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

HDTrace – A Tracing and Simulation Environment of Application and System Interaction

Julian Kunkel

23/01/2011

Abstract *HDTrace* is an environment which allows to trace and simulate the behavior of MPI programs on a cluster. It explicitly includes support to trace internals of MPICH2 and the parallel file system PVFS. With this support it enables to localize inefficiencies, to conduct research on new algorithms and to evaluate future systems. Simulation provides upper bounds of expected performance and helps to assess observed performance as potential performance gains of optimizations can be approximated.

In this paper the environment is introduced and several examples depict how it assists to reveal internal behavior and spot bottlenecks. In an example with PVFS the inefficient write-out of a matrix diagonal could be either identified by inspecting the PVFS server behavior or by simulation. Additionally the simulation showed that in theory the operation should finish 20 times faster on our cluster – by applying correct MPI hints this potential could be exploited.

Keywords Simulation · Tracing · MPI-IO · PVFS2

1 Introduction

In order to get insight about system and application behavior either all activity can be recorded and analyzed, or modeled and simulated. HDTrace is an environment which allows both kind of activity: MPI-IO applications can be traced to analyze the behavior and to spot regions which need most of the execution time. Also traced applications can be simulated in arbitrary cluster environments to increase understanding in the system and software behavior.

Julian Kunkel
University of Hamburg
E-mail: kunkel@informatik.uni-hamburg.de

This paper is organized as follows: In section 2 the software environment is introduced, which allows to trace internals of MPI and the parallel file system PVFS together with client activity. More details about the simulator are given in section 3. Sections 4, 5 and 6 provide several examples about visualization from client and server activities. A simple I/O simulation is compared with the observation in section 7. At last, state of the art is provided in section 8 and some future work is mentioned.

2 HDTrace Environment

The *HDTrace environment* is a tracing environment developed to gather information necessary for simulation. To assess simulation results and to increase insight it consists of extensions for tracing MPI internals and activity of the *Parallel Virtual File System* (PVFS). Compared to existing tracing environment HDTrace concentrates on evaluation of new ideas. It is developed under the GPL license and consists of the components shown in figure 1.

2.1 Components

The *TraceWritingCLibrary* is responsible to store events in XML trace files and statistics in a binary format with an XML description header. A project file links together all trace and statistic files of multiple sources. The format is designed to avoid post-mortem adjustment of trace files, instead, only the project file or trace file header must be adjusted. Statistics hold a group of arbitrary counters together with a timestamp. Relations between activities of different programs can be recorded explicitly. The library uses the local clock for

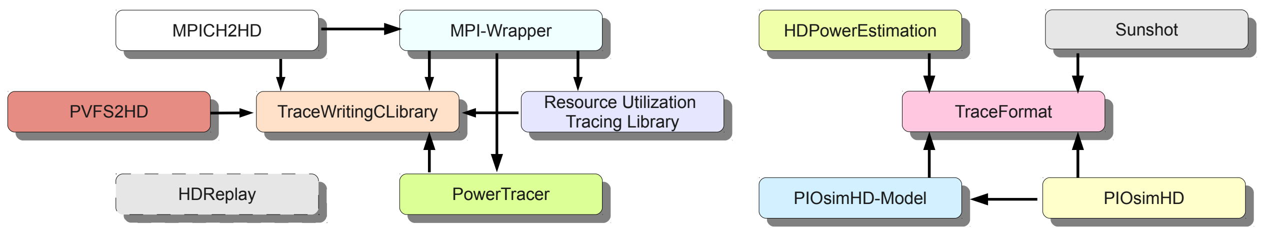


Fig. 1 HDTrace components and dependencies between different components.

timestamps, therefore, nodes must have synchronized clocks (e.g. by using NTP). Additionally offsets between timestamps of the trace files can be fixed post-mortem by adjusting the files' headers.

The *MPI-Wrapper* uses *PMPI*[1] to intercept MPI calls and the trace library to store the events. Additional information about file accesses, communicators and the mapping from application to hardware are maintained and recorded. It interfaces with the *Resource Utilization Tracing Library* and the PowerTracer.

The *Resource Utilization Tracing Library (RUT)* periodically gathers system information and utilization from the operating system, network, I/O and CPU which are stored in statistic files.

PVFS2HD contains modifications to enable tracing of PVFS activities within client and server. MPI activities from PVFS clients can be related with PVFS server activities and statistics about utilization of PVFS-internal layers are computed and stored. Therefore, server and client activity can be visualized together to understand causal relations. Note that it can use RUT to gather OS information on PVFS nodes as well. The current instrumented version of PVFS is orangefs-2.8.3-20101113.

In *MPICH2HD* the MPI implementation *MPICH2* is modified slightly to allow tracing of MPI internals for collective calls and for PVFS-internal calls. It is based on MPICH2-1.3.1.

PowerTracer is an extension to the trace environment, which periodically traces information about power usage from an external power meter in statistic files. This enables to visualize energy metrics of nodes together with their MPI activity.

HDReplay is a program which allows to replay recorded MPI-IO behavior on arbitrary environments to evaluate potential bottlenecks before the application is ported. It is currently under development.

So far all mentioned software is coded in C, the following components are written in Java and designed to support simulation.

The *TraceFormat* library provides interfaces to read and write trace files.

[1] The MPI Profiling Interface

In *PIOsimHD-Model* an abstraction layer for application traces and the cluster model is provided for the simulator. MPI commands which are read from the trace files are converted to a sequence of commands which can be executed by the simulator. Depending on the cluster model the required trace files are loaded either on demand (to deal with large trace files) or kept in memory.

PIOsimHD is the event based simulator which simulates hardware and software behavior. The simulation generates trace files of the run and internal components for further inspection.

Trace files of application, PVFS or simulation runs are visualized by *Sunshot*, a Java-Swing application, based on Jumpshot [11]. The original Jumpshot viewer is part of the MPI implementation MPICH2, which allows to visualize the *SLOG2* trace format of the *MPI Parallel Environment* (MPE).

With the *HDPowerEstimation* statistics recorded by RUT can be used to estimate the energy consumption of a node (see [8] for more information).

3 PIOsimHD

PIOsimHD is a sequential discrete event simulator written in Java. Its goal is to assist MPI(-IO) research and to support understanding performance factors in clusters. Arbitrary network topologies can be created and relevant characteristics of the components can be adjusted freely. The specification of the model can be either explicitly programmed or read from an XML file.

Internally, a discrete event simulator processes the events in the future and increments a global clock accordingly [10]. An event itself can create new (future) events. Output from the simulation can be stored in trace files and compared with the original run.

Due to limited space underlying model concepts are explained only in brief.

The hardware model reflects the common sense of a cluster computer. Several compute resources (CPUs) are hosted on a node which is connected to one or several networks via a network interface (NI). On each node one I/O server can be placed, each holds a cache layer and an I/O subsystem. A network topology defines how network edges are connected to intermediate nodes. Any network graph can be created.

Each component implementation uses characteristics to simulate hardware behavior. To cope with several levels of abstraction a component can have several implementations. During the model specification the characteristics and concrete model can be selected for each component individually. Usually, characteristics are provided in vendor specifications or obtained by benchmarking the existing system.

3.1 Software Model

PIOsimHD allows to run programs conforming to the MPI standard. Asynchronous communication and collectives of MPI-3 are supported[2]. To simulate execution of a particular MPI function at least one implementation must be provided within the simulator. Multiple implementations for a given MPI function can be programmed and selected in the model specification.

A particular MPI function has the global view of the simulation i.e. it is possible to see the state of other clients. For instance, this global world view allows to implement `MPI_Barrier()` without network communication at all – once all clients invoked the barrier the collective call finishes.

An abstract parallel file system defines how client and server interaction takes place. File data is partitioned among all servers as defined by a selectable distribution function. Metadata operations are not considered. Clients and servers interact in a similar fashion to the PVFS model. In the write path a client requests write operation from the server and then starts to transfer all data. In the simulator file sizes are updated once a write operation finishes. Non-contiguous I/O requests are supported.

3.2 Simulation Workflow

The workflow to increase insight in the interplay between system and applications is shown in figure figure 2. Components of *HDTrace* involved in the different steps of the process are included.

In general there are two ways to simulate application activities in *PIOsimHD*: either a running application is instrumented to generate trace files, or the communication and I/O behavior can be coded explicitly in the form of Java programs.

[2] Note that asynchronous collective break in case of multiple operations of the same type are processed on one client.

The figure depicts the case in which an MPI program is instrumented. In this case the MPI-wrapper is linked with the library, all MPI(-IO) activities are intercepted by the *PMPI* wrapper and events recorded by using the *TraceWritingCLibrary*. In case PVFS is used as underlying file system, then additional traces for the client and server activities can be included by enabling *HDTrace*.

Traces of the application and optionally PVFS can then be visualized by *Sunshot*. To perform simulation the model of the cluster must be created by the user and then applications to simulate must be mapped to available nodes. Note that the user can simulate concurrent processing of multiple applications at the same time to stress network and I/O infrastructure. *PIOsimHD-Model* reads this model and the application trace files required for simulation and feeds them to the simulator. *PIOsimHD* performs the discrete event simulation and if requested outputs simulation processing by using the Java trace library. Results of simulation can then be visualized by *Sunshot*. Comparison of the recorded application (and PVFS) activities and simulation results can be done by the user.

Instead of instrumenting a real application, helper classes in *PIOsimHD-Model* allow to explicitly program the cluster model and application behavior. This can be used to perform small tests of I/O systems or MPI-internal communication.

4 Tracing Energy Metrics

Next, a few excerpts of trace analysis are given to demonstrate the strength of the environment.

First Sunshot is explained on an excerpt of a run of the HPC Challenge (HPCC)[3], which includes energy metrics and client activity (figure 3). On the top row icons for user interaction are provided, below information is given about the current selected time interval and the event/statistic which is below the mouse pointer. In the left a tree view visualizes the mapping of the metrics and traces to nodes – in this case *hpcc* was run on node06 to node09, process 0 and process 4 are run on node06 and the energy metrics (I, P, U) belong to node06. Right of the tree view the activity and statistics for each timestamp are drawn – white areas in the process activity correspond to computation on the client processes, the colors encode calls to MPI (communication) functions. The statistic metrics compute the maximum independently for each timeline. Uninteresting timelines or statistics can be removed from the view to dig into the issues.

[3] http://icl.cs.utk.edu/hpcc/

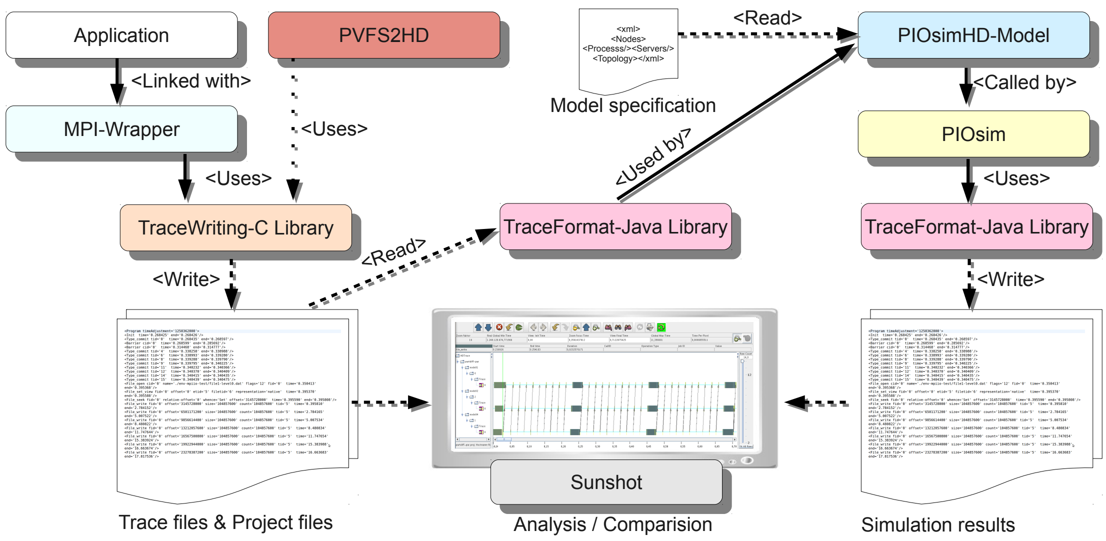


Fig. 2 System analysis workflow of PIOsimHD. In the example an application run serves as input for the simulation.

From this view a slight fluctuation of the power consumption can be observed between different nodes. During the broadcast operation (purple operation on the right) the power consumption is lower.

5 Tracing MPI Internals

Visualization from tracing an `MPI_Bcast()` with 100 MiB to 8 processes is provided in figure 4. Each process is located on a separate node equipped with Gigabit Ethernet and two Intel Westmere processors. More details about observed network characteristics are given in section 7. The screenshot from *Sunshot* is modified to visualize communication size and partners for process with rank 0.

At the beginning rank 0 sends 50 MiB to rank 4, then 26 MiB to rank 2, 13 MiB to rank 1, then 7 times 13 MiB are sent to rank 1 and received from rank 7.

From observing the pattern the sub-optimality of the pattern becomes clear. Internally, with the default configuration on our cluster MPICH2 performs a binomial tree scatter followed by an allgather operation, which explains the observed pattern.

6 Tracing MPI and PVFS Interplay

Tracing parallel file system activities together with MPI activities allows to spot bottlenecks which are hidden otherwise.

In this experiment our parallel PDE solver for a 2-dimensional problem (`partdiff-par`) is instrumented with HDTrace. The PDE solver allows to periodically

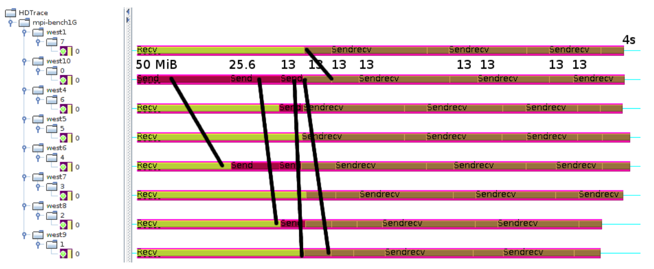


Fig. 4 `MPI_Bcast()` of 100 MiB of data to 8 processes.

store diagonals of the matrix in a file by using MPI-IO. This *progress* information then can be read-out during the processing to look at the convergence of the algorithm. Full matrices are written after a configurable number of iterations to allow checkpoint/restart and to look at the values of the matrix at fixed numbers of iterations.

In this (artificial) problem the matrix has a dimension of about 8000x8000 double values, which corresponds to a resident set of about 450 MiB for a single matrix. The application is run on only one processor. Every 5 iterations a checkpoint of the full matrix was made and the progress information data (diagonal is 64 KiB) are written in every iteration. The PDE uses `MPI_File_write_at()` to store data (without setting a file view), therefore access on disk is always contiguous. Access to the matrix diagonal in memory is performed by application of an derived datatype.

MPI and PVFS activities are provided in figure 5 and figure 6. In the first screenshot an overview of 20 iterations of the solver is provided. On top the client MPI activity including the long duration of the check-

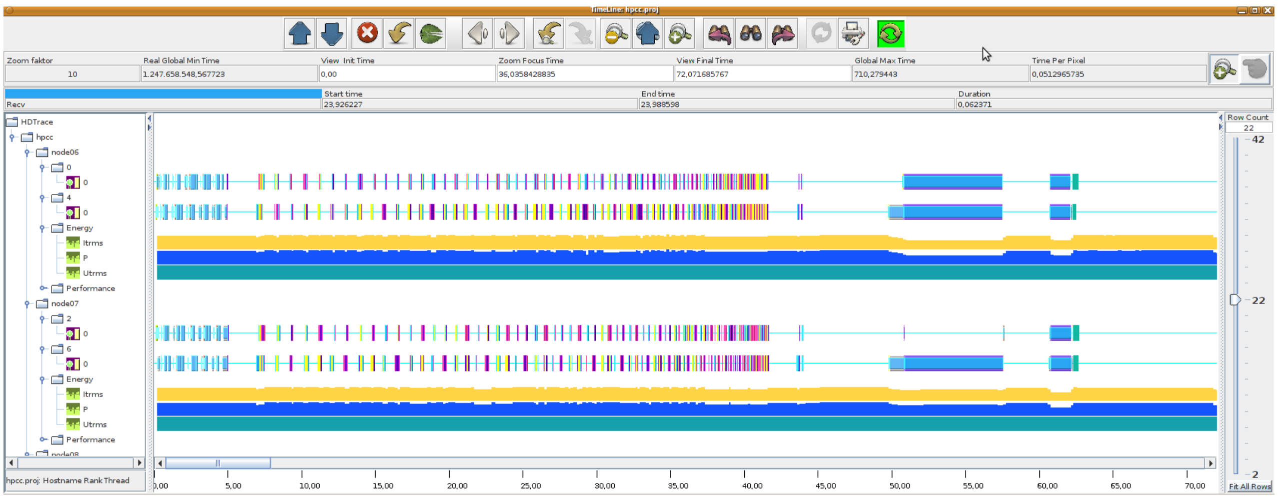


Fig. 3 Sunshot: Timelines of two nodes performing the first phase of the HPCC-run – including energy metrics.

pointing can be seen, computation is not shown explicitly. Below, the PVFS activity of one server is visualized. Each layer has events and/or statistics associated with the activities. Statistics encapsulate the number of concurrent operations of the particular layer, this information is not sampled. Therefore, at each point in time the value accurately represents the number of pending operations. *BMI* is the network activity, *FLOW* regular I/O operations (very small operations are not included), *REQ* are the number of outstanding requests. *SERVER* shows the pending statemachines and current step within each statemachine (here we can see performed activity by write_sm). *TROVE* is the persistency layer of PVFS. In TROVE not only concurrent operations are traceable, also each individual I/O operation which is given to the operating system can be traced with offset and size. Overlapping concurrent operations will be expanded automatically to multiple timelines by Sunshot. In the example up to 8 concurrent operations are observed, note this is the maximum number of outstanding operations FLOW enforces per request. Additionally several OS-related utilization values are given (the folder is called "Utilization" on the left), the CPU utilization, write bandwidth of the storage and network activity.

During the several checkpoints of the application one can follow the server operation in detail. Interestingly I/O operations are mostly performed sequentially. In fact it turns out the network configuration of the machines and switch degrades network bandwidth to 77 MiB/s while the disk has a performance of about 100 MiB/s, therefore the bottleneck is the network in this configuration. This can be observed also by looking at the BMI statistics, mostly network data is requested (BMI > 0) but at the short phases of stalled I/O the operations queue up on the TROVE layer. Effectively

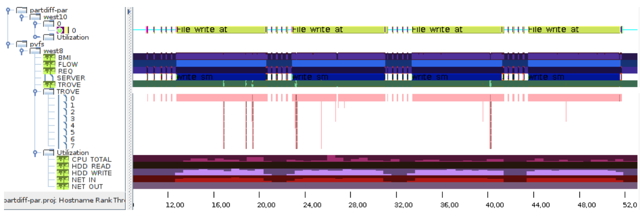


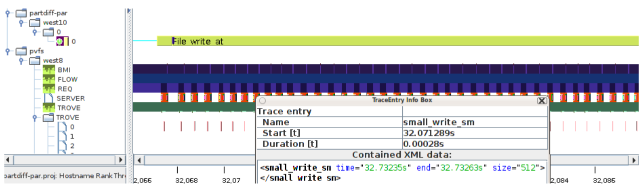
Fig. 5 Traced 2D-PDE solver and PVFS – first 10 iterations.

the FLOW layer waits for completion of outstanding operations before additional data can be transferred.

If we look at the process of writing the progress information of 64 KiB in figure 6 then we can see that many requests are observed on the SERVER, these are all small I/O operations. In fact writing the PDE progress generates 125 small I/O operations with a size of 512 Bytes and one with 72 byte.

This is due to the non-contiguous datatype in memory, MPI-IO does not use an additional buffer to store the data, internally PVFS allows to encapsulate a list of up to 64 non-contiguous operations with one request. PVFS is also aware of memory and file datatypes, however this feature must be explicitly enabled by hints like *romio_pvfs2_listio_write*, when this hint is enabled the time to write out the matrix diagonal is reduced from an average of 69 ms to 3.4 ms.

Activities from the PVFS client library are not included in these screenshots, but resemble the startup of multiple small I/O operations on the client side as well.



Trace entry	
Name	small_write_sm
Start [t]	32.071289s
Duration [t]	0.00028s

Contained XML data:

```
<small_write_sm time="32.73235s" end="32.73263s" size="512">
</small write sm>
```

Fig. 6 Traced 2D-PDE solver and PVFS – zoomed write of the progress information.

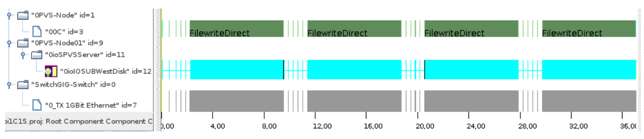


Fig. 7 Simulated processing of the traced 2D-PDE solver, the simulator used the trace from figure 5. The client activity is on the first timeline, the next one shows disk activity the last timeline the incoming network packets.

7 Simulating MPI-IO

In this experiment the trace from section 6 is fed into the simulator and the results of the simulation are compared to the observed behavior.

In the cluster model the characteristics and topology of the cluster are used. Two nodes are interconnected with a switch. One hosts the client process while the other one hosts the I/O server.

In detail the following characteristics were used: The node is equipped with 12000 MiB of memory and 1 processor (processor count is irrelevant in this test). The server disk uses an average seek time of 10 ms and a track-to-track seek time of 1 ms, RPM is set to 7200. Within 5 MiB of the last offset the track-to-track seek time is used to approximate the access time. A transfer rate of 100 MiB/s is selected[4]. The network card throughput is limited to the memory bandwidth of 7629 MiB/s[5]. A network link has a latency of 0.2875 ms and a bandwidth of 67 MiB/s[6]. The switch uses a store-and-forward architecture and has a total bandwidth of 48 GiB/s, because it has full-bisection bandwidth.

Now we can compare simulation and observation. Visualization of the whole program run is provided in figure 7. Activities of client, incoming network packets and disk are given. As the server model starts to write data out once a single packet arrives, all three components look synchronized in this screenshot. The startup of writing a checkpoint is displayed in figure 8. Here, at the beginning an average-seek operation is performed on disk, because the server wrote out progress information before (which is a different file).

During this startup the network is faster, the server caches I/O operations and then writes large chunks out in I/O operations. After a few writes the sequential write performance allows the disk to catch up with the network performance. Ultimately the disk reveals idle periods while the network is saturated. In our case the network limitation is clear by the model definition and can be identified by the screenshot. However, the identified behavior during the startup phase might be surprising.

In figure 9 write-out of a single process information is zoomed in. Here 4 bytes of data are written first to store the current iteration number, then the matrix diagonal (64 KiB) is written to disk. In the screenshot the deferred write of the disk and latencies between the data transfer of client, network and disk can be observed.

A few statistics about simulation results and observed behavior: the simulation takes 37.1 s while the original run is 46.8 s. An I/O checkpoint takes between 8.17 s and 10.6 s in the original program and always 7.3 s in the simulation (due to the fact that the server is utilized by only one client). On the client simulation shows that writing out progress information could require 3.01 ms for the 64 KiB and 1.02 ms for the first 4 bytes. Measured times were between 63 ms and 78 ms with an average of 69 ms, showing that this large discrepancy could be identified by simulation as well. In the measured run in which the hint for non-contiguous I/O was set, the observed time was between 3 ms and 3.5 ms – close to simulation results.

One question which might come to mind when an experiment is designed, is how much an I/O scheduler can improve performance. Simulation shows no benefit in this case due to the network bottleneck. However, during the startup of a checkpoint phase 22 network packets (each 100 KiB) could be aggregated to reduce the number of I/Os. In total the disk which aggregates I/Os performs 20482 I/O operations.

Of course the situation changes if multiple clients access one server concurrently, which then causes more seeks (depending on the I/O scheduler).

8 State of the art

Popular performance analysis tools/environments are Tau [9], Vampir [4] and Scalasca [2]. Yet, none of them trace MPI and parallel file system together. However, a recent funded project aims to extend TAU towards

[4] The transfer rate was measured by using `dd` of 2 GiB and 512 MiB of free memory.

[5] The value was obtained with our memory benchmark which measures the time for large `memset()` operations.

[6] Latency is measured with our MPI latency benchmark (`mpi-latency-bench`) and bandwidth with `iperf`.

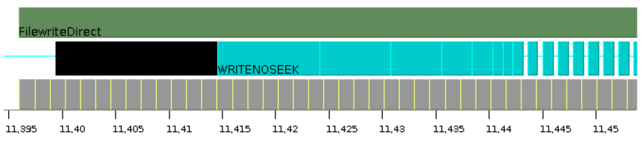


Fig. 8 Simulated processing of the traced 2D-PDE solver – zoom into the beginning of writing a checkpoint.

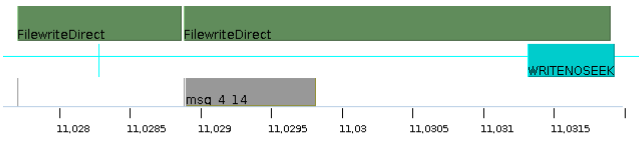


Fig. 9 Simulated processing of the traced 2D-PDE solver – zoom into the writing of file progress information.

this goal [1]. The Open Trace Format (OTF) is widely used in performance analysis tools, however when we designed our trace format it lacked the ability to add information about data types and the arguments of MPI functions. Support to add arbitrary data was added recently [3].

Our previous tracing environment (PIOviz [7]) was already used to trace MPI internals and localize bottlenecks in MPI and PVFS (see [6]). However, the new environment HDTrace extends capabilities to a new level.

Our work of HDTrace is different because as far as we know no other tracing environment allows to trace and visualize MPI internals, parallel file system activity and MPI activity together. PIOsimHD honors parallel I/O and allows to replay recorded MPI traces on a high level of abstraction. With its help an analysis of several I/O schedulers and collective I/O variants has been performed in [5]. The event-driven nature of PIOsimHD allows to localize network congestion and evaluate I/O optimization on client, server or disk side.

9 Summary and Future Work

HDTrace is an environment which allows to trace and simulate the behavior of MPI programs on a cluster and parallel file system. This allows to identify inefficiencies, to conduct research on new algorithms and to evaluate future systems. Several examples depict how the environment assists in revealing internal behavior and spotting bottlenecks. With HDTrace observations can be compared with simulation results which allows to assess the observations. In the MPI-IO example the simulation showed that in theory writing out the matrix diagonal could finish in 3 ms, which reveals a 20 fold speedup on our cluster – by applying correct MPI hints this potential could be exploited.

The following near future activity is planned: A release of the GPL-licensed software to the public. Accurate implementation of current collective algorithms and comparison with cluster-aware algorithms. Then the simulation model will be revalidated by comparing observations with simulation results. Further experiments of I/O and client interaction will be conducted.

References

1. Jason Cope Kamil Iskra Sam Lang Kwan-Liu Ma Chris Muelder Robert Ross Carmen Sigovan, P.B.: System Software Instrumentation to Support the Visual Characterization of I/O System Behavior for High-End Computing. Poster (2010)
2. Geimer, M., Wolf, F., Wylie, B.J.N., Becker, D., Böhme, D., gs, W.F., 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)
3. Knüpfer, A., Geimer, M., Spazier, J., Schuchart, J., Wagner, M., Eschweiler, D., Müller, M.S.: A generic attribute extension to OTF and its use for MPI replay. Procedia Computer Science **1**(1), 2109–2118 (2010). Proc. of the International Conference on Computational Science (ICCS)
4. 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)
5. Kuhn, M., Kunkel, J., Ludwig, T.: Optimizations for Two-Phase Collective I/O Submitted to ISC 2011
6. Kunkel, J., Tsujita, Y., Mordvinova, O., Ludwig, T.: Tracing Internal Communication in MPI and MPI-I/O. In: International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT, pp. 280–286. Hiroshima University, IEEE Computer Society, Washington, DC, USA (2009)
7. Ludwig, T., Krempel, S., Kunkel, J., Panse, F., Withanage, D.: Tracing the MPI-IO Calls' Disk Accesses. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface, no. 4192 in Lecture Notes in Computer Science, pp. 322–330. C&C Research Labs, NEC Europe Ltd., and the Research Centre Jülich, Springer, Berlin / Heidelberg, Germany (2006)
8. Minartz, T., Kunkel, J., Ludwig, T.: Simulation of power consumption of energy efficient cluster hardware. Computer Science - Research and Development pp. 165–175 (2010)
9. Shende, S.S., Malony, A.D.: The Tau Parallel Performance System. Int. J. High Perform. Comput. Appl. **20**(2), 287–311 (2006)
10. Wolfgang Kreutzer, B.P.: The Java Simulation Handbook: Simulating Discrete Event Systems with UML and Java. Shaker Verlag (2005)
11. Zaki, O., Lusk, E., Gropp, W., Swider, D.: Toward Scalable Performance Visualization with Jumpshot. High Performance Computing Applications **13**(2), 277–288 (1999)