

A Case for Self-Optimizing File Systems

Jason Liptak, Sam Burnett

Motivation

- Current file systems do not take data access patterns into account.
 - Most requests made are non-sequential.
- Improved disk access performance can result in improved application performance.

Application	CPU (s)	I/O wait (s)	Seq I/O (%)
firefox	1.95	5.43	13.97%
gedit	0.70	4.53	18.83%
gimp	2.67	4.37	50.49%
oowriter	4.83	10.82	10.47%
xemacs	0.74	3.59	27.03%
xinit	0.64	2.95	50.38%

Approach

- Build a self-optimizing file system that organizes data on disk according to data usage patterns.
- *Profile* data access patterns.
- *Analyze* the data by to create an abstraction.
- *Plan* new data layout.
- *Reconfigure* the data placement for optimal access.

Profiling

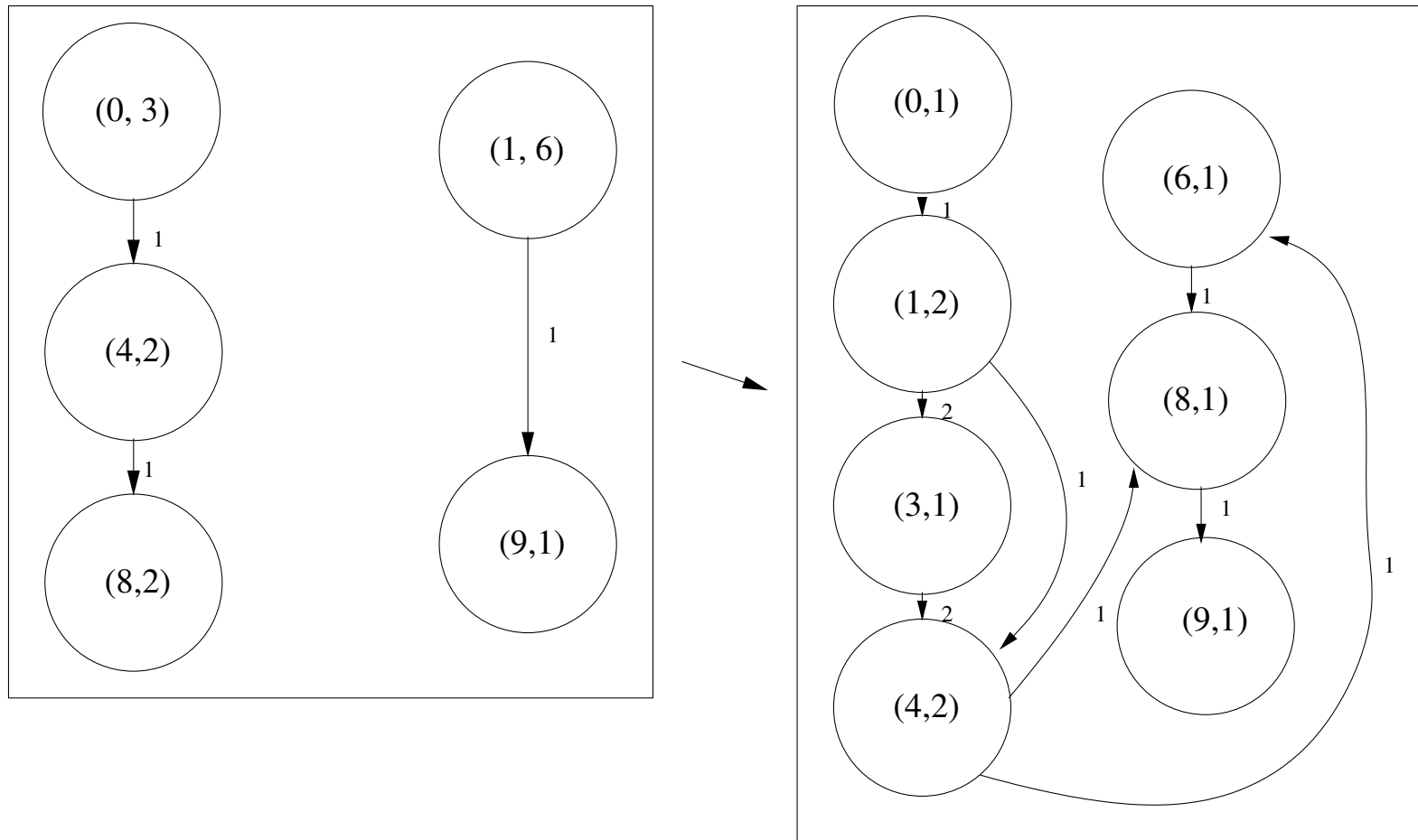
- Every IO that passes through the kernel is recorded.
- The kernel talks in terms of Logical Block Numbers (LBNs).
- Records are of the form `time pid program start length mode`
 - `time` is the current CPU clock value.
 - `pid` is the process ID of the process that submitted the request.
 - `program` is the executable name of the submitter.
 - `start` is the first LBN of the request.
 - `length` is the length of the request, in blocks.
 - `mode` is read or write.

Analysis

- We model disk access as a directed graph.
 - Vertices are LBNs.
 - Edges are the transitions between LBNs in a sequence.
- One graph is constructed for each PID.
- These graphs are merged together to form a comprehensive graph used in planning.

Example

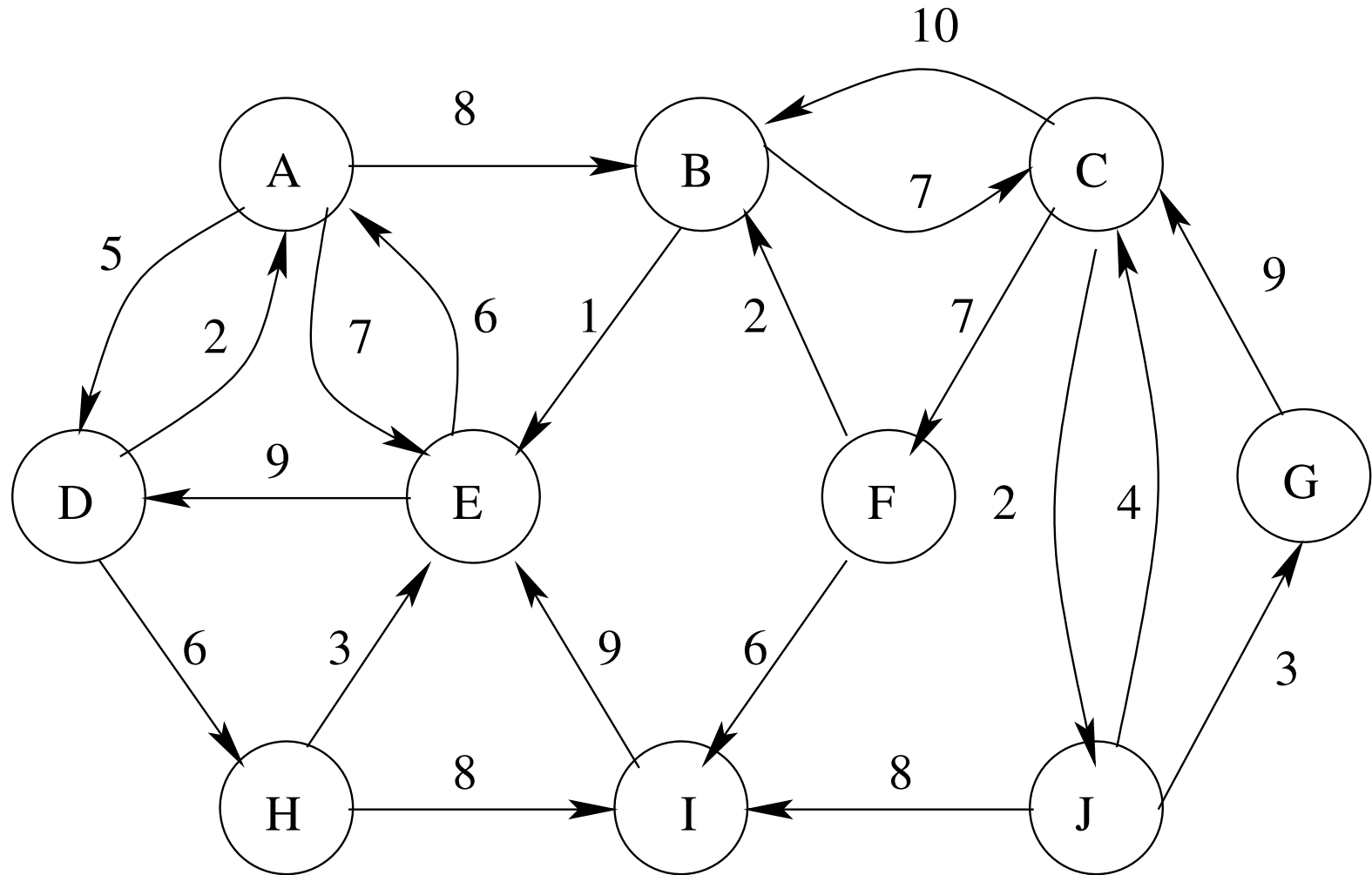
Here is an example graph, and how it would be merged.



Planning

- Use a greedy algorithm to place data:
 1. Find most connected vertex. Place this first. Call this vertex the “blob.”
 2. Find edge with highest weight connected to the blob.
 3. If the edge is coming into the blob, place the vertex at the other end of the edge before all other vertices. If the edge is coming out of the blob, place the next vertex after the blob.
 4. Move all edges of the next vertex to the blob.
 5. If there are edges connected to the blob, return to step 2.
 6. If there are any vertices left in the graph, return to step 1.
 7. Otherwise, we are done.
- This algorithm can be thought of as a function from LBNs to LBNs; An LBN is given to the algorithm and it returns a new location for that LBN based on the optimization heuristic.

Example



Placement Algorithms

- We use two placement techniques:
 - The first places all LBNs sequentially. This allows for optimal performance, but is unrealistic.
 - The second places LBNs in the original LBN space. This is more realistic, though in a real file system better performance will probably result.
- Together, these two algorithms represent bounds on the performance of the system.
- Real file system performance will likely be somewhere in between.

Reconfiguration

- At some convenient time, rearrange the data according to the mapping provided in the planning stage.
- We simulate this by reading from old LBN locations, then reading from new LBN locations. This is the sequence of operations that would be performed to move data in a real self-optimizing file system.

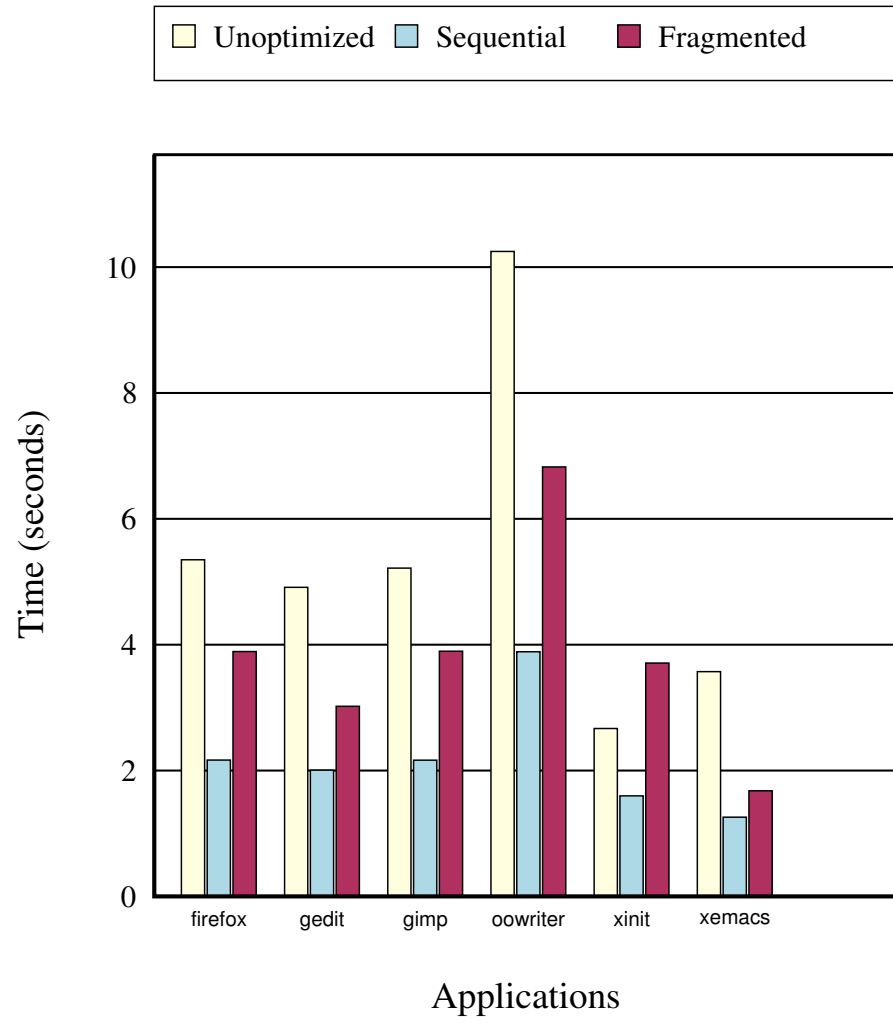
Evaluation Techniques

- Evaluate performance of
 - individual applications
 - the entire system
- Evaluate overhead of
 - profiling
 - data processing

Application Performance

- Collected disk access data for six Linux applications.
- Used this information to create a new disk layout.
- Simulated application disk access by performing the requests made by the application back-to-back.
- Compared this to access time if using the optimized layout.

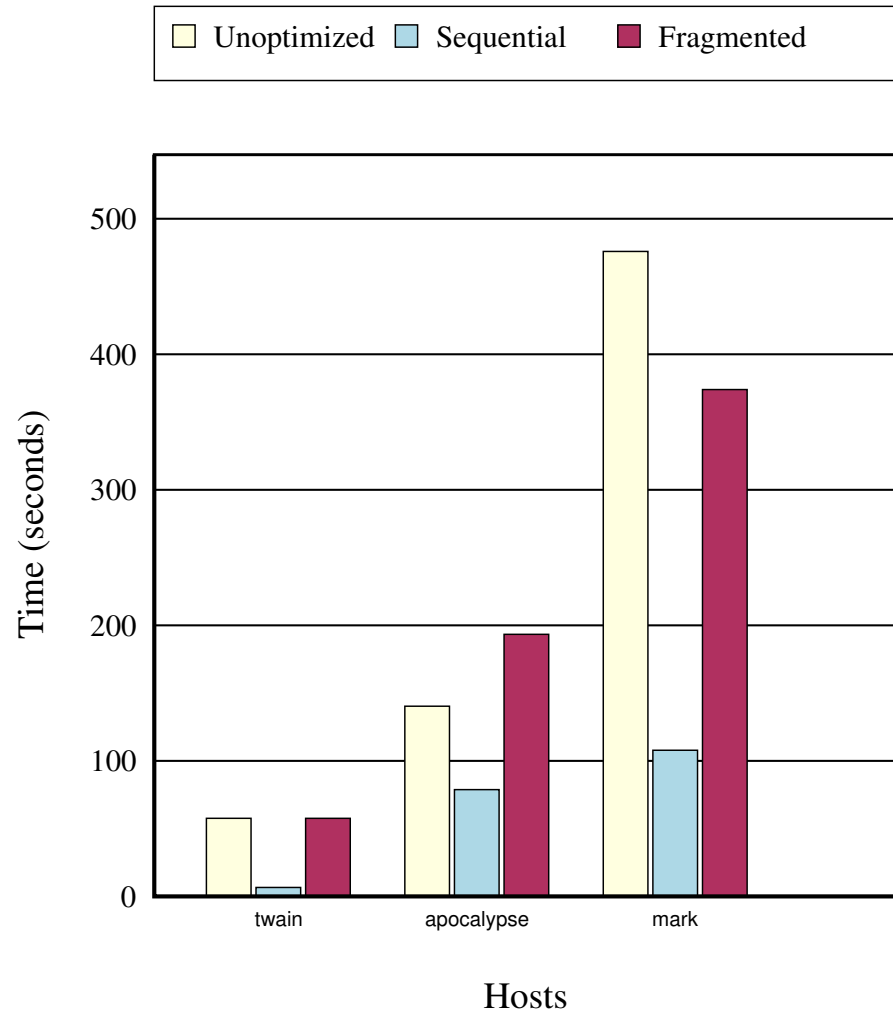
Application Performance



System Performance

- Collected disk access data for three hosts over several days.
- For each host, created a new disk layout.
- Compared access times using back-to-back access.

System Performance



Profiling Overhead

- Appears to be very small.
- At this time, all request data is sent to a remote server for storage.
 - Perhaps more overhead if everything is done locally?
- May become much higher if we probe before caching occurs.

Processing Overhead

- Processing (analysis, planning and reconfiguration) can be done “offline,” at whatever time is most convenient to the user.
- Current algorithms are slow. A real implementation would be optimized.

Host	Requests	Profiling(s)	Processing(s)	Overhead(%)	sec./req.
twain	43472	51336	789	1.54%	0.0181
apocalypse	68752	262615	1492	0.57%	0.0217
mark	181120	156531	12480	7.97%	0.0689

Overhead for Analysis, Planning and Reconfiguration

Implementation Issues

- Obtain all relevant information from the kernel.
Currently, information is obtained after the buffer cache.
We would like information from before the cache.
- Overhead
 - Where to store all the records (cannot dump to disk).
 - When to perform calculations (online or offline).
 - How to efficiently move data for optimal placement.

Future Work

- More rigorous testing of the layout algorithm.
- Find proper profiling point in kernel.
- Perhaps explore other algorithms?
- Implement an actual self-optimizing file system.