Incorporating Network RAM & Flash into Fast Backing Store for Clusters

Tia Newhall and Douglas Woos
Computer Science Department
Swarthmore College
Swarthmore, PA USA
{newhall,dwoos1}@cs.swarthmore.edu

Target Environment

- General Purpose Clusters and LAN systems
  - COTS
  - Variable, mixed workload
  - Imbalances in resource usage across nodes

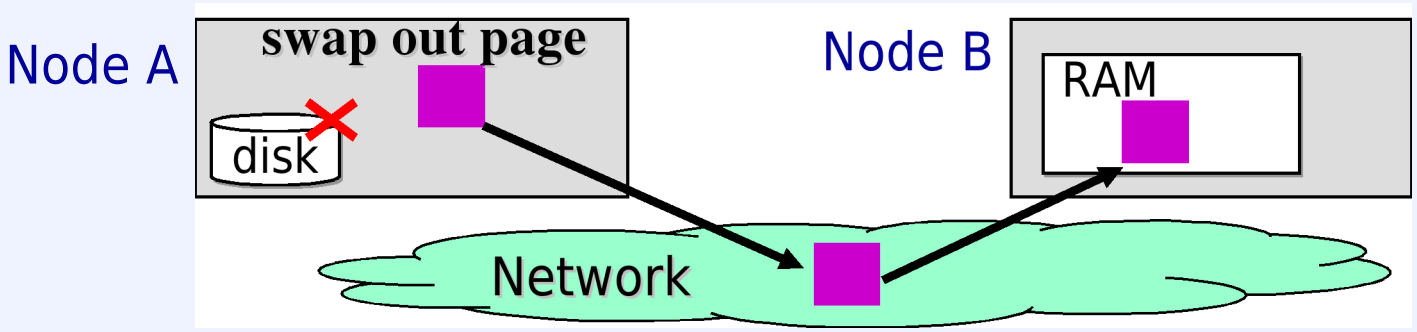
  → Some nodes have idle RAM, some overloaded RAM
- Data Intensive Computing on these systems
  - Use backing store for swap or temporary file space

  → Some nodes swapping or local disk I/O while others have idle RAM

Network RAM

- Cluster nodes share each other's idle RAM as a remote swap partition
  - Takes advantage of imbalances in RAM use across nodes
  - When one node's RAM is overcommitted, swap its pages out over the network to store in idle RAM of other nodes

+ Avoid swapping to slower local disk
+ Almost always significant amt idle RAM when some nodes overloaded
+ Free backing store



Future of Cluster Backing Store?

Disk? Flash SSD, PCM, Network RAM, …?

→ Likely more heterogeneous

- At least in the short term, but possibly indefinitely
- Different media have different strengths
  - Flash: fast reads, but erasure block cleaning, wear-out
  - Network RAM: fast reads & writes, but variable capacity, volatile

→ Likely less under control of local node's OS

- Network RAM and networked storage

Node Operating Systems

Designed assuming local disk is backing store for swap and local temporary files system data

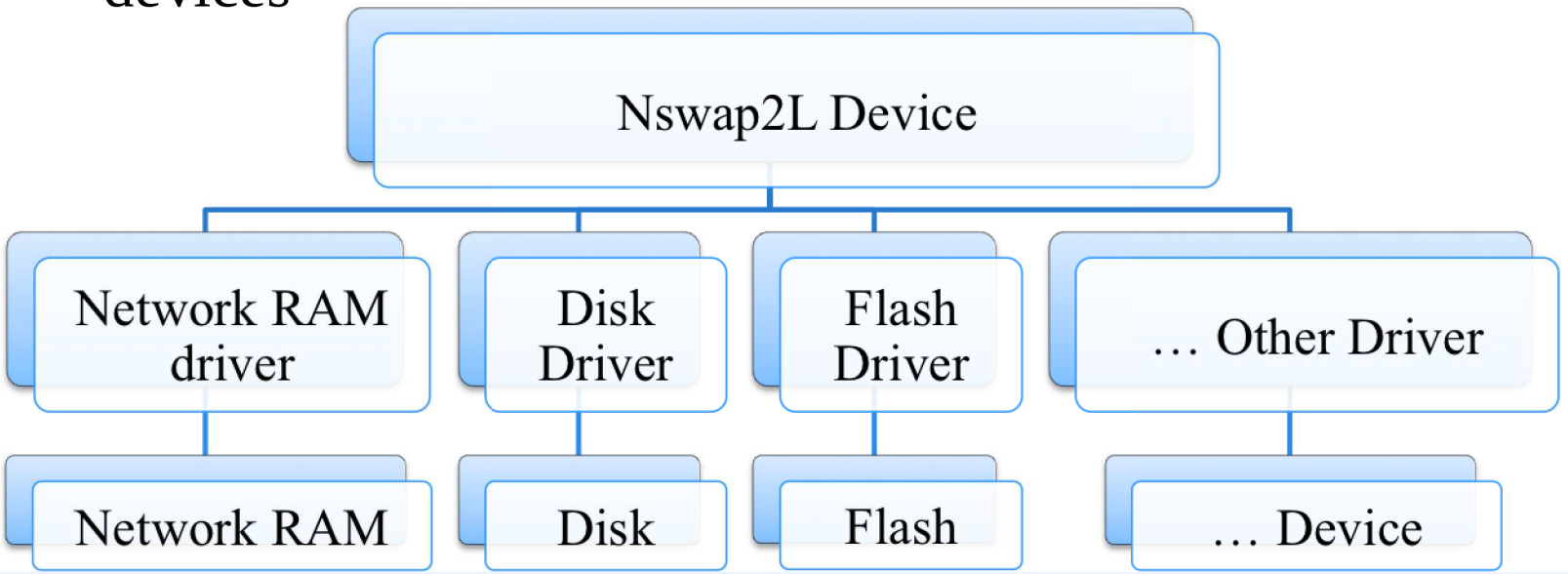
- Doesn’t fit well with new technologies
  - TRIM support helps
- Doesn’t fit well with heterogeneous set of technologies; one policy does not necessarily fit all
  - Flash: log structured writes, avoiding zero block writes, callback when data freed (to clean blocks)
  - Network RAM: noop scheduler, callback when data freed (to free remote RAM space)
  - Disk: elevator, sequential placement & prefetching

Want

- Easily incorporate new technologies as backing store for swap and local file system data
- Take advantage of strengths of different media
  - fast Writes to Network RAM, fast Reads from Flash
- Take advantage of increased I/O parallelism
- Remove from OS much of the complexity of interacting with heterogeneous set devices
  - OS sub-system policies free from assumptions about underlying backing storage device(s)
  - As technologies change, OS can still have same view of backing store “device”

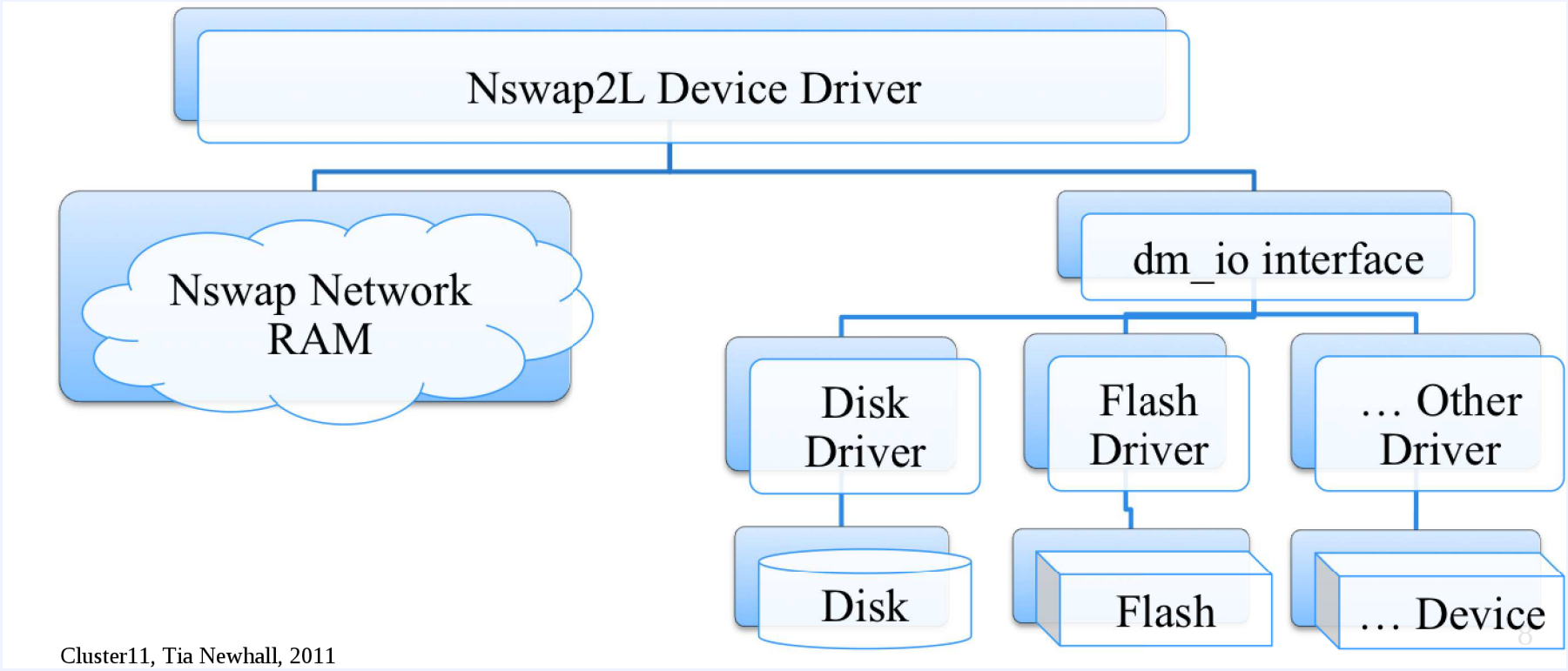
Our Solution: Nwap2L

- Conceptually, 2-levels of device driver
- Top-level Nswap2L driver is interface to OS
  - Appears as single, large, fast, random-access backing store
    - OS policies optimized for single top-level interface
  - Top-level implements data placement, migration, prefetching policies on heterogeneous low-level physical devices



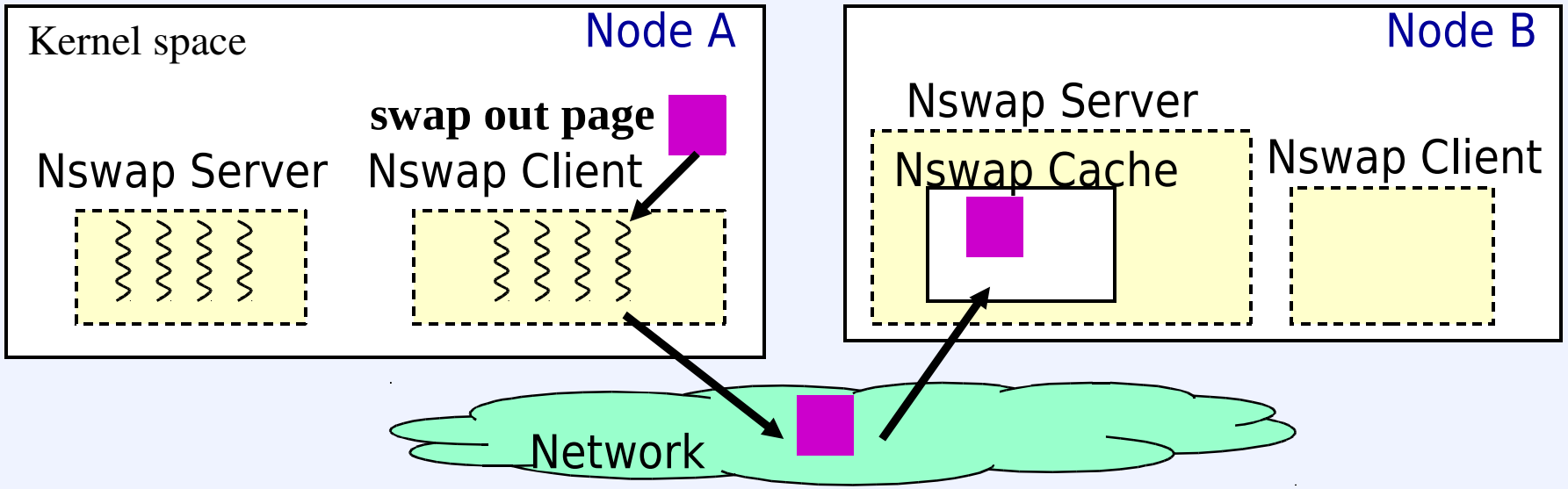
Prototype Implementation

- Top-level is Linux 2.6 lkm block device driver
  - Can be added as a swap device to individual cluster nodes
- Top-level directly manages Nswap Network RAM
- Top-level uses Red Hat's dmio interface to interact with other low-level device drivers (disk, Flash, …)



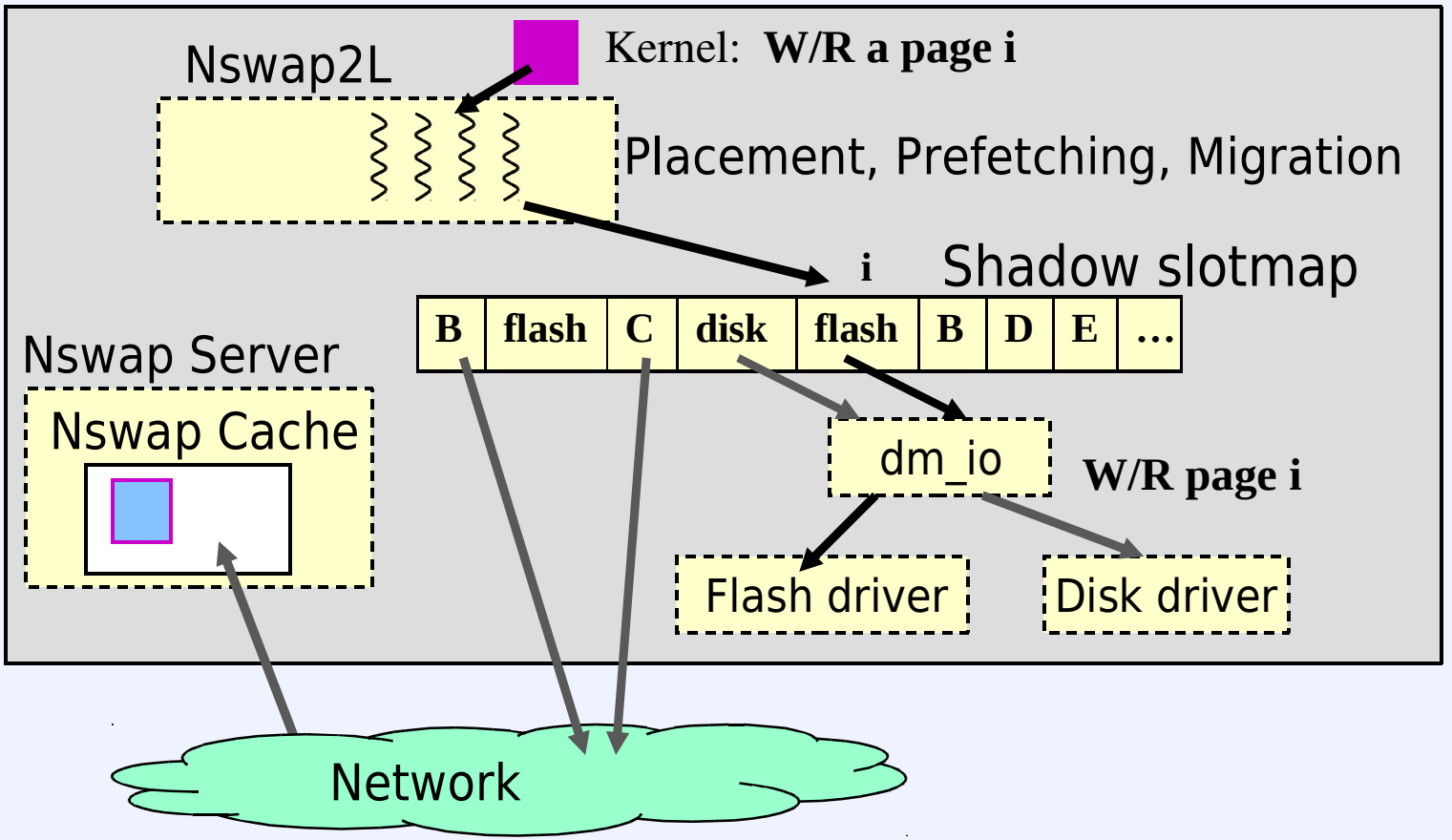
Nswap Adaptable Network RAM

- P2P Design: Each node runs a multi-threaded client & server
  - Client is active when node swapping (needs more RAM)
  - Server is active when node has idle RAM available
- Each node manages is part of RAM currently available for storing remotely swapped pages data (Nswap Cache)
  - Nswap Cache size grows/shrinks with local process needs
- Implemented as Linux lkm block device driver



Nswap2L Implementation

- Nswap2L Driver Client + Nswap Server
  - Shadow slotmap encodes placement on underlying device



Nswap2L vs. Other Swap Devices

Benchmark	Nswap2L (speedup)	Nswap	Flash	Disk
WL1	**443.0 (3.5)**	471.8	574.2	1,547.4
WL2	**591.6 (30.0)**	609.7	883.1	17,754.8
WL4	**578.9 (30.9)**	591.7	978.4	17,881.2
Radix	**110.7 (2.3)**	113.7	147.4	255.5
IS	**94.4 (2.4)**	93.1	107.6	224.4
HPL	536.1 (1.5)	**529.7**	598.7	815.3

- Nswap2L (to NW RAM only) and Nswap perform best
- Flash is close to Nswap and Nswap2L

Device Speeds in our System

	Direct Large Read via /dev	**Direct Large Write via /dev**
Flash SSD	23.5	32.7
Nswap	21.7	20.2

12 node cluster, 1GB Ethernet, Intel X25-M SATA1 80GB Flash SSD

- Nswap Network RAM is faster
- Flash reads are comparable to Nswap Reads

→ Write to Network RAM and Read from both

Prefetching between devices

- Take advantage of fast writes to Network RAM and fast reads from Flash
  - Increase write speed by always writing to fastest device
  - Prefetch some blocks from Network RAM to Flash which has better read performance than write
  - Results in increased read parallelism by distributing reads over multiple devices
- Prefetching between low level devices can be much more aggressive than prefetching from backing store to memory

Prefetching Policy Questions

Q1: When should prefetching occur?

if swapping since last check, periodically,…

Q2: How many pages should be prefetched?

fixed amount, % recently swapped, % total swapped

Q3: Which slots/pages should be prefetched?

RR, Random, LRS (LeastRecentlySwapped), MRS

Q4: From which device(s), to which device(s)?

from Network RAM to Flash

- Frees up Network RAM space for future writes
- Increases Parallel reads

Prefetching Experiments

- Placement policy: pick Network RAM first, Flash only when no available Network RAM
- Prefetching polices
  - Q1: periodically
  - Q2: 10% of number of swap outs since last prefetch activation
  - Q3: Random, LRS, MRS, RR of slots
  - Q4: From Network RAM to Flash

Degree of Read Parallelism

	WL1	WL2	IS	Radix	HPL
No prefetching	5.5	5.7	5.6	5.4	5.2
Prefetching	3.8	5.3	6.1	13.7	13.1

(ave number of concurrent reads)

Parallel workloads benefit more than sequential

13.7 vs. 5.4

Due to:

access patterns in sequential

multiple processes in parallel

Prefetching Read/Prefetch Ratios

Policy	WL1	WL2	IS	Radix	HPL
RR	1.1	3.0	1.7	0.5	0.9
Random	1.2	3.2	1.4	0.5	0.8
LRS	1.1	2.7	1.9	0.2	0.8
MRS	1.2	3.0	1.6	0.4	0.8

- Best Policy differs for different workloads
- MRS surprisingly isn't always best, but not ever the worst, might be good general policy

Computed Ideal Runtimes

- Measured parts of execution time
  - dmio adds 700% overhead to Flash I/O vs. direct read and write to Flash via /dev
- Ideal runtime (no dmio overheads) =

$$(P_{NS} \times TotalTime)$$

$$+(P_S \times (TotalTime - FR_{w\_dmio} + FR_{no\_dmio}))$$

- Can also use this to compute runtimes for cases when Flash is faster than Network RAM

Computed Runtimes

	Control	Ideal (no dmio)	Flash 10% < NW	Flash 20% < NW
WL1-Random	455.8	461.8	450.1	445.3
HPL-LRS	628.4	600.3	597.0	595.9

- HPL faster with prefetching to slower Flash than NW RAM alone (Control)
  - Increased parallelism in reads over NW and Flash
- On systems with faster Flash than NW, prefetching to Flash performs better for both workloads

Conclusions

- Nswap2L Provides a high-level interface of single, fast, random storage device on top of heterogeneous physical storage.
- Our prototype system supports general design, when used as fast swapping device in clusters
- Prefetching and placement policies result in faster execution times
  - Even when distributed over Network RAM and slower Flash faster than Network RAM alone

Future Work

- Implementation that removes high overheads with how we are using dmio
- Further investigate prefetching and placement policies
  - Adaptive policies?
- Add support for using Nswap2L as backing store for local temporary file system
  - STXXL, TPIE libraries for large data sets
  - FS block size vs. swap page size
  - Persistence guarantees?

Acknowledgements

- This work partially funded by NSF CSR-RUI
- Many Swarthmore undergraduate students involved in the Nswap project

- For more information:

`www.cs.swarthmore.edu/~newhall/nswap`

Questions?