SDS: A Framework for Scientific Data Services

Bin Dong, Suren Byna*, John Wu

Scientific Data Management Group
Lawrence Berkeley National Laboratory
Finding Newspaper Articles of Interest

- Finding news articles of your interest in the old era of print newspaper
  - Turn pages
  - Read headlines
  - Searching for specific news was time consuming

- Online newspapers
  - Search box
  - Limited to one newspaper

- Customized News
  - Articles are organized based on reader’s interest
  - Search numerous online newspapers
Finding Newspaper Articles of Interest

- **Finding news articles of your interest in the old era of print newspaper**
  - Turn pages
  - Read headlines
  - Searching for specific news was time consuming

- **Online newspapers**
  - Search box
  - Limited to one newspaper

- **Customized News**
  - Articles are organized based on reader’s interest
  - Search numerous online newspapers
Finding Newspaper Articles of Interest

- Finding news articles of your interest in the old era of print newspaper
  - Turn pages
  - Read headlines
  - Searching for specific news was time consuming
- Online newspapers
  - Search box
  - Limited to one newspaper
- Customized News
  - Articles are organized based on reader’s interest
  - Search numerous online newspapers
Finding Newspaper Articles of Interest

- Finding news articles of your interest in the old era of print newspaper
  - Turn pages
  - Read headlines
  - Searching for specific news was time consuming

- Online newspapers
  - Search box
  - Limited to one newspaper

- Customized News
  - Articles are organized based on reader’s interest
  - Search numerous online newspapers

→ Better organization, faster access to news
Modern scientific discoveries are driven by massive data
- Scientific simulations and experiments in many domains produce data in the range of terabytes and petabytes

Fast data analysis is an important requirement for discovery
- Data is typically stored as files on disks that are managed by file systems
- High data read performance from storage is critical
A Generic Data Analysis Use Case

Simulation Site

Exascale Simulation Machine + analysis

Parallel Storage

Archive

Need to reduce EBs and PBs of data, and move only TBs from simulation sites

Experiment/observation Site

Experiment/observation Processing Machine

(Parallel) Storage

Archive

Perform some data analysis on exascale machine (e.g. in situ pattern identification)

Analysis Sites

Analysis Machines

Shared storage

Reduce and prepare data for further exploratory Analysis (e.g., Data mining)
Data stored on file systems is immutable
- After data producers (simulations and experiments) store data, the file format/organization is unchanged over time
- Data stored in files, which is treated by the system as a sequence of bytes
- User code (and libraries) has to impose meaning of the file layout and conduct analyses
- Burden of optimizing read performance falls on application developer
- However, optimizations are complex due to several layers of parallel I/O stack
Changing layout of data on file systems is helpful

- **Separation of logical and physical data organization through reorganization**

- **Example**
  - Access all variables where $1.15 < \text{Energy} < 1.4$
  - Full scan of data or random accesses to non-contiguous data locations results in poor performance
  - Sorting the data and accessing contiguous chunks of data leads to good performance

- **Several studies showing reorganization can help**
  - Listed some in the related work section

<table>
<thead>
<tr>
<th>Energy</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1692</td>
<td>165.834</td>
<td>-28.0448</td>
<td>-2.76863</td>
<td>...</td>
</tr>
<tr>
<td>2.5364</td>
<td>166.249</td>
<td>-27.9321</td>
<td>-2.87165</td>
<td>...</td>
</tr>
<tr>
<td>1.4364</td>
<td>166.538</td>
<td>-27.9735</td>
<td>-3.16969</td>
<td>...</td>
</tr>
<tr>
<td>1.0862</td>
<td>166.663</td>
<td>-27.9769</td>
<td>-2.79716</td>
<td>...</td>
</tr>
<tr>
<td>1.2862</td>
<td>166.621</td>
<td>-27.9046</td>
<td>-2.78382</td>
<td>...</td>
</tr>
<tr>
<td>1.5862</td>
<td>167.001</td>
<td>-27.9366</td>
<td>-3.09605</td>
<td>...</td>
</tr>
<tr>
<td>1.3173</td>
<td>167.222</td>
<td>-27.9551</td>
<td>-3.22406</td>
<td>...</td>
</tr>
</tbody>
</table>
Changing layout of data on file systems is helpful

- **Separation of logical and physical data organization through reorganization**

- **Example**
  - Access all variables where $1.15 < \text{Energy} < 1.4$
  - Full scan of data or random accesses to non-contiguous data locations results in poor performance
  - Sorting the data and accessing contiguous chunks of data leads to good performance

- **Several studies showing reorganization can help**
  - Listed some in the related work section

| Energy  | X     | Y       | Z       | ...
|---------|-------|---------|---------|-------
| 1.1692  | 165.834 | -28.0448 | -2.76863 | ...
| 2.5364  | 166.249 | -27.9321 | -2.87165 | ...
| 1.4364  | 166.538 | -27.9735 | -3.16969 | ...
| 1.0862  | 166.663 | -27.9769 | -2.79716 | ...
| 1.2862  | 166.621 | -27.9046 | -2.78382 | ...
| 1.5862  | 167.001 | -27.9366 | -3.09605 | ...
| 1.3173  | 167.222 | -27.9551 | -3.22406 | ...

Energy $X$ $Y$ $Z$
Changing layout of data on file systems is helpful

- **Separation of logical and physical data organization through reorganization**

- **Example**
  - Access all variables where $1.15 < \text{Energy} < 1.4$
  - Full scan of data or random accesses to non-contiguous data locations results in poor performance
  - Sorting the data and accessing contiguous chunks of data leads to good performance

- **Several studies showing reorganization can help**
  - Listed some in the related work section

<table>
<thead>
<tr>
<th>Energy</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0862</td>
<td>166.663</td>
<td>-27.9769</td>
<td>-2.79716</td>
<td>...</td>
</tr>
<tr>
<td>1.1692</td>
<td>165.834</td>
<td>-28.0448</td>
<td>-2.76863</td>
<td>...</td>
</tr>
<tr>
<td>1.2862</td>
<td>166.621</td>
<td>-27.9046</td>
<td>-2.78382</td>
<td>...</td>
</tr>
<tr>
<td>1.3173</td>
<td>167.222</td>
<td>-27.9551</td>
<td>-3.22406</td>
<td>...</td>
</tr>
<tr>
<td>1.5862</td>
<td>167.001</td>
<td>-27.9366</td>
<td>-3.09605</td>
<td>...</td>
</tr>
<tr>
<td>1.4364</td>
<td>166.538</td>
<td>-27.9735</td>
<td>-3.16969</td>
<td>...</td>
</tr>
<tr>
<td>2.5364</td>
<td>166.249</td>
<td>-27.9321</td>
<td>-2.87165</td>
<td>...</td>
</tr>
</tbody>
</table>
Database Systems for Scientific Data Management

- Database management systems perform various optimizations without user involvement.
- Many efforts from database community reach out to manage scientific data.
  - Objectivity/DB, Array QL, SciQL, SciDB, etc.
- SciDB
  - A database system for scientific applications
  - Key features:
    - Array-oriented data model
    - Append-only storage
    - First-class support for user-defined functions
    - Massively parallel computations
More DB Optimizations, but …

- Database systems use various optimization tricks and perform them automatically
  - Cache frequently accessed data
  - Use indexes
  - Store materialized views
  - …

- However, preparing and loading of scientific data to fit into database schemas is cumbersome

  - Scientific data management requirements are different from DBMS
    - Established data formats (HDF5, NetCDF, ROOT, FITS, ADIOS-BP, etc.)
    - Arrays are first class citizens
    - Highly concurrent applications
Preserving scientific data formats and adding database optimizations as services

A framework for bringing the merits of database technologies to scientific data

Examples of services
  • Indexing
  • Sorting
  • Reorganization to improve data locality
  • Compression
  • Remote and selective data movement
  • …
First step: Automatic Data Reorganization

- **Finding an optimal data organization strategy**
  - Capture common read patterns
  - Estimate cost with various data organization strategies
  - Rank data organizations for each pattern and select the best for a pattern

- **Performing data reorganization**
  - Similar to database management systems, perform data reorganization transparently
  - Resolving permissions to the original data
  - Preserve the original permissions when data is reorganized

- **Using an optimally reorganized dataset transparently**
  - Based on a read pattern, recognize the best available organization of data
  - Redirect to read the selected organization
  - Perform any needed post processing, such as decompression, transposition
The Design of the SDS Framework

- **Initial implementation of the SDS framework**
  - Client-server approach
  - SDS Client library for each MPI process
  - SDS Server to find the best dataset from available reorganized datasets
  - Supporting HDF5 Read API
  - Added SDS Query API for answering range queries
HDF5 API

- The HDF5 Virtual Object Layer (VOL) feature allows capturing HDF5 calls
- Developed a VOL plugin for SDS for capturing file open, read, and close functions
SDS Client Implementation

SDS Query Interface
• An interface to perform SQL-style queries on arrays
• Function-based API that can be used from C/C++ applications
SDS Client Implementation

Parser
- Checks the conditions in a query
- Verifies the validity of files
**SDS Client Implementation**

- **Server Connector**
  - Packages a query or HDF5 read call information and sends to the SDS server
  - Using protocol buffers for communication
  - MPI Rank 0 communicates with the server and then informs the remaining MPI processes

---

**Diagram Details**

- **Application**
  - SDS Query Interface
  - HDFS API
  - SDS Client
    - SDS Client Connector
      - Packages a query or HDF5 read call information and sends to the SDS server
      - Using protocol buffers for communication
      - MPI Rank 0 communicates with the server and then informs the remaining MPI processes

- **Post Processor**
  - Requested Data
  - File Handle and Query String

- **Reader**
  - Data
  - Original File Metadata or Reorganized File Metadata
  - File Handle and Final Query String

- **Parallel File System**
  - File Handle and Read Parameters
SDS Client Implementation

**Reader**
- Reads data from the dataset location returned by the SDS Server
- Makes use of the native HDF5 read calls
SDS Client Implementation

Post Processor
- Performs any post-processing needed before returning the data to the application memory
- Eg. Decompression, transposition, etc.
SDS Server Implementation

Request Dispatcher

- Receives SDS client requests and SDS Admin interface
- SDS Admin interface issues reorganization commands
- Based on the request, dispatcher passes on the request to Query Evaluator and Reorganization Evaluator
SDS Server Implementation

Query Evaluator

- Looks up SDS Metadata for finding available reorganized datasets and their locations for a given dataset
- SDS Metadata
  - File name, HDF5 dataset info, permissions
Reorganization Evaluator

- Decides whether to reorganize based on the frequency of read accesses
- Takes commands from the Admin interface
- Instructs Data Reorganizer to create a reorganization job script
Data Organization Recommender

- Identifies optimal data reorganization
- Informs the Reorganization Evaluator with the selected strategy
SDS Server Implementation

Data Reorganizer

- Locates reorganization code, such as sorting, indexing algorithms
- Decides on the number of cores to use
- Prepares a batch job script
- Monitors the job execution
- After reorganization, stores the new data location in the SDS Metadata Manager
Performance Evaluation: Setup

- **NERSC Hopper supercomputer**
  - 6,384 compute nodes
  - 2 twelve-core AMD 'MagnyCours' 2.1-GHz processors per node
  - 32 GB DDR3 1333-MHz memory per node
  - Employs the Gemini interconnect with a 3D torus topology
  - Lustre parallel file system with 156 OSTs at a peak BW of 35 GB/s

- **Software**
  - Cray’s MPI library
  - HDF5 Virtual Object Layer code base

- **Data**
  - Particle data from a plasma physics simulation (VPIC), simulating magnetic reconnection phenomenon in space weather
Performance Evaluation: Overhead of SDS

- SDS Metadata Manager uses Berkeley DB for storing metadata of reorganized data
- Measured response time of multiple concurrent SDS Clients requesting metadata from one SDS Server
- Low overhead of < 0.5 seconds with 240 concurrent clients

![Graph showing performance evaluation of SDS](image-url)
Ran various queries on particle energy conditions

Performance benefit over full scan varies based on selectivity of data

20X to 50X speedup when data selectivity less than 5%

Performance Evaluation: Benefits of SDS

- Full Data Scan
- Read Reorganized Data

Selectivity: 79% 32% 11% 4.6% 2.2% 1.2% 0.75% 0.5% 0.33%
Conclusion and Future Work

- File systems on current supercomputers are analogous to print newspapers

- The SDS framework is vehicle for applying various optimizations
  - Live customization of data organization based on data usage (in progress)
  - To work with existing scientific data formats

- Status: Platform for performing various optimizations is ready

- Low overhead and high performance benefits

- Ongoing and future work:
  - Dynamic recognition of read patterns
  - Model-driven data organization optimizations
  - In-memory query execution and query plan optimization
  - FastBit indexing to support querying
  - Support for more data formats
Thanks!

Contact: Suren Byna [SByna@lbl.gov]

Thanks to: