STORAGE SYSTEMS RESEARCH CENTER

Efficiently Identifying Working Sets in Block I/O Streams

Avani Wildani
Ethan Miller
UC Santa Cruz

Lee Ward
Sandia Labs

Sandia National Laboratories

pdsi

Baskin Engineering
UC SANTA CRUZ

Key Ideas

- Grouping data is beneficial
- Even block I/O data can be grouped
  - Grouping block data can lead to high level insights
- The systems research world needs more trace data for modern systems

Why Group? - Power Efficiency

- Reduce the number of spin-ups
- Reduce on-disk seek time if grouping is done within a single disk
- Better pre-fetching while a drive is spinning, reducing further spin-ups

3

Why Group? - Reliability

- The domains affected by a given failure event are constrained to the domains that used the groups that failed
  - Better for one person to lose access to 80% of their data than for a hundred to lose 5%
  - A project often idles while restoring from backup even if it only lost a few scattered files
- Fewer spin-ups decrease probability of drive failure

Still Not Convinced?

- Priority: Working sets could be placed by desired performance, failure rate, etc
- Most files are still small
  - Median file size = 2475 bytes on a UNIX server
  - Moving small files around into working sets could yield huge results
  - Small files are frequently overwritten, but are they overwritten by the same programs?

What about Caching?

- Our system is designed to work on top of caching
  - Once a working set is identified, the system can choose to pull the entire set into cache once the set is accessed
- This makes working sets a meta-cache
- More work to be done in this intersection

What about Clustering?

- Tried unsupervised learning methods
  - k-means
    - Number of clusters is getting easier to predict
  - Expectation maximization
  - Agglomerative clustering
- Useless for our data
  - Scattered, omnipresent writes to hot area distorted data
  - Number of clusters changes

Grouping is hard (but great!)

- For semantically labeled data, we can an average power savings of 20% by grouping data
- Data is expensive to label
- Rich meta-data costs performance to collect
- Privacy concerns
- What if we just collect block level data?

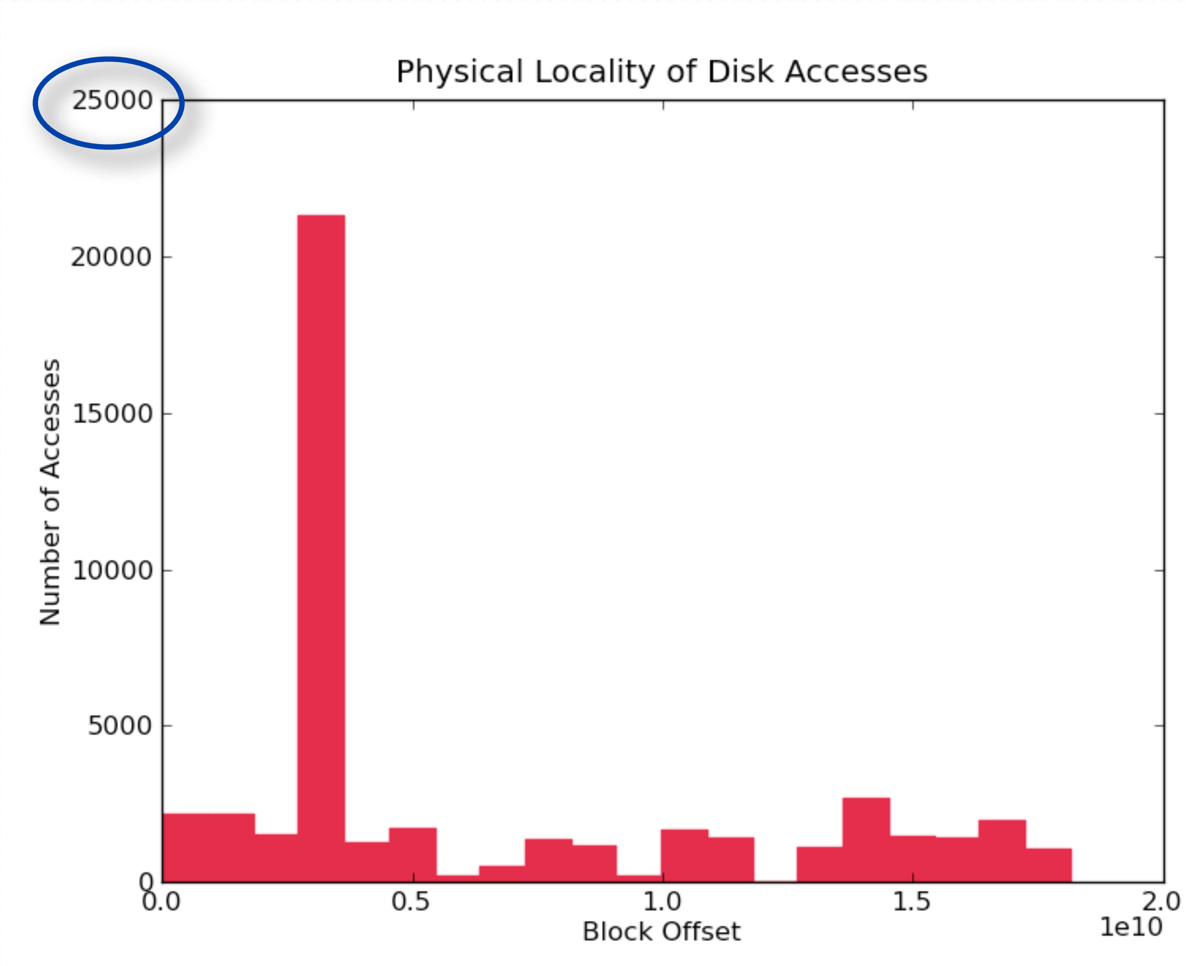
Data at the Block Level

- Block level data:
  - block offset - location of block on physical disk
  - timestamp - time of access
- Easy to collect
  - Low performance overhead
- No domain knowledge needed

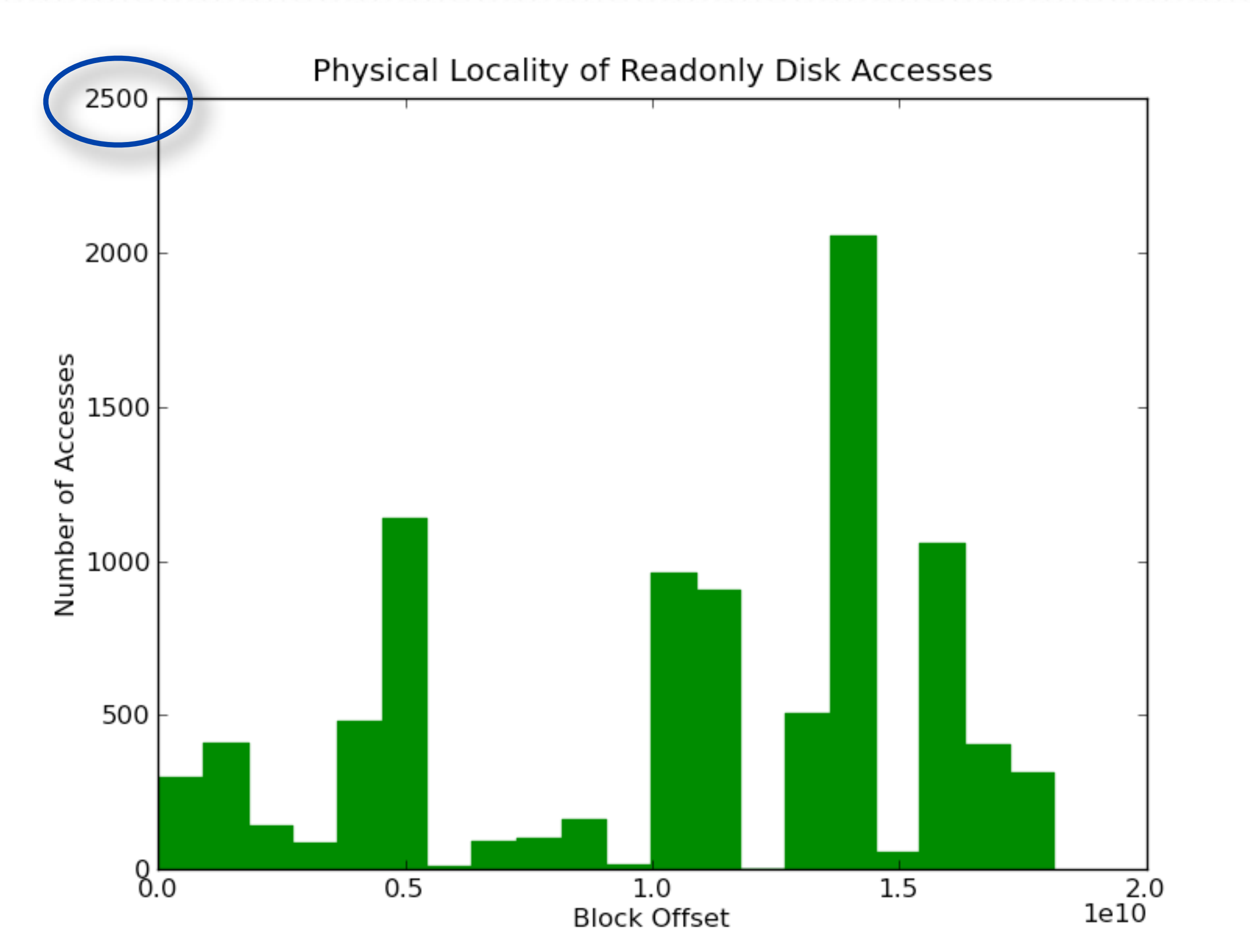
MSR Cambridge Data: Characteristics

- Total number of accesses: 1433655
- Total number of “unique” block offsets: 46718
  - Unique offset-size pairs: 108793
- NTFS

MSR Cambridge Data: Access Locality



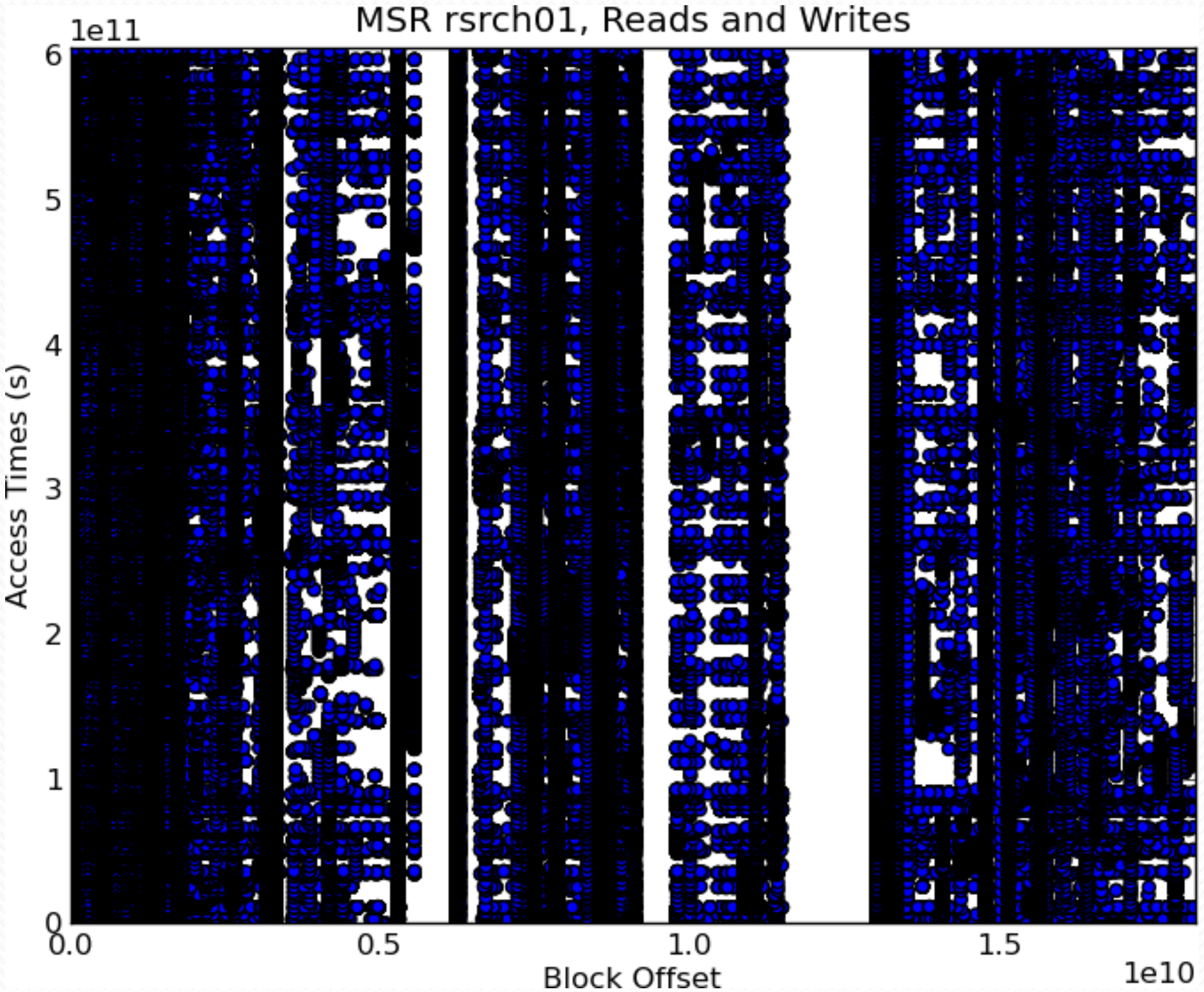
Reads and Writes



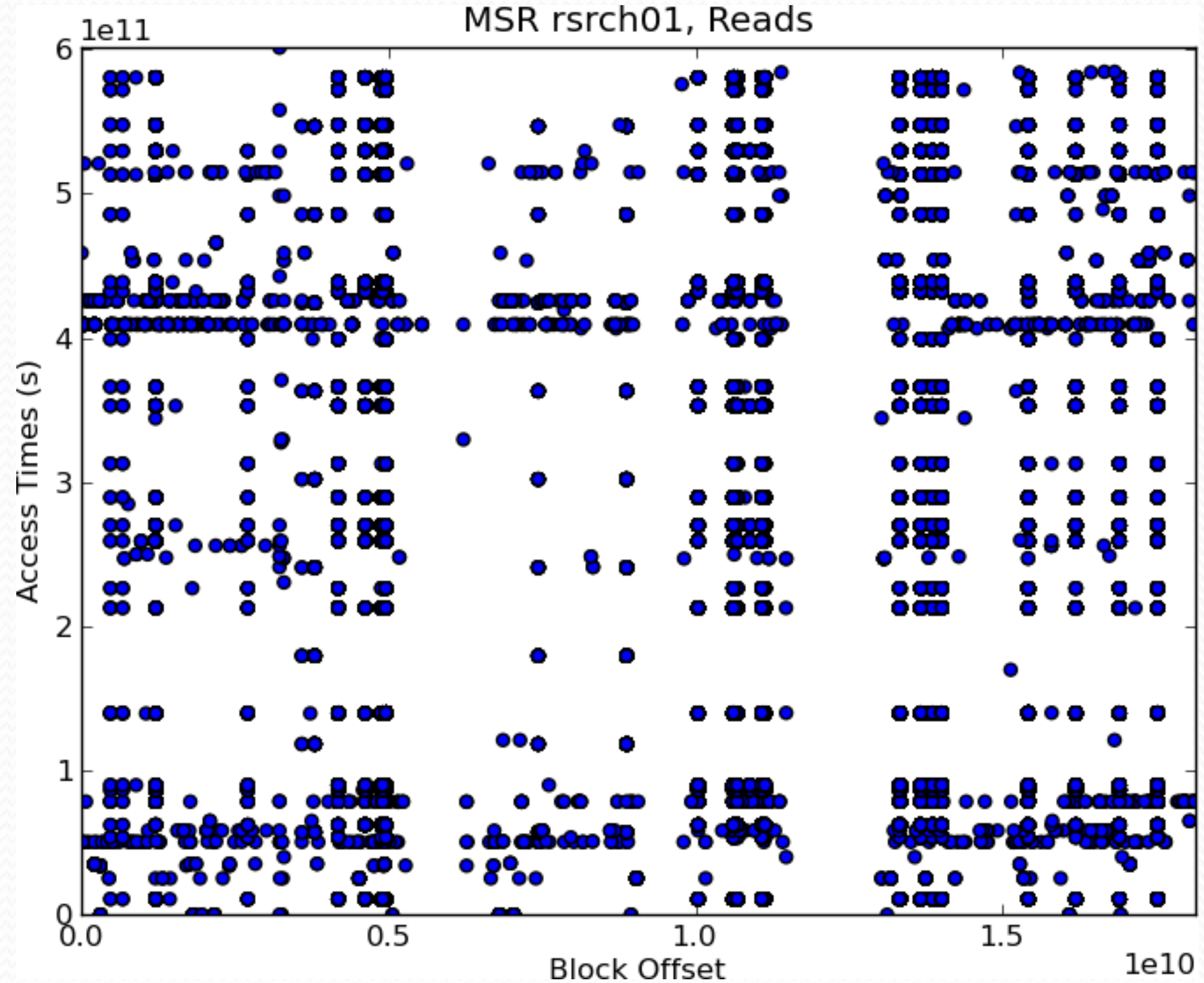
Reads

- Workload is skewed by writes to a hot area

MSR Cambridge Data: Accesses



Reads and Writes



Reads

- Read/Write ratio of **10/90**
  - Many writes are to the same blocks

MSR Cambridge Data: Misc

- Available from SNIA
- Collected in 2007
- Comes from a multi-application research machine (rsrch_0)
- Covers 7 days of machine operation
- Cases of consecutive writes to the same blocks

```
128166372454818843,rsrch,0,Write,3154137088,4096,1175
128166372454818856,rsrch,0,Write,3154137088,4096,1161
128166372472318823,rsrch,0,Write,3154137088,4096,1130
128166372507631099,rsrch,0,Write,3154137088,4096,1227
128166372524817728,rsrch,0,Write,3154137088,4096,2034
128166372524818590,rsrch,0,Write,3154137088,4096,1172
```

Grouping at the Block Level

- All you have is offset, timestamp pairs
- Offsets are likely to be accessed repeatedly
- Calculate similarity across accesses

Similarity - Distance Matrices

- *m*x*m* matrix to calculate the distance between two offsets given **all** pairwise occurrences
- m = # unique block offsets
- $T_k = |t_{ik} - t_{jk}|$
- o_i = offset i

$$d(o_i, o_j) = \sqrt{\left(\frac{\sum_{i=1}^{|T|} T_i}{|T|}\right)^2 + oscale \times (o_i - o_j)^2}$$

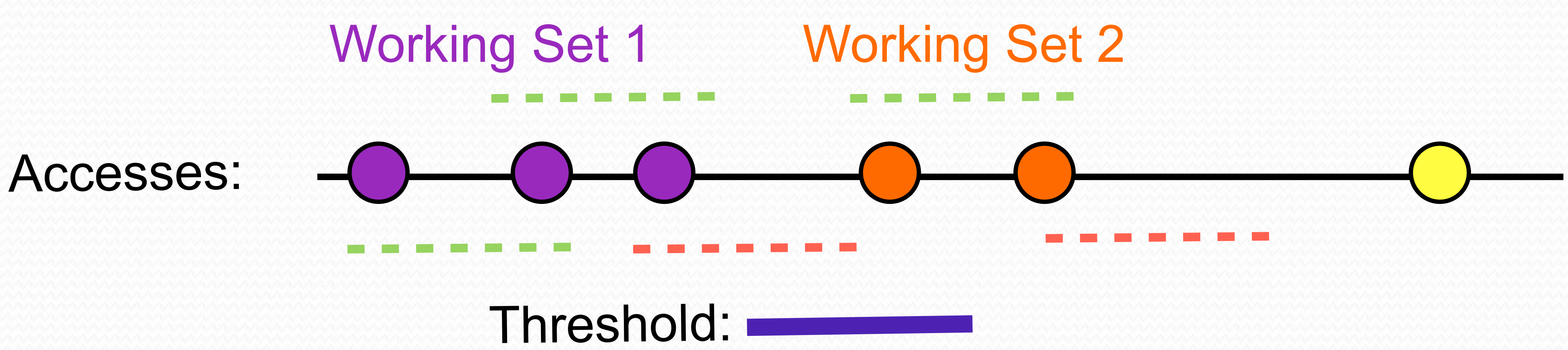
Similarity - Distance Lists

- Pick a threshold around an offset outside which similarities are not considered
- Combine lists of offset distances to get cumulative distance lists

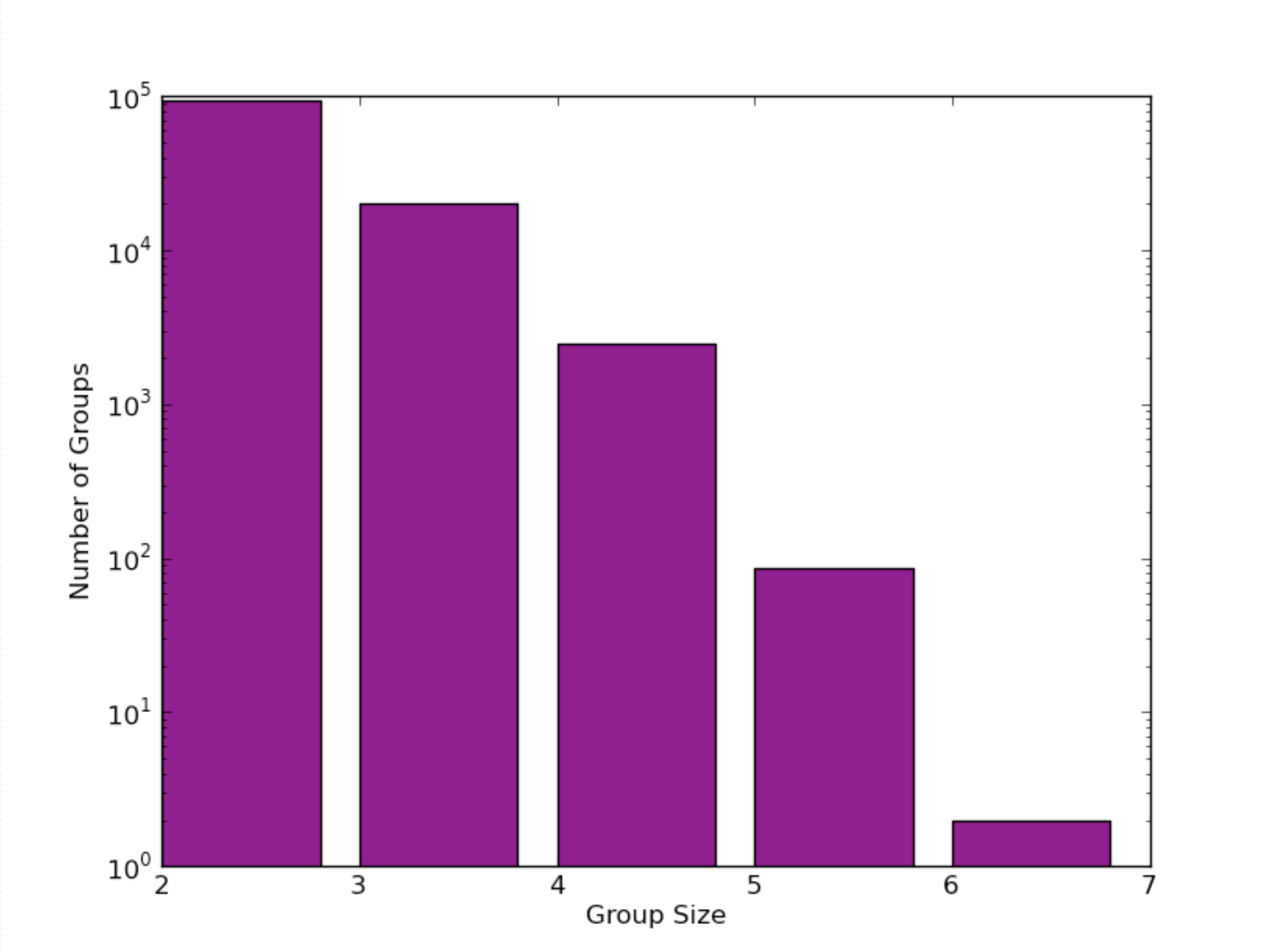
$$[(o, o_i, d(o, o_i)_1 + \frac{d(o, o_i)_2}{|t_1 - t_2|}, (o, o_j, d(o, o_j)_1, (o, o_m, d(o, o_m)_2)]$$

Comparison Technique: Neighborhood Partitioning

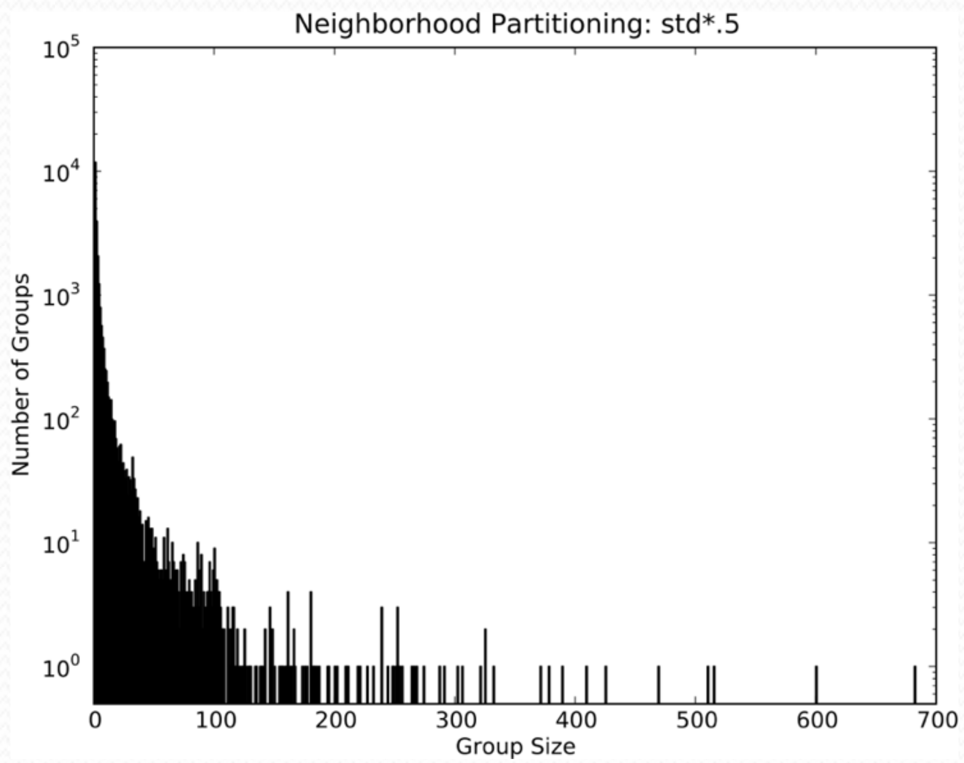
- Calculate global threshold based on mean and standard deviation between accesses
- Apply threshold to determine if adjacent accesses should be in the same working set

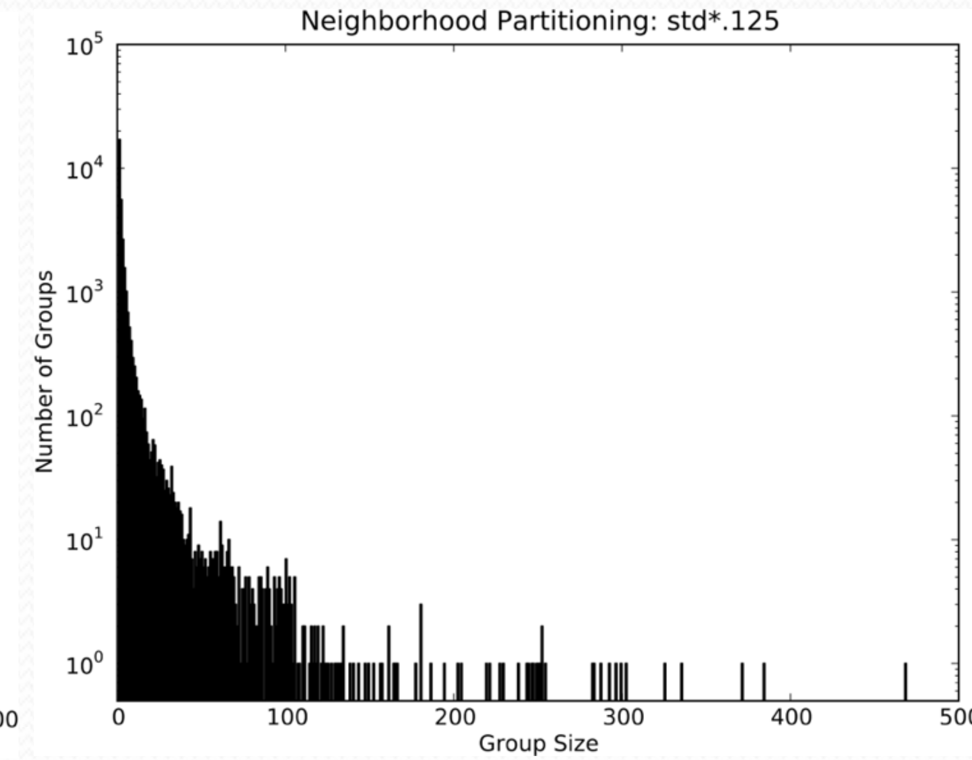


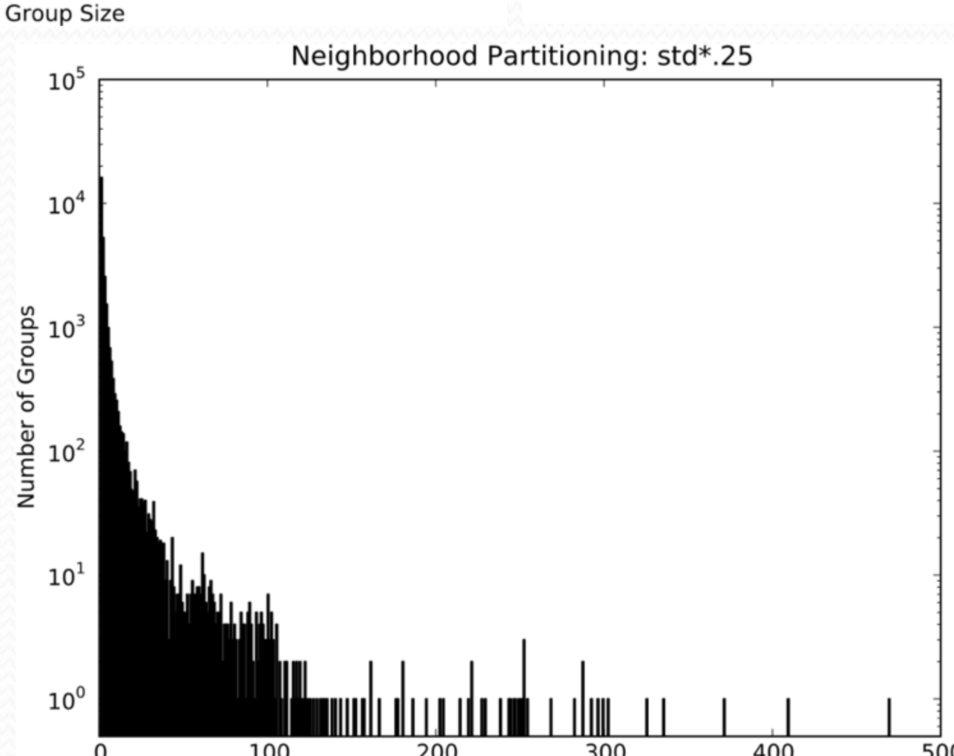
Results: Neighborhood Partitioning, Read-Write



Results: Neighborhood Partitioning, Read-Only

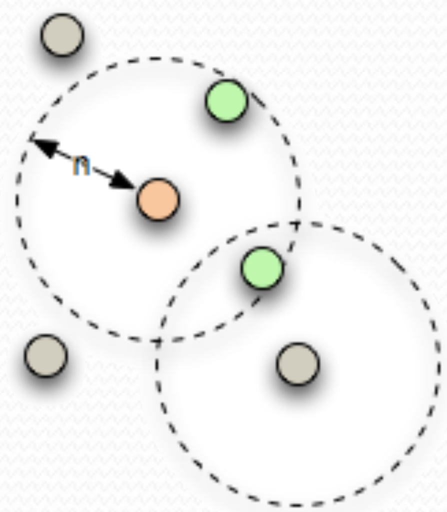






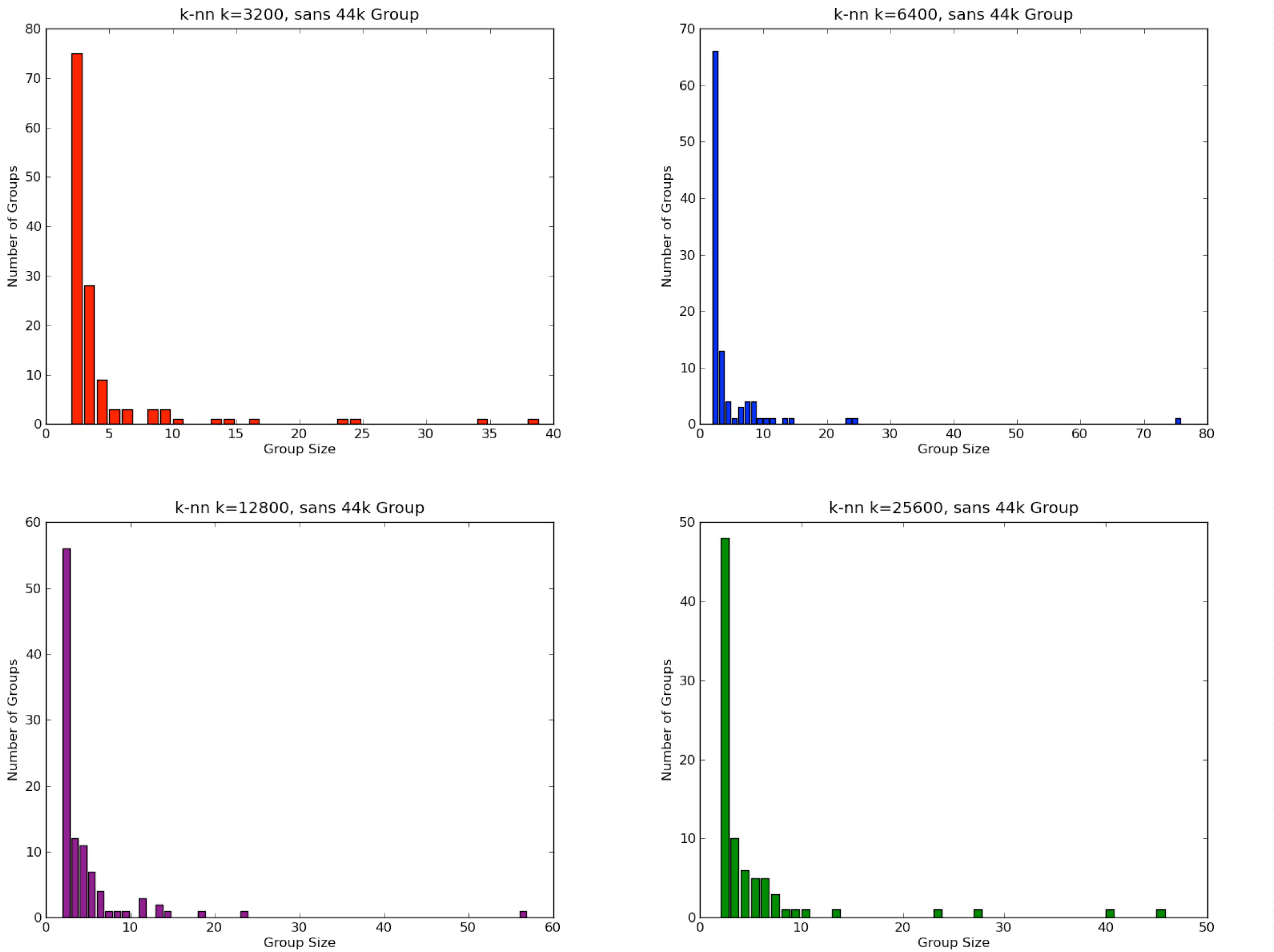
Comparison Technique: Nearest Neighbor (weighted)

- Calculate pairwise distances in spatial neighborhood *n*
- Calculate the average distance per pair
- Use average, scaled time/space distance to group files under different time/space thresholds.

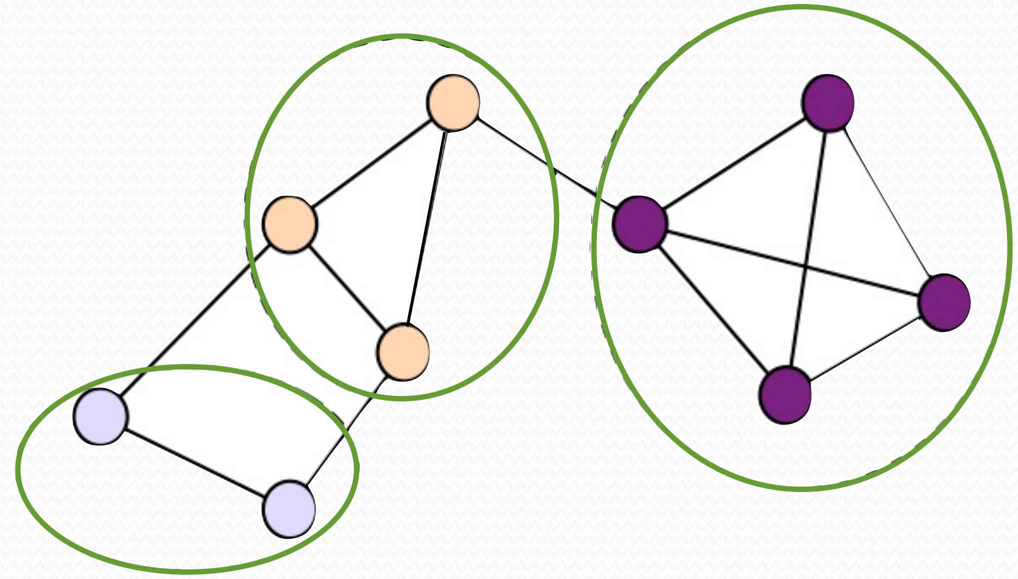


n = 3200 block offset

Results: Nearest Neighbor

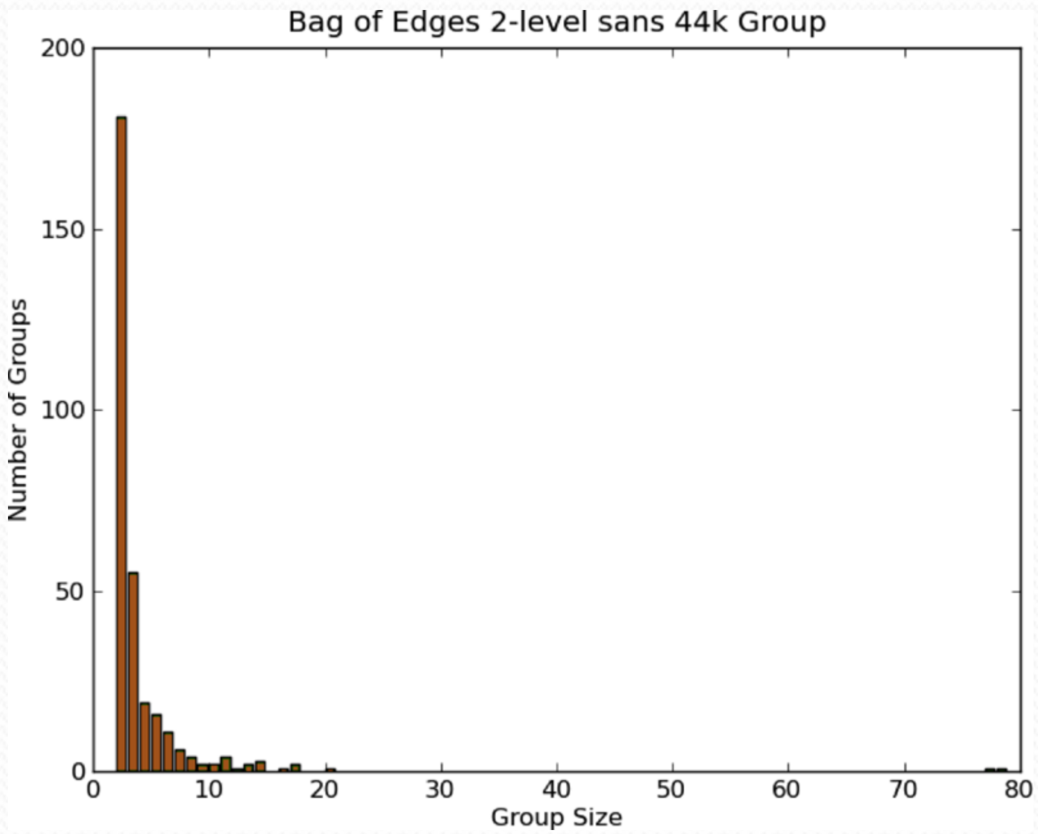


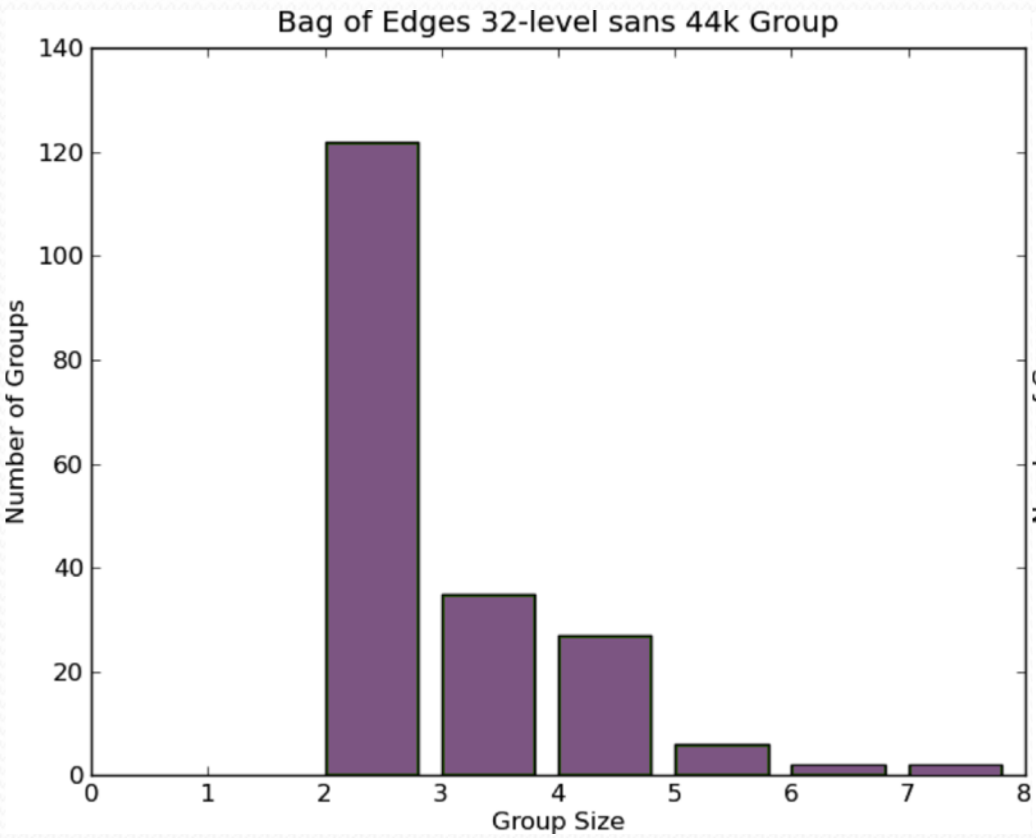
Comparison Technique: Bag-of-Edges

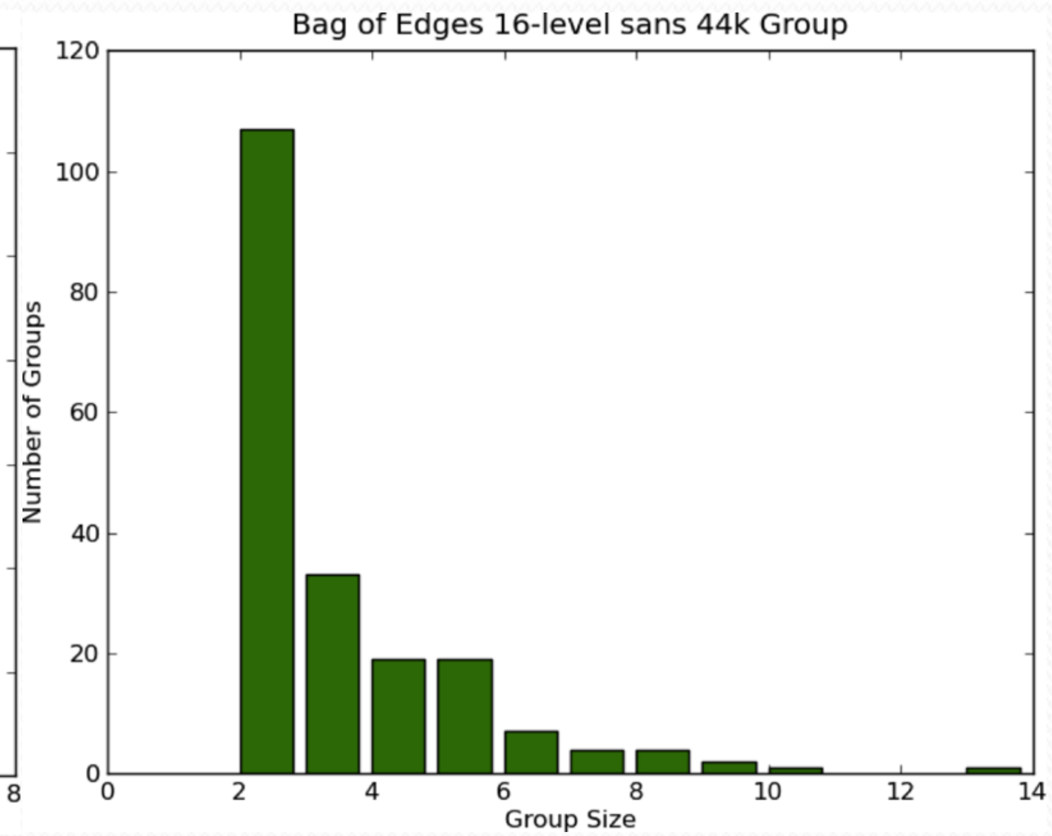


- Nodes = disk accesses
- Edge = two points have an acceptable distance between them (weight >= 0)
- Clique-cover seems right, but fails (and is NP-Complete)
- Less Restrictive: Longest path per connected component (Also NP-Complete)
  - Solution: Toss out weights; run shortest path over negated minimum spanning tree

Results: Bag-of-Edges







pdsi

Sandia National Laboratories

Baskin Engineering UC SANTA CRUZ

Validity

- Do our groupings stay consistent over time?
  - Groups are resistant to most distance scaling factors
  - Large jumps for levels or neighborhood distances
  - Could be natural, correct, usage shift

Validity Methods

- Group overlap
  - Lots of methods to weight here
- Rand criterion
  - $R(G_1, G_2) = \frac{(a+d)}{(a+b+c+d)}$
    - a = # pairs in G_1 and G_2
    - b = # pairs in G_1 not in G_2
    - c = # pairs in G_2 not in G_1
    - d = # pairs not in G_1 and not in G_2
  - $0 \leq R \leq 1$
- Mutual entropy
  - Define probability with set intersection

Next Steps

- Protocol analyzer to collect more block I/O data.
  - Mixed-use educational storage systems
  - HPC Systems
- Implement working set detection real-time
  - Track power savings
  - Track reliability savings
  - Track bad working sets

Future Work

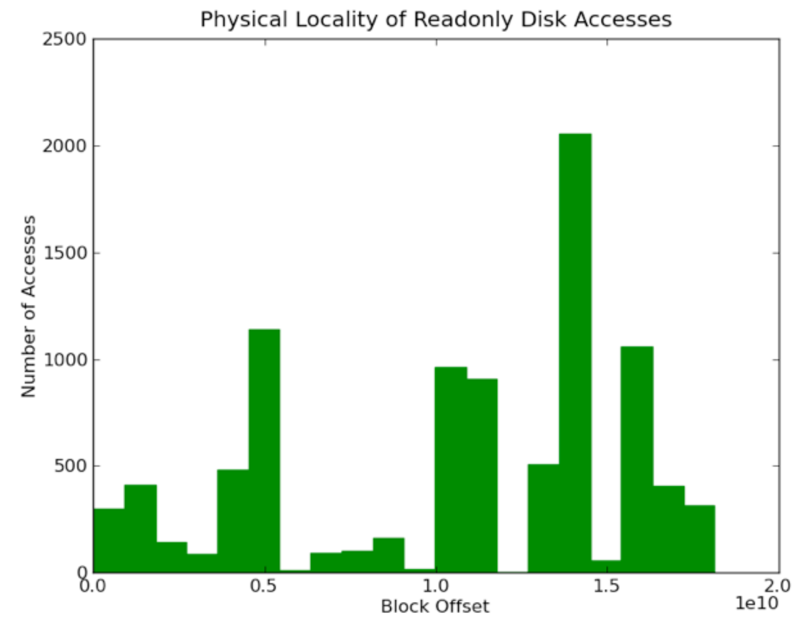
- Application Isolation
  - Working sets -> Application accesses
  - Goal: take a system and compartmentalize files that tend to be accessed by specific applications
    - Duplicating files in storage is OK!
- Workload Characterization
  - Can anything about groupings be transferred to different workloads with similar characteristics?
    - What are these characteristics?
  - Classify based on separability
  - HPC vs. Enterprise vs. User-facing

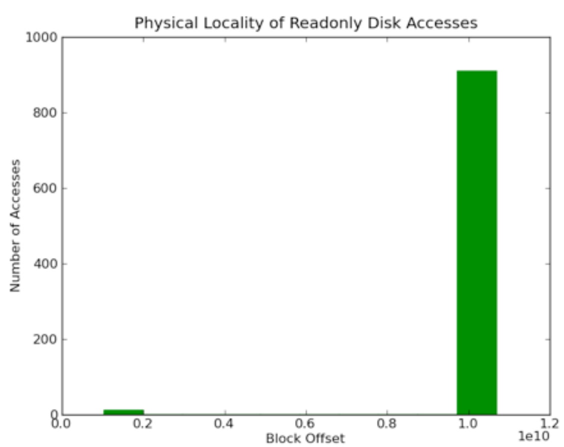
Please send all of your data to:

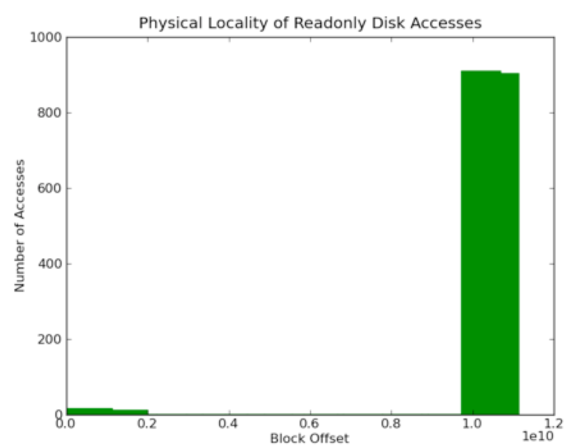
- Avani: avani@soe.ucsc.edu
- Ethan: elm@soe.ucsc.edu
- Lee: lee@sandia.gov

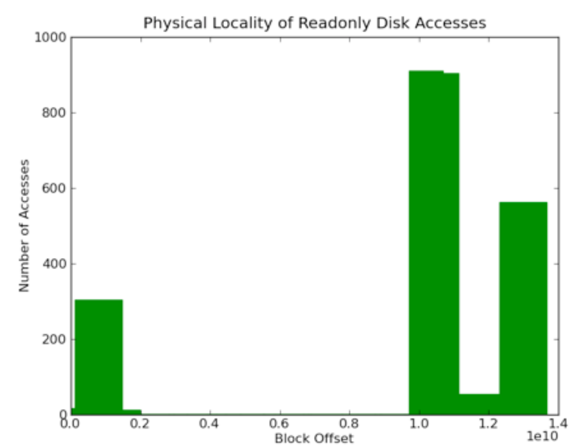
Thanks!

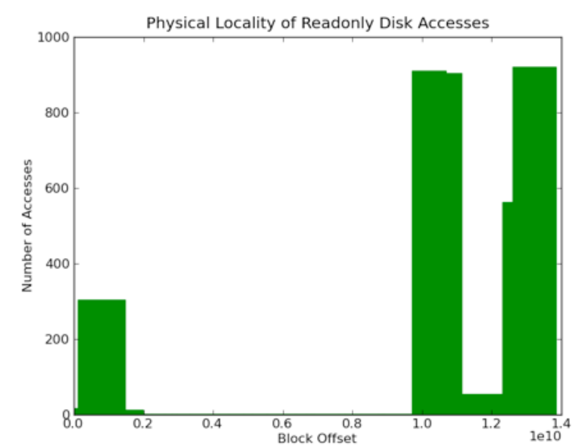
BACKUP: Read Accesses

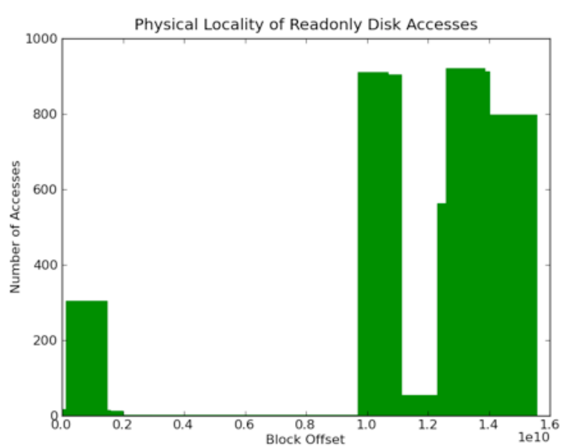


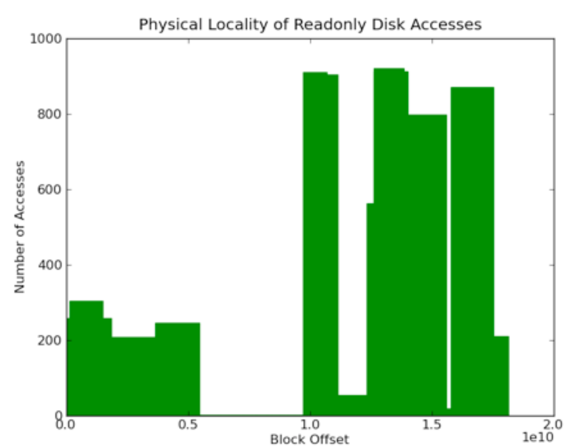


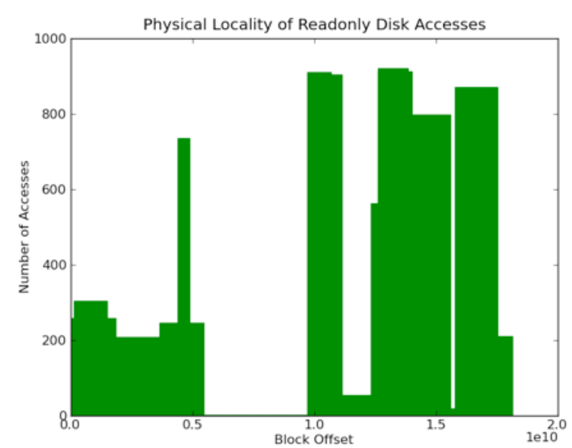


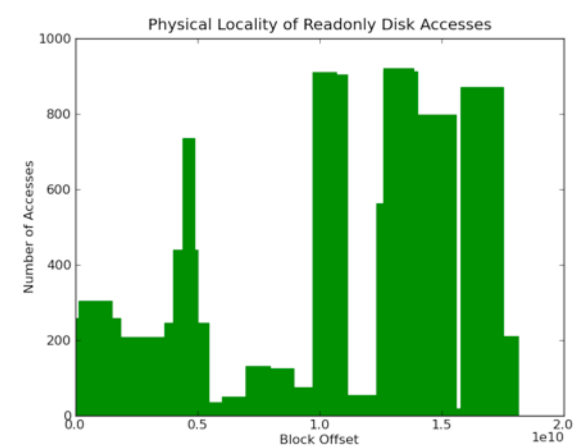












BACKUP: Comparison: Sliding Window

- Sliding window of nxn pairwise comparisons