Performance Optimization of Component-based Data Intensive Applications

Alan Sussman
Michael Beynon
Tahsin Kurc
Umit Catalyurek
Joel Saltz
University of Maryland
http://www.cs.umd.edu/projects/adr

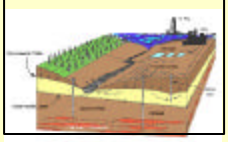
Outline

- Motivation
- Approach
- Optimization – Group Instances
- Optimization – Transparent Copies
- Ongoing Work

Targeted Applications

Pathology

Volume Rendering



Surface Groundwater Modeling

Satellite Data Analysis

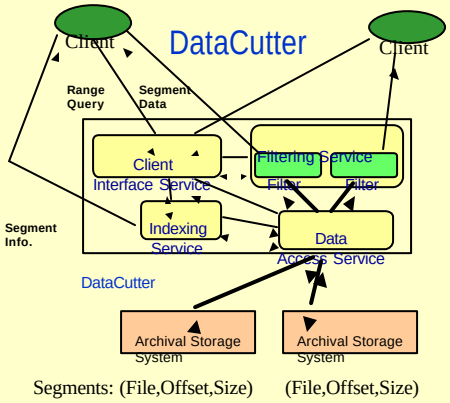
Runtime Environment

Heterogeneous Shared Resources:

- Host level: machine, CPUs, memory, disk storage
- Network connectivity

Many Remote Datasets:

- Inexpensive archival storage clusters (1TB ~ $10k)
- *Islands* of useful data
- Too large for replication



Segments: (File,Offset,Size) (File,Offset,Size)

DataCutter

Indexing Service

- Multilevel hierarchical indexes based on spatial indexing methods – e.g., R-trees
  - Relies on underlying multi-dimensional space
  - User can add new indexing methods

Filtering Service

- Distributed C++ component framework
- Transparent tuning and adaptation for heterogeneity
- Filters implemented as threads – 1 process per host

Versions of both services integrated into SDSC SRB

Indexing - Subsetting

Datasets are partitioned into segments

- used to index the dataset, unit of retrieval
- Spatial indexes built from bounding boxes of all elements in a segment

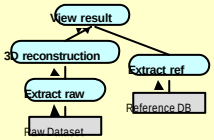
Indexing very large datasets

- 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

Filter-Stream Programming (FSP)

Purpose: Specialized components for processing data

- based on Active Disks research [Acharya, Uysal, Saltz: ASPLOS'98], dataflow, functional parallelism, message passing.
- filters – logical unit of computation
  - high level tasks
  - **init**,**process**,**finalize** interface
- streams – how filters communicate
  - unidirectional buffer pipes
  - uses fixed size buffers (min, good)
- manually specify filter connectivity and filter-level characteristics



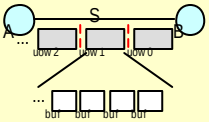
Placement

- The dynamic assignment of filters to particular hosts for execution is placement (or 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

FSP: Abstractions

Filter Group

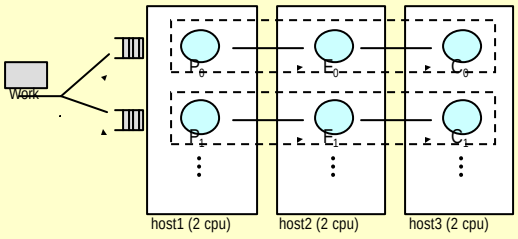
- logical collection of filters to use together
- application starts filter group *instances*



Unit-of-work cycle

- "work" is application defined (ex: a query)
- work is appended to running instances
- init(), process(), finalize() called for each work
- **process()** returns { EndOfWork | EndOfFilter }
- allows for adaptivity

Optimization - Group Instances

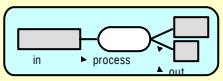


Match # instances to environment (CPU capacity, network)

Experiment - Application Emulator

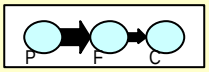
Parameterized dataflow filter

- consume from all inputs
- compute
- produce to all outputs

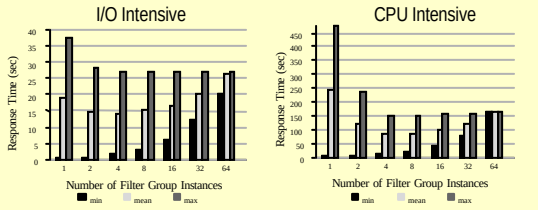


Application emulated:

- process 64 units of work
- single batch



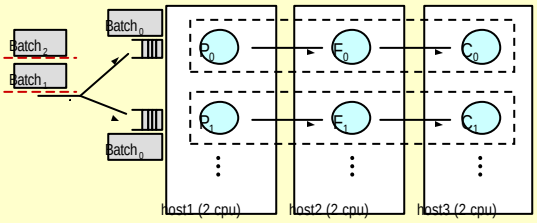
Instances: Vary Number, Application



Setup: UMD Red Linux cluster (2 processor PII 450 nodes)

Point: # instances depends on application and environment

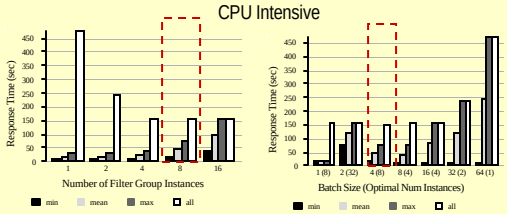
Group Instances (Batches)



Work issued in instance batches until all complete.

Matching # instances to environment (CPU capacity)

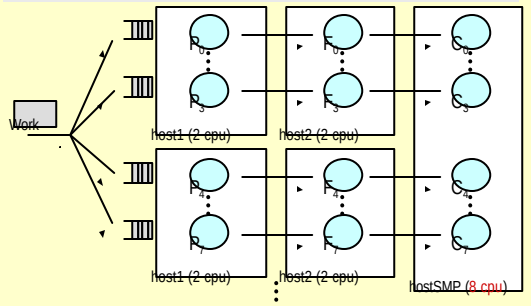
Instances: Vary Number, Batch Size



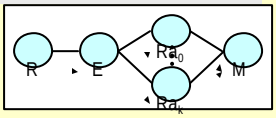
Setup: (Optimal) is lowest *all* time

Point: # instances depends on application and environment

Adding Heterogeneity



Optimization - Transparent Copies



FSP Abstraction

- replicate *individual* filters
- transparent
  - work-balance among copies
  - better tune resources to actual filter needs
- Provide "single-stream" illusion
  - Multiple producers and consumers, deadlock, flow control
  - Invariant: $UOW_i < UOW_{i+1}$
- Problem: filter state

Runtime Workload Balancing

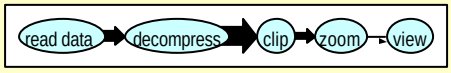
Use local information:

- queue size, send time / receiver acks

- Adjust number of transparent copies
- Demand based dataflow (choice of consumer)
  - Within a host – perfect shared queue among copies
  - Across hosts
    - Round Robin
    - Weighted Round Robin
    - Demand-Driven sliding window (on buffer consumption rate)
    - User-defined

Experiment – Virtual Microscope

- Client-server system for interactively visualizing digital slides
  Image Dataset (100MB to 5GB per focal plane)
  Rectangular region queries, multiple data chunk reply
- Hopkins Linux cluster– 4 1-processor, 1 2-processor PIII-800, 2 80GB IDE disks, 100Mbit Ethernet
- Decompress filter is most expensive, so good candidate for replication
- 50 queries at various magnifications, 512x512 pixel output

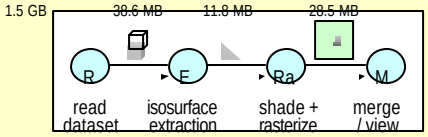


Virtual Microscope Results

R-D-C-Z-V	Response Time (seconds) Average	400x	200x	100x	50x
h-h-h-h-h	2.10	0.38	0.73	1.73	6.95
h-g-g-g-g	1.49	0.37	0.62	1.27	4.60
h-g(2)-g-g-g	1.15	0.39	0.50	0.95	3.41
h-g(2)-g(2)-g-g	1.15	0.37	0.49	0.95	3.43
h-g(4)-g(2)-g-g	1.17	0.39	0.50	0.96	3.50
h-g(2)-g(2)-b-b	1.68	0.45	0.68	1.27	5.34
g-g-g-g-g	1.44	0.33	0.58	1.26	4.46
g-g(2)-g-g-g	1.08	0.33	0.45	0.92	3.24

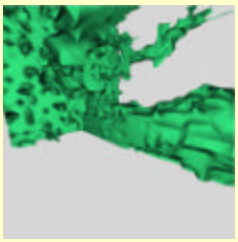
Experiment - Isosurface Rendering

- UT Austin ParSSim species transport simulation
  Single time step visualization, read all data
- Setup: UMD Red Linux cluster (2 processor PII 450 nodes)

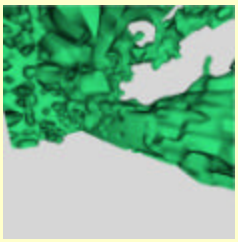


0.64s 4.3% | 1.64s 11.2% | 11.67s 79.5% | 0.73s 5.0% | = 14.68s (sum) = 12.65s (time)

Sample Isosurface Visualization



V = 0.35



V = 0.7

Transparent Copies: Replicate Raster

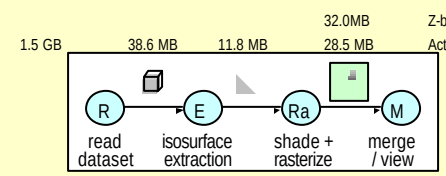
	1 node	2 nodes	4 nodes	8 nodes
1 copy of Raster	12.18s	7.32s	4.17s	3.00s
2 copies of Raster	8.16s (33%)	5.70s (22%)	3.88s (7%)	3.24s (– 8%)

Setup: SPMD style, partitioned input dataset per node
Point: copies of bottleneck filter enough to balance flow

Experiment – Resource Heterogeneity

- Isosurface rendering on Red, Blue, Rogue Linux clusters at Maryland
  - Red – 16 2-processor PII-450, 256MB, 18GB SCSI disk
  - Blue – 8 2-processor PIII-550, 1GB, 2-8GB SCSI disk + 1 8-processor PIII-450, 4GB, 2-18GB SCSI disk
  - Rogue – 81-processor PIII-650, 128MB, 2-75GB IDE disks
  - Red, Blue connected via Gigabit Ethernet, Rogue via 100Mbit Ethernet
- Two implementations of Raster filter – z-buffer and active pixels (the one used in previous experiment)

Experimental setup



Active Pixel	0.64s	1.64s	11.67s	0.73s	= 14.68s
Z-buffer	0.68s	1.65s	9.43s	0.90s	= 12.66s

Experiment to follow combines R and E filters, since that showed best performance in experiments not shown

Varying number of nodes

		Active Pixel Rendering				Z-buffer Rendering			
Configuration	#Ra	1 node	2 nodes	4 nodes	8 nodes	1 node	2 nodes	4 nodes	8 nodes
RE-Ra-M	1(0)	n/a	12.7	4.8	2.9	n/a	11.3	7.7	10.7
	1(1)	12.2	7.3	4.2	3.0	10.7	7.9	7.9	11.5
RE-Ra-M	2(0)	n/a	7.7	3.2	2.6	n/a	9.3	10.5	19.0
	2(2)	8.2	5.7	3.9	3.2	8.6	8.9	11.7	20.8

Only Red nodes used – each one runs 1 RE, 0, 1, or 2 RA, and one runs M

Heterogeneous nodes

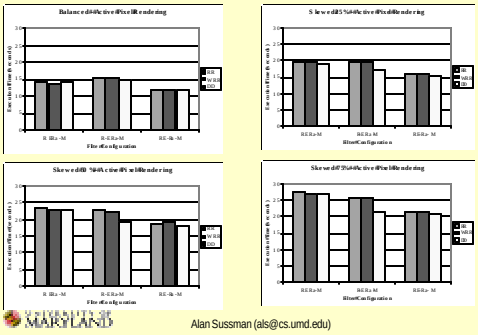
Conf.	1 data node			2 data nodes			4 data nodes			8 data nodes		
	RR	WRR	DD	RR	WRR	DD	RR	WRR	DD	RR	WRR	DD
RE-Ra-M	7.3	3.0	3.0	5.7	2.6	3.0	4.4	3.0	3.5	3.8	2.9	3.8
R-ERa-M	8.2	4.0	4.2	6.5	4.1	4.0	5.1	4.1	4.1	4.0	3.7	4.2

Active Pixel algorithm on 8-processor Blue node + Red data nodes
Blue node runs 7 Ra or ERa copies and M, Red nodes each run 1 of each except M

Skewed data distribution

- Experimental setup
  - 25GB dataset from UT ParSSim (bigger grid than earlier experiments)
  - Hilbert curve declustering onto disks on 2 Blue, 2 Rogue nodes
  - Skew moves part of data from Blue to Rogue nodes

Skewed data distribution



Experimental Implications

Multiple group instances
- Higher utilization of under-used resources
- Reduce application response time
- but ... requires work co-execution tolerance

Transparent copies can help
- Most filter decompositions are unbalanced
- Heterogeneous CPU capacity / load
- but ... requires buffer out-of-order tolerance

Ongoing and Future Work

Ongoing and Future work

- Automated placement, instances, transparent copies
  - predictive (cost models)
  - adaptive (work feedback)
- Filter accumulator support (partitioning, replication)
- Java filters
- CCA-compliant filters
- Very large datasets – including HDF5 format
  - Using storage clusters at UMD and OSU, then testbed at LLNL