Realizing value in shared compute infrastructures

Andrew Chung

Doctoral thesis defense

Computer Science Department

Nov. 18th, 2022
Talk outline

• Shared cluster environments + thesis statement
• 2 case studies: specializing application frameworks
• 2 case studies: from perspective of cluster operators
• Conclusion
Talk outline

- Shared cluster environments + thesis statement
- 2 case studies: specializing application frameworks
- 2 case studies: from perspective of cluster operators
- Conclusion
Shared cluster environments

• Highly heterogeneous resources and applications
  • Many users from various groups and organizations
  • Time varying load

• Examples of shared cluster environments:
  • Public clouds (AWS, Azure, GCE)
  • Private clouds (MS Cosmos, Google Borg)
Example: Shared cluster environment

Compute

Org 1

Org 2

Org 3

Running applications

Storage

W_1

W_2

W_3

R_2

R_3

Org 1, Org 2, and Org 3 connect to the shared cluster environment, which runs applications and stores data.

Storage resources (W_1, W_2, W_3) connect to the compute resources (R_2, R_3) through the shared cluster.

http://www.pdl.cmu.edu/
User goals in shared clusters

- **Users**: Run applications in shared environment
  - Goal 1: Meet application business requirements
  - Goal 2: Minimize cost of meeting requirements
- **Challenges**:
  - Resource heterogeneity
  - Wide variety of pricing mechanisms
Cluster operator goals in shared clusters

- **Cluster operators:** Maximize profit & satisfy users
  - Goal 1: Prioritize resource allocation to applications
    - Using some notion of “user value”
  - Goal 2: Maximize “profit” = “value” achieved - costs
- Challenges:
  - Resource heterogeneity and availability
  - Hidden user values and performance requirements
  - Cluster capacity + cost management
Value-realized in shared data environments can be improved both by value- and dependency-aware resource management systems from cluster operators and by cost- and heterogeneity-aware applications from users.
Talk outline

• Shared cluster environments + thesis statement
• 2 case studies: specializing application frameworks
• 2 case studies: from perspective of cluster operators
• Conclusion
Application-specific resource acquisition: Case studies

1. Elastic web services
   • Spot-dancing for elastic services with latency SLOs
   • Tributary [USENIX ATC 2018]

2. General containerized batch task scheduling
   • Cost-aware container scheduling in the public cloud
   • Stratus [ACM SoCC 2018]
     – Best student paper award
More background: Public clouds

- Public clouds offer a variety of resources
  - e.g., varying compute capacity, storage, HW accelerators
- Under different types of contracts
  - e.g., reliable, transient, and burst
- Difficult for users to choose resources cost-effectively!

Achieve user value through:
Application-specific, cost-aware resource acquisition
Transient/spot instances in AWS

Adv: Often > 50% cheaper vs on-demand, refund if revoked in 1st hr

$2.00

$1.00

$0.00

M4.10xlarge in us-east-1b

10/01 10/16 11/01 11/16 12/01
Transient/spot instances in AWS

**Adv:** Often > 50% cheaper vs on-demand, *refund if revoked in 1\textsuperscript{st} hr*

<table>
<thead>
<tr>
<th>Date</th>
<th>Bid price</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/01</td>
<td>$2.00</td>
</tr>
<tr>
<td>10/16</td>
<td>$1.00</td>
</tr>
<tr>
<td>11/01</td>
<td>$1.00</td>
</tr>
<tr>
<td>11/16</td>
<td>$0.00</td>
</tr>
<tr>
<td>12/01</td>
<td>$0.00</td>
</tr>
</tbody>
</table>

M4.10xlarge in us-east-1b
Transient/spot instances in AWS

**Adv:** Often > 50% cheaper vs on-demand, *refund if revoked in 1st hr*

![Graph showing bid price and market price over time](image-url)

- **M4.10xlarge in us-east-1b**
- **Bid price**
- **Market price**
Transient/spot instances in AWS

**Adv:** Often > 50% cheaper vs on-demand, *refund if revoked in 1st hr*

**Graph:**
- **Bid price**
- **Market price**
- **Revocation**

M4.10xlarge in us-east-1b

http://www.pdl.cmu.edu/
Transient/spot instances in AWS

**Adv:** Often > 50% cheaper vs on-demand, *refund if revoked in 1st hr*

![Graph showing on-demand price, bid price, and market price over time with revocation point.](http://www.pdl.cmu.edu/)

- **On-demand price**: $2.00
- **Bid price**: $1.00 (dash line)
- **Market price**: $0.00 to $2.00
- **Revocation point** on 11/16

---

M4.10xlarge in us-east-1b
Transient/spot instances in AWS

**Adv:** Often > 50% cheaper vs on-demand, *refund if revoked in 1st hr*

<table>
<thead>
<tr>
<th>Date</th>
<th>On-demand price</th>
<th>Bid price</th>
<th>Market price</th>
<th>Revocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/01</td>
<td>$2.00</td>
<td>$1.00</td>
<td>$0.00</td>
<td></td>
</tr>
<tr>
<td>10/16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

M4.10xlarge in us-east-1b

---

Carnegie Mellon
Parallel Data Laboratory

http://www.pdl.cmu.edu/
Transient/spot instances in AWS

**Adv:** Often > 50% cheaper vs on-demand, *refund if revoked in 1st hr*

![Graph showing on-demand and bid prices for M4.10xlarge instances across different availability zones from 10/01 to 12/16.](http://www.pdl.cmu.edu/)
Application-specific resource acquisition: Case studies

1. Elastic web services
   • Spot-dancing for elastic services with latency SLOs
   • Tributary [USENIX ATC 2018]

2. General containerized batch task scheduling
   • Cost-aware container scheduling in the public cloud
   • Stratus [ACM SoCC 2018]
     – Best student paper award
Elastic web services & spot instances

• Elastic web services
  • Manage a pool of VMs to serve client requests
  • Need to meet latency SLOs (e.g., request within X ms)
  • Stateless services (Tributary’s focus) allow quick scaling

• Spot instances *cheaper but riskier* than on-demand:
  • Instances can be revoked, leading to missed SLOs

Tributary embraces risk associated w/ spot instances to achieve lower cost while meeting SLOs
Exploiting spot resources

- Naïve selection of spot → bulk revocations
  - Large alloc of low cost → low # of VMs left if price spikes
- Observation: Spot market prices not too correlated
Tributary strategy

- Selects resources from multiple spot markets
  - Exploit pricing low or non-correlation
- Uses different bids within the same spot market
  - Higher/lower bid → less risk/more partial-hours
- ML-based prob model → extra resources acquired
  - Added benefit: soaks up unexpected spikes in requests
- Expected cost w.r.t. SLO
  - Cost offset by lower cost VMs and free partial hours
Tributary experimental setup

• 4 real world internet traces
  • Show Clarknet

• Compare vs 3 systems
  • AWS AutoScale shown

• AWS AutoScale:
  • Acquires lowest cost
  • Bid on-demand price
Tributary experimental results

• Tributary 40% lower cost, 60% less reqs violating SLO
Tributary experimental results

- Tributary 40% lower cost, 60% less reqs violating SLO
- AutoScale costs 60% more vs Tributary to match SLO attained
Tributary takeaway

• Diversified resource pools mitigate revocation risk
  • Prob model $\rightarrow$ diverse + extra resources $\rightarrow$ SLO attained
  • Considering expected cost + partial-hours $\rightarrow$ lower cost
• Reduces cost vs compared systems
Application-specific resource acquisition: Case studies

1. Elastic web services
   - Spot-dancing for elastic services with latency SLOs
   - Tributary [USENIX ATC 2018]

2. General containerized batch task scheduling
   - Cost-aware container scheduling in the public cloud
   - Stratus [ACM SoCC 2018]
     – Best student paper award
Background and motivation

- Virtual cluster (VC) scheduling:
  - Schedule containerized batch tasks on rented VMs
  - Different from traditional cluster scheduling:
    - Add/remove VMs any time → dynamically sized
    - VC can be highly heterogeneous

Diverse offerings + VC elasticity to lower cost of executing batch workloads
Stratus

• Stratus: Sched middleware that sizes VC + place tasks
• Goal: Lower cost of executing batch workloads
• Key: Wasted resource-time is wasted money
  • VMs should be highly utilized while rented
  • Use cost-efficient resources

Runtime binning: Pack tasks of similar runtime on to VM
Aligning runtimes: Runtime binning

- Runtime bins: *Logical* groups of tasks and VMs
- Idea: Tasks w/ similar predicted run times on same VM
  - Pluggable task run time predictor
  - VM highly utilized while rented → high tasks per dollar
Aligning runtimes: Runtime binning

- Runtime bins: *Logical* groups of tasks and VMs
- Idea: Tasks w/ similar predicted run times on same VM
  - Pluggable task run time predictor
  - VM highly utilized while rented → high tasks per dollar
Aligning runtimes: Runtime binning

- Runtime bins: *Logical* groups of tasks and VMs
- Idea: Tasks w/ similar predicted run times on same VM
  - Pluggable task run time predictor
  - VM highly utilized while rented → high tasks per dollar

**Bin colors**

1 2 4 8 16

Task A Task B Task C
Experimental setup

- Simulation-based experiments
  - Google and Two-Sigma cluster traces
- Focus on batch analytics jobs
- Spot market traces for dynamically priced VMs
  - Always bid on-demand price – little to no preemptions
- Compare against Fleet: SpotFleet + ECS (AWS)
  - SpotFleet: Scaling based on policies
  - ECS: Packing containers on to VMs
Stratus vs Fleet

- Fleet: SpotFleet + ECS (Amazon offerings)
- Stratus reduces cost by 17% (Google) and 22% (TwoSigma)
Stratus takeaway

• Runtime binning → high VM utilization during rental

• Simultaneous consideration of scaling, packing, and cost-per-resource leads to reduced cost
Talk outline

• Shared cluster environments + thesis statement
• 2 case studies: specializing application frameworks
• 2 case studies: from perspective of cluster operators
• Conclusion
Cluster-operator resource management: Case studies

1. Scheduling to increase attained utility in cluster
   • Unearthing inter-job dependencies for better scheduling
   • Wing [USENIX OSDI 2020]

2. Load-shifting to reduce cluster operation costs
   • Reducing costs with dependency-informed load-shifting
   • Talon [Submission-prep]
Background: Cosmos

- Microsoft’s internal data analytics platform
- Multiple multi-tenant clusters
  - Tens of thousands of nodes each
  - Shared by many teams and orgs
  - Primarily SCOPE jobs
    - Batch analytics jobs similar to Spark/MapReduce
    - 80% resource-time
Background: Inter-job dependencies

- Inter-job dependencies:
  - Occur when job dep on output of earlier job as input
  - Pervade shared envs, but ignored in resource mgmt
- GDPR enables inter-job dependency analysis
  - Untapped opportunities
  - Wing (discussed later) first to analyze in large cluster
    - Forms basis of next two case studies
Shared data environments

Org 1

Org 2

Org 3

Scheduler

Compute

Running jobs

Shared data env

Andrew Chung © November 22
http://www.pdl.cmu.edu/
Shared data environments

Shared data

Org 1

Org 2

Org 3

Scheduler

Compute

Running jobs

Shared data env

Andrew Chung © November 22
Shared data environments

- Org 1
- Org 2
- Org 3

Scheduler

Compute

- A
- B
- C

Running jobs

Shared data env

- A depends on B
- B depends on C

Shared data

W₁

W₂

W₃

C depends on B depends on A

http://www.pdl.cmu.edu/
Data from a Cosmos cluster

- 40k+ daily jobs
- 50k+ servers
- 100s of hierarchical queues (teams)
- Millions of daily tasks
- TBs of job + data
- prov logs daily
Data from a Cosmos cluster

- 100s of hierarchical queues (teams)
- 40k+ daily jobs
- 50k+ servers
- 160k+ daily inter-job dependencies
- 95% of queues inter-dependent
- Millions of daily tasks
- TBs of job + data prov logs daily
- 68% jobs recurring
- 80% jobs depend on other jobs

Andrew Chung © November 22
Cluster-operator resource scheduling: Case studies

1. Scheduling to increase attained utility in cluster
   • Unearthing inter-job dependencies for better scheduling
   • Wing [USENIX OSDI 2020]

2. Load-shifting to reduce cluster operation costs
   • Reducing costs with dependency-informed load-shifting
   • Talon [Submission-prep]
Wing summary

Shared cluster

Jobs

Scheduler
Wing summary

Wing: Scheduling for value with inter-job dependencies

Value

Default  w/ Wing-guidance

~1.6x
Problems when not considering deps

Inter-job dependencies pervade data envs, but are ignored in resource management
Problems when not considering deps

Inter-job dependencies pervade data envs, but are ignored in resource management. Missed deadlines, wasted resources, and untapped opportunities.

We can fix this, with recurring and predictable inter-job dependencies.
Towards addressing inter-job deps

**Wing**

Discovers + analyzes inter-job dependencies from data provenance

**Scheduling with Wing guidance**

Scheduling that prioritizes the most value-impactful jobs, informed with historical recurring inter-job dependencies

---

[Diagram showing the flow of data from Job logs, through Wing pipeline, to Results, which then guides the Shared cluster's Jobs and Scheduler.]
Job value & inter-job dependencies

- Failing/finishing jobs late can impact downstream jobs
- Wing analyzes the aggregate value (impact) of jobs

Value

Agg. value(A)
Wing-Agg: Wing-guided scheduling

- **Goal of value scheduling**: Achieve most value given workload

- **Wing-Agg**: YARN’s prio-based sched + Wing-guidance
  - Prioritize recurring jobs with high aggregate value efficiency

![Diagram of Wing pipeline and Cosmos/YARN Scheduler](https://www.pdl.cmu.edu/)

[Andrew Chung © November 22](http://www.pdl.cmu.edu/)
Experimental setup

- Trace-driven simulations on real cluster traces
  - Preserves inter-job dependencies and properties
- Goal: Attain more value from the same workload
  - Value metric: Total file output downloads attained
- Experiments at various cluster sizes (capacities)
  - To simulate resource-constrained clusters
Value-attainment

- **Wing-Agg**: Prio as historical \textit{agg} value / \textit{agg} compute

% Cosmos cluster capacity

Normalized value attained

- 1
- 0.8
- 0.6
- 0.4
- 0.2
- 0

100%  80%  60%  40%  20%
Value-attainment

- **Wing-Agg**: Prio as historical `agg` value / `agg` compute

![Graph showing normalized value attained vs % Cosmos cluster capacity]

- **Normalized value attained**
- **% Cosmos cluster capacity**

- **Default YARN**
- **Wing-Agg**
Wing takeaways

• Inter-job dependencies prevalent in real clusters
  • But, can be predictable with recurrence
• Inter-job dependencies need to be addressed
  • To ensure jobs meet their deadlines, reduce resource wastage, and improve value attained in shared clusters
Cluster-operator resource scheduling: Case studies

1. Scheduling to increase attained utility in cluster
   • Unearthing inter-job dependencies for better scheduling
   • Wing [USENIX OSDI 2020]

2. Load-shifting to reduce cluster operation costs
   • Reducing costs with dependency-informed load-shifting
   • Talon [Submission-prep]
Talon summary

- Talon: Workflow mgr that reduces cluster op cost by reducing expensive locked-in reserved capacity
  1. Load-shift workload off-peak using inter-job deps
  2. Exploiting low-cost *transient resources*
     - Reduce preemption impact w/ load-shifting

Reduces reserved resources by 38% with minimal deadline violations
Background

• Resource types common in shared clusters:
  • Reserved: Long-term committed
    – Expensive and inflexible (locked-in long-term)
    – On-prem/reserved instances/guaranteed cap in cluster
  • Transient: Low-priority, intermittently-available
    – Lower cost, no lock in, but preemption/revocation risk
    – Spot instances/opportunistic cap in cluster

• Load-shift jobs: Change when job is run
Cosmos workload capacity planning

Capacity planning Cosmos workload peak

![Graph showing normalized resources over days with peaks and valleys indicating workload fluctuations.](http://www.pdl.cmu.edu/)
Cosmos workload capacity planning

Capacity planning Cosmos workload peak

If only reserved, need this much cap (traditional approach)
Cosmos workload + load-shifting

Capacity planning Cosmos workload peak

Scenario:
yellow resource-time can be load-shifted off-peak
Cosmos workload + load-shifting

Capacity planning Cosmos workload peak

Scenario: yellow resource-time can be load-shifted off-peak
Cosmos workload + exploit transient

Capacity planning Cosmos workload peak

Scenario:
Low-cost transient resources available
Cosmos workload + exploit transient

Capacity planning Cosmos workload peak

Scenario:
Low-cost transient resources available
Talon: Reducing reserved lock-in

Capacity planning Cosmos workload peak

Talon:
(1) Reduce reserved resource cap + cost w/ load-shifting and transient resources
(2) Do so without more deadline violations
Two ways to reduce reserved peak

1. Inter-job dependency-based load-shifting

2. Use transient resources
Two modes of load-shifting

1. Delay: Run a job later, try not to violate job DL
   - Output + run time preds both need to be accurate
   - Little benefit (10% resource-time > 1 hr) + risk
   - Talon does not delay jobs

2. Advance: Run job earlier
   - Traditionally difficult, Talon uses inter-job deps
Advancing jobs: Opportunity analysis

- Job eligible to be advanced if:
  - All inputs ready and available
  - *Recurring* + *predictable* based on done jobs
  - *Predict* recurring job arrival if dep on + follows completed upstream job w/ high prob
    - e.g., Job B dep on Job A > 90%
  - Work with other WF Mgrs for advanceability

~24% resource-time advanceable > 1 hour
Two ways to reduce reserved peak

1. Inter-job dependency-based load-shifting

2. Use transient resources
Transient resource risks

• Want to: Use transient resources to reduce reserved
• Risks: Intermittent availability, (bulk) preemption
  • Task replication can help w/ preemptions and DL violations, but
  • Aggressive usage → retries → queueing & more DL violations

“Storm” of retries
Scheduling policy: Admission + placement

• Jobs eligible to start arrive at scheduling policy

• Policy admit + place jobs on reserved/transient:
  • Based on run time, time load-shifted, resources, etc
  • ex 1: Queue adv’d if low resource avail to reduce reserved
  • ex 2: Urgent jobs run reliably (reserved or transient + reps)

• Key to min DL violations: handling (bulk) preemptions:
  • Do not use transient too aggressively
  • Adv’d jobs w/ long slack can run transient w/o replicas
Experimental setup

- Simulation experiments on Cosmos traces
- Transient resources: Scaled Harvest (Spot) VM traces
- Jobs wait for inputs to start
  - Different from in Wing, where jobs may fail if missing input
- Deadline: Time of first non-job output usage
- Compared approaches:
  - Traditional: Peak-provisioned, reserved only
  - GHDP: GreenHadoop, a green-energy scheduler
  - GHDP-R: Replicas on transient to reduce violations
Experimental results

- GHDP (no rep) experience DL violations due to retries
- Talon 38% reduction vs Traditional
- Talon achieves lowest num of deadline violations
Talon takeaways

• Inter-job dependencies critical to exploit load-shifting
  • 24% job resource-time can be advanced by > 1 hr
• Talon can effectively reduce reserved committed capacity using combination of load-shifting + transient resources
  • Up to 38% reserved capacity reduction vs traditional
  • Lowest # of deadline violations under diff scenarios
Talk outline

• Shared cluster environments
• 2 case studies: specializing application frameworks
• 2 case studies: from perspective of cluster operators
• Conclusion
Value-realized in shared data environments can be improved both by value- and dependency-aware resource management systems from cluster operators and by cost- and heterogeneity-aware applications from users.
Thesis contributions: App-specific resource acquisition

1. Elastic web services
   • Spot-dancing for elastic services with latency SLOs
   • Tributary [USENIX ATC 2018]

2. General containerized batch task scheduling
   • Cost-aware container scheduling in the public cloud
   • Stratus [ACM SoCC 2018]
     – Best student paper award
Thesis contributions: Cluster operator resource mgmt

1. Scheduling to increase attained utility in cluster
   • Unearthing inter-job dependencies for better scheduling
   • Wing [USENIX OSDI 2020]

2. Load-shifting to reduce cluster operation costs
   • Reducing costs with dependency-informed load-shifting
   • Talon [Submission-prep]