TetriSched: Space-Time Scheduling for Heterogeneous Datacenters

Alexey Tumanov
Timothy Zhu, Michael Kozuch, Mor Harchol-Balter, Greg Ganger

Carnegie Mellon University

http://www.istc-cc.cmu.edu/
The Future of Datacenters

Consolidation

Specialization

Cluster Resource Scheduler
Explosion of Choices + Tradeoffs

Option 1

Option 2

Option N

GPU
MPI
Availability
Necessary Ingredients

First-class support needed for:

- Server-type preferences (FPGA > GPU > CPU)
- Server-set preferences (same rack <> diff racks)
- Timing preferences (right now == in 5 min > in 30 min)
Solution Space

Proposed Scheduler (TetriSched)

- Always wait
- Never wait
- Consider tradeoffs re: waiting
- Ignore preferences
- Consider tradeoffs re: preferences
- Treat prefs as required

Jockey

alsched  Hadoop  Mesos

HPC  LSF
Benefits of Spatial Flexibility

Treat prefs as required

Consider tradeoffs

GPU

Availability
Benefits of Temporal Flexibility

• Previously: just look at current state
  • give each job the best option now, OR
  • wait indefinitely for the preferred placement

• Plan-ahead: estimate runtimes and choices
  • should this job wait for better placement?
Key Questions

1. How does the user specify placement prefs and associated tradeoffs?

2. How can the scheduler efficiently cope with the combinatorial explosion of placement options?

3. How can the scheduler solve the aggregate, combinatorially complex scheduling problem?
TetriSched System Model

(1) User
Job definition including user objectives

(2) Wizard
Objective Wizard
Scheduler utility function (based on placement/schedule)

(3) Scheduler
MILP objective function
Utility function for MILP converter
MILP solver

Tetrisched Scheduler
Job schedule and placement

Framework
Plugins
MPI
Hadoop

Resources
Time
Utility

• User Objectives
  • response time (completion - arrival)
  • queueing delay (start - arrival)
  • availability (susceptibility to failure)

• Utility – common currency
  • a way of normalizing diverse objectives
  • each objective translated to generic utility
    – what am I willing to pay for …
Utility Example: MPI

- Example: MPI
  - fast duration (2) → higher utility
  - slow duration (4) → lower utility

Diagram:
- Time on the x-axis:
  - 2, 3, 4
- Utility on the y-axis:
  - u, u/2
- Desired utility curve from time 2 to 3 is shown.

Alexey Tumanov © Sep 2014
System Model

User

Objective Wizard

Scheduler

(2) Wizard

Tetriscched Scheduler

MILP objective function

and constraints

Utility function to MILP converter

MILP solver

Job schedule and placement

Resources

Time
Objective Wizard

- Input: high-level user objectives
- Output: scheduler-centric utility functions
  - encoded in an algebraic expression language
- Translates user inputs to internal form
  - the form needed for the sched. optimization
Scheduler Expression Language

• Utility $u(p,t)$: placement $p@t \rightarrow utility\ u$

• "n Choose k" (nCk)
  n $\rightarrow$ refers to a group of nodes to choose from
  k $\rightarrow$ how many nodes to choose

$nCk$ formula:

$nCk = \binom{n}{k} = \frac{n!}{k!(n-k)!}$

- For $k = 2, s = 1, d = 2$, $u = \frac{3}{2}$
- For $k = 2, s = 0, d = 4$, $u = \frac{1}{2}$

$\epsilon m_i \in \text{rack1}, \epsilon \cup m_i$
Scheduler Expression Composition

- Utility $u(p,t)$: placement $p@t \rightarrow$ utility $u$

\[ nCk \left( m_i \in \text{rack1}, k=2, s=1, d=2, u \right) \]

\[ nCk \left( m_i \in \text{rack2}, k=2, s=1, d=2, u \right) \]

\[ nCk \left( \bigcup m_i, k=2, s=0, d=4, u/2 \right) \]

max

OR
System Model

User

Objective Wizard
- Scheduler utility function (based on placement/schedule)
- Framework Plugins
  - MPI
  - Hadoop

Wizard

Tetrisched Scheduler

(3) Scheduler
- MILP objective function and constraints
- Utility function to MILP converter
- MILP solver

Job schedule and placement

Resources

Time
Experimental Results

- Real results from YARN-based TetriSched prototype on 73-node cluster
  - with soft constraints, plan-ahead, and gang-scheduling, TetriSched outperforms YARN
  - details on poster

- Simulation results for a 1000-node cluster
  - parameter sweeps for load, burstiness, plan-ahead, slowdown
  - variable workload compositions
  - sensitivity to job runtime mis-estimation
Benefits of Flexible Placement

Knowing placement constraints always helps
Placement flexibility enables better tradeoffs
Benefits of Plan-ahead

Plan ahead: know when to wait
When Users Err

TetriSched tolerates runtime estimate errors

Percent error in runtime estimates

TetriSched tolerates runtime estimate errors
Key Takeaways

• Problem: current schedulers don’t cope with
  • increased heterogeneity in datacenters
  • explosion of tradeoffs and choices

• Solution: TetriSched explicitly enables
  • spatial flexibility aware scheduling (soft constraints)
  • temporal flexibility aware scheduling (plan-ahead)

• End result:
  • better schedules of heterogeneous mixes
  • easier adoption of specialized hardware
Related Work References