Visualization of PVFS2 Server Activities (Extended Abstract)

Thomas Ludwig, Stephan Krempel, Julian Kunkel,
Frank Panse, Dulip Withanage

Ruprecht-Karls-Universität Heidelberg
Im Neuenheimer Feld 348, 69120 Heidelberg, Germany,
t.ludwig@computer.org,
WWW home page: http://pvs.informatik.uni-heidelberg.de/

Abstract. With parallel I/O we are faced with the situation that we do not have appropriate tools to get an insight into the I/O server behavior depending on the I/O calls in a parallel program. We present an approach that enhances a trace-based tool in a way to show us client and server behavior in relation to each other. We focus on the parallel file systems PVFS2 and the MPI tool Jumpshot.

1 The Problem of I/O Visualization

Parallel I/O is a complex concept. It involves client processes which usually form a parallel program using MPI for message passing and MPI-IO for disk access. Every I/O call refers to a set of servers which provide a parallel file system for persistent data storage. Usual concepts deploy a striping scheme (i.e. RAID-0 level) to distribute the data of a logical file onto physical files on several disks. Distribution functions control where the data goes to.

The mapping of clients and servers onto the nodes of a cluster is crucial for the overall performance we can get from I/O-bounded applications. There are tools that visualize the performance of the I/O-servers, like e.g. Karma that comes with PVFS2. The problem is however, that there are no tools that can visualize the servers' activities in depence of the clients' I/O calls. We would like to see the relations of the two of them in order to improve certain aspects of parallel I/O. First, the I/O system developers can use such a tool in order to improve internal concepts of their middle-ware. For application programers it will be interesting to see the effects of their I/O calls with respect to server activity.

As for MPI the idea behind the I/O API was to make it similar to the message passing API. Thus, reading is like receiving and writing like sending. Regular tools show the relations of sending and receiving e.g. with arrows in a trace visualization tool. We would like to have the same for reading and writing and the corresponding system activities.

2 The Approach

Our task can easily be described as follows: In order to have a combined view onto client and server activities we must be able to get traces during program runtime from both of them. In order to see the server activities induced by an MPI-IO call we must related the two traces. Visually this will be done by adding arrows between the events of clients and servers. This sounds pretty easy, however, there are several problematic issues involved that need complex solutions.

Double Trace Generation We need two event traces, one from the parallel program, the other from the parallel file system. Usually the parallel file system client library gets linked to the application program. Thus, its activities can be traced via the processes of the parallel program. With MPICH tracing comes for free and is handled in the MPE component. Also the viewer is included. MPICH tracing is based on the SLOG2 trace file format and the visualization is done via the Jumpshot tool.

We use the PVFS2 parallel file system for I/O. The trace support is rudimentary and not fully implemented. We enhanced the tracing management of PVFS2 and change the way of logging. PVFS2 uses the MPICH/MPE tracing facility to get a trace for each server. To get a combined trace for all servers we start them as an MPI program.

I/O Server Events The I/O servers have a complex internal structure and there are several places in the code where a tracing could be advantageous. From all the software layers in PVFS2 we will at the moment concentrate on the Trove layer. It is responsible for the disk access and thus is related to the load of physical I/O. The tricky issue here is that one client request causes several activities in several servers that will overlap with other activities from other requests. The orders of execution might be changed and the requests trigger different events in different layers. It is of extreme importance to always keep track of what event belongs to which request.

Trace Merging Both traces have now to be merged. The program trace is limited in its extend whereas for the servers' trace we have to provide a mechanism to select the correct section of all traced events. Remember that they act as daemons, i.e. have a start and stop time that is not related to that of the MPI program. The two corresponding parts found we have to merge them into a single trace. In addition we need some adaptation of the time lines of the clients' and the servers' trace. We end up with a single trace that has as many horizontal time lines as we had client and server processes.

This combined trace can already be visualized by Jumpshot. The tool shows just the horizontal line with their events and does not distinguish between lines that belong to user processes and to I/O server processes.

Relate Abstraction Layers We want to see the relation between program calls and server activities by drawing arrows between the corresponding events. For this to

be possible we need the appropriate information to be added into the traces. The concept is as follows: For every MPI-IO call issued we generate a unique call ID and make sure that this information is also written to the program trace. When passing requests from the client to the server these IDs have to be transferred, too. Normally the server is not interested in the MPI-IO call the request was generated from, however, now the situation changes. For every server activity of interest that gets recorded in the trace we need to know to which MPI-IO call it belongs. Thus, the call IDs need to be written to the server trace together with the regular event informations. After having the traces merged we can now easily detect corresponding events from clients and servers. Conceptionally this is simple, however, the modification of two environments like MPICH/MPE and PVFS2 is a challenge. Obviously the changes would be easy to do while doing a redesign of at least PVFS2 together with its client server request protocol.

Beautifying the Output Finally we see client and server events and many arrows in our trace visualization. Due to the asynchronous nature of the server we have many overlaps with the server events. We added another tool that distributes the overlapping events from one timeline onto as many timelines as are needed in order not to have any more overlaps. This results in a nice visualization.

3 A First Example

A first example is shown in the following screen dump. The program that produced these events is just a simple MPI program running in two processes and doing write calls. On the left side you see the information that Jumpshot displays in a pop-up window when you click on the MPI call and the corresponding arrows. The right window shows the Trove events (i.e. disk write activities) triggered by an MPI call in another process. We display here only write events on the client and server side. All other events are supressed. We can see clearly, that a single MPI-IO cal results in a set of Trove calls triggered. Other PFVS2 server activities are recorded, too and give a deeper insight into the servers' activities. The full paper will give more details on this.

4 Related Work

A related works section will only be integrated in the full paper. As for the functionality provided by our environment there is no other visualization tool available that serves for the same purpose. There are however tools with related functionality.

5 Current Work

The complete environment is already functional. All components are implemented and interact, however, we still need some time for tuning the components and intensive tests with applications.

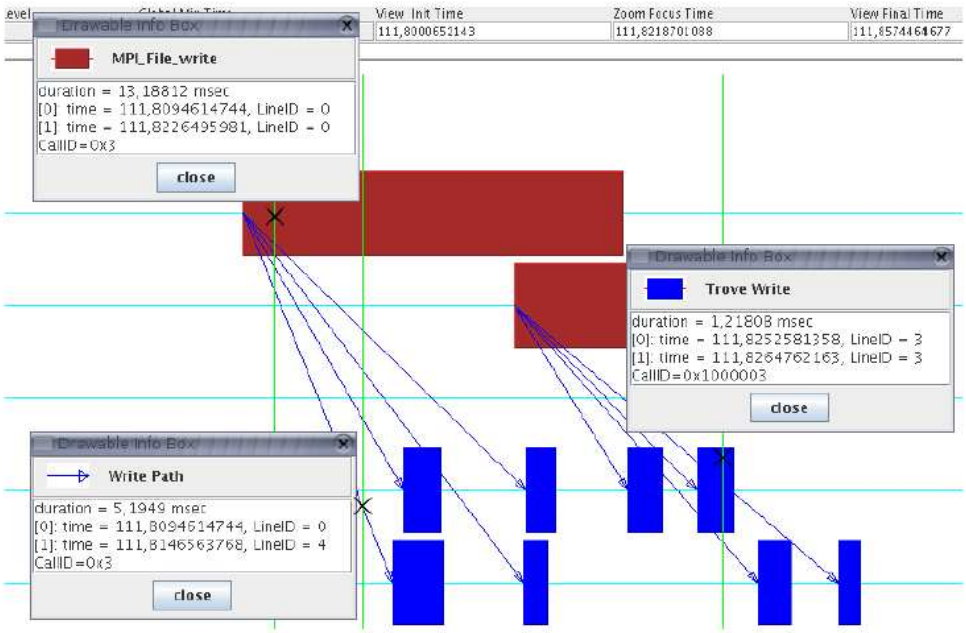

Fig. 1. Two MPI processes invoke write calls (upper two lines) and thus induce trove acticities in the PVFS2 layers (lower two lines). Arrows show the relations between these events.

References

1. Withanage, Dulip: Performance Visualization for the PVFS2 Environment, Bachelor's Thesis, November 2005, Ruprecht-Karls-Universität Heidelberg, Germany.
2. Krempel, Stephan: Tracing Connections Between MPI Calls and Resulting PVFS2 Disk Operations, Bachelor's Thesis, March 2006, Ruprecht-Karls-Universität Heidelberg, Germany.
3. Panse, Frank: Extended Tracing Capabilities and Optimization of the PVFS2 Event Logging Management, Diploma Thesis, to be submitted, Ruprecht-Karls-Universität Heidelberg, Germany.
4. Ludwig, Thomas:. Research Trends in High Performance Parallel Input/Output for Cluster Environments, Proceedings of the 4th International Scientific and Practical Conference on Programming UkrPROG2004, Pages 274-281, National Academy of Sciences of Ukraine, Kiev, Ukraine, 2004.