HPC Storage Service Autotuning Using Variational-Autoencoder-Guided Asynchronous Bayesian Optimization

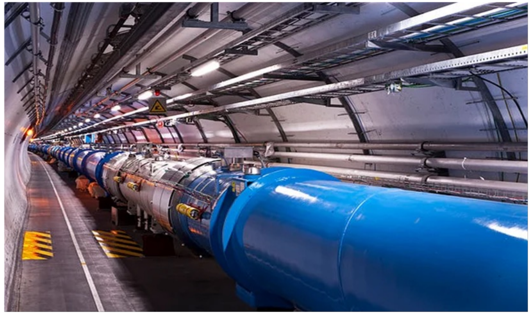
PRASANNA BALAPRAKASH

M. Dorier, R. Egele, P. Balaprakash, J. Koo, S. Madireddy, S. Ramesh, A. D. Malony, and R. Ross. "HPC Storage Service Autotuning Using Variational-Autoencoder-Guided Asynchronous Bayesian Optimization." In 2022 IEEE International Conference on Cluster Computing (CLUSTER), pp. 381-393. IEEE, 2022.

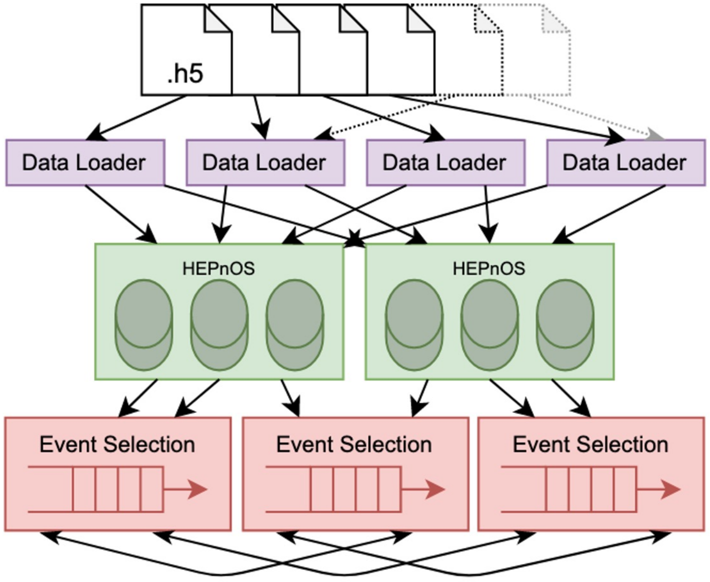
IT ALL STARTED WITH A HIGH ENERGY PHYSICS APPLICATION…

THE HEPnOS DATA SERVICE

- Motivated by scalability issues with filesystem-based storage strategies
- Designed to store “events” from HEP experiments (many small C++ objects)
- Transient storage system, in-memory or using local storage (e.g. SSDs)
- Developed using the Mochi suite of libraries for composable HPC data services
    - https://www.mcs.anl.gov/research/projects/mochi
- Provides lots of optimizations and lots of configuration knobs



The HEP Event Selection Workflow and its Parameter Space

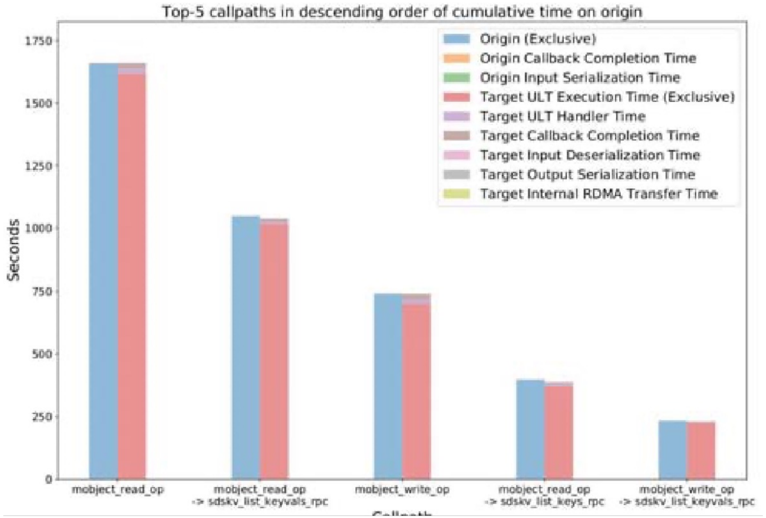


Parameter space (with values/ranges and distributions)

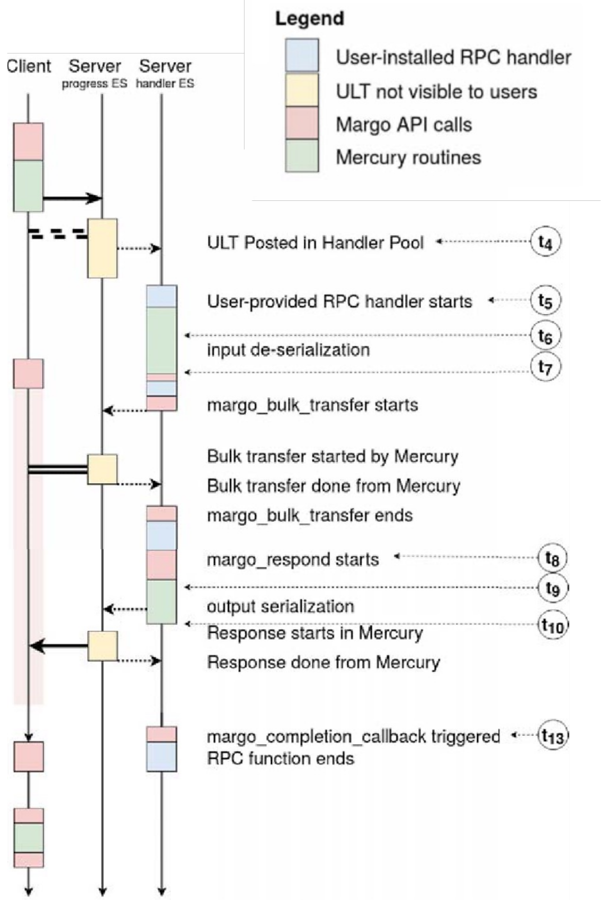
ProgressThread	(True/False, Uniform)	- Whether to use a dedicated network progress thread in Dataloader processes
WriteBatchSize	([1, 2048], Log-uniform)	- Size of the batches (in number of events used when sending data to HEPnOS
PESperNode	({1, 2, 4, 8, 16, 32}, Uniform)	- Number of Dataloader processes per physical node
LoaderAsync	(True/False, Uniform)	- Use threads to asynchronously send batches to HEPnOS
LoaderAsyncThreads	([1, 63], Log-uniform)	- Number of threads for asynchronous store in Dataloader
ProgressThread	(True/False, Uniform)	- Whether to use a dedicated network progress thread in HEPnOS servers
NumRPCthreads	([0, 63], Uniform)	- Number of threads used by HEPnOs servers to service RPC
NumEventDBs	([1, 16], Uniform)	- Number of database instances per HEPnOS server for Events
NumProductDBs	([1, 16], Uniform)	- Number of database instances per HEPnOS server for Products
NumProviders	([1, 32], Uniform)	- Number of database providers per HEPnOS server
ThreadPoolType*	({fifo, fifo_wait, prio_wait}, Uniform)	- Argobots thread pool type each provider uses
PESperNode*	({1, 2, 4, 8, 16, 32}, Uniform)	- Number of HEPnOS server processes per physical node
ProgressThread	(True/False, Uniform)	- Whether to use a dedicated network progress thread in PEP processes
NumThreads	([1, 31], Uniform)	- Uniform & Number of threads use to process data in parallel
InputBatchSize	([8, 1024], Log-uniform)	- Batch size (in number of events) to use when loading data from HEPnOS
OuputBatchSize	([8, 1024], Log-uniform)	- Batch size (in number of events) to use when sending data across PEP processes
PESperNode	({1, 2, 4, 8, 16, 32}, Uniform)	- Number of PEP processes per physical node
UsePreloading*	(True/False, Uniform)	- Use batch-prefetching of data products instead of per-product load
UseRDMA*	(True/False, Uniform)	- Use RDMA to transfer data
BusySpin	(True/False, Uniform)	- Network polling strategy (common to all three components)

MANUAL TUNING (IT'S HARD!)

Callpath ancestry appended to RPCs allows tracking and ranking distributed callpaths (e.g., by time in the callpath)



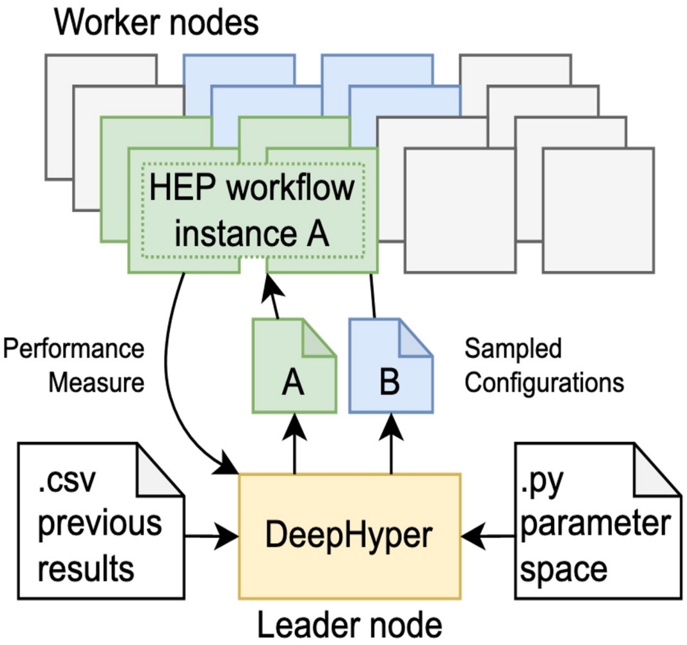
Performance variables exported by Mercury in conjunction with ULT data allow detailed analysis of timing.



AUTOMATE: BLACK-BOX TUNING WITH DEEPHYPER!

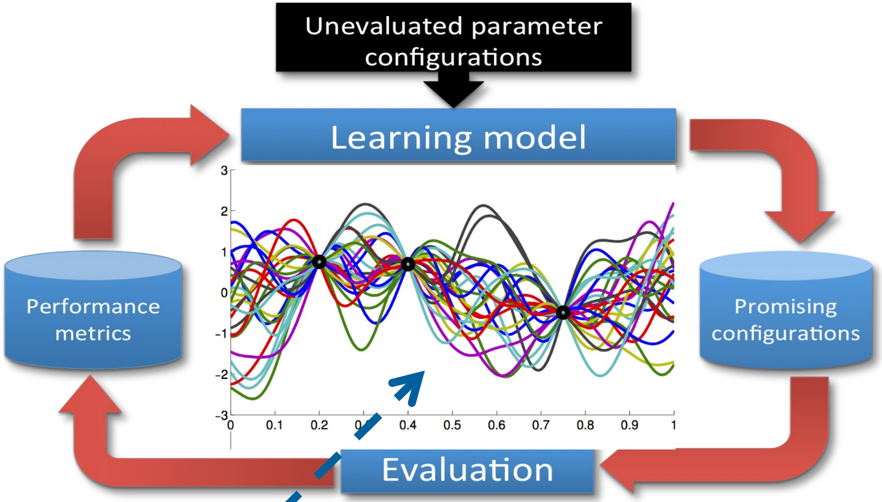
Parallel Asynchronous Bayesian Optimization

- Many instances evaluated **in parallel**
- **Asynchronous** updates

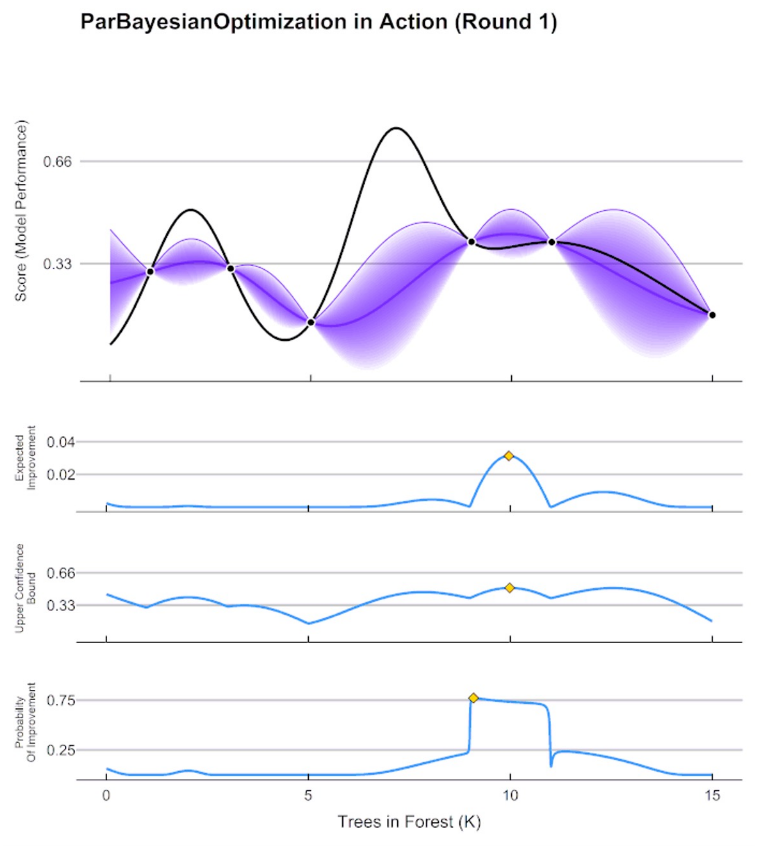


https://deephyper.readthedocs.io

Bayesian Optimization



Surrogate Model Fitted to Sampled Performance (iterative refinement improves the learning model)



https://en.wikipedia.org/wiki/Bayesian_optimization

Acquisition Functions

Upper confidence bound

$$UCB(x) = \mu(x) + \beta\sigma(x)$$

Expected improvement

$$PI(x) = \psi\left(\frac{\mu(x) - f(x^+) - \xi}{\sigma(x)}\right)$$

$\mu(x)$: mean

$\sigma(x)$: std.dev

ξ, β : parameter controlling exploration

ψ : CDF of standard Gaussian

ϕ : PDF of standard Gaussain

Probability of improvement

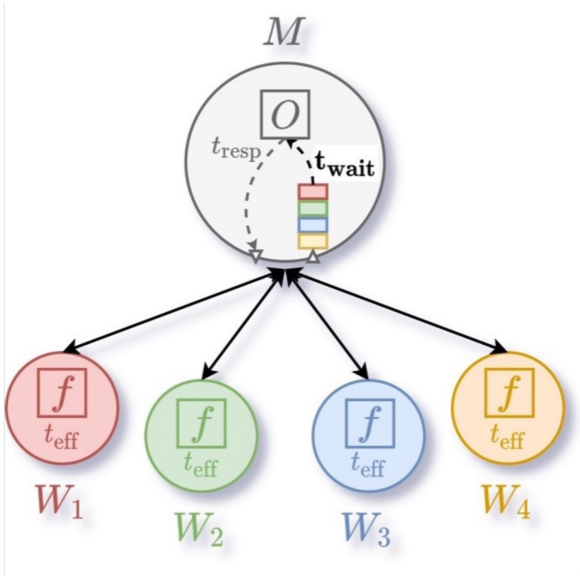
$$EI(x) = \left(\mu(x) - f(x^+) - \xi\right)\psi\left(\frac{\mu(x) - f(x^+) - \xi}{\sigma(x)}\right) + \sigma(x)\phi\left(\frac{\mu(x) - f(x^+) - \xi}{\sigma(x)}\right)$$

How to Scale?

CRAY
Argonne
NATIONAL LABORATORY
U.S. DEPARTMENT OF
ENERGY
intel
CRAY
Aurora

Asynchronous Multipoint Evaluation: Kriging believer (aka liar strategy)

- *Model M*
  - *Ensemble of regression trees*
    - *mixed integer input space*
    - *scalability due to parallelization*
    - *minimal tuning*
- Given a model M and an acquisition function *u*
  - Repeat K times (for K configurations)
    - select a point *x* that <u>maximizes</u> *acquisition function with M*
      - <u>sampling instead of optimization</u>
        - mixed integer space
        - faster
    - *predict the mean (mu) of* x using *M*
    - clone the model *M* to *M'*
    - <u>refit *M'* with x and *mu (lie)*</u>
      - *std.dev -> 0*
    - *set M' to M*



Each worker can use multiple nodes

TRASNFER LEARNING: REUSING PREVIOUS TUNING RESULTS TO SPEED UP BAYESIAN OPTIMIZATION

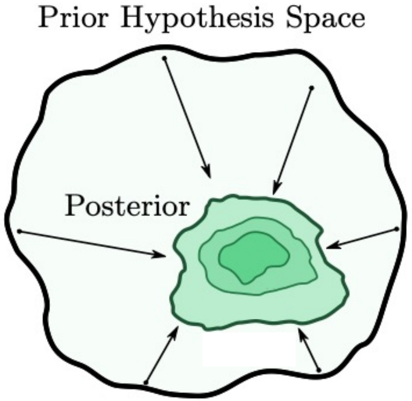
Transfer Learning

- Why? Data service tuning is **compute and resource intensive**
  - Large search spaces – (continuous/discrete)
  - Large/expensive black-box models

- **Transfer learning:** transfer the information gained from a previous related search to a new one
  - improve either the **search efficiency** or **accuracy**, or both

- **High-performing configurations** and their neighborhood from the previous (related) search
  - potentially high-performing configurations

- Define **informative prior distributions** for the parameter instead of typical non-informative (uniform distribution) prior

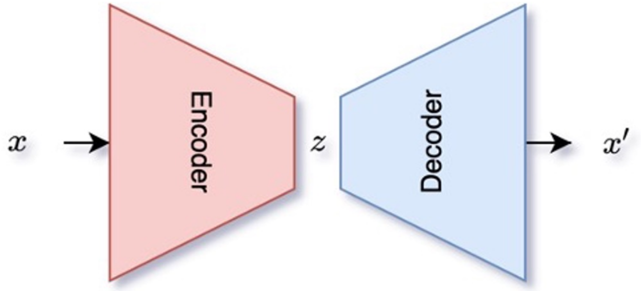


Transfer Learning with Variational Autoencoder

Hypothesis: High-performing configurations from one search can be used to bias a related search

Problem: Learn the distribution of high-performing configurations?

Solution: Density estimation



Tabular-VAE

Algorithm

- **Select** high-performing configurations from previous experiments
- **Fit** a Tabular-VAE to learn model the density of the high-performing configurations
  - p(z|x) (**encoder**)
  - p(x|z) (**decoder**)
- **Execute** BO with p(x|z) instead of p(x)

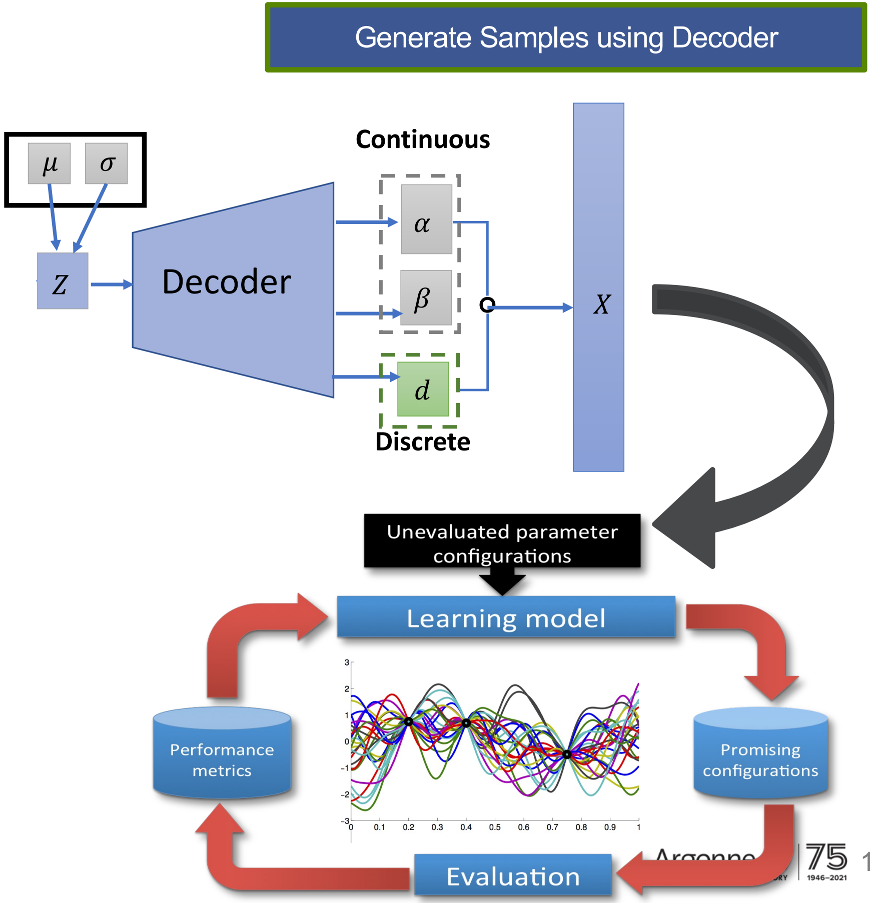
Xu, Lei, et al. "Modeling tabular data using conditional GAN." *Advances in Neural Information Processing Systems* 32 (2019).

VAE – ABO: Algorithm

- Sampling configurations in the **initialization phase** of BO

- Select candidates for evaluation in the **iterative phase** of BO

- Sampled configurations in the BO are **biased** toward the high-performing configurations from the previous run



VAE – ABO: Algorithm

Algorithm 1: Variational-Autoencoder-Guided Asynchronous BO (VAE-ABO)

inputs : $\mathbf{H}_p$: search history from previous autotuning, $q\%$: quantile value for high-performing parameter configurations selection, $\mathcal{D}^p$: previous parameter space, $\mathcal{D}^c$: current parameter space, W: workers

output: x^{curr*}: best configuration from the current autotuning, y^{curr*}, the performance metric of the x^{curr*}, $\mathcal{H}$, evaluations from the search

/* Informative prior initialization */

$\mathbf{Q}_p \leftarrow$ subset($\mathbf{H}_p$,$q\%$)

/* Fit tabular variational autoencoder using Bayesian Optimizer */

$\mathcal{P} \leftarrow$ TVAE($\mathbf{Q}_p$)

/* User-defined prior initialization for new parameters */

foreach $x_j \in \mathcal{D}^c$ **do**
    if $x_j \notin \mathcal{D}^p$ **then**
        if $x_j \in \mathcal{I}$ *or* $\mathcal{R}$ **then**
            $\mathcal{P}(x_j) =$ Uniform(l_j, u_j)
        else
            $\mathcal{P}(x_j) =$ Multinoulli(p_j)
        end
end

$\mathcal{H} \leftarrow \{\}$

optimizer $\leftarrow$ Bayesian_Optimizer($\mathcal{D}^c$,$\mathcal{P}$)

/* Initialization of BO */

for $i \leftarrow 1$ **to** W **do**
    $x_i \leftarrow$ sample_configuration($\mathcal{D}^c$ — $\mathcal{P}$)
    submit_evaluation(x_i) // Nonblocking
end

/* Optimization loop of BO */

while *stopping criterion not met* **do**
    // Query results
    $(\mathcal{X}_e, \mathcal{Y}_e) \leftarrow$ get_finished_evaluations ()
    $\mathcal{H} \leftarrow \mathcal{H} \cup (\mathcal{X}_e, \mathcal{Y}_e)$
    // Generate parameter configs
    optimizer.tell($\mathcal{X}_e, \mathcal{Y}_e$)
    $\mathcal{X}_{next} \leftarrow$ *optimizer*.ask($|\mathcal{Y}_i|$ — $\mathcal{P}$)
    submit_evaluation($\mathcal{X}_{next}$) // Nonblocking
end

x^{curr*}, y^{curr*} $\leftarrow$ find_best($\mathcal{H}$)

return x^{curr*}, y^{curr*}, $\mathcal{H}$

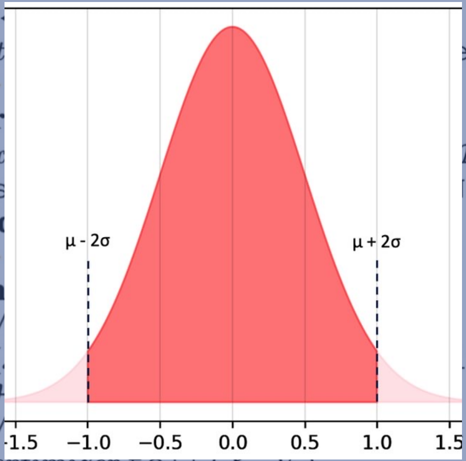
VAE – ABO: Algorithm

Algorithm 1: Variational-Autoencoder-Guided Asyncronous BO (VAE-ABO)

inputs : $\mathbf{H}_p$: search history from previous autotuning, $q\%$: quantile value for high-performing parameter configurations selection, $\mathcal{D}^p$: previous parameter space, $\mathcal{D}^c$: current parameter space, W: workers

output: x^{curr*}: best configuration from the current autotuning, y^{curr*}, the performance metric of the x^{curr*}, $\mathcal{H}$, evaluations from the search

```
/* Informative prior initialization */
Q_p ← subset(H_p, q%)
/* Fit tabular variational autoencoder
   using Bayesian Optimizer */
P ← TVAE(Q_p)
/* User-defined prior initialization for
   new parameters */
foreach x_j ∈ D^c do
    if x_j ∉ D^p then
        if x_j ∈ I or R then
            P(x_j) = Uniform(l_j, u_j)
        else
            P(x_j) = Multinoulli(p_j)
        end
end
H
opt ... er(D^c, P)
/* ... */
for
    x ... D^c — P)
    s ... onblocking
end
/* ... */
wh
    /
    ( ... uations ()
    H
    / ... figs
    optimizer.tell (X_e, Y_e)
    X_next ← optimizer.ask (|Y_i| — P)
    submit_evaluation (X_next) // Nonblocking
end
x^{curr*}, y^{curr*} ← find_best(H)
return x^{curr*}, y^{curr*}, H
```



VAE – ABO: Algorithm

Algorithm 1: Variational-Autoencoder-Guided Asyncronous BO (VAE-ABO)

inputs : $\mathbf{H}_p$: search history from previous autotuning, $q\%$: quantile value for high-performing parameter configurations selection, $\mathcal{D}^p$: previous parameter space, $\mathcal{D}^c$: current parameter space, W: workers

output: x^{curr*}: best configuration from the current autotuning, y^{curr*}, the performance metric of the x^{curr*}, $\mathcal{H}$, evaluations from the search

/* Informative prior initialization */

$\mathbf{Q}_p \leftarrow$ subset($\mathbf{H}_p$,$q\%$)

/* Fit tabular variational autoencoder using Bayesian Optimizer */

$\mathcal{P} \leftarrow$ TVAE($\mathbf{Q}_p$)

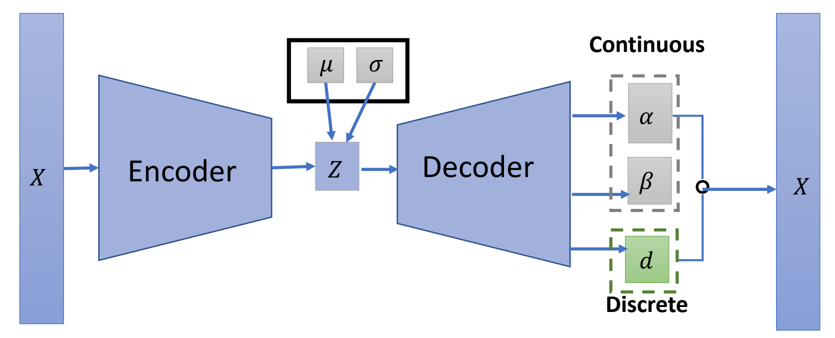
/* User-defined prior initialization for new parameters */

foreach $x_j \in \mathcal{D}^c$ **do**
  if $x_j \notin \mathcal{D}^p$ **then**
    if $x_j \in \mathcal{I}$ *or* $\mathcal{R}$ **then**
      $\mathcal{P}(x_j) =$ Uniform(l_j, u_j)
    else
      $\mathcal{P}(x_j) =$ Multinoulli(p_j)
    end
end

$\mathcal{H} \leftarrow \{\}$

optimizer $\leftarrow$ Bayesian_Optimizer($\mathcal{D}^c$,$\mathcal{P}$)

/* Initialization of BO */



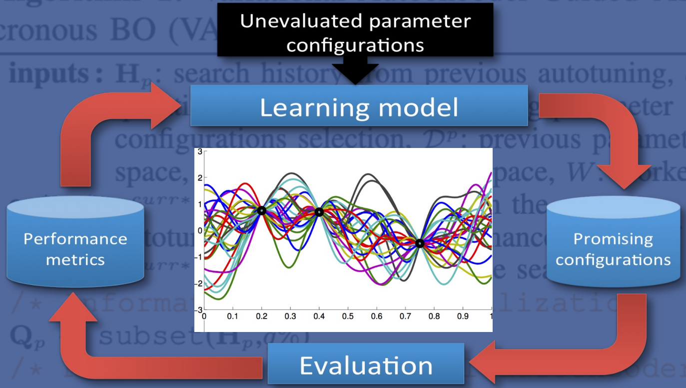
  optimizer.tell $(\mathcal{X}_e, \mathcal{Y}_e)$
  $\mathcal{X}_{next} \leftarrow$ *optimizer*.ask $(|\mathcal{Y}_i| - \mathcal{P})$
  submit_evaluation $(\mathcal{X}_{next})$ // Nonblocking
end

x^{curr*}, $y^{curr*} \leftarrow$ find_best($\mathcal{H}$)

return x^{curr*}, y^{curr*}, $\mathcal{H}$

VAE – ABO: Algorithm

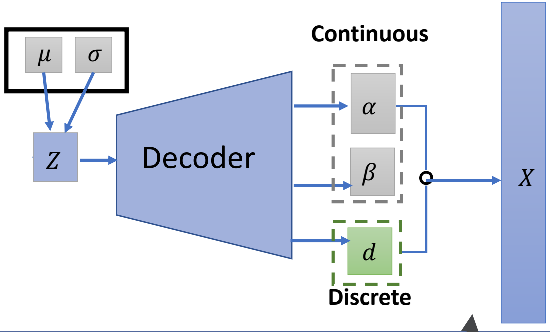
Algorithm 1: Variational-Autoencoder-Guided Asyn-cronous BO (VA...



$\mathcal{P} \leftarrow$ TVAE($\mathbf{Q}_p$)

/* User-defined prior initialization for new parameters */

foreach $x_j \in \mathcal{D}^c$ **do**
  if $x_j \notin \mathcal{D}^p$ **then**
    if $x_j \in \mathcal{I}$ *or* $\mathcal{R}$ **then**
      $\mathcal{P}(x_j) =$ Uniform(l_j, u_j)
    else
      $\mathcal{P}(x_j) =$ Multinoulli(p_j)
    end
end



$\mathcal{H} \leftarrow \{\}$
optimizer $\leftarrow$ Bayesian_Optimizer($\mathcal{D}^c$,$\mathcal{P}$)
/* Initialization of BO */
for $i \leftarrow 1$ **to** W **do**
  $x_i \leftarrow$ sample_configuration($\mathcal{D}^c$ — $\mathcal{P}$)
  submit_evaluation(x_i) // Nonblocking
end
/* Optimization loop of BO */
while *stopping criterion not met* **do**
  // Query results
  $(\mathcal{X}_e, \mathcal{Y}_e) \leftarrow$ get_finished_evaluations ()
  $\mathcal{H} \leftarrow \mathcal{H} \cup (\mathcal{X}_e, \mathcal{Y}_e)$
  // Generate parameter configs
  optimizer.tell $(\mathcal{X}_e, \mathcal{Y}_e)$
  $\mathcal{X}_{next} \leftarrow$ *optimizer*.ask $(|\mathcal{Y}_i|$ — $\mathcal{P})$
  submit_evaluation($\mathcal{X}_{next}$) // Nonblocking
end
x^{curr*}, $y^{curr*} \leftarrow$ find_best($\mathcal{H}$)
return x^{curr*}, y^{curr*}, $\mathcal{H}$

EXPERIMENTS

FIVE WAYS TO EVALUATE AN APPROACH

- Best-performing configuration
  - How good is it after 1h of autotuning?
- Mean best-performing configuration
  - Integrated best-performing time over 1h
- Number of evaluations
  - The more evaluations, the better
- Worker utilization
  - Idle workers are a waste of resources
- Search speedup
  - How much faster are we than pure luck (random sampling)?

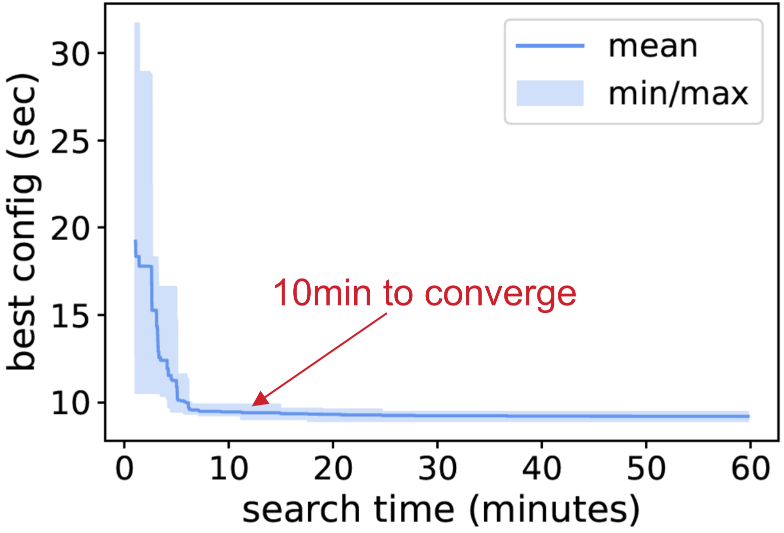
PLATFORM: THETA

Architecture	Intel-Cray XC40
Speed	11.7 petaflops
Processors per node	64 core, 1.3 GHz Intel Xeon Phi 7230
Nodes	4,392
Cores	281,088
Memory	843 TB
High-bandwidth memory	70 TB
Interconnect	Aries network with Dragonfly topology

FIVE EXPERIMENTAL SETUPS

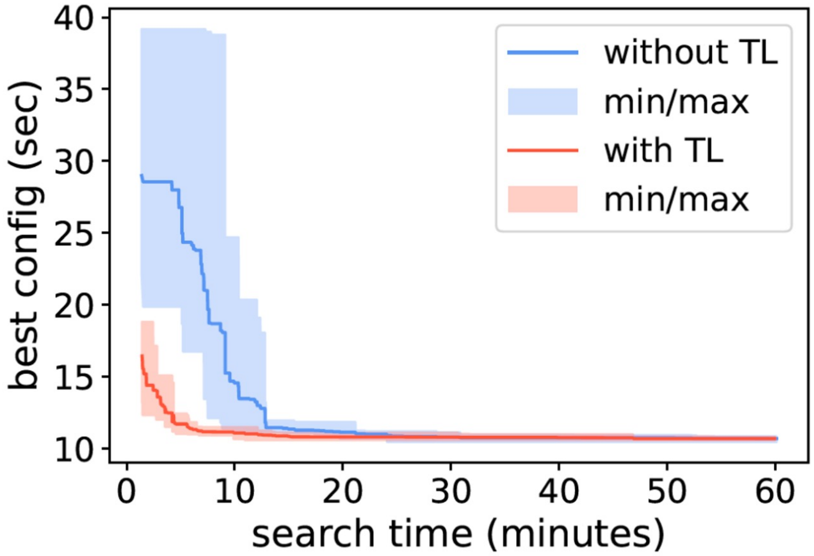
1. Initial: only the first step of the workflow, on 4 nodes per instance
   - 11 parameters
2. Full workflow: **2-steps workflow** on 4 nodes per instance
   - 16 parameters, w/ and w/o transfer-learning from setup 1
3. More parameters: 2-steps workflow on 4 nodes with **more parameters**
   - 20 parameters, w/ and w/o transfer-learning from setup 2
4. Full workflow with **8 nodes per instance**
   - 20 parameters, w/ and w/o transfer-learning from setup 3
5. Full workflow with **16 nodes per instance**
   - 20 parameters, w/ and w/o transfer-learning from setup 4

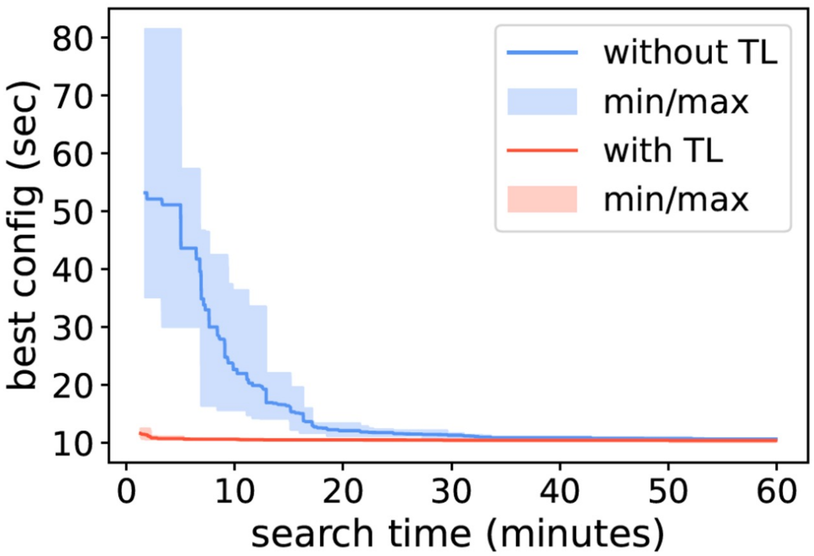
INITIAL EXPERIMENT



- Single-step workflow instances
- 4 nodes per instance
- 11 tuning parameters
- DeepHyper uses 128 nodes
- 32 instances evaluated in parallel
- Experiment repeated 5 times

TRANSFER-LEARNING
From small to larger search space

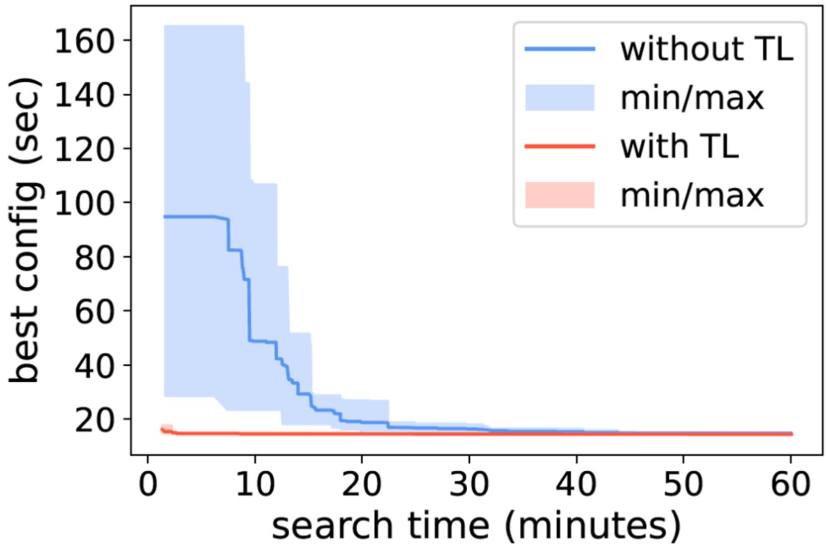




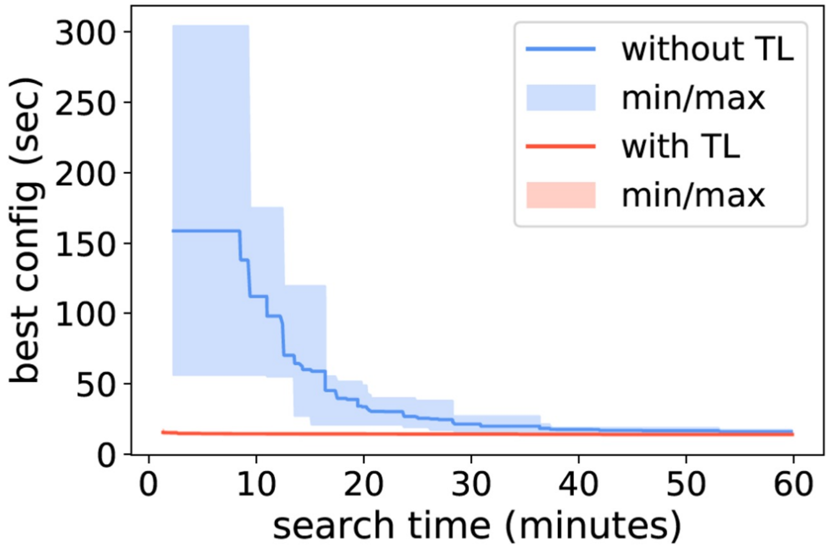
From 1-step to 2-step workflow
(11 to 16 parameters) on 4 nodes per instance

2-step workflow on 4 nodes per instance
From 16 to 20 parameters

TRANSFER-LEARNING
From small to larger instances



From 4 nodes to 8 nodes per instance
(20 parameters)



From 8 nodes to 16 nodes per instance
(20 parameters)

CONCLUSION

CONCLUSION: USE TRANSFER LEARNING!

Contributions

- We developed a **TVAE-based transfer-learning** technique
- We **integrated it into the DeepHyper** framework
- We enabled **autotuning of a HEP workflow** and its **storage service**

Results

- Transfer-learning enables finding better configurations faster
- Our framework outperforms state-of-the-art autotuning GPtune and HiPerBOt

Future work

- Provide a generic autotuning framework for Mochi-based storage services
- Handle complex service configuration (e.g., hierarchical/conditional parameter spaces)

THIS WORK IS IN PART SUPPORTED BY THE DIRECTOR, OFFICE OF ADVANCED SCIENTIFIC COMPUTING RESEARCH, OFFICE OF SCIENCE, OF THE U.S. DEPARTMENT OF ENERGY UNDER CONTRACT NO. DE-AC02-06CH11357; IN PART SUPPORTED BY THE EXASCALE COMPUTING PROJECT (17-SC-20-SC); AND IN PART SUPPORTED BY THE U.S. DEPARTMENT OF ENERGY, OFFICE OF SCIENCE, OFFICE OF ADVANCED SCIENTIFIC COMPUTING RESEARCH, SCIENTIFIC DISCOVERY THROUGH ADVANCED COMPUTING (SCIDAC) PROGRAM.

THANK YOU!