CAP^3 : A Cloud Auto-Provisioning Framework for Parallel Processing Using On-demand and Spot Instances

He Huang, Liqiang Wang
University of Wyoming

Byungchul Tak, Long Wang, Chunqiang Tang
IBM T.J. Watson Research Center

2013 SEA conference, April 2, 2013, Boulder, CO

Background: HPC and Cloud

- Parallel scientific applications run on dedicated high performance computing (HPC) clusters.
- Infrastructure-as-a-service (IaaS) Cloud such as Amazon EC2 makes it possible to run HPC applications in a pay-as-you-go fashion.
- Amazon EC2 users can construct a cluster by applying a group of instances (e.g., small, medium or large). Amazon also has Cluster Compute and Cluster GPU (higher computation capability, bandwidth and IO) dedicated for HPC.

Why virtual cluster in cloud?

Cons: virtual cluster in cloud cannot compete performance with traditional HPC cluster: lower CPU, IO, bandwidth and virtualization overhead.

Pros:

- Construct a virtual cluster at any time by renting virtual machines (VMs) from the Cloud provider.
- No need to wait for jobs to execute in the queue.
- A dictated cluster with customized software environment.
- Performance is not bad for loosely coupled applications.
- An alternative for a small scale cluster in department.
- Good for development and debugging purpose.
- Easily increase or decrease virtual cluster size on demand.

Background: Cloud

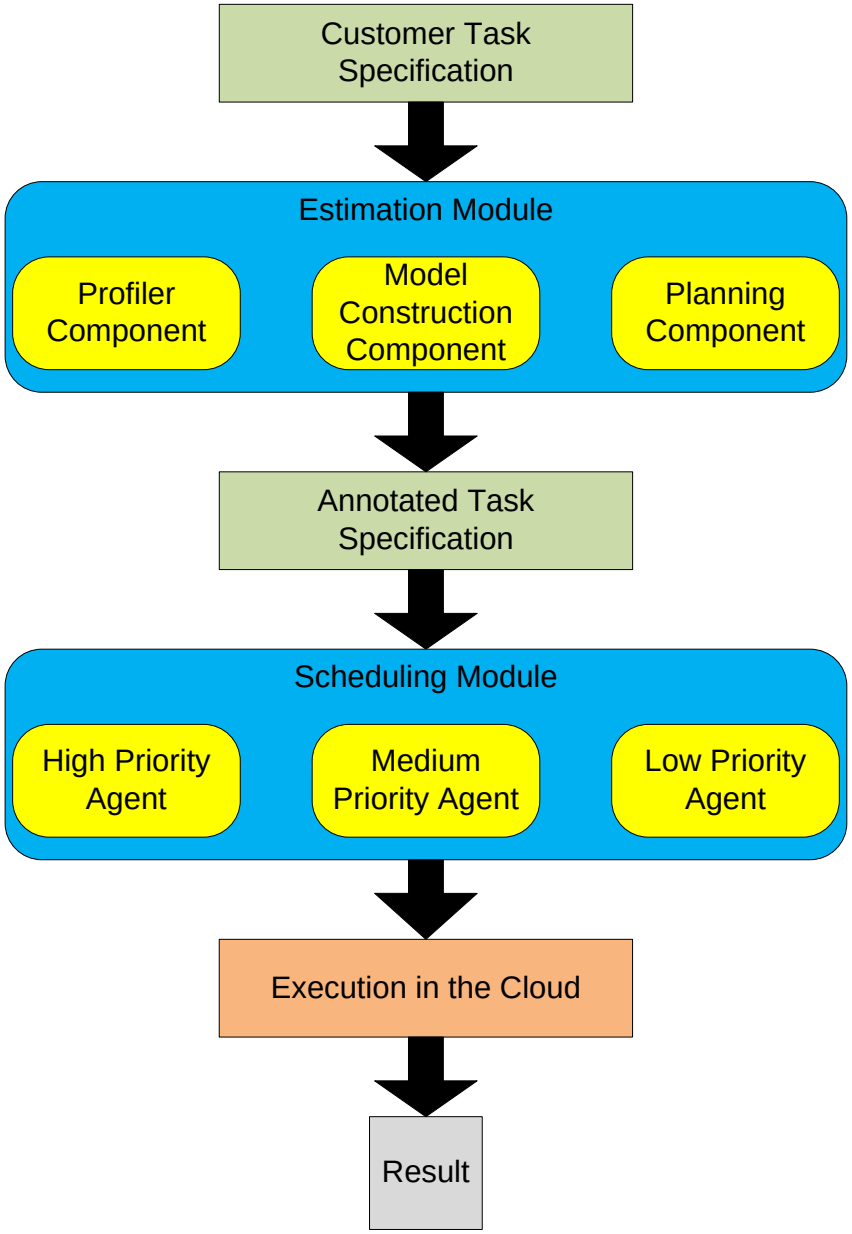
- Cloud providers offer various types of VM instances at different prices, with varying compute, network, and storage capabilities.
- Amazon EC2 provides three main pricing models:
  - On-demand instances: Charge for compute capacity by hour, no long-term commitment.
  - Reserved instances: One-time discount payment for long-term use.
  - Spot instances: Bid for spare Amazon EC2 instances.
    - Spot price is determined by supply and demand.
    - Spot price >= bid price, get instance.
    - Spot price < bid price, wait or interrupted.
    - Spot instances are often much cheaper than on-demand instances.
    - Risk 1: not get the compute resources if the bid price is too low.
    - Risk 2: executing program is interrupted without prior notice, if spot price exceeds bid price.
- Focus on on-demand and spot instances in this work.

Motivation: sizing problem for virtual cluster

- Size refers to number of VMs that construct the virtual cluster.
- Development and debugging purpose: cost is important, a small number of low-end instances is often sufficient.
- Production purpose: a balance between performance and cost, need a proper size.
- Sizing problem for virtual cluster: difficult for users to determine a proper virtual cluster size that can meet the job deadline while minimizing the cost.
  - Using a smaller cluster size, may not finish the job in time.
  - Using an unnecessarily large cluster size may incur high cost but not reduce the execution time. Because an HPC application usually does not scale beyond a certain cluster size.
- Propose an auto-provisioning framework to explore techniques to solve sizing problem.

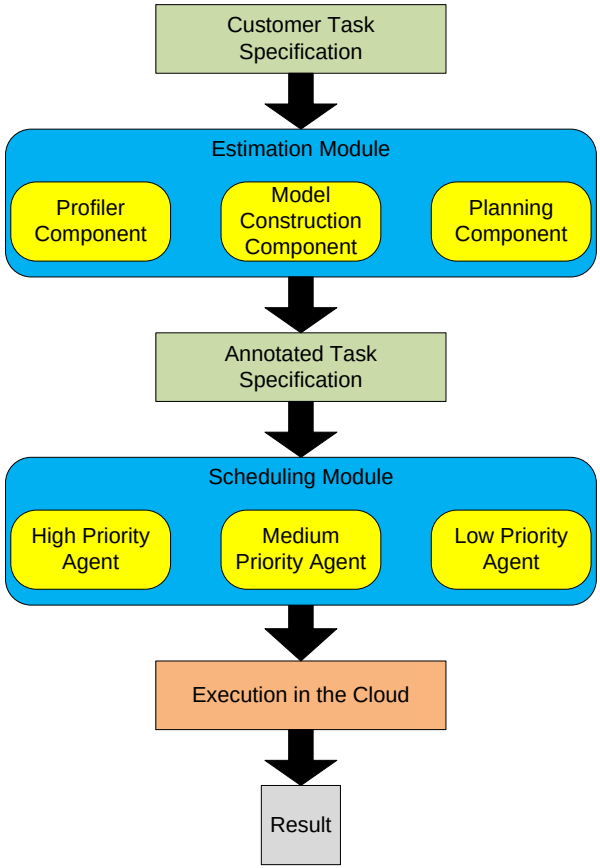
Auto provision framework: CAP^3 overview

- Objective: facilitate the execution of scientific parallel applications in the Cloud
  - Automate the entire process of cluster set-up and execute parallel application in Cloud.
  - Meet deadline.
  - Reduce cost.
- Estimation module: responsible for generating performance and cost estimates based on the given application specifications submitted by the user.
- Scheduling module: take the output from estimation module as input and schedule based on remaining time.
  - Whether and how to use the on-demand or spot instances.



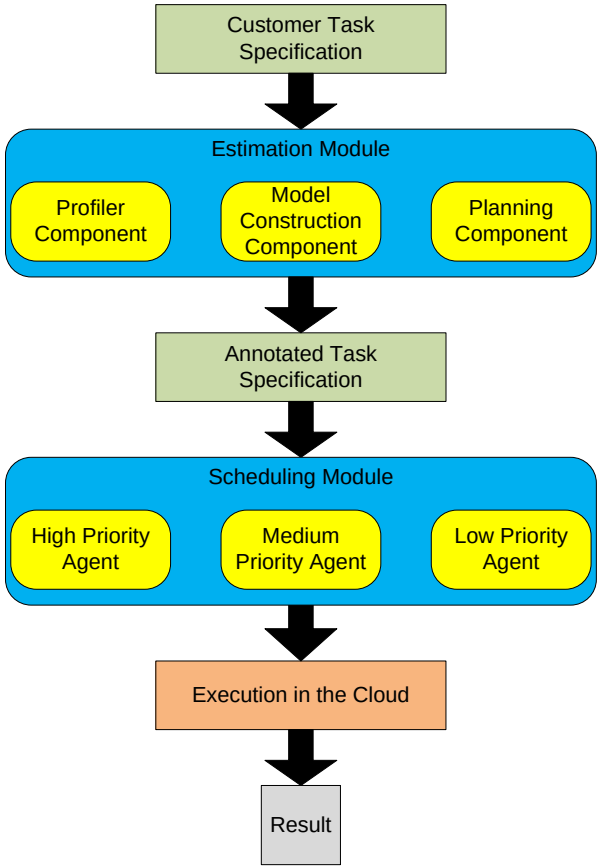
Workflow

1. User submits task specifications to CAP3.
2. The estimation module receives the task specification.
3. A profiler component prepares a small-sized virtual cluster and performs the sample runs using small training datasets for profiling. Application runtime features are extracted from this profiling run.
4. Model construction component takes these profiles to build performance models to estimate application execution time.



Workflow

5. Planning component combines Cloud-specific information and the prediction results, and then determines the appropriate cluster size.
6. Estimation module outputs the desired size and estimated execution time as annotated task specification to the scheduling module.
7. Based on the results from the estimation module and deadline constraints, the scheduling module invokes appropriate agents.
   - Agents: interact with the Cloud provider and run the application in Cloud.
8. Once an agent is selected, it creates a virtual cluster, sets up the application and data, and runs the task.



Estimation module: profiler component

- TAU (Tuning and Analysis Utilities) as a profiling tool in CAP[3].
  - TAU is a portable profiling and tracing toolkit in wide use for performance analysis of parallel scientific applications.
- Generate a script for application profiling based on the customer task specification.
- TAU automatically instruments the application code.
- Run with small sample data set on small size of cluster.
- TAU-instrumented code gathers performance related information during execution.
- Parser analyzes and extracts time of each function or major loops in the application code.

Estimation module : Model Construction Component

- Use profiles to build performance model that can estimate execution time.
- Numerous effort has been put on this topic, but no perfect solution.
- Use multiple linear regression to predict the execution time.
- Assume program execution time = computation time + communication time (no overlap for simplicity)
- The time is expressed as:

$$\log T_C = c_0 + \sum_{j=1}^{r} (c_j \cdot \log x_j) + c_{(r+1)} \cdot \log P$$

$$\log T_N = c_0 + \sum_{j=1}^{r} (c_j \cdot \log x_j) + c_{(r+1)} \cdot \log (\log P)$$

T_C , T_N is the execution time for commutation and communication, c_j is the coefficients, P is the number of MPI processes, x_j is the application-specific feature variables and r is the number of feature variables.

Estimation module : Model Construction Component

- Profiler component generates series of training data (T'_C, P', x'_1, x'_2, x'_3 ,…)
- Plug in training data to the equations.
- Use least square method to determine the model parameter c_j.

Estimation module: planning component

- Compute approximate cost and determine the number of instances used for real execution.
- Cost of executing a task

$$cost_{task} = c_{unit} \times \lceil \frac{num\_proc}{ppn} \rceil \times \lceil T_C + T_N \rceil$$

- Given the deadline and budget, the planning component iterates over the predefined range of number of processes, and calls the model construction component and the above equation to get the estimated time and cost.
- Choose size / number of processes that meet deadline and budget.

Scheduling module:

- Take annotated task specification as input, and schedules the task to appropriate virtual clusters to run.
- Priority Rate: priority of a task

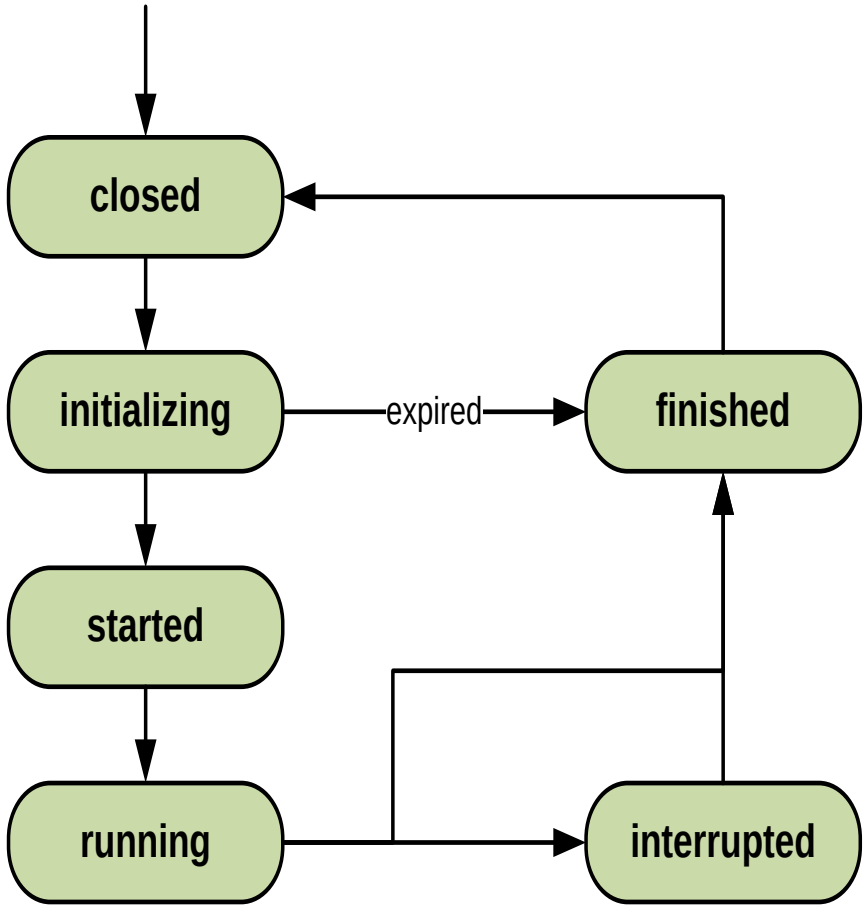
$$PR = \frac{T_{deadline} - T_{now} - T_{est} - T_{latency}}{T_{est}}$$

- Schedule to different agent according to PR

Agent	Cluster	PR range	Explanation
framework agent AG	N/A	all	update and check task.PR, distribute tasks to appropriate cluster agent
ag_{hi}	high priority on-demand instance cluster	**$PR < th_1$**	run high priority task on on-demand instances
ag_{me}	medium priority spot instance cluster	**$th_1 \leq PR \leq th_2$**	run medium priority tasks on spot instances, bid at relatively high price
ag_{lo}	low priority spot instance cluster	**$PR > th_2$**	run tasks with low priority on spot instances, bid relatively low price

Cluster agent

- Figure shows status transition of agent.
- Initialing: check and set spot price if using medium or low agent, then wait to started. Can expire if waiting too long for spot instances.
- Started: virtual cluster is ready to run applications.
- Running: run applications. Can be interrupted because of spot price fluctuation.
- Interrupted: applications is interrupted.
- Finished: virtual cluster is going to shut down, task is sent to framework agent if task failed.



Evaluation

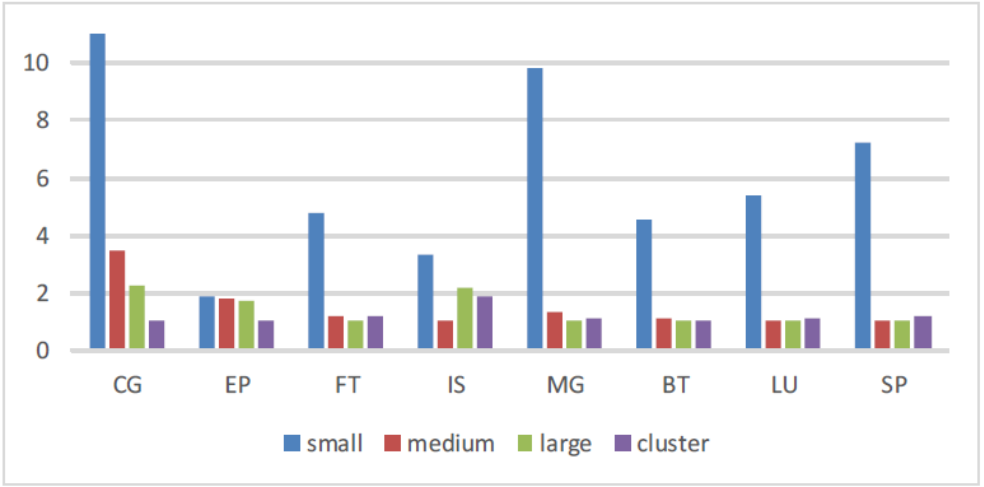
- Experiment environment:
  - Amazon EC2
  - StarCluster 0.93.3
  - TAU 2.22
  - Python 2.7 with NumPy 1.6.2 library
  - NAS Parallel Benchmarks (NPB) 3.3.1: MPI C or Fortran code
    - CG: conjugate gradient, irregular memory access and communication
    - FT: discrete 3D FFTs, all-to-all communication
    - MG: multi-grid on a sequence of meshes, long- and short distance communication, memory intensive
    - EP: embarrassingly parallel
    - IS: integer sort, random memory access
    - BT: block tri-diagonal solver
    - SP: scalar penta-diagonal solver
    - LU: lower-upper Gauss Seidel

Instance type

- Table: attributes of four typical instance type.
- Cluster instance is the cheapest per computer unit (CU).

- Figure: Cost comparison using different instance types for different tasks on small data set.
- Cluster instance is the best for most cases. Consistent with the table.
- All the following experiments use cluster instance except IS using medium instance.

Instance Type Name	Memory	CU	PPN	On-demand Price	Price Per CU
small (m1.small)	1.7 GB	1	1	$0.06	$0.06
medium (m1.medium)	3.75 GB	2	1	$0.12	$0.06
large (m1.large)	7.5 GB	4	2	$0.24	$0.06
cluster (cc1.4xlarge)	23 GB	33.5	16	$1.30	$0.039

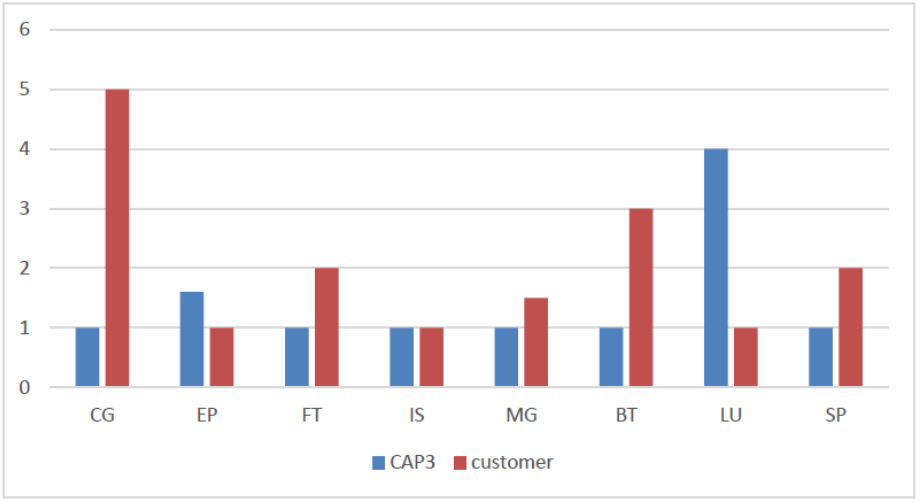


Constrains satisfaction and cost comparison of on-demand instance

- Table (top): configuration provided by CAP^3 and cost ignorant customer.
- The #proc given by CAP^3 are given by evaluation module.
- CAP^3 estimation module tends to give a smaller size (16 processes) for 6 out of 8 tasks,
- Customer tends to use a reasonable large size based on prior experiences on HPC clusters.
- CAP^3's scheduling module applies for certain number of instances according to the size provided by the estimation module.

App	CAP^3 #proc	customer #proc	priority	instance type
CG	16	64	medium	cluster
EP	96	48	high	cluster
FT	16	32	high	cluster
IS	16	32	high	medium
MG	16	32	low	cluster
BT	16	25	low	cluster
LU	128	64	low	cluster
SP	16	25	medium	cluster

	CG	EP	FT	IS	MG	BT	LU	SP	sum
CAP^3 dl	✓	✓	✓	✓	✓	✓	✓	✓	8
Cus dl	✓		✓	✓	✓	✓	✓	✓	7
CAP^3 bd	✓		✓	✓	✓	✓		✓	6
Cus bd		✓	✓	✓	✓		✓		5

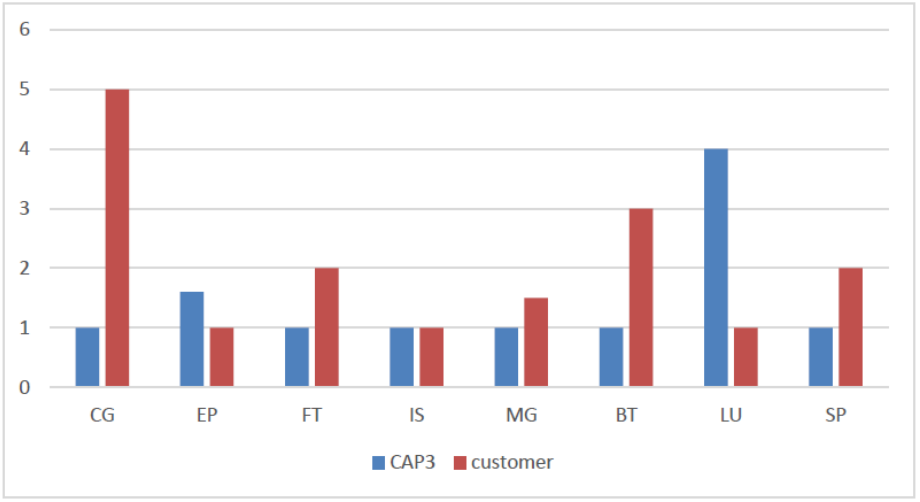


Constrains satisfaction and cost comparison of on-demand instance

- Table (middle): deadline and budget satisfaction. dl: deadline, bg: budget, cus: customer. Checked sign indicates constrain is met.
- CAP3 meets all deadline, while customer missed one deadline.
- Six out of eight tasks using CAP3 were within budget, while five were within budget for customer.
- CAP3 can meet deadline and budget in most cases, better than the customer.

App	CAP3 #proc	customer #proc	priority	instance type
CG	16	64	medium	cluster
EP	96	48	high	cluster
FT	16	32	high	cluster
IS	16	32	high	medium
MG	16	32	low	cluster
BT	16	25	low	cluster
LU	128	64	low	cluster
SP	16	25	medium	cluster

	CG	EP	FT	IS	MG	BT	LU	SP	sum
CAP3 dl	✓	✓	✓	✓	✓	✓	✓	✓	8
Cus dl	✓		✓	✓	✓	✓	✓	✓	7
CAP3 bd	✓		✓	✓	✓	✓		✓	6
Cus bd		✓	✓	✓	✓		✓		5

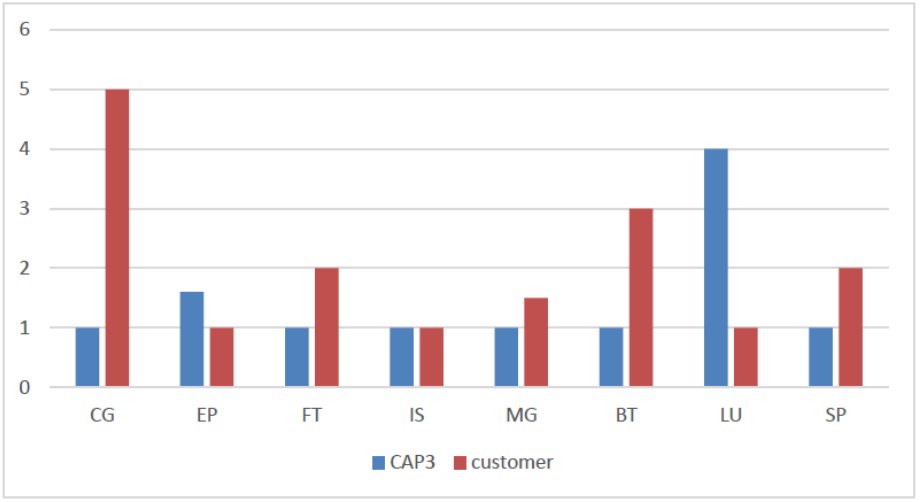


Constrains satisfaction and cost comparison of on-demand instance

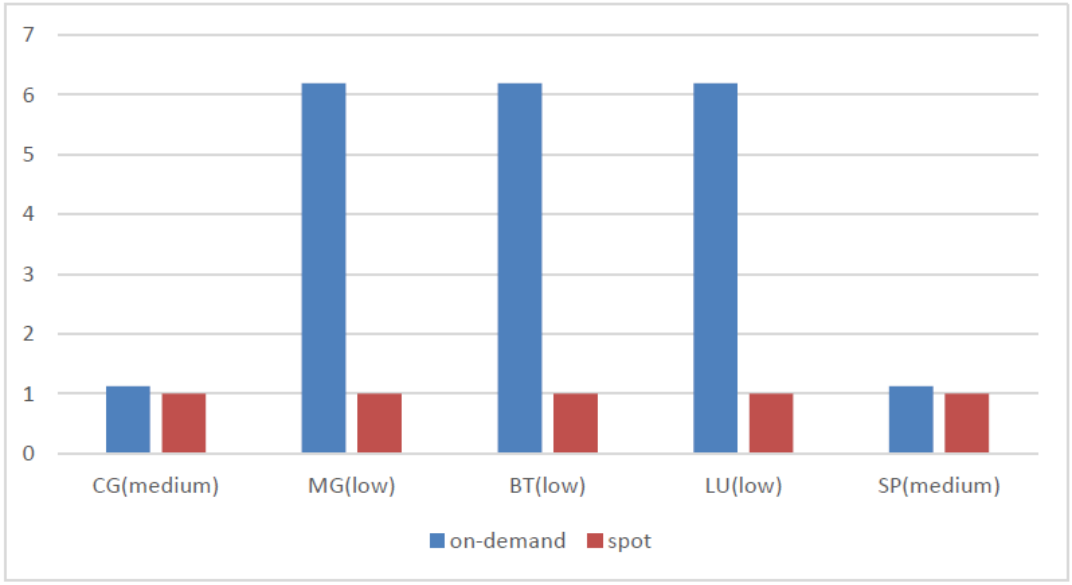
- Figure (bottom): compare cost of using configuration provided by CAP3 and cost ignorant customer.
- Use on-demand instances for simple comparison. Will discuss spot instance soon.
- Five (CG, FT, MG, BT, SP) of tasks using CAP3 cost less than the customers, one (IS) task cost the same, and two (EP, LU) tasks cost more.
- On average, CAP3's solution costs less than customer's solution.
- False size of LU: can solve this problem by running training dataset with larger number of processes

App	CAP3 #proc	customer #proc	priority	instance type
CG	16	64	medium	cluster
EP	96	48	high	cluster
FT	16	32	high	cluster
IS	16	32	high	medium
MG	16	32	low	cluster
BT	16	25	low	cluster
LU	128	64	low	cluster
SP	16	25	medium	cluster

	CG	EP	FT	IS	MG	BT	LU	SP	sum
CAP3 dl	✓	✓	✓	✓	✓	✓	✓	✓	8
Cus dl	✓		✓	✓	✓	✓	✓	✓	7
CAP3 bd	✓		✓	✓	✓	✓		✓	6
Cus bd		✓	✓	✓	✓		✓		5

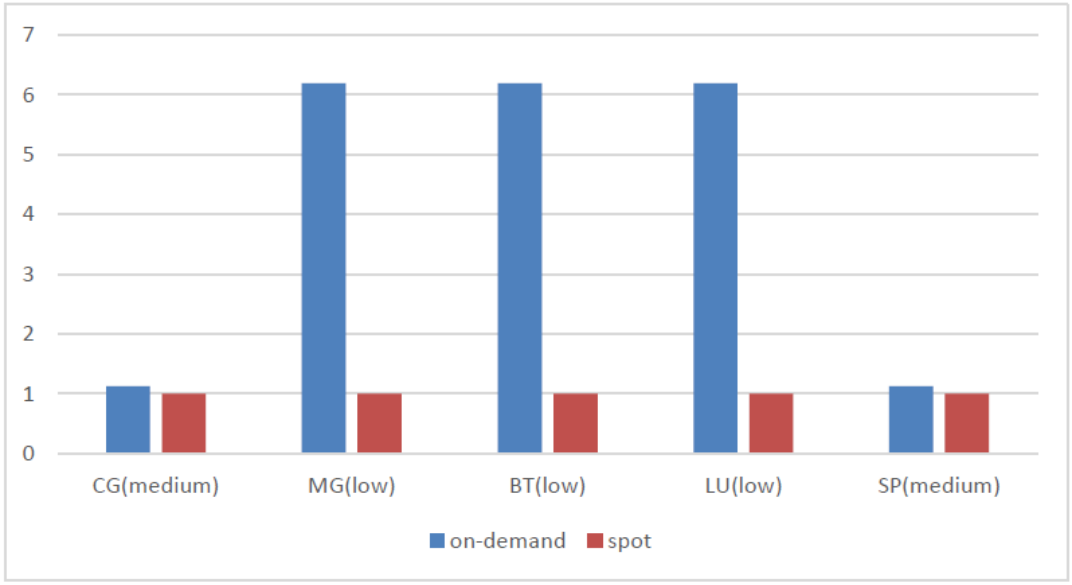


Cost Comparison of Spot Instance



- CAP3 can intelligently choose between on-demand instance and spot instance for each task according to remaining time.
- According to scheduling, CG and SP use medium priority agent to bid at higher price for spot instance, and MG, BT, LU use low priority agent to bid at lower price for spot instances.
- Figure: compare cost between on-demand and spot instances for tasks that used medium or low priority agent.

Cost Comparison of Spot Instance



- Cost computed by assuming charged at bid price.
  - Actually charged at spot price lower than bid price.
  - Actual cost is equal or lower than the cost in the figure.
- By automatically choosing spot instance, the cost was further reduced significantly.
- Risk of interruption:
  - Low priority agent faces the higher risk of interruption because of low bid price.
  - Deliver the failed task to framework agent.
  - Framework agent will reassign the failed task to appropriate cluster agent.

Conclusion and future work

- CAP3 is an auto-provisioning framework for HPC applications in Cloud.
- Help a Cloud user to apply for instances and run applications without users interference.
- Reduce the monetary expense by automatically choosing the proper number of instances and the right instance type (on-demand or spot).

- Will benefit from a more sophisticated and more accurate performance estimation module. Current is simple and not accurate sometimes.
- Only support one task per user. Extend to handle multiple tasks for multiple users.