Very Large Dataset Access and Manipulation: 
Active Data Repository (ADR) 
and 
DataCutter

Joel Saltz  
University of Maryland, College Park  
and  
Johns Hopkins Medical Institutions  
http://www.cs.umd.edu/projects/adr

Data Intensive Research Group
University of Maryland/Johns Hopkins  
• Mike Beynon  
• Umit Catalyurek  
• Chialin Chang  
• Renato Ferreira  
• Tahsin Kurc  
• Alan Sussman

Tools to Manage Storage Hierarchy
• Mass Storage:  
  • Load subset of data from tertiary storage into disk cache or client  
  • Access data from distributed data collections  
  • Preprocess close to data sources  
• Fast secondary storage  
  • Tools for on-demand data product generation, interactive data exploration, visualization  
  • Target closely coupled sets of processors/disk

Irregular Multi-dimensional Datasets
• Spatial/multi-dimensional multi-scale, multi-resolution datasets  
• Applications select portions of one or more datasets  
• Selection of data subset makes use of spatial index (e.g., R-tree, quad-tree, etc.)  
• Data not used “as-is”, generally preprocessing is needed - often to reduce data volumes

DataCutter
• A suite of Middleware for subsetting and filtering multi-dimensional datasets stored on archival storage systems  
• Subsetting through Range Queries  
  • a hyperbox in dataset’s multi-dimensional space  
  • retrieve items with multi-dimensional coordinates in box  
• Processing (filtering/aggregations) through Filters  
  • Carry out processing near data, compute servers

Active Data Repository (ADR)
• Set of services for building parallel databases of multi-dimensional datasets  
  • enables integration of storage, retrieval and processing of multi-dimensional datasets on parallel machines.  
  • can maintain and jointly process multiple datasets.  
  • provides support and runtime system for common operations such as  
    • data retrieval,  
    • memory management,  
    • scheduling of processing across a parallel machine.  
  • customizable for various application specific processing.
**Querying Irregular Multi-dimensional Datasets**

- **Irregular datasets**
  - Think of disk-based unstructured meshes, data structures used in adaptive multiple grid calculations, sensor data
  - indexed by spatial location (e.g., position on earth, position of microscope stage)

- **Spatial query used to specify iterator**
  - computation on data obtained from spatial query
  - computation aggregates data - resulting data product size significantly smaller than results of range query

**Dataset Structure**

- **Spatial and temporal resolution may depend on spatial location**
- **Physical quantities computed and stored vary with spatial location**

**Application Scenarios**

- **Locate TB spatio-temporal region in multi-scale, multi-resolution PB dataset, project data onto new spatio-temporal grid**
  - Ad-hoc queries, data products from satellite sensor data
  - Browse or analyze (multi-resolution) digitized slides from high power light or electron microscopy
    - 1-50 GBytes per digitized slide, 5-50 slides per case, 100’s of cases per day per hospital

- **Sensor data, fluid dynamics and chemistry codes to predict condition of waterways (e.g. Chesapeake bay simulation) and to carry out petroleum reservoir simulation**
- **Predict materials properties using electron microscope computerized tomography sensor data**
- **Post-processing, analysis and visualization of data generated by large scientific simulations**
**Processing Remotely Sensed Data**

NOAA TIROS-N w/ AVHRR sensor

AVHRR Level 1 Data
- As the TIROS-N satellite orbits, the Advanced Very High Resolution Radiometer (AVHRR) sensor scans perpendicular to the satellite's track.
- At regular intervals along a scan line measurements are gathered to form an instantaneous field of view (IFOV).
- Scan lines are aggregated into Level 1 data sets.

A single file of Global Area Coverage (GAC) data represents:
- ~one full earth orbit.
- ~110 minutes.
- ~40 megabytes.
- ~15,000 scan lines.

One scan line is 409 IFOV's.

---

**Spatial Irregularity**

AVHRR Level 1B NOAA-7 Satellite 16x16 IFOV blocks.

---

**Active Data Repository**

---

**Application Processing Loop**

O ← Output dataset, I ← Input dataset
A ← Accumulator (intermediate results)
[S_I, S_O] ← Intersect(I, O, R_query)
foreach o in S_O do
    read o
    a_o ← Initialize(o)
foreach i in S_I do
    read i
    S_A ← Map(i) ∩ S_O
    foreach a in S_A do
        a_o ← Aggregate(a, a_o)
foreach a in S_O do
    o_o ← Output(a)
write o_o

---

**Typical Query**

Output grid onto which a projection is carried out

Specify portion of raw sensor data corresponding to some search criterion

---

**Architecture of Active Data Repository**

Client 1 (sequential)

Client 2 (parallel)

Query

Results
**Loading Datasets into ADR**

- A user
  - should decompose dataset into data chunks
  - optionally can distribute chunks across the disks, and provide an index for accessing them
- ADR, given data chunks and associated minimum bounding rectangles in a set of files
  - can distribute data chunks across the disks using a Hilbert-curve based declustering algorithm,
  - can create an R-tree based index on the dataset.

**Data Loading Service**

- User must decompose the dataset into chunks
- For a fully cooked dataset, User
  - moves the data and index files to disks (via ftp, for example)
  - registers the dataset using ADR utility programs
- For a half cooked dataset, ADR
  - computes placement information using a Hilbert curve-based declustering algorithm,
  - builds an R-tree index,
  - moves the data chunks to the disks
  - registers the dataset

**Query Execution in Active Data Repository**

- An ADR Query contains a reference to
  - the data set of interest,
  - a query window (a multi-dimensional bounding box in input dataset’s attribute space),
  - default or user defined index lookup functions,
  - user-defined accumulator,
  - user-defined projection and aggregation functions,
  - how the results are handled (write to disk, or send back to the client).
- ADR handles multiple simultaneous active queries

**ADR Query Execution**

- Query
  - Index lookup
  - Generates a plan
  - Aggregate local input data to output
  - Initialize output
  - Send output to clients
  - Combine partial output results
  - Global Combine Phase
  - Output Handling Phase
  - Initialization Phase
**DataCutter**

- A suite of Middleware for subsetting and filtering multi-dimensional datasets stored on archival storage systems
- Integrated with NPACI Storage Resource Broker (SRB)
- Standalone Prototype

**Processing**

- Processing (filtering/aggregations) through Filters
  - to reduce the amount of data transferred to the client
  - filters can run anywhere, but intended to run near (i.e., over local area network) storage system
  - Standalone system allows multiple filters placed on different platforms
  - SRB release allows only a single filter which can be placed anywhere
  - Motivated by Uysal’s disklet work

**Filter Framework**

```cpp
class MyFilter : public AS_Filter_Base {
public:
    int init(int argc, char *argv[]) {
        ... };
    int process(stream_t st) {
        ... };
    int finalize(void) {
        ... ;
    }
};
```

**DataCutter -- Subsetting**

- Datasets are partitioned into segments
  - used to index the dataset, unit of retrieval
- Indexing very large datasets
  - Multi-level hierarchical indexing scheme
  - Summary index files -- to index a group of segments or detailed index files
  - Detailed index files -- to index the segments
**Placement**

- The dynamic assignment of filters to particular hosts for execution is placement (mapping)
- Optimization criteria:
  - Communication
    - leverage filter affinity to dataset
    - minimize communication volume on slower connections
    - co-locate filters with large communication volume
  - Computation
    - expensive computation on faster, less loaded hosts

**Integration of DataCutter with the Storage Resource Broker**

**Storage Resource Broker (SRB)**

- Middleware between clients and storage resources
- Remote Access to storage resources.
  - Various types:
    - File Systems - UNIX, HPSS, UniTree, DPSS (LBL).
    - DB large objects - Oracle, DB2, Illustra.
    - Uniform client interface (API).

**SRB/DataCutter**

- Support for Range Queries
  - Creation of indices over data sets (composed set of data files)
  - Subsetting of data sets
    - Search for files or portions of files that intersect a given range query
  - Restricted filter operations on portions of files (data segments) before returning them to the client (to perform filtering or aggregation to reduce data volume)

**Storage Resource Broker (SRB)**

- MCAT - MetaData Catalog
  - Datasets (files) and Collections (directories) - inodes and more.
  - Storage resources
  - User information - authentication, access privileges, etc.
- Software package
  - Server, client library, UNIX-like utilities, Java GUI
  - Platforms - Solaris, Sun OS, Digital Unix, SGI Irix, Cray T90.
SRB/DataCutter Client Interface

- Creating and Deleting Index

  ```c
  int sfoCreateIndex(srbConn* conn, sfoClass class, int catType,
                      char *inIndexName, char *outIndexName,
                      char *resourceName);
  ```

  ```c
  int sfoDeleteIndex(srbConn* conn, sfoClass class, int catType,
                     char *indexName);
  ```

- Searching Index -- R-tree index

  ```c
  typedef struct {
    int    dim;
    double *min, *max;
  } rangeQuery;
  ```

  ```c
  int sfoSearchIndex(srbConn* conn, sfoClass class,
                     char *indexName, void *query,
                     indexSearchResult* myresult,
                     int maxSegCount);
  ```

  ```c
  typedef struct {
    segmentInfo   segmentMBR ;   /* bounding box of the segment */
    char     * objID ; /* object in SRB that contains the segment */
    char     * collectionName ; /* collection where object is stored */
    unsigned int offset;         /* offset of the segment in the object */
    unsigned int size;             /* size of segment */
  } segmentInfo ; /* segment meta-data information */
  ```

  ```c
  typedef struct {
    int         segmentCount ;       /* number of segments returned */
    segmentInfo *segments;     /* segment meta-data information */
  } indexSearchResult ; /* search result structure */
  ```

- Applying Filters

  ```c
  typedef struct {
    segmentInfo segInfo ;  /* info on segment data buffer after filter oper. */
    char        *segment; /* segment data buffer after filter is applied */
  } segmentData;
  ```

  ```c
  typedef struct {
    int         segmentDataCount; /* #segments in segmentData array */
    segmentData *segments;    /* segmentData array */
  } filterDataResult;
  ```

  ```c
  int sfoApplyFilter(srbConn* conn, sfoClass class, char *hostName,
                     int filterID, char *filterArg,
                     int numOfInputSegments,
                     segmentInfo* inputSegments,
                     filterDataResult *myresult,
                     int maxSegCount);
  ```

  ```c
  int sfoGetMoreFilterResult(srbConn* conn, int continueIndex,
                              filterDataResult *myresult,
                              int maxSegCount);
  ```

Application: Virtual Microscope

- Interactive software emulation of high power light microscope for processing/visualizing image datasets
- 3-D Image Dataset (100MB to 5GB per focal plane)
- Client-server system organization
- Rectangular region queries, multiple data chunk reply
- Pipeline style processing
Virtual Microscope Client

VM Application using SRB/DataCutter

Experimental Setup
- UMD 10 node IBM SP (1 4CPU, 3 2CPU, 6 1CPU)
- HPSS system (10TB tape storage, 500GB disk cache)
- 4GB JPEG compressed dataset (90GB uncompressed), 180k x 180k RGB pixels (200 x 200 jpeg blocks of 900x900 pixels each)
- 250GB JPEG compressed dataset (5.6TB uncompressed), 1.44Mx1.44M RGB pixels (1600x1600 jpeg blocks)
- Rtree index based query lookups
- server host = SP 2CPU node
- Read, Decompress, Clip, Zoom, View distributed between client and server

Dataset --250 GB (Compressed)
All Computation on Server

<table>
<thead>
<tr>
<th>Query Size</th>
<th>Cold Cache (Sec)</th>
<th>Warm Cache (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500x500</td>
<td>31</td>
<td>5</td>
</tr>
<tr>
<td>1000x1000</td>
<td>244</td>
<td>48</td>
</tr>
<tr>
<td>1800x1800</td>
<td>416</td>
<td>100</td>
</tr>
</tbody>
</table>

Breakdown of DataCutter Costs
250 GB dataset, 9600x9600 query

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cold Cache (Sec)</th>
<th>Warm Cache (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Query</td>
<td>244</td>
<td>48</td>
</tr>
<tr>
<td>Compute Index</td>
<td>102</td>
<td>3</td>
</tr>
<tr>
<td>Lookup Data</td>
<td>115</td>
<td>25</td>
</tr>
</tbody>
</table>

Effect of Filter Placement
9600x9600 Query Warm Cache

<table>
<thead>
<tr>
<th>Everything but View on Server (Seconds)</th>
<th>Server:Read Decompress, Clip (Seconds)</th>
<th>Server just reads, client does all else (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5K</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>5K</td>
<td>48</td>
<td>46</td>
</tr>
<tr>
<td>6.6K</td>
<td>48</td>
<td>46</td>
</tr>
<tr>
<td>8K</td>
<td>80</td>
<td>86</td>
</tr>
<tr>
<td>16K</td>
<td>180</td>
<td>101</td>
</tr>
</tbody>
</table>
**Effect of Dataset Size**

4.5Kx4.5K Query

Server does Everything but View

Warm Cache

<table>
<thead>
<tr>
<th>Dataset Size</th>
<th>Size Uncompressed</th>
<th>Total Time (Sec)</th>
<th>DataCutter Indexing (Sec)</th>
<th>DataCutter Data Retrieval (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4GB</td>
<td>90GB</td>
<td>49</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>250GB</td>
<td>5.6TB</td>
<td>75</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

---

**The Future**

- Integrated suite of tools for handling very deep memory hierarchies
  - Common set of tools for grid and disk cache computations
- **Programmability**
  - Use XML metadata
  - Ongoing data parallel compiler project -- uses Java based user defined functions
  - Applications development toolkit (Visual DataCutter)
- **Implementation**
  - NPACI
  - Private sector (?)