Comput Sci Res Dev (2013) 28:241–251
DOI 10.1007/s00450-012-0221-5

SPECIAL ISSUE PAPER

Towards I/O analysis of HPC systems and a generic architecture to collect access patterns

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

Published online: 23 May 2012

Abstract In high-performance computing applications, a high-level I/O call will trigger activities on a multitude of hardware components. These are massively parallel systems supported by huge storage systems and internal software layers. Their complex interplay currently makes it impossible to identify the causes for and the locations of I/O bottlenecks. Existing tools indicate when a bottleneck occurs but provide little guidance in identifying the cause or improving the situation.

We have thus initiated *Scalable I/O for Extreme Performance* 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, to recognize access patterns, and to characterize the I/O system. The system will ultimately be able to recognize the causes of the I/O bottlenecks and propose optimizations for the I/O middleware that can improve I/O performance, such as throughput rate and latency. Furthermore, the SIOX system will be able to support decision making while planning new I/O systems.

In this paper, we introduce the SIOX system and describe its current status: We first outline our approach for collecting the required access information. We then provide the architectural concept, the methods for reconstructing the I/O path and an excerpt of the interface for data collection. This paper focuses especially on the architecture, which collects and combines the relevant access information along the I/O path, and which is responsible for the efficient transfer of this information. An abstract modelling approach allows us to better understand the complexity of the analysis of the I/O activities on parallel computing systems, and an abstract interface allows us to adapt the SIOX system to various HPC file systems.

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

1 Introduction

This paper is structured as follows. Section 1 states the scientific problem and offers a possible solution with the SIOX approach. Section 2 gives an overview of the I/O software and *parallel file systems* (*PFS*) used in HPC. We further detail our solution to the problem and show the locations of information extraction in Sect. 3.

Section 4 presents the architecture, the SIOX system's general workflow with the cause-and-effect chain, the general communication between the components and the fine structure of activity data collection from clients. Section 5 provides an implementation concept of the SIOX system. The SIOX interface between client processes and the SIOX system is given as an excerpt. In Sect. 6, we reconstruct the causal I/O path through functional nodes using a graphical

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.

M.C. Wiedemann (✉)
Bundesstraße 45a, 20146 Hamburg, Germany
e-mail: marc.wiedemann@informatik.uni-hamburg.de

M.C. Wiedemann · J.M. Kunkel · M. Zimmer · T. Ludwig
Universität Hamburg—Deutsches Klimarechenzentrum GmbH, Hamburg, Germany

M. Resch · T. Bönisch · X. Wang · A. Chut
High Performance Computing Center Stuttgart (HLRS), Universität Stuttgart, Stuttgart, Germany

A. Aguilera · W.E. Nagel · M. Kluge · H. Mickler
Zentrum für Informationsdienste und Hochleistungsrechnen, Technische Universität Dresden, Dresden, Germany

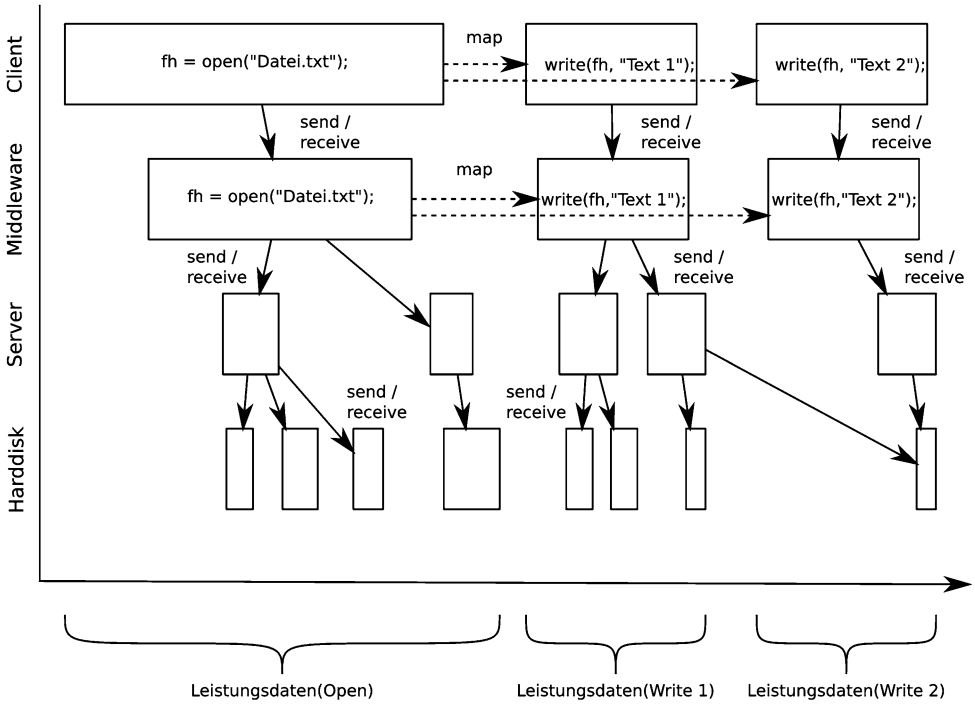

Fig. 1 An example of the two distinct types of causal connections on the I/O path: Along the vertical axis, communications are transmitted through the software/hardware layers; along the time axis, function calls are connected via the use of identical or derived descriptors, such as file name or file handle

model. Section 7 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 that of CPUs. To mitigate this problem, the PFSs scale horizontally to many individual components. The increasing complexity and the anonymous storage of blocks of bytes yield no relation to user application software. This exacerbates the difficulties in 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, because 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 [5], 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 problematic, because time-shared accesses to one specific file further add to the difficulty of I/O analysis.

An example of the complex association of activities on the data path is shown in Fig. 1. Vertically, communications run through a simplified hierarchy from client via middleware and server to block storage. As these steps characteristically involve the passing of some kind of *descriptor*, such as file name or logical block address, we refer to them as *send/receive* relations. Horizontally, function calls are connected through the use of identical or derived descriptors, like a file handle derived from a file name, giving rise to the term *mapping* relations. Note that effects such as cache writes can cause activities, like the rightmost one on block storage level, that cannot be unambiguously attributed.

2 State of the art

The toolchain Magpie [2] 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 of a request for the causal chain. The parser analyzes the file to reach an understanding of the input's semantic so it can take the appropriate action afterwards. Stardust [15] has introduced activity tracking of per-client, per-request, as well as per-workload latency map information in a Storage Area Network system (SAN) by keeping all traces. Optional pruning of trace information is possible. 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. Activity Records are used to measure resource consumption and request latencies. Resource consumption of interest in that SAN includes

the CPU demand, buffer cache demand, network demand and disk (Abstraction: storage device disk or SSD) demand. Request latencies include NFS (Abstraction: PFS) service, metadata service, and storage node types. In shared environments, however, the aggregate performance counters do not differentiate between process loads and present only combined workload measurements [15]. In contrast, SIOX aims to analyze 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 execution of the database plan. Various information is collected within the limits of the SAN, including: the physical and logical configuration of components, or 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 system activity can be examined with a trace visualization tool. The state-of-the-art trace environments are TAU [13], VAMPIR [7], and SCALASCA [4]. None of them trace MPI and a parallel FS together, as the visualization tool SUNSHOT [11] does. The software suite Score-P[1] works with Scalasca, Vampir, and Tau and is open for other tools. Score-P uses OTF2, the CUBE4 profiling format and the Opari2 instrumenter. 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.

Regarding system I/O middleware, once again diversity abounds. The Adaptive I/O System (ADIOS) [10], 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 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 [14], 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 non-contiguous regions of data, while the latter one handles the collective I/O operations.

Comparing to ROMIO, OMPIO [3], 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 analyze. In SIOX, we present an approach to instrument all of them.

3 SIOX: scalable I/O for extreme performance

The SIOX project is organized as a collaboration of the Technische Universität Dresden—Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH), Universität Stuttgart—Höchstleistungsrechenzentrum Stuttgart (HLRS), Universität Hamburg, 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 in order to extract their causal relations and to reconstruct access patterns for automatic optimization proposals. Further, SIOX aims to provide tracking information about the internal behaviour of the different MPI I/O implementations. The integrated analysis of application and PFS and HPC hardware is a basis for optimization in other scenarios. For instance, the computing centre may identify unfavourable access patterns and suboptimal applications.

Communications and mappings will be monitored through the SIOX system in order to collect associations between system activities and to reconstruct the causal tree of connected activities. Synchronization of software clocks and hardware clocks on all nodes is a key requirement for 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 continually improving HPC systems.

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

During development, the 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.

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 in relating 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.

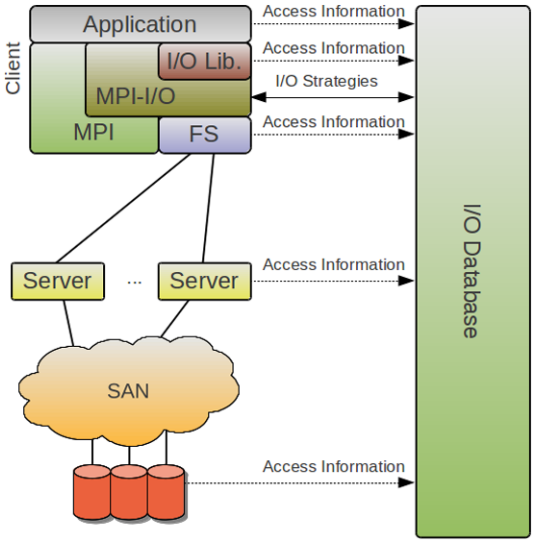

Fig. 2 Information extraction on four software layers and at two hardware locations (server and I/O nodes)

4 Architecture

The proposed architecture of the SIOX system is depicted in Fig. 3. The design allows for the system to submit optimization calls at runtime using an automatic, iterative improvement procedure.

The architecture consists of five main software components: 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. "A drop in I/O throughput was observed 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

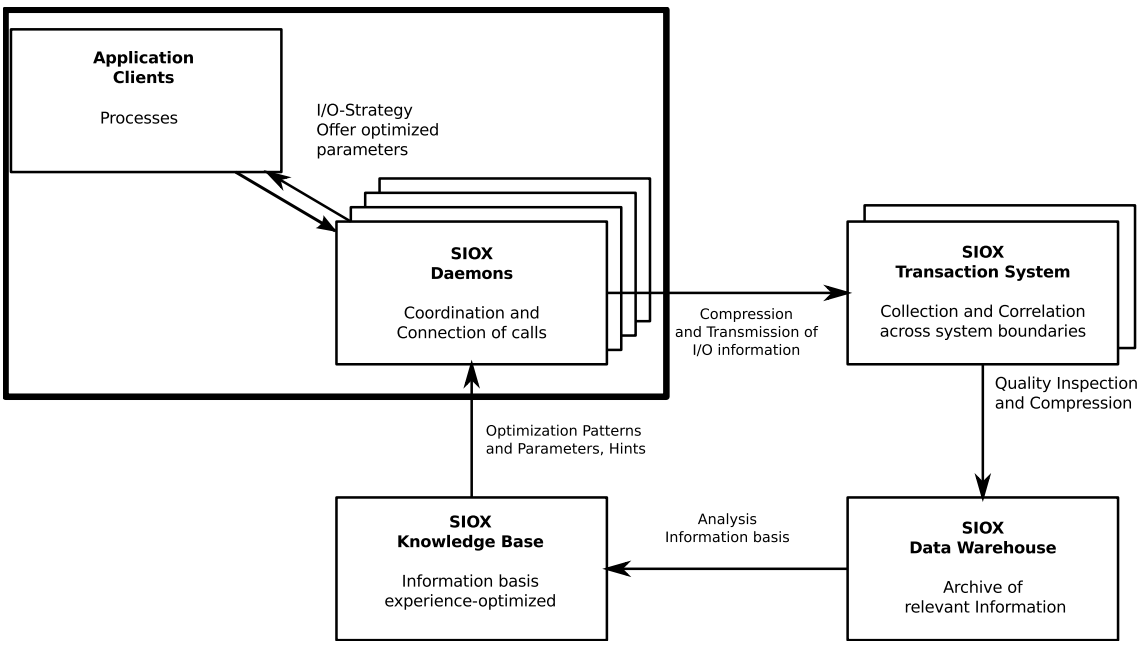

Fig. 3 Architecture of the SIOX system

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 optimizations based on expertise.

4.1 Definitions

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

Nodes are functional logical units in a parallel computing system provided by 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)* in order to distinguish instances with the same HWID and SWID.

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

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 that are defined for the identification of entities or activities in nodes and along edges.

4.2 Fine structure

Many operations in a HPC system 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 SIOX data collection subsystem will be implemented as a set of concentrator daemons, passing on the data to the SIOX server, which in turn will correlate node connections, reduce redundancies, aggregate the results and enter them into the transaction system. Optionally, the server interface can be accessed directly by a client via TCP, as shown in Fig. 4. Threads will be created for each client to serve as interface between clients and data collection via SQL over socket communication; while concentrators and server communicate via TCP or shared memory (see Fig. 5). All data storage is generally layer-independent.

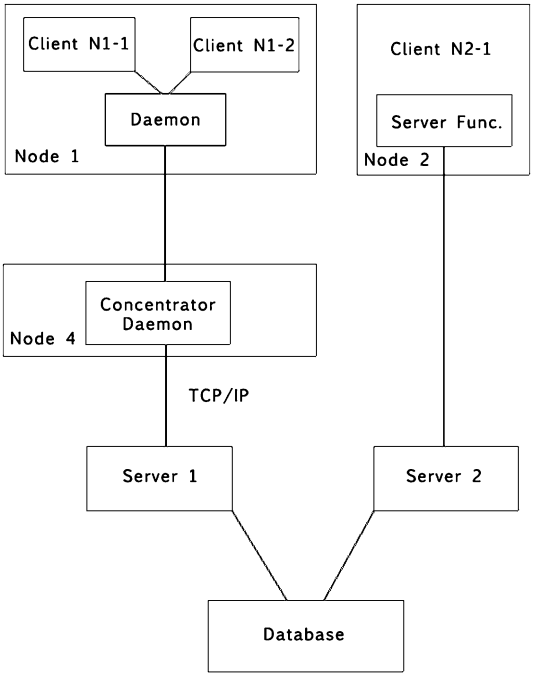

Fig. 4 Two possible data paths from client to database

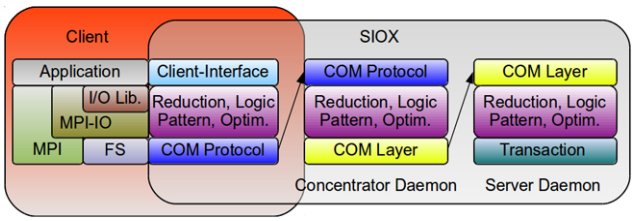

Fig. 5 Overview of the SIOX system

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 in the workflow: there is a great compression potential in the I/O traces generated by SPMD applications since they generally behave similarly. 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 a structuring according to basic functionalities following the I/O PATH MODEL [9]. Thus, focusing on abstract functionalities allows us to cover all the diversity of HPC systems.

5 Realization

Information is collected from different layers and different components of the PFS. For example, in the 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 an enterprise transaction system database (i.e. PostgreSQL).

5.1 SIOX system interfaces

We have designed a first interface of the SIOX system. It covers the functions for registering, assigning and unregistering nodes, edges and descriptor mappings, as 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 Sect. 4.3, it would be necessary to adapt one of the existing MPI trace compression techniques such as ScalaTrace [12] or compressed Complete Call Graphs (cCCG) [6] 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 for 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 labour intensive and result in high expenses beacause 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 an activity on the parent level will report significant call parameters such as descriptors and any child activity will report receiving the same parameters upon being called. 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.

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 cache to block storage. Interfaces will be provided 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.

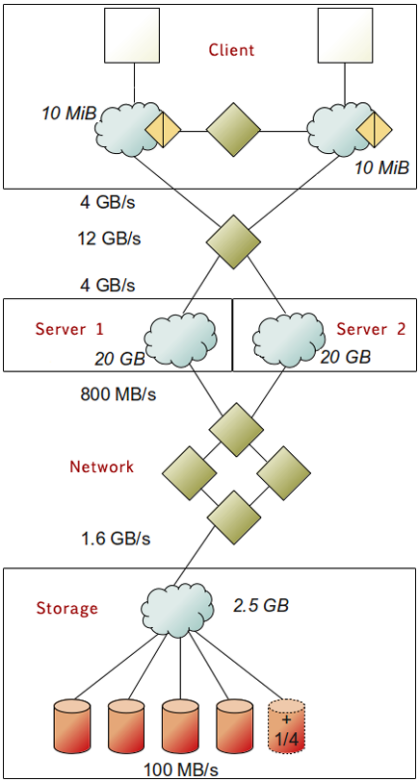

Fig. 6 The activity oriented I/O PATH MODEL [9] of two servers, connected via network to a RAID-5 storage system. The numbers represent bandwidths between nodes and normal (no read-ahead or write-behind) caches, shown as clouds with data size

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*, and *Server 1*. The SWID would be a character set such as "POSIX-I/O". An example of the registration process of two nodes instrumented for SIOX is shown in Fig. 7. Here, node A, logically representing software component Alpha running on hardware component Server 1, uses node B, representing software component Beta, also on Server 1. Additional calls to report further attributes and capabilities are omitted for clarity, see Fig. 8.

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

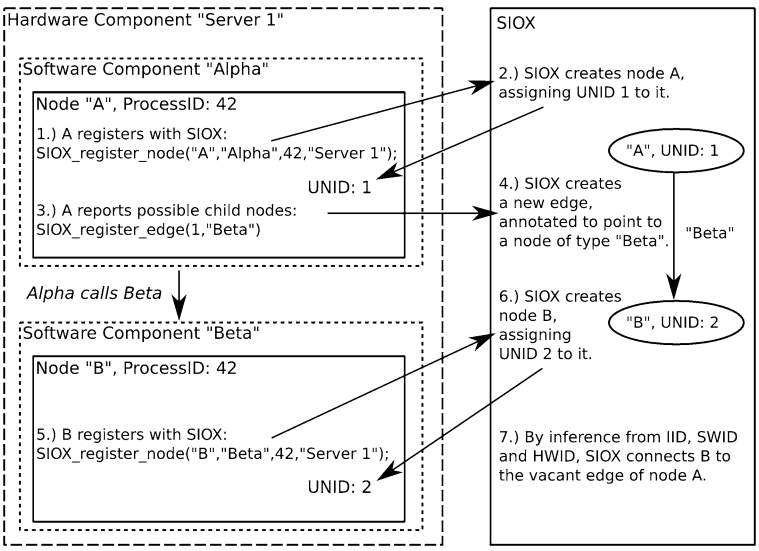

Fig. 7 Constructing the I/O PATH MODEL graph during runtime

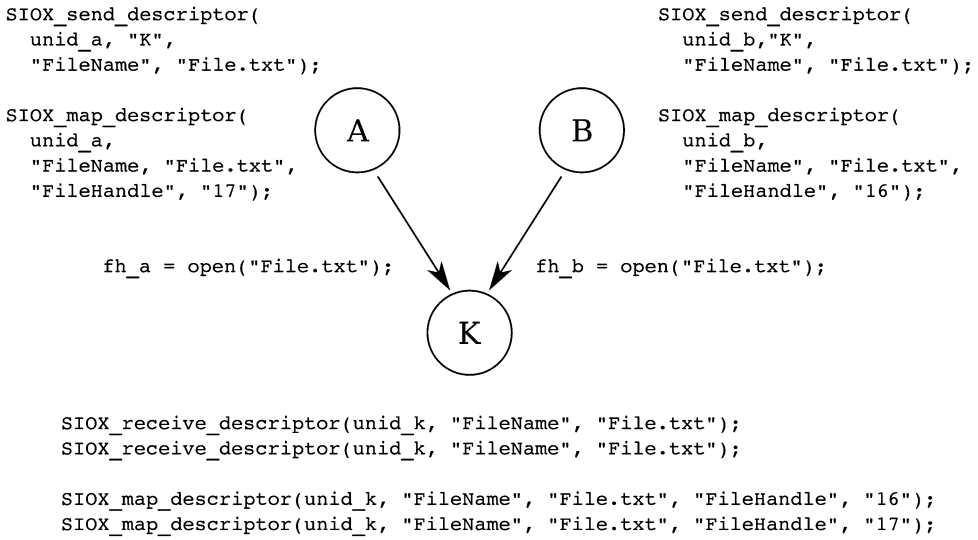

Fig. 8 Special case where two UNIDs on instrumented software layers open the same file on `unid_k`. Without reporting the descriptor mappings of `unid_ a`, `unid_b` and `unid_k`, no distinction between actions initiated by `unid_a` and `unid_b` is possible

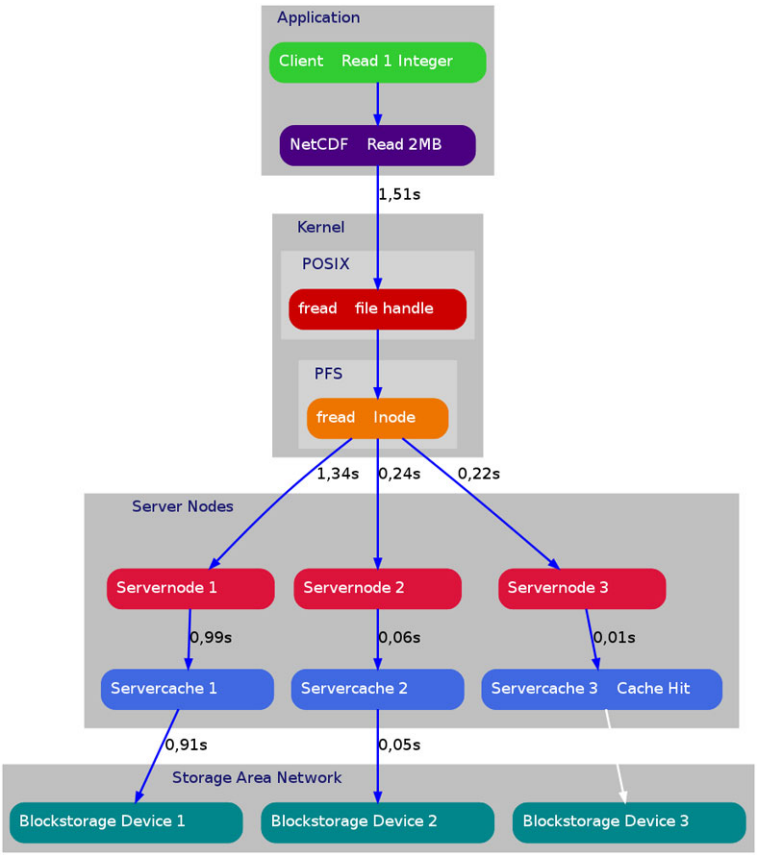

Fig. 9 A SIOX activity graph. Along the hierarchy of activities resulting from a `read()` call, total inclusive execution times are shown, pointing out the exact locations of performance problems

sub-frameworks as well as from the different modules. If ROMIO is selected, the Abstract-Device interface for I/O (ADIO) within ROMIO reads the control information of the I/O operation from the 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.

6.3 Visualizing I/O and activities

To draw meaning from spatially, logically, temporally and causally interrelated activities, the SIOX system will compute an activity graph, depicting slices of performance data

along the hierarchy of activities. Thus, bottlenecks and component performance become apparent, hinting at problems in system design or configuration.

In Fig. 9, a simplified example graph depicts activities resulting from reading a single integer variable with the `nc_ get_var_int()` call that is issued on client level. NetCDF manages a cache; all the data of the page accessed must be read. Assuming the page size is 2 MiB, this results in a single read call of 2 MiB. Between nodes representing logical functionalities according to the I/O PATH MODEL, edges are annotated with the total inclusive execution time for the pertinent activity, as any sub-calls resulting from a `read()` will return only after completion of their last child call. On the server level, there are three calls to read from block storage devices. The first takes 0.99 s to complete, the second 0.06 s and the third 0.01 s by hitting the server cache. Total execution times logged at client level will indicate a performance problem, but only the detailed reconstruction of all activities involved makes it possible to pinpoint the component at fault; in this case, the leftmost block storage device.

7 Conclusion

We have taken steps towards comprehensive solutions to all problems described in Sect. 1. SIOX's aim is to create a system capable of analyzing the I/O requests in HPC systems on a per-request base.

In this paper, we introduce the following first approach: by abstracting from the 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 the client-server daemon structure. Layers with potential optimization given specific application scenarios include applications, high-level I/O such as MPI-IO and NetCDF, HDF5, the operating system, the HPC file system and hardware locations such as I/O nodes, computing nodes and storage controllers. The next steps will include evaluation of the interfaces built, instrumentation of MPI-IO and NetCDF, and the theoretical survey for the pattern analysis phase. Also, more internal interfaces will be defined and the first implementations started.

Interested readers are kindly encouraged to become involved in the project through our internet presence.[2]

References

1. Babu S, Borisov N, Uttamchandani S, Routray R, Singh A (2009) DIADS: addressing the "My-problem-or-yours" syndrome with integrated SAN and database diagnosis. In: FAST'09: proceedings of the 7th conference on file and storage technologies. USENIX Association, Berkeley, pp 57–70
2. Barham P, Donnelly ARI, Mortier R (2004) Using magpie for request extraction and workload modelling. Microsoft Research
3. Chaarawi M, Gabriel E, Keller R, Graham RL, Dongarra JJ (2011) OMPIO: a modular software architecture for MPI I/O. Springer, Berlin/Heidelberg
4. Geimer M, Wolf F, Wylie BJN, Becker D, Böhme D, Frings W, Hermanns MA, Mohr B, Szebenyi Z (2009) Recent developments in the scalasca toolset. In: Tools for high performance computing, proceedings of the 3rd international workshop on parallel tools. Springer, Berlin
5. Hermanns MA, Geimer M, Wolf F, Wylie BJN (2009) Verifying causality between distant performance phenomena in large-scale MPI applications. In: Proceedings of the 17th Euromicro international conference on parallel, distributed, and network-based processing (PDP), Weimar, Germany. IEEE Computer Society Press, Los Alamitos, pp 78–84
6. Knüpfer A, Nagel WE (2006) Compressible memory data structures for event-based trace analysis. Future Gener Comput Syst 22:359–368
7. Knüpfer A, Brunst H, Doleschal J, Jurenz M, Lieber M, Mickler H, Müller MS, Nagel WE (2008) The Vampir performance analysis tool-set. In: Tools for high performance computing, proceedings of the 2nd international workshop on parallel tools. Springer, Berlin, pp 139–155
8. Kunkel J (2011) HDTrace—a tracing and simulation environment of application and system interaction. Tech. Rep. 2, Deutsches Klimarechenzentrum GmbH, Bundesstraße 45a, 20146, Hamburg
9. Kunkel J, Ludwig T (2011) IOPm—modeling the I/O path with a functional representation of parallel file system and hardware architecture, to be published
10. Lofstead J, Zheng F, Klasky S, Schwan K (2009) Adaptable, metadata rich IO methods for portable high performance IO. IEEE Computer Society, Washington
11. Minartz T, Molka D, Kunkel J, Knobloch M, Kuhn M, Ludwig T (2012) Handbook of energy-aware and green computing. Chapman and Hall/CRC Press Taylor and Francis Group LLC, Boca Raton, p 600
12. Noeth M, Ratn P, Mueller F, Schulz M, de Supinski BR (2009) ScalaTrace: scalable compression and replay of communication traces for high performance computing. J Parallel Distrib Comput 69:696–710
13. Shende SS, Malony AD (2006) The TAU parallel performance system. Int J High Perform Comput Appl 20(2):287–311
14. Thakur R, Gropp W, Lusk E (1999) On implementing MPI-IO portably and with high performance. ACM Press, New York
15. Thereska E, Salmon B, Salmon O, Strunk J, Wachs M, Abd-el-Malek M, Lopez J, Ganger GR (2006) Stardust: tracking activity in a distributed storage system. In: ACM SIGMETRICS conference on measurement and modeling of computer systems. ACM Press, New York, pp 3–14

[2] http://www.hpc-io.org.

Marc C. Wiedemann is a computational researcher at the German Climate Computing Center (DKRZ) and the University of Hamburg and holds a diploma in physics (thesis in theoretical simulations and experimental highest energy nonlinear optics on ultrashort fs timescales). His research focuses on the overall system analysis of parallel computing systems including I/O systems. For scientific institutes in Europe he excelled in Surface Science Analytics by optimizing instruments and methods for gaining an analytical insight to physical-chemical interaction systems. For that he accounted for the nominal transaction volume of the market leader in Germany, Denmark, Sweden, Finland and Norway. From 2009 to 2011 he worked in the function of Germany wide project management in a team renewing all weather radar systems with networked controlling Linux servers from Hohenpeißenberg in the Alps to Warnemünde.

Julian M. Kunkel received his M.Sc. degree in computer science at the University of Heidelberg in 2007. Currently, he is employed at the German Climate Computing Centre. He invests his leisure time in his doctoral dissertation at the University of Hamburg. His interests cover high performance file systems and performance modeling of cluster systems.

Michaela Zimmer holds a Diploma in Mathematics and a B.Sc. in Informatics, both from the University of Hamburg. Currently, she works there as a Ph.D. student on the SIOX project, combining her main research interests of artificial intelligence and high-performance computing.

Thomas Ludwig received his doctoral degree and the German habilitation degree at the Technische Universität München, where he conducted research on HPC from 1988 to 2001. From 2001 to 2009 he had a chair for parallel computing at the Universität Heidelberg. Since 2009 he is the director of the German Climate Computing Centre (DKRZ) and professor at the Universität Hamburg. His research activity is in the fields of high volume data storage, energy efficiency, and performance analysis concepts and tools for parallel systems. At DKRZ Prof. Ludwig takes the responsibility for accomplishing its mission: to provide high performance computing platforms, sophisticated and high capacity data management, and superior service for premium climate science.

Thomas Bönisch received his doctoral degree in 2006 from the Universität Stuttgart, Germany. He is currently head of a working group at the High Performance Computing Center Stuttgart (HLRS) and responsible for data management in the center. His current research interests include I/O architectures and I/O software for exascale systems, Scalable Parallel File Systems, I/O monitoring and I/O middleware for HPC systems.

Xuan Wang born on Mai first in 1982 in Yunnan. He studied computational science and technology from 2001 to 2005 at the Tongji university in Shanghai and from 2006 to 2011 at the university of Stuttgart. He holds a diploma in computational science. From 2011 until now he works as a scientific researcher in the field of parallel I/O. His scientific specialties are JAVA programs and GridFTP measurements.

Alvaro Aguilera works as a researcher in the field of distributed and data intensive computing at the Center for Information Services and High Performance Computing (ZIH) of the Technische Universität Dresden. He obtained his MSc in Computer Science at the same institution in 2011.

Wolfgang E. Nagel holds the Chair for Computer Architecture at Technische Universität Dresden (TUD). After his university studies of computer science at RWTH Aachen from 1979 to 1985, he worked in the area of parallel computing at the Central Institute for Applied Mathematics, Research Center Jülich, and at the Center for Advanced Computing Research (CACR), Caltech. Since 1997, he is director of the Center for Information Services and High Performance Computing (ZIH)—the former Center for HPC (ZHR)—at TUD, which were founded by him. From 2006 to 2009, he served as dean of the Computer Science department at TUD. His research profile covers modern programming concepts and software tools to support complex compute intensive applications, analysis of innovative computer architectures, and the development of efficient algorithms and methods. He published about 120 papers in those areas, and contributed as program committee member, program chair, or general chair to more than 45 conferences and workshops.

Michael Kluge is a researcher and software engineer at the Center for High Performance Computing (ZIH) at Technische Universität Dresden. His research focuses on the performance analysis of parallel file systems and the development of tools to support I/O centric analysis approaches. Michael received a Masters degree in Information Systems Technology in 2004 and a Ph.D. in Computer Science from the Technische Universität Dresden in 2011. He was part of the team winning the Bandwidth Challenge at SC'07.

Holger Mickler achieved his diploma in computer science in 2007 at Technische Universität Dresden. Since then he works at the Center for Information Services and High Performance Computing (ZIH) where he had dedicated himself to advancing tools for performance analysis of parallel programs on multi-core architectures, and making HPC administrators' lifes less tedious within the ParMA and TIMaCS research projects, respectively. Currently, he got some feathers for the SIOX project, trying to anatomize I/O processing of HPC systems as a whole with the ultimate goal of automatically increasing overall I/O performance.