# Why Read It All? Just Read What Matters: A Path to Faster Scientific Visualization

Qing Zheng
qzheng@lanl.gov
Los Alamos National Laboratory
USA

Brian Atkinson
batkinson@lanl.gov
Los Alamos National Laboratory
USA

Jason Lee
jasonlee@lanl.gov
Los Alamos National Laboratory
USA

Gary Grider
ggrider@lanl.gov
Los Alamos National Laboratory
USA

## Abstract

As HPC simulations generate ever-larger datasets, reducing the volume of data that must be loaded into compute node memory for analysis has become essential for unlocking insights efficiently. In-storage analysis achieves this by processing data directly at the storage servers, allowing them to return only compact results that match regions of interest instead of raw datasets that may be orders of magnitude larger—significantly reducing data footprint.

This e-poster showcases a novel compute-near-storage architecture based on pNFS that enables secure in-storage analysis of scientific data with industry-standard software, including Arrow, Parquet, Substrait, and DuckDB. Using a real-world asteroid impact dataset, we present a live demonstration of a VTK visualization pipeline modified to offload analysis to pNFS data servers, tracing the aftermath of the impact over time and rendering the results on the client as 3D visuals. We show substantial data reduction by pushing down analysis and transmitting only insight-relevant information.

## Interactive ePoster Summary (800 Words)

Modern HPC data centers now support simulations of unprecedented scale and fidelity, generating data that can reach petabytes in a single timestep. Extracting scientific insight from these massive datasets requires advanced analysis pipelines, most often built around visualization workflows that move data from storage into compute node memory where it converts the data into 3D visuals for interactive exploration. Yet, conventional tools are limited by their need to load entire datasets, even though the regions of true scientific interest—such as the forefront of a shockwave—are typically far more focused. This mismatch between data transfer volume and insight focus leads to major inefficiencies in time, energy, and resource utilization, creating the need for new approaches that drastically reduce data read from storage, enabling efficient insight extraction with minimal data movement.

A promising direction is in-storage analysis, which shifts part of the computation from client nodes to the storage servers that hold the data. By processing data directly at its source, irrelevant data regions can be discarded and essential computations applied early. As a result, only compact, derived results that correspond to the sought insights need to be returned, avoiding expensive transfers of raw datasets that can be orders of magnitude larger. Clients then complete the remaining analysis steps based on the returned results, greatly reducing the overall data transfer volume.

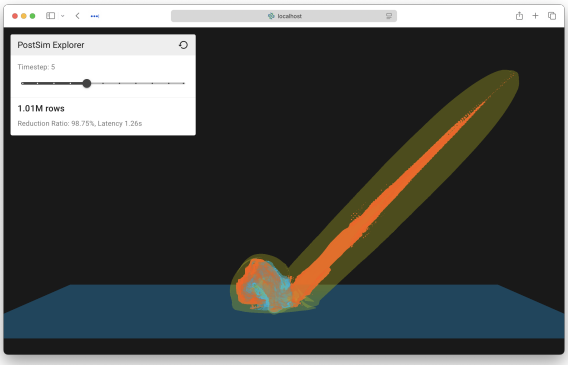


**Figure 1: Interactive analysis of a real-world deep water asteroid impact dataset with contours highlighting water (blue), asteroid (orange), and hot air (yellow). Offloading analysis to storage reduces data movement and client memory usage by up to seven orders of magnitude by transforming only insight-relevant data. Timesteps can be selected for live analysis, with outputs rendered dynamically on screen.**

To showcase the effectiveness of this new approach to data analysis, this e-poster presents a live demonstration of a scientific visualization pipeline built with VTK and adapted for pushdown execution through a novel compute-near-storage architecture based on pNFS. The demo traces the impact of an asteroid colliding with Earth ocean water, using a real-world scientific dataset produced by a multiphysics asteroid-impact simulation.

The pipeline consists of three stages: (1) a reader that retrieves simulation data from storage, (2) a contour filter that generates surfaces highlighting the asteroid, the ocean, and the surrounding hot air, and (3) a renderer that projects these contours onto the screen. Figure 1 shows a snapshot of the visualization at a particular simulation timestep, where the asteroid appears in orange, the ocean in blue, and the hot air in yellow. Through the control panel on the top left, the audience can select different timesteps for live analysis, with results dynamically rendered on the screen.

Traditionally, all components of a visualization pipeline—the reader, filter, and renderer—execute on the compute nodes. This requires the entire raw simulation dataset to be transferred from storage before any processing can occur to generate the contours

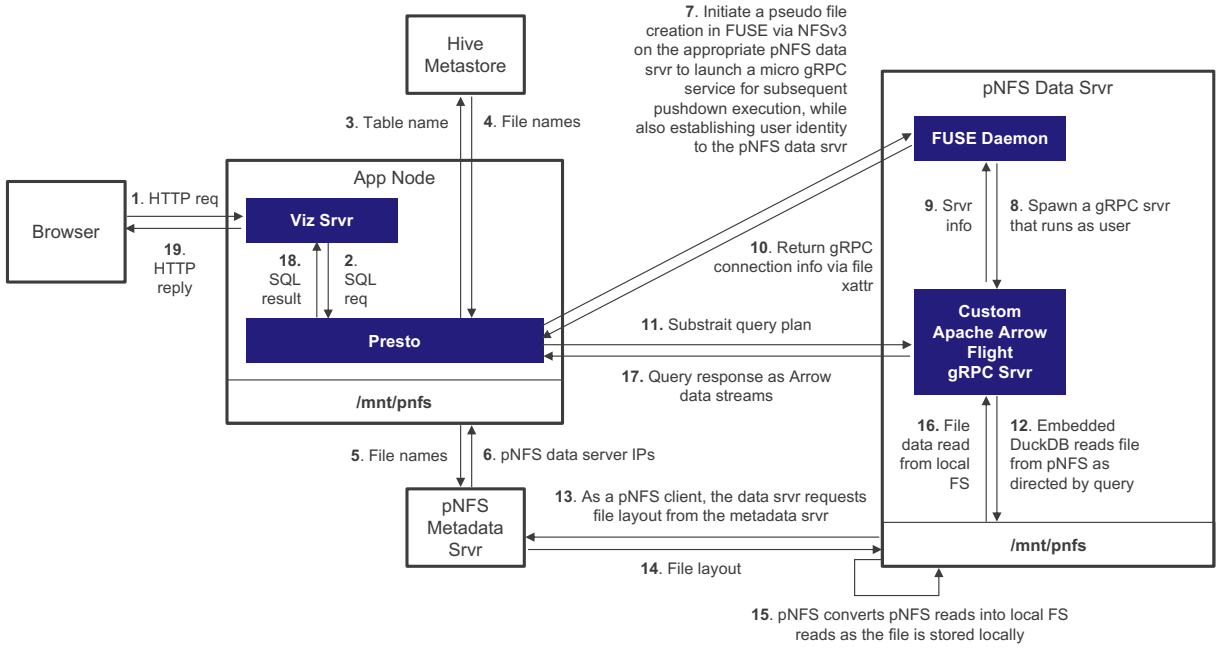


**Figure 2: Overview of our pushdown architecture built on standard pNFS protocols and open-source analytics tools. Client applications issue SQL queries—with visualization operations such as data contouring expressed as custom SQL functions—to distributed engines like Presto (selected for this demo), Spark, or DataFusion (step 2). These engines use custom plugins to decompose the queries into Substrait plans for offloading to storage (step 11). The pNFS metadata server provides file layout information, mapping file names to the specific data servers that hold them (steps 5–6). Through a FUSE mount, users securely launch Arrow Flight gRPC servers on those data servers, enabling queries to be executed directly where the data resides (steps 7–10). Each gRPC server uses customized DuckDB to parse Substrait plans, process data stored in scientific formats such as VTK and HDF5—as well as more broadly adopted industry-standard formats such as Apache Parquet, which we are encouraging the scientific community to embrace—and return results as Arrow streams (steps 12 & 17). A recent Linux kernel patch from Hammerspace allows pNFS read operations to be transparently converted into efficient local reads when the data server hosts the file being read (steps 13–16), completing the end-to-end flow for near-data query execution.**

needed for visualization. Consequently, massive volumes of data are moved even though the actual region of interest—in this case, the contours representing the surfaces of the asteroid, the ocean, and the surrounding air—constitutes only a tiny fraction of the overall simulation output, as Table 1 shows.

With pushdown, we modify the pipeline to offload its reader and filter components to the storage servers that hold the raw simulation output. This enables data to be accessed and analyzed directly where it resides, without first moving it to compute nodes. As a result, contours are computed at the storage layer, and only the derived contour data—up to seven orders of magnitude smaller than the original simulation output—is sent to the compute nodes for final rendering. This drastically reduces data movement.

Our setup consists of a laptop running a lightweight demo UI in a web browser, while the actual analysis is carried out on a remote 3-node pNFS testbed hosted at a data center. The testbed includes a pNFS metadata server, a pNFS data server, and a pNFS client. The data server stores the simulation output, while the client mounts the pNFS filesystem and runs a visualization server that responds to requests from the demo UI on the laptop. All analysis is pushed down to the data server, with insights computed directly where the data resides. The laptop and backend visualization server communicate through a secure SSH tunnel.

Our pushdown protocol extends pNFS with a seamless offload path that delegates analysis operations directly to the data servers holding the files. It leverages the pNFS client's ability to locate the right server and builds on a recent Linux enhancement from Hammerspace that enables efficient local data access using global pNFS file paths without exposing filesystem internal mappings. Offloaded analysis runs under user credentials to receive normal filesystem permission checks. The system integrates industry-standard analytics toolkits including Arrow, Parquet, Substrait, and DuckDB, creating a broadly adoptable platform that applies big data advances to scientific computing while minimizing data movement. Figure 2 illustrates the internal workings of our pushdown architecture.

Future work will focus on applying pushdown across a wider range of scientific applications and more complex visualization pipelines. We are actively seeking collaborations with researchers and developers to apply pushdown in their workflows and expand adoption across the scientific computing community.

**Table 1: Per-timestep mesh complexity and data size comparison between loading raw data versus loading only the pre-derived insight—in our case, the contours representing the surfaces of the ocean water, the asteroid, and the surrounding air—via in-storage analysis. Loading only insight-relevant information reduces data movement by 94%–99.9999%, a reduction of up to seven orders of magnitude.**

| | **Original Simulation Mesh** | | | **Ocean Water Contour** | | | | **Asteroid Contour** | | | | **Hot Air Contour** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Timestep | Cells | Points | Data Size | Cells | Points | Data Size | Reduction Ratio | Cells | Points | Data Size | Reduction Ratio | Cells | Points | Data Size | Reduction Ratio |
| **00000** | 20 M | 26 M | 1.9 GiB | 3.5 M | 1.9 M | 0.12 GiB | **6.44%** | 1.1 K | 0.6 K | 0.04 MiB | **0.0021%** | 0 | 0 | 2.09 KiB | **0.0001%** |
| **06817** | 29 M | 36 M | 2.8 GiB | 3.5 M | 1.9 M | 0.13 GiB | **4.53%** | 0.6 M | 0.3 M | 0.02 GiB | **0.74%** | 0.3 M | 0.2 M | 0.01 GiB | **0.40%** |
| **12244** | 45 M | 54 M | 4.3 GiB | 4.2 M | 2.3 M | 0.15 GiB | **3.55%** | 1.5 M | 0.8 M | 0.05 GiB | **1.23%** | 0.7 M | 0.4 M | 0.03 GiB | **0.59%** |
| **18124** | 125 M | 147 M | 11.8 GiB | 6.0 M | 3.2 M | 0.21 GiB | **1.81%** | 1.2 M | 0.6 M | 0.04 GiB | **0.36%** | 1.4 M | 0.8 M | 0.05 GiB | **0.42%** |
| **24095** | 247 M | 283 M | 23.2 GiB | 6.6 M | 3.6 M | 0.24 GiB | **1.03%** | 1.7 M | 0.9 M | 0.06 GiB | **0.27%** | 1.5 M | 0.8 M | 0.05 GiB | **0.23%** |
| **30068** | 240 M | 277 M | 22.5 GiB | 7.0 M | 3.7 M | 0.25 GiB | **1.11%** | 0.9 M | 0.5 M | 0.03 GiB | **0.14%** | 2.5 M | 1.4 M | 0.09 GiB | **0.40%** |
| **36069** | 111 M | 131 M | 10.5 GiB | 3.3 M | 1.8 M | 0.12 GiB | **1.11%** | 9.7 K | 5.9 K | 0.36 MiB | **0.0034%** | 0.9 M | 0.5 M | 0.03 GiB | **0.31%** |
| **42021** | 92 M | 109 M | 8.7 GiB | 3.1 M | 1.7 M | 0.11 GiB | **1.28%** | 6.9 K | 4.0 K | 0.26 MiB | **0.0029%** | 0.12 M | 0.07 M | 4.4 MiB | **0.05%** |
| **48013** | 135 M | 157 M | 12.7 GiB | 2.3 M | 1.3 M | 0.08 GiB | **0.65%** | 0 | 0 | 2.09 KiB | **0.000016%** | 0.02 M | 0.01 M | 0.9 MiB | **0.0063%** |