Visualizing a Supercomputer: a Case of Objects Regrouping

V. L. Averbukh[1,2], A. S. Bersenev[1], M.A. Forghani[1,2], A. S. Igumnov[1],
D.V. Manakov[1], A. A. Popel[1], S. V. Sharf [1], P. A. Vasev[1]
vasev@imm.uran.ru
[1]N.N. Krasovskii Institute of Mathematics and Mechanics of the Russian Academy of Sciences, Ekaterinburg, Russia;
[2]Ural Federal University named after the first President of Russia B.N. Yeltsin, Ekaterinburg, Russia;

In the paper we present the situation which had required visualization of a large amount of non-trivial objects, such as supercomputer's tasks. The method of visualization of these objects was hard to find. Then we used additional information about an extra structure on those objects. This knowledge led us to an idea of grouping the objects into new generalized ones. Those new artificial objects were easy to visualize due to their small quantity. And they happened to be enough for the cognition of the original problem. That was a successful change of point of view. As a whole, our work belongs to a high-performance computing performance visualization area. It gains valuable attention from scientists over the whole world, for example [1-2].

Keywords: *high-performance computing, software visualization, projection, multi-dimensional visualization.*

1. Introduction. The environment

In high-performance computing (HPC), a typical program consists of many processes running simultaneously on a set of computing nodes. Typically, those processes communicate with each other, for example to interchange with partial results. Aside from that, processes perform read and write of data placed on external storage. This data may be a program's input, output, or intermediate values.

In Krasovskii Institute a supercomputer named *Uran* uses a dedicated file server as external storage. The server contains an array of disks attached using RAID. The server is connected to computing nodes using a dedicated network and is accessible via a network file system (NFS) protocol. HPC programs access user's files via a computing node's local file systems, while physically files are located on external storage.

During supercomputer operation, we observed one problem in a long time span. File operations with the storage had become occasionally laggy. For example, the creation of a zero-byte length file was performing for up to 10 seconds and more. This situation lasted for a few minutes about 1-2 times a month.

To investigate the problem we started to collect statistics. We started probing each computing node of the supercomputer at every 30 seconds for the following data:

- count of NFS read operations performed by the node since the last probe,
- count of NFS write operations,
- NFS traffic amount of the node,
- other information, including CPU load and memory usage.

Then we analyzed the achieved data and came to the conclusion: file storage lagging correlates with a total count of NFS operations per second (IOPS) performed on that storage, especially when IOPS exceeds some limits. It was regardless to read or write direction, as well as to traffic amount. The details of this conclusion have been reported in [3].

Accordingly, we made a hypothesis: spikes in NFS IOPS are generated by a stable set of concrete supercomputer programs. Thus, if we determine these programs, we might re-factor them and so eliminate the spikes, and so eliminate the file server lags.

To check the stated hypothesis we decided to visualize the collected data. The final target for visualization was defined as: allow to visually find the programs which cause NFS spikes.

2. Our way and the problem

First of all, we were able to visualize the existing data of per-node *NFS activity*. Here the term "NFS activity" means a count of NFS operations (both read and write) accumulated during a probing period. It is shown in figure 1.

The view is the following:

1. The time goes from the bottom to the top (covering the range of one month).
2. Each blue column corresponds to one computing node of a supercomputer.
3. The width of a node's column is determined by the function *nfs_activity(node, time)→int*. The more activity a node had at a specified time moment, the wider is its column.

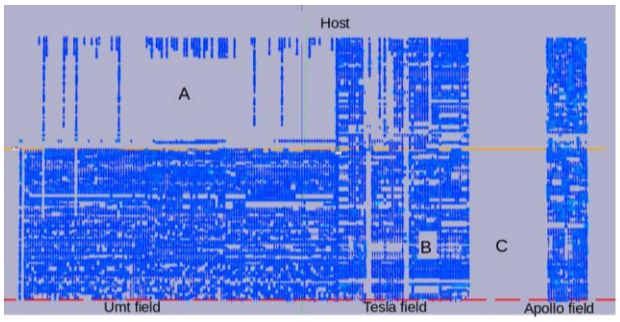

Fig. 1. NFS activity of all nodes of the Uran supercomputer during one month. "Fields" are a node types.

The figure contains three interesting regions, which are empty — A, B and C. The A region is empty due to error in collecting data: it was suspended on some nodes. The B region is empty due to maintenance for those nodes for a few days. The C region is a set of nodes under long-time maintenance.

Although the obtained view seemed interesting, it doesn't answer the predefined question of finding programs with NFS spikes. So we continued our visualization efforts.

When an HPC program is launched, a set of parallel processes is started on computing nodes. We will call such a launch as a *task*. At the Uran supercomputer, the Slurm scheduling system is used to create and assign a task's processes to hardware computing nodes.

Slurm provides historical information on such assignments that determines: which tasks were running on which nodes at which times. Each task record includes:

– user id,
– program name,
– nodes list, where the task's processes were assigned to,
– the start time and the finish time of the task.

This information allowed us to recalculate NFS activity from per-node to per-task basis. Thus we achieved a function *nfs_activity_t(task, time)→int* which returns NFS activity of the given task at the given time.

However, we felt it to be unusable to visualize the tasks "as is" because of their amount. For example, the Uran supercomputer executes about 100 000 of tasks each month. Actually, we even had not found any ideas on how to do that.

Alternatively, there was an idea – to group tasks into programs, and try to visualize them. Unfortunately, it turned to be technically impossible due to the following. Slurm provides corresponding program name for each task, but those names are not stable. For example, suppose there are three tasks. Slurm might provide the following program names for them: «wrap-414», «wrap-5020», «wrap-0103». The first two names might correspond to some real program P1 and the last to a real program P2. Thus the real program name is hidden.

The names are so strange because often users ask supercomputer to run bash script as a program, and those scripts are in turn run real program images. Slurm encodes script names as «wrap_NN» for some reason.

Respectively, the following question arose: how task's visual representation might help us to find programs with NFS spikes? Or at least, which representation method we might use for visualization of tasks?

3. Solution using grouping

The stated problem has been solved by the following.

We used our background observation: for almost all time each user runs only a few programs that are often unique. For example, a user N1 usually spawns only program P1, a user N2 spawns only programs P1 and P2, and user N3 spawns only P3.

This observation allowed us to change our visualization idea: instead of focusing on programs, we may focus on users.

In contrast with the idea of visualizing per-program activity, the per-user activity is possible to compute. It allows determining users who cause NFS spikes and further analyze their programs manually.

In comparison with the attempt for visualization of tasks, the mentioned approach seems to be simpler, since working with users, which is about 200 in our case, is much convenient than the number of tasks.

We constructed the following view, figure 2:

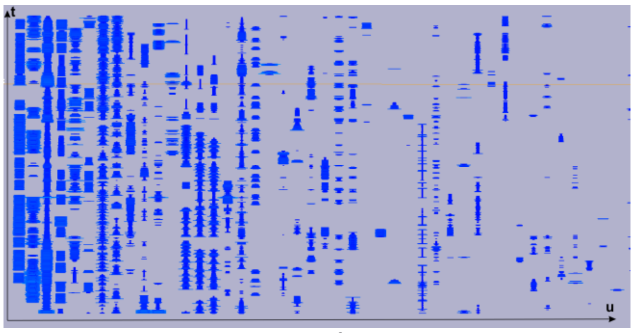

Fig. 2. NFS activity of supercomputer users during 2-weeks period.

1. The time goes from the bottom to the top.
2. Each column corresponds to a user of the supercomputer.
3. The width of the column at each specific time is determined by the value of function *nfs_activity_u(user,time)→ int*, e.g. the total count of NFS operations generated by all of the running tasks of that user at that time.

Additionally, users' columns were sorted by the sum of NFS usage. The motivation for that idea was to visually emphasize most active users and group them. This "group" was under suspicion as a source of storage problems and thus was subject to manual investigation. Now we saw this group visually.

We found the stated view as interesting and informative. It allowed us to determine the most NFS-active users and their programs. But this was not the end of the story.

4. Enhancing the view

During visualization research, we conducted more steps than stated above.

We added 3D graphics for the view, and camera manipulation tools such as zooming, movement, and rotation were used. It allowed us to get extra angles of view to data. It also provided an ability to use another visual method - view of function graphs (see below) simultaneously with columns view. Figure 3 illustrates this effect.

We added visualization for summarized supercomputer activity over time:

– total number of running tasks at supercomputer,
– total count of NFS operations at supercomputer,
– amount of running tasks,
– amount of used CPUs of supercomputer,
– duration of empty file creation.

The meaning of this data is obvious with except of last one. We constructed special test for our NFS storage: create empty file and delete it, and measure the time used by that operation. We started to run it every few seconds and collect times. This measurement occurred to be very expressive for registering storage lagging. Sometime, this operation performed for more than tens of seconds!

The example of summarized activity is presented on the left side in figure 3 as graphs. We decided to show them as function graphs, to visually emphasis the difference of this data in contrast with user's activity (which is shown as columns).

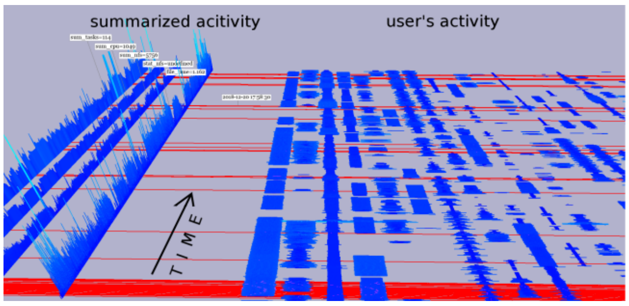

Fig. 3. Summarized supercomputer activity and per-user NFS activity in a 3D view.

Also we added an ability for an investigator to highlight time moments on the view when summarized activity meets some criteria. For example, in figure 3 the red lines depict time moments when empty file creation lasted more than 10 seconds for 2 minutes. This example is crucial – it means that NFS storage naturally hanged.

In common, the criteria used for highlighting is the following: whenever selected column of summarized activity is more or less than specified constant for specified number of sequential probe values. This was enough for our aims.

Another option we added is an ability to perform multiple highlights, one with red, and another with green or blue. For example, we were able to highlight moments of slow empty file creation with red color and moments with huge NFS operations count with green color. It was nice to visually see correlation between these highlights, displayed simultaneously.

Working with this highlights opened an unexpected observations and conclusions on our problem, described in the following section.

5. Unexpected observations

One may look at figure 2 or 3 and decide that most active users (with fat columns) are "guilty" in NFS storage hangs. But that occurred not to be true. Figure 4 illustrates that.

Only user activities are shown. Time goes from right to left. The time moments when NFS storage hangs are depicted by grey lines. The corresponding per-user NFS activities, during storage hangs, are highlighted with purple color.

Naturally speaking, all the users whose columns contain purple color had contributed to storage hangs.

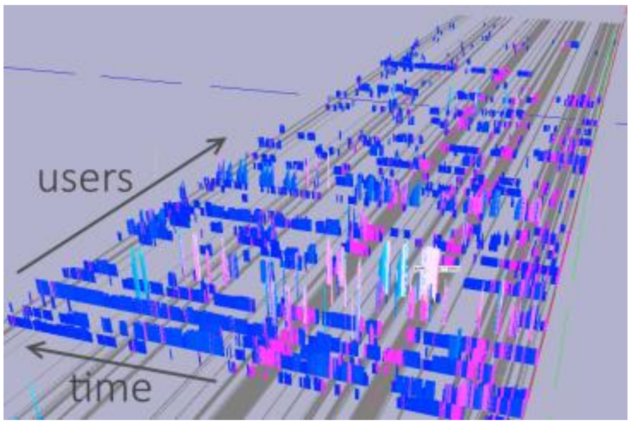

Fig. 4. NFS activities of users making contribution to storage hang are highlighted with purple.

The figure visually explains that not only most NFS-active users are contributing to storage hang, but a lot of users, even lesser-active (located most far at the figure).

We concluded: storage problems are a result of a combination of contributions from various users, and this combination is different from time to time.

Considering the formula "one user = few unique programs", it means that there are no problematic programs subset, as we considered at the beginning of our research. The roots of the problem are that all programs perform NFS operations simultaneously, and their total activity sometimes excess limits leading to storage hang.

The set of programs caused storage problems each time seems to be different. At least it is not connected with most-active NFS users. Probably, there are other patterns that we have not determined yet.

One interesting part of this section is the following. Using purple highlight of users, we achieved visual representation of problematic time moments and their participants, e.g. reasons for the problems. It means that we achieved a visual method of solving our task (of finding programs causing NFS lags).

Of course, we performed extra computations that gave us the same conclusions numerically. But also we achieved that visualization has "computed" the result for us.

We find such cases when visualization makes an answer obvious (to human) as very remarkable. A great example of such a case is a functions graphs *y=f(x):* when it crosses the OX axis it corresponds to "roots" of function's solution and these roots are visually obvious to an observer.

An important detail in our case was the coloring of a user's columns (in contrast to just highlighting time moments by grey lines). If a user has valuable NFS activity at a problematic moment, his column is visually emphasized. Please note that in figure 4 if some user has small NFS activity at problematic moments, his column is visually not emphasized due to corresponding small shape, even still colored.

We see here that it was required to perform additional visual representation changes so human become able to detect what he needs to. And in turn, after that we didn't need any further steps: visualization began to work.

If the illustrated approach will be successful, an interesting theoretical question arises: which visual adjustments should we make to visual representations to make them provide "obvious" answers to questions valuable to a human?

6. Technical background

The visualization was created using web technologies, including WebGL. A 3D visualization framework *Viewlang.ru* was used for graphics programming. It allows programmer to specify objects like Lines, Points, Spheres, Triangles, etc, and then Viewlang translates them into WebGL calls. Thus all the graphics was programmed in Javascript in reactive fashion.

The data is divided into month slices. One-month view was considered as enough for our research. On the visualization index page, a user selects which month he wants to be visualized and visualization is launched, parametrized by the month selected.

In experimentation purposes, we spawned visualizations in Virtual Reality mode using WebVR. We used Oculus Rift DK2 headset for this. It was nice to see the stated views in VR mode.

7. Conclusion

The presented problem and the solution demonstrates the transformation of the initial visualization idea into a simple and more descriptive presentation.

We had a final goal – visually find programs that cause NFS spikes. We had obtained information about tasks from the supercomputer's scheduler. Our initial idea was to visualize these tasks and their NFS activity. But we had no way to associate tasks with programs, due to limitations in scheduler software. So this effort looked as useless in terms of the final goal.

And then we found out that there is no need to visualize tasks. Instead, we had synthesized new objects – the "NFS activity of users". We redirected our view on tasks from per-program to per-user basis. Thus tasks were regrouped.

It was reasonable due to our preliminary knowledge that each user runs a small subset of programs, different from other users. A lot of users run just one self-written program. Furthermore, the number of users of the supercomputer is much less than the number of tasks executed each month. So the visualization had become easier – both in implementation and understanding.

The focus of an investigator's mind was also transformed. Instead of thinking in terms of original objects, it was moved to think about new ones. Instead of thinking about "bad programs", we started thinking about "bad users".

The final of our story is tricky. We found out that there were no "bad users". The overload of supercomputer's storage was caused at time moments when some subset of users together generated an NFS spike. This subset was not stable. Individually, each spike has its specific subset of users who contributed NFS load and thus generated that spike.

We consider the presented case as pretty remarkable to share with the community in order of our common efforts in the evolution of visualization theory and practice.

8. References

[1] Showerman M. Real Time Visualization of Monitoring Data for Large Scale HPC Systems // 2015 IEEE International Conference on Cluster Computing. IEEE, 2015. Pp. 706-709. doi.org/10.1109/CLUSTER.2015.122

[2] Nikitenko, D., Zhumatiy, S., Shvets, P.: Making large-scale systems observable – another inescapable step towards exascale. Supercomput. Front. Innov. J. 3(2), 72–79 (2016). doi.org/10.14529/jsfi160205

[3] Igumnov A. S., Bersenev A. Y., Popel A. A. and Vasev P. A. Studying the Latency of Concurrent Access to Network Storage // In proceedings of 13th international conference on Parallel computational technologies (PCT'2019), Kaliningrad, Russia, 2019. Pp. 66-77. elibrary.ru/item.asp?id=37327250