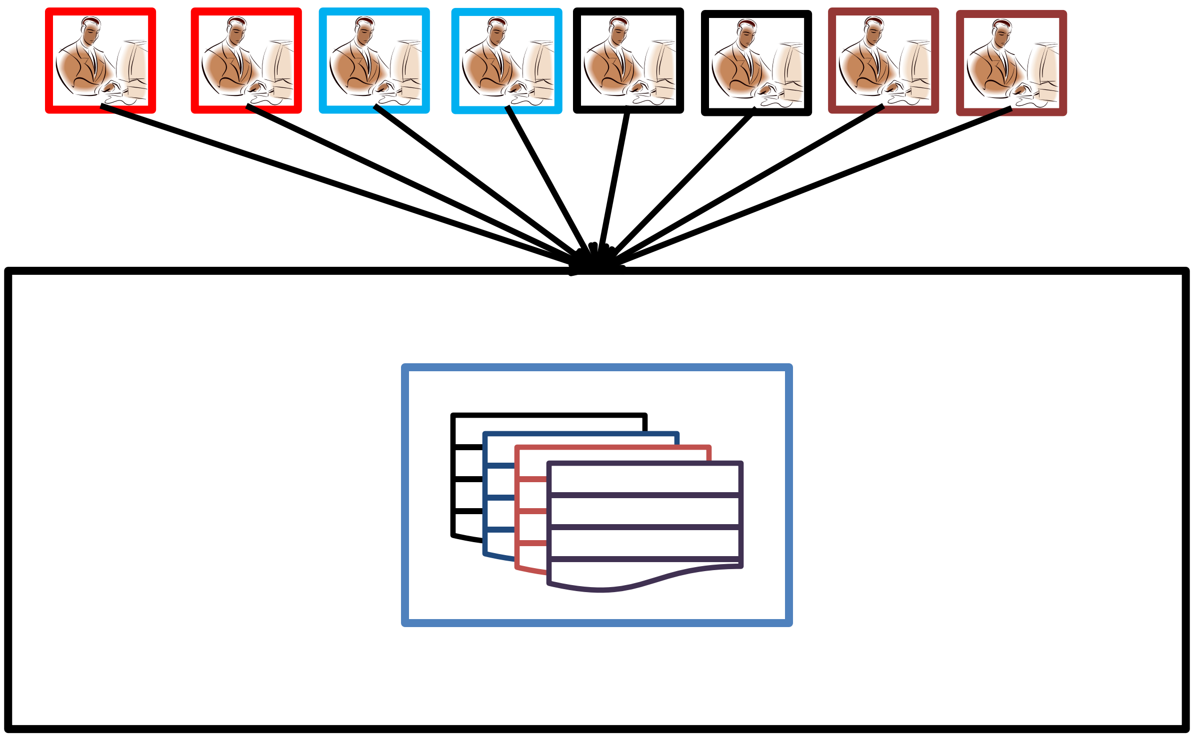
URSA: Scalable Load Balancing and Power Management in Cluster Storage Systems

Seung-won Hwang
POSTECH, Korea
Joint work with Gae-won You (POSTECH),
Navendu Jain (Microsoft Research), Hua-jun Zeng (Microsoft)

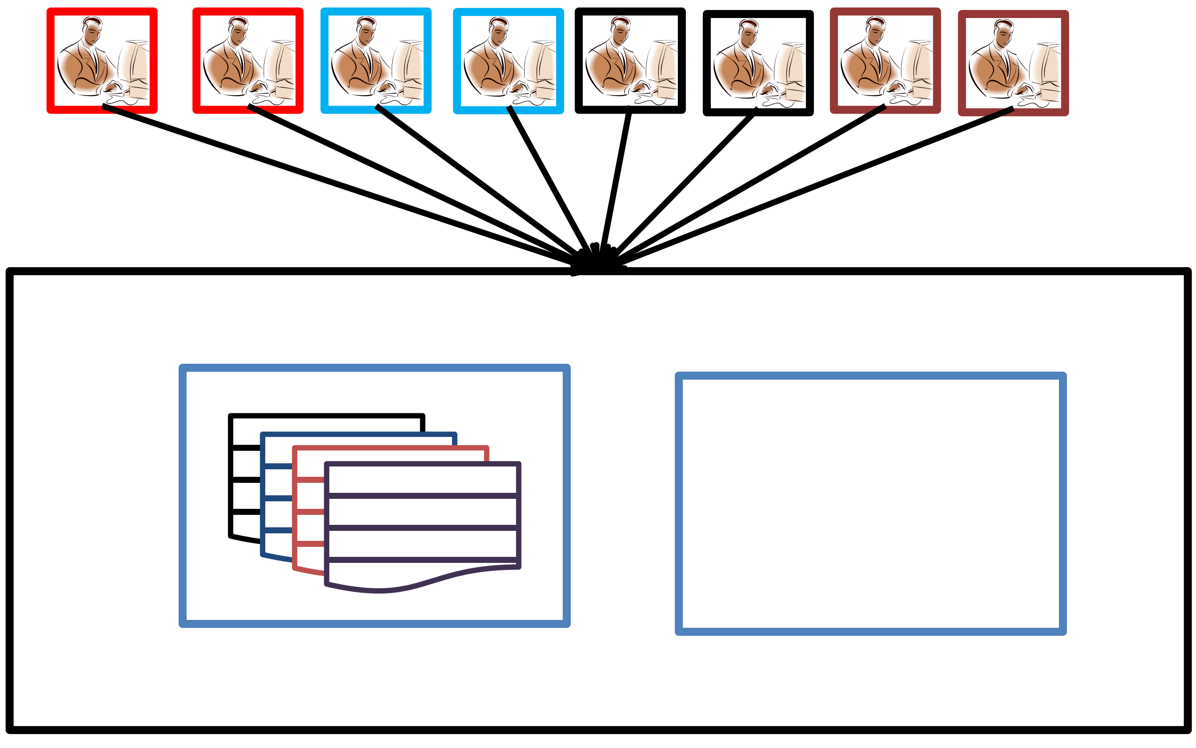
POhang university of Science and TECHnology

Elasticity

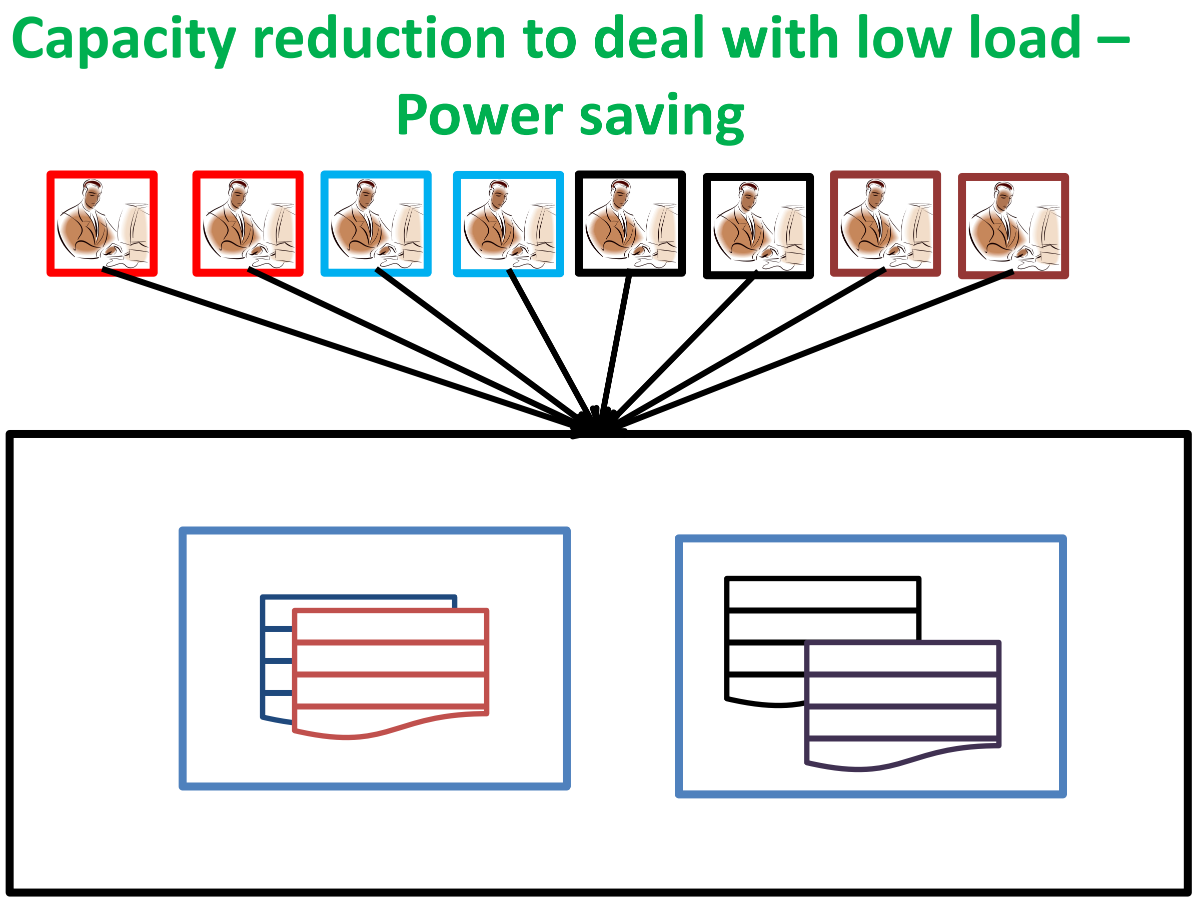


Elasticity goal I – load balancing

Capacity expansion to deal with high load – Guarantee good performance



Elasticity goal II – power management



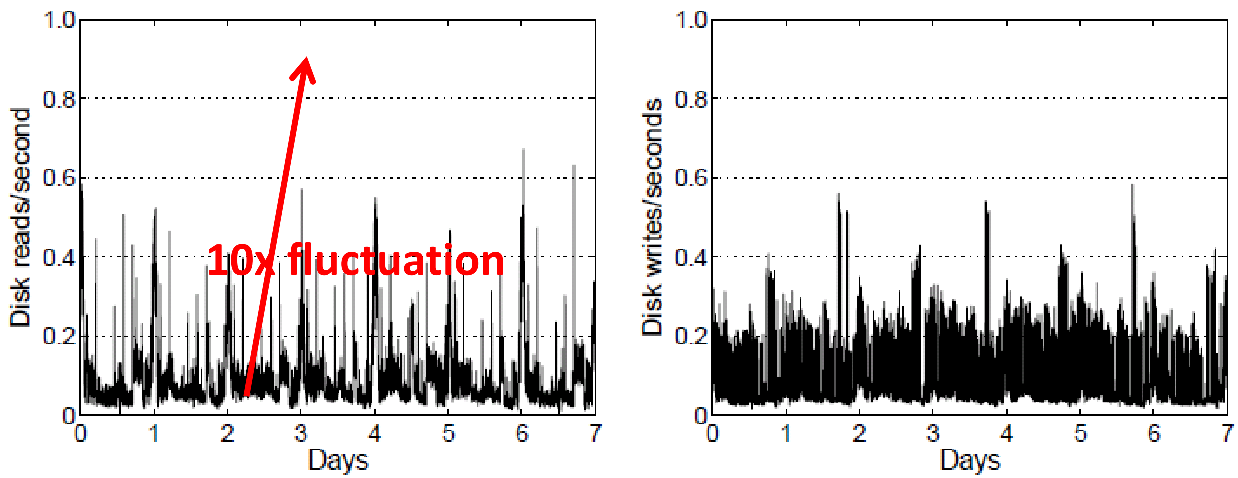
Problem statement

- Pursuing the two elasticity goals
  1. Load balancing: “spread” load across servers
  2. Power management: “consolidate” load to few servers

- Target scenario: industry-scale (e.g., Hotmail) in cloud data center

Three requirements (R1~R3)

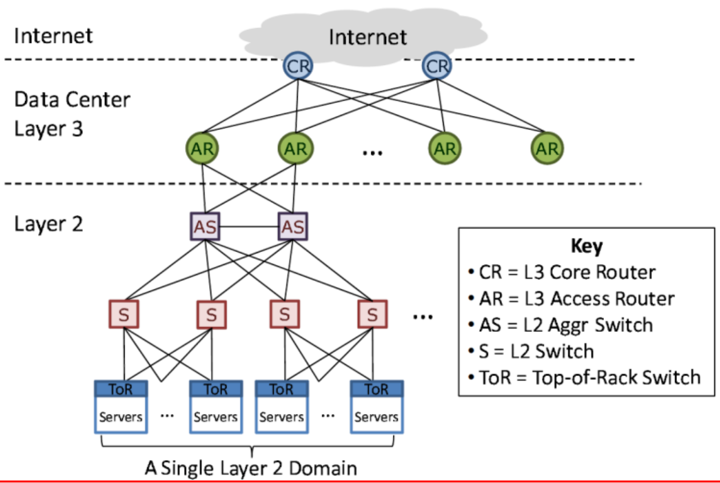
- R1: Scalability: scaling to a large number of nodes and objects, e.g., 10k+ nodes, 10M+ objects
- R2: Dynamic balancing: (normalized) weekly loads



(a) Reads (b) Writes

Three requirements (R1~R3)

- R3: Topology-aware cost model: differentiating reconfiguration within rack, nearby racks, farther racks
  - "conventional network configuration: As traffic moves up through the layers of switches and routers, the over-subscription ratio increases rapidly" **[VL2:SIGCOMM09]**



1:1 over-subscription to other servers in the same rack
1:5 to 1:20 oversubscription to up-links (cross rack)
1:240 oversubscription to the highest layer of tree

Existing work

- Static approach
  - Do not address dynamic load reconfiguration

- Dynamic approach (live migration)
  - Greedy approach
    - Cost model not reflecting topology
  - Integer linear programming (ILP)
    - Cost optimization, not really scalable

Static approach

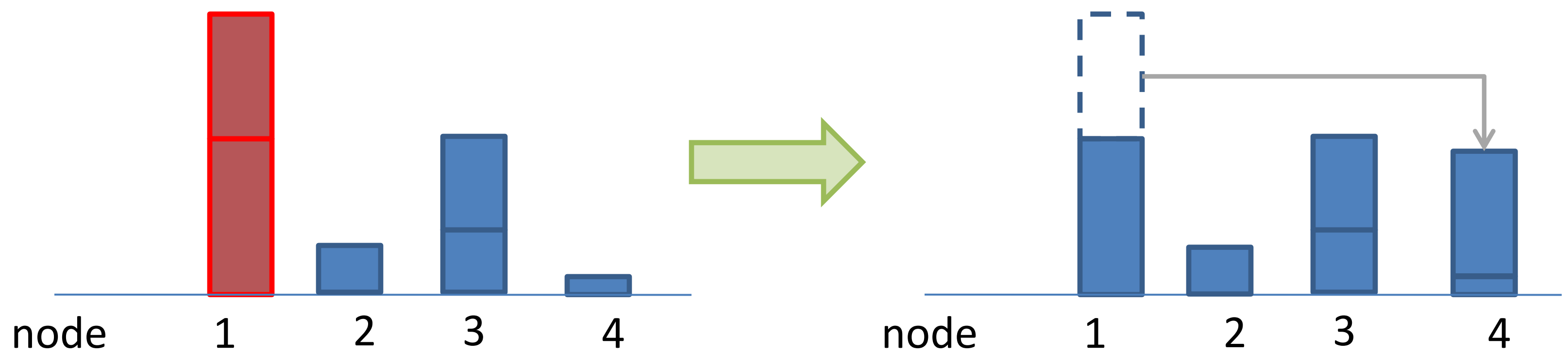
- Evenly distribute loads over all nodes offline
- A variant of the bin-packing or knapsack problem
  - NP hard problem
  - R1: Computationally intensive offline solution
  - R2: Hard to adopt for dynamic balancing
  - R3: topology not considered

Existing work

- Static approach
  - Do not address dynamic load reconfiguration
- Dynamic approach (live migration)
  - Greedy approach
    - Cost model not reflecting topology
  - Integer linear programming (ILP)
    - Cost optimization, not really scalable

Greedy approach (1/2)

- Move objects the hottest node to the coldest (HtoC)
  - Minimize load imbalance
  - Iteratively find pairs of nodes with highest/lowest loads
  - Move objects between them

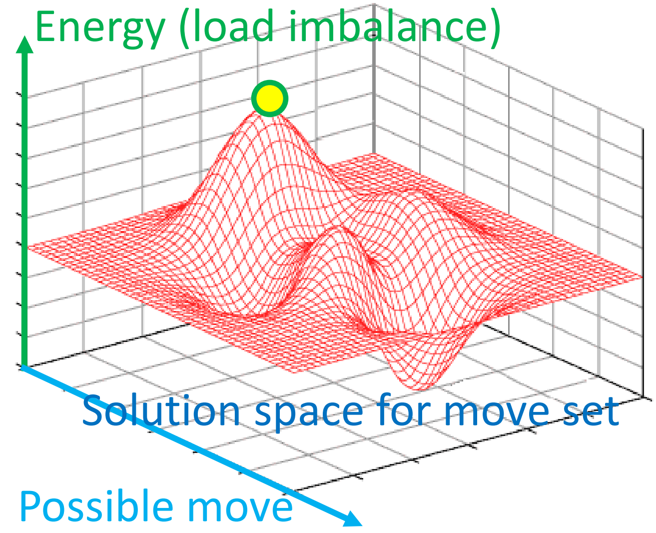


R3: High bandwidth cost and high latency without considering topology, e.g., 1→2 can be better than 1→4

Greedy approach (2/2)

- Simulated Annealing (SA)
  - Minimize load imbalance/variance
  - Search for best set of moves before actually execute it
  - A "temperature" parameter to control the search process **to avoid local optima problem**

R1: Low performance in large system due to too big solution space, e.g., 100 nodes, 1K objects
→ 100k possible moves
R3: topology not considered



Over-simplified illustrative example with possible moves = 2

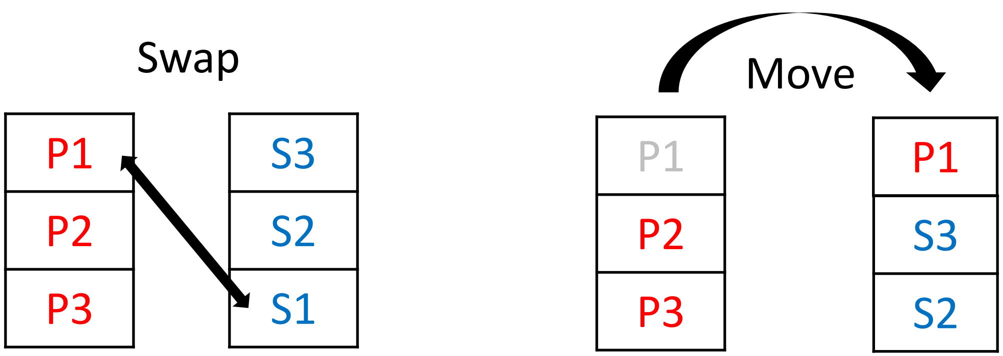
Integer linear programming (ILP)

- Optimization on placement of load components using ILP model
  - Ensure that no node is overloaded
  - Ensure that maximum reconfig cost is not exceeded (given just a constraint)

R1: Not scalable due to too many decision variables and slow Integer LP solver

Proposed framework

- Goal: **Eliminate hot spots with minimal reconfig cost**
- Two available knobs:
  - **Swap**: Switch roles of primary & secondary replicas; cheap
  - **Move**: Move replicas to low-utilization nodes; bandwidth cost of migration -- much more expensive than swap



LB: constraints and assumptions

- Constraints
  - Capacity constraints for each node
    - Load: Total read/write load <= load capacity
    - Storage: Size volume of objects <= storage capacity
  - Fault tolerance constraints
    - Two replicas of the same object should not be in the same failure domain (e.g., node, rack)

ILP model for LB cost-optimization

A: original Y: migration B: cost

minimize $$\sum_i \sum_j \sum_k A_{ijk} Y_{ijk} B_{ijk}$$

subject to

$$\forall i : \sum_j \sum_{k=0}^{|p_j|-1} Y_{ijk} L_{jk} \leq C_i$$

$$\forall i : \sum_j \sum_{k=0}^{|p_j|-1} Y_{ijk} \leq T_i$$

$$\forall j, \forall k : \sum_i Y_{ijk} = 1$$

$$\forall j : \sum_i \sum_{k=0}^{|p_j|-1} Y_{ijk} = |p_j|$$

$$\forall i, \forall j : \sum_k Y_{ijk} \leq 1$$

$$\forall i, \forall j, \forall k : Y_{ijk} = 0 \text{ or } 1$$

Load capacity constraint

Storage capacity constraint

Conflict constraint for fault tolerance

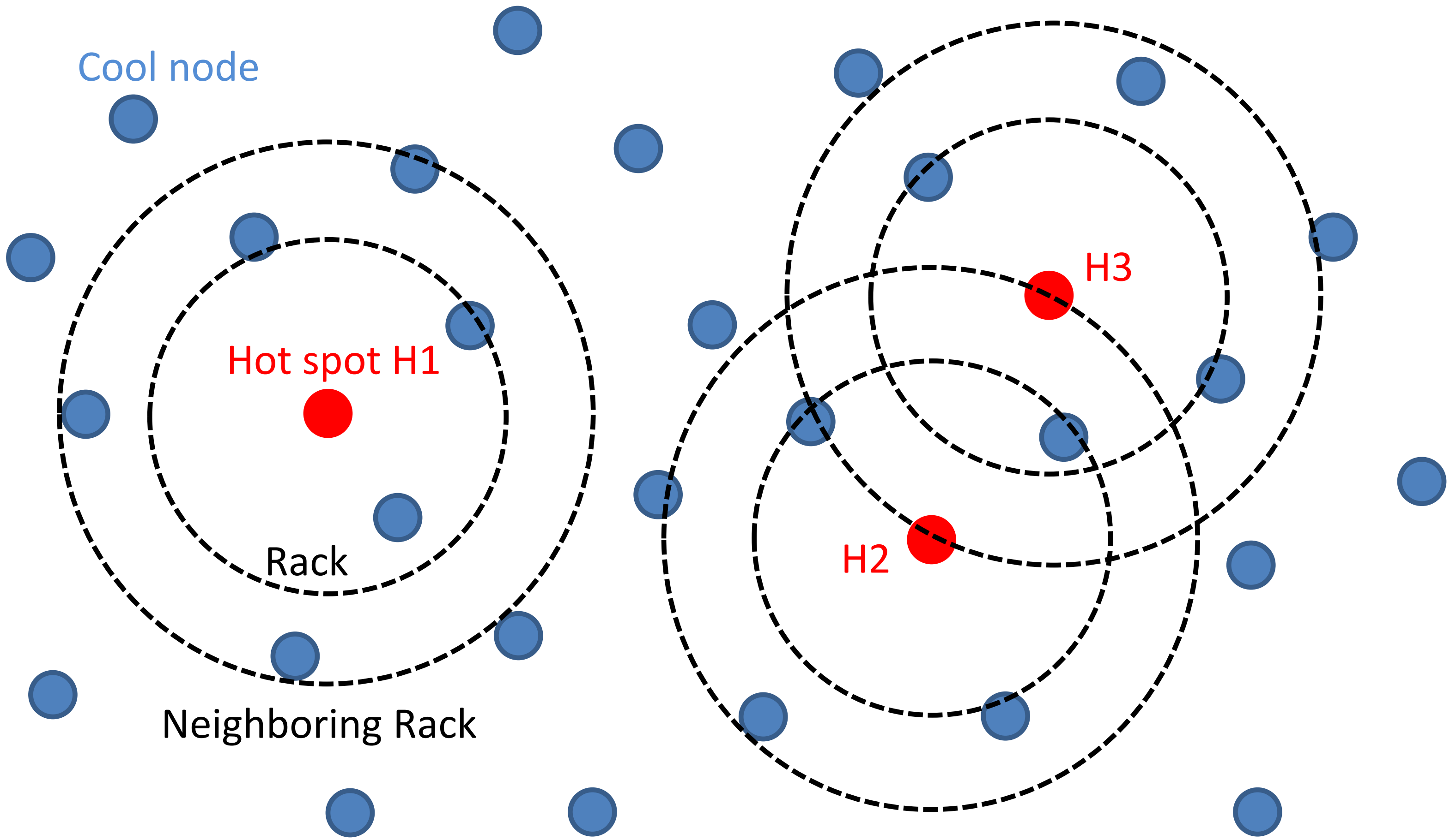
- Binary ... ation
  Y_{ijk}
- Const ... lacement
  A_{ijk} :
- B_{ijk}: Constant, bandwidth cost for moving k-th replica of j-th partition into i-th node
- C_i: Constant, load capacity of i-th node
- T_i: Constant, storage capacity of i-th node
- L_{jk}: Constant, load of k-th replicas of j-th partition
- $|p_j|$: ...

 # decision variables (Y_{ijk}) = (# nodes) * (# partitions) * (# replicas)
1,000 * 10,000,000 * 3 = 3 * 10^{10}
→ Not scalable!

Observations for enhancing scalability

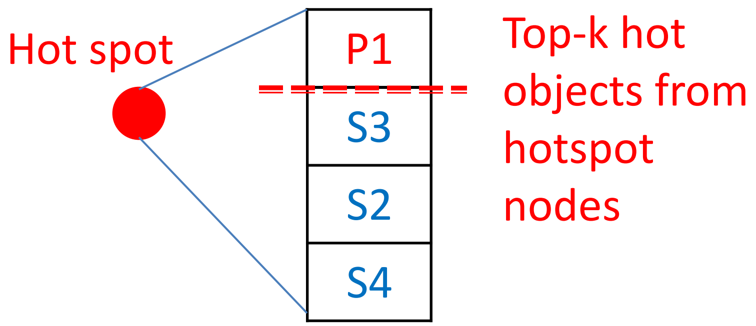
- Power law distribution
  - #hot spots << #nodes
- Topology-aware cost model limiting solution space to neighboring racks
  - Progressive strategy: e.g., first move within rack, then to neighbor rack, and so on
- Approximation
  - ILP→ LP relaxation

Progressive target node selection (logical view)

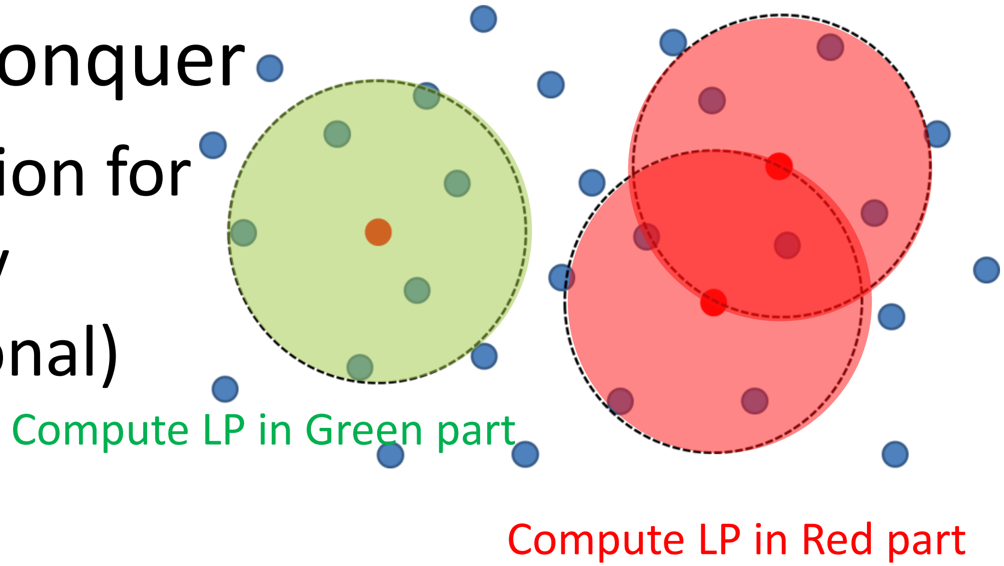


LB algo.: putting the pieces together

- Step 1: Select top-k-load objects from hotspots
  - Reduce # decision variables



- Step 2: Divide-and-conquer
  - Solve LP approximation for each hotspot (binary placement → fractional)



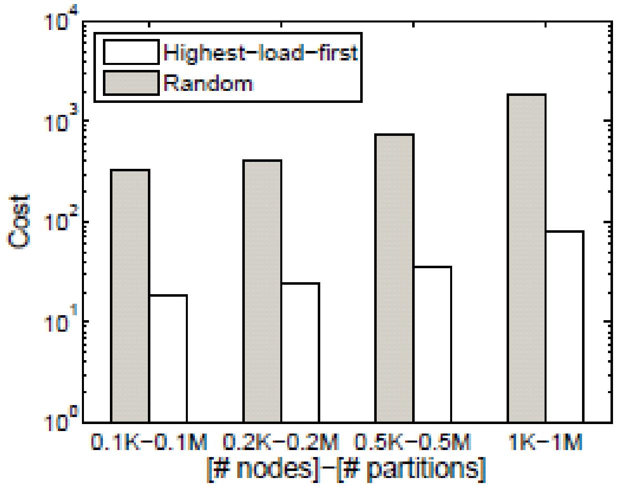
LB evaluation using simulator

- Randomly generated to emulate Hotmail traces
- Scale: 0.1K-10K nodes, 0.1M-10M objects

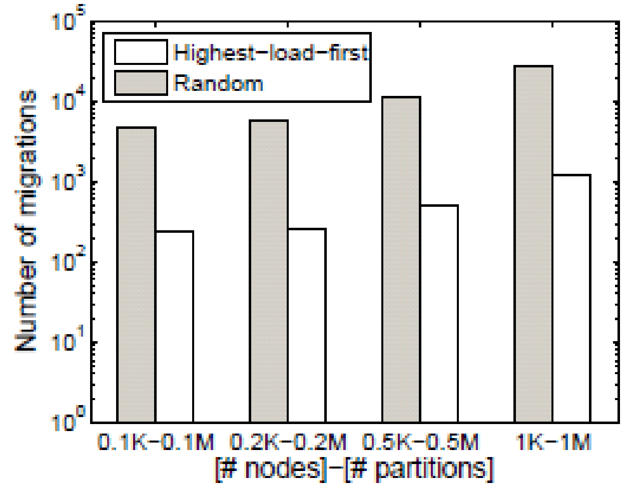
(denoted as [nodes]-[objects], e.g., 0.1K-0.1M)

- Cost: Distance
- Other parameters
  - # replicas for each partition = 3
  - Load capacity of each node = 70%
  - Swap probability = 90% (e.g., out-of-date replicas)

Highest-load-first strategy

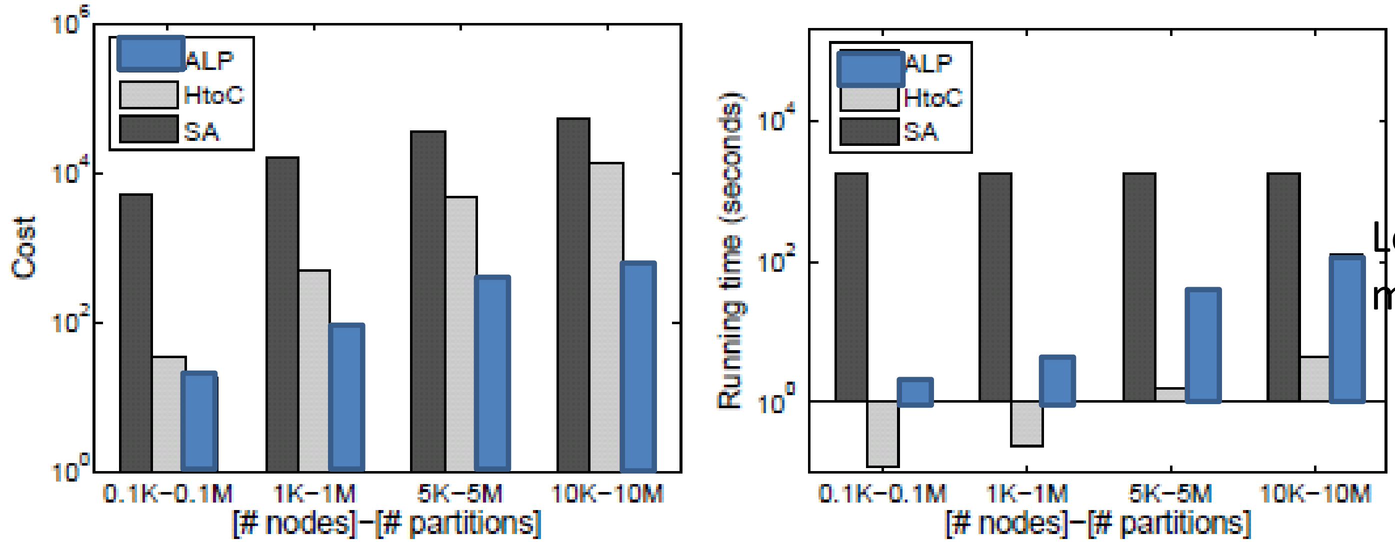


(a) Cost



(b) Number of migrations

Against baselines



(a) Cost (b) Running time

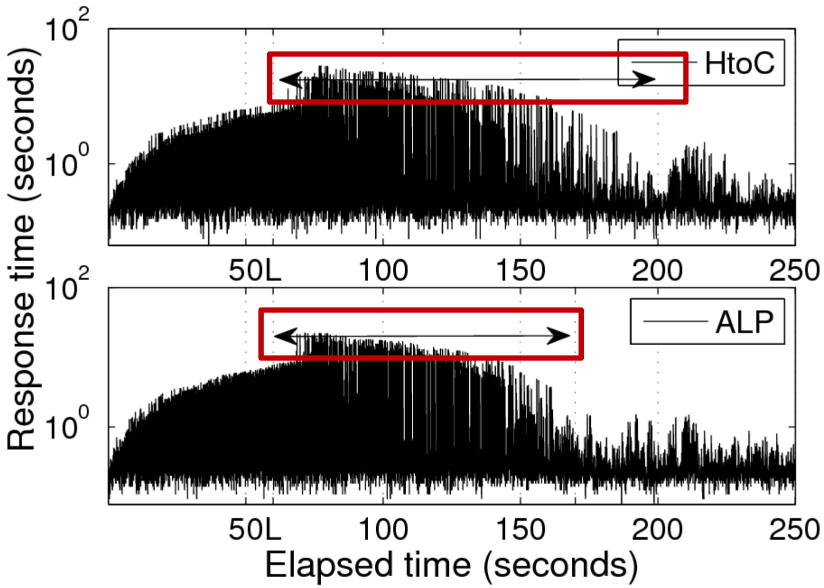
Less than two minutes

LB evaluation using Azure

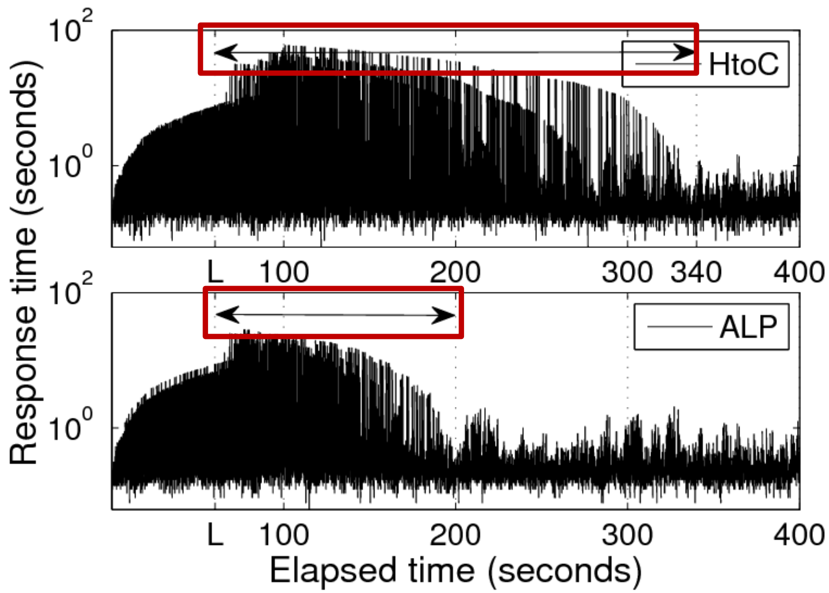
- To observe cost/running time trade-off
- Two deployment scenarios:
  - Random: Leaving Azure to deploy nodes, and as a result, have nodes deployed randomly in the near vicinity (~ almost uniform cost matrix)
  - Controlled: Emulating data center scenarios with control over which nodes go to which racks (~skewed cost matrix)

LB evaluation using Azure (@30-900)

Time to stabilization shown in arrows



Random

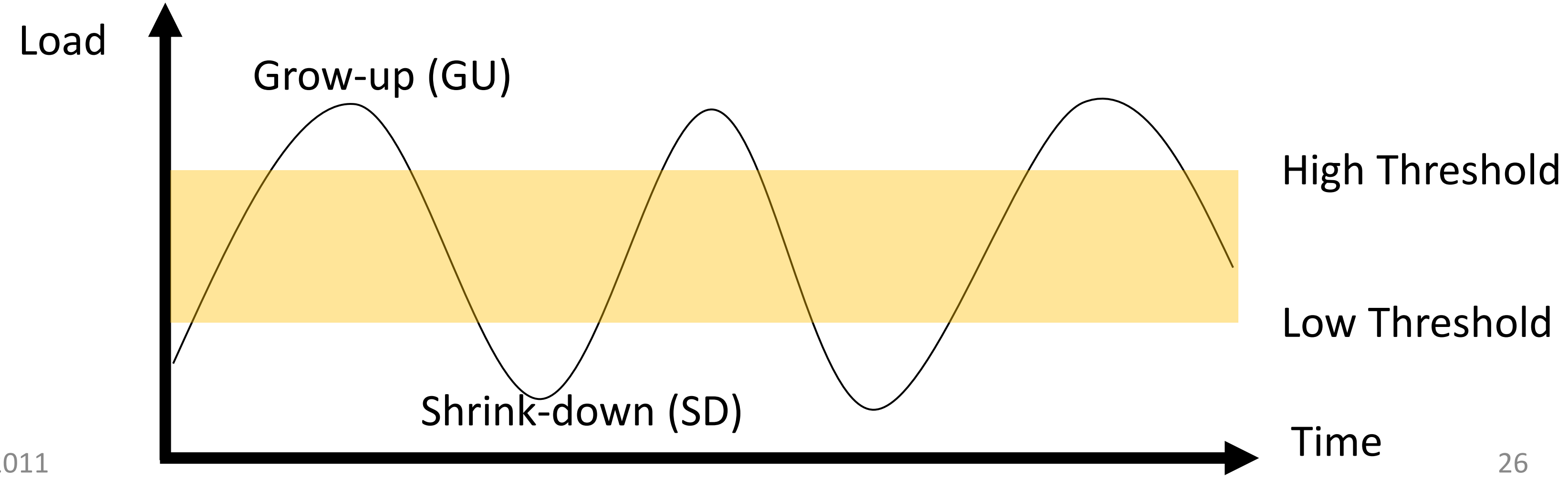


Controlled

Up to 2x differences

Power management state-of-the-arts

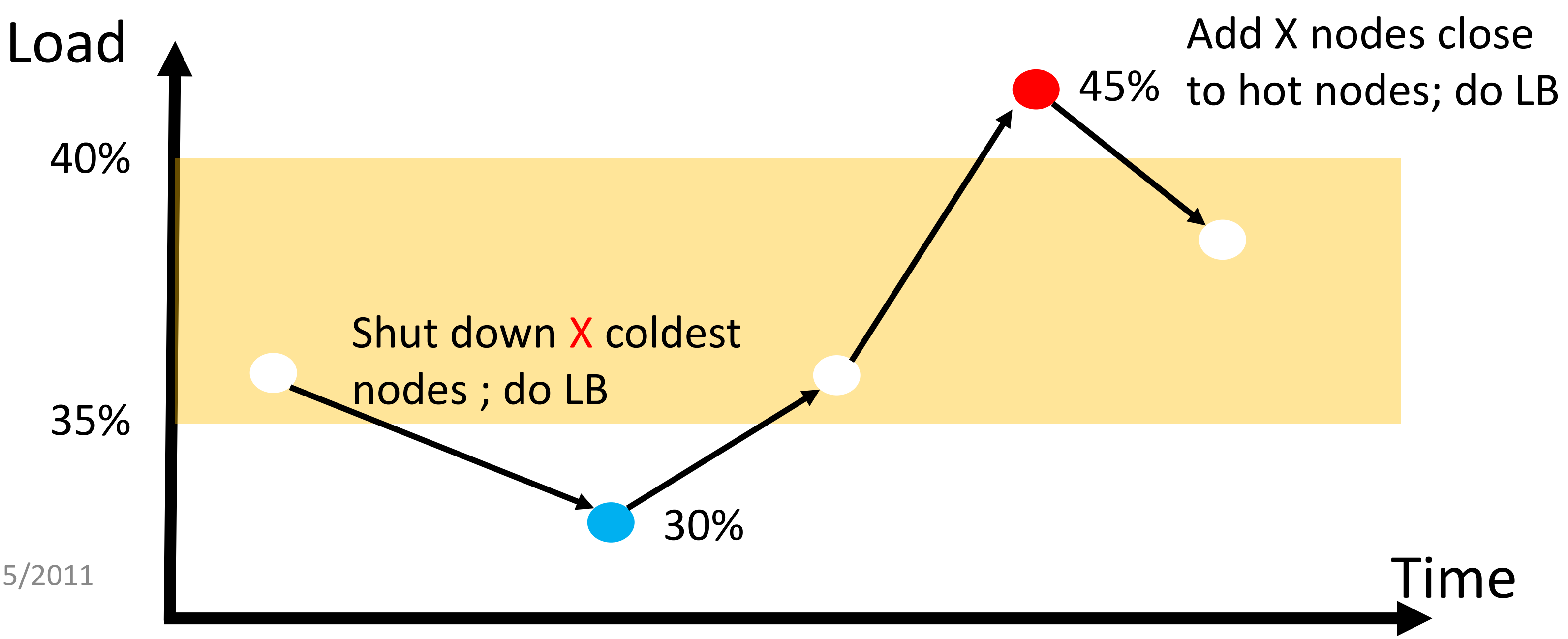
- Simple threshold-based approach
  - Avg Load < Low threshold: Shutdown X nodes
    - Swaps and moves: No primary on shutdown nodes
  - Avg Load > Upper threshold: Boot-up X nodes
    - Select new nodes to be in proximity of hotspots
- Existing research only focuses on setting X



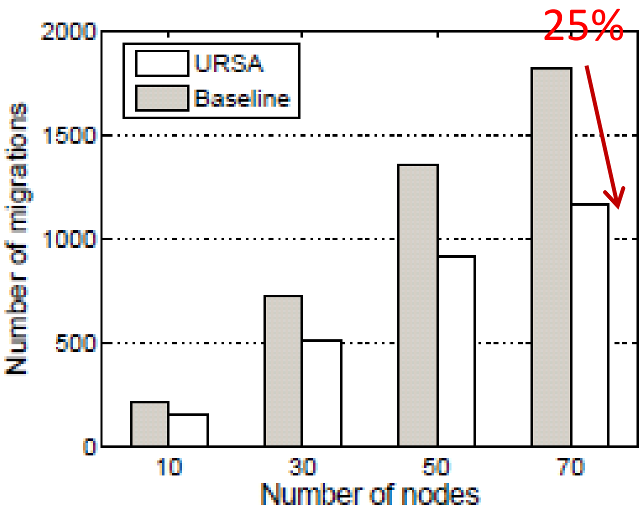
Using LB to decide

- Cost-optimal migration, after shutting down X coldest nodes (SD)
- OR, migration after adding X new nodes near hottest nodes (GU)

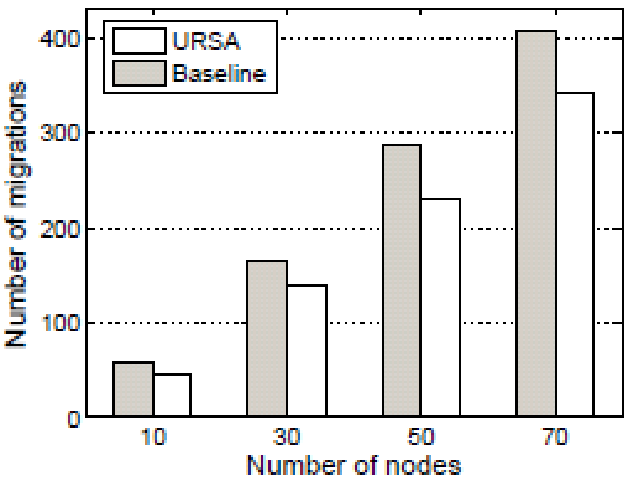
Compared with baseline: using HtoC for LB



Number of migrations

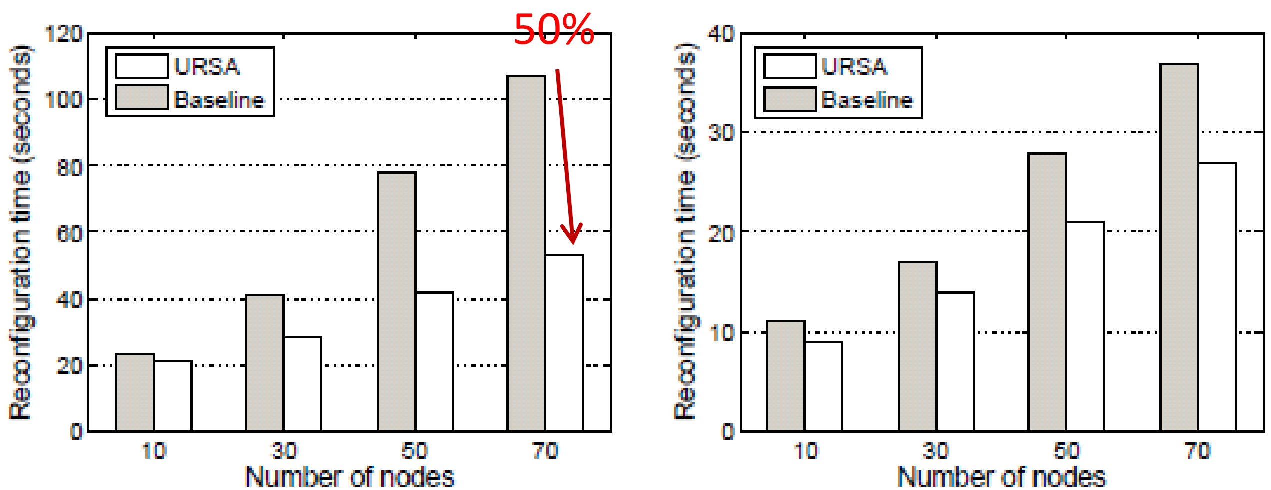


(a) Shrink-down



(b) Grow-up

Reconfiguration time



(a) Shrink-down (b) Grow-up

Ref: power saving up to 37%

Summary of contributions

- A simple, adaptive data management algorithm
  - Integrating two goals: LB and PM
  - Observations: Power law, divide and conquer, topology-aware and approximation
  - PM: Simple shrink-down and grow-up strategies
- Encouraging results on Hotmail dataset

Thank you!

www.postech.ac.kr/~swhwang