CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics

Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen,
Shivaram Venkataraman, Minlan Yu, Ming Zhang

Yale

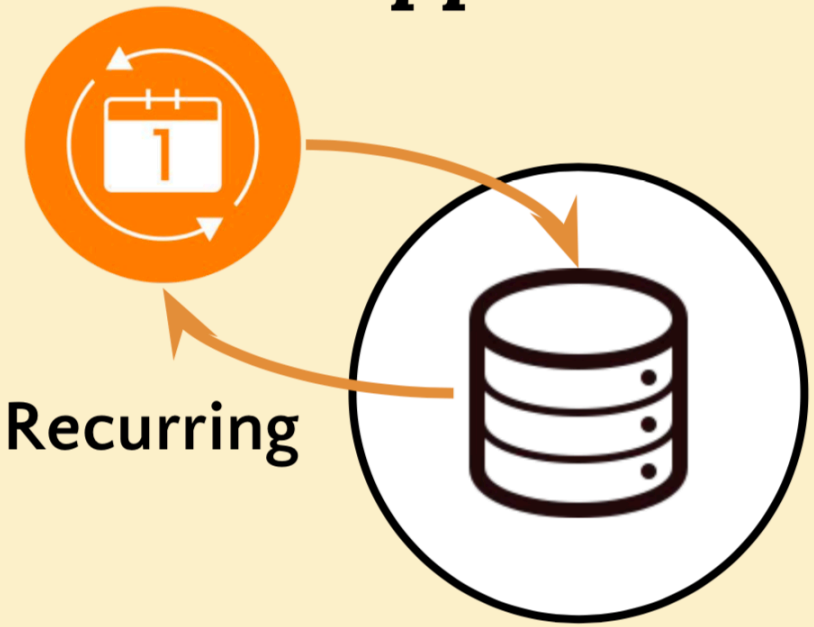
Microsoft Research

Berkeley
UNIVERSITY OF CALIFORNIA

Alibaba.com

Recurring jobs are popular

Apps & Data



Frameworks

Microsoft reports 40% of key jobs at Bing are rerun periodically.

Providers	Machine Type	Cluster Size
AWS	r3.8xlarge, i2.8xlarge, m4.8xlarge, c4.8xlarge,	
Azure	A12, D1, D2, D3, L4s, ...	
Google Cloud Platform	n1-standard-4, n1-highmem-2, n1-highcpu-4, f1-micro, ... + **configurable VMs**	

Hundreds of instance types and instance count combinations

Choosing a good configuration is important

Better performance:

- For the same cost: best/worst running time is up to 3x
  - Worst case has good CPUs whereas the memory is bottlenecked.

Lower cost:

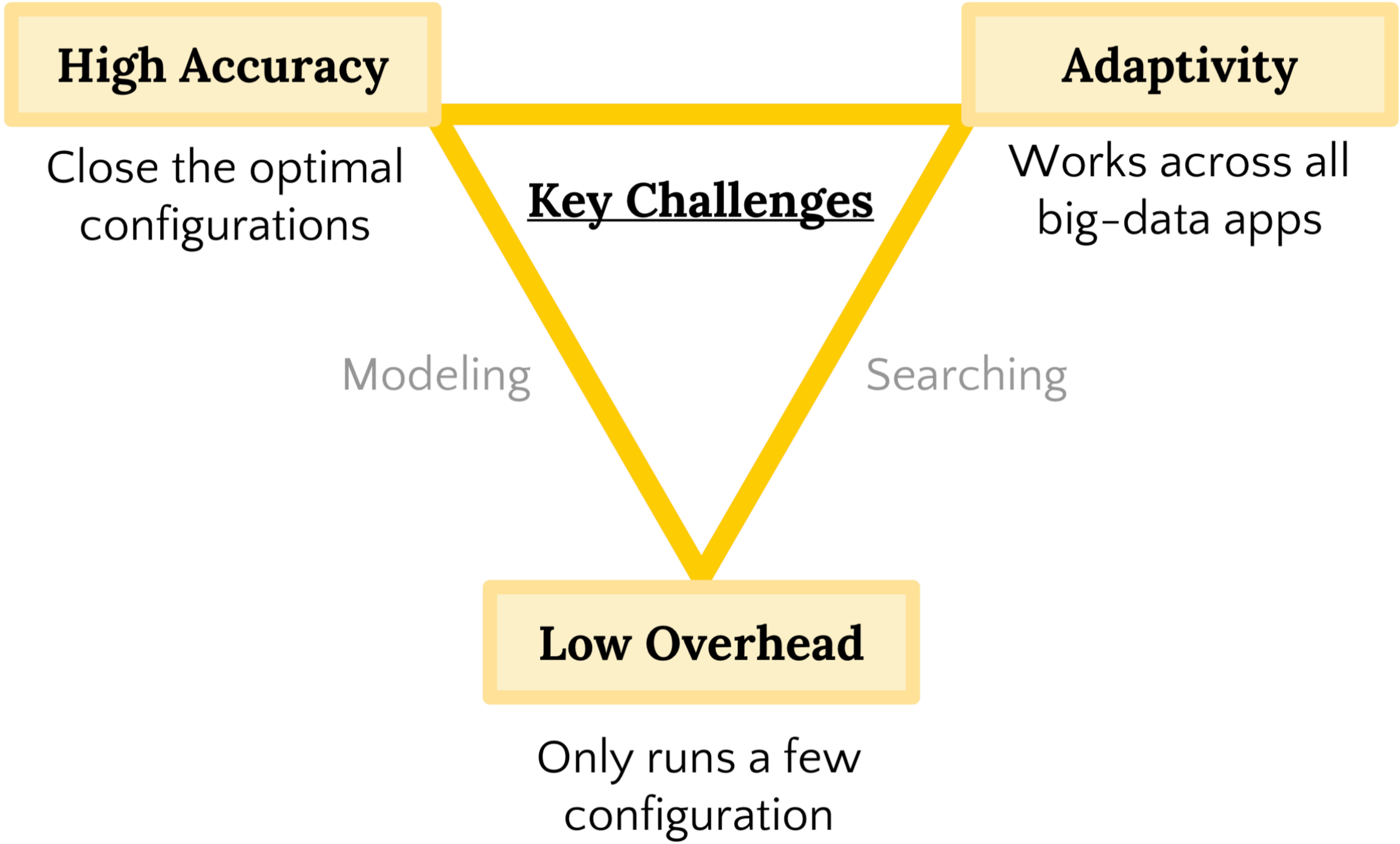
- For the same performance: best/worst cost is up to 12x
  - No need for expensive dedicated disks in the worst config.
  - 2$ vs. 24$ per job with 100s of monthly runs

Application	Avg/min	Max/min
TPC-DS	3.4	9.6
TPC-H	2.9	12
Regression (SparkML)	2.6	5.2
TeraSort	1.6	3.0

How to find the best cloud configuration

One that minimizes the cost given a performance constraint

for a recurring job, given its representative workload?


High Accuracy
Close the optimal configurations
Adaptivity
Works across all big-data apps
Key Challenges
Modeling
Searching
Low Overhead
Only runs a few configuration

Existing solution: searching

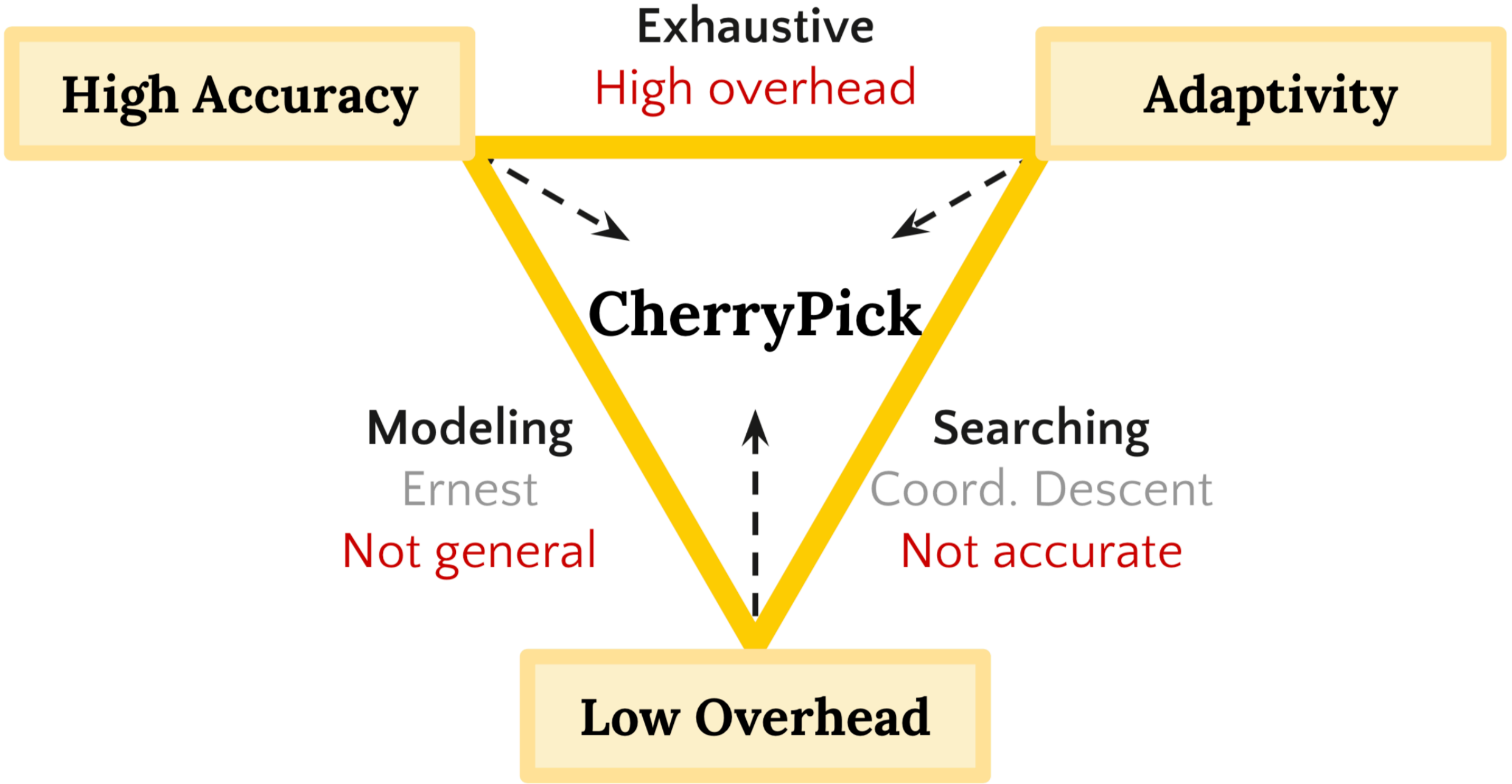
- Systematically search each dimension (Coordinate descent)
  - On each resources: RAM, CPU, disk, cluster sizes

- Problem: not accurate
  - Non-convex performance/cost curves across many resources
  - If you search one dimension, drop early, it would mislead you later

Existing solution: modeling

- Modeling the resource-perf/cost tradeoffs
  - Ernest [NSDI 16] models machine learning apps for each machine type

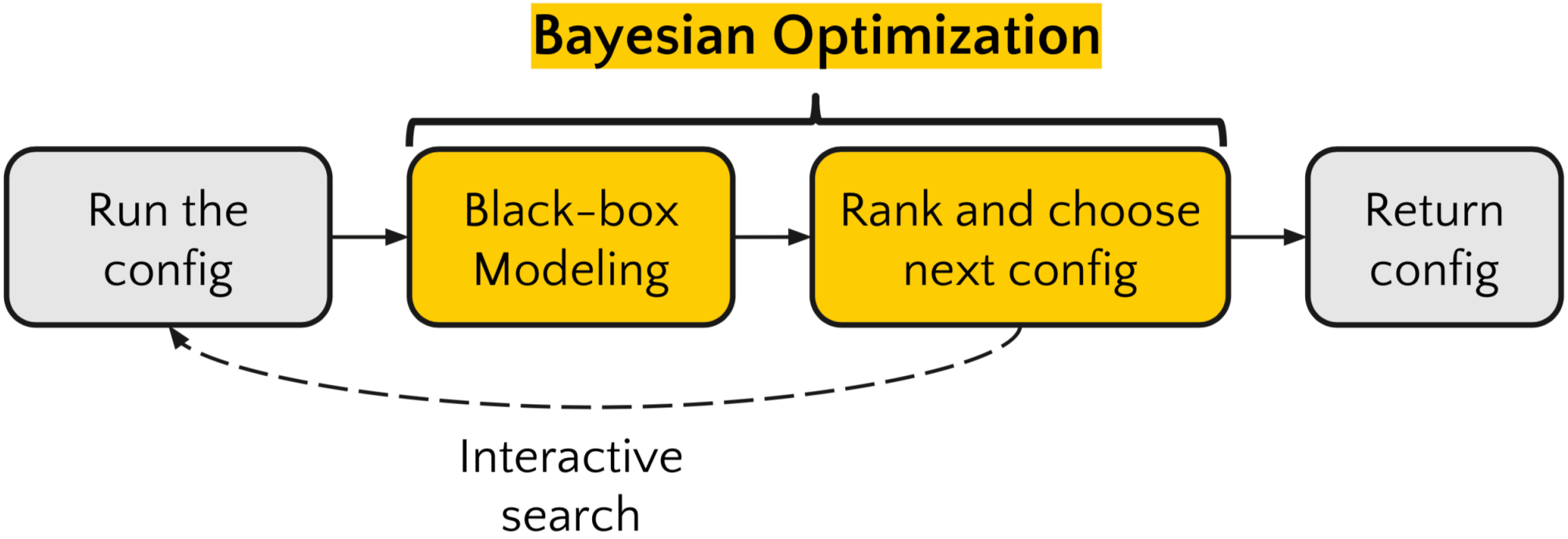
- Problem: not adaptive
  - Frameworks (e.g., MapReduce or Spark)
  - Applications (e.g., machine learning or database)
  - Machine type (e.g., memory vs CPU intensive)


High Accuracy
Exhaustive
High overhead
Adaptivity
CherryPick
Modeling
Ernest
Not general
Searching
Coord. Descent
Not accurate
Low Overhead

Key idea of CherryPick

- Adaptivity: black-box modeling
  - Without knowing the structure of each application
- Accuracy: modeling for ranking configurations
  - No need to be accurate everywhere
- Low overhead: interactive searching
  - Smartly select next run based on existing runs

Workflow of CherryPick

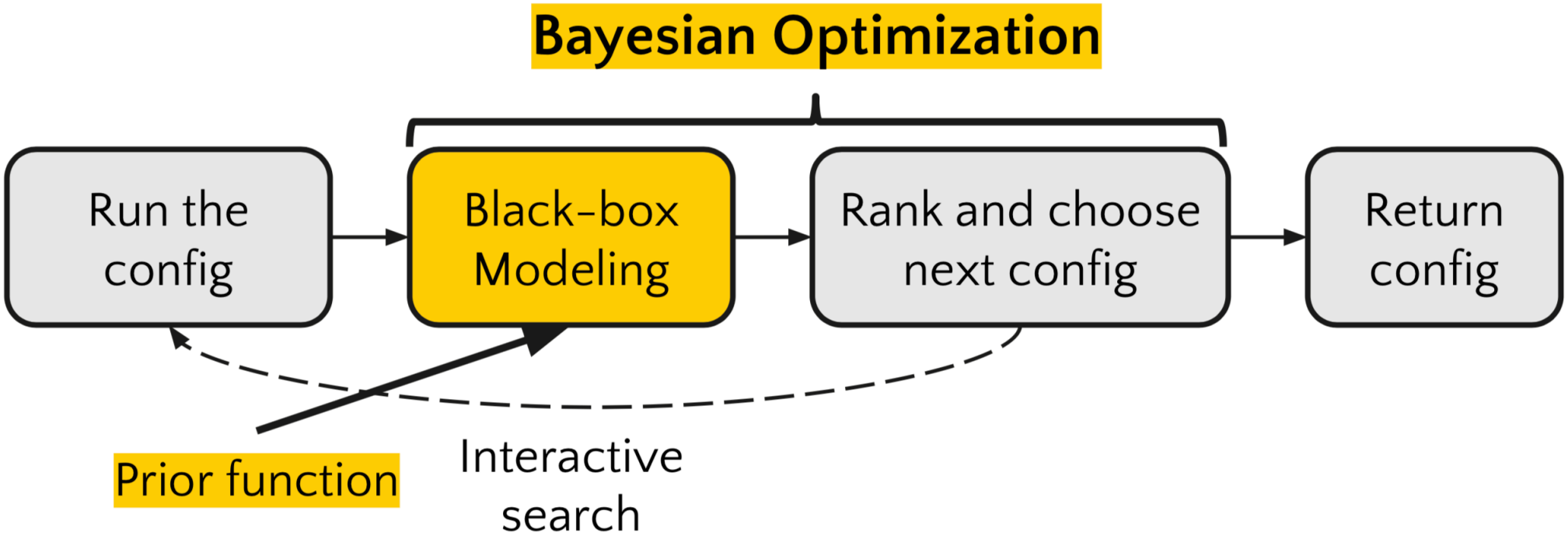


Workflow of CherryPick

$$\underset{\vec{x}}{\text{minimize}} \quad C(\vec{x}) = P(\vec{x}) \times T(\vec{x})$$

$$\text{subject to} \quad T(\vec{x}) \leq \mathscr{T}_{max}$$

Workflow of CherryPick



Workflow of CherryPick

- Noises are common in the cloud
  - Shared environment means inherent noise from other tenants
  - Even more noisy under failures, stragglers
  - **Strawman solution:** run multiple times, but high overhead.
  -
- Bayesian optimization is good at handling additive noise
  - Merge the noise in the confidence interval

$$f(\vec{X}) + \epsilon$$ ← Noise (learned by monitoring or historical data)

Workflow of CherryPick

A lot of the noise in cloud is multiplicative

- Example: if VMs IO slows down due to overloading
  - Writing 1G to disk (5 sec normally) now takes 6 secs (by 20%)
  - Writing 10G to disk (50 sec normally) now takes 60 secs (by 20%)

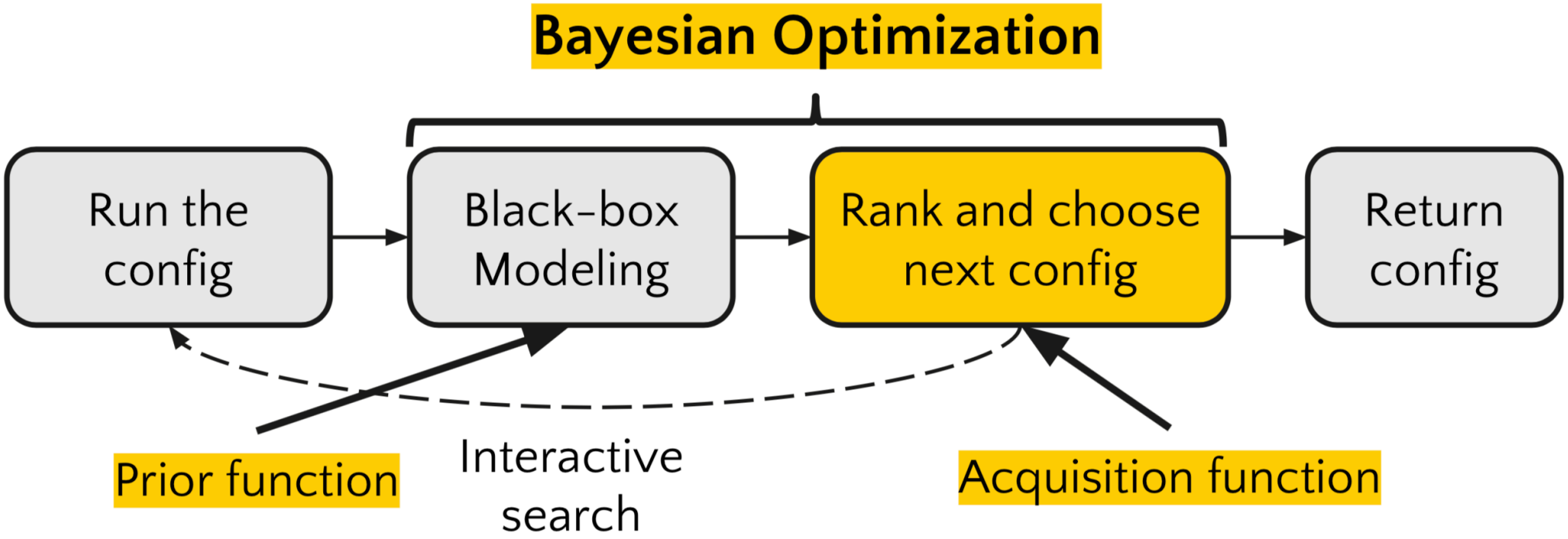
$$\hat{C}(\vec{X}) = C(\vec{X}) \times (1 + \epsilon)$$

The max. relative variance in a given cloud.

Use the log function to change to additive noise

$$\log \hat{C}(\vec{X}) = \log C(\vec{X}) + \log(1 + \epsilon)$$

Workflow of CherryPick



Workflow of CherryPick

$$EI(\vec{x})' = P[T(\vec{x}) \leq \mathscr{T}_{max}] \times EI(\vec{x})$$

Further customizations

- Discretize features:
  - Deal with infeasible configs and large searching space
  - Discretize the feature space
- Stopping condition:
  - Trade-off between accuracy and searching cost
  - Use the acquisition function knob
- Starting condition:
  - Should fully cover the whole space
  - Use quasi-random search

Evaluation

5 big-data benchmarks

- Database:
  - TPC-DS
  - TPC-H
- MapReduce:
  - TeraSort
- Machine Learning:
  - SparkML Regression
  - SparkML Kmeans

66 cloud configurations

- 30 GB-854 GB RAM
- 12-112 cores
- 5 machine types

Evaluation

Metrics

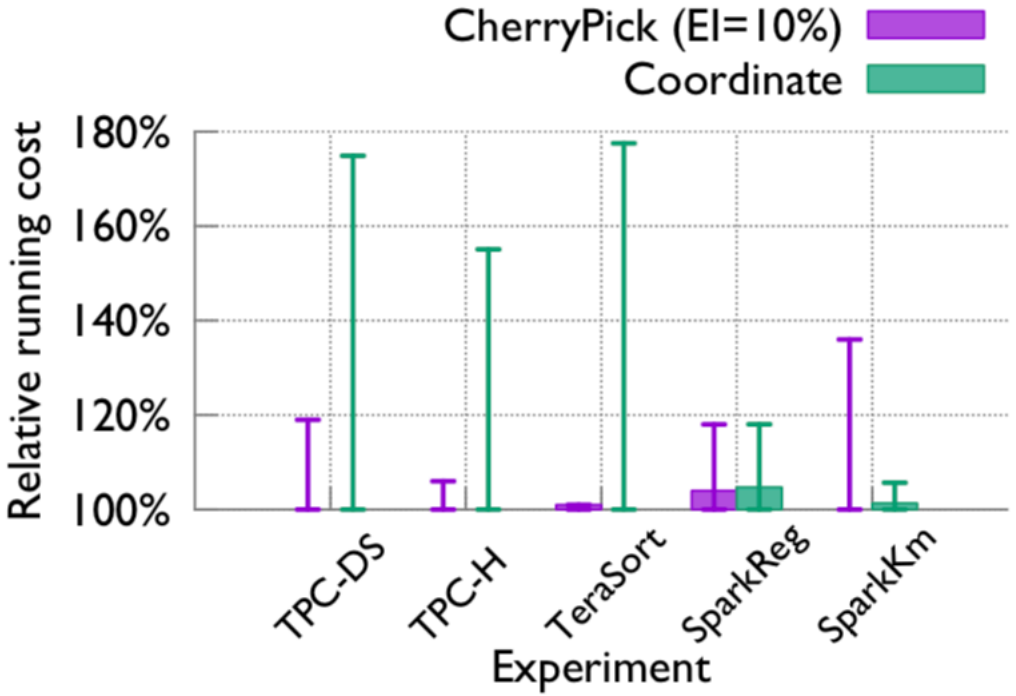
- Accuracy: running cost compared to the optimal configuration
- Overhead: searching cost of suggesting a configuration

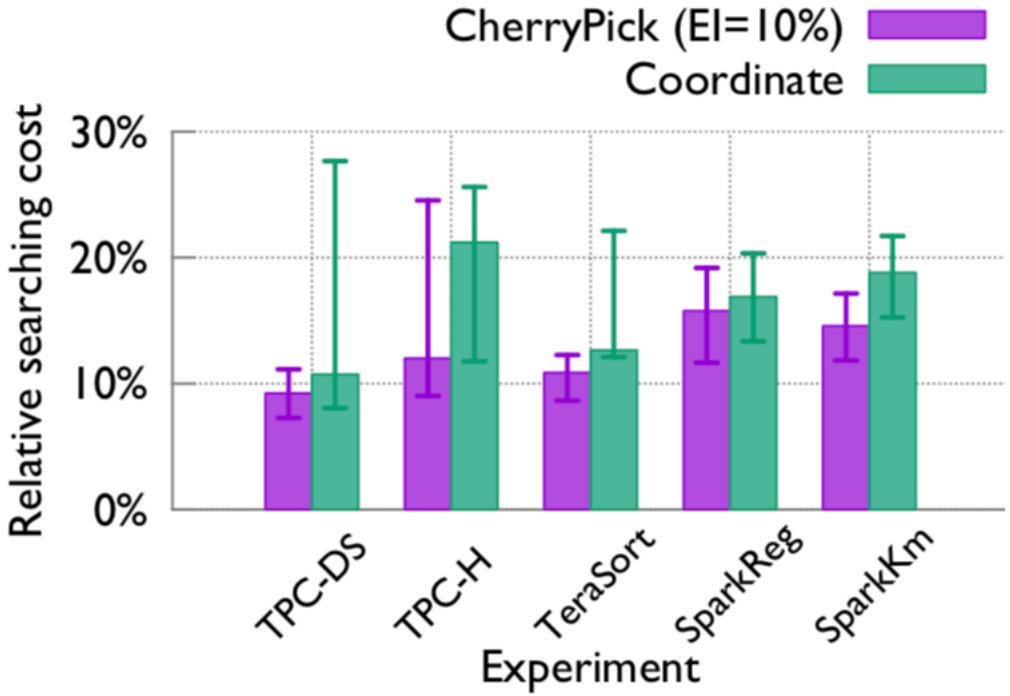
Comparing with

- Searching: random search, coordinate descent
- Modeling: Ernest [NSDI'16]

Evaluation

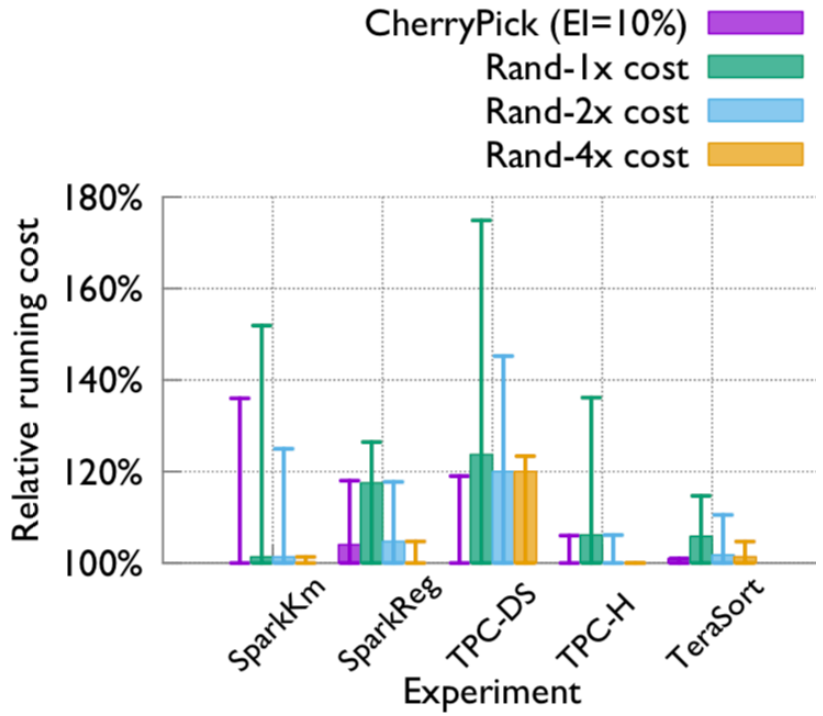
- CD: Finds opt/near-opt conf with low search cost and time
- CD: higher chance, more stable

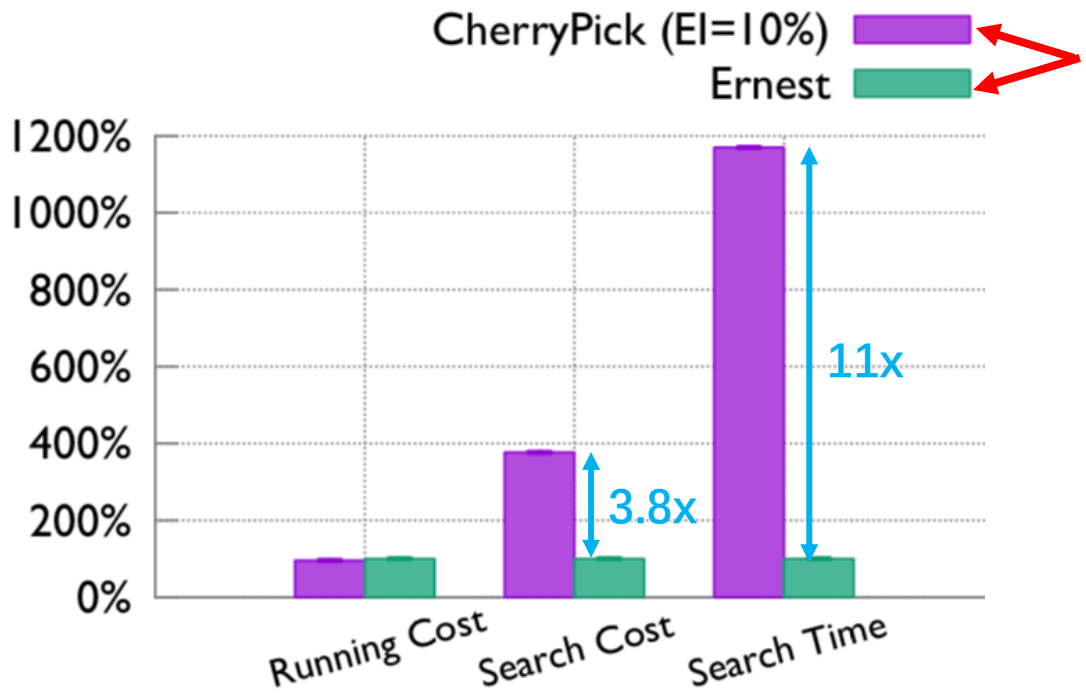




Evaluation

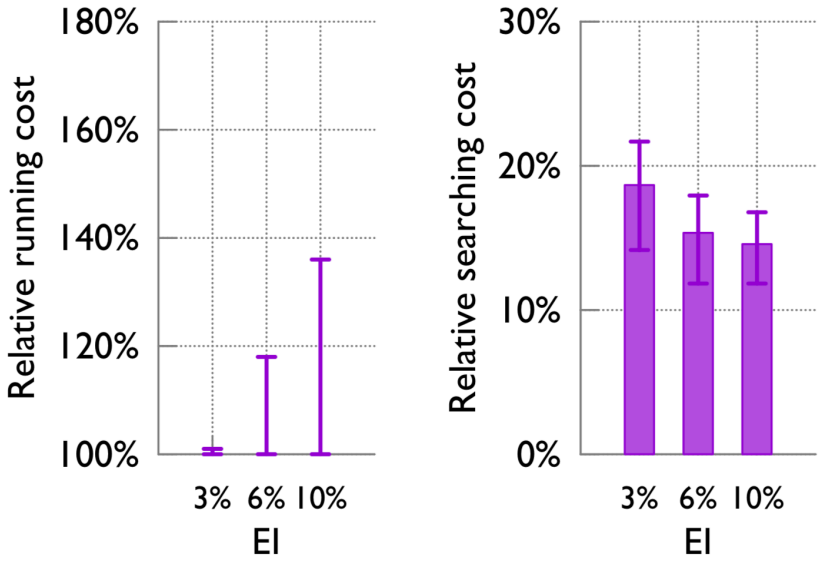
- Random: better conf & more stability with similar budget
- Ernest: similar running cost, lower search cost and time



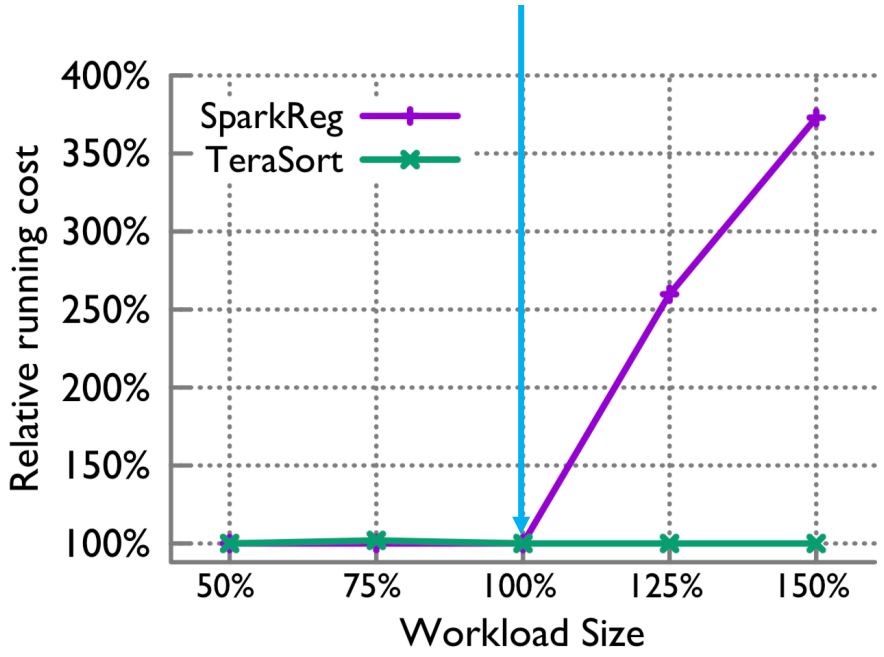


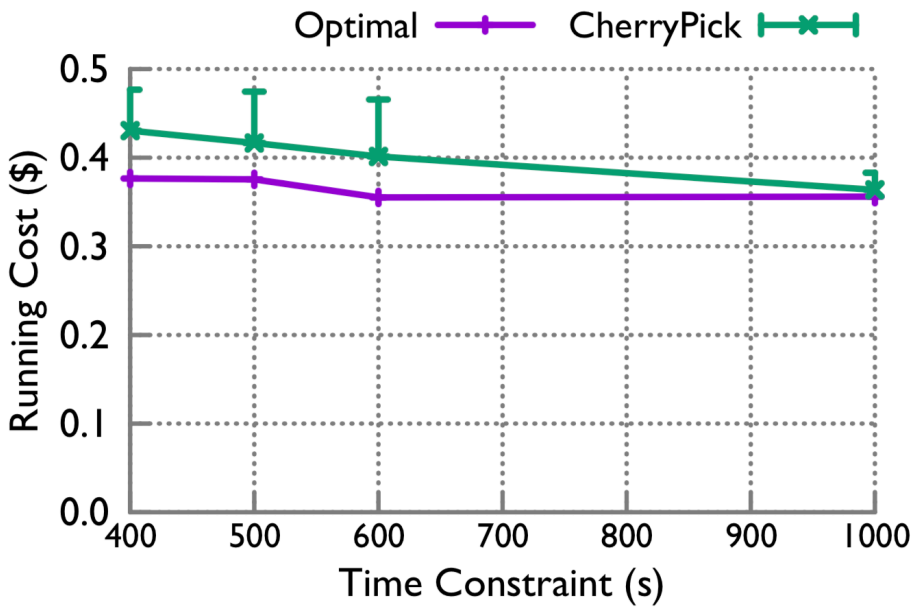
Evaluation

- Stable against EI and time constraint
- Rethink representativeness of workload



Exceed RAM budget





Conclusion

Adaptivity

Black-box modeling:
Requires running the cloud configuration.

Low overhead

Restricted amount of information:
Only a few runs available.

High accuracy

"... do not solve a more general problem ...
try to get the answer that you really need"
—Vladimir Vapnik