Dynamic Allocation of Nodes on a Large Space-shared Cluster

Lisa Kennicott   Lee Ann Fisk

Sandia National Laboratories

Albuquerque, New Mexico

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

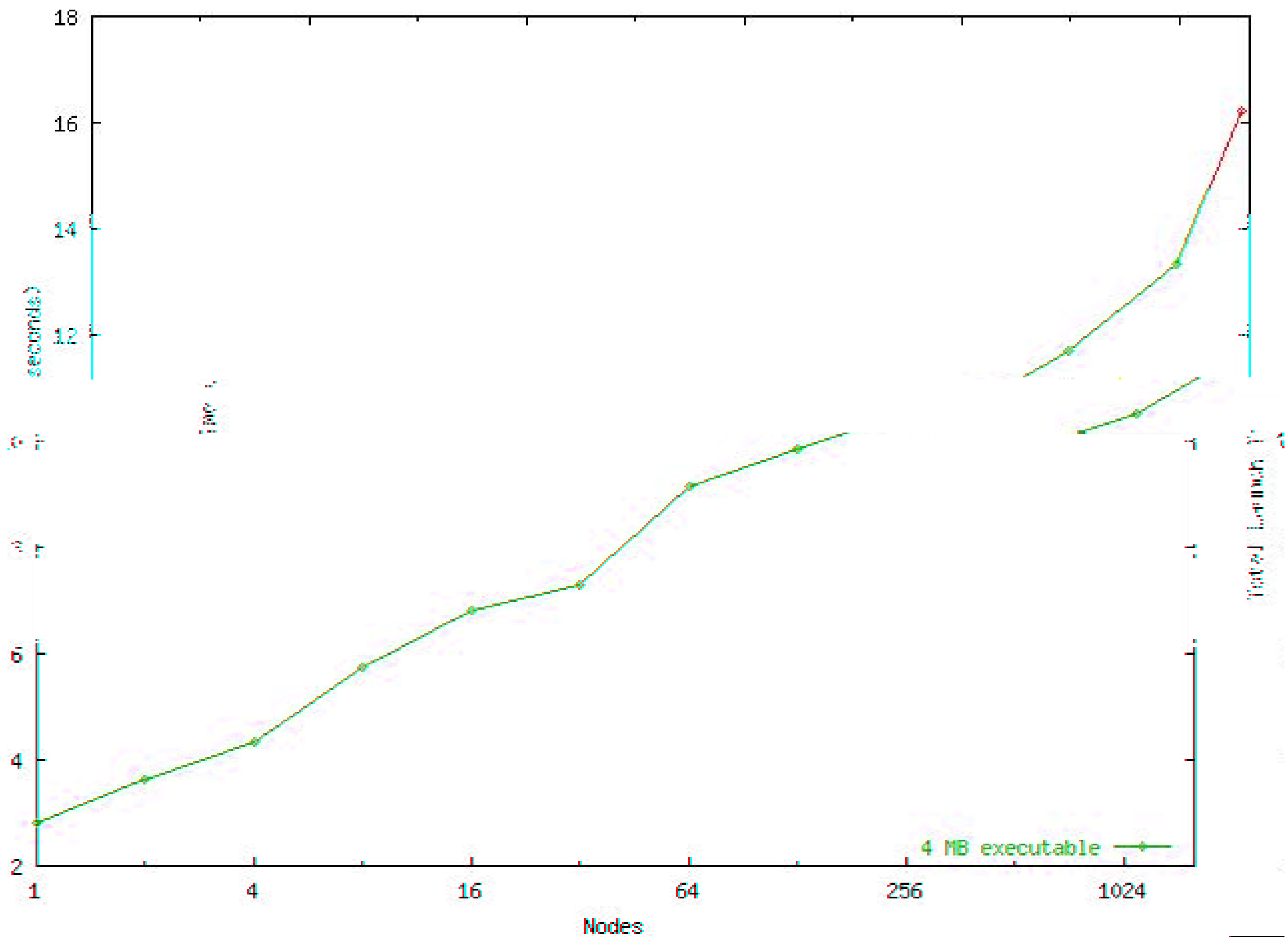
Outline

- Cplant™ runtime system overview
- Dynamic allocation motivation
- Simulation model
- Results
- Implementation decisions
- MPI-2 implementation

Runtime Environment Components

- **Yod**
  - **Parallel job launcher**
  - **Fans out executable and environment to compute nodes**
- **Bebopd**
  - **Compute node allocator**
  - **Allocates compute nodes to parallel jobs**
- **PCT (Process Control Thread)**
  - **Compute node daemon**
  - **Manages the compute and memory resources on a node**
- **Pingd/Showmesh**
  - **Compute node status tool**
  - **Displays the state of the compute nodes**

Application Launch Performance



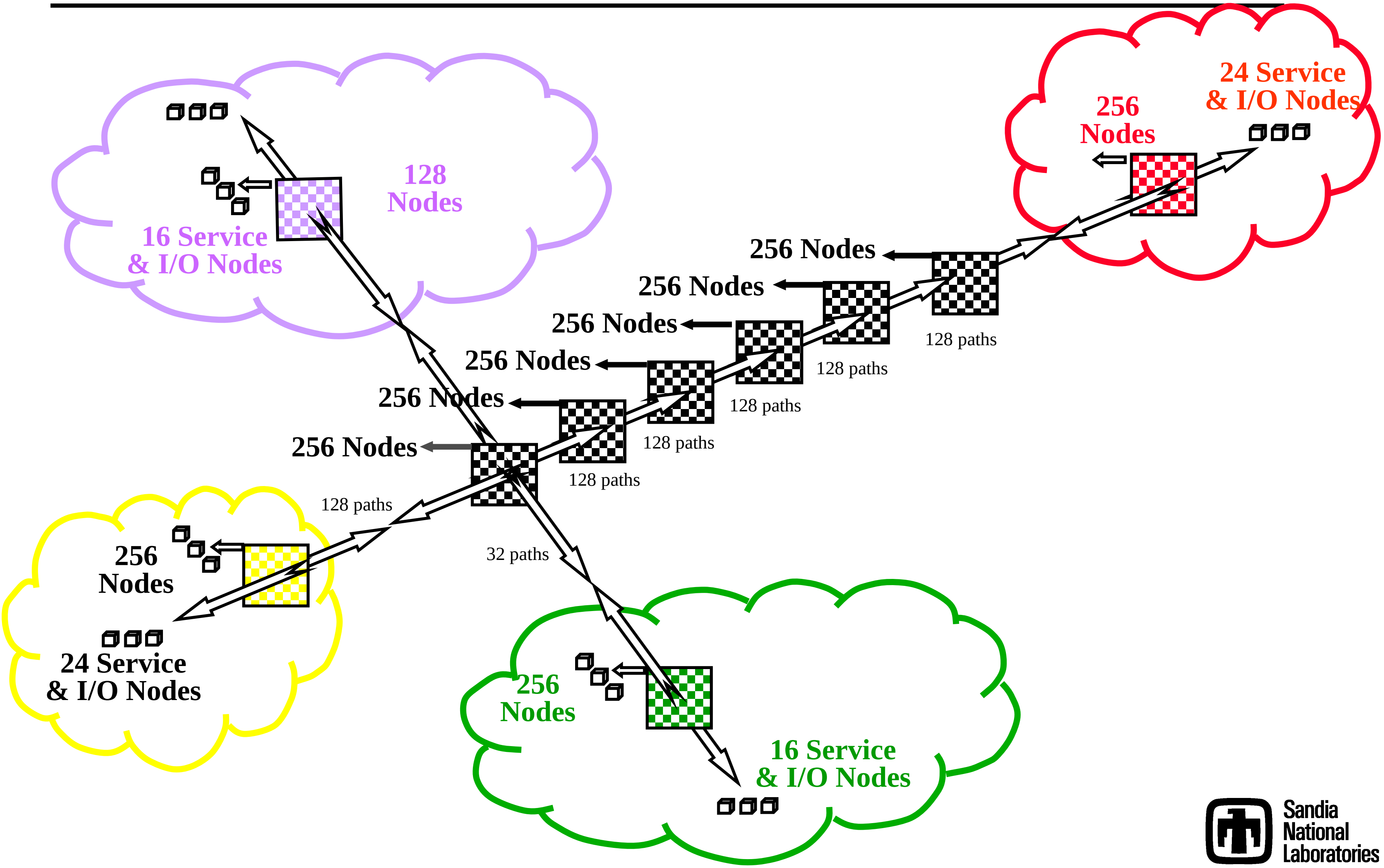
PBS on Cplant™

- Added a "size" abstract resource to PBS
- PBS knows about total size and how much size is requested by jobs in the queue
- Bebopd notifies the PBS server when the number of compute nodes available to PBS changes
- Bebopd knows if a yod request is originating from a PBS job script and insures that a job does not exceed its size
- PBS insures that jobs do not exceed their allocated wall time
- PBS MOM daemons only run on each service node, not each compute node

Cplant™ Alaska Cluster

- 400 Digital PWS 500a (Miata)
- 500 MHz Alpha 21164 CPU
- 2 MB L3 Cache
- 192 MB ECC SDRAM
- 16-port Myrinet SAN/LAN switch
- 32-bit, 33 MHz LANai-4 NIC
- 6 DEC AS1200, 12 RAID (.75 Tbyte) || file server
- 1 DEC AS4100 compile & user file server
- Integrated by Compaq

Cplant™ Antarctica Clusters



Dynamic Allocation Overview

- Size (# nodes) of a running job can increase or decrease mid-run, new nodes are dynamically allocated to the job.
- System software detects conditions that are appropriate for DA and tells job(s) to reconfigure.
- Job reconfigures to more/less nodes at next convenient point (user-defined)
- Proposed to improve scheduling, system utilization, fragmentation, improved running time of some jobs
- Can be a very responsive scheduling policy to sudden changes in system load

But…. It's a mixed bag

- Some studies find medium/large jobs do well - others find small jobs do better (until job arrival rates go up)
- Reconfiguration costs must be kept low - then it's great. But how easy is this?
- Low job arrival rate (more time to determine best reallocation) vs. high job arrival rate (less time available)
- High system utilization rates and high reconfiguration costs can cause thrashing
- Costs/benefits difficult to quantify and depend on implementation scheme
- So… Some groups are considering other scheduling policies such as backfilling and gang-scheduling

Different View of Dynamic Allocation

- Application decides when it needs more nodes vs. system detects conditions and orders job to change
- Application requests additional nodes. System fulfills it at earliest convenient time
- Goal: minimize interference of this new scheduling policy on OTHER jobs in the system (non-DA jobs)
- Metric is job queue time. In a space-sharing machine run-time of non-DA jobs will be minimally impacted
- We don't see dynamic allocation as a performance enhancement tool, but as a necessary and desirable scheduling feature for certain classes of jobs

Why are we interested in Dynamic Allocation?

User surveys and discussions revealed:

- Some jobs NEED it (calculations cannot continue unless additional nodes are available at some point mid-run). Otherwise you have to request many nodes and leave them sitting idle for potentially long times.
- Some jobs WANT it (load balancing is complex code; running time may improve)
- NEED: Finite element codes that employ adaptive mesh refinement
- WANT: Simulation codes (e.g. shock physics codes)

Why simulate?

- Cost of developing it and being wrong too high
- Test out several different possible implementations in a short period of time
- Provide users a reasonable picture of costs/benefits to dynamic allocation

- But, it's important to validate the model to make sure we can fully trust our results

Simulation Models

- DA jobs can make a request for additional nodes at any time
  - "Need" request model
    - Additional nodes requested all at once
    - Single request for 50% more nodes 60% of the way through original job run
  - "Want" request model
    - Smaller number of nodes is requested several times throughout the run
    - Requests 20 % of original node count at 25%, 50%, and 75% of the way through the original job run

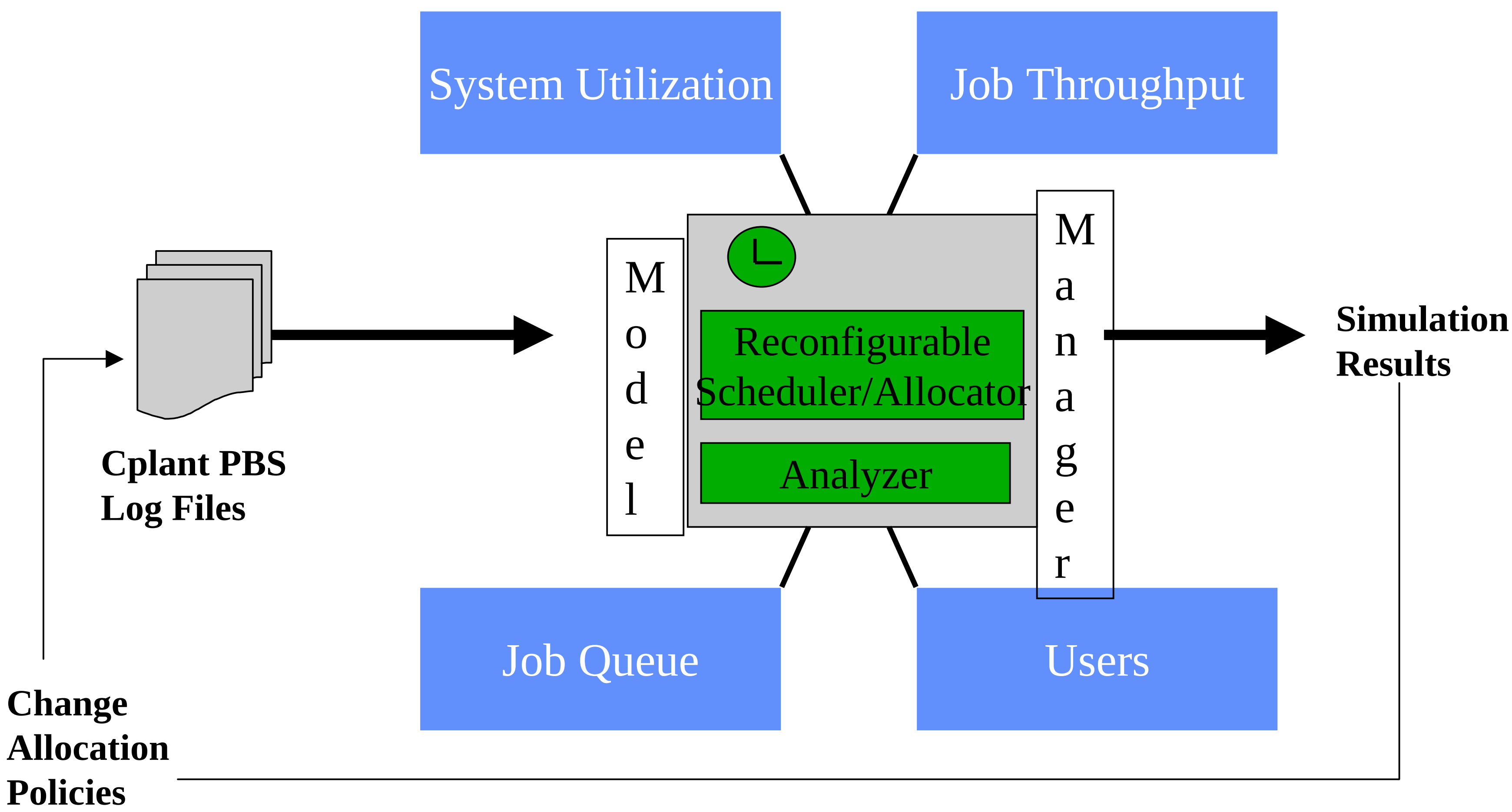
Simulation Models (cont'd)

- DA "children" jobs must enter the job queue and are subject to various combinations of scheduling policies:
  - Priority - system gives scheduling priority to DA children jobs over other non-DA jobs in the queue
  - Non-priority - system gives equal priority to DA children and other non-DA jobs in the queue
  - Blocking vs. non-blocking - can the job do useful work until the additional nodes are allocated?
  - What are the costs of blocking? Should blocking be prohibited?

Simulator Description

- OO model of jobs, queues, processors, scheduling policies
- Scheduling policies can be quickly changed to play “what if” games
- Closely models Cplant™ scheduling (fair share, large vs. small job priorities, starvation policies, scheduled system time, etc.)
- Relied on past history of system usage rather than probabilistic predictions of job size, arrival rates, etc. We believe this gives a more accurate view of what’s really going on in the system
- Verified by taking historical job logs and matching results of simulation with history
  – Closely matched historical runs on Cplant with the exception of some human operator interference (minimal)
- All jobs - even those that immediately failed - were modeled. Even jobs that fail quickly affect job scheduling especially if they’re large

Simulation Model



Dynamic Allocation Results

Average wait times. Jobs requesting dynamic allocation are < 64 nodes in size.

SINGLE REQUEST	No Priority		Priority	
	Blocking	*Non-blocking*	*Blocking*	*Non-blocking*
Very small (p<16)	2621	2615	2607	2604
Small (16≤p<64)	3161	3161	3164	3164
Medium (64≤p<128)	8999	8970	8945	8937
Large (128≤p)	103041	103041	103041	103041

TRICKLE REQUESTS	No Priority		Priority	
	Blocking	*Non-blocking*	*Blocking*	*Non-blocking*
Very small (p<16)	2602	2586	2571	2572
Small (16≤p<64)	3161	3161	3171	3163
Medium (64≤p<128)	9098	8971	8954	8936
Large (128≤p)	103041	103041	103041	103041

Dynamic Allocation Results

Average wait times. Jobs requesting dynamic allocation are 64 -128 nodes in size.

One Request	**No priority**		**Priority**	
	Blocking	**Non-blocking**	**Blocking**	**Non-blocking**
Very Small (p<16)	2413	2361	3937	3538
Small (16≤p<64)	4186	3649	4171	4430
Medium (64≤p<128)				
Large (128≤p)	103041	103041	134607	103041

Trickle requests	**No priority**		**Priority**	
	Blocking	**Non-blocking**	**Blocking**	**Non-blocking**
Very Small (p<16)	5130	2360	5004	2955
Small (16≤p<64)	7700	3646	21618	3846
Medium (64≤p<128)				
Large (128≤p)	104350	103044	121908	103042

Dynamic Allocation Results

- With our current system usage and job distribution, it doesn't matter how we implement dynamic allocation.
- As the job size increases or the system utilization increases, this is no longer true.
- Cost of dynamic allocation on the rest of the system is heaviest for the "trickle" requests. Worse effect on small jobs than on large jobs.
- In general, it is better not to give scheduling priority to "dynamic children" for our implementation.
- Blocking until additional nodes are granted generally affects the rest of the system more than non-blocking.
- Small jobs do not receive additional nodes immediately 13-17% of the time. Large jobs 14-26% of the time.

Implementation Decisions

- Implementing DA in a no-priority, no-blocking (when possible) approach minimizes effects on other jobs in the queue
- Compromise: Go with no-priority but with blocking (ease of implementation) and give user the ability to call to layers below MPI for non-blocking
- Also found our scheduling policies could be improved
- Considering cycle-stealing jobs and backfilling

MPI-2 DA Support

- Use MPI_Info object for
  - wdir – working directory for new processes
  - timeout - timeout in seconds to wait for new processes to start
  - soft – extended specification of the number of new nodes that could be allocated (2:2:10 or 16,32)
- Provide non-portable library routines for non-blocking spawning

Conclusions

- Based on simulation data, implementing DA in a no-priority, non-blocking fashion is preferable
- Currently have an approach that allows for blocking, no-priority to be consistent with MPI-2
- Will also provide non-portable non-blocking spawning as well

Future Work

- **Deploying backfilling policy to alleviate fragmentation as workload increases**
- **Experimenting with cycle-stealing jobs that can be killed when nodes are required for DA or queued jobs**
- **Evaluate these policies with regard to DA strategies**
- **Analyze log data for larger Antarctica clusters**