The RAMDISK Storage Accelerator

A Method of Accelerating I/O Performance on HPC Systems Using RAMDISKs

Tim Wickberg, Christopher D. Carothers
wickbt@rpi.edu, chrisc@cs.rpi.edu

Rensselaer Polytechnic Institute

Background

- CPUs performance doubles every 18 months
  - HPC system performance follows this trend
- Disk I/O throughput doubles once every **10 years**

- This creates an exponentially widening gap between compute and storage systems
  - We can't continue to throw disks at the problem to keep current compute to I/O ratio intact
  - More disks not only cost more, but failure rates also cause problems

RAMDISK Storage Accelerator

- Introduce new layer to HPC data storage – the RAMDISK Storage Accelerator (RSA)
- Functions as a high-speed data staging area
- Allocated per job, in proportion to compute resources
- Requires no application modification, only a few settings in job scripts to enable

RSA

- Aggregate RAMDISKs on RSA nodes together using a parallel filesystem
  - PVFS in our tests
  - Lustre, GPFS, Ceph and others possible as well
- Parallel RAMDISK is exported to I/O nodes in the system

Bind mount

- `mount -o bind /rsa /place/on/diskfs`

- Application sees a single FS hierarchy, doesn't need to know if the RSA is functional or not
  - Decouples RSA from the compute system, allows the application to function regardless of RSA availability

Scheduling

- Set aside half the I/O nodes for active jobs, and the other half for jobs that have finished or will start soon
- Allocate RSA nodes in proportion to the compute system
- Data moves asynchronous to job execution, and frees the compute system up sooner.

Example job flow

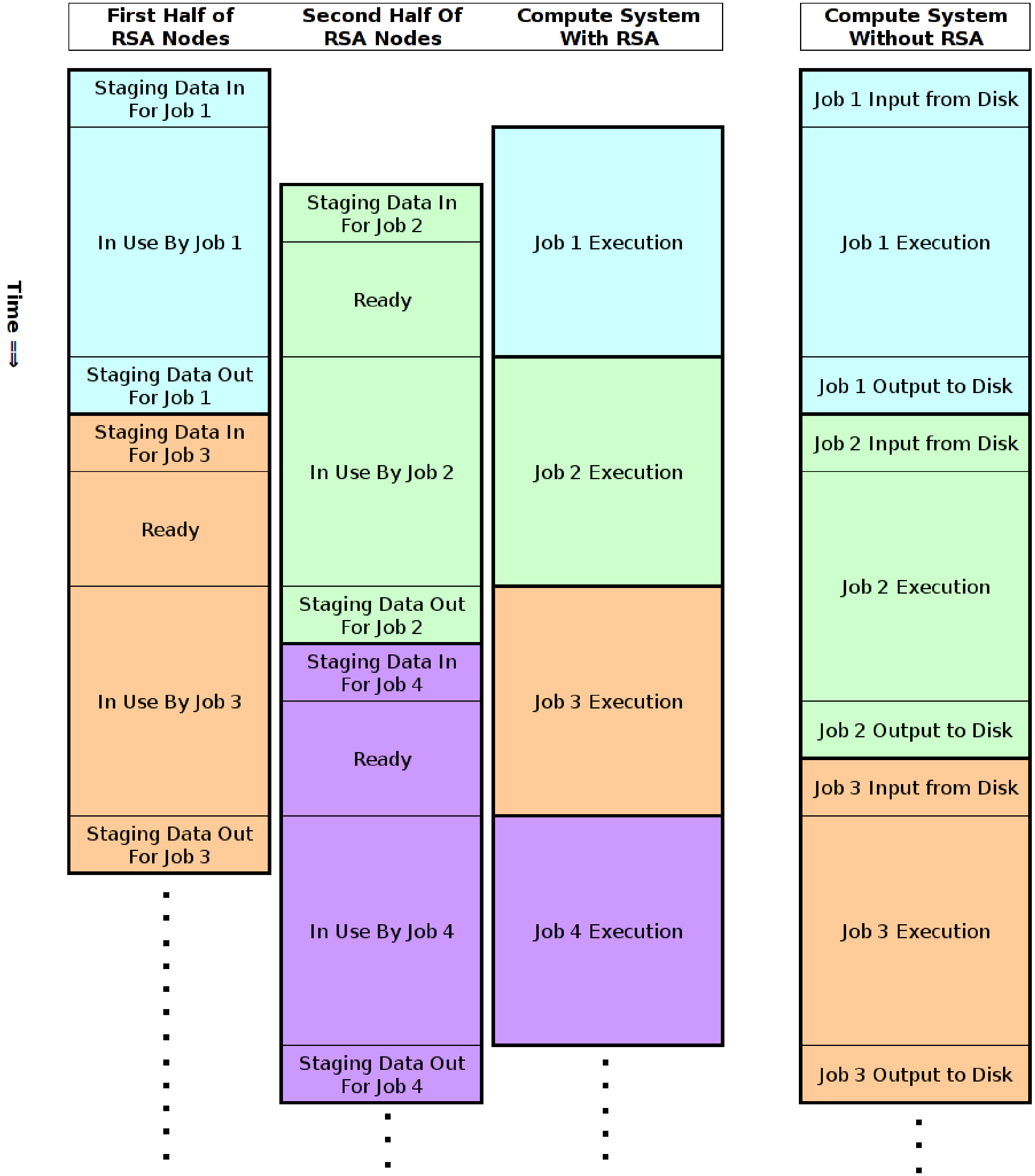
- Before job begins, selected as next-likely to start.
  - Stage data in to RSA.
- Job Starts on compute system
  - Reads data in from RSA
- Checkpoints to RSA
- Job execution finishes
  - results written to RSA
- Compute system released
- RSA pushes results back to disk storage **after** compute system has moved on to the next job

Compare to traditional job flow

- Initial load: 15 minute read in from disk
- Checkpoints: 10 minutes per checkpoint, once an hour, 24 hour job run
- Results back out: 10 minutes
- 225 minutes spent on I/O

RSA

- Initial load: happens before compute job starts
- Checkpoints: 1 minute each
- Results out: 1 minute to RSA
- Afterwards: results from RSA back to disk storage
- 25 minutes spent on I/O
  - Saved 200 minutes on the compute system
  - Asynchronous data staging seems likely for **any** large scale system at this point

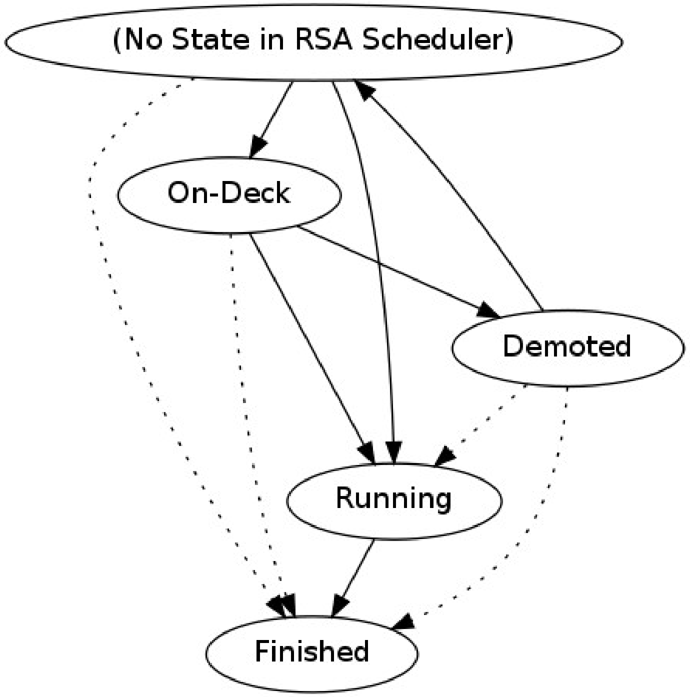

First Half of RSA Nodes
Second Half Of RSA Nodes
Compute System With RSA
Compute System Without RSA
Time ⇒
Staging Data In For Job 1
In Use By Job 1
Staging Data Out For Job 1
Staging Data In For Job 3
Ready
In Use By Job 3
Staging Data Out For Job 3
Staging Data In For Job 2
Ready
In Use By Job 2
Staging Data Out For Job 2
Staging Data In For Job 4
Ready
In Use By Job 4
Staging Data Out For Job 4
Job 1 Execution
Job 2 Execution
Job 3 Execution
Job 4 Execution
Job 1 Input from Disk
Job 1 Execution
Job 1 Output to Disk
Job 2 Input from Disk
Job 2 Execution
Job 2 Output to Disk
Job 3 Input from Disk
Job 3 Execution
Job 3 Output to Disk

Test System

- 1-rack Blue Gene/L
- 16 RSA nodes (borrowed from another cluster), 32GB RAM each
- Due to networking constraints, a single Gigabit Ethernet link connects the RSA to the BG/L functional network.
    - Bottleneck evident in results.

Example RSA Scheduler

- RSA scheduler implements the scheduling, RSA construction/destruction, data staging
- Proof-of-concept constructed alongside the SLURM job scheduler



Test Results

- More details in the paper, but briefly:
  - Example test: file-per-process, 2048 processes, 2GB data total
  - GPFS disk: 1100 seconds to write out test data
    - High contention exacerbates problems with GPFS metadata locking
  - RSA: 36 seconds to write to RSA
    - 222 seconds after compute system is released to push data back to GPFS
      - Data staged back from single system avoids GPFS contention issues
  - 2800% speedup from the application / compute system's viewpoint

Table 2: Comparison of output performance of GPFS disk storage vs. RSA, 1024 nodes (2048 processes) in file-per-process, with normal and custom post-processing scripts. Times are in seconds.

	No RSA	RSA With Default Stage-Out		RSA with Custom Stage-Out	
	Output	Output	Stage-Out	Output	Stage-Out
Run 1	1158.81	36.11	222	35.76	179
Run 2	1193.92	36.25	227	36.25	178
Run 3	976.84	35.26	224	35.97	181
Mean	1109.86	35.87	224	36.43	179

Future Work

- Full-scale test system to be implemented @ CCNI in the next six months
- 1-rack (200 TFLOPS) Blue Gene/Q
- 32 RSA Nodes, each 128GB+ RAM, 4TB+ total.
- FDR (56Gbps) Infiniband fabric

Extensions

- RAM is faster, but SSDs are catching up, and provide better price/capacity.
  - Everything shown here can extend to SSD-backed systems as well.
- Overhead in Linux memory management.
  - Data copied in-memory ~5 times on the way in or out, this could be reduced with major modification to the kernel. Or, perhaps a simplified OS could be developed to support this.
- Handle RSA scheduling directly in job scheduler, rather than external

Conclusions:

- I/O continues to fall behind compute capacity
- The RSA provides a method to mitigate this problem
  - Frees the compute system faster, reduce pressure on disk storage I/O
- Possible to integrate into HPC systems **without** changing applications

Thank You

Questions?