St Petersburg
University
www.spbu.ru

HPC workload balancing algorithm for co-scheduling environments.

R. Kuchumov, V. Korkhov

HPC job schedulers

1. HPC schedulers assign tasks from the queue to computing nodes
   a. The whole node may be assigned to a single job, or
   b. Nodes are divided into slots and jobs are distributed among them
   c. Only one job can claim a node or a slot
2. There are multiple resources in a node that can be used by a job
3. Jobs rarely fully utilize all of them

HPC job schedulers

1. Underutilized resources lead to
   a. An increase of queue wait time
   b. Increase of the cost of computations (e.g. in clouds)
2. Jobs with different resource demands (e.g, cpu-, network- and gpu-intensive) can potentially be scheduled on the same node
3. Computational resource oversubscription is not a common practice

Problems in co-scheduling

1. Jobs may compete for the same shared resources, fair resource allocation is not always possible
2. Resource requirements have to be formalized for making decisions on co-scheduling
   a. They should be measurable in practice
   b. They should not depend on the state of the scheduler
   c. Dependencies between resource usage rates should be represented

Assumptions about applications resources usage

We have limited our scope to iterative HPC applications, and that the following statements are valid:

1. Application consists of multiple low-resolution stages, where resource throughputs are periodic or constant
   a. Stationary problem definition is considered for simplicity, dynamic problem is a future work
2. During a single stage the amount of consumed resource does not change regardless of the available bandwidth
   a. Applications does not wait actively on resources (i.e. no user-level polling, e.g. no spin-locks)

Measuring application processing speed

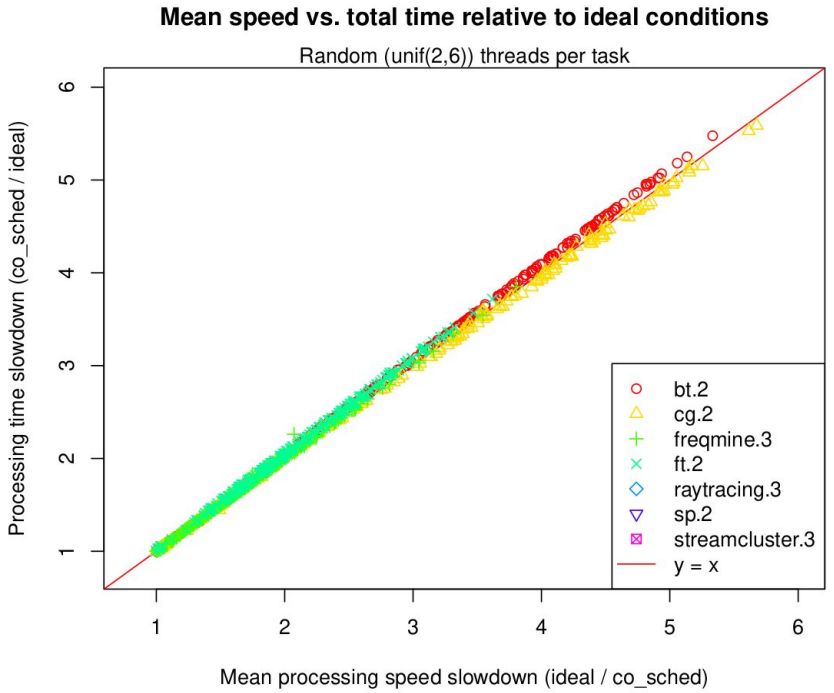
Application processing speed should be:

- invariant of available resources and runtime environment (cpu time, clock freq., memory bus BW, etc)
- measurable in runtime with low overhead

Speed = IPC * cputime_fraction:

$$\nu(t) = \frac{inst(t) - inst(t-\Delta t)}{cycl(t) - cycl(t-\Delta t)} \frac{cpu(t) - cpu(t-\Delta t)}{\Delta t}$$

Correlates with experimental data on NPB and Parsec benchmarks in different environments



Optimal strategy for minimizing makespan

- Makespan optimization problem can be reduced to linear programming problem:

$$\begin{array}{lll} \text{minimize} & \sum_{j=1}^{2^n-1} x_j & \\ \text{subject to} & \sum_{j=1}^{2^n-1} a_{i,j} x_j = b_i, & i = 1, \ldots, n \\ & x_j \geq 0, & j = 1, \ldots, 2^n - 1 \end{array}$$

- $a_{i,j}$ -- acceleration of task T_i in combination S_j
- b_i -- total amount of required work to complete task T_i
- x_j -- processing time of combination S_j
- Optimal schedule can be recovered from LP solution x

Optimal strategy for minimizing makespan

- Matrix A has exponential number of columns (2^n)
  - Additional tasks information and “tricks” can be used to reduce it
- Optimal solution can only be found when all its parameters (A,b) are known or estimated
- Still can be useful in practice in an interval form (when $A_{min} < A < A_{max}$)
- Can be used as a reference for comparing (numerically and analytically) other strategies and objective functions

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{2^n-1} x_j \\ \text{subject to} \quad & \sum_{j=1}^{2^n-1} a_{i,j} x_j = b_i, \quad i = 1, \ldots, n \\ & x_j \geq 0, \quad\quad\quad\quad j = 1, \ldots, 2^n - 1 \end{aligned}$$

- $a_{i,j}$ -- acceleration of task T_i in combination S_j
- b_i -- total amount of required work to complete task T_i
- x_j -- processing time of combination S_j

Practical results from the optimal strategy

We used LP problem to

1. Derive criteria for choosing “naive” strategy of running all jobs in parallel based only on the value of acceleration of max. combination and its size,
2. Showed that optimizing total combination speed at every time point worsens makespan at most by 2 times.

The later result allows to substitute solution of LP problem with heuristic strategy of choosing combination with maximum acceleration sum. Resulting makespan will be worse by at most 2 times compared to the optimal value.

Real-time balancing algorithm

Based on the heuristic strategy, we implemented global-optimization (Bayesian optimization) algorithm for selecting applications for parallel execution.

Implementation uses Cgroup Freezer to preempt running applications. Linux Perf and ProcFS data is used to measure application processing speed in real-time.

Implementation showed it was possible to reduce makespan of a schedule 1.5-2 times (rel. to sequential execution), by allowing 1.5 slowdown of each application.

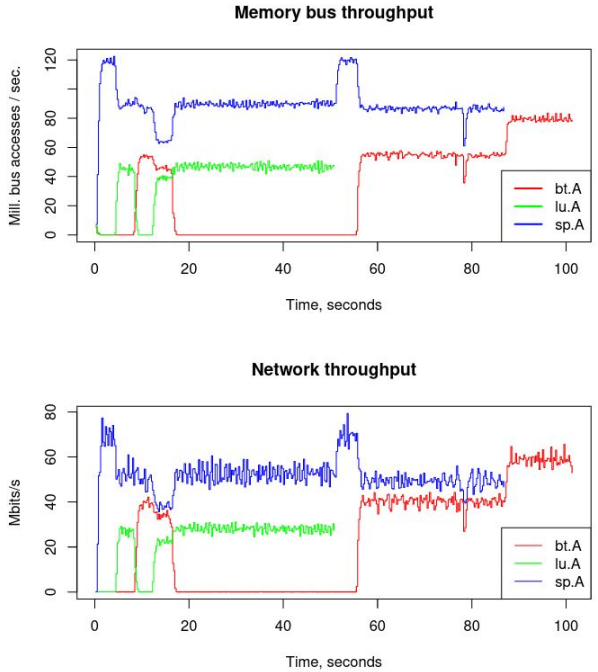


Fig. 4. Co-scheduling algorithm illustration. Plots show how resource throughputs change in time for each application. When throughput values are zero, the application was frozen.

Future work

- Solving the scheduling problem with tasks precedence constraints (task graph).
- Modeling bandwidth vs. throughput dependency
  - By how much can we limit bandwidth without exceeding slowdown threshold?
  - How to control resources BW to affect processing speed and overall schedule performance
- Analysis of historical data to estimate initial conditions

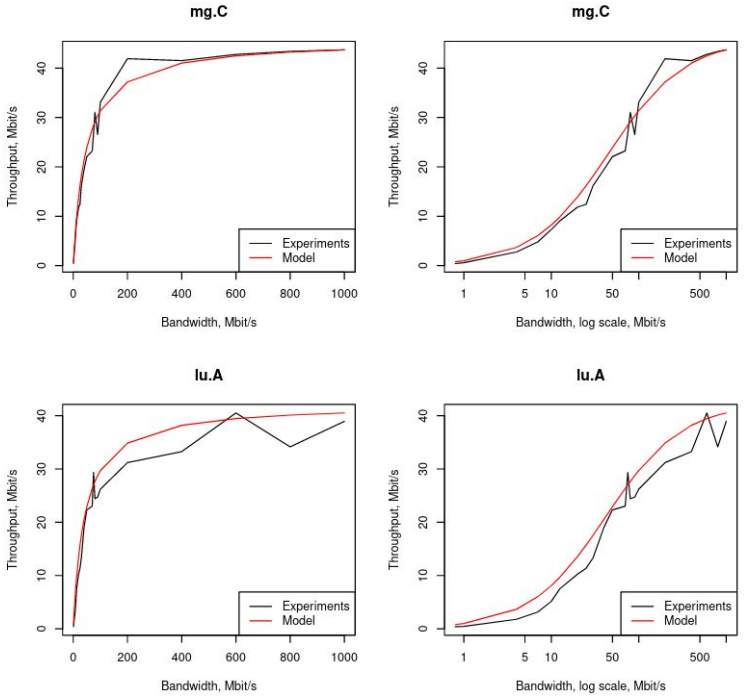


Figure 2: Experimental data and modelled data for network throughput as a function of bandwidth

Thank you for attention!