Algorithms for Efficient Data Migration

Jared Saia
University of Washington

Joint work with
Joe Hall, Jason Hartline, Anna R. Karlin
University of Washington
John Wilkes
HP Labs

Motivation

- Performance of large storage system depends on having data assignment which balances request load
- Optimal layout of data changes with changing workloads
- Want to periodically recompute a new optimal assignment of data to devices [Borowsky et al, 1997][Gavish et al, 1990]
- Must move data to new optimal assignment **as quickly as possible**

Migration Problem

Given:

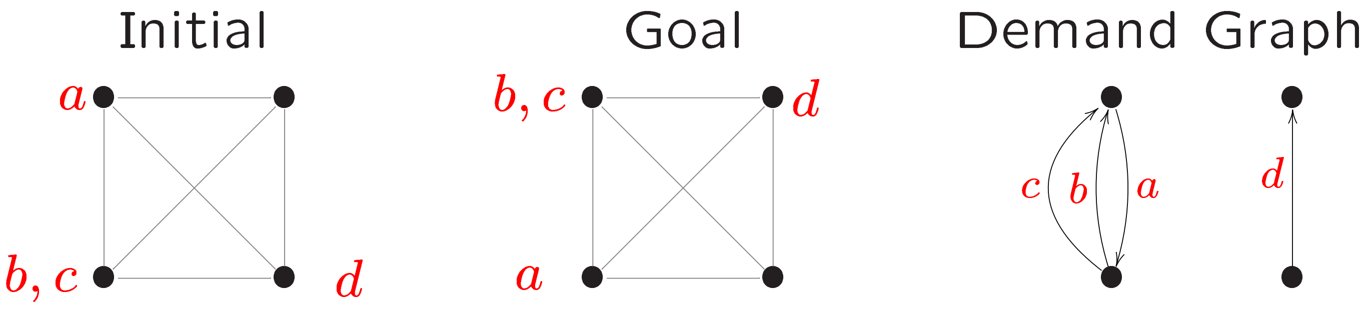
- Initial and final configuration of data objects on devices
- Description of the storage system.

Our Goal: To compute a *migration plan* obeying network constraints and using minimal number of time steps.

Key constraint:
In one time step each storage device can be involved in the transfer of one data object.

Demand Graph

Assume network is a complete graph



Output: Assignment of time steps to each edge in demand graph such that no two edges incident to same node are assigned the same time step

Multigraph Edge Coloring

- Minimum number of colors for edge coloring is χ'
- Maximum degree of graph (Δ) is trivial lower bound on χ'
- Computing χ' is NP-Complete [Hoyer, 1981]
- Best known approximation algorithm for multigraphs is $9/8\chi' + 3/4$ [Hochbaum et al, 1996]

Beyond Edge Coloring

- Indirect migration
  - Allowed to send an object through intermediate or *bypass nodes*
  - This can decrease the number of time steps required
- Migration with memory constraints
  - In any intermediate stage of the plan, all machines must have a number of objects stored on them less than or equal to the memory of that machine.

Indirect Migration

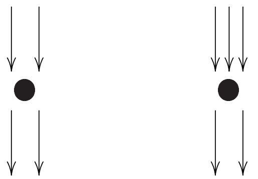
Memory Constraints

Let $d_i(v)$, $d_o(v)$ be the in,out degree of node v respectively

- Need $d_o(v)$ memory at start at node v
- Need $d_i(v)$ memory at the end at node v
- $\Longrightarrow$ need $\max(d_o(v), d_i(v))$ memory total

Let $M(v) = \max(d_o(v), d_i(v)) + 1$

Examples:



What can we hope for?

Given: arbitrary directed multigraph G with max degree Δ and n nodes.

- Clearly need $\geq \Delta$ steps for any migration
- In worst case, need $n/3$ bypass nodes to achieve Δ

$n/3$

Direct migrations

- need $\geq \chi'$ steps
- in worst case, $\chi' = \frac{3}{2}\Delta$

Our Results

Given: arbitrary directed multigraph G with max degree Δ and n nodes.

No memory constraints:

- $2\lceil \Delta/2 \rceil$ steps using $\leq n/3$ bypass nodes

With memory constraints:

- $4\lceil \Delta/4 \rceil$ steps using $\leq n/3$ bypass nodes
- $6\lceil \Delta/4 \rceil$ steps for direct migration

Essentially match worst case bounds

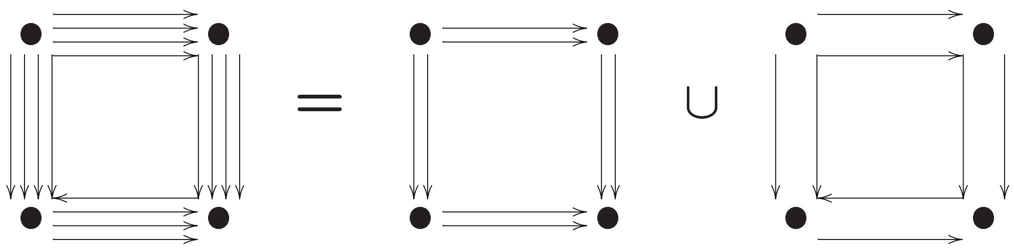
Indirect with Memory Contraints

1. Factor graph into *good* set of 4-regular subgraphs (4-factors)

   4-factor Property: After any 4-factor is satisfied with memory constraints, there is at least one free space at every vertex.

2. Satisfy each 4-factor while obeying memory constraints

Getting good 4-factors



4-factor Property: After any 4-factor is satisfied with memory constraints, there is at least one free space at every vertex.

Idea: Construct factoring such that in-edges on original graph distributed "evenly" across the 4-factors.

Two cases:

- $d_o \geq d_i$ - each 4-factor can only generate free space
- $d_i < d_o$ - never use more than initial amount of free space

Satisfying 4-factors

Hard vertices: $\rightrightarrows\bullet\rightrightarrows$

Goal: label edges with letters A and B such that

- each vertex has 2 edges labelled A and two labelled B
- each *hard* vertex has one out edge labelled A

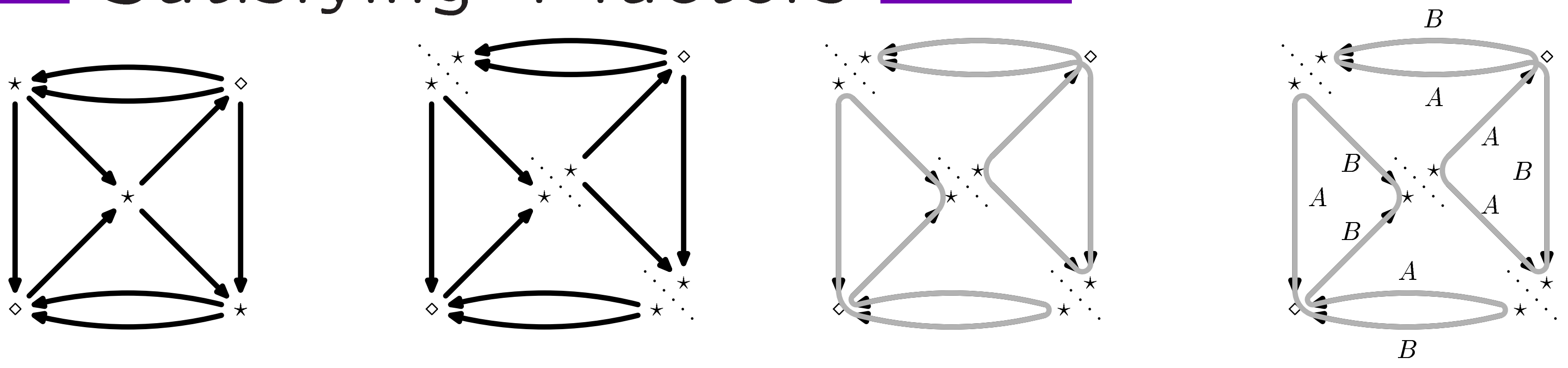
With such A/B labelling, we have two stages:

1. Perform migration on A labelled edges
2. Perform migration on B labelled edges

Two time steps for each stage with $\leq n/3$ bypass nodes.

Problem: There are 4-regular multigraphs in which **no** such A/B labelling exists.

Satisfying 4-factors



Original Graph | Split Hard Vertices | Euler-Tours | *A*/*B* labelling complete

Even length components: A/B label with Euler tour
Problem: odd length components

First idea: introduce one bypass node per odd component
Problem: can be too many odd components ($2n/5$)

Solution: Merge *adjacent* odd components and judiciously choose which edge to bypass in remaining odd components.

Conclusion and Open Problems

Results:

- First results on migration with indirection and memory constraints
- For worst case inputs, results are optimal!

Many open problems:

- Is chromatic index with space constraints $>$ chromatic index without?
- What is tradeoff between number of bypass nodes available and number of steps required?
- What can we say about migration on incomplete topologies?