German Research School
for Simulation Sciences

A Batch System for Adaptive Parallel Programs

Suraj Prabhakaran, Marcel Neumann, Felix Wolf – GRS
Abhishek Gupta, Laxmikant V. Kale - UIUC

JÜLICH
FORSCHUNGSZENTRUM

RWTH AACHEN
UNIVERSITY

Outline

- Motivation
- Objective
- Torque/Maui Overview
- Expand/Shrink in the Torque RMS for Charm++
- Scheduling strategies and results
- Conclusion

Motivation

- Batch systems support only rigid or moldable jobs (static allocation)
- Complex scientific simulations getting more adaptive

Applications Examples

- Multiscale analysis
  - Flow solvers (Quadflow – solves compressible navier stokes equations
  - Grid size may increase, more computations
  - Cannot predict the increase before run
- Adaptive Mesh Refinement
  - Astrophysics
  - Grid size increases or decreases
  - Cannot predict pattern
- Secondary simulations for analysis
  - Weather simulations, brain simulations

Motivation

- Batch systems support only rigid or moldable jobs (static allocation)
- Complex scientific simulations getting more adaptive
- Evolving – application initiates expand/shrink
  - Grow in data size, computations
  - Need more resources to finish on time
- Malleable – batch system initiates expand/shrink
  - Can adapt to changing resource availability

Dynamic Allocation - Benefits

- Unpredictably evolving applications can get resources on-the-fly
- Resources can be released when not needed any more
- Avoids abrupt termination and restart for such programs
- Better resource utilization
  - Use idle resources for evolving or malleable jobs
- Better throughput and response time
- Fault tolerance

Objective - Dynamic Batch System

- An RMS with dynamic allocation/deallocation facilities
- Effective scheduling strategy for evolving and mallable jobs
- Dynamic Torque/Maui batch system
  - Can also be used independently and integrated with other schedulers/RMS

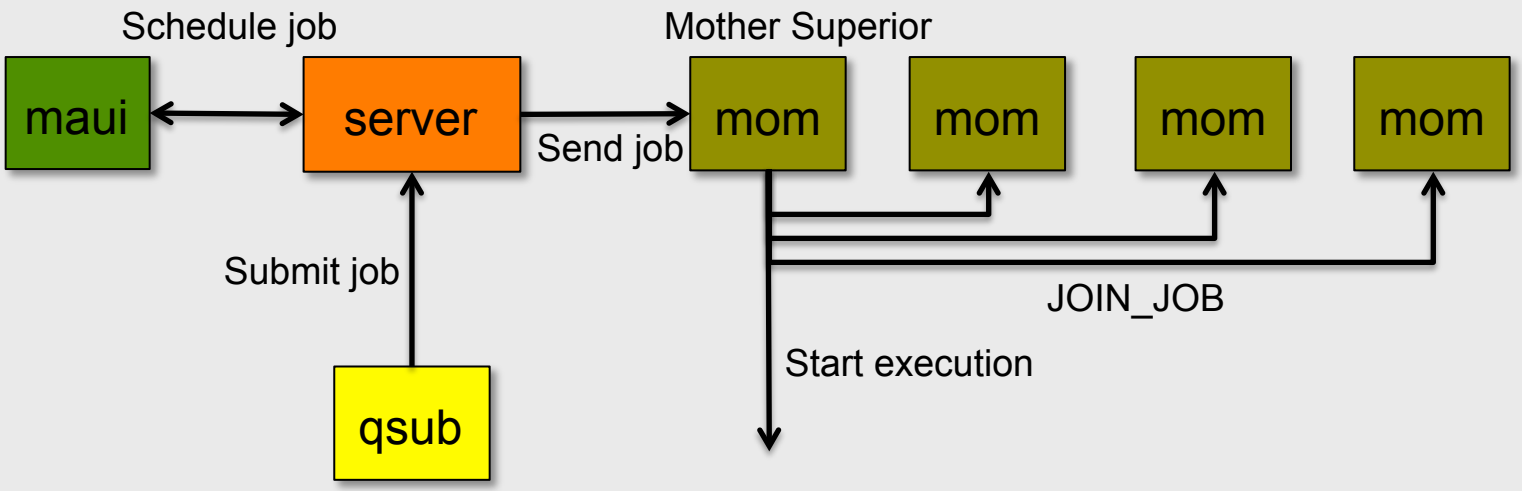
Batch Systems Review

- SLURM
  - Dynamic feature available
  - Get and release a set of nodes
- KOALA
  - Effective malleable scheduling strategy
- OAR
  - Malleable OpenMP and MPI
- CooRMv2
  - Support unpredictably evolving job
  - Scheduling against rigid jobs weak

Objective - Dynamic Batch System

- An RMS with dynamic allocation/deallocation facilities
- Effective scheduling strategy for evolving and mallable jobs
- Dynamic Torque/Maui batch system
  - Can also be used independently and integrated with other schedulers/RMS
- Must compliment the programming model running the application
- Provide generic interfaces for evolving/malleable scenarios
- This work – integrates support for adaptive charm++ jobs

Overview of Torque/Maui



Torque/Maui & Charm++

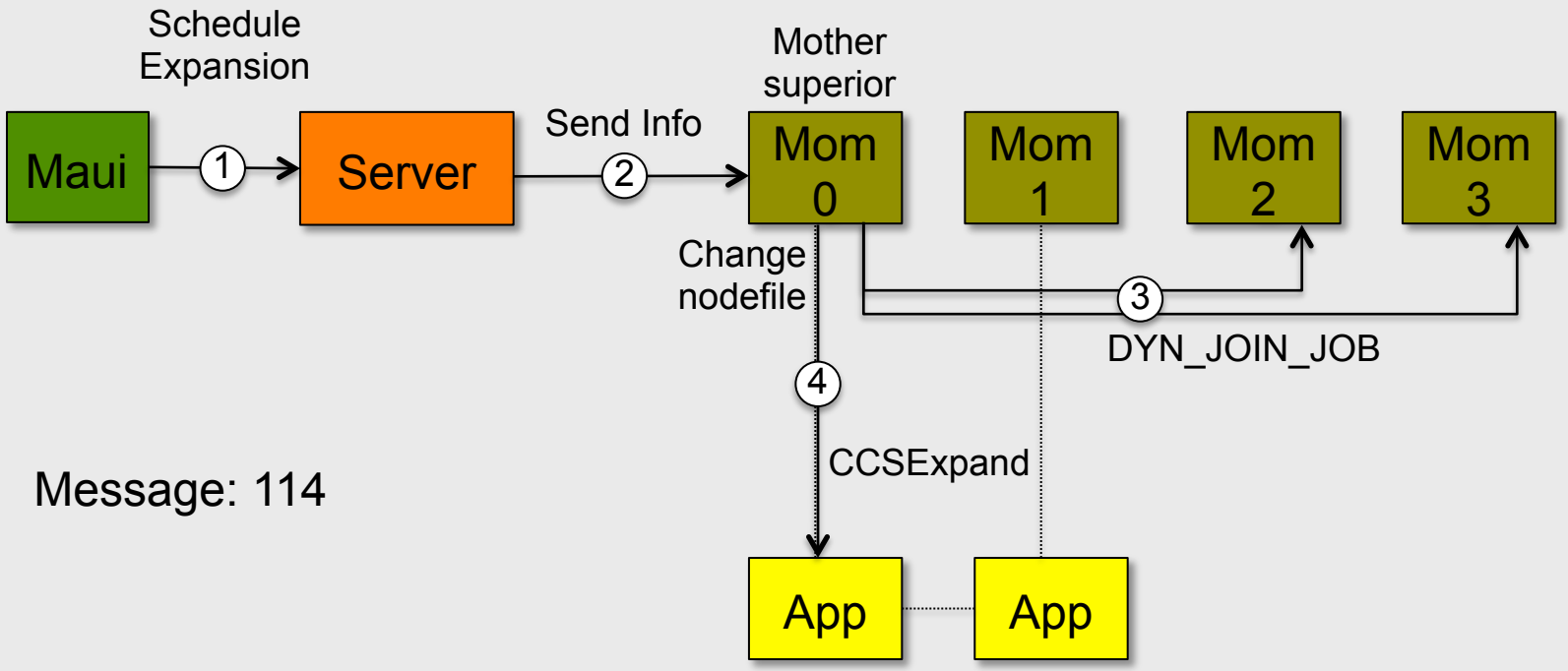
1. User submits a Charm++ job
   qsub –l nodes=x -L min,max jobscript.sh

2. Mother superior creates nodelist in charm++ format under $PBS_CNODEFILE

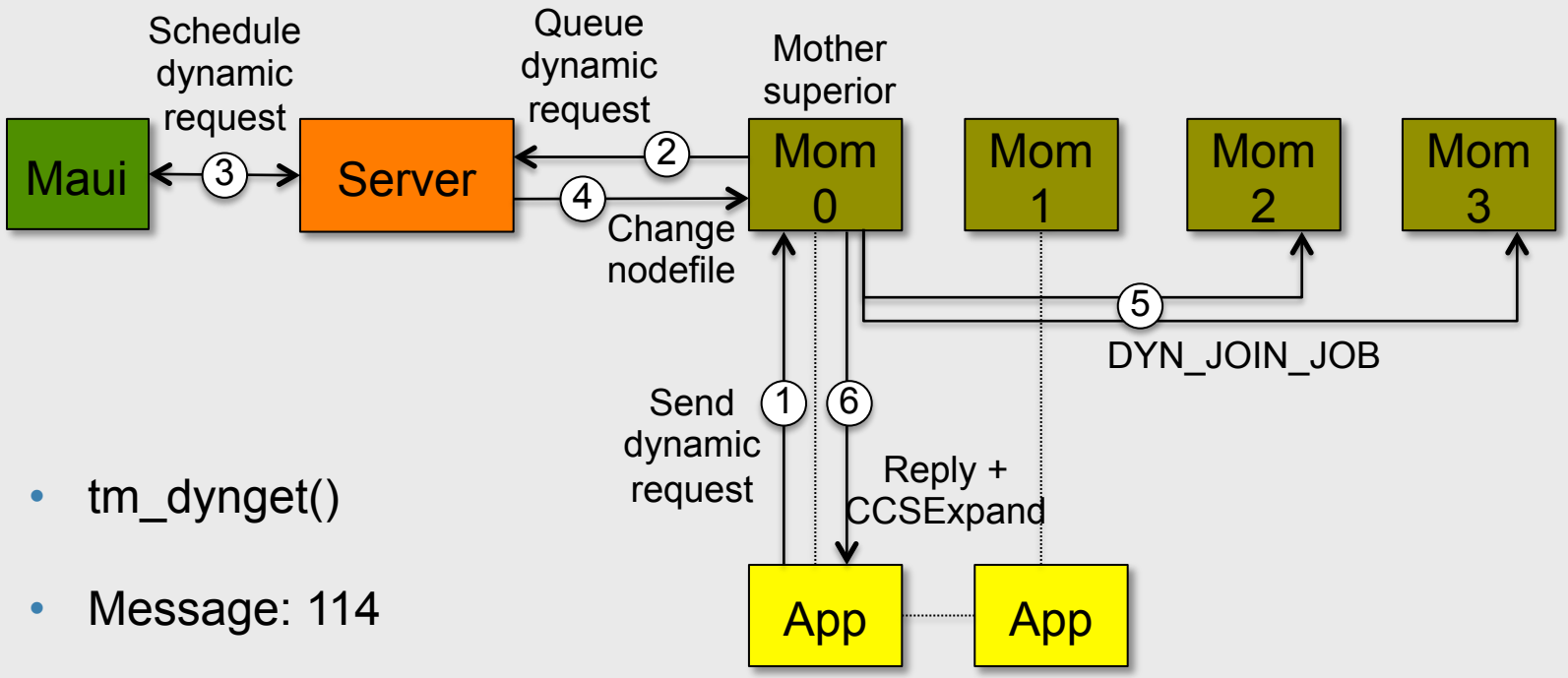
3. Mother superior appends charmrun line in the jobscript before execution.
   charmrun +px ./app ++nodelist $PBS_CNODEFILE ++server ++server-port $portno$

4. Execution starts

Expand - Malleable

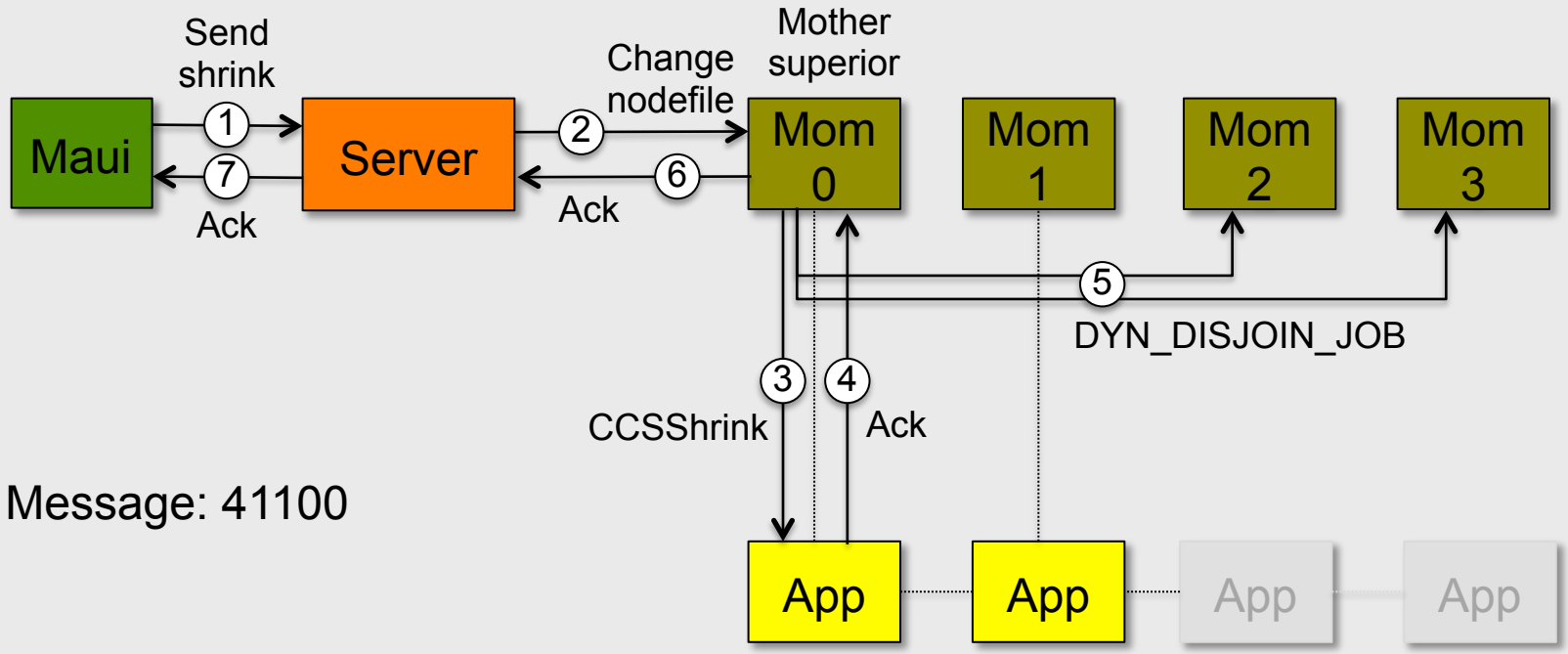


Expand - Evolving



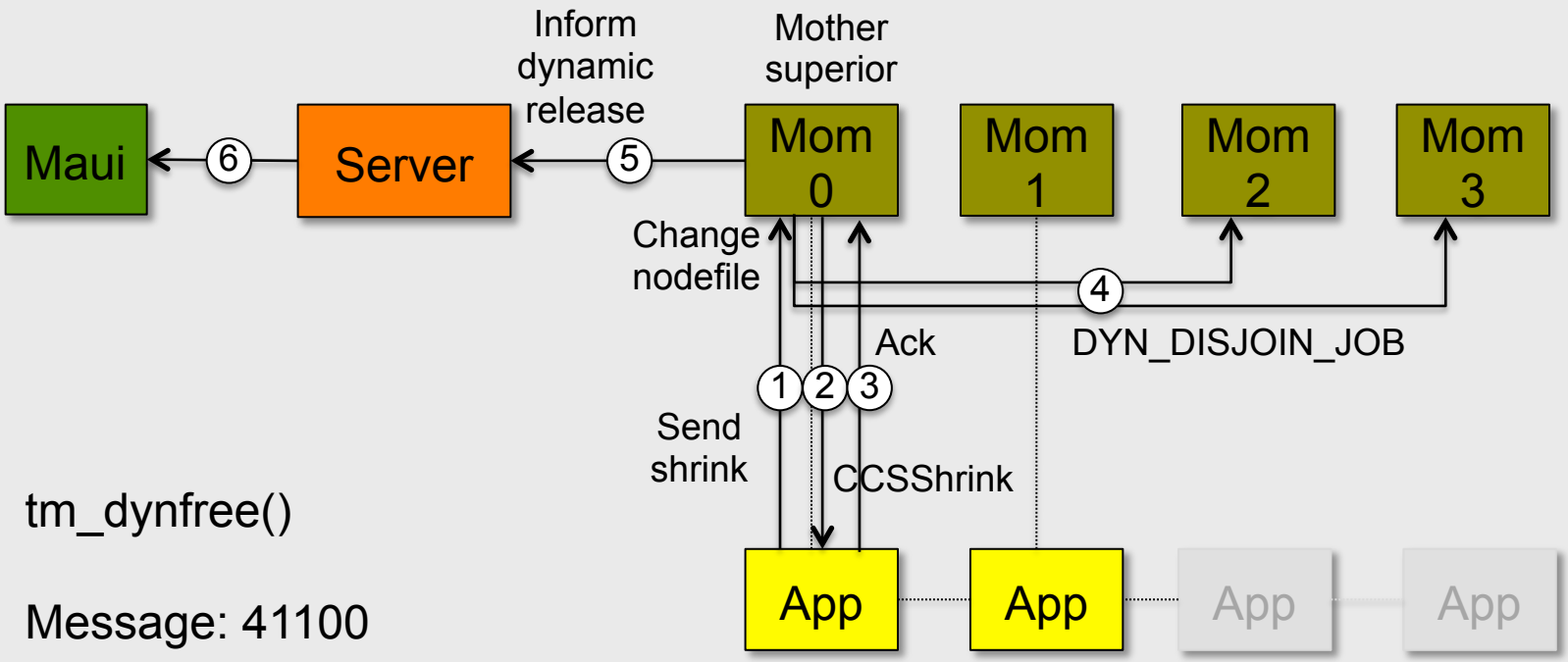
- tm_dynget()
- Message: 114

Shrink - Malleable



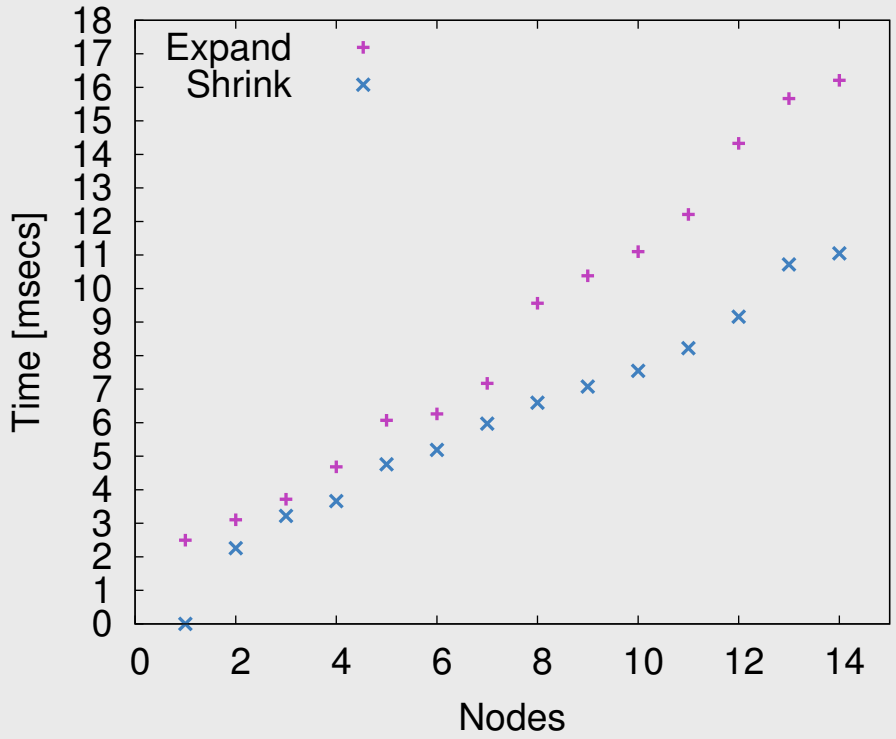
Message: 41100

Shrink - Evolving

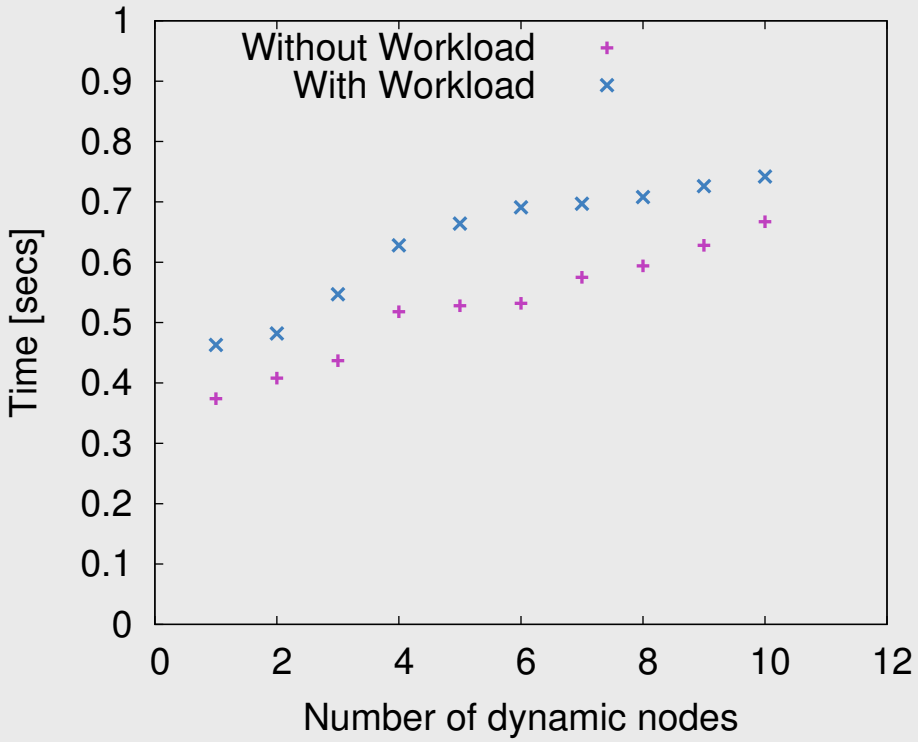


- tm_dynfree()
- Message: 41100

Overhead – Malleable



Overhead - Evolving



Scheduling Evolving Jobs

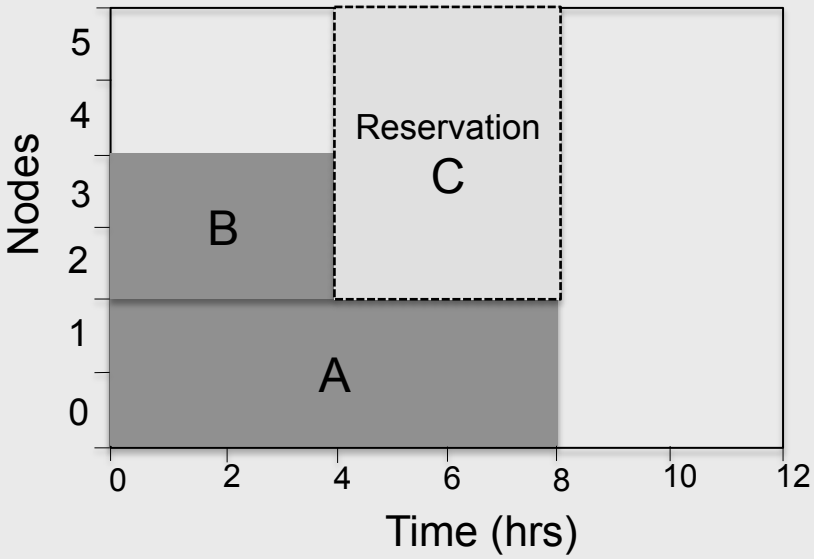
1. Unexpected resource requests from running jobs

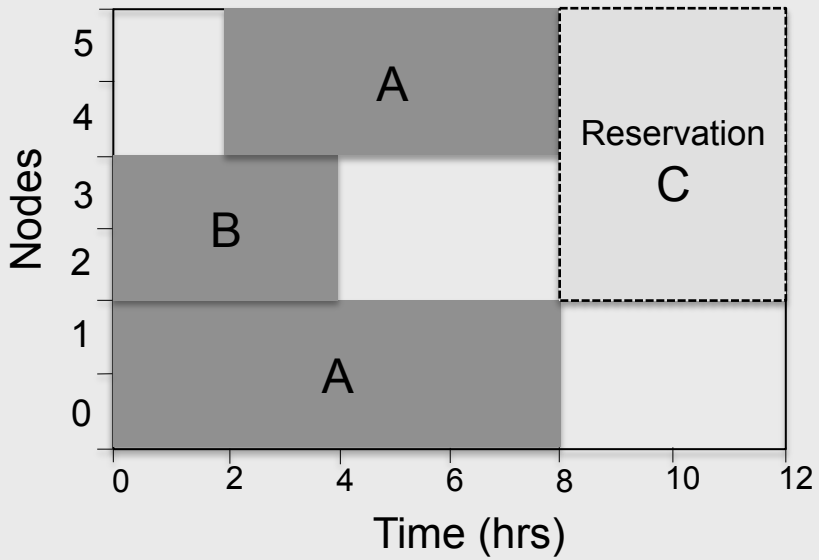
2. Cannot guarantee resources

3. Availability can be increased
   - Idle resources
   - Separate partition for dynamic requests
   - Prempt backfilled jobs
   - Steal from malleable jobs

4. Biggest challenge – Fairness
   - Who to serve? Static or dynamic request?
   - Unfair to specific jobs/users

Scheduling Evolving Jobs (II)





Scheduling Evolving Jobs (III)

- Configurable new Maui parameters control static/dynamic scheduling
- Separate queues of static and dynamic requests
  - Calculate delays caused by dynamic request
  - Satisfy if delay under permissible *limit*
- Limits can be set for users, groups, classes, accounts and QOS

Scheduling Evolving Jobs (IV)

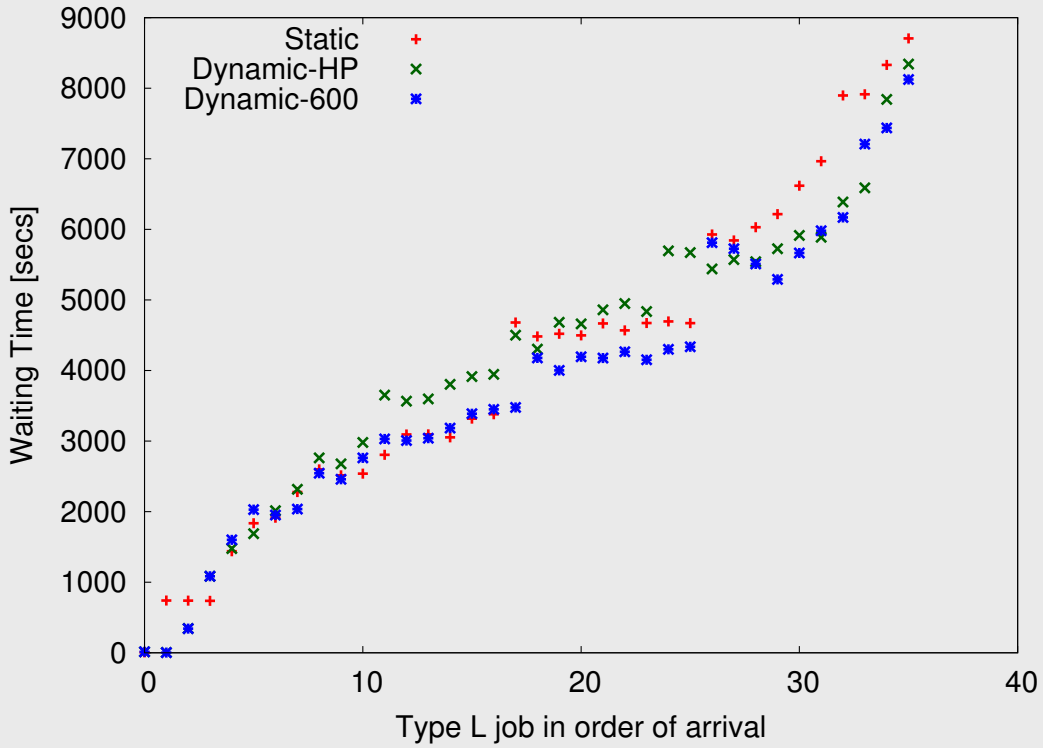
```
DFSPOLICY                DFSSINGLEANDTARGETDELAY
DFSINTERVAL              06:00:00
DFSDECAY                 0.4

USERCFG[user01]          DFSDYNDELAYPERM=1 DFSTARGETDELAYTIME=3600 \
                         DFSSINGLEDELAYTIME=0

USERCFG[user02]          DFSDYNDELAYPERM=0

USERCFG[user03]          DFSDYNDELAYPERM=1  DFSTARGETDELAYTIME=0 \
                         DFSSINGLEDELAYTIME=00:30:00

USERCFG[user04]          DFSDYNDELAYPERM=1 DFSTARGETDELAYTIME=02:00:00 \
                         DFSSINGLEDELAYTIME=00:15:00

GROUPCFG[group05]        DFSTARGETDELAYTIME=04:00:00

GROUPCFG[group06]        DFSDYNDELAYPERM=0
```

Scheduling Evolving Jobs (V)

- Dynamic ESP benchmark – 230 jobs, 14 types
- 30% (69) evolving jobs, 70% (161) rigid jobs
- Dynamic jobs finish faster, linear scaling

Config	Time [mins]	Satisfied Dynamic Jobs	Util %	TP [Jobs/ min]	TP % Inc
Static	265.78	0	77.45	0.86	-
Dyn-HP	238.78	43	85.02	0.96	11.3
Dyn-600	241.06	27	83.57	0.95	10.2

Scheduling Evolving Jobs (VI)



Scheduling Malleable Jobs

1. Most famous – equipartitioning
   - Shrink and start new job
   - When resources available, distribute equally to malleable jobs

2. Can we do better? Main goal: improve throughput
   - Scheduler prediction with min and max walltimes

3. Combined scheduling
   - Backfilling malleable jobs – good for evolving jobs

What HPC Wants

- Users should not care which batch system and programming model
- A standard interface required
- Evolving

```
Batch_get_resources();
Batch_release_resources();
```

- Malleable

```
Batch_query_resources();
```

Conclusion

- Dynamic resource management most needed at this point of time
- Parallel programming paradigms and job management systems should become more tightly coupled
- This work:
  - Enriching Torque/Maui with dynamic (de)allocation facilities
  - Integrating Charm++ and Torque/Maui batch system for evolving and malleable scenarios
- Next steps:
  - Malleable scheduling strategies
  - Towards standardization