# Optimizing Scientific I/O Patterns using Advice Based Knowledge


Giuseppe Congiu*, Matthias Grawinkel†, Federico Padua†, James Morse*, Tim Süß†, André Brinkmann†

*Emerging Technology Group Seagate Technology Havant, United Kingdom. Email: {giuseppe.congiu, james.s.morse}@seagate.com
†Zentrum für Datenverarbeitung Johannes Gutenberg-University Mainz, Germany. Email: {grawinkel, padua, t.suess, brinkman}@uni-mainz.de


## Introduction

The performance gap between processing and I/O represents a serious scalability limitation for scientific applications running on high-end computing clusters. Parallel file systems often provide mechanisms that allow programmers to disclose their I/O pattern knowledge to the lower layers of the I/O stack through a hints API. This information can be used by the file system to boost the application performance, for example, through data prefetching.

Unfortunately, programmers rarely make use of these features, missing the opportunity to exploit the full potential of the storage system. Additionally, scientific applications frequently perform small non-contiguous accesses to files using the POSIX I/O interface. This makes it impossible for them to take advantage of automatic optimizations, such as collective I/O or data-sieving enabled by the MPI I/O middleware [4] – [6]. As a result these applications perform poorly. More significantly they can negatively impact the whole storage system's efficiency.

We propose and evaluate a novel advice infrastructure able to optimize file access patterns at runtime through data prefetching using these hints mechanisms. The advice infrastructure communicates file I/O pattern information to the file system on behalf of running applications asynchronously, with very low overhead, and without any modification of the original application.

We demonstrate that our approach is effective in improving the I/O bandwidth, reducing the number of I/O requests and reducing the execution time of a 'ROOT' [1] based application.

Additionally, we propose and evaluate a modification to the Linux kernel that makes it possible for Lustre and other networked file systems to participate in activity triggered by the posix_fadvise system call, thus allowing it to take advantage of our advice infrastructure benefits.

## Background on Guided I/O Interfaces

The Linux kernel provides users with the capability to communicate access pattern information to the local file system through the posix_fadvise [2] system call:

```
int posix_fadvise(int fd, off_t offset, off_t length, int advice)
```

The file system can use this information to improve page cache efficiency, for example, by prefetching (or releasing) data that will (or will not) be required soon in the future or by disabling read-ahead in the case of random read patterns. Table 1 summarizes all the advice accepted by the system call.

| Advice | Description |
| --- | --- |
| POSIX_FADV_SEQUENTIAL | file access pattern is sequential |
| POSIX_FADV_RANDOM | file access pattern is random |
| POSIX_FADV_NORMAL | reset file access pattern to normal |
| POSIX_FADV_WILLNEED | a file region will be needed |
| POSIX_FADV_DONTNEED | a file region will not be needed |
| POSIX_FADV_NOREUSE | file is read once (not implemented) |

**Table 1**: *Values for **advice** in the posix_fadvise() system call*

The General Parallel File System (GPFS) compensates for the lack of POSIX advice support through a hints API that users can access by linking their programs against a service library. Hints are passed to GPFS through the gpfs_fcntl [3] function and can be used to guide prefetching of file blocks in the page pool (GPFS cache memory):

```
int gpfs_fcntl(int fileDesc, void* fcntlArgP)
```

| Hint data structure | Description |
| --- | --- |
| gpfsAccessRange_t | defines a region of the file that needs to be accessed |
| gpfsFreeRange_t | defines a region of the file that needs to be released |
| gpfsMultipleAccessRange_t | defines multiple regions of the file that needs to be accessed |
| gpfsClearFileCache_t | releases all the page pool buffers held by a certain file |

**Table 2**: *Data structures provided by GPFS to describe different hints*

Table 2 summarizes the available hints and corresponding data structures to be passed to GPFS as `fcntlArgP` argument in the previous routine.

## Proposed Solution

The proposed advice infrastructure communicates file I/O pattern information to the file system on behalf of running applications using a dedicated process that we call *Advice Manager*. Processes access their files using an *Interposing I/O Library* that transparently forwards intercepted requests to the local *Advice Manager*. This uses `posix_fadvise` and `gpfs_fcntl` to prefetch (or release) data into (or from) the client's file system data cache (Figure 1).

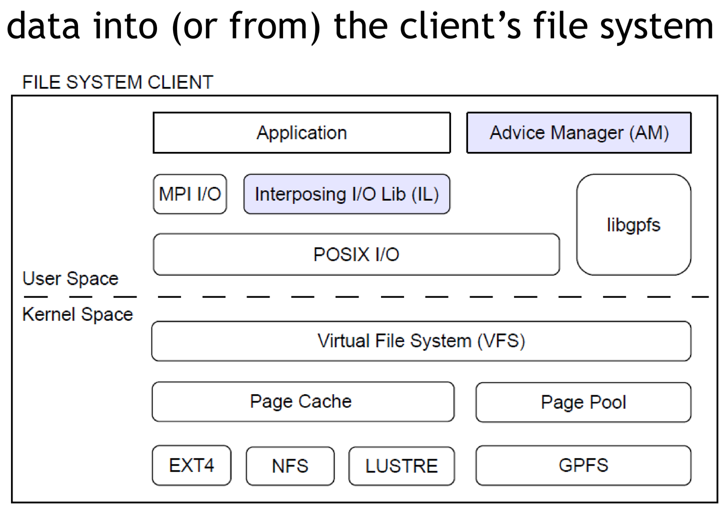


**Figure 1**: *I/O software stack of the advice infrastructure.*

The *Interposing I/O Library* controls for which files advice or hints should be given, while the *Advice Manager* controls how much data to prefetch (or release) from each file. Monitored file paths and prefetching information are contained in a configuration file that can be generated either manually or automatically once the I/O behaviour of the target application is known. The configuration file mechanism allows us to decouple the specific hints API provided by the back-end file system from the generic interface exposed to the final user thus making our infrastructure portable.

Figure 2 and Figure 3 show, respectively, the detailed architecture of the *Advice Manager* (AM) module and the prefetching mechanism used in the *Advisor Thread* (AT).

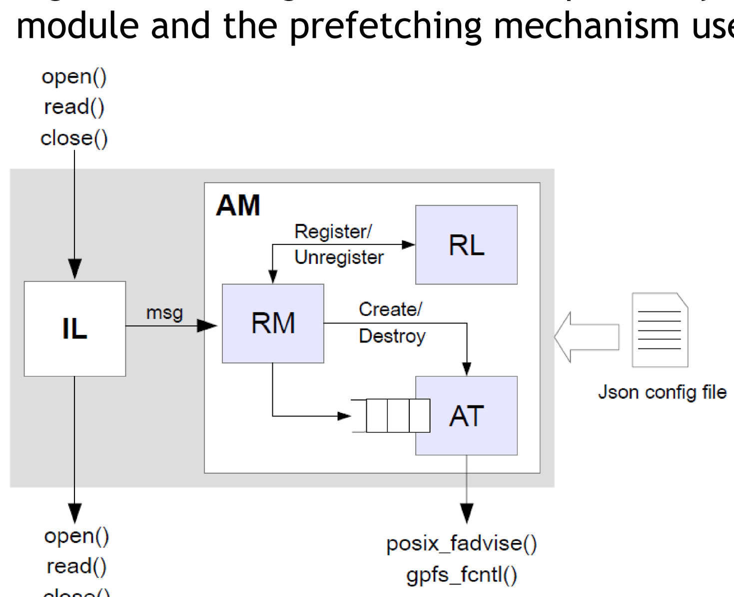


**Figure 2**: *Advice Manager (AM) component architecture, further divided in three blocks: Request Manager (RM), Register Log (RL) and Advisor Thread (AT).*

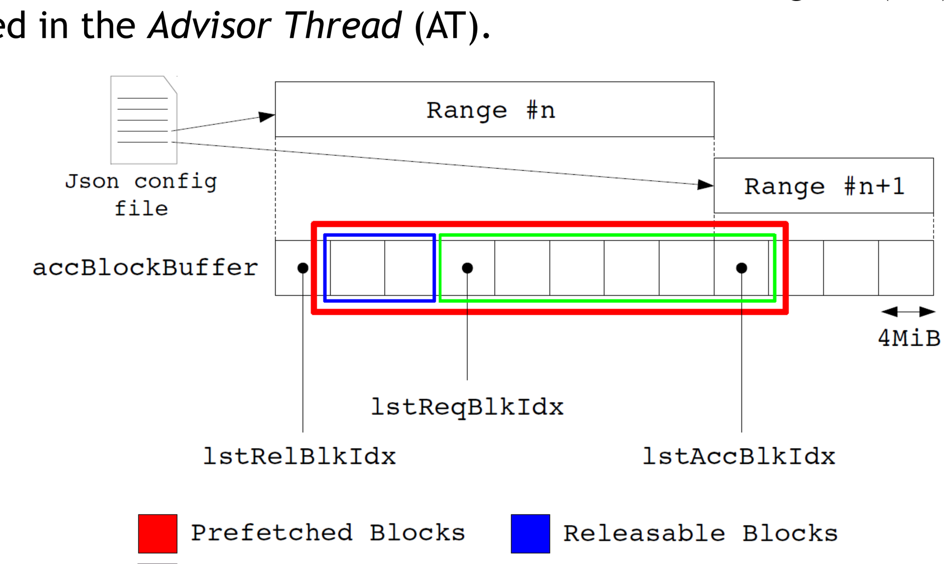


**Figure 3**: *Advisor Thread (AT) prefetching mechanism*

## POSIX Advice Integration with Lustre

Lustre is a high performance parallel file system for Linux clusters. It works in kernel space and takes advantage of the available page cache infrastructure. Additionally, it extends POSIX read and write operations with distributed locks to provide data consistency across the whole cluster. Even though Lustre makes use of the Linux kernel page cache, the previously described POSIX advice syscall has no effect on Lustre (Figure 4).

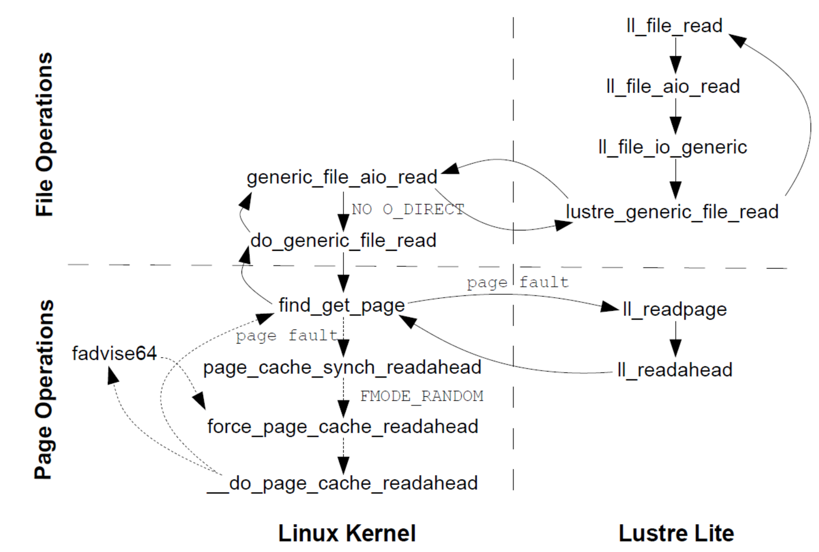


**Figure 4**: *Simplified function call graph for the read operation in Lustre. The picture also shows the call graph for local reads and POSIX_FADV_WILLNEED in the posix_fadvise() implementation (dashed line).*

Lustre extends the kernel code with additional file and page operations through the Lustre Lite component. These are the functions used by the kernel to fill the file operations table and the address space operations table. POSIX advice in the kernel translates into fadvise64. In the case of 'willneed' this function directly invokes force_page_cache_readahead which has no effect on ll_readpage. In order to enable 'willneed' in Lustre we modified the call graph of fadvise64 presented in Figure 4 to invoke the aio_read operation and block until all the data has been read into the page cache. In this way we can force the kernel to invoke the Lustre read operation, acquiring locks as appropriate.

## Evaluation

We evaluate the performance of our infrastructure using the execution time and the number of reads completed by every target file system: ext4, Lustre and GPFS. Our testbed is composed by a test cluster of seven nodes (intended to evaluate the proposed Linux kernel modification with the Lustre file system) and the Mogon cluster (the production system at the ZDV). The target application used to evaluate our advice infrastructure is written using 'ROOT', an object-oriented framework widely adopted to build software for data analysis. The application analyzes data read from an input file in the 'ROOT' format (structured file format of 5GB).

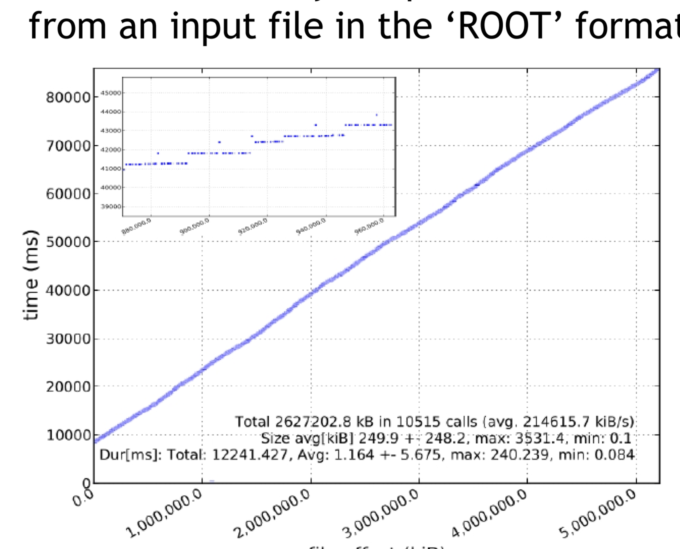


**Figure 5**: *Read profile for target application*

The I/O behaviour of the application (Figure 5) looks linear, most of the accesses to the file follow an increasing offset. Nevertheless, adjacent reads are separated by gaps (strided read pattern). We divided the target file into contiguous non overlapping ranges in which reads happen to have increasing offset. The information extracted was then used to tailor a configuration file. Additionally, we also used a configuration file containing only one prefetching region covering the whole file to describe the general I/O behaviour of the application.

Figures 6 and 7 show the runtime and the number of completed reads for the target application in both clusters. The runtime improvement is 11% (-10 sec) on Test Cluster and 6% (-13 sec) on Mogon for GPFS and 44% (-50 sec) on Test Cluster for Lustre. The number of reads is reduced by up to 83% (-3956) on both clusters for GPFS and up to 52% (-5727) on Test Cluster for Lustre.

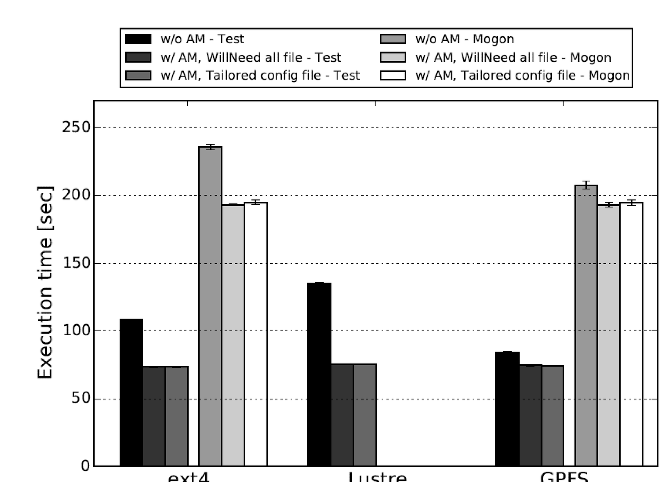


**Figure 6**: *Runtime*

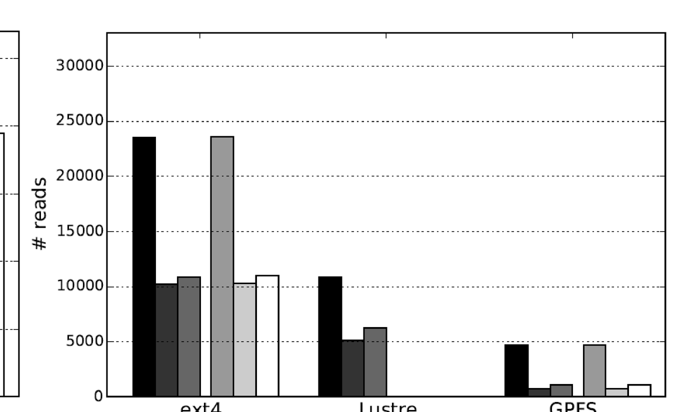


**Figure 7**: *Completed reads*

## Related Work & Conclusion

Before us, other works have used data prefetching to boost applications performance [7] – [14]. Our approach differs from those works since we do not rely on precise I/O pattern information to predict and prefetch every chunck of data in advance. Instead we use data prefetching to group many small requests in a few big ones, improving applications performance and utilization of the whole storage system. Moreover, we provide the infrastructure that enables users to access file system specific interfaces for guided I/O without modifying applications and hiding the intrinsic complexity that such interfaces introduce.

## References


1. ROOT, A data analysis framework. http://root.cern.ch/drupal.
2. posix_fadvise. http://linux.die.net/man/2/posix_fadvise.
3. gpfs_fcntl subroutine. https://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=%2Fcom.ibm.cluster.gpfs.v3r5.gpfs100.doc%2Fbl1adm_fcntl.htm.
4. R. Thakur, W. Gropp, and E. Lusk, "An abstract-device interface for implementing portable parallel-i/o intefaces".
5. R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective i/o in romio".
6. L. Ying, "Lustre ADIO collective write driver".
7. F. Chang and G. A. Gibson, "Automatic i/o hint generation through speculative execution".
8. Y. Chen, S. Byna, X.-H. Sun, R. Thakur, and W. Gropp, "Hiding i/o latency with pre-execution prefetching for parallel applications".
9. S. VanDeBogart, C. Frost, and E. Kohler, "Reducing seek overhead with application-directed prefetching".
10. N. Tran and D. A. Reed, "Automatic arima time series modeling for adaptive i/o prefetching".
11. J. He, J. Bent, A. Torres, G. Grider, G. Gibson, C. Maltzahn, and X.-H. Sun, "I/o acceleration with pattern detection".
12. S. Byna, Y. Chen, X. he Sun, and R. Thakur, "Parallel i/o prefetching using mpi file caching and i/o signatures".
13. Y. Chen and P. Roth, "Collective prefetching for parallel i/o systems".
14. J. He, X.-H. Sun, and R. Thakur, "Knowac: I/o prefetch via accumulated knowledge".