2012 IEEE 11th International Symposium on Network Computing and Applications

A Pipelining Approach to Informed Prefetching in Distributed Multi-Level Storage Systems

Maen M. Al Assaf†, Mohammed I. Alghamdi⋆, Xunfei Jiang‡, Ji Zhang‡, and Xiao Qin‡

† King Abdullah II School for Information Technology, University of Jordan, Amman, Jordan
‡ Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849-5347
⋆Department of Computer Science, Al-Baha University, Al-Baha City, Kingdom of Saudi Arabia

***Abstract*—In this paper, we present an informed prefetching technique called IPODS that makes use of application-disclosed access patterns to prefetch hinted blocks in distributed multi-level storage systems. We develop a prefetching pipeline in IPODS, where an informed prefetching process is divided into a set of independent prefetching steps among multiple storage levels in a distributed system. In the IPODS system, while data blocks are prefetched from hard disks to memory buffers in remote storage servers, data blocks buffered in the servers are prefetched through networks to clients' local cache. We show that these two prefetching steps can be handled in a pipelining manner to improve I/O performance of distributed storage systems. Our IPODS technique differs itself from existing prefetching schemes in two ways. First, IPODS reduces applications' I/O stalls by keeping hinted data in clients' local caches and storage servers' fast buffers (e.g., solid state disks). Second, in a prefetching pipeline, multiple informed prefetching mechanisms semi-dependently coordinate to fetch blocks (1) from low-level (slow) to high-level (fast) storage devices in servers and (2) from high-level devices in servers to clients' local cache. The prefetching pipeline in IPODS judiciously hides network latencies in distributed storage systems, thereby reducing the overall I/O access time in distributed systems. Using a wide range of real-world I/O traces, our experiments show that IPODS can improve noticeably I/O performance of distributed storage systems.**

***Keywords*-Informed prefetching, pipelining, parallel storage systems, distributed multi-level storage system.**

I. Introduction

This paper reports the effectiveness of an informed prefetching technique - IPODS - that relies on application-disclosed access hints to prefetch blocks from remote servers to clients' local cache in distributed multi-level storage systems. Our IPODS technique incorporates a prefetching pipeline, that divides an informed prefetching process into a number of independent prefetching steps. In the pipeline, while data blocks are prefetched from low-level devices (e.g., hard disks) to up-level devices (e.g., memory buffers) in remote storage servers, data blocks buffered in the up-level devices of the servers are prefetched through networks to local cache at client sites.

Recent work (see, for example, [16][12]) has shown that prefetching techniques can solve the I/O bottleneck problem in large-scale computing systems. Both predictive prefetching [2] and informed prefetching [1] - two popular types of approaches - improve I/O performance by preloading data from disks into the main memory prior to data accesses. In this study, we focus on informed prefetching, in which prefetching decisions are made based on applications' future access hints. We investigate the performance impacts of informed prefetching on distributed multi-level storage systems.

We are particularly interested in informed prefetching, because evidence [1] shows that informed prefetching can bridge the performance gap between the CPU and I/O. In a study conducted by Patterson *et al.*, an informed prefetching algorithm - TIP - improves performance of I/O-intensive applications by applying cost-benefit analysis to allocate buffers for both prefetching and caching [1]. Although we pay attention to informed prefetching, the pipelining idea incorporated in IPODS can be applied to predictive prefetching in the realm of distributed multi-level storage systems. In our future study, we plan to propose a pipelining approach to predictive prefetching for multi-level storage systems in a distributed computing environment.

The following three factors motivated us to consider a pipelining approach to informed prefetching in distributed multi-level storage systems:

1) the growing needs of distributed multi-level storage systems,
2) the I/O access hints offered by applications, and
3) the possibility of a prefetching pipeline for distributed storage systems powered by multi-level storage devices.

When parallel disk subsystems are extended into multiple-level storage systems [14][8], a hierarchy of multiple storage devices increases data access latency if the data are residing in a lower level of the systems. To shorten long data transfer latency, popular data or future accessed data may be stored in the upper level of the storage systems.

In a distributed storage system, large disk access latency due to network and server delays can be hidden by informed prefetching. We develop an informed prefetching technique tailored for distributed multi-level storage systems, each of which consists of a group of multi-level storage servers.

We demonstrate that a pipeline mechanism can be used to efficiently prefetch data blocks from a low-level storage device to a up-level storage device before moving the data blocks to the clients.

Application disclosures of future I/O accesses can be used to boost I/O performance of distributed multi-level storage systems. We address three distinct issues related to I/O access hints provided by applications. First, how to use application hints to prefetch data among multiple storage levels (e.g., main memory, solid state disks, and hard disk drives) in remote storage servers. Second, how to use access hints to prefetch data from remote storage servers to local clients' cache. The centerpiece of our approach is a pipeline in which we split the informed prefetching process into a set of independent prefetching steps among the multiple storage levels. Third, how to coordinate the above two prefetching modules in a pipeline manner.

Existing informed prefetching algorithms rely on the assumption that parallel disks offer enough I/O bandwidth for prefetching without encountering I/O congestion. Under such an assumption, an informed prefetching mechanism can prefetch a large number of data blocks in parallel. Our preliminary results show that distributed storage systems may not have unlimited I/O bandwidth; this observation is especially true for storage systems where nodes are connected with slow networks. In this study, we address the slow network issue by proposing a prefetching pipeline to hide long latencies imposed by the networks.

This study has the following two major research contributions:

- To reduce I/O delays and application's elapsed time in distributed multi-level storage systems, we propose new informed prefetching approaches to coordinate multiple prefetching mechanisms in the form of a pipeline. The informed prefetching algorithm developed in this study is called IPODS. We show that with our prefetching pipeline in place, multiple prefetching operations can be processed in parallel by both upper-level and lower-level prefetching mechanisms.
- We developed a simulated distributed multiple-level storage system, in which the two prefetching algorithms are implemented. Simulation results show that our prefetching mechanism powered by a data fetching pipeline reduces applications' stall and execution times.

The rest of the paper explains and justifies a pipelined informed prefetching scheme in distributed multiple-level storage system. Section II reviews the related work. We outline the IPODS architecture in Section III. Section IV describes the design and implementation issues of the IPODS prefetching mechanism. Section V presents our experimental framework and results. Finally, Section VI provides conclusions and directions for future studies.

II. Related Work

Previous researchers have suggested that application-disclosed hints can be used by prefetching mechanisms to dramatically improve I/O performance. To our best knowledge, however, ours is the first study to focus on informed prefetching in distributed multi-level storage systems, the first to construct a pipeline to coordinate multiple prefetching mechanisms in a distributed multi-level storage system, and the first to offer a systematic performance evaluation of informed prefetching in distributed multi-level hybrid storage systems.

A. Multi-level storage systems

A multi-level storage system consists of a hierarchy of heterogeneous storage devices that differ in their hardware, speed, size, and other specifications [17]. Multilevel storage systems provide cost-effective solutions for large-scale data centers without significantly affecting I/O response times. The I/O performance of a multi-level storage system depends on data placement of the system. Ideally, a high-level storage device should store two types of data: (1) popular data that are frequently accessed and (2) data that are likely to be accessed in the not-too-distant future.

Typical storage devices in a modern multi-level storage system include main memory, solid state disks, hard disks, and magnetic tape subsystems (see, for example, [8]).

B. Informed Prefetching

A study conducted by Patterson *et al.* [1] [5] [4] [6] [10] inspired us to concentrate on informed prefetching issues. Patterson *et. al* suggested that informed prefetching algorithms - invoking storage parallelisms - can take advantage of application-disclosed I/O access hints to eliminate I/O stalls through aggressive prefetching [10][1][5]. Performance benefits of informed prefetching were confirmed by other studies undertaken by Huizinga *et al.* [3] and Chen *et al.* [11].

When it comes to disk arrays, parallel informed prefetching aims to leverage parallel I/O to improve prefetching performance. Parallel informed prefetching is made possible, because data blocks are striped across an array of disks [9][13]. Parallel informed prefetching eliminates I/O stalls by placing hinted data in the cache before the data are requested by applications. To improve cache usage for both prefetching and caching, Patterson *et al.* proposed a cost-benefit model, which assists their prefetching mechanism to balance cache/buffer space shared between the LRU (least-recently-used) cache and the prefetching buffer [1]. The model decides the benefit of using more buffers for prefetching and the cost of ejecting a LRU block or a prefetched block.

C. Prefetching in Distributed and Parallel Storage Systems

Apart from multiple-level storage systems, parallel and distributed storage systems can benefit from prefetching schemes. For example, Patterson *et al.* proposed and implemented an informed prefetching model in a distributed storage system where data are allocated in remote nodes [1]. Unlike multi-level storage systems, distributed storage systems have to face the challenge of reducing latencies of accessing remote storage nodes through networks. To address this challenge, Rochberg and Gibson extended a network file system by integrating an informed prefetching mechanism [7]. Rochberg and Gibson's approach hides disk latency while exposing storage parallelism. Their experimental results show that informed prefetching over network reduces application's execution time by anywhere from 17% to 69% [7].

Our informed prefetching strategies are quite different from the aforementioned approaches, because ours leverage a prefetching pipeline to efficiently migrate hinted data to upper-level storage devices, thereby significantly reducing data access times.

III. System Design

Compared with existing prefetching schemes, IPODS introduces a set of salient features: support application-disclosed I/O access hints, multiple informed prefetching mechanisms, and support a prefetching pipeline across multiple storage devices. Before presenting the IPODS implementation details, we first outline a high-level overview of IPODS's system design.

A. Hardware Architecture of IPODS

Figure 1 illustrates the hardware architecture of IPODS. The system consists of client nodes and distributed storage nodes. Each client node maintains a local buffer caches; applications running on client nodes retrieve data from remote storage nodes. Applications disclose their future access hints to client nodes, which prefetch data from remote storage nodes to local buffers. Each storage node builds with N-level storage devices. In this paper, we configure each storage node as a two-level storage device - the upper level is a solid state disk and the lower level is a hard drive. As we descend in the hierarchy, the disk read latency increases. A large data block are stripped and stored across multiple storage nodes. Data blocks are transferred among client and I/O storage nodes through networks.

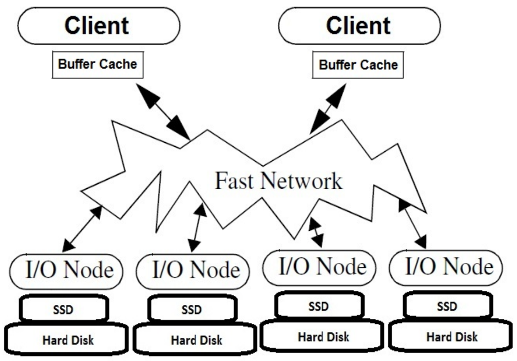


Figure 1: Distributed/Parallel Multi-level Storage System. (see also [15])

B. Assumptions

As a conservative assumption, hinted data blocks are initially allocated to HDDs thanks to the large capacity of the HDDs. It is noteworthy that I/O performance of multi-level storage systems can be improved if hinted blocks are initially placed in SSDs rather than HDDs.

In a multiple level storage system, a small portion of SSD space is reserved for retaining copies of the prefetched data. Instead of migrating data from HDDs to SSDs, IPODS keeps original copies at the HDD level while fetching duplicated copies to SSDs. In addition, IPODS adjusts SSD space reserved for prefetching.

IV. IPODS Algorithm

The informed prefetching algorithm - TIP (see [1] for details) - reduces applications' stalls and elapsed time. TIP makes use of applications' hints of future I/O accesses. In this section, we extend the informed prefetching technique by creating a pipeline in which we split the informed prefetching process into a set of independent prefetching steps among the multiple storage levels. In this section, we show that how multiple prefetching mechanisms can coordinate to prefetch hinted blocks in parallel. We call our new informed prefetching technique powered by a pipeline in a distributed system as IPODS.

Our empirical experiments indicate that parallel storage systems may have I/O congestion. Evidence shows that there is a maximum number of read requests being concurrently processed in a parallel storage system. In the event that an upper-level prefetching mechanism does not fully utilize available I/O bandwidth, unused I/O bandwidth can be allocated for lower-level prefetching mechanisms to bring hinted blocks from lower-level to upper-level storage in a pipeline manner. This observation suggests that the pipelining process largely depends on available I/O bandwidth for lower-level prefetchers.

This section presents an algorithm that guides us in implementing the IPODS mechanism for distributed multi-level storage systems.

A. Definitions

IPODS handles the informed prefetching process between SSDs and HDDs. When an application starts its execution, the TIP module assigns a number of buffers for prefetching (X_{cache}) based on the cost benefit model. This means that the informed prefetching module continues to issue a number of concurrent read requests that equals to (X_{cache}). The concurrent reads utilize either a part or all of the parallel storage system's bandwidth. Based on the non-utilized portion of the I/O bandwidth, IPODS assigns its pipelined prefetching depth.

Let Max_{BW} be the maximum number of read requests that may take place concurrently in the parallel storage system. The value of Max_{BW} depends on the number of nodes and the available aggregated I/O bandwidth. In the event that TIP is assigning X_{cache} buffers for prefetching, the difference between X_{cache} and Max_{BW} is the pipelined prefetching depth of IPODS. The pipelined prefetching depth is determined by Equation 1; this number represents the maximum space that needs to be reserved for pipelined prefetching in the uppermost level (i.e, SSDs). Pipelined prefetching depth does not consume the SSD storage space because it will reserve only a single data block space at each SSD of the array.

$$P_{depth} = Max_{BW} - X_{cache} \tag{1}$$

where;
P_{depth}: IPODS pipelined prefetching depth
Max_{BW}: Maximum bandwidth
X_{cache} : Number of prefetching buffers

The TIP module continues prefetching X_{cache} hinted data blocks over the network concurrently. At the same time, IPODS keeps fetching P_{depth} hinted data blocks from HDDs to the SSDs in a pipeline manner with TIP. IPODS consists of two algorithms determining what the first hinted block to be fetched from HDDs and the number of subsequent blocks to be fetched. Initially, most of TIP's prefetching requests are found in the lowest level (i.e., HDDs) until the hinted data blocks arrive in the uppermost ones (i.e., SSDs). At this point, TIP fetches hinted blocks from one of the two storage levels. As more I/O bandwidth becomes available for IPODS to fetch hinted blocks in a pipeline manner, more prefetching requests are handled in the uppermost level. Every time a data block is prefetched by TIP, it will be consumed from the SSD's pipelined prefetching buffer and a new pipelined prefetching request is initiated. As more I/O bandwidth becomes available for pipelining, application stalls and elapsed time will be reduced.

T_{cpu} + T_{hit} + T_{driver} represents the system's single time unit needed for an application to consume a prefetched data block. In addition, a distributed multi-level storage system consists of several storage nodes where each one has an HDD and an SSD. $T_{hdd-network-cache}$ is the time spent in retrieving a single data block over a network from a remote storage node's HDD. $T_{ss-network-cache}$ is the time needed to fetch a single data block over a network from a remote node's SSD. T_{hdd-ss} is the disk read latency from the HDD to SSD of a remote node. $T_{ss-network-cache}$ is less than $T_{hdd-network-cache}$, because the performance of SSD is higher than that of HDD.

B. The Pstart and Pnext Algorithms

IPODS pipelines the prefetching processes. When informed prefetching initiates its first prefetching requests, IPODS begins fetching requests from HDDs to SSDs. The position of the first block to be fetched from HDD by IPODS is calculated by the Pstart algorithm. The pipelined prefetching depth spans until a number of data blocks that are calculated by equation 1. When a data block is requested by TIP, the block will be consumed from the pipelining buffer in SSD after the block is fully fetched from SSD to the main memory of the client node. At this time, IPODS issues a new pipelined prefetching request for another hinted data block. The position of the next accessed data block is calculated by the Pnext algorithm. $T_{consume-ss}$ represents the time needed to consume a single pipelined preftching block from SSD (i.e. $T_{consume-ss} = T_{ss-network-cache}$).

The Pstart Algorithm: Determines the first block fetched from HDD to SSD

```
xcachecounter = 0
accesstime = - ($T_{cpu}$ + $T_{hit}$ + $T_{driver}$)
**for** blockcounter = 1 to $T_{hdd-ss}$ / ($T_{cpu}$ + $T_{hit}$ + $T_{driver}$)
**do**
  xcachecounter ++
  accesstime += ($T_{cpu}$ + $T_{hit}$ + $T_{driver}$)
  **if** accesstime $\geq$ $T_{hdd-ss}$ **then**
    Pstart = blockcounter
    return Pstart
  **end if**
  **if** xcachecounter = $X_{cache}$ **then**
    **if** $T_{stall-ss(Xcache)} > 0$ **then**
      accesstime += $T_{stall-ss(Xcache)}$
    **end if**
    **if** accesstime $\geq$ $T_{hdd-ss}$ **then**
      Pstart = blockcounter + 1
      return Pstart
    **end if**
    xcachecounter = 0
  **end if**
**end for**
```

Algorithm 1. returns the hinted block position for which IPODS begins pipelining. The first hinted block to be fetched from HDD depends on the T_{hdd-ss} and the stall values that may take place during the beginning of informed prefetching ($T_{stall-ss}(X_{cache})$), assuming that all data is already in the SSD. The algorithm calculates the hinted data block's position that will be accessed after enough time to have it read from HDD to SSD.

The Pnext Algorithm:

```
totaltime = $T_{hdd-ss}$ + $T_{consume-ss}$
xcachecounter = 0
accesstime = - ($T_{cpu}$ + $T_{hit}$ + $T_{driver}$)
**for** blockcounter = 1 to totaltime / ($T_{cpu}$ + $T_{hit}$ + $T_{driver}$)
**do**
  xcachecounter ++
  accesstime += ($T_{cpu}$ + $T_{hit}$ + $T_{driver}$)
  **if** accesstime $\geq$ totaltime **then**
    Pnext = Pstart + blockcounter - 1
    return Pnext
  **end if**
  **if** xcachecounter = $X_{cache}$ **then**
    **if** $T_{stall-ss(Xcache)} > 0$ **then**
      accesstime += $T_{stall-ss(Xcache)}$
    **end if**
    **if** accesstime $\geq$ totaltime **then**
      Pnext = Pstart + blockcounter
      return Pnext
    **end if**
    xcachecounter = 0
  **end if**
**end for**
```

Algorithm 2. returns the next hinted data block that IPODS should fetched from HDD to SDD when a previously pipelined data block is consumed from buffer in SDD. $T_{consume-ss}$ is the time to consume a block in SSD. The application stalls for at least $T_{stall-ss}$(X_{cache}). These stalls reduce the prefetching depth.

For $T_{stall-ss}(X_{cache})$ calculation, we used TIP's mathematical model for stalls calculation by applying $T_{ss-network-cache}$. (see [1]).

Figure 2 illustrates an example of IPODS working with the TIP module. Max_{BW} equals 5 and X_{cache} equals 2. IPODS uses the remaining 3 slots for pipelined prefetching. Due to the space limitation, we cannot illustrate a complete motivational example for IPODS. An example of informed prefetching without pipelining can be found in TIP' paper (see [1]).

C. The IPODS Algorithm

Algorithm 3: IPODS

```
Pstart = call pipelining start block
$Max_{BW}$ = Maximum number of concurrent reading requests
Pdepth = Pstart + ($Max_{BW}$ - $X_{cache}$) - 1
**while** informed prefetching **do**
  **if** bandwidth shortage **then**
    shrink the pipelining depth by 1
  **end if**
  **if** pipelined data block is altered **then**
    discard the pipelined data block
  **end if**
  **if** is the first prefetching **then**
    **for** block = Pstart To block = Pdepth **do**
      pipeline block to SSD
    **end for**
  **else if** SSD buffer is consumed **then**
    Pnext = call (Next to Prefetch) for the consumed block
    **if** the Pnext data block is not already in the SSD **then**
      pipeline Pnext to the SSD
    **end if**
  **end if**
**end while**
```

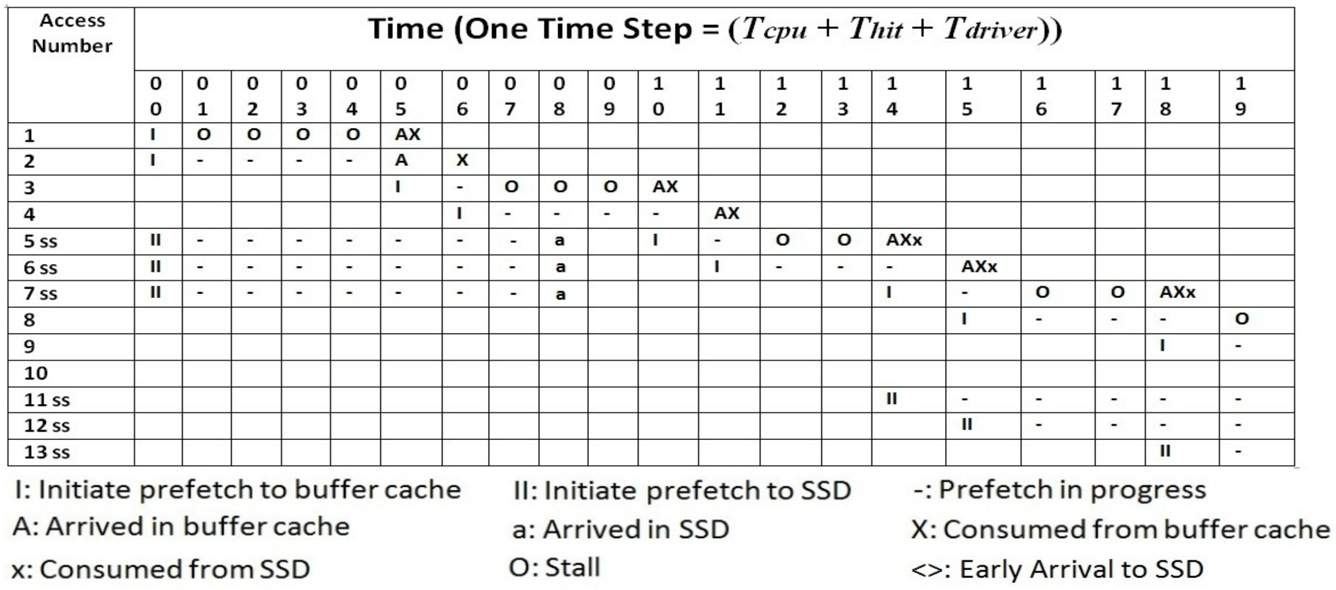
The IPODS algorithm calls the Pstart algorithm to determine the first hinted block to be fetched from HDD. Next IPODS calculates the pipelined prefetching depth (i.e., Pdepth), which is affected by the available I/O bandwidth. Then, IPODS initiates a prefetching request to fetch hinted blocks from HDD to SSD. Every time a cached block is consumed from SSD by the TIP module, IPODS starts prefetching the next hinted block from HDD to SSD. IPODS can handle any bandwidth shortage or data block modification.

V. IPODS Simulation and performance evaluation

We implement IPODS in a trace-driven simulator written in C++. The simulated distributed system consists of a client node offering local buffers and a set of remote storage nodes built with two-level storage devices (i.e., a SSD level and a HDD level). The client and all the storage nodes are connected by a network. Figure 1 shows the architecture of the simulated distributed storage system. The metric evaluated in the simulated two-level (i.e., SSD and HDD) distributed storage system (see Figure 1) is elapsed time with various Max_{BW} values. Each Max_{BW} case is evaluated by varying the numbers of prefetching buffers of the buffer cache. We compare an IPODS-enabled prefetching mechanism against the same system without deploying IPODS. Data blocks are initially placed in HDD. The IPODS mechanism coordinates two prefeching modules: the first one fetches hinted blocks from HDDs to SSDs, the second one uses TIP [1] to further fetch hinted blocks from SSDs to main memory over the network.

A. System Setup

In our simulation studies, we use two LASR traces: machine01 (LASR1) and machine06 (LASR2) [19][20] as guides for the application's I/O requests, which consist of 11686 and 51206 I/O read system calls, respectively. Without loss of generality, we assume that each I/O read system call requests an entire data block. As a conservative



Access Number	Time (One Time Step = ($T_{cpu} + T_{hit} + T_{driver}$))																			
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
1	I	O	O	O	O	AX														
2	I	-	-	-	-	A	X													
3						I	-	O	O	O	AX									
4							I	-	-	-	-	AX								
5 ss	II	-	-	-	-	-	-	-	a		I	-	O	O	AXx					
6 ss	II	-	-	-	-	-	-	-	a			I	-	-	-	AXx				
7 ss	II	-	-	-	-	-	-	-	a						I	-	O	O	AXx	
8																I	-	-	-	O
9																			I	-
10																				
11 ss															II	-	-	-	-	-
12 ss																II	-	-	-	-
13 ss																			II	-

I: Initiate prefetch to buffer cache   II: Initiate prefetch to SSD   -: Prefetch in progress
A: Arrived in buffer cache   a: Arrived in SSD   X: Consumed from buffer cache
x: Consumed from SSD   O: Stall   <>: Early Arrival to SSD

Figure 2: Average stalls when using IPODS and a fixed number of buffers for parallel prefetching in the buffer cache. The maximum number (Max_{BW}) of read requests is 5. Informed prefetching buffers = 2, and the rest 3 spaces of the bandwidth are used for pipelined prefetching. $T_{hdd-network-cache}$ = 5, T_{hdd-ss} = 8, $T_{ss-network-cache}$ = 4, and X_{cache} = 2. The first accesses stall for 5 time units. Before IPODS fetches hinted blocks from HDD to SSD, the application stalls for $T_{stall-hdd}$(X$_{cache}$) = $T_{hdd-network-cache}$ - 2(T_{cpu} + T_{hit} + T_{driver}) = 3 time units every 2 accesses. IPODS continues to fetch 2 hinted blocks each time from HDD. When a prefetched block is consumed from SSD, a new pipelined prefetching request is initiated by IPODS. When TIP prefetches a pipelined data block, the application stalls for $T_{stall-ss}$(X$_{cache}$) = $T_{ss-network-cache}$ - 2(T_{cpu} + T_{hit} + T_{driver}) = 2 time units every 2 accesses.

assumption, the average I/O arrival interval is T_{cpu} + T_{hit} + T_{driver}, representing the worst case. If the interval is larger than T_{cpu} + T_{hit} + T_{driver}, IPODS can achieve even better I/O improvement.

All the system parameters used in our simulator are validated by the testbed in our laboratory at Auburn. The following are storage devices tested in our laboratory:

- Memory: Samsung 3GB RAM Main Memory.
- HDD : Western Digital 500GB SATA 16 MB Cache WD5000AAKS.
- SSD: Intel 2Gb/s SATA SSD 80G sV 1A.
- Network Switch: [18] Network Dell Power Connect 2824.

Our preliminary results based on our storage devices indicate that an SSD is guaranteed to exhibit better performance than HDD when the data block size is 200 MB for a local system. According to our research lab testbed validation performed at Auburn, the system parameters used in our simulation study are: block size equals 200 MB, and (T$_{cpu}$ + T$_{hit}$ + T$_{driver}$) equals 0.037 seconds. Data is located in the HDDs. The average latency of reading a 200MB data block from a remote SSD and HDD to the buffer cache over a LAN network is 4.158 and 4.43 seconds, respectively. The time spent reading a 200MB block from the HDD to the SSD T$_{hdd-ss}$ equals 4.5 seconds. There is no write latency to the buffer cache; similar assumption can be found in [1].

Since both TIP and IPODS work concurrently, the maximum number of read requests to HDDs is set to Max$_{BW}$. According to TIP's prefetching horizon equation (see [1]),the prefetching horizon of reading the data from HDD is 120 and the maximum needed (Max$_{BW}$) equals to 354. We simulate IPODS by setting Max$_{BW}$ to 15 concurrent read requests. We choose 15 because in the TIP study, 15 disks were tested in the performance evaluation [1]. In the IPODS simulator, each storage node's disks can individually handle a single read request without I/O congestion.

The IPODS pipelined prefetching depth does not consume the SSD space. If TIP is assigning (X$_{cache}$) = 1, pipelined prefetching depth becomes 14. In this case, IPODS needs 2.8 GB.

B. Elapsed Time Improvement

The simulation results show that IPODS reduces the total elapsed time by about 6% when the TIP module is using very few buffers for prefetching (e.g., X$_{cache}$ is set to 1, 2, and 3). Figure 3 shows the total elapsed time for varying numbers of prefetching buffers when Max$_{BW}$ equals 15. As

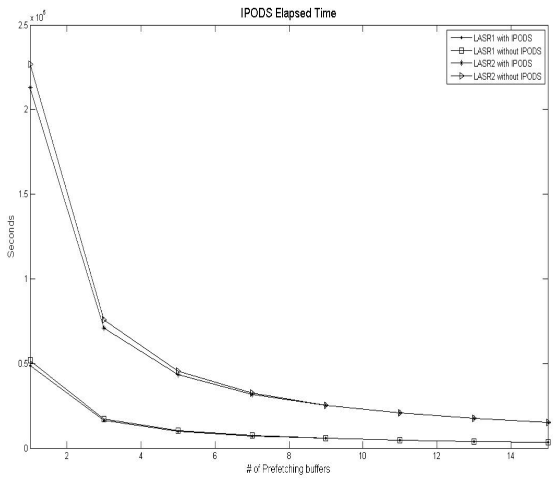


Figure 3: Total elapsed time when the number of prefetching buffers is varied from 1 to 15. Max$_{BW}$ = 15. IPODS reduces the elapsed time.

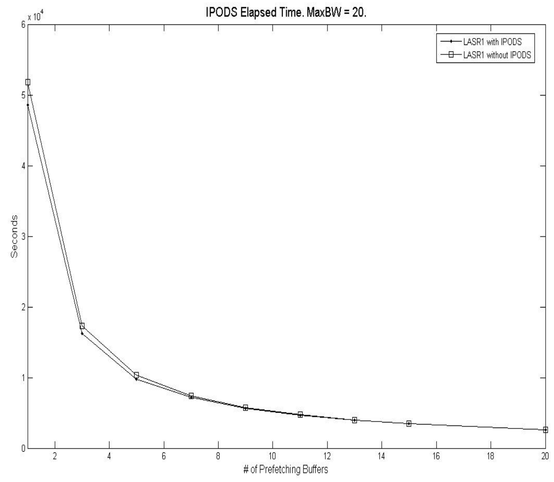


Figure 4: Total elapsed time when the X_{cache} value is increased from 1 to 20. Max_{BW} is set to 20.

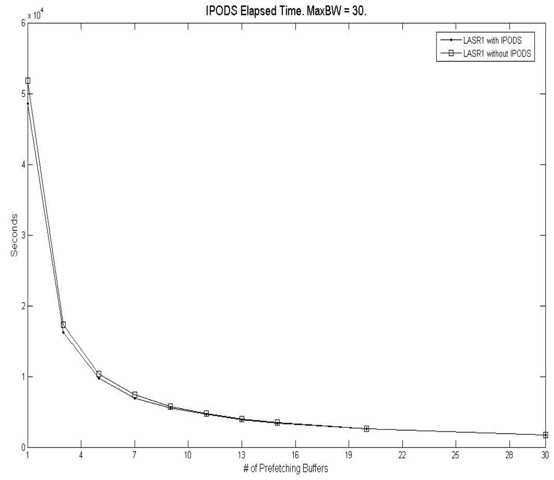


Figure 5: Total elapsed time when the X_{cache} value is increased from 1 to 30. Max_{BW} is set to 30.

the number of X$_{cache}$ increases, the performance difference between the IPODS and non-IPODS cases decreases because TIP becomes more efficient in reducing the application's stalls.

Importantly, the IPODS pipelined prefetching process shows its best performance improvement when TIP uses few X_{cache} buffers (e.g. X_{cache} = 1). The experimental results suggest that IPODS can enhance I/O performance of distributed and parallel storage systems where the TIP module has limited or no cache buffer to utilize. On the other hand, when X_{cache} is greater than or equal to 9 especially when (Max$_{BW}$) is limited, IPODS can not exhibit any performance improvement.

C. Increasing the Max_{BW} Value

Performance improvement offered by IPODS becomes more pronounced when I/O bandwidth of the parallel storage system increases. This is a result IPODS being able to issue more pipelined prefetching requests to HDDs and bring more data to the uppermost level.

When using Max$_{BW}$ that equals 15, we observed that IPODS's performance improvement is not as significant because of the limited I/O bandwidth. Figures 4, 5, 6, 7, 8, and 9 show application elapsed time when the X_{cache} value is increased from 1 to the prefetching horizon of reading the data from HDD (120) and Max_{BW} value is 20, 30, 50, 100, 200, and 354 respectively. The LASR1 trace is used in this simulation study. As we mentioned previously, the maximum needed (Max$_{BW}$) equals to 354.

As Max$_{BW}$ increases, IPODS becomes able to do more pipelining and to exhibit a significant performance improvement. It also can reduce the total elapsed time by approximately 6% with more X_{cache} cases.

VI. Conclusion & Future Work

In a distributed storage system, large disk access latencies due to network and server delays can be hidden by informed prefetching. In this paper, we showed that the pipelining approach (i.e., IPODS) to informed prefetching can improve the I/O performance of distributed multi-level storage systems. We illustrated how to use application hints to prefetch data from low-level devices to up-level devices in remote storage servers and from storage servers to local cache of clients. The centerpiece of our IPODS approach is a pipeline in which an informed prefetching process is divided

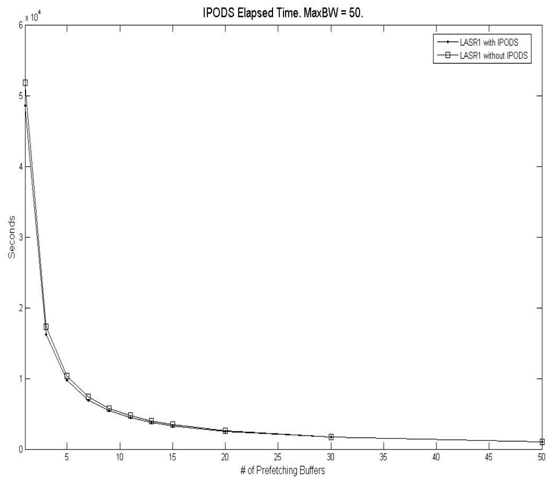


Figure 6: Total elapsed time when the X_{cache} value is increased from 1 to 50. Max_{BW} is set to 50.

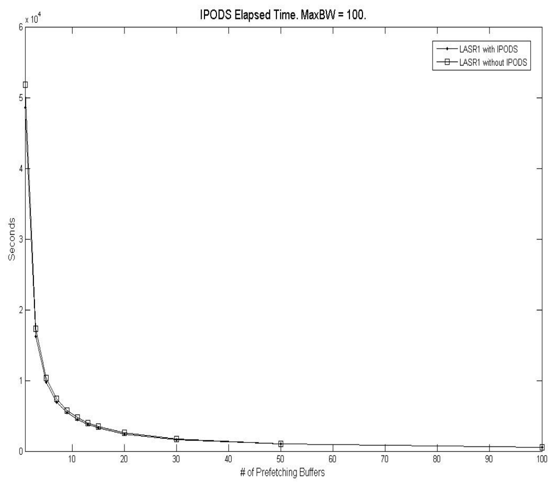


Figure 7: Total elapsed time when the X_{cache} value is increased from 1 to 100. Max_{BW} is set to 100.

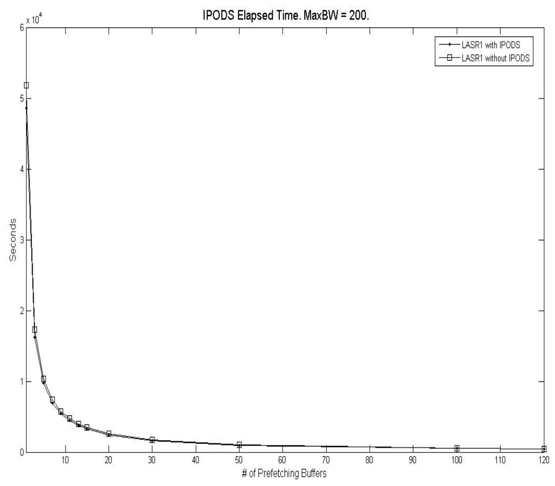


Figure 8: Total elapsed time when the X_{cache} value is increased from 1 to 120. Max_{BW} is set to 200.

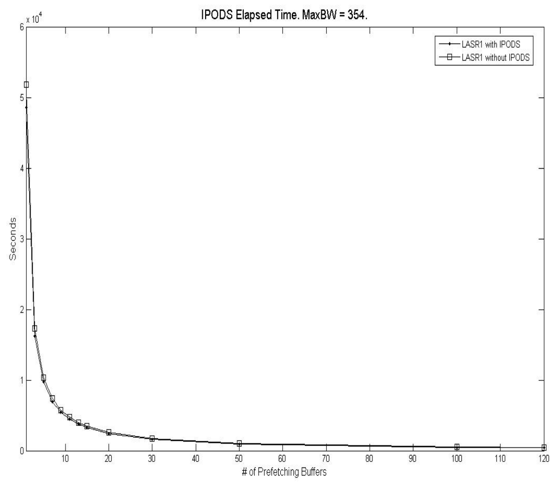


Figure 9: Total elapsed time when the X_{cache} value is increased from 1 to 120. Max_{BW} is set to 354.

into a number of concurrent prefetching modules. More specifically, while data blocks are prefetched from low-level devices to up-level devices in remote storage servers, data blocks buffered in the up-level devices of the servers can be prefetched through networks to local cache at client sites.

The implementation of IPODS contains three important elements related to application disclosures of future I/O accesses. First, an informed prefetching module uses application hints to prefetch data from low-level to up-level storage devices in storage servers. Second, an informed prefetching module prefetches data from storage servers to local cache at client sites. Third, a prefetching pipeline coordinates the above two informed prefetching modules, enabling the two prefetchers to perform in parallel.

We performed the experiments using a validated simulator driven by real-world I/O traces representing I/O-intensive applications. The experimental results show that the IPODS approach can be employed to efficiently prefetch data blocks from low-level storage devices in remote servers to local cache buffers in clients. IPODS noticeably reduces the elapsed times of applications running in distributed multi-level storage systems.

ACKNOWLEDGMENT

This research was supported by the U.S. National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0917137 (CSR), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS). Mohammed Alghamdi's research was supported by AL-Baha University.

REFERENCES

[1] R. Patterson, Hugo, G. Gibson, D. Stodolsky, and J. Zelenka: Informed prefetching and caching, In Proceedings of the 15th ACM Symposium on Operating System Principles, pages 79-95, CO, USA, 1995.

[2] J. Griffioen and R. Appleton: Reducing file system latency using a predictive approach, In Proceedings of the 1994 USENIX Annual Technical Conference, pages 197 207, Berkeley, CA, USA, 1994.

[3] M. Huizinga, D. and S. Desai: Implementation of informed prefetching and caching in linux, In Proceedings of the International Conference on Information Technology, pages 443 -448, Las Vegas, NV, USA, 2000.

[4] R. Hugo Patterson , Garth A. Gibson , M. Satyanarayanan: A status report on research in transparent informed prefetching, ACM SIGOPS Operating Systems Review, v.27 n.2, pages: 21-34, 1993.

[5] A.Tomkins, R. Hugo Patterson and G. Gibson: Informed multi-process prefetching and caching, In Proceedings of the 1997 ACM SIGMETRICS international conference on measurement and modeling of computer systems, pages 100 - 114, Seattle, WA , USA, 1997.

[6] R.H. Patterson, G.A. Gibson, M. Satyanarayanan: Using Transparent Informed Prefetching (TIP) to Reduce File Read Latency , In Proceedings of Conference on Mass Storage Systems and Technologies, Pages: 329-342, Greenbelt, MD, 1992.

[7] D. Rochberg, G.A. Gibson: Prefetching over a network: early experience with CTIP, *ACM SIGMETRICS Performance Evaluation Review*, v.25 n.3, Pages: 29-36, 1997.

[8] M. Nijim: Modelling Speculative Prefetching for Hybrid Storage Systems, In Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference on, pages 143 - 151, Macau, 2010.

[9] T. Kimbrel, P. Cao, E. Felten, A. Karlin, K. Li: Integrated Parallel Prefetching and Caching, Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages 262 - 263, PA,USA, 1996.

[10] R. Hugo Patterson, G. Gibson: Exposing I/O concurrency with informed prefetching, Proceedings of the third international conference on on Parallel and distributed information systems, pages 7 - 16, Austin, TX, USA, 1994.

[11] Y. Chen ; Byna, S. ; X. Sun ; Thakur, R. ; Gropp, W: Exploring Parallel I/O Concurrency with Speculative Prefetching, Proceedings of the 2008 37th International Conference on Parallel Processing, pages 422 - 429, Portland, OR, USA, 2008.

[12] Y. Chen ; Byna, S. ; X. Sun ; Thakur, R. ; Gropp, W: Hiding I/O Latency with Pre-execution Prefetching for Parallel Applications, Proceedings of the 2008 ACM/IEEE conference on Supercomputing, pages 1 - 10, Austin, TX, USA, 2008.

[13] Ganger, G.R. ; Worthington, B.L. ; Hou, R.Y. ; Patt, Y.N: Disk arrays: high-performance, high-reliability storage subsystems, Journal: Computer, issn: 0018-9162, volume 27, pages 30-36, doi: 10.1109/2.268882 Ann Arbor, MI, USA, 1994.

[14] Alexander Thomasian: Multi-level RAID for very large disk arrays, ACM SIGMETRICS Performance Evaluation Review, v.33 n.4, March 2006 [doi¿10.1145/1138085.1138091]

[15] T.Madhyastha; G. Gibson; C. Faloutsos: Informed prefetching of collective input/output requests, *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, Portland, Oregon, 1999.

[16] C.K. Yang, T. Mitra and T. Chiueh: A Decoupled Architecture for Application-Specific File Prefetching, Freenix Track of USENIX 2002 Annual Conference, 2002.

[17] T. Kaneko: Optimal Task Switching Policy for a Multilevel Storage System, IBM Journal of Research and Development, vol.18, no.4, pp.310-315, July 1974.

[18] DELL PowerConnect 2824 Switch, doi:http://www.dell.com/us/business/p/ powerconnect-2824/pd.

[19] Lasr trace machine01, doi:http://iotta.snia.org/traces/list/ Subtrace?parent=LASR+Traces.

[20] Lasr trace machine06, doi:http://iotta.snia.org/traces/list/ Subtrace?parent=LASR+Traces.