DataCutter: Middleware for Filtering Very Large Scientific Datasets on Archival Storage Systems

Michael D. Beynon, Renato Ferreira,
Tahsin Kurc, Alan Sussman, Joel Saltz

High Performance Systems Lab
Department of Computer Science
University of Maryland, College Park, MD 20742

http://www.cs.umd.edu/projects/hpsl/

Multi-dimensional Datasets

Spatial/multi-dimensional multi-scale, multi-resolution datasets

- Satellite data processing
- Coupled simulation systems
- The Virtual Microscope

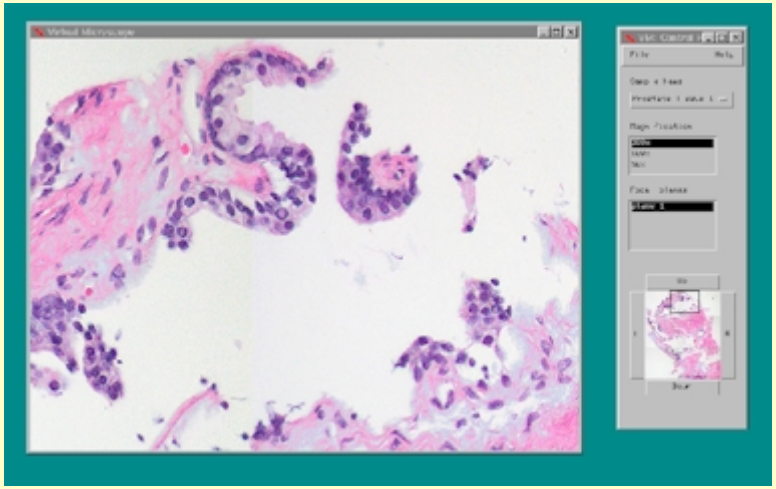
Applications access a subset of the dataset

- Subsetting done by using spatial indexes (e.g., R-trees)

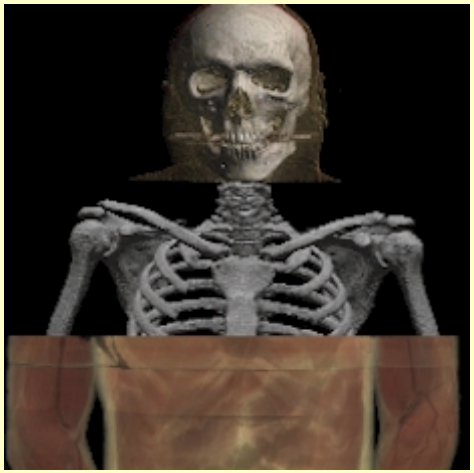
Processing can be performed before returning data to client

- often to reduce volume of data

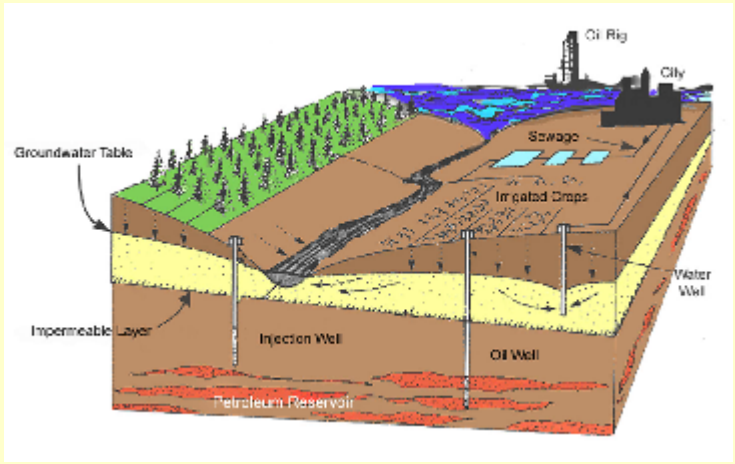
Applications



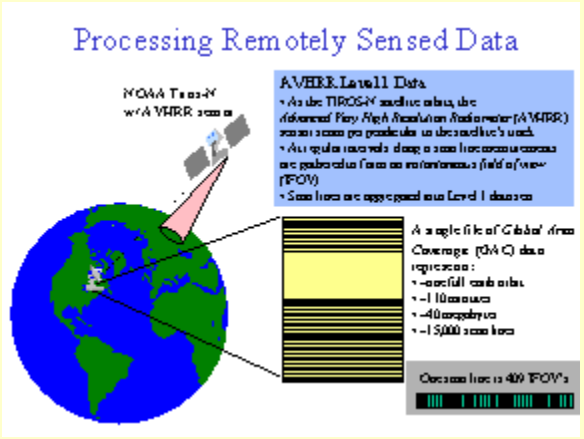
Pathology



Volume Rendering



Surface Groundwater Modeling



Satellite Data Analysis

A Motivating Scenario

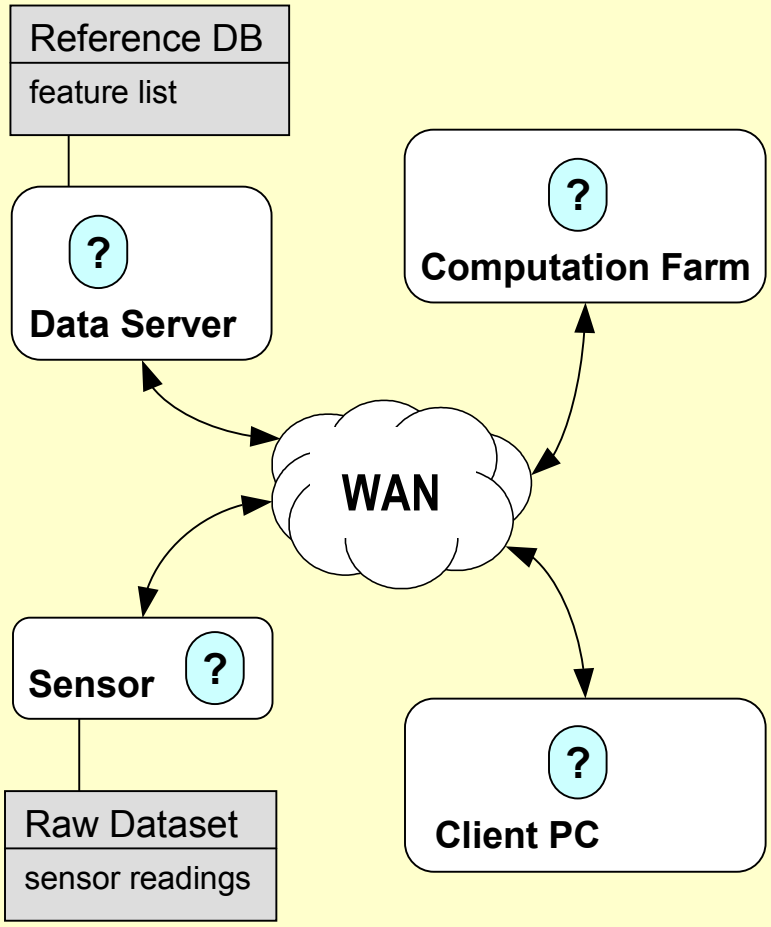
Sample Application:

- generate 3D reconstructed view from new set of sensor readings (computationally intensive)
- compare features with reference db

Configuration:

- remote data server - reference db
- sensor host - large raw readings
- parallel computation farm available

How to design application?



A Motivating Scenario (2)

Application :

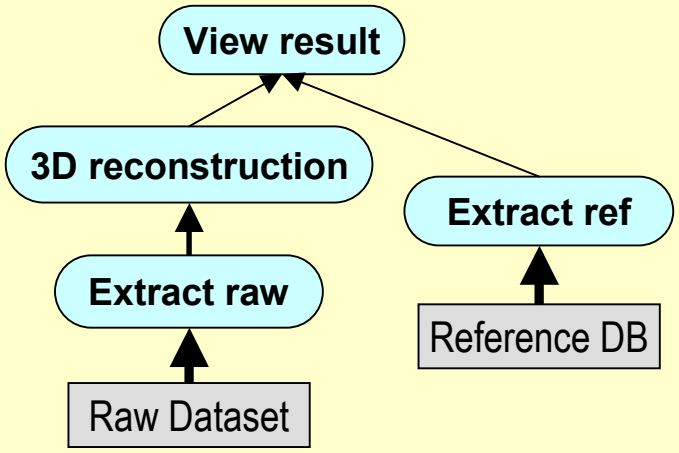
// process relevant raw readings

// generate 3D view

// compute features of 3D view

// find similar features in reference db

// display new view and similar cases



A Motivating Scenario (2)

Application :

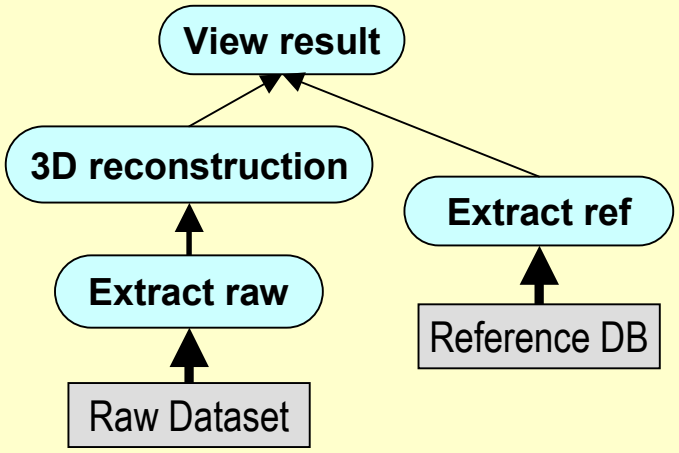
// process relevant raw readings

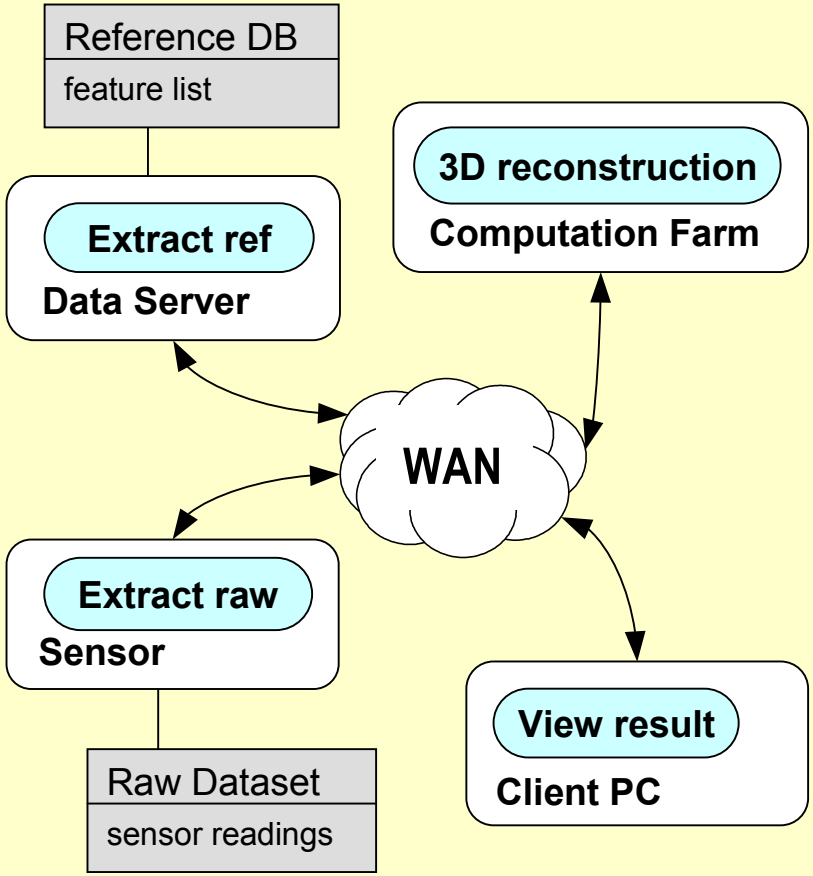
// generate 3D view

// compute features of 3D view

// find similar features in reference db

// display new view and similar cases





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/disks

DataCutter Middleware

Targets multi-dim datasets in archival storage systems

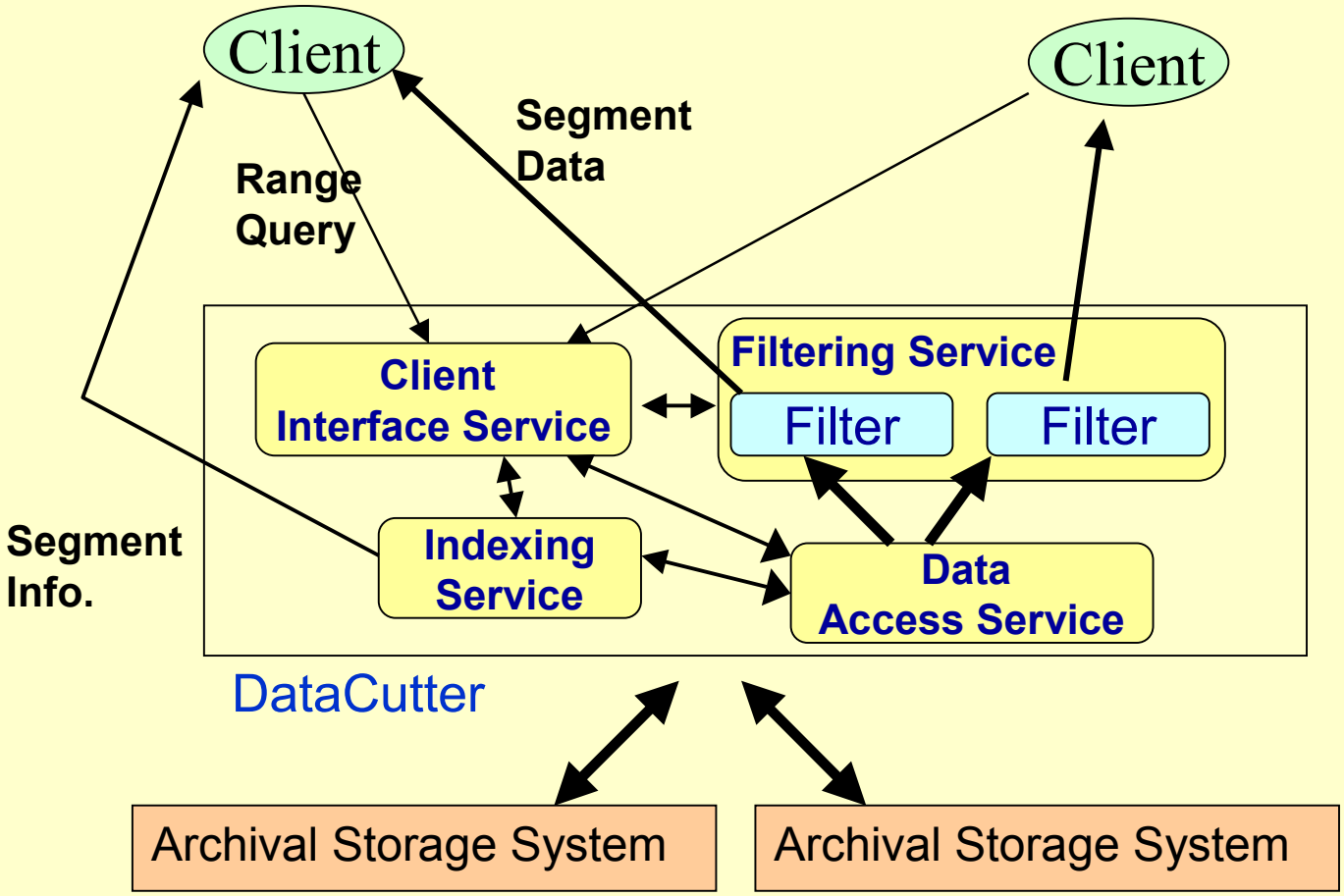
Subsetting through *Range Queries*

- range defines a hyperbox in the multi-dimensional attribute space underlying the dataset
- retrieve items whose coordinates fall within box

Restricted application processing as *Filters*

- intended to execute near (LAN) storage system
- purpose is to reduce amount of data transmitted to client

DataCutter Architecture



Subsetting

Datasets are partitioned into segments

- used to index the dataset, unit of retrieval

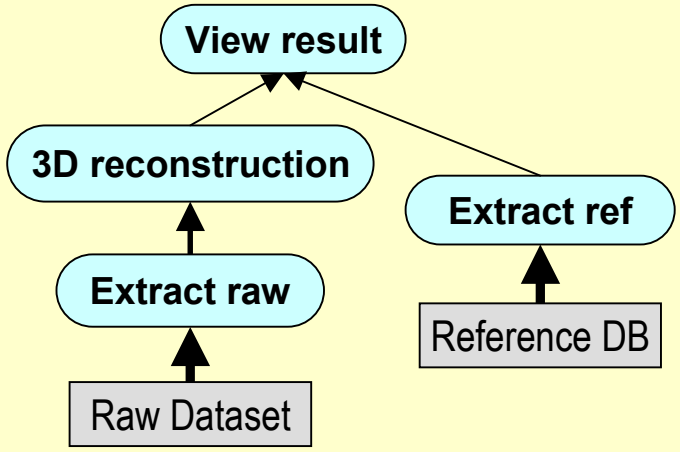
Indexing very large datasets

- Multi-level hierarchical indexing scheme
- Summary index files -- for a collection of segments or detailed index files
- Detailed index files -- to index the individual segments

Application Filters

Purpose: Specialized user code for processing data segments before returning them to the client

- based on Active Disks [Acharya, Uysal, Saltz]
- filters are the unit of computation
  - high level tasks
  - init,process,finalize interface
- streams are how filters communicate
  - unidirectional buffer pipes



Placement

The dynamic assignment of filters to particular hosts for execution is placement (mapping)

Possible Optimization Goals:

Communication

- leverage filter affinity to dataset (contrast with Distributed Storage)
- minimize communication volume on slower connections
- co-locate filters with large communication volume

Computation

- expensive computation on faster, less loaded hosts

Current Software Infrastructure

Prototype implementation of filter framework

- C++ language binding
- one thread for each instantiated filter
- manual placement; wide-area execution service

Prototype indexing/data access service

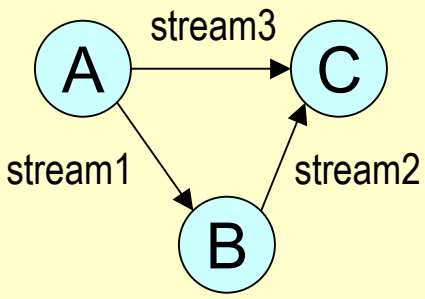
- UNIX filesystem, HPSS archival storage
- R-tree indexes supported
- being integrated with NPACI Storage Resource Broker (SRB)

Filter Framework

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

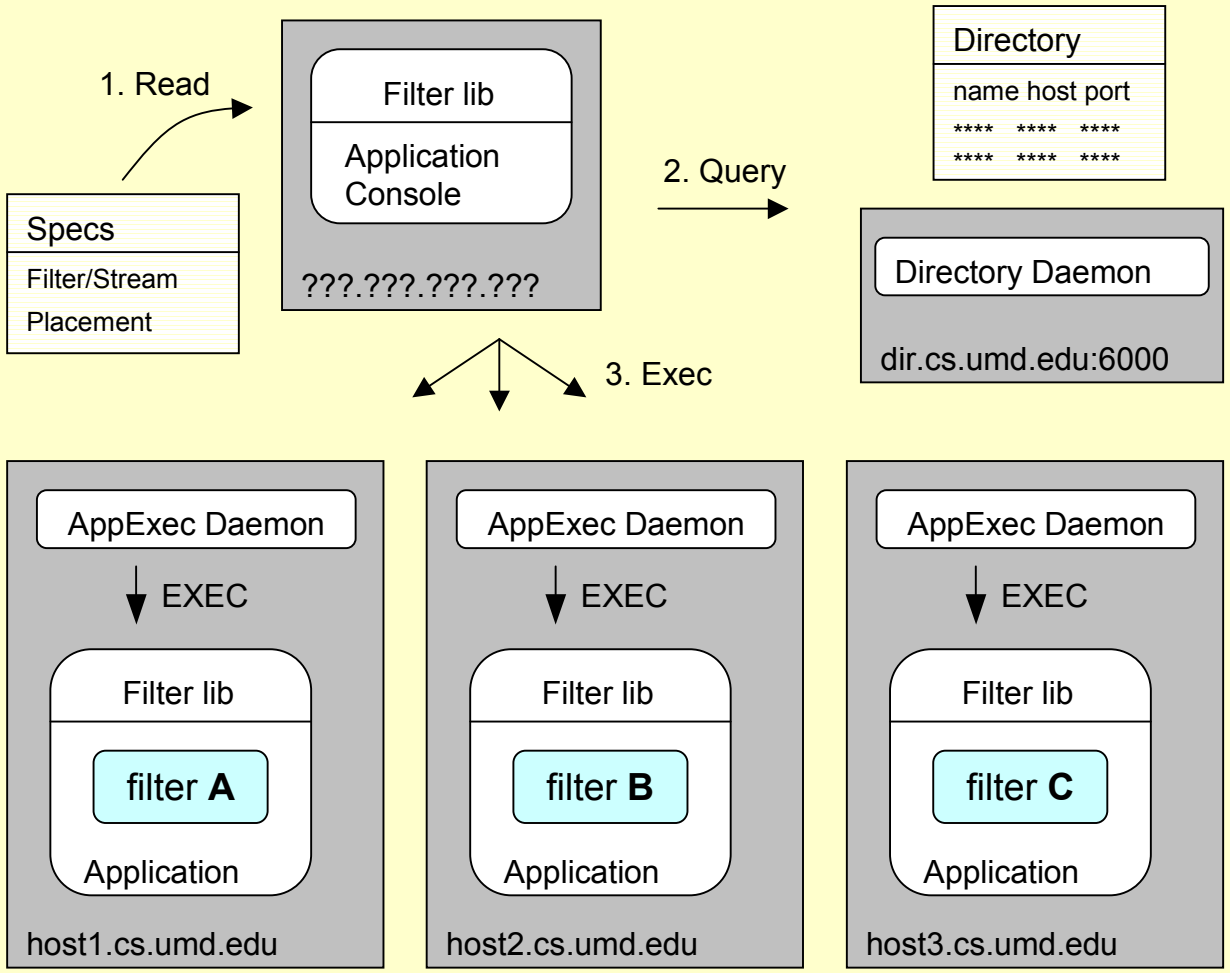
Filter Connectivity / Placement

[filter.A]
outs = stream1 stream3
[filter.B]
ins = stream1
outs = stream2
[filter.C]
ins = stream2 stream3



[placement]
A = host1.cs.umd.edu
B = host2.cs.umd.edu
C = host3.cs.umd.edu

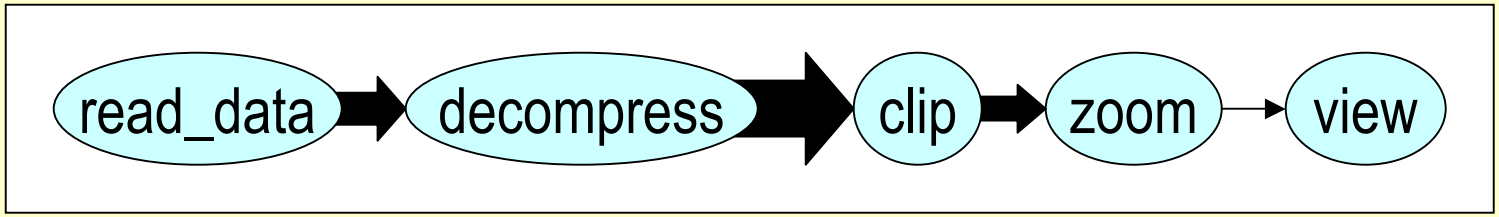
Execution Service



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

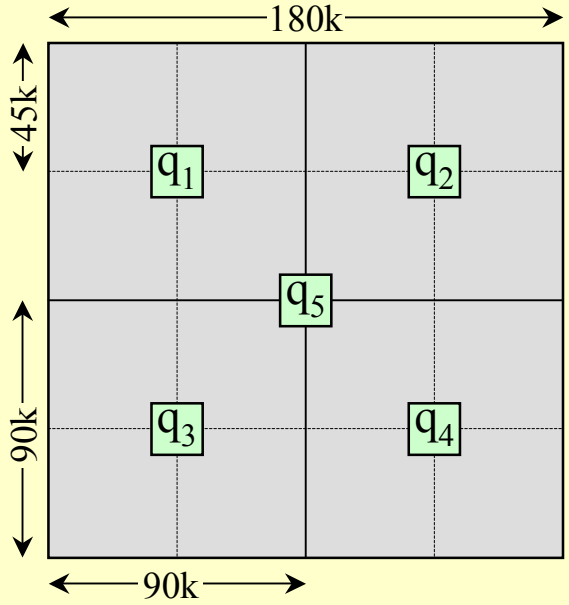
- pipeline style processing



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 image chunks)
- 250GB JPEG compressed dataset (5.6TB uncompressed), 1440k x 1440k RGB pixels (1600 x 1600 image chunks)
- Queries: 4500x4500 q1-q5,q6; 9000x9000 q7; 18000x18000 q8 pixels
- {server}--{client}
  - server host = SP node (1CPU or 2CPU nodes)
  - client host = User Workstation (SPARC 10 or Ultra1)

Experimental Setup (2)

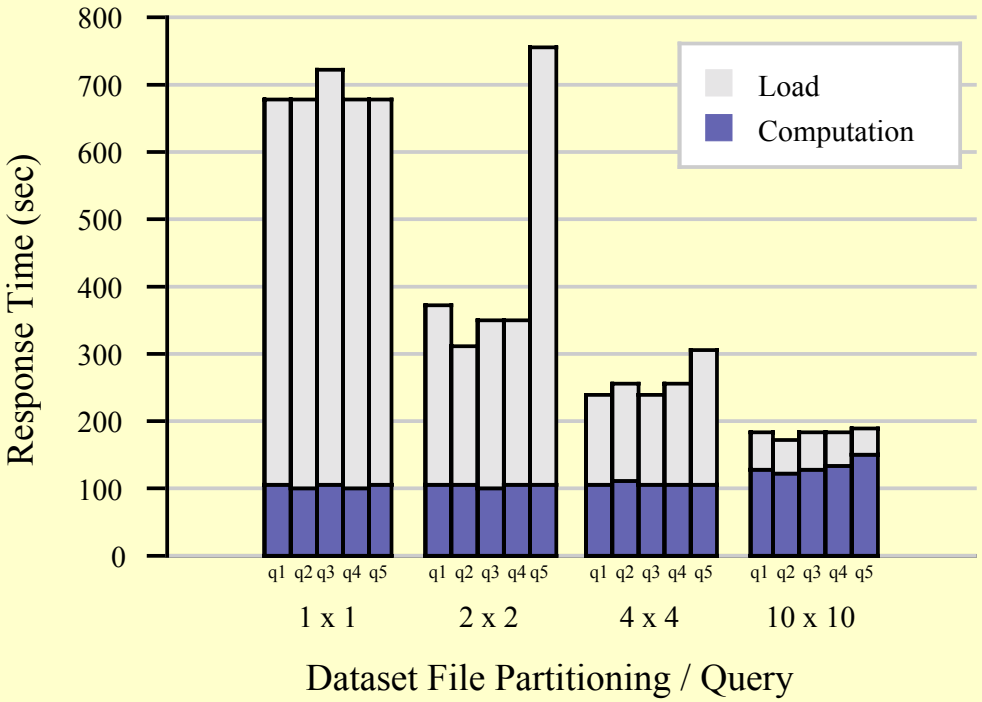


4GB dataset

Filter	*Total Volume*	*Volume / Chunk*
read_data	3.60 MB	102.52 KB
decompress	83.42 MB	2373.04 KB
clip	57.83 MB	1645.02 KB
zoom (no)	57.83 MB	1645.02 KB
zoom (8)	.90 MB	25.70 KB

q_5 sample volume

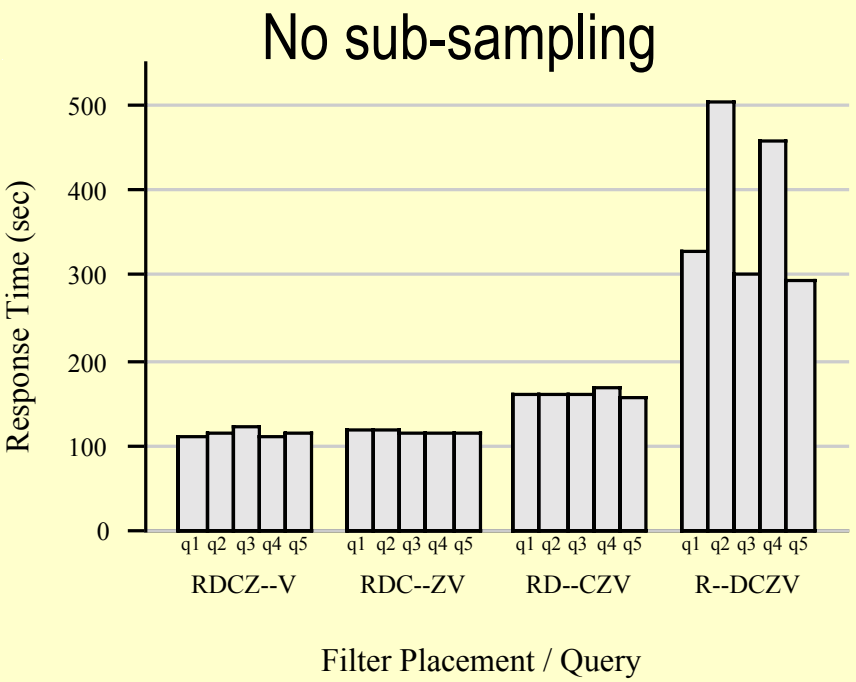
Experiment: Dataset Tiling

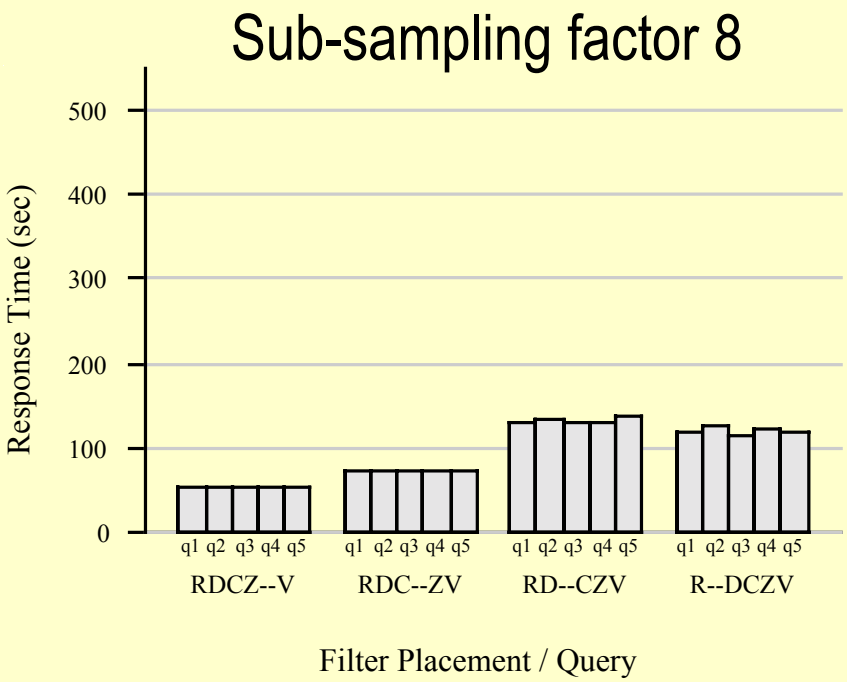


Setup: 4GB dataset, cold cache

Point: dataset organization can be controlled, and affects performance

Experiment: Computation / Volume

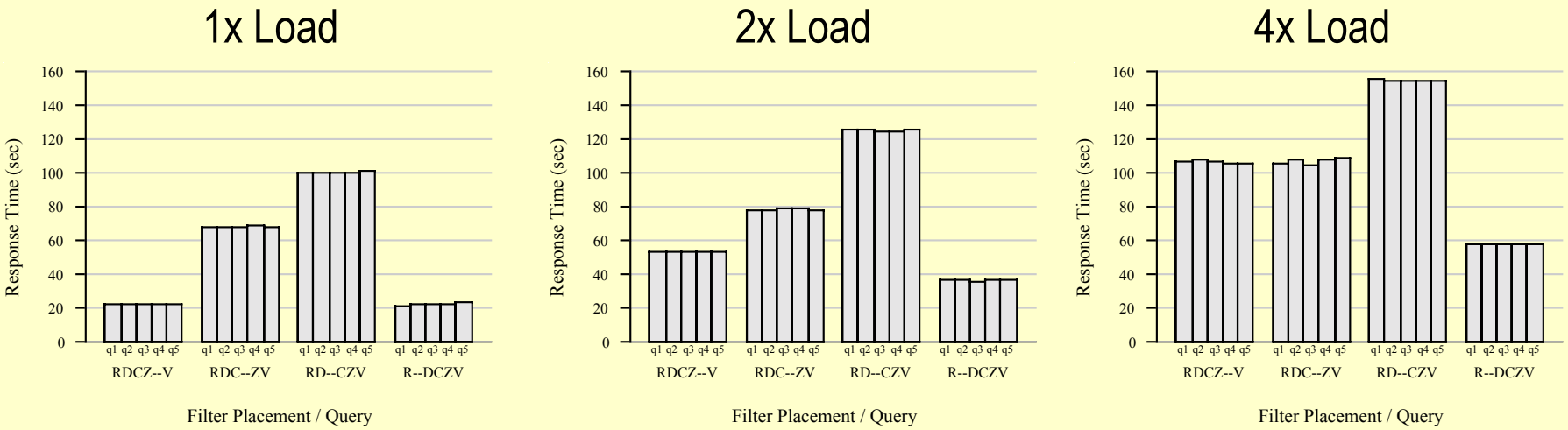




Setup: 4GB dataset, warm cache

Point: amount of computation and communication makes a significant difference for efficient placement.

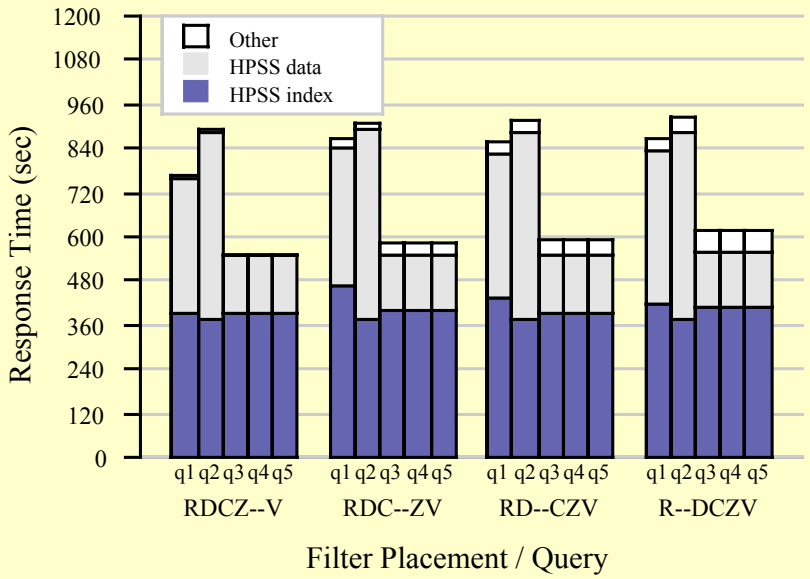
Experiment: Server Load

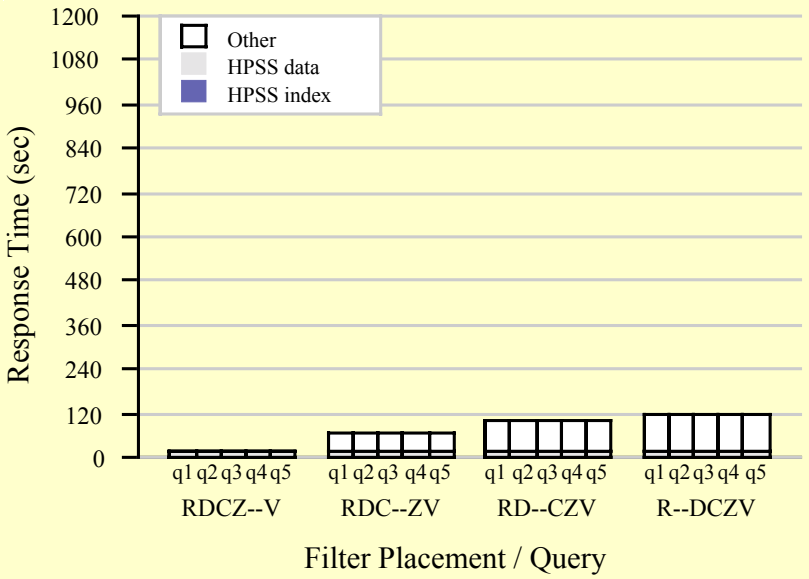


Setup:4GB dataset, zoom 8, warm cache

Point: server load has a dramatic effect on efficient placement.

Experiment: Cold vs. Warm Cache

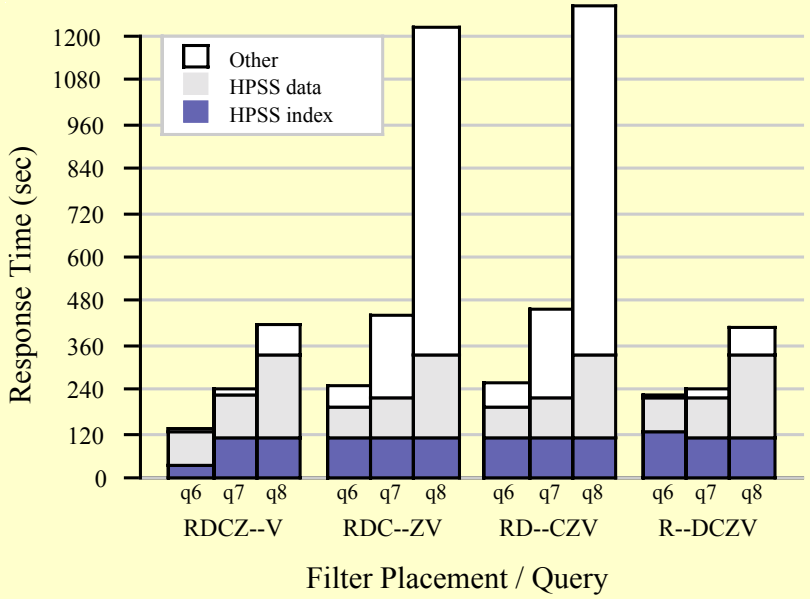


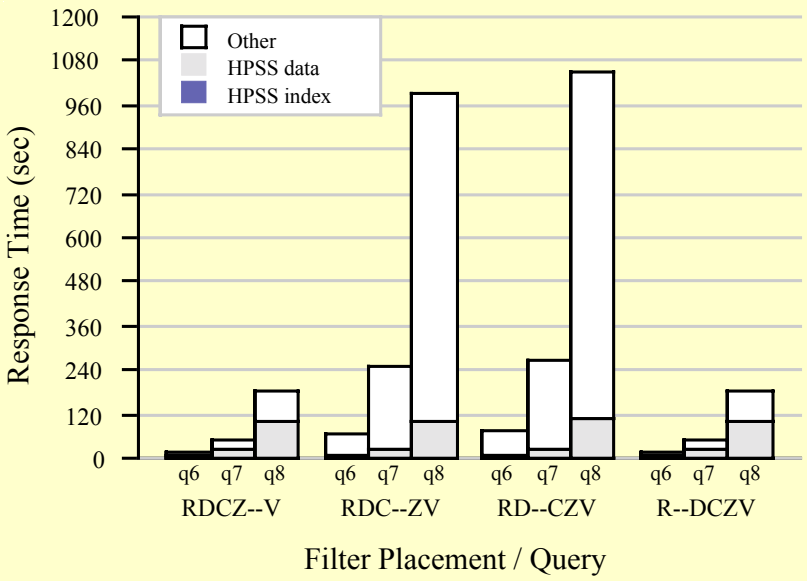


Setup: 250GB dataset, zoom 8

Point: q1-q5 dominated by HPSS access, tape storage location dependent

Experiment: Scaling Query Size





Setup: 250GB dataset, zoom 8

Point: q6-q8 larger, but not dominated as much HPSS access, network volume still important

Experiment: Filter Overhead

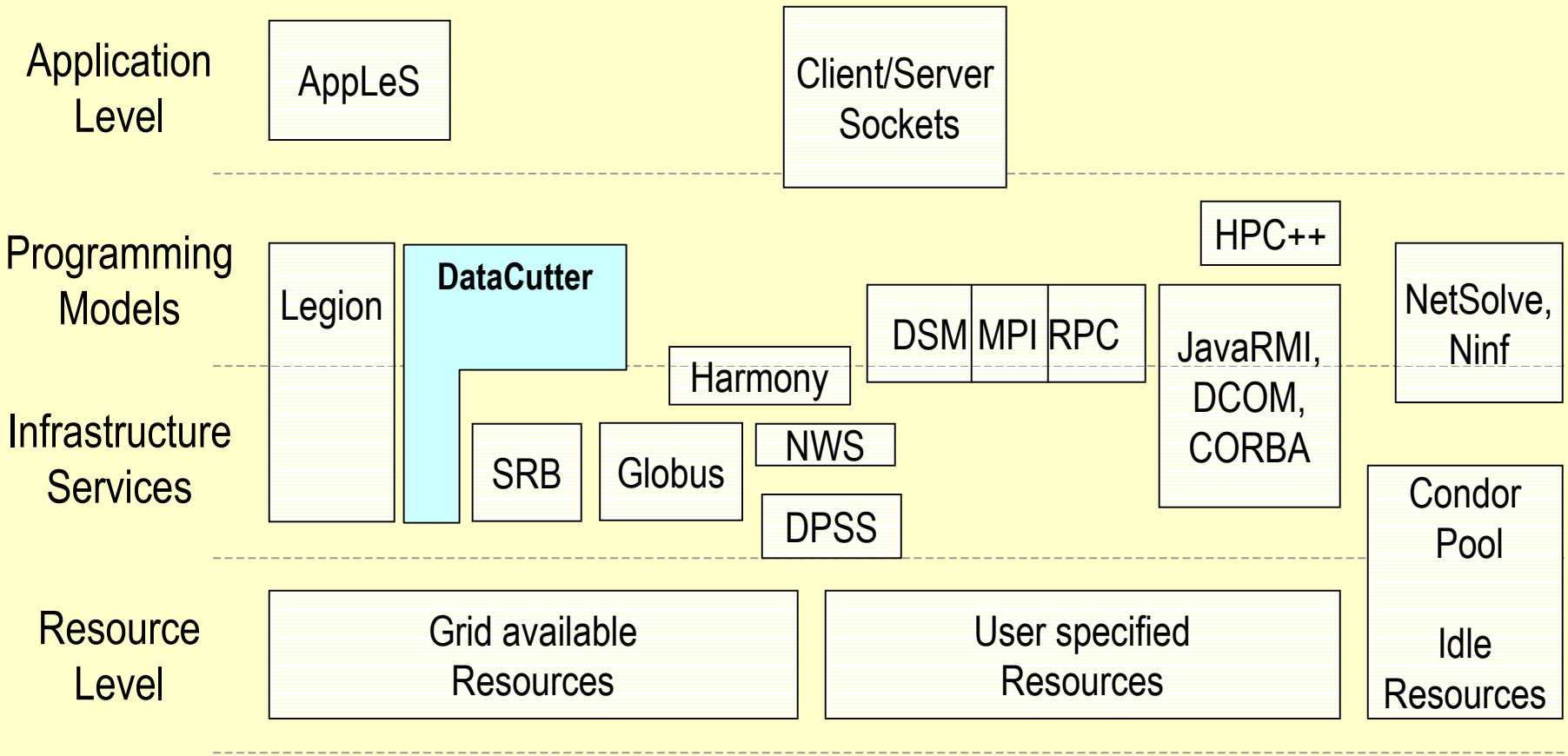
6% - 30% overhead

Point: overhead is not very large for the unoptimized filter prototype

Future Work

- New applications
  - Database (Sort, Select/Project/Join)
  - Visualization
  - Satellite data processing
- Automated placement of filters
- Tighter integration with NPACI Storage Resource Broker (SRB)

Related Work



<End of Talk>

Filter Environment

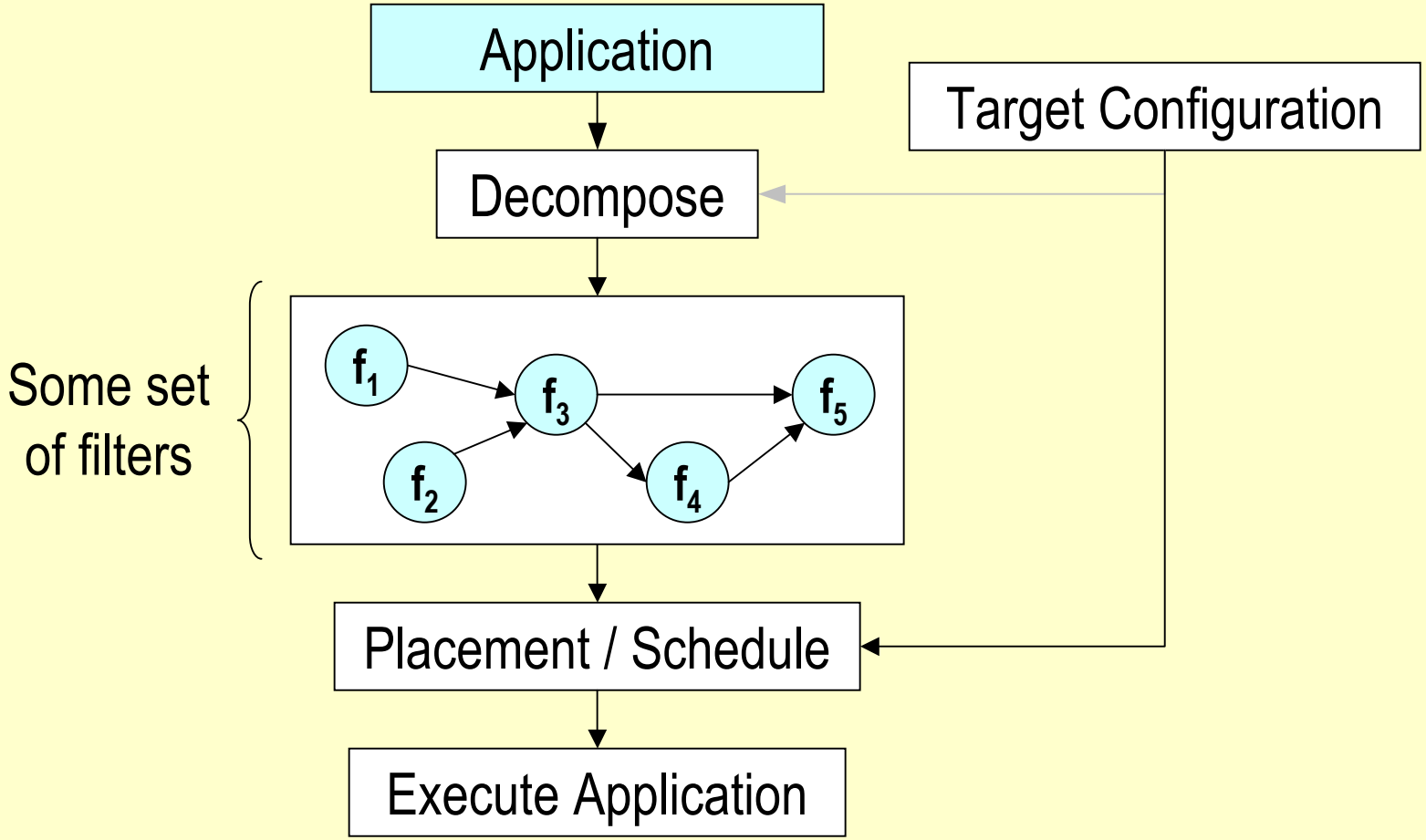
Filter constraints

- communicate with other filters *only* using streams
- cannot change stream endpoints
- may disclose dynamic allocation of memory/scratch space before execution

Advantages

- location independence
- easier scheduling of resources
- filter stop and restart is defined explicitly in model

Manual Restructuring Process

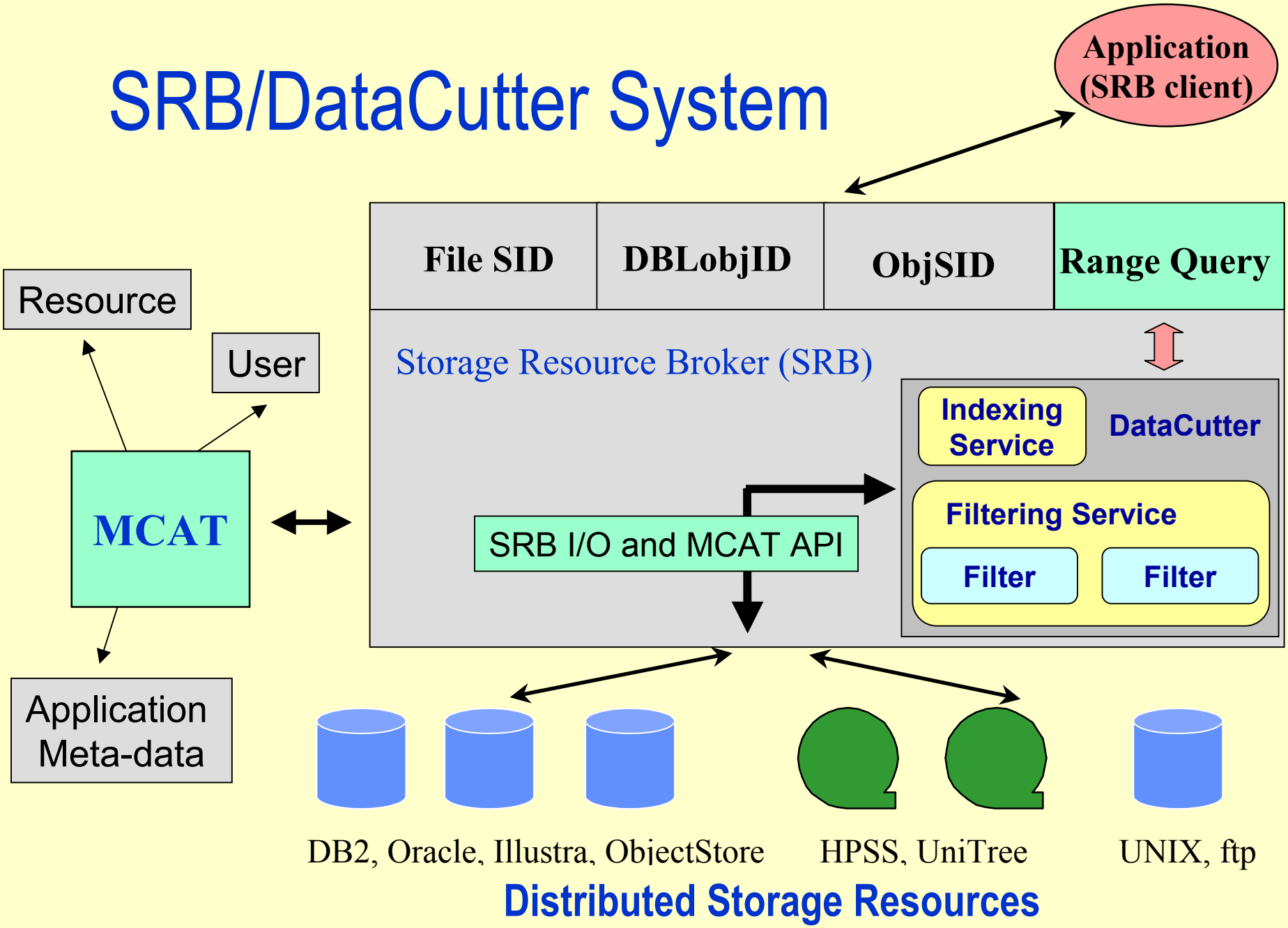


SRB/DataCutter integration

API for SRB clients to

- create indexes (R-tree index)
- subset data sets
- carry out restricted filter operations on portions of files (data segments) before returning them to the client (to perform filtering or aggregation to reduce data volume)
- only one filter can run per query

SRB/DataCutter System



Extensions to Filter Framework

- cost model based placement 1
- externalize filter library (inter-application) 2
- dynamic attachment 2
  partial filter graphs left running (server applications)
- history feedback into cost models 2
- parallel filters 3

Adaptive Placement

- Applications expose unit-of-work
  - unit of work boundaries are when decisions are made
  - a decision to modify placement is made globally
    - new instance created, and all *new* work uses it
    - for filters with state, model defines explicit state transfer
      old instance sends data over temporary special stream to new instance from <old>:finalize() to <new>:init()
    - not really “migrated”, but effect is similar without high cost
  - ex: many clients or server load high, push filters toward clients

Why not XXX programming model?

- Ex: MPI, PVM, etc. (can be used instead)
  - decide which machines to use based on dataset affinity
  - detect where running and use functional parallelism to control which filters runs
  - add disclosure of comm. pattern and resource usage
  - add mechanism for shutting down nodes and adding new during computation for adaptation
- Why use filter-stream programming model
  - because it provides these services in a natural way
  - encourages minimized state and resource usage

What will not work well?

- Fine grained tasks
- Fine coordination between filters
- Long request-response chains with dependencies between queries

Annotation Detail

- Static annotations
  - host (cpu_power, cost)
  - filter (cpu_time, size_transform, num_transform, content_transform)
  - stream (size_unit)
- Runtime properties
  - host (cpu_load, memory_free, scratch_space_free)
  - network (bw, latency, jitter)

Related Projects

- ADR (Active Data Repository) [Chang et al.]
  - Support optimized associative access and processing of persistent multi-dimensional data
  - Customizable for various applications
  - Integrate and overlap a wide range of order-independent user-defined aggregation operations with data retrieval
  - Target a shared-nothing architecture with disk farm
  - Compiler to generate ADR code from Titanium, a Java dialect
  - Caching proxy for co-located clients

Related Projects

- Active Disks [Uysal, Acharya, Saltz]
  - Putting significant processing and memory on disks
  - Execution of data-intensive parts of algorithms on disk resident processors
  - Stream-based programming model