

# Active Disks:

## Programming Model, Algorithms and Evaluation

By Anurag Acharya, Mustafa Uysal, Joel Saltz

Published in ASPLOS VIII (1998)

Presented by Ryan Farrell

## Overview

- **Motivation**
  - Large Data sets are growing more quickly than processor speeds, but Customers want to be able to process their data sets in the same amount of time
    - Customers doubling storage every 5 months
    - CPU speeds double every 18 months [Moore's law]
  - While CPUs are “cheap” compared to storage (\$100 vs. \$1100-1750), CPU can only keep a small number of high-performance drives busy.
- **Proposed Solution**
  - Offload processing onto disk drives using drives' embedded processor.

## Active Disks Architecture

- **Host-resident Component**
  - Primarily for coordination, scheduling and combining results
- **Disk-resident Component (“Disklets”)**
  - Downloaded to from host, does lion's share of processing
  - Stream processor:
    - Reads input streams (files) from disk
    - Writes output streams (files) to disk
    - Optional long-term scratch space

## Disk/Host-level OS

- **DiskOS**
  - **Memory Management** – allocation and management of buffers
  - **Stream Communication** – possible to overlap data movement and communication
  - **Disklet Scheduling** – disklets ready to run whenever there is data available
- **Host-level OS Support**
  - **Installation** of disklets
  - **Management** of host-resident buffers (streams)

## Disklet/Host Code

```

disklet convolve(hoststream in,
                hoststream out)
integer hsize[3][3], vsize[3];
integer i,j,k,l,m;
buffer buffer;

function init(integer filter[3][3],
             integer imgsize) {
  copy(filter,buffer);
  imgsize = imgsize;
}

function read {
  /* get next buffer */
  buffer = getnextbuffer(in);
  /* perform convolution */
  for (i = 0; i < vsize[0]; i++) {
    for (j = 0; j < vsize[1]; j++) {
      for (k = 0; k < 3; k++) {
        for (l = 0; l < 3; l++) {
          for (m = 0; m < 3; m++) {
            * outptr[i][j][k] +=
              hsize[i][k] *
            }
            buffer[j][k] = *i;
          }
        }
      }
    }
  }
  send(out,buffer);
}
end

/* host-resident code
integer filter[3][3] = {{-1,-2,-1},
                       {0, 0, 0},
                       {1, 2, 1}};

for (i = 0; i < HOSTSD; i++) {
  /* open image file on every disk */
  apoint[frames] = "image1.tif";
  f[i] = open(frames,0,MODERT);
  fsize[i] = filesize(f[i]);
  /* create a stream for every disk.
  * output from disklet; 256K buffer */
  apoint[frames] = "convimage1.tif";
  stream[i] = open_streams_READ_256K(44);

  /* install the disklet */
  install_convolve(filter,i);
  /* launch disklet; while file as input
  * input stream open has descriptor,
  * offset and length; 256K buffer */
  run_convolve(install(f[i],fsize[i],0)(44),
              stream[i]);
}

/* while not all streams have terminated.
* str_select blocks, returns a stream
* that has new data */
while (i = str_select(streams)) != NULL {
  buf = getnextbuffer(buf);
  process(buf);
}

for (i = 0; i < HOSTSD; i++) {
  close(f[i]);
  close(stream[i]);
}
}

```

Figure 1. Pseudo-code for partitioned version of convolution filtering. The filter used in this example detects horizontal lines in images.

## Algorithms (1/6)

- **SQL SELECT**
  - Filter tuples from a relation based on a user-specified predicate
- **Conventional Disk Algorithm**
  - Read **all tuples** from disk, keep those that match filter
- **Active Disk Algorithm**
  - On disk, read tuples from input stream (file), write matching tuples to output file, send **only matching results** when finished (or partial results when output file fills up)

## Algorithms (2/6)

- **SQL GROUP-BY**
  - Compute a vector of aggregates indexed by a list of attributes
  - **Conventional Disk Algorithm**
    - Read **all tuples** from disk, accumulating group-by results
  - **Active Disk Algorithm**
    - Perform **local** group-bys
    - When out of space, **ship partial result** to host.

## Algorithms (3/6)

- **External Sort**
  - Sort database tuples (details?)
  - **Conventional Disk Algorithm** (NOWsort)
    - Reader-Thread **reads data** from disk and moves tuple pointers to buckets
    - Writer-Thread sorts each bucket with partial-radix sort
  - **Active Disk Algorithm** (NOWsort)
    - Partitioner divides records into buckets and when bucket fills, **sends to host**
    - When host buffer fills, **host sends to Sorter** which sorts and **writes to output stream**
    - Merging done **locally** on disk, final results sent to host

## Algorithms (4/6)

- **Datacube**
  - In effect compute group-bys for all possible combinations of a list of attributes
  - **Conventional Disk Algorithm** (PipeHash)
    - Schedule group-bys as sequence of pipelines, each computed **from disk** and **written back to disk** for next pipeline
  - **Active Disk Algorithm**
    - Separate disklet for each pipeline
    - Perform **local** group-bys, host accumulates partial results and stores for use by later pipelines

## Algorithms (5/6)

- **Image Convolution** (Batch Processing)
  - Image processing operations such as Smoothing, Sharpening Edge Detection,
  - **Conventional Disk Algorithm**
    - Read **all images**, concatenate into single file and stripe across disks (then process?)
  - **Active Disk Algorithm**
    - Does convolution of individual images **locally**
    - Sends **processed image** to host

## Algorithms (6/6)

- **Compositing Satellite Images**
  - Project individual satellite images onto much larger composite image (image registration)
  - **Conventional Disk Algorithm** (from NASA)
    - Read **large chunks** of individual satellite images,
    - Composites with accumulator for each output pixel.
  - **Active Disk Algorithm**
    - Perform pre-processing and mapping at disk
    - Perform most of composition **locally**

## Performance Comparison (Simulated)

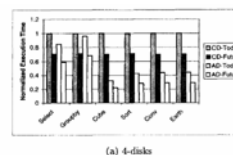


Figure 4: Comparison of Active Disks with conventional disks. The legend indicates the architecture type (CD/AD) and the configuration (Today/Future). These results are for the smaller datasets.

## Performance Comparison (Simulated)

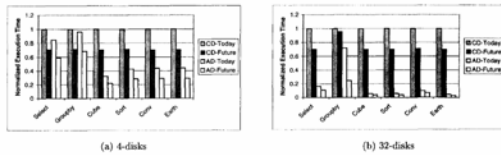


Figure 4: Comparison of Active Disks with conventional disks. The legend indicates the architecture type (CD/AD) and the configuration (Today/Future). These results are for the smaller datasets.

## Performance Comparison (Simulated)

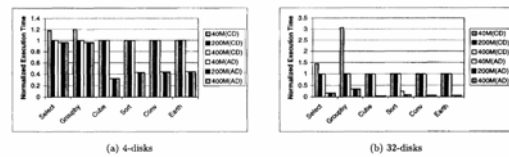


Figure 5: Impact of variation in the interconnect bandwidth. These results are for Today configurations and the smaller datasets.

## Performance Comparison (Simulated)

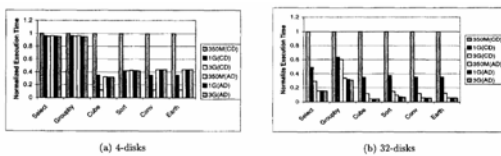


Figure 8: Impact of variation in the central processor. These results are for Today configurations and the smaller datasets.

## Results/Conclusion

### • Theoretical Performance

- Active Disk architecture outperforms Conventional disks **1-3x** for 4-disk configuration and **3-30x** for 32-disk configuration

### • Today (2005)

- Today, disks are relatively cheap, \$500/TB  
\$0.15/GB on Black Friday) and have larger caches and more powerful controllers.
- Though it seems like a great idea, I couldn't find evidence of mainstream usage.

## What are the Authors doing now?

- Who (At time of publication) - Currently
- **Anurag Acharya** (PostDoc under Saltz) – Professor at UC Santa Barbara.  
– Current Research: “Rapidly Evolvable Network Services”
- **Mustafa Uysal** (PhD under Saltz) – Now At HP Labs  
– Current Research: “Design, implementation and analysis of large-scale storage systems and distributed, data-intensive computing”.
- **Joel Saltz** (Professor) – Chair of Ohio State Biomedical Informatics Department (joint appointments at JHU, UMIACS)  
– Current Research: “Data Intensive and Grid Computing, Distributed and Parallel Systems, High End Medical Applications.”

## Additional Work on Active Disks

- **CMU (dates back to '97)**  
– <http://www.pdl.cmu.edu/Active/>
- **HPCA 2000 (Best Student Paper)**  
– [http://www.hpl.hp.com/personal/Mustafa\\_Uysal/papers/2000-hpca/hpca.pdf](http://www.hpl.hp.com/personal/Mustafa_Uysal/papers/2000-hpca/hpca.pdf)
- **“NAS (Network Attached Storage)”**  
– <http://www.google.com/search?q=NAS+network+attached+storage>