POSTER: Optimizing Scientific File I/O Patterns using Advice Based Knowledge

Giuseppe Congiu*, Matthias Grawinkel†, Federico Padua†, James Morse*, Tim Süß†, André Brinkmann†
*Emerging Technology Group Xyratex Technology LTD Havant, United Kingdom
Email: {giuseppe_congiu, james_morse}@xyratex.com
†Zentrum für Datenverarbeitung Johannes Gutenberg-University Mainz, Germany
Email: {grawinkel, padua, t.suess, brinkman}@uni-mainz.de

I. 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. As a result these applications perform poorly. More significantly they can negatively impact the whole storage system's efficiency.

In this paper we propose and evaluate a novel advice infrastructure able to optimize file access patterns at runtime through data prefetching using these hints mechanisms (Figure 1). It 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. 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.

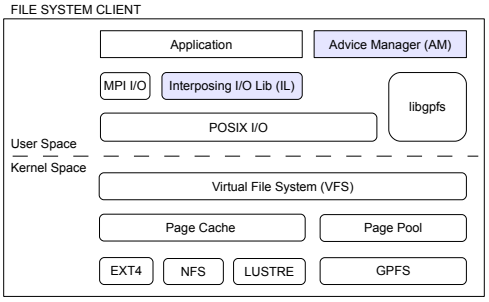

Fig. 1. I/O software stack of the advice infrastructure. *Interposing I/O Library* and *Advice Manager* communicate through UNIX domain sockets.

With this approach we are able to generate POSIX advice and GPFS hints for applications that do not use them but can receive a benefit from their use. We accomplish this 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. In order to enable `POSIX_FADV_WILLNEED` in Lustre we modified the call graph of `fadvise64()` presented in Figure 2 to invoke the `aio_read()` operation in the file operations table for the open file and block until all the data has been read into the page cache. In this way we can force the kernel to invoke the corresponding file read operation in Lustre, acquiring locks as appropriate.

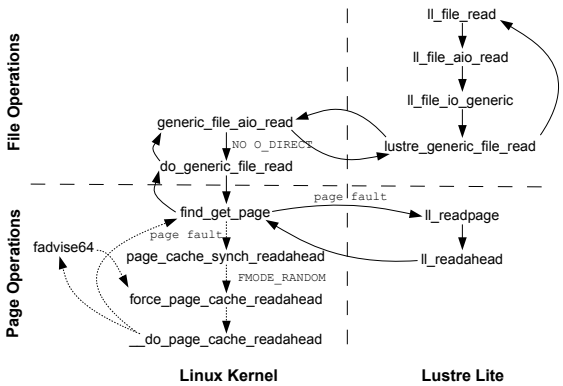

Fig. 2. Simplified function call graph for the read operation in Lustre. The picture also shows the call graph for local reads and `POSIX_FADV_WILLNEED` in the `posix_fadvise()` implementation (dashed line).

[1]Data analysis framework developed at CERN.

II. Evaluation

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

Figure 3 shows the I/O pattern of the application along with some additional statistics. As it can be seen, the application issues a total of 10515 `read()` system calls to read about 2.6 GB of data. The average request size is 250 kiB and the time spent waiting for I/O is 12 seconds, when running on the test cluster. At a first glance the general I/O behaviour of the

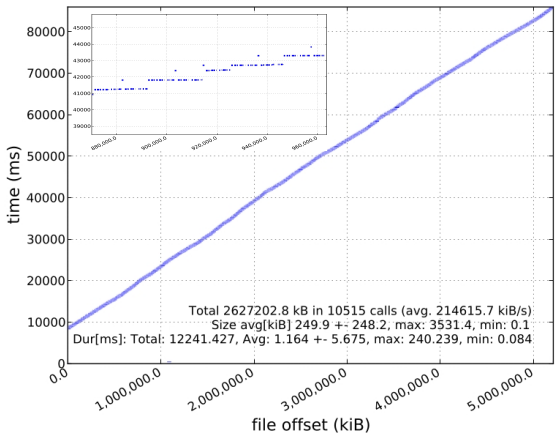

Fig. 3. I/O read profile of the target application under analysis extracted from the the GPFS file system in the test cluster.

application looks linear, most of the accesses to the file follow an increasing offset. Nevertheless, adjacent reads are separated by gaps (strided read pattern). In a few cases this gap becomes negative, meaning that the application is moving backwards in the file to read some data previously left behind.After a detailed I/O pattern analysis we could divide the target file into contiguous non overlapping ranges. Within these ranges 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 prefetch region covering the whole file. This is intended to evaluate the relationship between costs and benefits when building a complex configuration file, instead of using a simple one that describes the general I/O behaviour of the application.

Figure 4 and 5 show, respectively, the execution time and the number of completed reads for the target application in both clusters. The runtime shows an improvement of 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

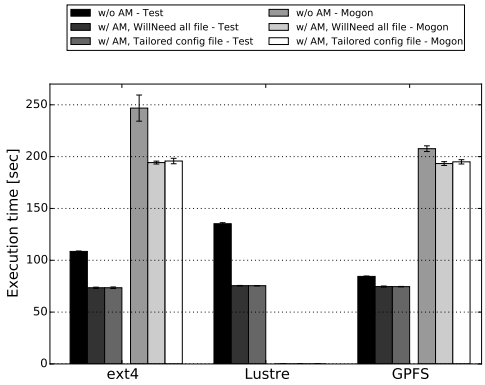

Fig. 4. Execution time of the target application on available file systems.

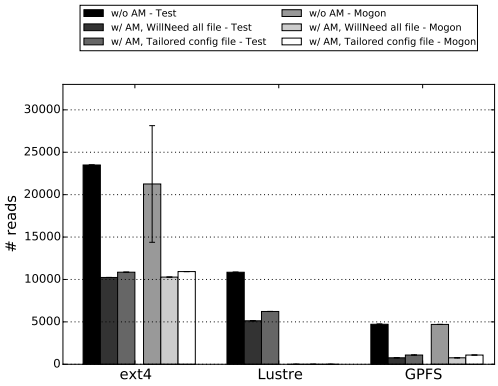

Fig. 5. Number of read operations completed by the different file systems.

Test Cluster and Mogon for GPFS and up to 52% (-5727) on Test Cluster for Lustre.

III. Conclusions

Before us, other works have used data prefetching to boost applications performance [3] [1] [2] [4]. 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.

References

[1] F. Chang and G. A. Gibson, "Automatic i/o hint generation through speculative execution," in *Proc. of the 3rd Conference on Operating Systems Design and Implementation (OSDI)*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 1–14.

[2] Y. Chen, S. Byna, X.-H. Sun, R. Thakur, and W. Gropp, "Hiding i/o latency with pre-execution prefetching for parallel applications," in *Proc. of the ACM/IEEE Conference on Supercomputing (SC)*, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 40:1–40:10.

[3] J. He, J. Bent, A. Torres, G. Grider, G. Gibson, C. Maltzahn, and X.-H. Sun, "I/o acceleration with pattern detection," in *Proc. of the 22nd*, ser. HPDC '13. New York, NY, USA: ACM, 2013, pp. 25–36.

[4] S. VanDeBogart, C. Frost, and E. Kohler, "Reducing seek overhead with application-directed prefetching," in *Proceedings of the 2009 conference on USENIX Annual technical conference*, ser. USENIX'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 24–24.