running within a VM periodically writes the logs of the file system and the operating system, which are normally at the start of the VM image; and (2) the VM process accesses the *inode* of the VM image when updating the metadata of the image, such as the access time, the size of image, and the changes of file content. The position of the *inode* is generally at the start of the VM image. Moreover, the applications with random access introduce high I/O operations of the journal process and system process in contrast to sequential access applications, such as in VM3 in Fig. 1(a). Furthermore, for write applications, the log writing and the metadata modifications are frequent; we observe a growing number of sub-regions with low access frequencies.

In summary, with respect to the case when multiple applications run within a VM, *we observe the special spatial locality in virtualized environments: regional spatial locality across VMs and sub-regional spatial locality between requests from the same VM*. Moreover, the sub-regional spatial locality of a VM is obvious when applications with sequential access run within a VM. The lack of coordination between VMs and the hypervisor (virtualization transparency) results in inefficient exploitation of spatial locality, which leads to low disk utilization (*i.e.*, increases disk head seeks (movement) between sub-regions); traditional non-work-conserving scheduling, including the CFQ [23] scheduler and Anticipatory scheduler (AS) [24], are effective at preserving the spatial locality exhibited by individual processes, but treat a VM process simply as a general user process and do not recognize the sub-regional access locality of a VM, resulting in the low spatial locality (*e.g.*, as shown in section VI, the seek distance at zero is only 46% under CFQ in the read-dominated applications). The aim of our work is to take advantage of the special spatial locality of VMs to improve physical disk

TABLE II: **Variables of the *vNavigator* model**

Var.	Description	Var.	Description
VM	a guest VM	$\triangle n$	the number of requests accessing Z_j during interval $[T_r, T]$
P(R)	the LBA of a request *R*	U_i	the i^{th} sub-region unit with sub-regional spatial locality of a VM in current prediction window
T(R)	the arrival time of a request *R*	λ	a decay factor
B	the offset in the disk	$W(R_j, T)$	the contribution of R_j to spatial locality at prediction window *T*
Z_j	the j^{th} equal-sized zone whose size is B	$D(Z_j, T)$	the temporal access-density of Z_j at prediction window *T*
R_j	a request accessing Z_j	$\delta(VM, T)$	the temporal access-density threshold of a VM at prediction window *T*
R_j^m	the m^th request accessing Z_j	$ZT(Z_j, T(R_j))$	the average access time interval of Z_j when a request R_j access Z_j
T	the current prediction window	$SR(U_i)$	the range of U_i
T_r	a prediction window when R_j arrives	$ST(U_i)$	the future access interval of U_i

efficiency and thus enhance the performance of applications in the virtualized environment.

IV. Prediction Model of the Locality and Regularity of Disk Accesses

As discussed in section III, the regional spatial locality can be easily observed according to the VM image size. The sub-regional spatial locality, however, cannot be observed in the virtualized environment due to the transparency feature of virtualization. Consequently, the disk scheduler in the hypervisor cannot efficiently exploit the special spatial locality. To this end, we design an intelligent prediction model, named *vNavigator*, to predict the regularity of the sub-regional spatial locality (*i.e.*, the distribution of sub-regions with spatial locality within a VM and access intervals of these sub-regions). Hence the *vNavigator* model helps to guide the scheduling of I/O requests in the hypervisor to efficiently exploit the sub-regional spatial locality within VMs. Based on the trace of sub-regions in section III, the *vNavigator* model prediction faces three challenges: (1) the distribution of sub-regions with spatial locality is changing with time, and varies based on the access patterns of applications; (2) requests from background processes (named discrete requests) in a VM interfere with the prediction of future requests with sub-regional spatial locality; and (3) different sub-regions with spatial locality may have different access regularity. For clarity, we first list the variables used in the *vNavigator* model, which are shown in Table II.

The *vNavigator* model uses a *temporal access-density clustering* (*TAC*) algorithm to analyze data access within a VM image file in the past to predict the sub-regional access locality of the VM in the near future. As shown in Fig. 2, considering the frequent changes in both the range and regularity of the sub-regions with spatial locality, the *TAC* algorithm divides the disk space into a series of equal-sized *zones* (denoted by $Z = \{Z_1, Z_2, ..., Z_n, n = \frac{length(disk)}{B}\}$). To ease capturing and predicting of the change of the spatial locality in time, the *TAC* algorithm also divides the allocated serving time of a VM into several equal time windows (*prediction window*). By tracking the arrival time and the LBAs of VM's requests, the *TAC* algorithm quantizes the zones' access frequencies for each VM in previous windows to estimate the spatial locality of zones in the current prediction window.

Zones with possible spatial locality draw out the distribution of the sub-regions with relative spatial locality of a VM: given that zones with similar spatial locality may have different access frequencies and access intervals, the *TAC* algorithm therefore groups, within the same prediction window, neighboring zones into larger units (sub-region units). The distribution of sub-regions with spatial locality can be represented as $U = \{U_1, U_2, ..., U_i, U_i \subseteq VM\}$, where U_i represents the i^{th} sub-region unit with sub-regional spatial locality. A sub-region unit may consist of one or two neighboring zones (further details are provided in the following subsections). Accordingly, the *vNavigator* model actually predicts the range of U_i and the arrival time interval of future requests accessing U_i.

A. Quantization of Access Frequency

The temporal access-density of a zone is the sum of the contributions of historical requests to the future possibility of spatial locality of the zone in the current prediction window. Considering the frequent change of the sub-regional spatial locality, the impact of recently arrived requests is greater than that of older requests when predicting sub-regional spatial locality within the current prediction window. The contribution of historical requests should therefore decline with time. Accordingly, we introduce an access weight of the request to represent the contribution of a request in the current prediction window. The access weight uses a decay factor (λ) to quantize the relationship between the contribution of requests and time.
Definition 1: The access weight of a request (R_j) to the sub-regional spatial locality of accessing a zone (Z_j) in the current prediction window is:

$$W(R_j, T) = \lambda^{-(T-T_r)} \tag{1}$$

where $\lambda > 1$ and R_j accesses Z_j in T_r.

According to (1), the access weight of a historical request declines with time. For instance, the access weight is 1 at the beginning, then decays toward zero with passing time. Thus the temporal access-density of the zone is defined as follows:
Definition 2: The temporal access-density of a zone (Z_j) is the sum of the access weights of the requests accessing this zone in the current prediction window:

$$D(Z_j, T) = \sum_{P(R) \in Z_j} W(R, T) \tag{2}$$

where R represents all requests accessing Z_j until T.

According to (1) and (2), and to simplify the computational cost of the temporal access-density of a zone, we obtain the following *Lemma 1*.
Lemma 1: Given that $\triangle n$ is the number of arrived requests accessing Z_j between T_r and T ($T_r < T$), $D(Z_j, T)$ is given by:

$$D(Z_j, T) = \lambda^{-(T-T_r)} D(Z_j, T_r) + \triangle n. \tag{3}$$

Proof: n is the number of requests accessing Z_j until T, and

R_j^k represents the k^{th} request to access Z_j.

$$\begin{aligned} D(Z_j,T) &= \sum_{k=1}^{n+\Delta n} W(R_j^k,T) = \sum_{k=1}^{n} W(R_j^k,T) + \sum_{k=n+1}^{n+\Delta n} W(R_j^k,T) \\ &= \lambda^{-(T-T_r)} \sum_{k=1}^{n} W(R_j^k,T_r) + \sum_{k=n+1}^{n+\Delta n} \lambda^{T-T} \\ &= \lambda^{-(T-T_r)} D(Z_j,T_r) + \triangle n \end{aligned}$$

According to *Lemma 1*, the temporal access-density of a zone consists of the number of newly arrived requests during the current prediction window and the decay of the temporal access-density of the previous prediction windows. Hence the temporal access-density captures the impact of the zone's access: requests to access a zone at different times have different effects on the future probability of the spatial locality of the zone.

B. Explore Sub-regional Spatial Locality

Depending on the temporal access-densities of the zones, we discuss the sub-regional spatial locality of zones in the current prediction window. In a guest VM, the requests from background processes access some disk zones periodically. These zones do not have spatial locality, although their temporal access-densities are larger than zero. Therefore, the *TAC* algorithm introduces a temporal access-density threshold for each VM to distinguish zones with future spatial locality. Because of the frequent changes of the access frequencies of zones and interference of system processes and journal processes in a VM, the temporal access-density threshold of a VM should meet two conditions: (1) changing over time (*i.e.*, when updating the temporal access-densities of zones); and (2) never dropping suddenly when increasing the number of zones with low temporal access-densities. Thus the temporal access-density threshold of the VM in the current prediction window can be stated as the mean of the current accessed zones' temporal access-densities that are larger than 1. By excluding zones whose temporal access-densities are lower than 1, we reduce the impacts of the zones that were accessed a long time ago, and therefore avoid any sudden drop of the threshold.

Definition 3: The temporal access-density threshold of a VM (*VM*) in the current prediction window is:

$$\delta(VM,T) = \sum_{y=1}^{N(T)} D(Z_y,T)/N(T) \tag{4}$$

where $D(Z_y, T) \geq 1$ and $N(T)$ is the number of zones whose temporal access-densities are greater than 1 in T.

Based on the temporal access-density threshold of the VM in the current prediction window, we explore the possibility of sub-regional spatial locality of VM. When $D(Z_j, T)$ is larger than $\delta(VM, T)$, the access of data in Z_j has sub-regional spatial locality in the next prediction window. Besides, the range of the sub-region with spatial locality may include zones with temporal access-densities more than $\delta(VM, T)$ and with temporal access-densities lower than $\delta(VM, T)$. For example, as shown in Fig. 2, in T_2, $D(Z_j, T_2)$ is larger than $\delta(VM, T_2)$ and $D(Z_{j+1}, T_2)$ is smaller than $\delta(VM, T_2)$, but Z_{j+1} also has sub-regional spatial locality in T_3. Therefore, when $D(Z_j, T)$ is larger than $\delta(VM, T)$, but the temporal access-density of Z_{j+1} is smaller than $\delta(VM, T)$, the *TAC* algorithm

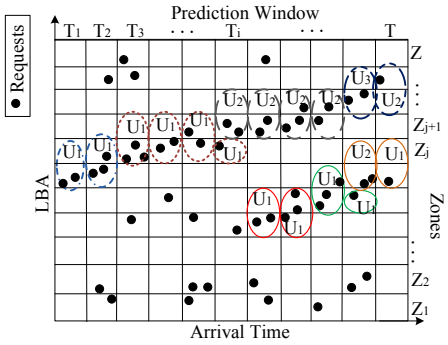


Fig. 2: ***vNavigator* model:** sub-region unit distribution

considers that both Z_j and Z_{j+1}[2] have sub-regional spatial locality of access in the next prediction window. Accordingly, the current distribution of the sub-regions with spatial locality of VM consists of zones with higher temporal access-densities compared to the current temporal access-density threshold, and their neighbors with temporal access-densities lower than the current temporal access-density threshold.

C. Access Regularity of Sub-regional Spatial Locality

Given that different sub-regions have different access regularity and different zones may have different access regularity, we introduce a sub-region unit which comprises one or two neighboring zones with the same access regularity and relative spatial locality. Therefore, the range of a sub-region unit is defined as follows, in accordance with the above subsection on exploring sub-regional spatial locality:

Definition 4: The range of a sub-region unit in the current distribution of sub-regions with spatial locality of the VM is:

$$SR(U_i) = \begin{cases} (Z_j, Z_{j+1}); \\ \quad D(Z_j,T) \geq \delta(VM,T), D(Z_{j+1},T) < \delta(VM,T) \\ Z_j; \\ \quad D(Z_j,T) \geq \delta(VM,T), D(Z_{j+1},T) \geq \delta(VM,T) \end{cases} \tag{5}$$

where Z_{j+1} belongs to other sub-region units with relative sub-regional spatial locality when $D(Z_{j+1}, T) \geq \delta(VM, T)$.

According to *Definition 4*, the sub-region unit with spatial locality includes one zone whose temporal access-density is more than the temporal access-density threshold of the VM, and its neighbor with temporal access-density lower than this threshold. This allows us to gather arrival intervals between historical requests with relative sub-regional spatial locality to predict the arrival time interval of future requests. To reduce the cost of the model and remove the interference of discrete requests, we analyze the average access time interval of the zone with temporal access-density more than the threshold, in order to estimate the access time interval of the corresponding sub-region unit with sub-regional spatial locality.

Definition 5: The future access interval of a sub-region unit with sub-regional spatial locality is:

$$ST(U_i) = ZT(Z_j, T(R_j^m)), D(Z_j,T) \geq \delta(VM,T) \tag{6}$$

where $ZT(Z_j, T(R_j^m))$ is the average access interval of Z_j when R_j^m access Z_j, and is denoted by:

$$ZT(Z_j,T(R_j^m)) = \frac{ZT(Z_j,T(R_j^{m-1})) * (m-1) + T(R_j^m) - T(R_j^{m-1})}{m} \tag{7}$$

[2] Z_{j-1} is not included due to the following reasons :(1) reducing disk backward seek and rotation overheads; and (2) avoiding overlapping zones.

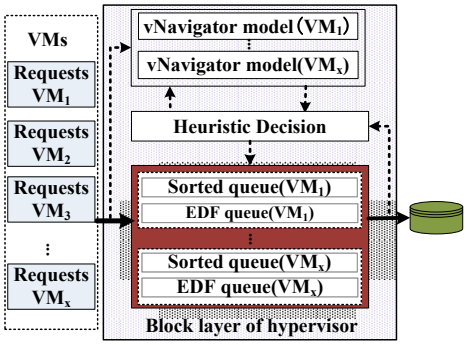


Fig. 3: **Architecture of *Pregather***

V. *Pregather* DISK SCHEDULING

The aim of our approach is to exploit the special spatial locality of VM accesses to improve the performance of *mixed* applications and disk I/O efficiency in virtualized environments while preserving *virtualization transparency*. We design and implement an adaptive non-work-conserving disk scheduling framework with a *spatial-locality-aware* (*SPLA*) heuristic algorithm in the hypervisor, named *Pregather*. When dispatching a pending request, *Pregather* decides whether or not to dispatch this request without starving other requests. The *SPLA* heuristic algorithm takes advantage of the regional spatial locality across VMs and the sub-regional spatial locality prediction from the *vNavigator* model, to guide *Pregather* to make the decision.

A. Spatial-locality-aware Heuristic Algorithm

After completing a request (*i.e.*, *LR*) from the current serving VM (*i.e.*, VM_x), *Pregather* introduces a coarse waiting time or a fine waiting time to exploit the special spatial locality (shown in Algorithm 1). If the hypervisor does not have any pending requests from VM_x, and the average seek distance of VM_x (*i.e.*, $AvgD(VM_x)$) is smaller than the distance between the LBA of the pending request from a close neighbor VM of VM_x and the LBA of *LR*(*i.e*, *P(LR)*), the coarse waiting time is set to the average arrival time interval of VM_x (*i.e.*, $AvgT(VM_x)$) to lower the disk seek across VMs. Otherwise, if the pending request queue of VM_x is not empty, *Pregather* consults the *vNavigator* model to predict whether the LBA of a future request is close to *LR* and predict the arrival time of the future request (*i.e.*, detect whether data regions around the current location of the disk head will have sub-regional spatial locality in the near future). If *P(LR)* is in the range of a sub-region unit with sub-regional locality (*i.e.*, $SR(U_i)$), then the fine waiting time is set to the average access interval of the sub-region unit (*i.e.*, $ST(U_i)$), to reduce the disk seek overhead within a VM.

Algorithm 1: Timer adjustment

Input:*P(LR)*: the LBA of the last completed request; $P(neighbor(VM_x).PR)$: the LBA of the pending request from a close neighbor VM_x; $Q(VM_x)$ the request queue of VM_x; U_i: the i^{th} sub-region unit of the *vNavigator* model
Output:*coarseTimer or fineTimer is set*
/*make the decision on setting a timer after completing a request*/
if $Q(\text{VM}_x) == Null \&\& (AvgD(\text{VM}_x) < |P(neighbor(\text{VM}_x).PR) - P(LR)|)$ **then**
  $coarseTimer = AvgT(VM_x) + currentTime$
else if $Q(\text{VM}_x)! = Null \&\& P(LR) \in SR(U_i)$ **then**
  $fineTimer = ST(U_i) + currentTime$
end

Algorithm 2: *SPLA* heuristic algorithm

Input:*PR*: the pending request; *P(LR)*; coarseTimer; fineTimer;U_i: the i^{th} sub-region unit corresponding to *P(LR)*;*newR*: a new request; $AvgT(VM_x)$
Output:*dispatch_req: a dispatching request*
/*make the decision when preparing to dispatch a request*/
begin
  $dispatch\_req = LP$
  if coarseTimer *is not over*$\&\&(PR \in \text{VM}_x \parallel AvgT(\text{VM}_x) \geq SeekTime(P(LR), P(PR)))$ **then**
    $dispatch\_req = PR$; turn off $coarseTimer$
  else if fineTimer *is not over* $\&\& P(PR) \in SR(U_i) \parallel ST(U_i) \geq SeekTime(P(LR), P(PR)))$ **then**
    $dispatch\_req = PR$; turn off $fineTimer$
  end
  /*Procedure invoked upon waiting for a future request*/
  while coarseTimer *or* fineTimer *is not over* && PR.deadline *is not over* $\&\&(dispatch\_req == LR)$ **do**
    /*on the arrival of the new request $newR$*/
    if $coarseTimer \&\& (newR \subseteq \text{VM}_x \parallel AvgT(\text{VM}_x) \geq SeekTime(P(newR), P(PR)))$ **then**
      $dispatch\_req = new\_req$; turn off $coarseTimer$
    end
    else if $(fine\_timer \&\& (P(newR) \in SR(U_i) \parallel ST(U_i) \geq SeekTime(P(newR), P(PR)))$ **then**
      $dispatch\_req = PR$; turn off $fineTimer$
    end
  end
  /*Procedure invoked upon expiration of coarse_timer or fine_timer or deadline of *PR*/
  if $(dispatch\_req == LR)$ **then**
    $dispatch\_req = PR$; turn off all timers
  end
end

When ready to dispatch a request, as shown in Algorithm 2, *Pregather* selects the closest pending request (*i.e.*, *PR*) to *LR* in the request queue of hypervisor (according to the LBAs). *Pregather* calculates the disk seek time between the LBAs of *LR* and *PR* (*SeekTime(P(LR), P(PR))*). Within the coarse waiting time, if *PR* meets one of the following two conditions: either the estimated seek time between the request and *P(LR)* is smaller than $AvgT(VM_x)$; or the request is from VM_x, the heuristic algorithm decides to dispatch *PR*. Otherwise, the heuristic algorithm decides to wait for a new request that meets one of these two conditions. On the other hand, within the fine waiting time, if *PR* also belongs to $SR(U_i)$, or the estimated seek time between the request and *P(LR)* is smaller than $ST(U_i)$, the heuristic algorithm then decides to dispatch *PR*. Otherwise, the heuristic algorithm decides to wait for a new request that satisfies one of these conditions. Over the coarse waiting time or the fine waiting time or the deadline of *PR*, *Pregather* dispatches *PR*. Therefore, *Pregather* ensures the long distance seek outweighs the cost of idle waiting and reduces the waiting without starving any requests.

B. Implementation

We implement a prototype of *Pregather* in the Xen-hosted platform. As shown in Fig. 3, *Pregather* consists of the *vNavigator* models corresponding to VMs, and the *heuristic decision* module that implements the *SPLA* heuristic algorithm, at the block layer of the hypervisor. To make full use of the special spatial locality without starving requests, *Pregather*, similar to CFQ, allocates two types of request queues for each

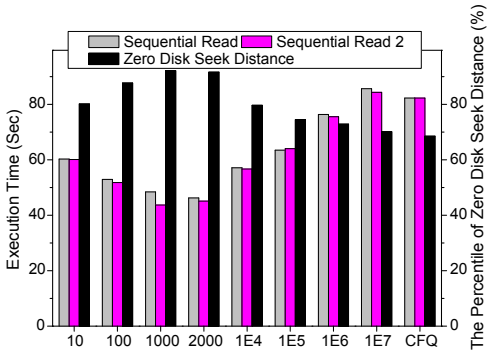


Fig. 4: **The execution time of two sequential read applications and the proportion of seek distance at zero point under *Pregather* with different offsets, CFQ**

VM: the sorted queue and the earlier deadline first (EDF) queue. The sorted queue contains requests in the order of LBAs of the requests, while the EDF queue contains requests in the order of deadline (the deadline value of *Pregather* is the same as of CFQ). Also, *Pregather* allocates each VM an equal serving time slice and serves VMs in a round robin fashion, to reduce I/O interference among VMs and batch as many requests from the same VM as possible. When a new request from a VM arrives in the hypervisor, *Pregather* assigns the request a deadline and queues the request in both the sort queue and EDF queue of the VM. Then *Pregather* computes the average arrival time and the average seek distance of the VM. Meanwhile, *Pregather* triggers the corresponding *vNavigator* model to analyze and update the range and arrival time interval of the sub-region units immediately.

The *heuristic decision* module guides *Pregather* to schedule I/O requests. After completing a request from the current serving VM, the *heuristic decision* module sets a coarse timer or a fine timer based on the average arrival time and average seek distance of the VM and the *vNavigator* model. When the hypervisor unplugs the block device and prepares to dispatch a request, *Pregather* selects a pending request (named *PR*) before triggering the *heuristic decision* module. If the request queues of the current serving VM are empty, *Pregather* selects a request from the request queue of a VM which has a close location (in the disk) to the current serving VM. Otherwise, *Pregather* first checks the deadline of the head request in the EDF queue of the current serving VM. If the deadline of the head request has expired, *Pregather* dispatches this request immediately and turns off the timer. Otherwise, *Pregather* selects a request next to the last completed request in the sorted queue of the VM as *PR*. Then, if the timer is active, the *heuristic decision* module decides to either dispatch *PR* or to wait for a future request, without exceeding the deadline of *PR*. Once the *heuristic decision* module schedules a future request, *Pregather* maintains the idle state of the disk head until the arrival of a suitable request. If the timer runs out or the deadline of *PR* expires, *Pregather* dispatches *PR*.

It is important to note that *Pregather* is not limited to Xen and can be implemented in other hypervisors (*e.g.*, KVM [25] and Linux-VServer [26]). The prediction model (introduced in section IV) uses hypervisor-independent parameters including the arrival times and LBAs of requests. Also, the *historical decision* module is implemented as a separate module at the block I/O layer of the hypervisor.

VI. Performance Evaluation

We run a suite of experiments evaluating our *Pregather* on the Xen-hosted platform using both synthetic benchmarks and a MapReduce application. The first set of experiments is to verify the *vNavigator* model, including evaluation of the sensitive parameter and verification of its accuracy. The second set of experiments is to evaluate the overall performance of *Pregather* and also to justify the efficiency of our heuristic algorithm in exploiting the special spatial locality when there are multiple VMs with different access patterns. The third set of experiments is to evaluate the overhead of memory caused by *Pregather*. The experimental setup is the same that described as in section III.

A. *Verification of* vNavigator *Model*

An accurate sub-regional spatial locality prediction is a key factor to achieve the high disk utilization with *Pregather*: *Pregather* uses the sub-regional spatial locality prediction of the *vNavigator* model to help schedule new requests. In this subsection we evaluate the accuracy of the prediction model. **The impact of B value.** As discussed in section IV, the offset *B* of the *vNavigator* model impacts the temporal access-densities of zones and the accuracy of clustering zones with sub-regional spatial locality, especially the interference of discrete requests. Thus it is very important to define B in our platform. Accordingly, we run two sequential read applications (16 threads sequentially read 128 files, whose total size is 2GB) in one VM, and capture their performance variation when changing B (we fix λ at 2, and the prediction window size to 20ms). The higher the accuracy of the prediction of the sub-regional spatial locality of the *vNavigator* model, the lower the frequency of disk seeking, because *Pregather* waits for a suitable future request with a minimal (zero) seek distance according to the *vNavigator* model. We use *blktrace* to trace the disk seek distance to discuss the prediction of the *vNavigator* model further.

Fig. 4 shows the execution time of sequential read applications and the proportion of disk seek distances at zero under different B values ($B = 1$ means that it is equal to the size of a sector, *i.e.*, 512bytes). Increasing B from 10 to 1000 reduces the execution time of applications and increases the proportion of minimal seek distance. The reason for this is that small B leads to the temporal access-densities of all zones tending toward the same value. The *vNavigator* model may treat a zone accessed by a background process as a zone with sub-regional spatial locality, thus introducing unnecessary waiting, especially when B is equal to the size of the request. On the other hand, when increasing B from 1000 to 2000, *Pregather* slightly decreases the average execution time of applications by 2%, but maintains the same proportion of minimal seek distance. The reason for this is that the number of zones at 1000 is more than at 2000, introducing more time overheads when updating all the temporal access-densities of zones (the 91.6% of seek distance at the zero point means that our model predicts the sub-regional spatial locality and regularity of access precisely when B is 1000 and 2000). However, when increasing B from 2000 to 10^7, the execution time of both applications increases from 46s to 85s, while the proportion of seek distance at zero drops from 91.6% to 71.2%. The performance of *Pregather* at 10^7 is the same as that of CFQ. This is because the size of the zone may cover the

TABLE III: **The Execution time of applications running on a VM under *Pregather*, CFQ and AS**

VM	Workload	*Pregather*	CFQ	AS
SRRR	Sequential Read	26.92s	44.0s	45.83s
VM	Random Read	33.74s	44.75s	45.94
RRRR	Random Read	101.17s	119.18s	113.18s
VM	Random Read	108.29s	119.18s	112.39s
SWSW	Sequential Write	27.27s	40.07s	39.66s
VM	Sequential Write	28.04s	42.72s	39.35s
SWRW	Sequential Write	15.04s	25.79s	26.07s
VM	Random Write	47.06s	53.64s	53.05s
RWRW	Random Write	57.56s	66.39s	66.63s
VM	Random Write	57.56s	65.81s	66.63s
SWRR	Sequential Write	58.76s	81.10s	85.92s
VM	Random Read	81.78s	89.50s	88.80s
SRRW	Sequential Read	33.03s	44.06s	50.03s
VM	Random write	49.98s	52.75s	56.23s

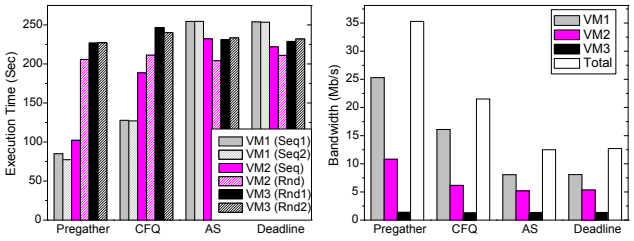
complete disk region taken by the VM image with increasing B. This in turn means that our model cannot detect the possible sub-regional spatial locality. Therefore, B is restricted to the range between the size of the request and the size of the VM image.

The ratio of successful waiting. Based on the above discussion, we set B to 2000. Then, we compare the performance of mixed applications with different access patterns running on a VM under *Pregather*, CFQ and AS.

We evaluate the performance of applications in seven different scenarios: Table III shows the execution time of the applications described in Table I. We find that the performance of applications under *Pregather* is better than under CFQ and AS. In particular, *Pregather* outperforms CFQ and AS by 33% and 31%, respectively, for the sequential write applications in the *SWSW* VM. From tracing the I/O behaviors of requests from the VM, *Pregather* achieves a 90.6% success ratio on waiting for a suitable future request, and therefore reduces the seek time. In contrast, CFQ and AS treat the VM as a general process and thus hardly wait for the future request, although they also employ the non-work-conserving scheduling. Moreover, the *vNavigator* model also captures the spatial locality of access between applications when sequential applications are mixed with random applications, and consequently improves the performance of the applications, as in *SRRR* VM, *SWRW* VM, *SWRR* VM, and *SRRW* VM. For instance, in *SRRR* VM, *Pregather* reduces the execution time of sequential read and random read by 38% and 22%, respectively, compared with CFQ and AS. In *SWRR* VM, *Pregather* outperforms CFQ for sequential write and random read applications by 27% and 9%, respectively. On the other hand, with *Pregather*, the improvement of random applications in *RWRW* VM and in *RRRR* VM is 12% and 10%, respectively. This is because the access pattern of random applications leads to the weak sub-regional spatial locality for a VM, as discussed in section III. Fortunately, the *vNavigator* model still prevents the interference of requests from background processes of a VM, and achieves an 80.4% success ratio on waiting for a suitable future request. From the above experimental results, we can conclude that the *vNavigator* model can predict sub-regional spatial locality and the access regularity of a VM when mixed applications run on a VM.

B. Spatial-Locality-Aware Disk Scheduling for Multiple VMs

We design experiments to measure the efficiency of *Pregather* with the *SPLA* heuristic algorithm for exploiting both



(a) Execution time (b) Average bandwidth

Fig. 5: **The performance of mixed applications when three VMs are running under *Pregather*, CFQ, AS and Deadline**

TABLE IV: **The distribution of seek distance when three VMs are running under *Pregather*, CFQ, AS and Deadline**

Distance	*Pregather*	CFQ	AS	Deadline
0	74.84%	46.04%	41.07%	12.49%
[-2.4E7,2.4E7]	97.44%	95.84%	95.95%	79.06%
[-1E9,-2.4E7]$\bigcup$ [2.4E7,1E9]	2.56%	4.16%	4.05%	20.94%

the regional and sub-regional spatial locality (*B* is set to 2000, λ is set to 2, the prediction window size is set to 20ms for the *vNavigator* model and the serving time slice for each VM is set to 200ms). Therefore, we compare the performance of multiple VMs under *Pregather* with that under three schedulers of the Xen-hosted platform (CFQ, AS and Deadline). Unlike CFQ and AS, Deadline is a work-conserving scheduler that dispatches neighboring requests without waiting for suitable future requests.

1) Disk I/O Efficiency for Multiple VMs with Different Access Patterns: We test *Pregather* with the read-dominated applications using the same experimental setup (as in section III). Fig. 5 shows the performance of applications under four schedulers in the read-dominated experiment. As shown in Fig. 5(a), *Pregather* reduces the execution time of both sequential applications in VM1 by 36%, 68% and 68% in comparison with CFQ, AS and Deadline, respectively. In VM2, the execution time of the sequential application with *Pregather* is reduced by 40% compared with CFQ. When *Pregather* is used for the random application in VM2, it outperforms the other schedulers by almost 2%. This is because the random request has the cost of high disk seeking, and this cost uses a considerable part of the serving time slice of VM2. Fig. 5(b) shows that *Pregather* improves the average bandwidth of VM2 by capturing sub-regional spatial locality between sequential and random applications in the hypervisor. In addition, because of the limitation of bandwidth of the physical server and the seeking overhead of the random applications, *Pregather* reduces the execution time of the random applications in VM3 by 6% compared with CFQ. Although the performance of Deadline and AS for VM3 is almost the same as that of *Pregather*, they sacrifice the bandwidth of VM1 and VM2. Fig. 5(b) also shows that the total bandwidth with *Pregather* is improved by 1.6x, 2.6x and 2.6x in comparison with CFQ, AS and Deadline, respectively.

To further illustrate disk I/O efficiency with *Pregather*, Table IV presents the distribution of the range of disk seeking under the four schedulers. The disk seek distance with *Pregather*, CFQ and AS is mainly concentrated in the range from -2.4×10^7 to 2.4×10^7, because the VM image occupies 2.4×10^7 sectors and the three schedulers exploit regional

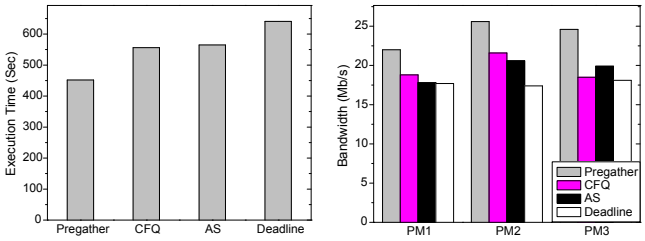
spatial locality across VMs to batch the requests from the same VMs. Besides, the proportion of disk seek distance at the zero point with *Pregather* is 77%, higher than the other schedulers, because the *SPLA* heuristic algorithm for each VM continuously dispatches requests with the sub-regional spatial locality by waiting for a future request successfully. When the current dispatched request and the last dispatched request are not from the same VM, or from system processes or journal processes of the hypervisor, the disk seek distance is distributed in the range between -1×10^9 and -2.4×10^7 or from 2.4×10^7 to 1×10^9.

2) Disk I/O efficiency for Data Intensive Applications: To discuss the performance of *Pregather* for data intensive applications, we evaluate *Pregather* with Hadoop [27] by comparing *Pregather* with the other three schedulers in two different scenarios: with and without a background VM that run different applications. In the first scenario, we deploy Hadoop (Hadoop-0.20.2) on a thirteen-node virtual cluster running on a three-nodes physical cluster, and then use the sort benchmark [3] to evaluate the efficiency of *Pregather*. In this setup, we deploy five VMs in physical machine 1 (PM1), and four VMs each in PM2 and PM3. Each PM has four data nodes on each of which eight maps run simultaneously. The data set is 6GB for the sort benchmark (64MB block size). Fig. 6 shows the execution time and average bandwidth of each physical machine under *Pregather*, CFQ, AS, and Deadline. As shown in Fig. 6(b), *Pregather* improves the total bandwidth of the physical cluster by 26%, 28%, and 38% compared with CFQ, AS, and Deadline, respectively. Because of the improvement of disk bandwidth, the execution time of *Pregather* outperforms CFQ by 18% and AS by 20%. The improvement of bandwidth under *Pregather* is due to the correct judgment of the *SPLA* heuristic algorithm. Accordingly, we analyze the distribution of the seek distance for each physical machine of the cluster in Fig. 7. In contrast with the non-work-conserving scheduling, Deadline sends as many adjacent requests as possible without waiting for future closed requests. Hence the seek overhead of Deadline is higher than that of the other three schedulers, especially for PM2 and PM3 where only data nodes run. Although the distribution of disk seek distance mainly concentrates the space range of the image with *Pregather*, CFQ and AS for exploiting the spatial locality among VMs, the proportion of seek distances at zero with *Pregather* is much higher than with CFQ and AS.

In practice, data intensive applications and other applications may be deployed on a virtualized server simultaneously. Given this situation, we run two different TPC-H [4] instances (q2 and q19) on a VM (named TPC-H VM) when Hadoop runs on other VMs in the second scenario. In this setup, we deploy Hadoop VMs and TPC-H VM on a physical node where six VMs run. In Hadoop, we generate a 2GB data set in four VMs each of which has two maps. Fig. 8 shows the execution time of the applications and the analysis of seek distance under *Pregather*, CFQ, AS and Deadline. We still observe that *Pregather* improves the performance of three applications in

[3]Sort benchmark: each mapper sorts the data locally, and each reducer merges the results from different mappers. The map input and the reduce output have the same data size as the input data. The sort benchmark has complicated disk access patterns due to mixing the sequential access with the random access.

[4]TPC-H is a decision support benchmark that processes business-oriented queries against a database system to examine large volumes of data.



(a) Execution time (b) Average bandwidth

Fig. 6: **The performance and throughput of three VMs when running sort benchmark under *Pregather*, CFQ, AS and Deadline**

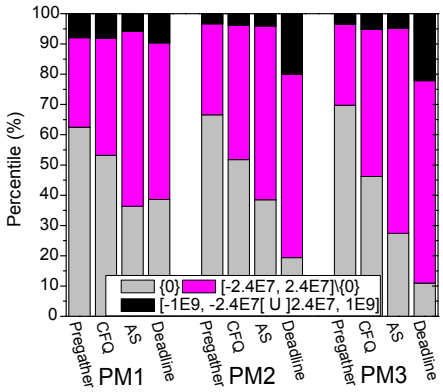
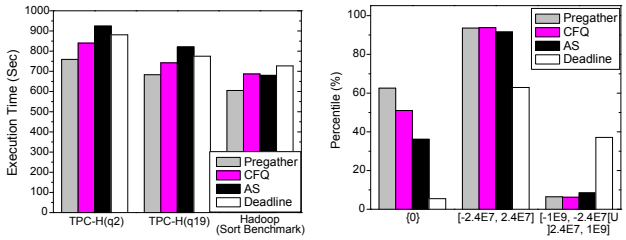


Fig. 7: **The distribution of seek distance of PMs when running sort benchmark under *Pregather*, CFQ, AS and Deadline**



(a) Execution time (b) The distribution of seek distance

Fig. 8: **The performance and distribution of seek distance when the sort benchmark and TPC-H are running together under *Pregather*, CFQ, AS and Deadline**

contrast to other schedulers. For instance, compared with CFQ, *Pregather* reduces the execution time of the q2, q19, and sort benchmarks by 10%, 8% and 12%, respectively, because the proportion of minimal seek distance with *Pregather* is 63%.

C. Memory Overhead

To predict the access regularity of sub-regions with future spatial locality, the *vNavigator* model stores the historical information about the accessed zone including the arrival time and LBA of the last request, the temporal access-density, the number of arrived requests and the total arrival time interval. Therefore, our model may cause extra memory overhead[5] in the hypervisor when guiding *Pregather* to exploit the sub-regional spatial locality of VMs. To evaluate the overhead of the *vNavigator* model, we monitor the memory utilization of the hypervisor in the read dominated experiment. As shown in

[5]The *vNavigator* model also introduces a small CPU overhead; however, as discussed earlier in Section VI-A, the cost of the CPU overhead is trivial and negligible with respect to the performance improvement.

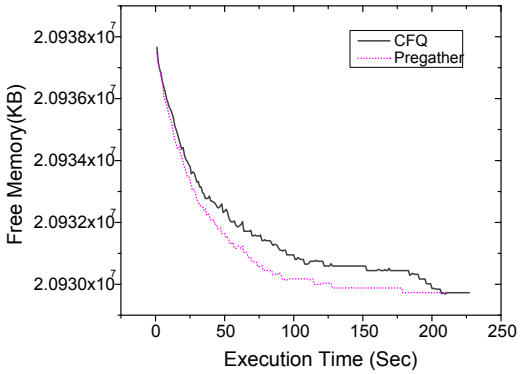


Fig. 9: **The change of free memory of hypervisor under *Pregather* and CFQ**

Fig. 9, we compare the change of free memory under *Pregather* and under CFQ, because CFQ also stores the historical access information of each VM. With the arrival of requests from VMs, the rate of decline of free memory under *Pregather* is faster than under CFQ. Compared with CFQ, *Pregather* uses no more than 916KB additional free memory. Therefore, in contrast to the total free memory, the memory overheads caused by *Pregather* can be neglected.

VII. Conclusion

In this study, we investigate the disk access patterns of VMs encapsulating mixed applications. Our studies reveal that disk accesses can be grouped into regions bounded by the virtual disks sizes, and within each region the disk data accesses are grouped into sub-regions, which correspond to the applications' access patterns. Ignoring the two types of spatial locality (*i.e.*, the regional and sub-regional spatial locality) when scheduling I/O requests causes performance degradation due to high seek delay and the rotation overhead. We address this issue by developing *Pregather*, a new spatial-locality-aware disk scheduler that makes full use of this special spatial locality for improving disk-intensive applications. *Pregather* embraces an intelligent prediction model, named *vNavigator*, to predict the distribution of sub-regions' accesses within each region, and the arrival times of future requests accessing these sub-regions. We perform extensive experiments that involve multiple simultaneous applications of both synthetic benchmarks and a MapReduce application on Xen-based platforms. Our experiments demonstrate the accuracy of our prediction model and indicate that *Pregather* results in the high spatial locality and a significant improvement in disk throughput. Regarding future work, to alleviate the lower spatial locality that occurs in the presence of disk fragments in virtualized environments (*i.e.*, the disk space taken by the VM image is not continuous, and therefore the size of the region may be larger than the image size), we intend to extend *Pregather* to enable an intelligent allocation of physical blocks.

Acknowledgements

The research is supported by National Science Foundation of China under grant No.61232008, National 863 Hi-Tech Research and Development Program under grant No.2013AA01A213, EU FP7 MONICA Project under grant No.295222, Chinese Universities Scientific Fund under grant No. 2013TS094, Guangzhou Science and Technology Program under grant 2012Y2-00040, and the ANR MapReduce grant (ANR-10-SEGI-001).

References

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

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[3] D. Le, H. Huang, and H. Wang, "Understanding performance implications of nested file systems in a virtualized environment," in *Proc. FAST'12*, 2012, pp. 8–8.

[4] S. Jones, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Antfarm: Tracking processes in a virtual machine environment," in *Proc. ATC'06*, 2006, pp. 1–14.

[5] S. Ibrahim, H. Jin, L. Lu, B. He, and S. Wu, "Adaptive disk I/O scheduling for mapreduce in virtualized environment," in *Proc. ICPP'11*, 2011, pp. 335–344.

[6] M. Kesavan, A. Gavrilovska, and K. Schwan, "On disk I/O scheduling in virtual machines," in *Proc. WIOV'10*, 2010, pp. 6–6.

[7] S. Seelam and P. Teller, "Virtual I/O scheduler: a scheduler of schedulers for performance virtualization," in *Proc. VEE'07*, 2007, pp. 105–115.

[8] X. Ling, H. Jin, S. Ibrahim, W. Cao, and S. Wu, "Efficient disk I/O scheduling with qos guarantee for xen-based hosting platforms," in *Proc. CCGrid'12*, 2012, pp. 81–89.

[9] Y. Xu and S. Jiang, "A scheduling framework that makes any disk schedulers non-work-conserving solely based on request characteristics," in *Proc. FAST'11*, 2011, pp. 9–9.

[10] D. Boutcher and A. Chandra, "Does virtualization make disk scheduling passé?" *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 20–24, Mar. 2010.

[11] Y. Hu, X. Long, and J. Zhang, "I/O behavior characterizing and predicting of virtualization workloads," *Journal of Computers*, vol. 7, no. 7, pp. 1712–1725, 2012.

[12] I. Ahmad, "Easy and efficient disk I/O workload characterization in vmware esx server," in *Proc. IISWC'07*, 2007, pp. 149–158.

[13] I. Ahmad, J. Anderson, A. Holler, R. Kambo, and V. Makhija, "An analysis of disk performance in VMware ESX server virtual machines," in *Proc. WWC'03*, 2003, pp. 65–76.

[14] A. Gulati, C. Kumar, and I. Ahmad, "Storage workload characterization and consolidation in virtualized environments," in *Proc. VPACT'09*, 2009.

[15] N. Tran and D. Reed, "Automatic arima time series modeling for adaptive I/O prefetching," *IEEE TPDS*, vol. 15, no. 4, pp. 362–377, 2004.

[16] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang, "Diskseen: exploiting disk layout and access history to enhance I/O prefetch," in *Proc. USENIX ATC'07*, 2007, pp. 1–14.

[17] Z. Li, Z. Chen, S. Srinivasan, and Y. Zhou, "C-miner: Mining block correlations in storage systems," in *Proc. FAST'04*, vol. 186, 2004.

[18] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.

[19] A. Kopytov, "Sysbench manual," http://sysbench.sourceforge.net/docs/.

[20] J. Axboe and A. D. Brunelle, "Blktrace User Guide," http://www.cse.unsw.edu.au/aaronc/iosched/doc/blktrace.html.

[21] The QCOW Image Format. http://people.gnome.org/ markmc/qcow-image-format-version-1.html.

[22] C. Tang, "FVD: a high-performance virtualmachine image format for cloud," in *Proc. USENIX ATC'11*, 2011, pp. 18–18.

[23] J. Axboe. Completely Fair Queuing (CFQ). http://en.wikipedia.org/wiki/CFQ, 2010.

[24] S. Iyer and P. Druschel, "Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O," *ACM SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 117–130, 2001.

[25] KVM. http://www.linux-kvm.org/page/mainpage.

[26] Linux VServer. http://linux-vserver.org/Documentation, 2010.

[27] Apache Hadoop Project. http://hadoop.apache.org.