



ADR and DataCutter

Sergey Koren
CMSC818S

Thursday March 4th, 2004



Active Data Repository

- ◆ Used for building parallel databases from multi-dimensional data sets
- ◆ Integrates storage, retrieval, and processing of multi-dimensional data sets
- ◆ Customization for application specific processing (Initialize, Map, Aggregate, Output)
- ◆ Support common operations like memory management, data retrieval, and scheduling

Application Properties

- ◆ Irregular multi-dimensional datasets
 - Meshes, sensor data
- ◆ Input and output often disk resident
- ◆ Use a subset of data, access data through a range query (bounding box)

Application Processing

- ◆ Retrieve data that matches the range query
- ◆ Mapping input to output
- ◆ Accumulator used to hold intermediate result
 - Aggregate input value with the intermediate result
 - Aggregate operation usually commutative and associative so can aggregate in any order
- ◆ Output is usually much smaller so the processing steps are called a reduction

Processing Loop

$O \leftarrow$ Output Dataset, $I \leftarrow$ Input Dataset
(* Initialization *)

1. **foreach** o_e **in** O **do**
2. **read** o_e
3. $a_e \leftarrow \text{Initialize}(o_e)$
(* Reduction *)
4. **foreach** i_e **in** I **do**
5. **read** i_e
6. $S_A \leftarrow \text{Map}(i_e)$
7. **foreach** a_e **in** S_A **do**
8. $a_e \leftarrow \text{Aggregate}(i_e, a_e)$
(* Output *)
9. **foreach** a_e **do**
10. $o_e \leftarrow \text{Output}(a_e)$
11. **write** o_e

Figure 1: The basic processing loop in the target applications.



Architecture



- ◆ Front-end
 - Interacts with clients, forwards range queries to back-end
- ◆ Parallel Back-end
 - Retrieve data items that intersect query
 - Perform user-defined functions to generate output

Services

- ◆ Attribute Space Service
 - Manages use of attribute spaces and user map functions
- ◆ Dataset Service
 - Manages datasets stored on ADR back-end
- ◆ Indexing Service
 - Manages indexes (user defined and ADR)
- ◆ Data Aggregation Service
 - Manages user provided function for aggregation
- ◆ Query Planning Service
 - Determines query plan to process a set of queries
- ◆ Query Execution Service
 - Manages resources and carries out query plan

ADR Application Suite

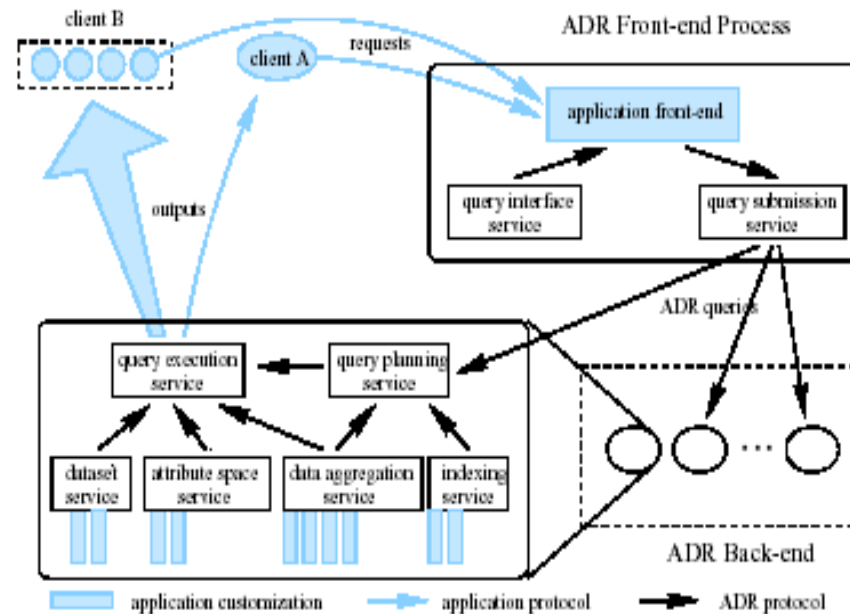


Figure 2: A complete application suite implemented as a customized ADR application. The shaded bars represent functions added to ADR by the user as part of the customization process. Client A is a sequential program while client B is a parallel program.

Loading Data

- ◆ User decomposes data into chunks
- ◆ Compute placement information
- ◆ Move to location and compute an index
- ◆ Minimum Bounding Rectangle for each chunk
 - Distribute chunks across disks using declustering algorithm
 - R-Tree index using MBR
- ◆ Chunks are units of access and only read/written by local processor

Query Planning

- ◆ Tiling
 - If the output is too large to fit entirely into memory
 - Each tile is a subset of output chunks
 - Results in implicit tiling on input chunks
- ◆ Workload Partitioning
 - Each processor gets responsibility for processing subset of input/accumulator chunks

Query Execution

- ◆ Processing operations on storage manager to avoid copying
- ◆ Four steps
 - 1. Initialization
 - Accumulator elements for current tile allocated
 - 2. Local Reduction
 - Processors retrieve chunks on local disk and aggregate into accumulator
 - 3. Global Combine
 - Partial results from step 2 are combined
 - 4. Output Handling
 - Output computed from accumulator
- ◆ Overlap disk, network, and processing operations

Processing Strategies

- ◆ FRA (Fully Replicated Accumulator)
 - Processor performs processing associated with its local input chunks
 - Output partitioned among processors. When output chunk is assigned to a tile, corresponding accumulator chunk is put into a set of local chunks of processor that owns the output chunk
 - Accumulator chunk assigned as ghost chunk on all other processors
 - Effectively replicates all accumulator chunks on all processors
 - Processors generates partial results using its input chunks

Processing Strategies (2)

- ◆ SRA (Sparsely Replicated Accumulator)
 - FRA wasteful of memory because replicates all, even those that have no input chunks mapping to them
 - Initialization overhead in Initialization phase and communication and computation overhead in Global Combine phase
 - Same as FRA but, a ghost chunk is allocated only if the processor has a local input chunk that projects to the accumulator chunk

Processing Strategies (3)

- ◆ DA (Distributed Accumulator)
 - Each processor is given responsibility to carry out work associated with a set of output chunks
 - Accumulator chunks are not replicated to remote processes
 - Remote input chunks that map to local output chunks must be forwarded to processor

Processing Strategies (4)

◆ Comparison

- DA avoid inter-processor communication for accumulator chunks in initialize and for ghost chunks in the global combine
- FRA and SRA avoid inter-processor communication for input chunks
- DA introduces communication in local reduction phase for input chunks



Results

- ◆ Communication volume and computation time per processor for DA decreases as processors increase
- ◆ Initialization and global combine phases for FRA and SRA remain constant
- ◆ DA has higher communication volume and more load imbalance when processors and data size increase
- ◆ DA communication volume is proportional to number of input chunks and fan-out
- ◆ FRA proportional to number of output chunks



Current Version



- ◆ <http://www.cs.umd.edu/projects/hpsl/chaos/ResearchAreas/adr/>
- ◆ Version 1.0 – Updated in December 2000



DataCutter

- ◆ Indexing
 - Works with multi-dimensional data
- ◆ Filter-based programming model
 - Location independence
 - Users define connectivity
 - Communicate through uni-directional streams



Indexing

- ◆ Data and index files
- ◆ Segments contain some data tied to the multi-dimensional space
- ◆ MBR (Minimum Bounding Rectangle) that encompasses data points in segment
- ◆ Segments are units of access

Indexing (cont...)

- ◆ Use R-Trees, but a lot of data makes index large
- ◆ Use both summary index files and detailed index files
 - Summary associates with multiple segments and detailed index files
 - Detailed index specifies segments
- ◆ Can be organized into groups



Filters

- ◆ Component based model, components are called filters
- ◆ Communicate through uni-directional pipes
- ◆ Location independent, connections through names
- ◆ Optimize use of limited resources

Filter Implementation

- ◆ Filter provides three functions
 - Init (initializeWork in new version)
 - Called whenever a new unit of work arrives
 - Creates structures
 - Process
 - Called to process incoming data (can return end of work or end of filter)
 - Finalize (finalizeWork in new version)
 - Called to clean up



Placement

◆ Communication

- Place filters with large volume of communications close (on same host)
- Filters close to data they use
- Minimize communications on slow links

◆ Computation

- Heavy computations on less loaded hosts



Streams

- ◆ Streams for files and for inter-filter communications
- ◆ Buffer abstraction used to send data
- ◆ Default is TCP, if Globus is installed, can use GSS TCP

Applications

- ◆ Decompose application into set of filters
 - Console process, can initiate work or forward work it gets
 - Can receive output from filters or not
- ◆ Filters placed on hosts, can have multiple filters on one host
- ◆ Filters can move between units of work
 - However, doesn't move state/code
 - Can send a buffer to restore state
 - Can pin to a host

Execution

- ◆ Dird
 - A directory daemon in a well known location
- ◆ Appd
 - Register location with the dird
 - Must be running on all hosts where a filter will exist
- ◆ Placement specification says what filter goes on what hosts
- ◆ No copying of code/binaries, must be done manually



Examples

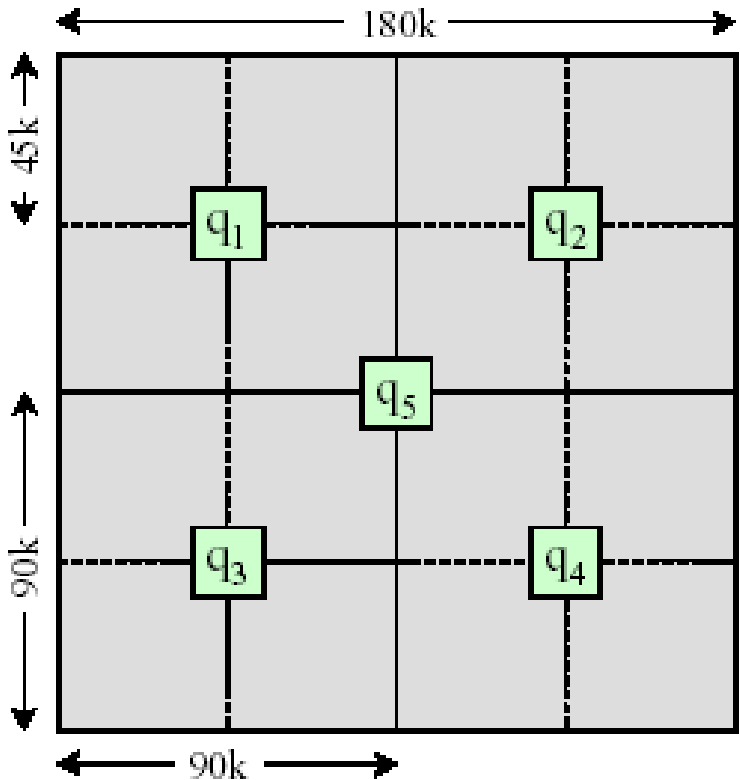
- ◆ Virtual Microscope
 - Read, Decompress, Clip, Zoom, View
- ◆ External Sort
 - Partitioner and Sorter



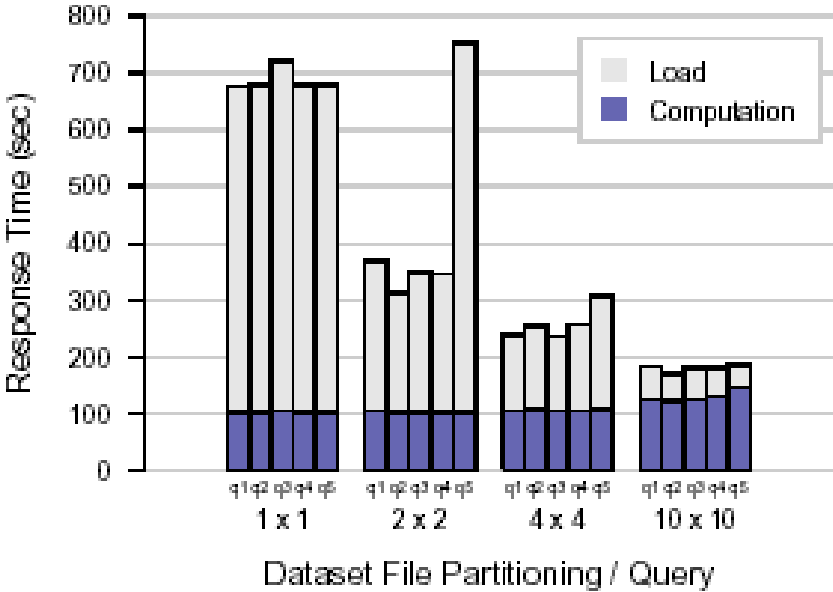
Virtual Microscope

- ◆ Shows that splitting into files can reduce load times (because loading file from HPSS forces entire file to be placed in cache)
- ◆ Too much splitting overhead of opening too many files
- ◆ Indexing the data properly is important

Results



(a)

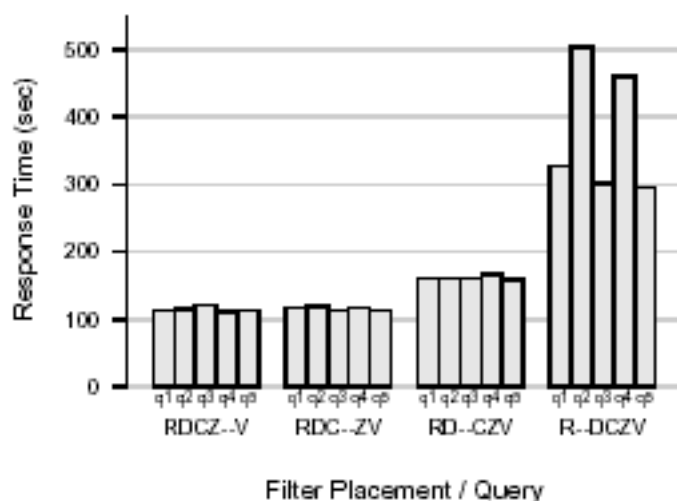


(b)

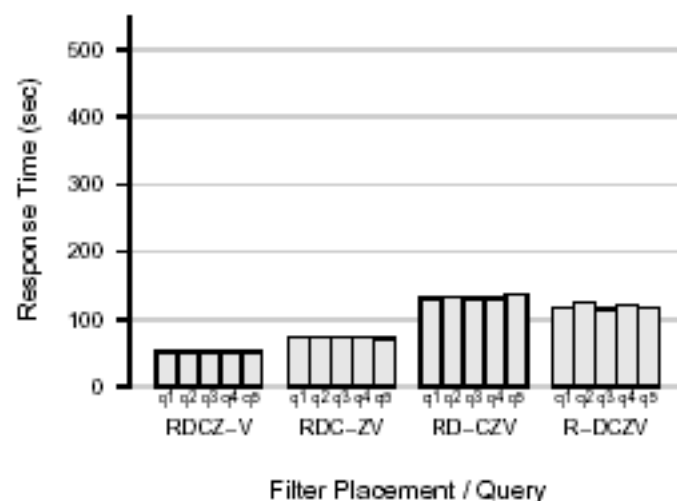
Virtual Microscope (cont ...)

- ◆ Two machines, server and client
- ◆ Vary placement of filters, the zoom and the decompress filters
- ◆ Complex relation on placement
 - Decompress most computational
 - Communication between two machines
 - Communication between view filter and client driver

Results



(a) no zooming (no subsampling)



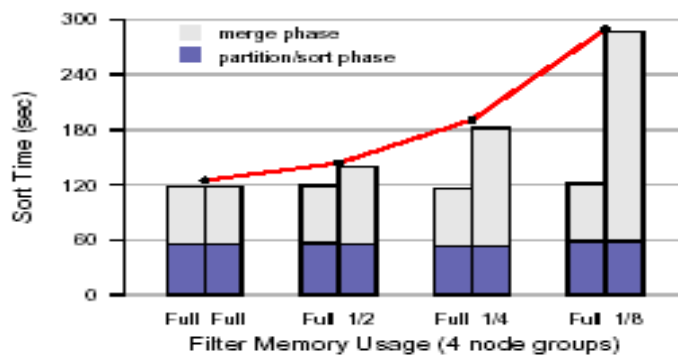
(b) subsampling by a factor of 8

Fig. 7. Execution time of queries under varying processing (subsampling). R,D,C,Z,V denote the filters *read-data*, *decompress*, *clip*, *zoom*, and *view*, respectively. <server>-<client> denotes the placement of the filters in each set.

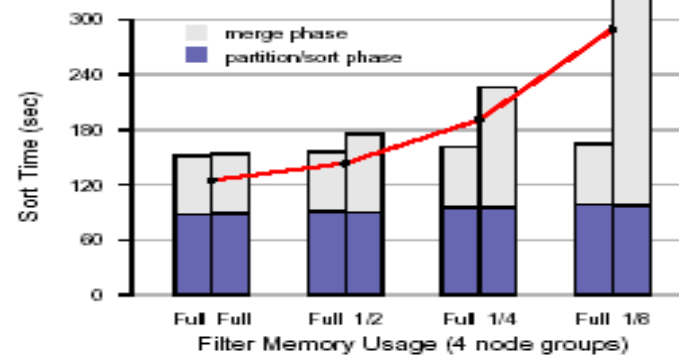
External Sort

- ◆ Vary available memory
 - Baseline, comparison where memory is reduced on all machines
 - Several results with half nodes with reduced memory, half with full
 - Placement can balance heterogeneity, but have to be careful

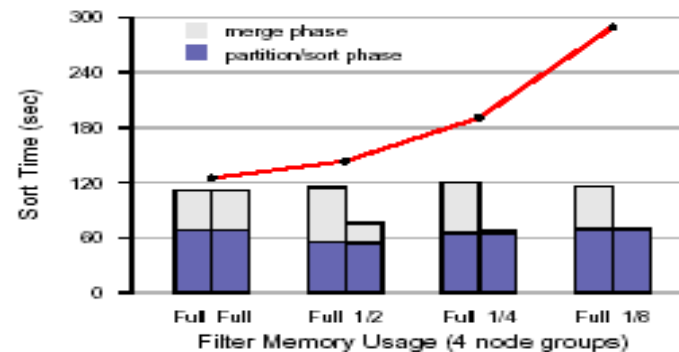
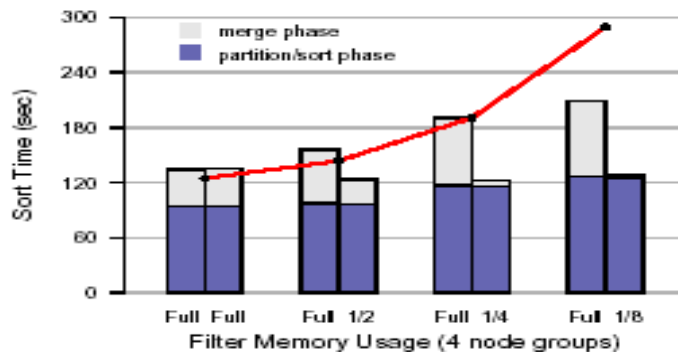
Results



(a) Regular partitioning of input data and Partitioner filter output



(b) Irregular partitioning of input data





Parallel Support

- ◆ Groups of filter
 - Multiple instances handle different units of work at the same time
 - When to create new instances or reuse
- ◆ Multiple copies of filter
 - Either a filter is a parallel program – 1-filter
 - Or multiple instances of a filter– n-filter
- ◆ Transparent copies
 - Filter unaware of the copy



Transparent Copies



- ◆ Filter service decides which one of the copies gets working data
- ◆ Not applicable for all filters, a filter can request not to be copied
- ◆ Performance results in an application where one filter is the most expensive show significant improvement

Transparent Copies (cont ...)

- ◆ How to decide which one of the copies gets data?
- ◆ For copies on one machine, they share a queue
- ◆ For copies on different machines different policies
 - Round Robin
 - Weighted Round Robin
 - Demand-based Sliding Window
 - User Defined

Additions to DataCutter

- ◆ DataCutter integrated with SRB
- ◆ Java filters, can mix and match Java and C++ in an application
 - Of course, overhead of virtual machine creation
 - But, multiple java filters on one machine much better, JIT compilation helps
 - Testing was done with version 1.3
 - Most expensive part is the JNI calls



Current Version

- ◆ <http://www.datacutter.org/>
 - Version 3.1.1 now available
 - Support for external tools to add functionality (like Globus toolkit)
 - Improvements of function names and filter set up

References

- Assigned papers
- Michael Beynon, Chialin Chang, Umit Catalyurek, Tahsin Kurc, Alan Sussman, Henrique Andrade, Renato Ferreira and Joel Saltz. Processing Large-Scale Multidimensional Data in Parallel and Distributed Environments. *Parallel Computing*. 28(5):827-859. May 2002. Special issue on Data Intensive Computing.
- DataCutter 3.0.0 Manual from <http://www.datacutter.org>
- NPACI All Hands Meeting Tutorial, San Diego Supercomputing Center, San Diego, CA, February 9, 2000.
- Very Large Dataset Access and Manipulation, Lawrence Livermore National Laboratory, March 2000.
- Performance Optimization of Component-based Data Intensive Applications, Lawrence Livermore National Laboratory, July 2001.