Trace-based Visualization as a Tool to Understand Applications' I/O Performance

Rodrigo V. Kassick, Francieli Boito, Matthias Diener,
Norton Barbieri, Philippe O. A. Navaux,
Claudio Schepke, Nicolas Maillard
Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, RS, Brazil
{*rvkassick, fzboito, mdiener,*
nlbarbieri, navaux, cschepke, nicolas}*@inf.ufrgs.br*

Yves Denneulin
Laboratoire d'Informatique de Grenoble (LIG)
Montbonnot-Saint Martin, France
yves.denneulin@imag.fr

Carla Osthoff, Pablo Grunmann, Pedro Dias
Laboratório Nacional de Computação Científica
LNCC
Petrópolis, RJ, Brazil
{*osthoff, pablojg, pldsdias*}*@lncc.br*

Jairo Panetta
Centro de Previsão de Tempo e Estudos Climáticos
CPTEC/INPE
Cachoeira Paulista, SP, Brazil
jairo.panetta@cptec.inpe.br

Abstract*—This paper presents the use of trace-based performance visualization of a large scale atmospheric model, the Ocean-Land-Atmosphere Model (OLAM). The trace was obtained with the libRastro library, and the visualization was done with Pajé and TRIVA. The use of visualization aimed to analyze OLAM's performance and to identify its bottlenecks. Especially, we are interested in the model's I/O operations, since it was proved to be the main issue for the model's performance. OLAM's poor I/O performance is due to the generation of a large number of small files. Therefore, PVFS was used in our analysis since it has optimizations for the small-files situation. We show that ...** ***TODO (completar aqui quando tiver resultados)

***Keywords*-Atmospheric Models; Parallel File Systems; OLAM; PVFS; I/O Performance Evaluation; Performance Visualization; libRastro; Pajé, TRIVA, HDF5**

I. Introduction

High Performance Computing (HPC) applications, such as weather forecast and DNA sequencing systems, generate large amounts of data which are shared by all computing instances. Usually, Parallel File Systems (PFS) are used to allow this sharing of data in an efficient way. Since the performance of input/output (I/O) operations performance depends on comparatively slow components such as disks and interconnections, they are often considerably slower than processing. In this scenario, the performance of the applications ends up being limited by the performance of its I/O operations.

The performance that an application will obtain depends on several factors, such as its data access patterns, concurrency on the file system, number of data servers, optimizations of the file system and I/O libraries. Since there are a lot of impacting factors, it is often very difficult to understand why the I/O performance is poor. Because of that, it is also difficult to identify the changes that can be made in order to improve the I/O performance.

This paper proposes the use of trace-based visualization in order to understand the I/O performance of applications. As an example, we used the *Ocean-Land-Atmosphere Model (OLAM)* [1] on the *PVFS parallel file system* [2]. Execution traces were obtained with a trace library, libRastro [3], which we integrated into the source code of OLAM and PVFS. The traces were then processed and visualized with Pajé [4] and TRIVA [5].

Numerical models have been used extensively in the last decades for atmosphere modeling to understand and predict climate and weather phenomena. High performance implementations of these models are fundamental for the activities of weather and climate prediction, since increasing performance results in higher achievable resolution and accuracy [6].

One characteristic of most weather and climate forecast models is that the data generated during the execution is stored on a large amount of small files. This has a large impact on the scalability of the system, specially when executing on parallel file systems: the large amount of metadata operations for opening and closing files, allied with small read and write operations, can transform the I/O subroutines in a significant bottleneck, limiting the system's scalability [7].

The remainder of the paper is organized as follows: Section II presents OLAM and its I/O properties. Section III describes the tools used in this work and their integration with OLAM and PVFS. The results of the visualization are shown and analyzed in Section IV. Section V discusses some related works, and Section VI summarizes conclusions and presents future directions.

II. The Ocean-Land-Atmosphere Model (OLAM)

This section presents the *Ocean-Land-Atmosphere Model* (OLAM) [1], developed at Duke University. The model aims to represent both global and local scale phenomena, as well as bi-directional iterations among regions on different scales. This is achieved by a global grid that can be locally (statically) refined on given points of interest. OLAM was developed to extend features of the *Regional Atmospheric Modeling System* (RAMS) [6] to the global domain. It uses many of RAMS' concepts, such as physical parametrization and I/O formats. OLAM was developed in FORTRAN 90 and parallelized using the Message Passing Interface (MPI).

The model consists of a global geodesic grid with triangular mesh cells with local refinement capability, the full comprehensible non-hydrostatic Navier-Stokes equations, a finite volume formulation of conservation laws for mass, momentum and potential temperature, and others. While the global domain expands the range of atmospheric systems and scale interactions that can be represented in the model, local refinements can be specified to cover areas with more resolution through recursion.

OLAM's grid structure is organized as a mesh of spherical triangles, where the triangles' arcs follow great circles on the sphere. This grid offers advantages over the commonly used latitude-longitude grid, such as an approximately uniform mesh size over the globe. The grid begins from an icosahedron inscribed in the spherical earth. An icosahedron is a regular polyhedron composed of 20 triangle faces, 30 triangle edges and 12 vertices, with 5 edges meeting at each vertex. In order to construct a mesh of higher resolution, the division of each triangle happens in an uniform fashion into N^2 smaller triangles. All new vertices and all edges are then projected radially outward to the sphere to form geodesics.

OLAM is an iterative model, where each timestep may result in the output of data as defined in its parameters. The model's workflow is illustrated in Figure 1. In order to simulate a typical analysis environment for this case study, each triangular face of the icosahedron was subdivided into 18^2 parts, which results in a resolution of 280.5 km. The atmosphere was divided into 28 layers. The simulation is organized in timesteps of integration of 60 seconds. The total simulation calculates 24 hours of integration of the equations of atmosphere dynamics.

For this case study, OLAM requires reading 4.6 MB of input files and initial conditions. The amount of written data varies with the number of processes running the simulation. In the output phase, each process creates a history file, writes its atmospheric state and closes the history file. Therefore, the number of independent output files increases with the the number of processes. The history files are considered to be of small size for the standards of scientific applications: file size ranges from 100 to 600 KB, depending on the grid definition and the number of processes. This leads to an access pattern of "large amount of small files" that comes with a great cost in terms of I/O performance: the overhead of file creation and the small size of the writes causes the file system to perform poorly.

The *Parallel Virtual File System* (PVFS) is a good choice of parallel file system for this situation since it includes optimizations for small files [8]. However, despite such optimizations, OLAM's I/O performance on PVFS is still limited severely[7], [9].

In order to avoid the overhead imposed by the creation of a large number of files, an alternative version of OLAM was recently developed which uses OpenMP in addition to MPI. In this version, the MPI processes create OpenMP threads at the start of each timestep and destroy them before the output of the results. Therefore, this version (in this paper called *OLAM-OMP*) uses a different level of parallelism and is generating less files. This happens because it maintains the same degree of parallelism, but with a smaller number of MPI processes (each of them creating a number of OpenMP threads). As each output file correspond to one MPI Rank, the total number of generated files decreases compared to the previous version of OLAM. In a previous work [7] we have shown that generating 8 times less files resulted in 20 times better I/O performance when running on PVFS.

However, despite the I/O performance increase in the hybrid version of the model, scalability issues still remain. This led to the current approach of using visualization to analyze the application's performance. With this technique, we aim to understand what can be done in order to improve OLAM's performance and scalability.

III. Trace-based Visualization of OLAM's Performance

In order to obtain the performance visualization, we needed three tools: LibRastro [3], Pajé [4] and TRIVA [5]. The Section III-A presents these tools, and their integration with OLAM and PVFS is presented in Section III-B.

A. Tools: LibRastro, Pajé and TRIVA

To obtain the trace of the application we used the libRastro tracing library. LibRastro is used to generate execution traces of applications with a minimal impact on its behavior. To use the library, the source code of the target application must be modified at the points of interest, by calling functions to register events (as well as all the data necessary to describe these events). During execution, each process of the target application generates a binary trace file. These trace files must be merged and converted to a higher level language by an application-specific tool, because the semantics of the events change from one application to the other.

One high-level event description language is Pajé. Pajé allows the developer to describe events, states and messages between distinct containers (a container being a process, a computing node, a cluster, a file or any element that may

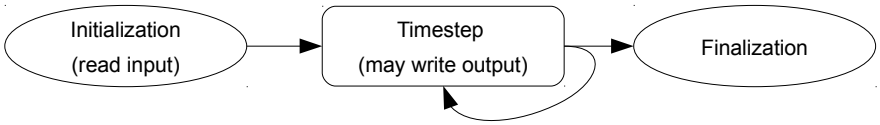


Figure 1. OLAM's iterative organization.

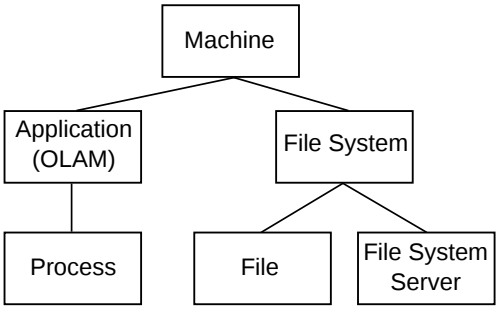


Figure 2. Types of containers in Pajé's representation

have states, events or be source or destination of a message). The developer has a great flexibility to create containers and the associated events in a way which best describes his code.

The visualization of the events can be done either via the Pajé Visualization Tool or TRIVA, which is built on top of the Pajé infrastructure. Pajé allows for a *gantt-chart* style, time based visualization of the events and states of the containers. TRIVA, on the other hand, allows for visualization of other kinds of relations, such as analysis of the time spent in each kind of state. The next section will discuss the integration of the presented tools with OLAM and PVFS.

B. OLAM's instrumentation with LibRastro

This section describes the instrumentation of OLAM and PVFS with libRastro and interpretation of the traces' with Pajé. This modeling affects how the visualization tool (Pajé or TRIVA) shows the results and, therefore, how easy to analyze they will be.

We added calls to LibRastro library functions to OLAM's source code, thus generating events. Beginning and end of these events are specified by two subroutine calls: `IN` and `OUT`, respectively. Each event has a name (typically the name of the subroutine being traced) and optional parameters. In the case of I/O operations, the parameters are the name of the file, amount of data written/read, among others. Similar calls were added to the PVFS source code to trace the I/O operations on the server side.

Pajé typically represents processes as containers. Containers can have states, that have initial and final points in the timeline. Containers can also have arrows to other containers, usually representing the exchange message.

Figure 2 shows the types of containers used in our representation. *Machine* is the standard container type, and can be of *Application* or *File System* type. *Application* containers represent the application (OLAM), with one container per MPI rank (*Process* type). The states of this container are the events obtained from the trace. Therefore, there are states inside other states (when one subroutine calls another).

File System containers represent the file system (PVFS) and can be of two types: *File* or *File System Server*. Each node of the file system (each server) is represented by one container of *File System Server* type. The I/O operations are states of this container, similarly as in *Process* containers. Each node has also a set of *File* containers, one per file stored in the server. Since the files are distributed among the servers, each file is represented by a set of containers: one per server. The states of the *File* containers are I/O requests of the represented file.

Despite all containers being of the *Machine* type, there is no relation between the number of containers and the number of machines in the execution. This happens because more than one process can execute in the same node. Besides, there are containers for the files.

Since the PVFS is dedicated to OLAM in the tests, the I/O operations in the file system are caused by OLAM. However, the applications' requests are usually separated in a larger number of requests in the file system level. Therefore, one I/O operation event on the application's side corresponds to a set of I/O operation events on the file system's side, distributed to a set of servers.

We represented the same operations in both sides intending to visualize the distribution of one request into several smaller requests to different servers. We also intended to identify how much of the time spent by the I/O operation was spent in the servers. The *File* containers were created to provide a visual representation of the access pattern of the application.

Besides the I/O operations, we created events in all of the most important subroutines of OLAM, with the goal of identifying portions of the execution which are impaired by the I/O operations or other factors. Moreover, the detailed analysis of the application can identify the parts that do not scale. The next section presents the results obtained by this approach.

Each event must be of a predefined *type*. Pajé groups events of the same type in an execution flow and automatically stacks one state inside the other as in the case of function calls. For the *application* container, we defined the `APP_STATE` type in which we map events related to the OLAM application. The *process* container has the `P_STATE`

type, which corresponds to I/O utility functions, and the `MPI_STATE` type to which MPI events are mapped.

IV. Results of the I/O Performance Visualization

In order to obtain traces from OLAM, we executed the instrumented version of the application on the clusters *Adonis* and *Edel* of Grid5000 [10]. The tests were executed with 8 nodes using either the local file system or the shared NFS volume to store the execution output. We tested OLAM with and without OpenMP threads.

Figure IV presents part of the Pajé visualization for the execution of OLAM with 8 processes, each with 8 threads, over local files and NFS. The rectangles on the left of the graph mirror the hierarchy shown in Section III-B. The events regarding the application functions are grouped by the *APP_STATE* flow, while the functions regarding I/O calls are grouped by the *P_STATE*. When the application enters the *OLAM_OUTPUT* state, the underlying I/O functions are presented in the process container below. This can be more easily observed in Figure 3(a) at around 11 seconds: when the application enters the *O_OUPUT* state, the process calls a sequence of HDF5 helper functions, of which *SHDF5_OREC* takes the longest. This functions is responsible for writing the variables describing the atmospheric conditions to the output file of the process.

In Figure 3(a) we can see the first 9 seconds of execution for some of the processes of OLAM. In the *APP_STATE* flow, we can observe the execution of a sequence of *timesteps* (event TIMESTEP) after which the *olam_output* (event O_OUTPUT) function is called. At around 6.5 and 11 seconds of execution, we can observe that one O_OUTPUT event that takes longer to complete, something that can be observed in other parts of the execution and in other processes. The execution over NFS (Figure 3(b)) has a similar behavior, but the divergence between the normal I/O phases and the long ones is smaller. In these cases, we can observe that the function responsible for the divergence is *hdf5_close_write*, called from within *SHDF5_OREC*.

Figure 4 presents the visualization of traces generated when executing 64 processes of OLAM with no threads (classic MPI). In this case, since the data division becomes more fine-grained process-wise, the time of each timestep is smaller and there are more frequent I/O calls. Despite the overhead associated with creating more files for the same work, the performance was not penalized in the execution with local files (Figure 4(a)) due to the use of a fast local disk shared by each 8 processes. This overhead was made clear when output files are stored in the shared NFS volume (Figure 4(b)). In this case, most of the time spent in the P_STATE flow is in closing the file (SHDF5_CLOSE): small writes end up being delegated to the write-back mechanism in kernel, but the requests must be flushed by the time the file is closed.

Additionally, we can observe that some O_OUTPUT events last longer than the corresponding events in the P_STATE flow. As Figure 5 shows, this is due to the periodic call of *plot_fields* to generate the graphical representation of the model.

Figure 6 presents the time spent on the I/O states of the application for 3 different processes from the trace of Figure 4(b). We can observe that I/O functions occupy around 25% of the execution time. Most of this time is spent on the *SHDF5_CLOSE_WRITE* function. This is due to the write-back mechanism, as said before, indicating that this is a point to be further studied to identify possible optimizations.

V. Related Work

Scalability of Parallel applications is the focus of several papers in the literature. In [11] the authors execute the high resolution WRF weather forecast code on 15k cores and conclude that one of the greatest bottlenecks was data storage. This same code was used in [12] to evaluate file system caching and pre-fetching optimizations to many-core concurrent I/O, achieving improvements of 100

I/O contention on multi-core systems is also a known issue in the literature and a few strategies to mitigate the performance loss can be found. In [13], the authors improve the performance of multiple concurrent I/O streams on multi-core nodes using data aggregation, forcing contiguous access on the storage nodes. A similar strategy is used in IOFSL [14], an initiative from the PVFS team.

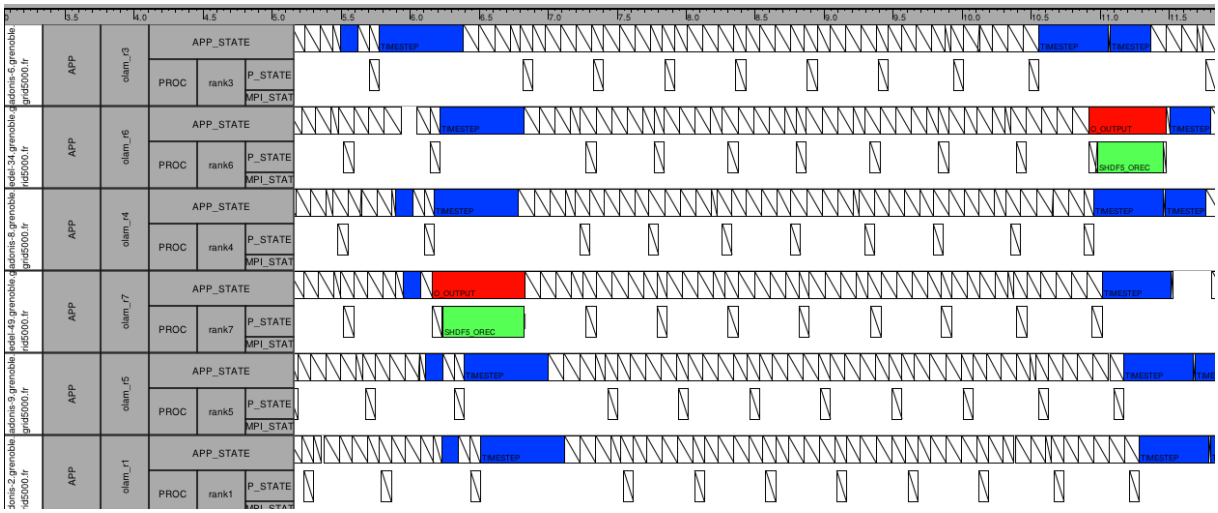
Carns et al. [2] employed five techniques on PVFS in order to improve its performance for small-files scenarios: precreation of objects, stuffing of files, coalescing metadata commits, eager I/O and POSIX extensions for directory access. In local file systems, the log structure is commonly used to improve the performance of operations in small files [15], [16].

As explained before, optimizing OLAM scalability is an ongoing research problem. [17] presents several results regarding the model scalability on multi-core clusters and identifies points of contention.

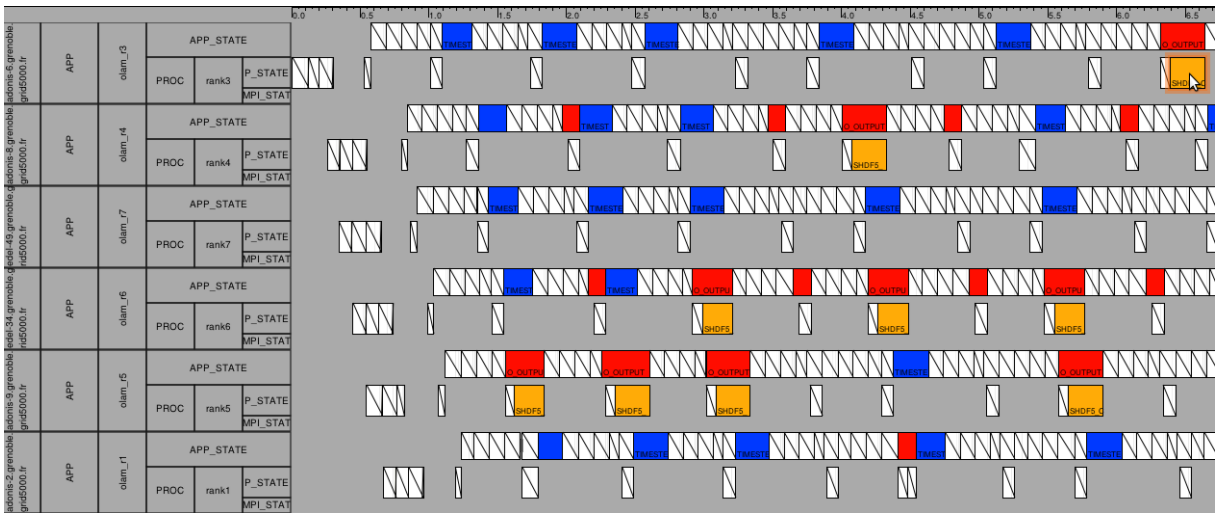
More recently, [18] presents an evaluation of OLAM with the Intel Vtune Analyzer and identifies a large amount of cache misses when using 8 cores.

Performance visualization is currently being studied to make it easier to analyze performance issues in parallel programs. Muelder et al. [19] created *IOVIS*, which is a system to analyze I/O system behavior. It consists of two parts: a data collection tool which generates traces by hooking into I/O system software, and a data analysis and visualization tool to present the I/O behavior of an application.

Another approach was taken by Uselton et al. [20]. They extended an existing performance monitoring tool to include I/O tracing and used the generated I/O statistics to



(a) local files



(b) NFS

O_OUTPUT SHDF5_CLOSE SHDF5_OPEN Aggregated Event
PLOT_FIELDS SHDF5_OREC TIMESTEP

Figure 3. OLAM execution with 8 Processes, 8 Threads each (64cores)

find performance problems in several HPC applications. By fixing these problems, they achieved a speedup of up to 4x.

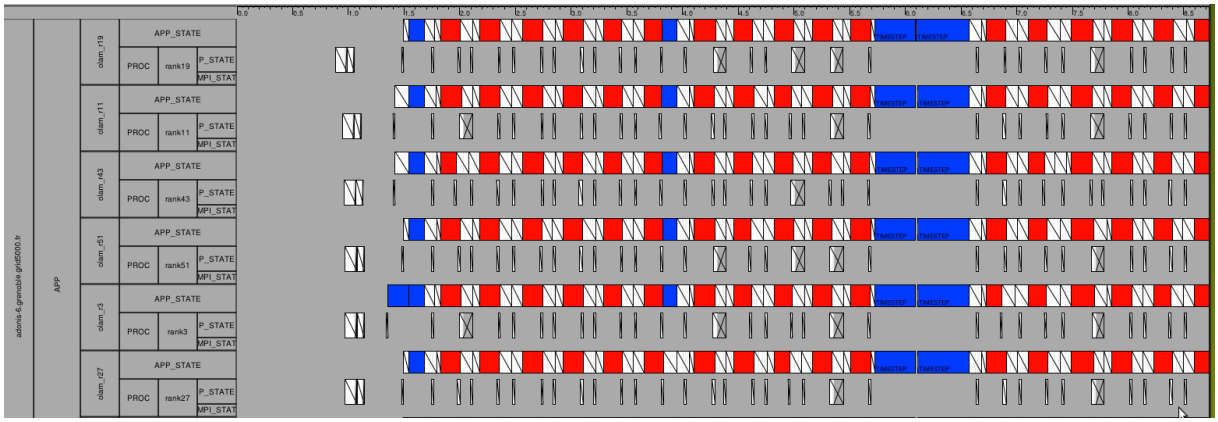
Ross et al. [21] examine the performance of analyzing parallel applications, taking into account steps such as preprocessing and organizing data. They conclude that I/O performance is critical and recommend analyzing data it is being computed.

VI. Conclusions and Future Work

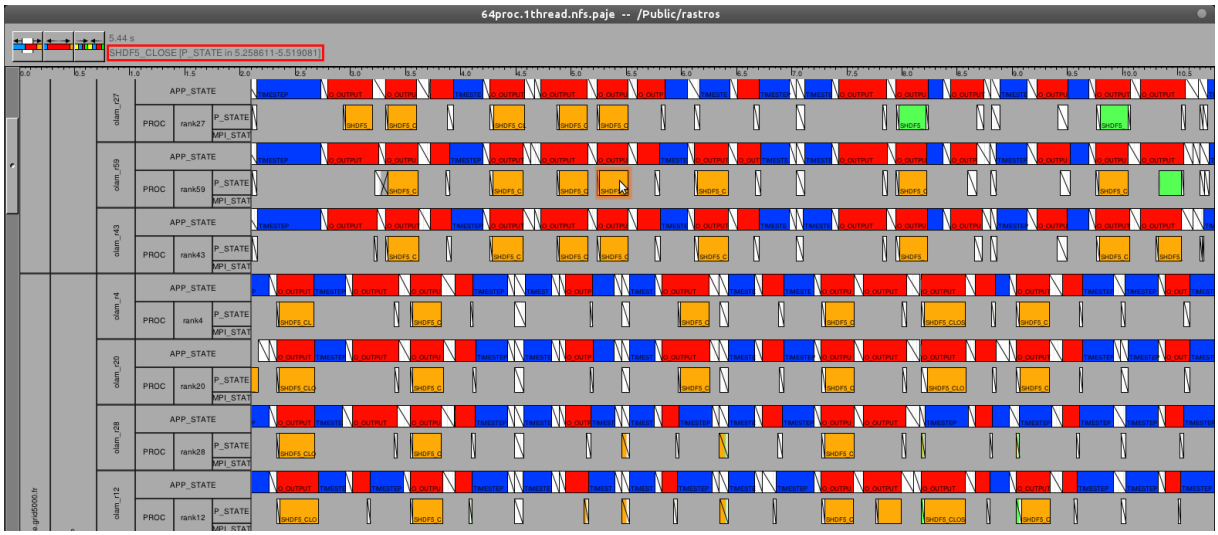
This paper presented a trace-based visualization of I/O performance of a large atmospheric model, the Ocean-Land-Atmosphere Model (OLAM), using the PVFS parallel file system. OLAM has a well-known scalability issue, and its performance is proved to be limited by its I/O operations. The model's access pattern, generating a large number of small files, performs poorly in parallel file systems. The analysis was done with the intent to understand the model's performance and how it is impaired by the I/O operations.

The libRastro library was used to instrument and obtain the traces of the application. The traces were analyzed and visualized with the Pajé and TRIVA tools. We observed that... TODO

As future work, we intend to refine the execution's representation in the visualization in order to improve readability.



(a) local files



(b) NFS

Figure 4. OLAM execution with 64 Processes, 1 Thread each (64cores)

One way of doing it would be creating arrows between states that represent I/O operations. It would connect the operations of the server with the requests of the application that originated them. Also, we must study ways of improving the performance of the close operation.

Acknowledgment

This work was partially supported by CNPq, CAPES, FAPERGS and FINEP.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] R. Walko and R. Avissar, "The Ocean–Land–Atmosphere Model (OLAM). Part I: Shallow-Water Tests," *Monthly Weather Review*, vol. 136, pp. 4033–4044, 2008.

[2] P. H. Carns, W. B. Ligon III, and R. B. Ross, "PVFS: A Parallel File System for Linux Clusters," in *Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4 (ALS '00)*. Berkeley, USA: USENIX Association, 2000, p. 28. [Online]. Available: http://portal.acm.org/citation.cfm?id=1268407

[3] G. J. da Silva, L. M. Schnorr, and B. de Oliveira Stein, "Jrastro: A trace agent for debugging multithreaded and distributed java programs," *Computer Architecture and High Performance Computing, Symposium on*, vol. 0, p. 46, 2003.

Figure 5. `plot_fields` function with 64 processes

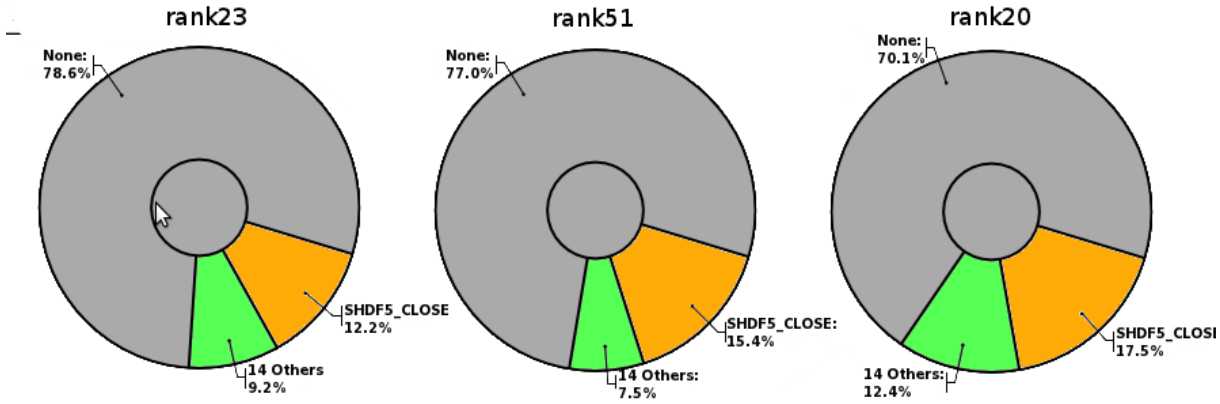


Figure 6. Relative time spent on I/O states

[4] J. C. de Kergommeaux, B. Stein, and P. E. Bernard, "Pajé, an interactive visualization tool for tuning multi-threaded parallel applications," *Parallel Computing*, vol. 26, no. 10, pp. 1253 – 1274, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819100000107

[5] L. M. Schnorr, G. Huard, and P. O. Navaux, "Triva: Interactive 3d visualization for performance analysis of parallel applications," *Future Generation Computer Systems*, vol. 26, no. 3, pp. 348 – 358, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X09001563

[6] R. Walko, C. Tremback, and R. Hertenstein, *RAMS - The Regional Atmospheric Modeling System - Version 3b - Users Guide*, ASTER Division, Fort Collins, CO, 1995.

[7] F. Boito, R. Kassick, L. Pilla, N. Barbieri, C. Schepke, P. Navaux, N. Maillard, Y. Denneulin, C. Osthoff, P. Grunmann, P. Dias, and J. Panetta, "I/O performance of a large atmospheric model using PVFS," in *Rencontres francophones du Parallélisme (RenPar'20)*, 2011.

[8] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig, "Small-file access in parallel file systems," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–11. [Online]. Available: http://portal.acm.org/citation.cfm?id=1586640.1587397

[9] C. Osthoff, P. J. Grunmann, F. Boito, R. Kassick, L. Pilla, P. O. Navaux, C. Schepke, J. Panetta, N. B. Maillard, P. L. S. Dias, and R. Walko, "Improving performance on atmospheric models through a hybrid OpenMP/MPI implementation," in *Proceedings of the 9th IEEE International Symposium on Parallel and Distributed Processing with Applications*, 2011.

[10] R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche, "Grid'5000: A large scale and highly reconfigurable experimental grid testbed," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, 2006. [Online]. Available: http://hpc.sagepub.com/cgi/content/abstract/20/4/481

[11] J. Michalakes, J. Hacker, R. Loft, M. O. McCracken, a. Snavely, N. J. Wright, T. Spelce, B. Gorda, and R. Walkup, "WRF nature run," *Journal of Physics: Conference Series*, vol. 125, p. 012022, Jul. 2008. [Online]. Available: http://stacks.iop.org/1742-6596/125/i=1/a=012022?key=crossref.0b0a842e3b50700109397fd81b130dc6

[12] S. Seelam, I.-H. Chung, J. Bauer, H. Yu, and H.-F. Wem, "Application level I/O caching on Blue Gene/P systems," in *IEEE International Symposium on Parallel Distributed Processing (IPDPS 2009)*. IEEE, 2009. [Online]. Available: http://ieeexplore.ieee.org/iel5/5136864/5160846/05161230.pdf?arnumber=5161230

[13] K. Ohta, H. Matsuba, and Y. Ishikawa, "Improving Parallel Write by Node-Level Request Scheduling," in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. Ieee, 2009, pp. 196–203. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5071872

[14] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan, "Scalable I/O forwarding framework for high-performance computing systems," *2009 IEEE International Conference on Cluster Computing and Workshops*, pp. 1–10, 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5289188

[15] Z. Zhang and K. Ghose, "hFS : A Hybrid File System Prototype for Improving Small File and Metadata Performance," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 175–187, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.160.507&rep=rep1&type=pdf

[16] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM SIGOPS Operating Systems Review*, vol. 25, no. 5, pp. 1–15, 1992. [Online]. Available: http://www.cs.berkeley.edu/~brewer/cs262/LFS.pdf

[17] C. Osthoff, P. Grunmann, F. Boito, R. Kassick, L. Pilla, P. Navaux, C. Schepke, J. Panetta, N. Maillard, P. L. S. Dias, and R. Walko, "Improving Performance on Atmospheric Models through a Hybrid OpenMP / MPI Implementation," in *Proceedings of the 9th IEEE International Symposium on Parallel and Distributed Processing with Applications*. IEEE, 2011. [Online]. Available: http://www.lncc.br/pdf_consultar.php?idt_arquivo=4347&mostrar=1&teste=1

[18] C. Schepke, N. Maillard, C. Osthoff, and P. Dias, "Performance Evaluation of an Atmospheric Simulation Model on Multi-Core Environments," in *Proceedings of the Latin American Conference on High Performance Computing 2010*, 2010. [Online]. Available: http://www.lncc.br/pdf_consultar.php?idt_arquivo=2160&mostrar=1&teste=1

[19] C. Muelder, C. Sigovan, K.-l. Ma, J. Cope, S. Lang, K. Iskra, P. Beckman, and R. Ross, "Visual Analysis of I/O System Behavior for High – End Computing," in *Proceedings of the third international workshop on Large-scale system and application performance (LSAP '11)*. New York: ACM, 2011, pp. 19–26. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=1996036&ftid=983800&dwn=1&CFID=35763732&CFTOKEN=45788168

[20] A. Uselton, M. Howison, N. J. Wright, D. Skinner, N. Keen, J. Shalf, K. L. Karavanic, and L. Oliker, "Parallel I/O Performance : From Events to Ensembles," *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5470424

[21] R. B. Ross, T. Peterka, H.-W. Shen, Y. Hong, K.-L. Ma, H. Yu, and K. Moreland, "Visualization and parallel I/O at extreme scale," *Journal of Physics: Conference Series*, vol. 125, p. 012099, Jul. 2008. [Online]. Available: http://stacks.iop.org/1742-6596/125/i=1/a=012099?key=crossref.5eba184b16b9e1a86b37d92687f6beda