Scheduling Many-Task Workloads on Supercomputers: Dealing with Trailing Tasks

Timothy Armstrong, Mike Wilde, Daniel Katz, Zhao Zhang, Ian Foster.

Agnieszka Podsiadło
Abstract

Many-task applications

Reducing time
Efficient use of resources
Introduction

Many-task application

• Comprises many independent tasks coupled with explicit I/O dependencies (tasks single-threaded or supporting parallelism within one node)

• Focuses on high-performance
Introduction

Trailing task problem

- Increasing number of workers remain idle
- Tail of some number of tasks continues to execute for some time
Problem description

Introduction

Fixed Node Count
Dynamic Allocation
Tail-chopping
Simulation
Experiment
Results
Discussion
Conclusion

Workers

Tasks
Problem description

Workers

Tasks

Introduction
Fixed Node Count
Dynamic Allocation
Tail-chopping
Simulation
Experiment
Results
Discussion
Conclusion

23/02/12
Presentation
Constraints

• Workers allocated in a way fitting allocation policies
• Tasks scheduled on the available worker
• Tasks are not preemptable
Problem description

Constraints

- Workers allocated in a way fitting allocation policies
- Tasks scheduled on the available worker
- Tasks are not preemptable

Optimization

- Minimizing:
  - Time to solution
  - Utilization $u = \frac{\text{time spent on tasks}}{\text{total allocated time}}$
Algorithms for fixed worker counts

- Fixed number of workers
  - Both goals are equivalent
- NP-hard (bin-packing)
- Simple approaches – queues
  - Random: \((2 - \frac{1}{m}) \times \text{OPT}\)
  - Sorted: \((\frac{4}{3} - \frac{1}{m}) \times \text{OPT}\)
Factors causing long-tail

- Variance in task duration
- Number of tasks not divisible by the number of workers
Dynamic allocation

- Few less workers (works for sorted or short tasks)

- Tail-chopping – after chopping smaller resources are allocated
Tail-chopping assumptions

- Only one partition of processors will be used for the target at a given time
- No time limit – allocation requested for any duration
- Constant time required to start and stop an allocation
- Tasks cannot migrate – to move a task we need to cancel and restart the task
Tail-chopping heuristics

How many workers to allocate?
- minimum task/worker ratio

When to shrink the number of workers?
- maximum fraction of idle workers

![Diagram showing utilization over time with a time threshold for tail-chopping.](image)
Tail-chopping hypothesis

- Tail-chopping will not completely solve the utilization problem
- Hard to achieve high utilization if the minimum allocation is high
- Tail-chopping more beneficial for skewed distribution with much-longer-than-average-task-tail
- Tail-chopping provides greater benefit for not sorted tasks – otherwise reallocating loses a lot of precious work done
- No benefit combined with sorting if \( \text{max}_\text{length} / \text{mean}_\text{length} > \text{task} / \text{worker} \)
• All tasks single threaded
• 12 different numbers of CPU cores
• First measured and then used:
  • Time from request to manager reporting all partitions ready to go
  • Time between requesting to terminate and the allocation finishing
• Control over 3 parameters:
  • Scheduling order
  • Task / worker
  • Fraction of idle workers
Simulation - results

- Tail-chopping improved utilization for many sets of parameters
- Increased time to solution (as expected)
Simulation - results

- Introduction
- Fixed Node Count
- Dynamic Allocation
- Tail-chopping
- Simulation
- Experiment
- Results
- Discussion
- Conclusion
Simulation - results

Introduction

Fixed Node Count

Dynamic Allocation

Tail-chopping

Simulation

Experiment

Results

Discussion

Conclusion
Skewedness of the distribution is crucial for assessing the tail-chopping method.

There is no better and worse fraction idle parameter.
- 0.8 – aiming for quick solution

No further benefit on sorted tasks
Experiment

- With and without tail-chopping
- 15,000 tasks
- Task/worker = 5
- Chopping when 50% of workers are idle
Results

Without tail-chopping

With tail-chopping when 50% workers are idle
Discussion

Major problems:

- Time spent on waiting for new allocation
- Canceling the tasks when reallocated
- Current heuristics are not sophisticated
Time spent on waiting for new allocation

- Smaller processors alongside
- Ability of downsizing the current allocation
Discussion

Canceling the tasks when reallocated

- Ability to migrate tasks
Current heuristics are not sophisticated

- Heuristics using more information
Conclusion

- Described trailing task problem
- Tail-chopping as a promising way to address the problem
- Several directions for further research