A2L2:
an Application Aware Flexible HPC Scheduling Model for Low-Latency Allocation

Gonzalo P. Rodrigo - gonzalo@cs.umu.se
P-O Östberg – p-o@cs.umu.se
Lavanya Ramakrishnan – lramakrishnan@lbl.gov
Erik Elmroth – elmroth@cs.umu.se

Distributed Systems Group – Umeå University, Sweden
Data Science & Technology – Lawrence Berkeley National Lab
Disclaimer

This is a presentation about a position paper.

All the proposals and concepts are presented to encourage discussion.

Further research to evaluate and apply this work is ongoing.
Outline

- Batch schedulers in HPC
- Game change
- Cloud inspiration
- A2L2
- Next steps
- Conclusions
Current HPC batch schedulers

- Static jobs, tightly coupled
- Target: Utilization and short turnaround time
- Homogeneous resources

**Diagram:**
- FCFS
- Backfilling
- Priority
Game changer: Job Heterogeneity

Application characteristics

- Tightly coupled vs. Data intensive
  - Analysis vs. simulation
  - Workflows vs. large jobs
    - vs. stream

Geometry

- Long vs. short jobs
  - Large parallel vs. serial

Different requirements

Gonzalo P. Rodrigo – gonzalo@cs.umu.se
Game changer: Live experiments data processing (stream)

- Live experiment
- Produces data (large amounts)
- Required to be processed on a super computer
- Processed results one day later

- Experiment would benefit of live feedback!
- Reservations are hard to align to reality!
Game changer: Dynamically malleable jobs in HPC

- Data Centered
- Workflows
- No data dependence
- Performance requirements, not resources
- Input can be divided in independent quants

- Resource allocation can change during runtime with small performance penalty.
- Managed execution framework

Programming model allows re-scaling
Game changer: Exascale (steps towards)

Burst buffer scheduling

Another resource to be scheduled
- I/O closer to compute nodes
- Lower I/O latency
- Possible distributed file system
- Possible storage on node

Compute vs. Memory, I/O
Expensive data
Resource Heterogeneity
Current HPC schedulers

- Static jobs
- FCFS
- Backfilling
- Some prioritization schemas
- Need for low latency allocation
- Target: Utilization

- Node storage
- Low latency storage
- Distributed FS?
- Heterogeneity
Are batch schedulers ready for the future (present)?

What can do different and, maybe, better?

- Burst buffer scheduling
- Expensive data
- Exascale
- Compute vs. Memory, I/O
- Data intensive
- Stream processing
- Batch
- Heterogeneity
- Applications
Looking for inspiration… in the clouds.

Cloud infrastructures have faced similar challenges…
Similarities

Cloud

Applications

Heterogeneous Workload

Batch Jobs

Data is Key

Wait Time is important

Heterogeneous Workload

Many non tightly coupled

Non-classical HPC

Response time

SSDs on Nodes

Distributed Filesystems

Heterogeneous resources

Burst Buffer

Accelerator HW

BB nodes

Compute nodes

Infrastructure
Application aware scheduling: Aware of characteristics, performance models, different rules for different types of job.

Dynamically malleable management: runtime re-scaling of jobs, performance based allocation.

Flexible backfilling: for better utilization

Low latency allocation: To allow allocation of jobs a short time after submission (stream job)
Cloud borrowed solution: **Two level scheduling**
One scheduler per application
Smart resource manager: distributes resources, gatekeeper to access resources

Diagram:
- Request 4 nodes
- Ready
- Request 2 nodes
- Ready
- Resource Manager
- Allocate
- N4
- N5
- N6
- N3
- Allocate
- N7
- N8
Research questions

- What is the best model? Share state? Resource offers?
- What is the best way of having policies to control resources allocated to schedulers? Fairshare? Priority of jobs? Preemption?
- Is there a need of location aware resource allocation?
Dynamic allocation of resources: Integrate framework

Control data intensive applications

Change resources during runtime

Adapt the allocation to overall system state

Dynamically Malleable Applications
Scheduler

Control Framework

Resource Manager

Add 3 nodes

Free 1 node

Allocate for Batch
Dynamic allocation of resources: Integrate framework

Research questions
• What is the real performance impact of runtime alteration of resources for the data intensive applications
Flexible backfilling

Temporary “reservation” of resources that could be returned immediately

**Resource Reclamation:**
Borrow and return actions

Request phase
- **Offer** Free+borrowed nodes

Borrow Phase
- Offer Free nodes

Run Job
- **App Leader**
  - N3
  - N4
  - N5
  - N6

Return
- **App 1 Leader**
  - N2
  - N1

Borrow
- 1 node

Allocate for Batch
- N4
- N5

Control Framework

Batch Scheduler

Dynamically Malleable Applications Scheduler

Request
- 2 nodes

Ready
- 2 nodes
Flexible backfilling

Diagram showing the concept of flexible backfilling with tasks J1, J2, and J3. The diagram illustrates the allocation of compute nodes over time, with segments indicating batch and dynamic malleable tasks. Arrows indicate the borrowing and return of compute nodes.
Flexible backfilling

Research questions

• What workload configuration (e.g. %batch jobs vs. %dynamic) is best case and worst case for this technique?
• What do we do with competing dynamic apps? What is best, to accelerate one a lot, or many a little bit?
• What is the real performance penalty?
Resource Expropriation: Low latency allocation

Temporary “expropriation” of resources assigned assigned to dynamically malleable applications

Expropriate and return actions

- Low Latency Scheduler
  - Expropriate 4 nodes
  - Ready
  - App 1 Leader
  - Free 3 nodes
  - Run Job
  - N3
  - N4, N5, N6
  - Stream Job

- Dynamically Malleable Applications Scheduler
  - Expropriate 4 nodes
  - Free 1 node
  - App 1 Leader
  - N1

- Control Framework

Gonzalo P. Rodrigo – gonzalo@cs.umu.se
Resource Expropriation: Low latency allocation
Resource Expropriation: Low latency allocation

Research questions

- What workload configuration (e.g. %batch jobs vs. %dynamic) is best case and worst case for this technique?
- What is the best technique to choose the jobs to steal from? Better a lot from one, better a little from many?
- What is the real performance penalty for the “expropriated jobs”? Do they get to end?
Next Steps

- Model
- Implement
- Emulate

Workload
Resources
Slurm
Enveloping Slurm
Conclusions

Application heterogeneity are a trait of both cloud and HPC applications

- Application Aware
- Two level scheduling

Flexible nature of malleable applications can be useful (and there maybe enough malleable workload to make be useful)

- Application Management
- Better utilization
- Stream job allocation
Thanks
Related work

- Moldable Applications
  - Decision before execution

- Malleable Jobs (MPI)
  - MPI v2 Primitives to resize jobs

- GRID schedulers
  - Dynamic allocation jobs
    - Multilevel scheduling