Noname manuscript No.
(will be inserted by the editor)

Towards I/O Analysis of HPC Systems and a Generic Architecture to Collect Access Patterns

Marc C. Wiedemann[1] · **Julian M. Kunkel**[1] · **Michaela Zimmer**[1] · **Thomas Ludwig**[1] · **Michael Resch**[2] · **Thomas Bönisch**[2] · **Xuan Wang**[2] · **Andriy Chut**[2] · **Alvaro Aguilera**[3] · **Wolfgang E. Nagel**[3] **Michael Kluge**[3] · **Holger Mickler**[3]

Received: / Accepted:

Abstract In high-performance computing (HPC) applications, a high-level I/O call will trigger activities on a multitude of hardware components such as massively parallel systems supported by huge storage systems and internal software layers. Currently, their complex interplay makes it impossible to identify the causes for and the locations of I/O bottlenecks. Existing tools indicate the bottleneck but provide little guidance to identify the cause and how to improve the situation.

Our project *Scalable I/O for Extreme Performance* was initiated to find solutions for this problem.
To achieve this goal in *SIOX*, we will build a system to record access information on all layers and components, recognize access patterns, and characterize the I/O system. Ultimately, it will localize the reasons for I/O bottlenecks and propose optimizations for the I/O middleware that improve I/O performance, such as throughput rate and latency. Furthermore, the SIOX system will support decision making while planning new I/O systems.

In this paper, we introduce the SIOX system and present its current status: the intended approach to collect the required access information, an architectural concept, methods to reconstruct the I/O path and an excerpt of the interface for data collection. The focus lies on the architecture, which collects and combines the relevant access information along the I/O path, and the efficient transfer of this information. An abstract modelling approach allows us to reduce the complexity of the I/O activities on parallel computing systems, while an abstract interface allows us to adapt the SIOX system to various HPC file systems.

Keywords I/O analysis · I/O path · Causality tree

We want to express our gratitude to the "Deutsches Zentrum für Luft- und Raumfahrt e.V." as responsible project agency and to the "Bundesministerium für Bildung und Forschung" for the financial support under grant `01 IH 11008 A-C`.

Marc C. Wiedemann[1]
E-mail: marc.wiedemann@informatik.uni-hamburg.de
Bundesstraße 45a - 20146 Hamburg
Julian M. Kunkel[1] - Michaela Zimmer[1]
Prof. Dr. Thomas Ludwig[1]
[1] Universität Hamburg - Deutsches Klimarechenzentrum GmbH

Prof. Dr.-Ing. Dr. h.c. Dr. h.c. Michael Resch[2]
Dr.-Ing. Thomas Bönisch[2]
Xuan Wang[2] - Andriy Chut[2]
[2] High Performance Computing Center Stuttgart (HLRS) Universität Stuttgart

Alvaro Aguilera[3] - Prof. Dr. Wolfgang E. Nagel[3]
Dr.-Ing. Michael Kluge[3] - Holger Mickler[3]
[3] Technische Universität Dresden
Zentrum für Informationsdienste und Hochleistungsrechnen

1 Introduction

This paper is structured as follows: Chapter 1 states the scientific problem and a possible way to solve it with the SIOX approach. Chapter 2 gives an overview of the I/O software and *parallel file systems* (*PFS*) used in HPC. We propose a solution to the problem and show the locations of information extraction in Chapter 3.
Chapter 4 presents the architecture, the SIOX system's general workflow with the cause-and-effect chain, general communication between the components and the fine structure of activity data collection from clients. Chapter 5 shows how the SIOX system is going to be implemented. A SIOX interface is shown as an excerpt. In Chapter 6, we are reconstructing the causal I/O path through functional nodes by using a graphical model.
In Chapter 7, we innovate a combination graph of the

The final publication is available at Springer via http://dx.doi.org/10.1007/s00450-012-0221-5

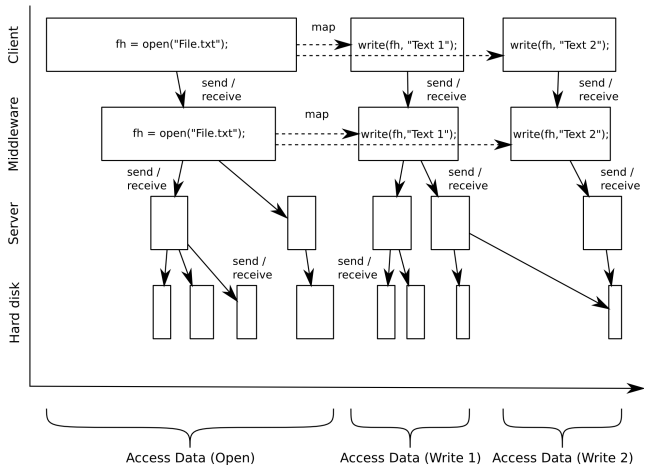

Fig. 1 An example of the two distinct types of causal connections on the I/O path: Vertically, there are the *send/receive* communications, while horizontally, there are the *mappings*, e. g. translations from file names to file handles. Access patterns with layer-wise interactions are initiated through the `open()`-call of a client process, possibly leading to activities that cannot be unambiguously attributed to one single cause.

causally dependent PFS activities, linking the application processes to the I/O nodes involved in these activities. Chapter 8 consolidates the main aspects of our scientific findings within the project.

One of the most pressing problems in HPC systems is the I/O bottleneck: the performance of individual storage units does not grow at the same rate as CPUs. To mitigate this problem, the PFSs vertically scale to many individual components. This increasing complexity and the anonymous storage of blocks of bytes without any relation to user application software exacerbates the analysis and identification of system bottlenecks in I/O transactions. Due to contradictory requirements of different user groups, the global optimization of a PFS is a complex task, while the application related optimization takes place as a long communication process between the users and the system custodian.

It is difficult to know whether a once recognized I/O pattern of an application leads to a basis for the interpretation of diagnosed performance problems. In [4], it is stated that performance tools often detect symptoms of performance problems rather than causes. Especially in PFSs, the symptoms may appear much later than the causing events, and might be located on another physical node. The I/O bottleneck may not be identified in timelines because of caching. This makes the attribution of dependent calls and their actions difficult.

Time-shared accesses to one specific file further adds to the difficulty of I/O analysis.

An example of the complex association of activities on the data path, is shown in FIGURE 1. Along the vertical axis, send/receive communications are transmitted through the software/hardware layers, while along the time axis, function calls are connected via the use of identical or derived descriptors, such as file name or file handle. As a consequence, causal attribution of activities may well be ambiguous.

2 State of the Art

The toolchain Magpie [11] accurately attributes the actual usage of CPU, disk, and network to the appropriate request by correlating the events that were generated on live requests using a schema of the event relationships. Magpie's approach relies on high precision time stamps. It is generic and flexible, because the parser can look at any attribute when performing a join for the causal chain of a request.

Stardust [14] introduced activity tracking of per-client, per-request, as well as per-workload latency maps information in a Storage Area Network system (SAN) by keeping all traces. The system incorporates the full distribution of access locations, direct end-to-end tracing and online monitoring of a specific SAN system, where the types of requests have to be known in advance. Resources of interest in that SAN include the CPUs, cache-levels, network layout and storage devices. In shared environments, however, the aggregate performance counters do not differentiate between processloads and present only combined workload measurements [14]. In contrast, SIOX aims to analyse I/O requests in all HPC systems.
Annotated Plan Graphs (APG) [1] differentiate between inner, direct, and outer dependencies indirectly influencing the performance of a database operator on the inner path through components. Each APG graph component is annotated with appropriate monitoring data collected during the database plan's execution. Various data are collected within the limits of the SAN, among them: the physical and logical configuration of components, changes in configuration and connectivity in time, performance metrics, system-generated events (disk failure, RAID rebuild) and user-defined triggers (e. g. degradation in volume performance, high workload on SAN).

The causal relation between parent and child processes, and the system activity can be examined with a trace visualization tool (e. g. [15, SUNSHOT]). Other trace environments are TAU [12], Vampir [6], and Scalasca [3]. None of them trace MPI and a parallel FS together. Score-P[1] aims to be a successor of these trace environments using OTF2 which will become a

[1] `www.vi-hps.org/projects/score-p`

standard in TAU, Vampir and Scalasca. The maximum number of trace files that can be monitored in this 2D-environment depends on the screen's pixel-size, total dimensions and multiplicity. The tracing API HDTrace [8] allows the connection to modified PFSs such as GPFS, Lustre and PVFS2.

The Adaptive I/O System (ADIOS) [9], a scalable, portable and efficient component of the I/O system on different platforms, provides all users with the means to choose their optimal I/O transport methods based on their I/O patterns. It utilizes XML for its configuration file to define or control the I/O access behaviours. A file format, BP, introduced by ADIOS, plays the role of an intermediate format and can be easily converted to HDF5 or other file formats.

One of today's most popular implementations of MPI I/O is ROMIO [13], which runs on various machines and is included in several MPI implementations. The two I/O optimization techniques implemented by ROMIO are data sieving and two-phase I/O. The former one improves the performance of accessing noncontiguous regions of data, while the latter one handles the collective I/O operations.

Comparing to ROMIO, OMPIO [2], a new parallel I/O architecture for Open MPI, takes advantage of different frameworks to provide more finely separated I/O operations. The fine grained frameworks increase the modularization of the parallel I/O library and utilize various algorithms such as two-phase I/O to adapt to the different parallel I/O operations. Special focus is set on MPI I/O, where we provide I/O hints to the MPI library to make future improvements possible.

These state of the art approaches are as diverse as the systems to analyse. In SIOX, we present an approach to instrument all these systems.

3 SIOX: Scalable I/O for Extreme Performance

The SIOX project is organized as a collaboration of the university partners named at the beginning, DKRZ GmbH and IBM AG, and aims to identify any user's application program and the I/O bottleneck related to it. To this end, we construct an open source system for tracking and relating system activities on all abstraction layers, to extract their causal relations and to reconstruct access patterns for automatic optimization proposals. Furthermore, SIOX aims to provide tracking information about the internal behaviour of the different MPI I/O implementations. The integrated analysis of application, PFS and HPC hardware is a basis for optimization in other scenarios. For instance, the computing center may identify unfavourable access patterns and suboptimal applications.

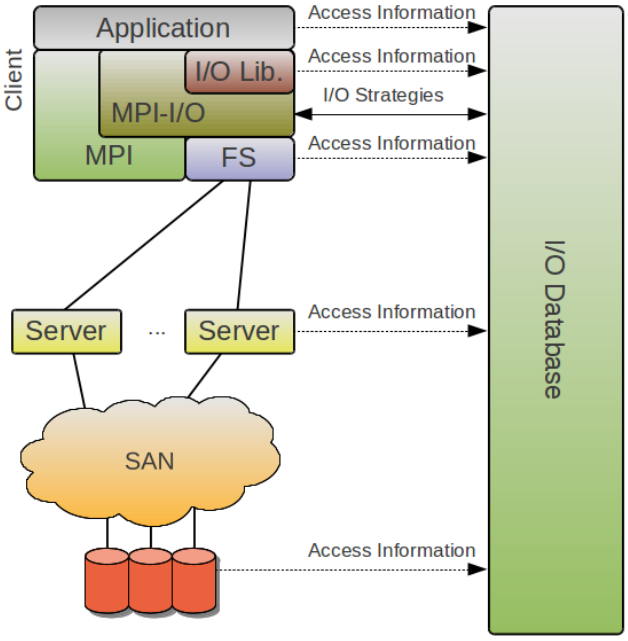

Fig. 2 Information extraction on four software layers and at two hardware locations (server and I/O nodes).

Communications and mappings will be monitored through the SIOX system to collect associations between system activities and to reconstruct the causal tree of connected activities. Synchronized timing of software and hardware clocks is a key requirement for all I/O analysis. It must be assured that at any point where data is captured, a synchronized time source is used by the system. The necessary accuracy has yet to be determined, but synchronization based on the network time protocol is expected to be sufficient.

Automatic analysis of access patterns will help to estimate the efficiency of the patterns observed. With this objective, the individual layers of a PFS, which perform transformations while processing I/O calls, ought to be analytically described. Based on knowledge about the identified access patterns, the SIOX system aims to propose possible application-oriented performance optimizations.

For I/O analysis, the utilization profiles are to be continually written to memory. For fast analysis of unexpected problems and planning of HPC system investments, multiple parameters of the HPC system and the current I/O patterns need to be stored. The provisioning of information about different I/O patterns allows to continually improve HPC systems.

Parallel to development, first components will be tested with real-world applications. Applications of the DKRZ, for instance, will be optimized by the SIOX system during the second half of the project's time frame.

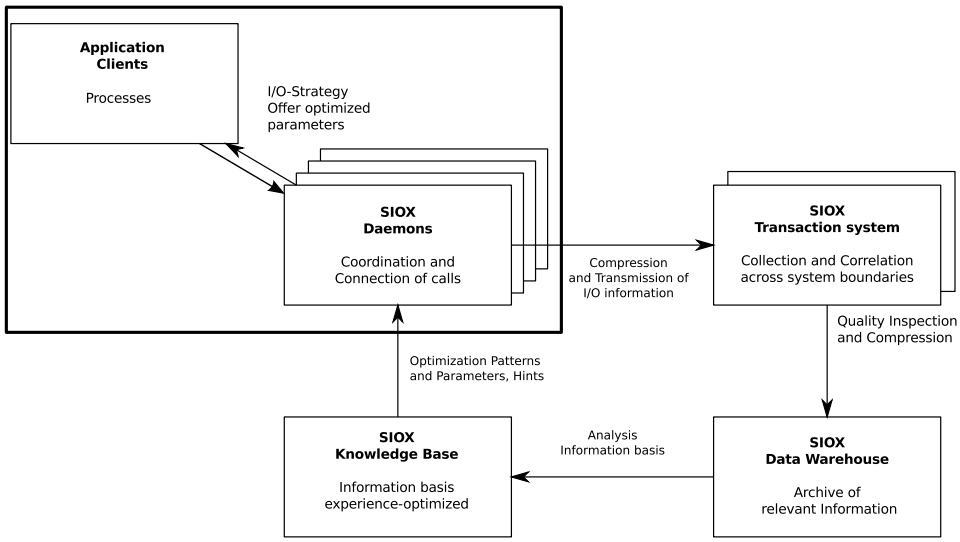

Fig. 3 Architecture for the development of the SIOX system.

FIGURE 2 shows from which locations in hardware and software layers data is to be extracted via the SIOX interface. In particular, these include parallel applications, the HDF5 interface, the PFS involved (GPFS, Lustre, etc.), the server nodes, and the SAN system (HPSS, RAIDs with dedicated storage nodes, etc.). We aim to collect, compress and permanently store access information on all relevant layers.

Translations are a key element to relate observed activity with operations on the call sequence, because the reference, on which an operation is performed, usually changes.

The access information can be stored on each layer, including a reference to the object it operates on. If the mappings on all translation nodes are recorded, a causal relationship can be inferred, e. g. whenever access to a file caused activity on a device. If multiple clients access the same file (and offset), a global unique identifier is necessary to record which client caused a particular activity on a device.

4 Architecture

The conceptual architecture to develop the SIOX system is depicted in FIGURE 3. The design allows for the system to propose optimization calls at runtime using an automatic, iterative improvement procedure.

The architecture consists of five main software construction parts: the I/O strategy for the clients, the daemons, the transaction system, the data warehouse, and the knowledge base. The knowledge base contains the knowledge about aggregated information of the data warehouse, the hardware topology (i. e. network connections) and characteristics, and the parameters for optimization (e. g. "Observed I/O throughput drop for packets smaller than 4 MiBs, therefore aggregation of smaller packets to 4 MiB packets is proposed by the SIOX system").

The databases are differentiated into OLTP data transaction and OLAP data analysis. The data warehouse uses the extract-transform-load (ETL) process, working in parallel cluster mode (not a single disk or RAM is used by all associated nodes, data is only exchanged over the network, every node adds data storage and I/O bandwidth).

The interoperation of all architectural parts will result in an automatically improving system through quality inspection, analysis and expertise optimizations.

4.1 Definitions

In order to foster a common understanding of the SIOX system, we introduce the following general terms and identifiers (ID) for active SIOX developers.

Nodes are functional logical units in a parallel computing system consisting of hardware components and software-layers. The SIOX system refers to every node by a unique node ID *UNID*, made up of a *hardware ID (HWID)*, *software ID (SWID)*, and optionally a *unique instance ID (IID)* on the same HWID and SWID.

Edges are logical pathways from one node to another forming the logical connection network, see Chapter 6.

A *component* is a real hardware or software unit consisting of at least one node.

An *activity* is an intended act in the progression of the data flow. Activities physically take place on components and logically on nodes. Activities can cause cascading I/O activities.

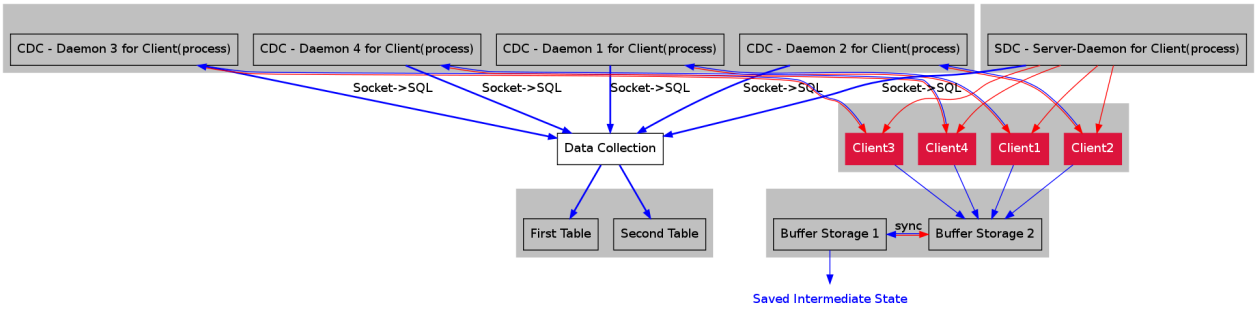

Fig. 5 Subsystem communication between client and daemons with double buffering, saved intermediate states, and collection of SQL querys. A server daemon creates daemons for client processes to collect client information.

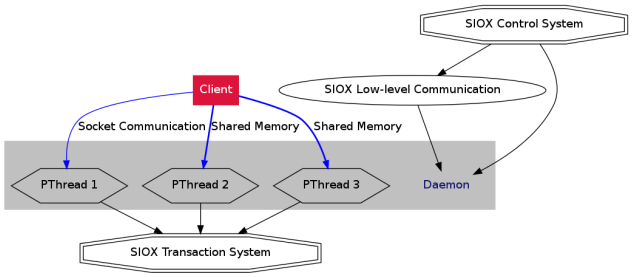

Fig. 4 Fine structure of the interaction between a client, the hidden processes (daemons) and the transaction system. The interaction is modularly organized using either network socket communication or shared memory .

The *activity identifier (AID)* is used to attribute the performance data nodes report at the beginning and at the end of an activity.

Descriptors are various designators, which are defined for the identification of entities or activities in nodes and along edges.

4.2 Fine Structure

The SIOX daemon system functions as a client-server interface (see FIGURE 4). In this fine structure, SIOX control organizes the low-level communication interface and the client-server interface.

In a HPC system, many operations take place concurrently, producing fine-grained data, measured at a large number of I/O locations. Therefore, we need local preprocessing to reduce the volume of collected data. It will be critical to implement a step-by-step aggregation process based on the hardware units, core, node, subsystem (maybe nodes in one rack), and total HPC system available to the particular program. The client-server interface will be implemented as a daemon that correlates node connections, reduces redundancies, aggregates the results and sends them to the data collection center (see FIGURE 5).

The interaction of the daemon modules is shown in FIGURE 5. Daemons are created for each client by a server daemon. After creation, they serve as the interface between clients and the data collection via SQL over socket communication. All data storage is generally layer-independent.

4.3 Compression Design

The SIOX system workflow requires the online transmission, analysis and storage of I/O events produced by the HPC applications. To ensure scalability, it is necessary to reduce the amount of data being transmitted to, and handled by the SIOX system. This can be aided by the use of different compression methods at key parts of the workflow: There is a great compression potential in the I/O traces generated by SPMD applications since they generally have a similar content. The knowledge base could be stored using a compressed searchable format to save storage space, optimize queries, and ease management. In order to reduce the network load during I/O peaks, the transmission of trace data could optionally be compressed using a light-weight compression method without adding much latency to the communication. Finally, it could be advantageous to use compressed data structures for the data analysis to speed up the pattern search and reduce the SIOX system's main memory footprint.

4.4 Causal Tree and I/O Path Model

Focusing on functionalities rather than on the components they are hosted on, is one way to simplify the software-hardware-interaction in order to gain a clear system overview. On the one hand, the type of collected data varies from component to component, on the other, the interrelations between entities in the context of internal data transactions are difficult to visualize. To solve this problem, we employ the structuring in

form of basic functionalities according to the I/O PATH MODEL [5]. Thus, focussing on abstract functionalities allow us to cover the diversity of HPC systems.

5 Realization

Information is collected from different layers and different components of the PFS. For example, in case of GPFS, information can be retrieved from the PFS client and Network Shared Disk (NSD) client component, which is a software layer providing a virtual view of the underlying disks. From the client, we collect information on a per I/O call basis. Furthermore, information from the NSD server layer on accessing storage devices is gathered. The data warehouse can be a RAM-centralized table system which scales with the number of I/O nodes and should store vector-oriented columns in parallel files to reduce response time. Furthermore, it should be compatible with the enterprise transaction system database (i. e. PostgreSQL).

5.1 SIOX System Interfaces

In the SIOX system, standardized C interfaces allow us to map all available HPC file systems. One covers the functions for registering, assigning and unregistering nodes, edges and descriptor mappings, well as descriptor creation, transfer, mapping and release. The identifiers are issued by the SIOX system when the nodes are initialized via

```
SIOX_register_node()
```

or released when they sign off. In contrast, edges are formed when connections are registered with

```
SIOX_register_edge()
```

between parent nodes and child nodes. The reporting of nodes' attributes delivers information on capacity and other component capabilities. Activities have a starting point, an endpoint and attributes that can be reported via timestamps.

5.2 Compression Techniques

For the technical realization of Chapter 4.3, it would be necessary to adapt one of the existing MPI trace compression techniques such as ScalaTrace [10] or compressed Complete Call Graphs (cCCG) [7] to work on the fly and with a SIOX-specific trace format. The compression of the knowledge base will depend on the capabilities of the storage solution of choice. As for the compression of the network communications, it can be accomplished using well-known LZ-libraries like snappy, lzop, quicklz, etc. and a combination of small dictionary sizes and time-out triggered buffering. Last but not least, the memory structures used in the SIOX system's learning cycle could be based on cCCGs or developed ad-hoc exploiting the previous compression.

5.3 Tracking the Cause-and-Effect chain

There are two approaches to keeping track of the cause-and-effect chain. Either access information is transported through software layers with metadata, or accesses are implicitly linked together through an object reference (or an aggregated reference of sub information). In the first case, the interfaces of intermediate layers must be modified to transport a unique call-ID. This may be connected with high effort and expenses, because of the required adaptation of the ROM-code on host bus adapters (HBA), and because communication protocols such as iSCSI or proprietary layers have to be adapted out of their standard. In the second case, the reconstruction of the causal relations is possible because it is known that access on the upper call activity precedes activity on the ones below. However, write-behind caches defer activity – and therefore make the temporal correlation harder, but other components trigger immediate action on the connected components. Caches may aggregate operations from multiple clients into one large request, therefore the detailed knowledge offered by the unique call ID might be unnecessary for many use-cases. Read-ahead is also problematic, a read operation might trigger further activity, but the later usage of this cached data might be caused by another client.

Taking advantage of the I/O PATH MODEL, we have decided to use the implicit approach. By utilizing the causal path view of generic calls (e. g. open, close, write, read), we will construct a dependency graph of the whole I/O system touched by the SIOX system and continuously update it with the information collected.

6 Reconstructing the I/O Path from Activities

In the SIOX context, we need to assign logical entities to functionalities and hardware to monitor the PFS's activity.

The first step towards this goal is the abstraction to graphical nodes combining the I/O PATH MODEL with activity identifiers (see Chapter 6.2). FIGURE 6 depicts an example I/O path that shows the I/O data flow from two servers over caches to a switch, to network cache, to an interconnected switch system and over a SAN

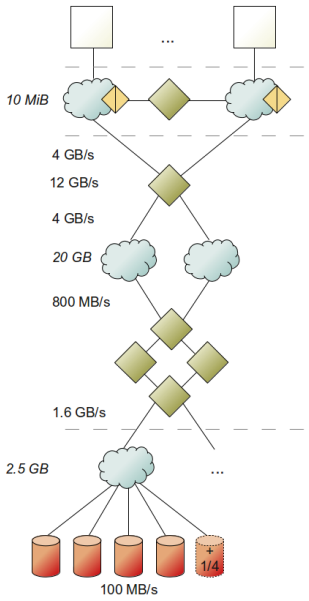

Fig. 6 The activity oriented I/O PATH MODEL [5] of two servers, connected via network to a RAID-5 storage system. The numbers represent bandwidths between nodes and normal (without read-ahead and write-behind) caches, shown as clouds with data size. Note that the total amount of outgoing data at one cross-section of the graph equals the total amount of incoming data there.

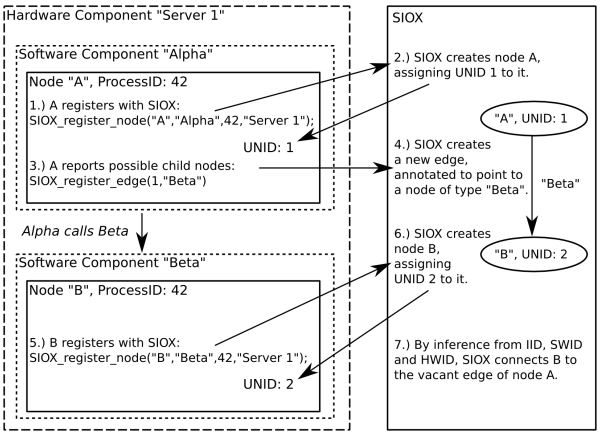

Fig. 7 Causal communication path during initial modelling phase. Node A on server 1 with PID 42 uses node B on server 1. The nodes are instrumented for SIOX. Additional calls to report further attributes and capabilities are omitted for clarity.

cache to block storage. Interfaces can be built for each of the mentioned functionalities. The I/O PATH MODEL indicates where these interfaces must be installed on a given PFS deployment, and how those layers interact. This will enable us to assess and optimize the layers.

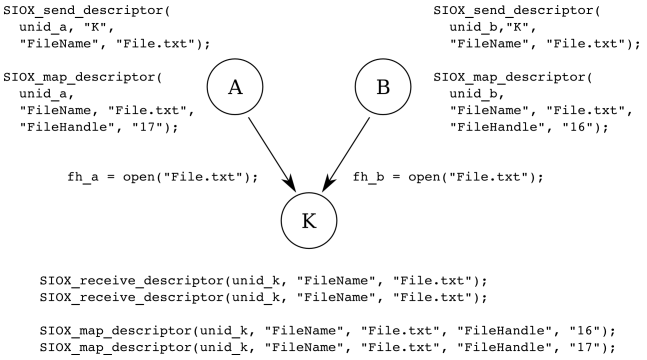

Fig. 8 Special case where two UNIDs on instrumented software-layers open the same file on `unid_k`. Without reporting the descriptor mappings of `unid_a`, `unid_b` and `unid_k`, no distinction between actions initiated by `unid_a` and `unid_b` is possible.

6.1 SIOX System Communication Procedure

On initialization, every node in an HPC system running the SIOX system has to register with its HWID, SWID and PID. Examples for HWIDs are *Node 12, HD 6784*, or *Server 1*. The SWID would be a character set such as "POSIX". An example of the registration process of two nodes instrumented for SIOX is shown in FIGURE 7.

6.2 Activity Information Collection

In order to distinguish different "I/O activities", an AID is issued for each. Redundant information will be discarded as soon as possible to reduce the memory footprint and speed up the processing.

Activity information is collected from the different organizational layers of the access path. In the case of OMPIO (in Open MPI), information can be obtained from the different internal layers. Upon receiving a call to open a file with MPI I/O, Open MPI selects either ROMIO or OMPIO to perform the I/O operations, similar to the selection logic of other Open MPI frameworks. If OMPIO is selected, it initializes all sub-frameworks and these query their available modules. The best fitting module will be used for the following set of operations. For example, if GPFS is used, the GPFS specific module or the module optimized for GPFS will be selected. In the same way, a module with a suitable I/O algorithm will be chosen. The information about the modules used will be provided to the SIOX system. In addition, access information will be gathered from the sub-frameworks as well as from the different modules. If ROMIO is selected, the Abstract-Device interface for I/O (ADIO) within ROMIO gains the controlling information of the I/O operation via the

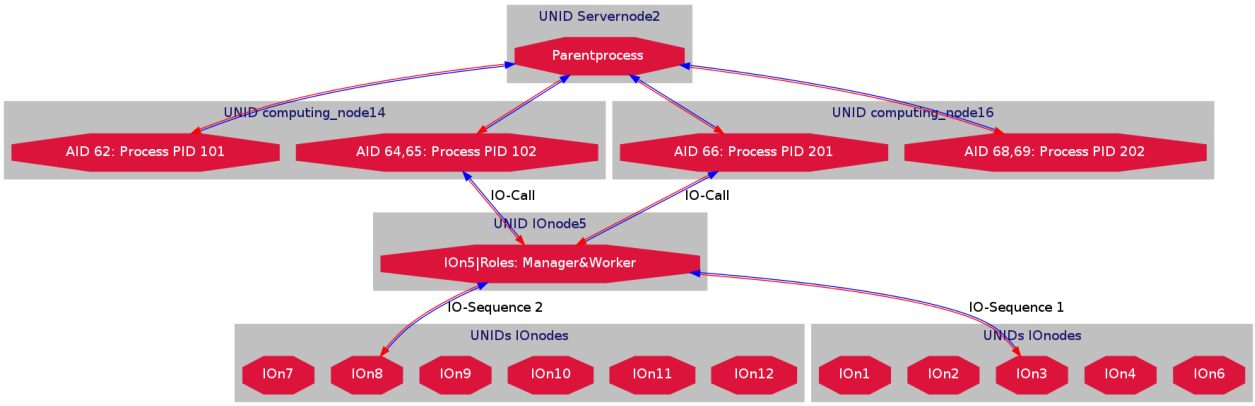

Fig. 9 ActivitynetGraph: An illustration of activity I/O on a graph combining hardware locations and software process activity identifiers.

hints attached to the file-access operations. In this case, we provide information about the chosen algorithm and further available performance information.

Access information is correlated, transformed, compressed, and stored as an access pattern which allows both modelling and analysis. Subsequently gained knowledge enables us to optimize the I/O strategies.

7 Combination of Activity Entities and Node Information

For information extraction, it will be necessary to have a multidimensional approach, covering the complex network of physical nodes and interacting entities such as activity nodes in FIGURE 9. One way is to state the abilities of the distributed entities first, then the relations between them. A graph would integrate these relations grouping similar entities together as functional groups. Functional groups for client activities are `access()`, `open()`, and other calls, for storage activity the amount of data written/read per time frame, for network activity the network packets.

Activity nodes initiate activities or react to activities, such as when answering a call. To make it possible to track AIDs and Node-IDs together, the graphviz libraries[2] can be used to draw changeable graphs of complex systems.

With the the analysis and correlation of the AID and node information, the SIOX knowledge base (see FIGURE 3) will accumulate specific expertise about which calls on which nodes may cause unwanted system activity, and will be able to propose optimizations: At this point, I/O calls could be rewritten to produce more favourable access patterns or a summary which specific PFS or Open MPI adjustment parameters could be issued to the (user) application.

8 Conclusion

Although we have not solved all problems of Chapter 1 yet, we have taken steps towards a comprehensive solution. SIOX's aim is to create a system able to analyze the I/O requests in HPC systems on a per-request base. In this paper, we introduce first approaches: Due to the abstraction of technical details of the hardware and software used, future systems can be instrumented in a generic, structured, maintainable and modular way. Present achievements of the SIOX project include the system architecture, the locations of information extraction, first interfaces and a client-server daemon structure. We introduced a graph of the combination of I/O hardware locations and software process activities. Optimization potential for specific application scenarios exists on the layers application, high-level I/O such as MPI-I/O and NetCDF, HDF5, the operating system, the HPC file system and at hardware locations such as I/O nodes, computing nodes and storage controllers. The next steps include an evaluation of the interfaces built, the instrumentation of MPI-I/O and NetCDF and the theoretical survey for the analysis of patterns. Also, more interfaces will be defined, and the first implementations started.
Interested readers are kindly encouraged to become involved in the project through our internet presence[3].

[2] `www.graphviz.org/pdf/libguide.pdf`

[3] `www.hpc-io.org`

References

1. Babu, S., Borisov, N., Uttamchandani, S., Routray, R., Singh, A.: DIADS: Addressing the "My-Problem-or-Yours" Syndrome with Integrated SAN and Database Diagnosis. In: FAST'09: Proccedings of the 7th conference on File and storage technologies, pp. 57–70. USENIX Association, Berkeley, CA, USA (2009)
2. Chaarawi, M., Gabriel, E., Keller, R., Graham, R.L., Dongarra, J.J.: Ompio: A modular software architecture for mpi i/o (2011)
3. Geimer, M., Wolf, F., Wylie, B.J.N., Becker, D., Böhme, D., Frings, W., Hermanns, M.A., Mohr, B., Szebenyi, Z.: Recent Developments in the Scalasca Toolset. In: Tools for High Performance Computing, Proceedings of the 3rd International Workshop on Parallel Tools. Springer (2009)
4. Hermanns, M.A., Geimer, M., Wolf, F., Wylie, B.J.N.: Verifying Causality Between Distant Performance Phenomena in Large-Scale MPI Applications. In: Proc. of the 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), Weimar, Germany, pp. 78–84. IEEE Computer Society (2009)
5. Julian Kunkel, Thomas Ludwig: IOPm - Modeling the I/O Path with a Functional Representation of Parallel File System and Hardware Architecture. to be published (2011)
6. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M.S., Nagel, W.E.: The Vampir Performance Analysis Tool-Set. In: Tools for High Performance Computing, Proceedings of the 2nd International Workshop on Parallel Tools, pp. 139–155. Springer (2008)
7. Knüpfer, A., Nagel, W.E.: Compressible memory data structures for event-based trace analysis. Future Gener. Comput. Syst. **22**, 359–368 (2006)
8. Kunkel, J.: HDTrace – A Tracing and Simulation Environment of Application and System Interaction. Tech. Rep. 2, Deutsches Klimarechenzentrum GmbH, Bundesstraße 45a, D-20146 Hamburg (2011)
9. Lofstead, J., Zheng, F., Klasky, S., Schwan, K.: Adaptable, metadata rich io methods for portable high performance io (2009)
10. Noeth, M., Ratn, P., Mueller, F., Schulz, M., de Supinski, B.R.: Scalatrace: Scalable compression and replay of communication traces for high performance computing. J. Parallel Distrib. Comput. **69**, 696–710 (2009)
11. Paul Barham Austin Donnelly, R.I., Mortier, R.: Using magpie for request extraction and workload modelling. Microsoft Research (2004)
12. Shende, S.S., Malony, A.D.: The TAU Parallel Performance System. Int. J. High Perform. Comput. Appl. **20**(2), 287–311 (2006)
13. Thakur, R., Gropp, W., Lusk, E.: On implementing mpi-io portably and with high performance (1999)
14. Thereska, E., Salmon, B., Salmon, O., Strunk, J., Wachs, M., el malek, M.A., Lopez, J., Ganger, G.R.: Stardust: Tracking Activity in a Distributed Storage System. In: ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 3–14. ACM Press (2006)
15. Timo Minartz Daniel Molka, J.K.M.K.M.K.T.L.: Handbook of energy-aware and green computing. Chapman and Hall/CRC Press Taylor and Francis Group LLC (2012)