On the effectiveness of application-aware self-management for scientific discovery in volunteer computing systems

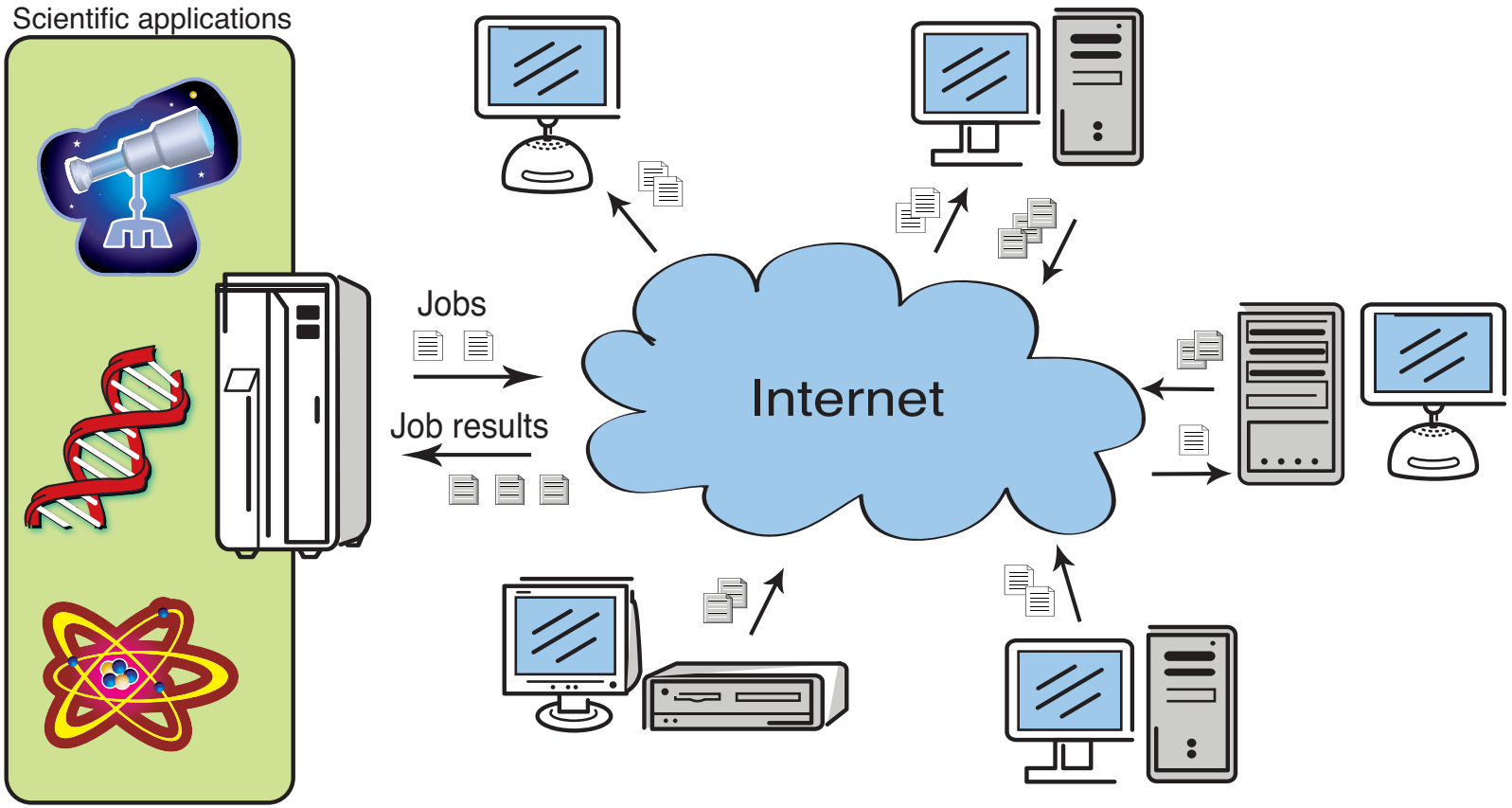
SC 2012

Trilce Estrada and Michela Taufer

University of Delaware

November 15, 2012

Volunteer Computing (VC)



Self-management in VC and in other distributed systems

Self-management

Refers to the capability of a system to reconfigure or adapt itself **without direct human intervention**.

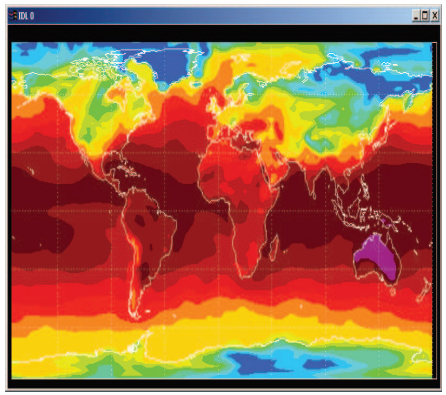
Self management in distributed systems provides:

- Resilience to resource volatility and workload changes.
- Minimum turnaround time of jobs and maximum throughput .

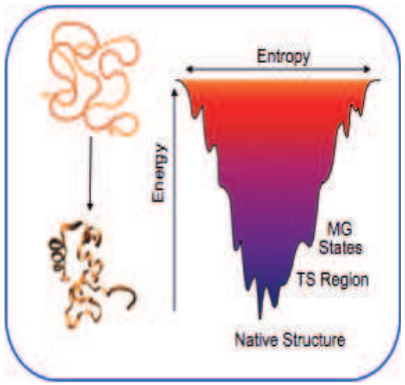
Diversity of goals in VC scientific applications

Astronomical applications: exploring as many regions of the sky as possible

Earthquake detection: returning readings as fast as possible



Building a global model for climate prediction: modeling accurately every single region



Biochemical applications: finding a single global accurate solution

Taxonomy of VC applications based on their goals:

- **Coverage-oriented** require higher throughput.
- **Latency-oriented** require reduced time to solution.
- **Accuracy-oriented** require accurate individual results.
- **Convergence-oriented** require finding a global solution regardless of throughput.

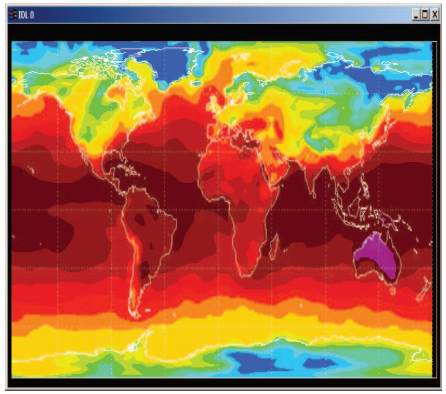
From system to application goals

Traditional performance metrics such as throughput and latency, cannot capture application-specific needs

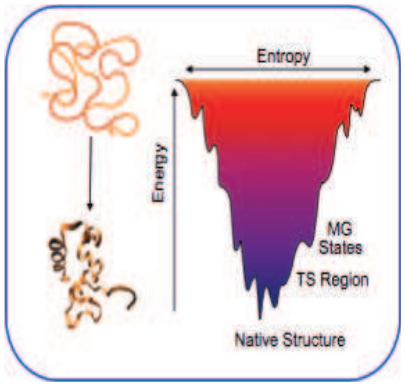
How do we provide self-management from the application perspective in a way that is general and covers all the different types of applications?

Astronomical applications: exploring as many regions of the sky as possible

Earthquake detection: returning readings as fast as possible



Building a global model for climate prediction: modeling accurately every single region



Biochemical applications: finding a single global accurate solution

Outline

1. **Introduction**
2. **Motivation**
   - Challenges of parametric scientific applications
   - Need for application-aware self-managed systems
3. **Method**
   - Towards a general application-aware self-managed VC system
   - Using KOtrees for parameter prediction and exploration
   - Integrated modular framework
4. **Evaluation**
   - Description
   - Results
5. **Discussion**
   - Related work
   - Conclusion

1 Introduction

2 Motivation
- Challenges of parametric scientific applications
- Need for application-aware self-managed systems

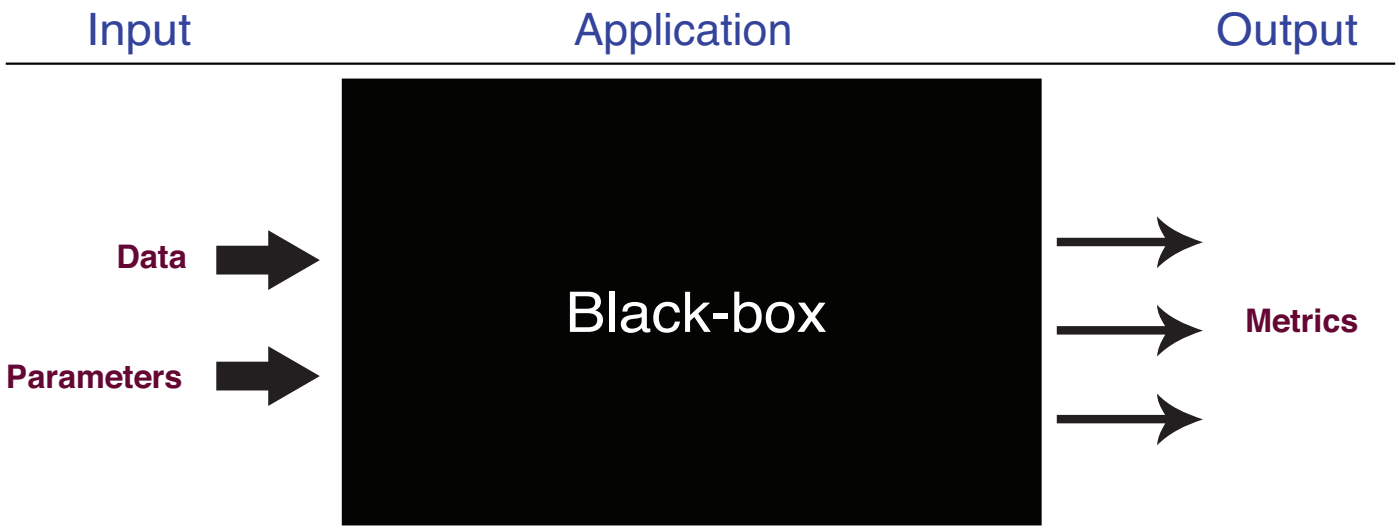
3 Method
- Towards a general application-aware self-managed VC system
- Using KOtrees for parameter prediction and exploration
- Integrated modular framework

4 Evaluation
- Description
- Results

5 Discussion
- Related work
- Conclusion

Parametric VC applications

The general case

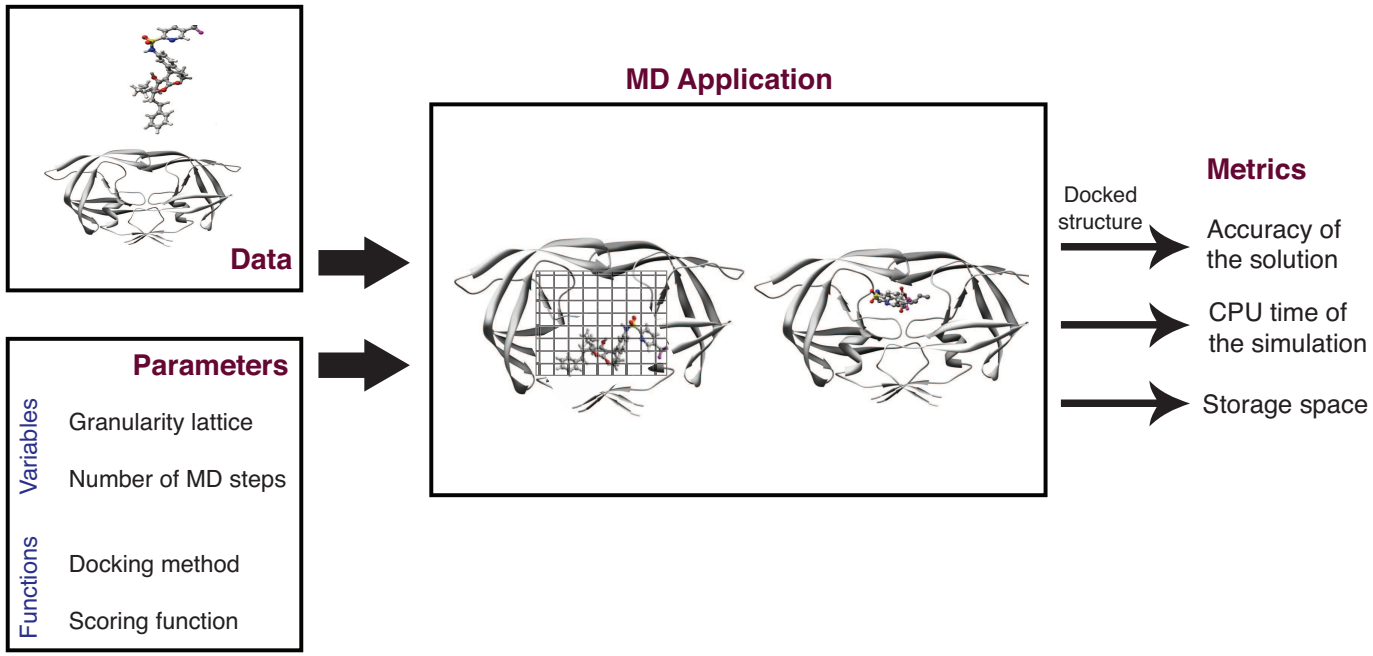


Scientific applications can be expressed as parametric functions

$$f(data, parameters) \rightarrow metrics$$

Parametric application

A protein-ligand docking application

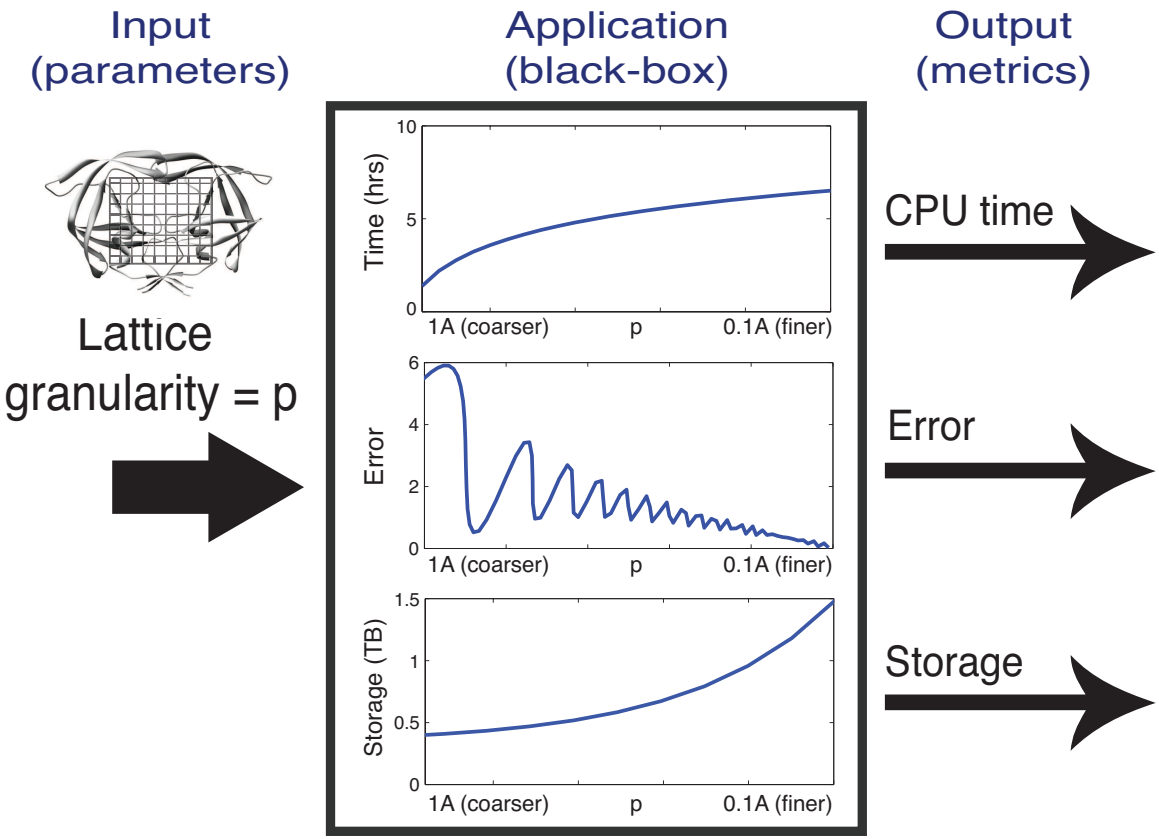


Scientific applications can be expressed as parametric functions

$$f(data, parameters) \rightarrow metrics$$

Parametric application

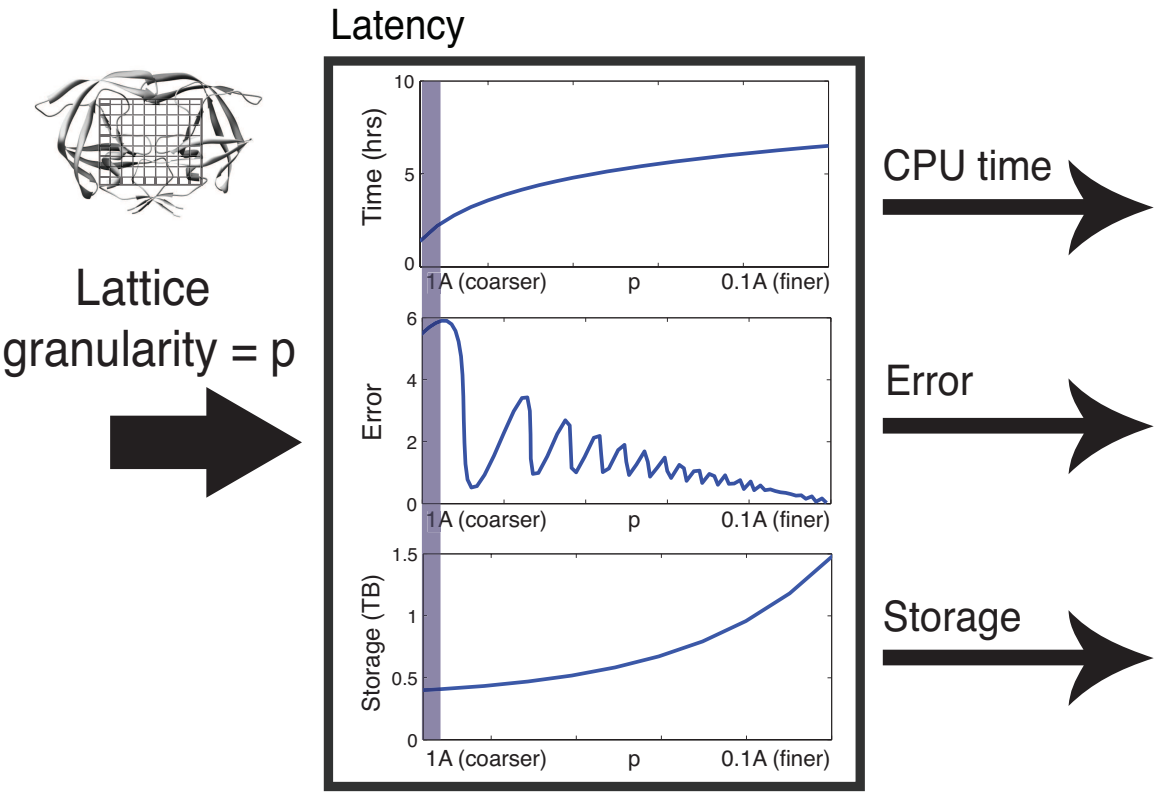
The general case: optimizing parameter selection



Parameters affect differently the application metrics.

Parametric application

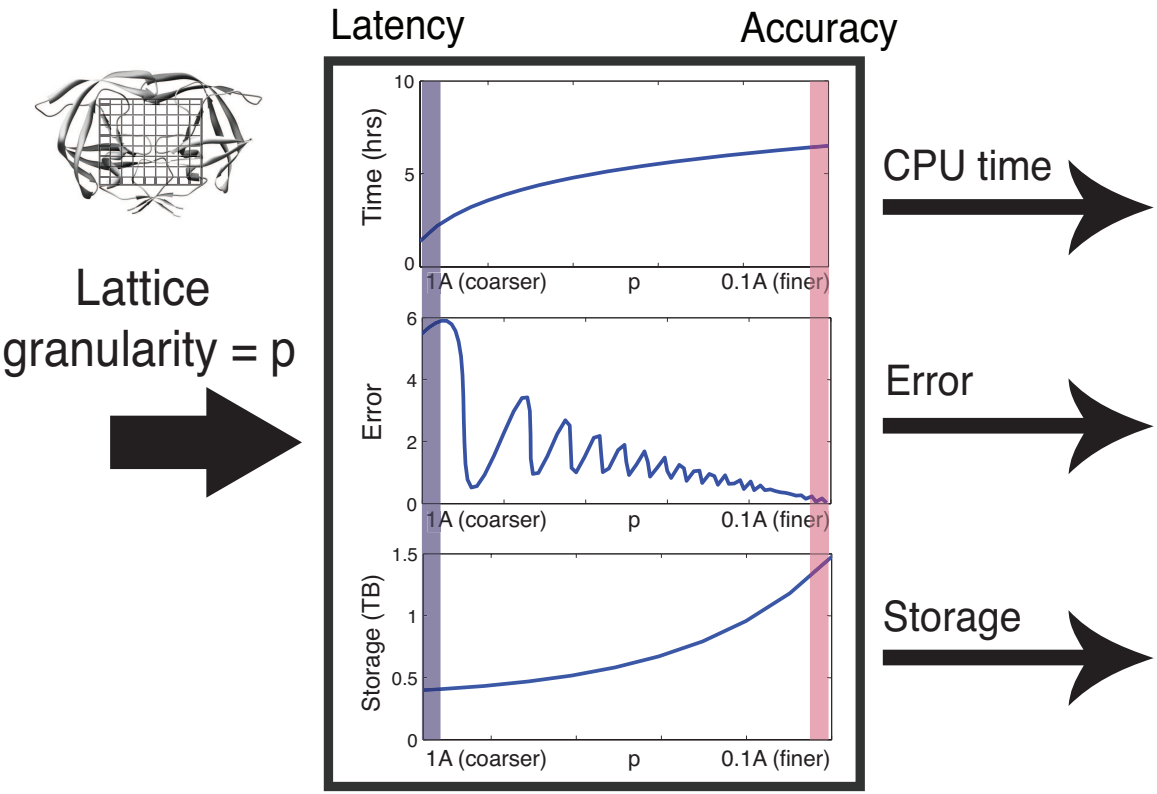
The general case: optimizing parameter selection



Finding optimal parameter values depends on application-specific goals.

Parametric application

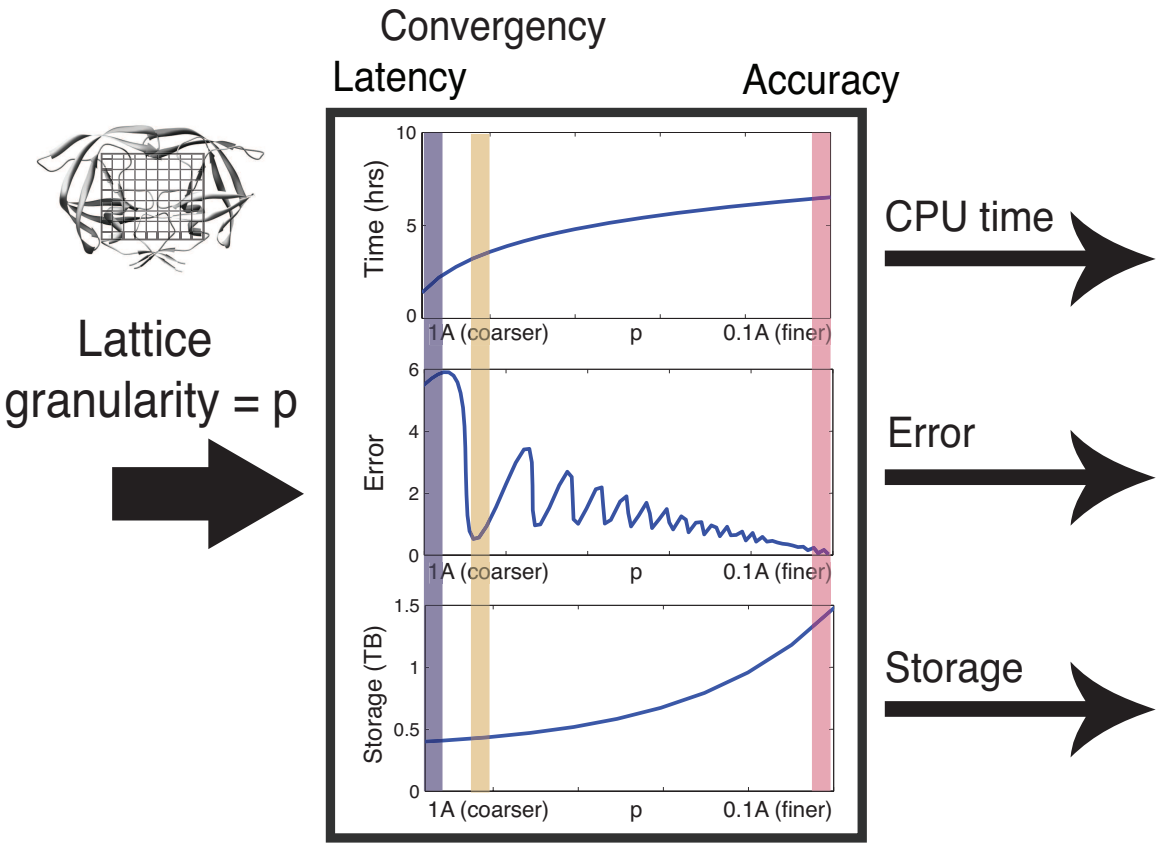
The general case: optimizing parameter selection



Finding optimal parameter values depends on application-specific goals.

Parametric application

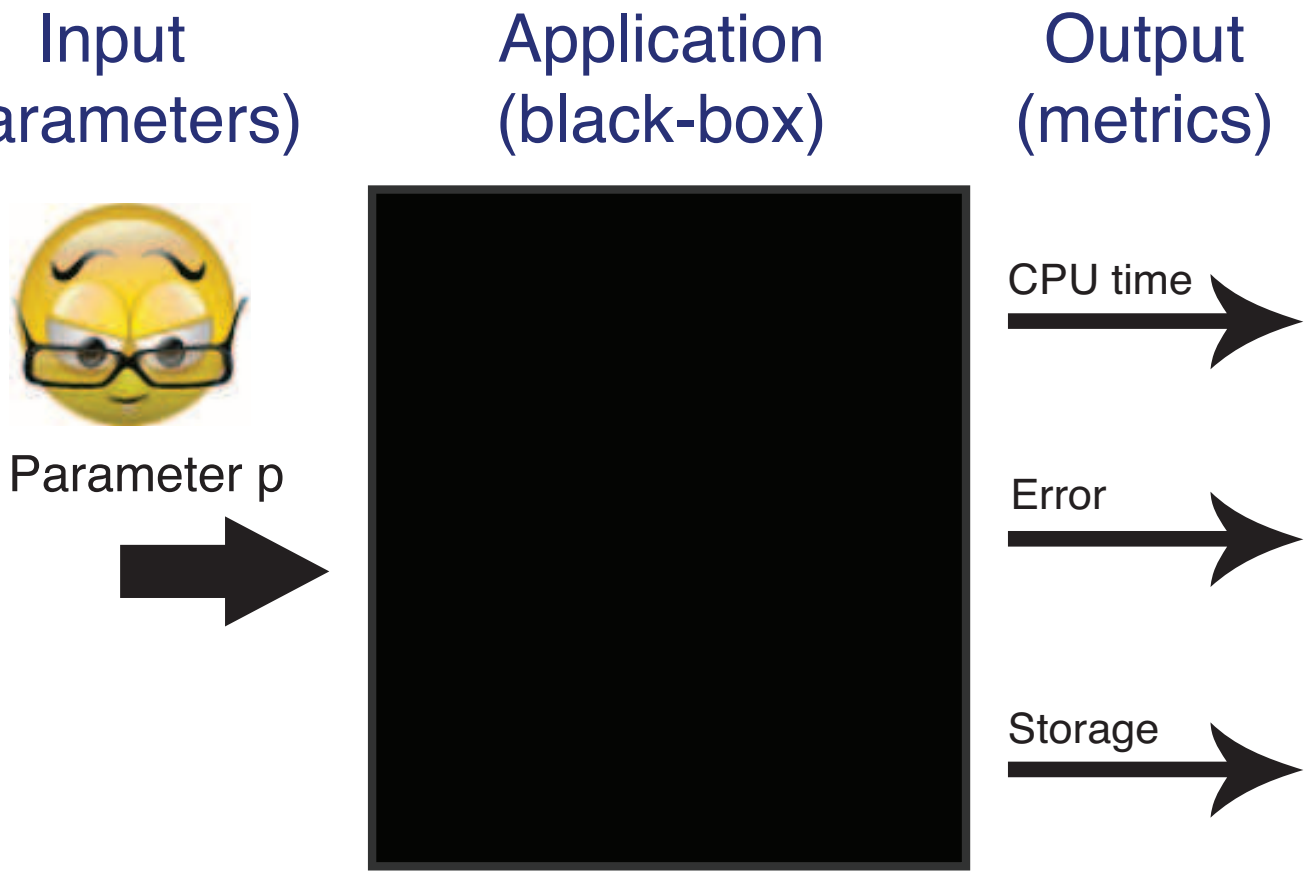
The general case: optimizing parameter selection



Finding optimal parameter values depends on application-specific goals.

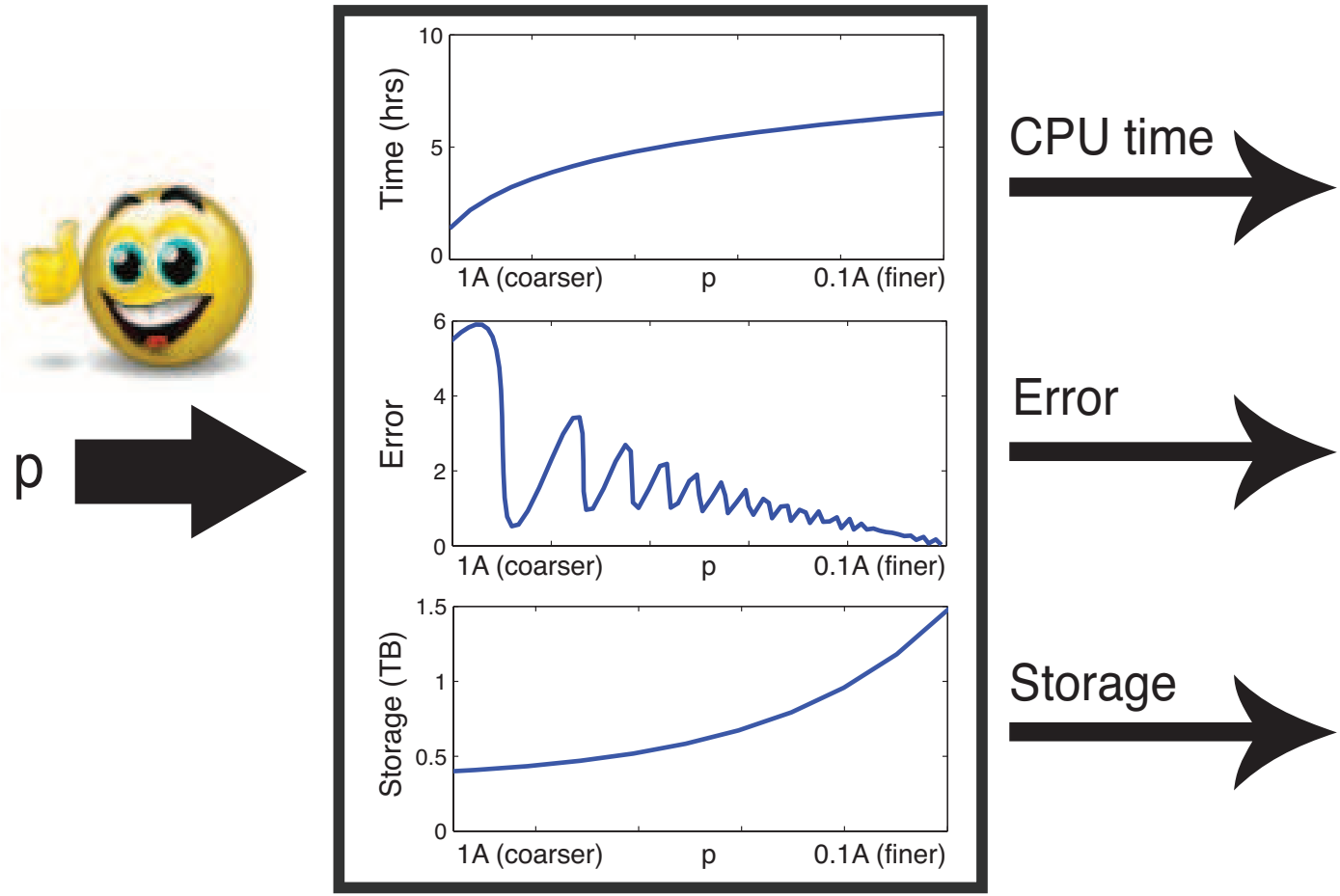
Manual parameter reconfiguration

Requires an expert analysing the application, continuously monitoring results, and tuning parameters



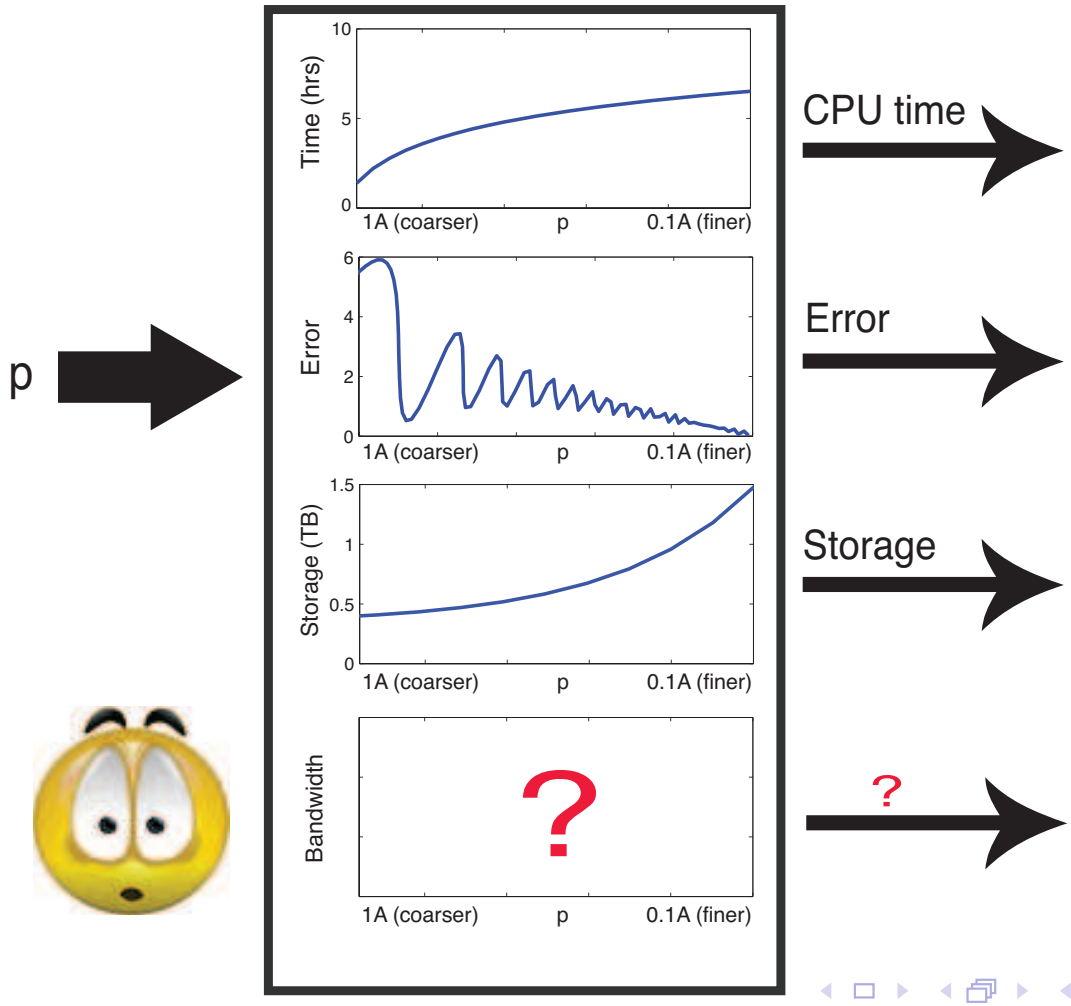
Manual parameter reconfiguration

It could be done for the simplest case, but ...



Manual parameter reconfiguration

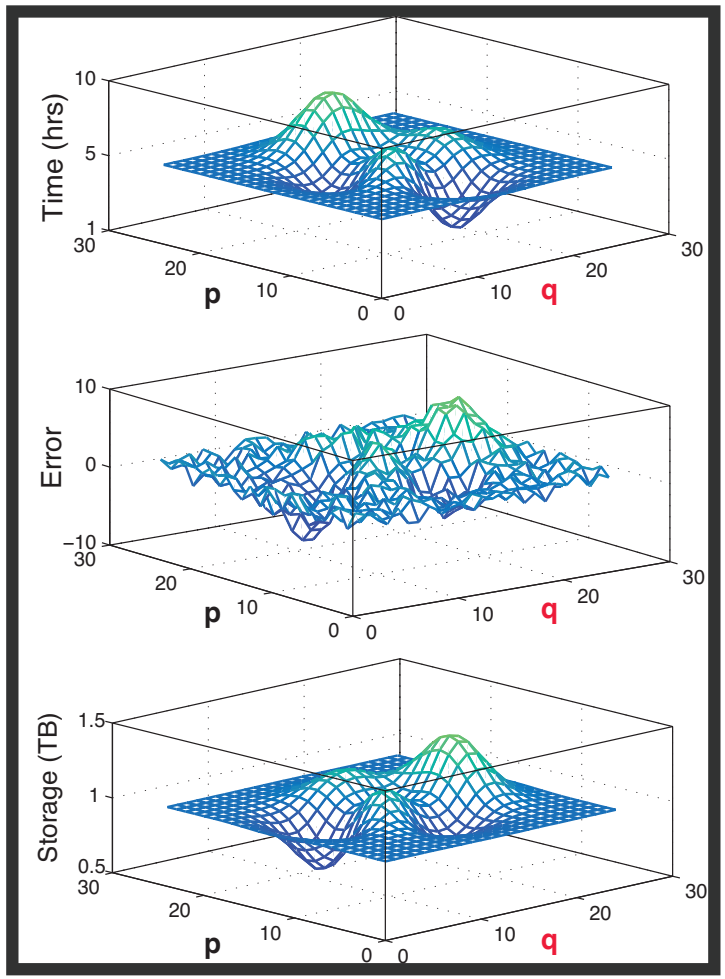
What if we need to predict additional metrics?



Manual parameter reconfiguration

What if we need to include additional parameters?

p, q



CPU time

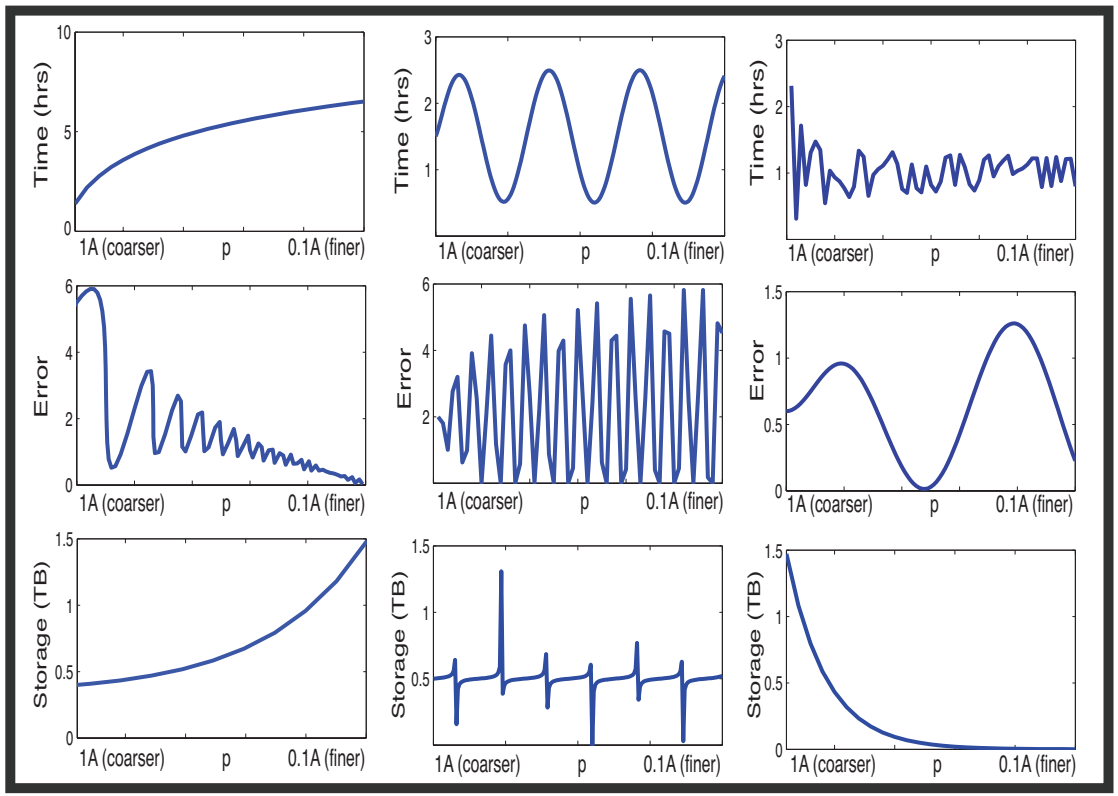
Error

Storage

Manual parameter reconfiguration

What if we need to change or add software modules?

p



CPU time

Error

Storage

Need for application-aware self-managed VC systems

Manual reconfiguration is error-prone, inefficient, and promotes resource wasting.

Thus, we need to provide self-management from the application perspective.

Need for application-aware self-managed VC systems

Manual reconfiguration is error-prone, inefficient, and promotes resource wasting.

Thus, we need to provide self-management from the application perspective.

Definition

We define **application-aware self-management** as the ability of a system to guarantee the accomplishment of application-specific goals without direct human intervention.

1 Introduction

2 Motivation
- Challenges of parametric scientific applications
- Need for application-aware self-managed systems

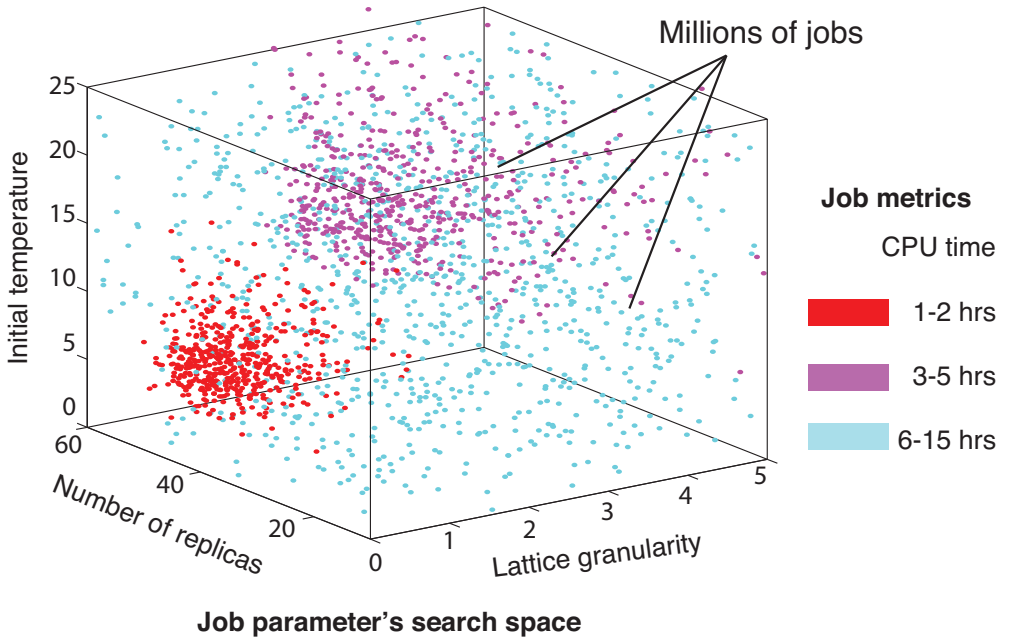
3 Method
- Towards a general application-aware self-managed VC system
- Using KOtrees for parameter prediction and exploration
- Integrated modular framework

4 Evaluation
- Description
- Results

5 Discussion
- Related work
- Conclusion

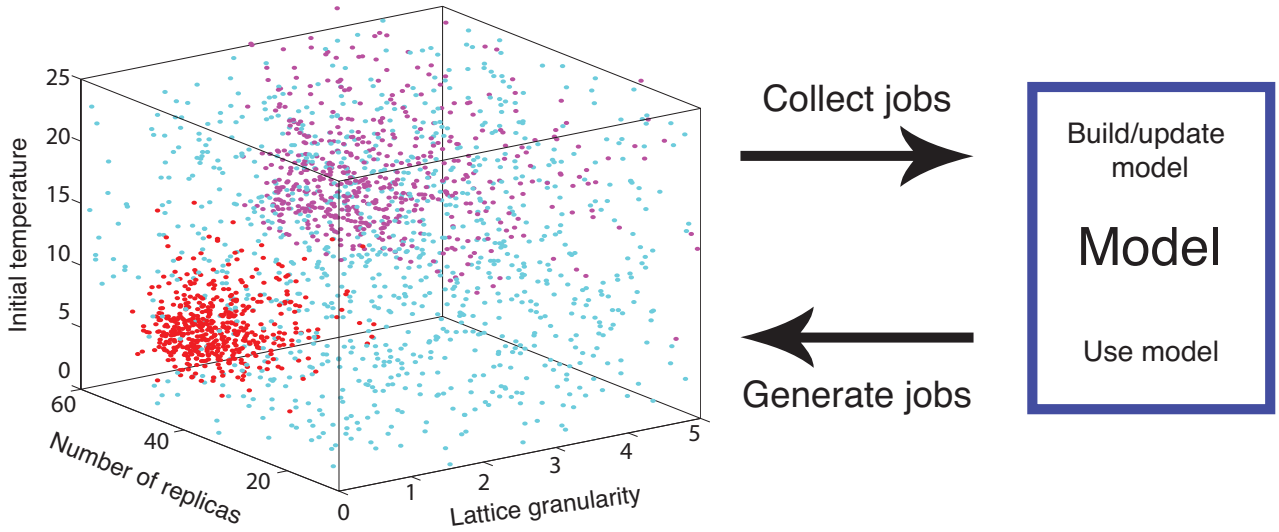
Requirements of a self-managed VC system

Example: A replica exchange application with 3 parameters



- A job x_i is a tuple of N parameters.
- When a job is collected, we obtain a tuple y_i with M metrics representing measures relevant to the application.

Requirements of a self-managed VC system



Method requirements:

1. Building and updating the model at runtime
2. Making predictions of up to M metrics in near real time
3. Learning from observed data in one pass
4. Identifying sets of parameter combinations that can advance the application goal

Matching requirements to existing algorithms

1. Building and updating the model at runtime
   - Lazy learning and nearest neighbors
   - Clustering
   - Neural networks
   - Bayesian learning
   - Decision trees
   - Hoeffding trees

Matching requirements to existing algorithms

1. Building and updating the model at runtime
2. Making predictions of up to M metrics in near real time

- Lazy learning and nearest neighbors
- Clustering
- Neural networks
- Bayesian learning
- Decision trees
- Hoeffding trees

Matching requirements to existing algorithms

1. Building and updating the model at runtime
2. Making predictions of up to M metrics in near real time
3. Learning from observed data in one pass

- Lazy learning and nearest neighbors
- Clustering
- Neural networks
- Bayesian learning
- Decision trees
- Hoeffding trees

Matching requirements to existing algorithms

1. Building and updating the model at runtime
2. Making predictions of up to M metrics in near real time
3. Learning from observed data in one pass
4. Identifying sets of parameter combinations that can advance the application goal

- Lazy learning and nearest neighbors
- Clustering
- Neural networks
- Bayesian learning
- Decision trees
- Hoeffding trees

Knowledge organization trees (KOTrees)

Our contribution:

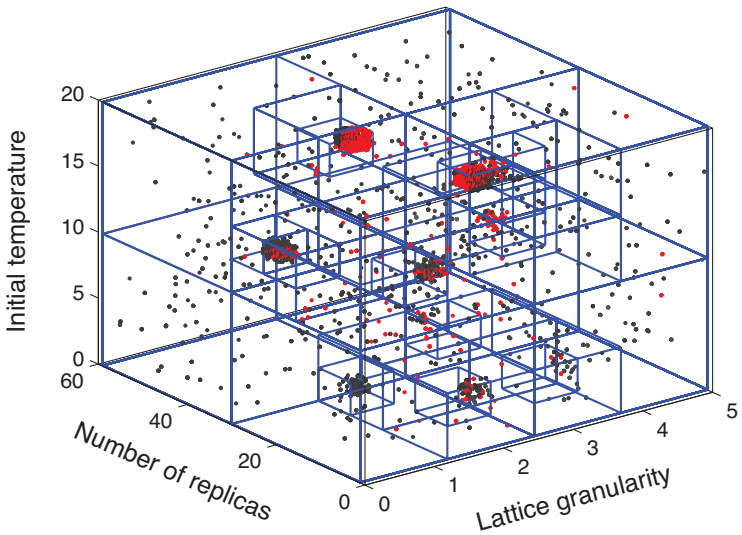
A statistical data structure, in the form of a tree, that enables prediction of multiple application **metrics** and exploration of the multi-dimensional **parameter** space effectively, while being built incrementally at runtime.

Our data structure/algorithm can:

1. Learn from observed data in one pass
2. Build and update the model at runtime
3. Make predictions of up to M metrics in near real time
4. Identify sets of parameter combinations that can advance the application goal

Knowledge organization trees (KOTrees)

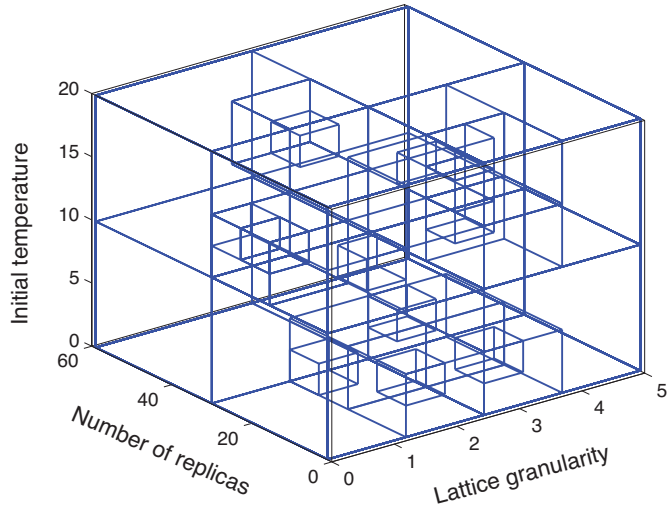
Partition of parameter space



Collect jobs

Generate jobs

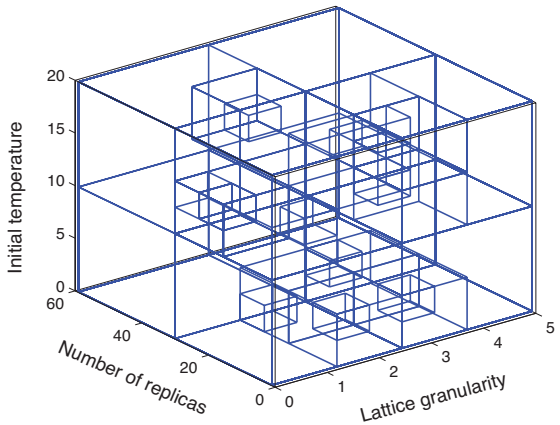
Statistical structure



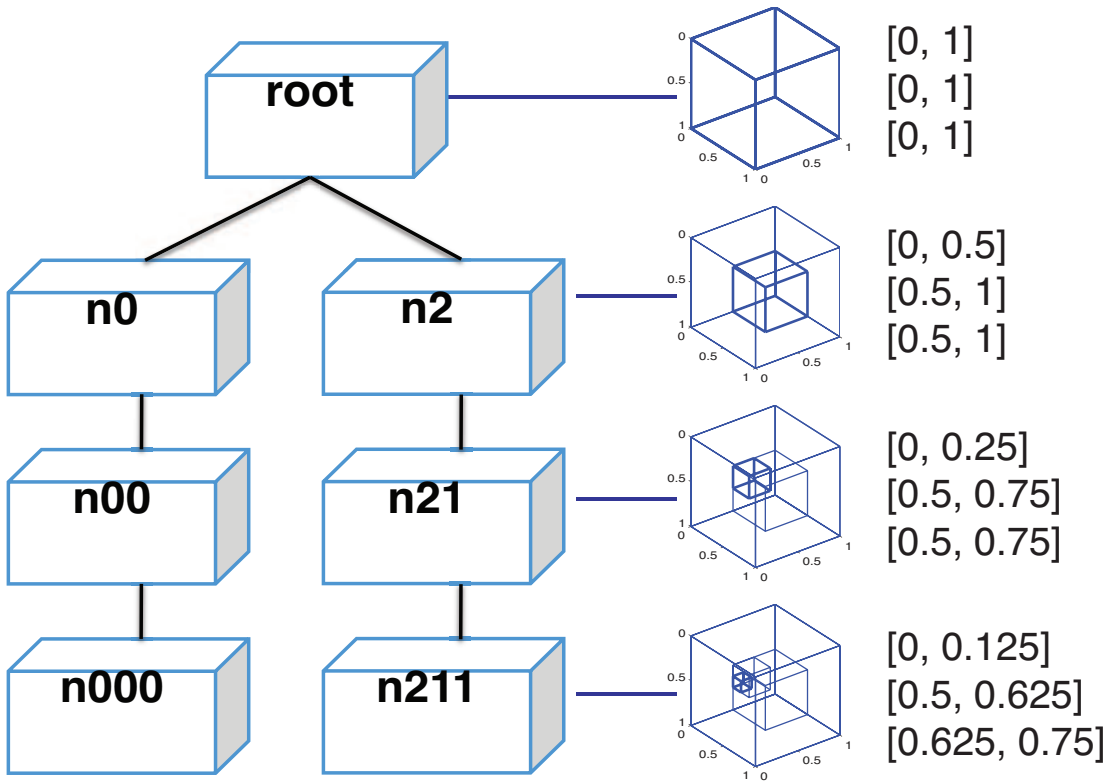
- We partition the parameter space recursively and build a tree-like structure of statistical knowledge.
- We use the statistical knowledge embedded in this structure to drive job generation

Knowledge organization trees (KOTrees)

Tree organization

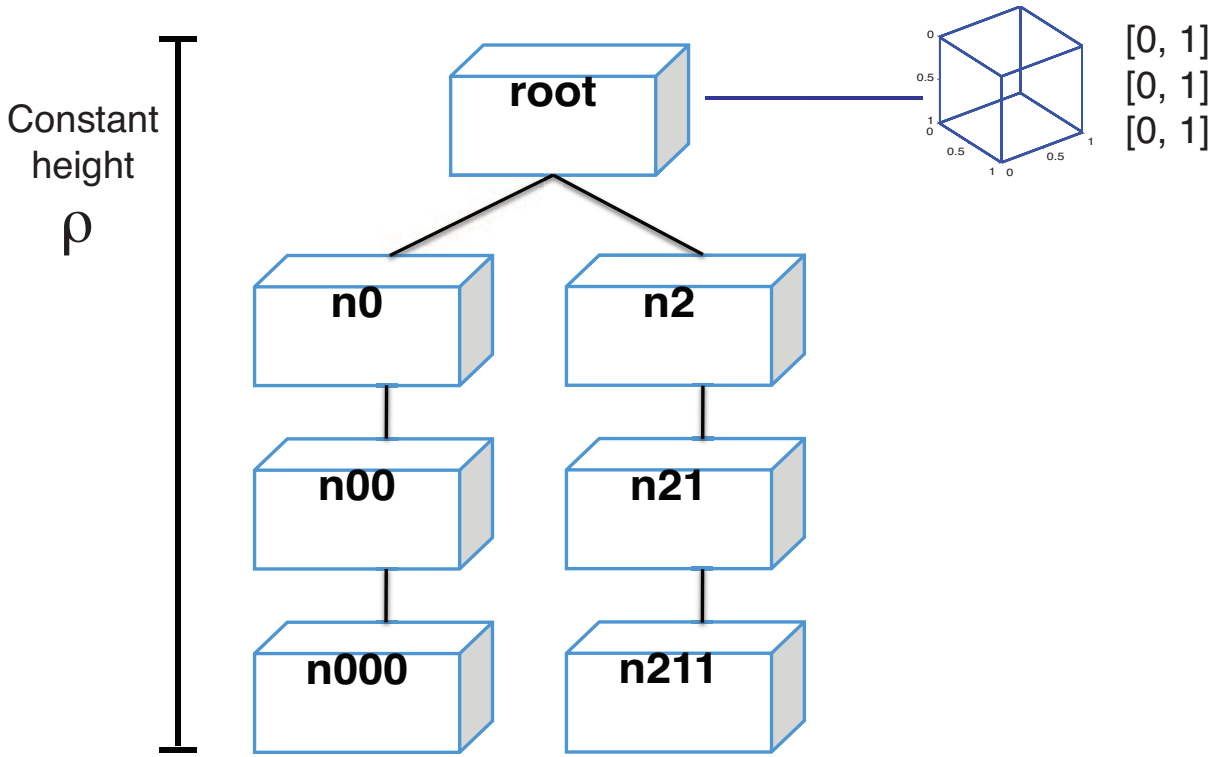


Every node corresponds to an hypercube in the N-dimensional space of parameters.



Knowledge organization trees (KOTrees)

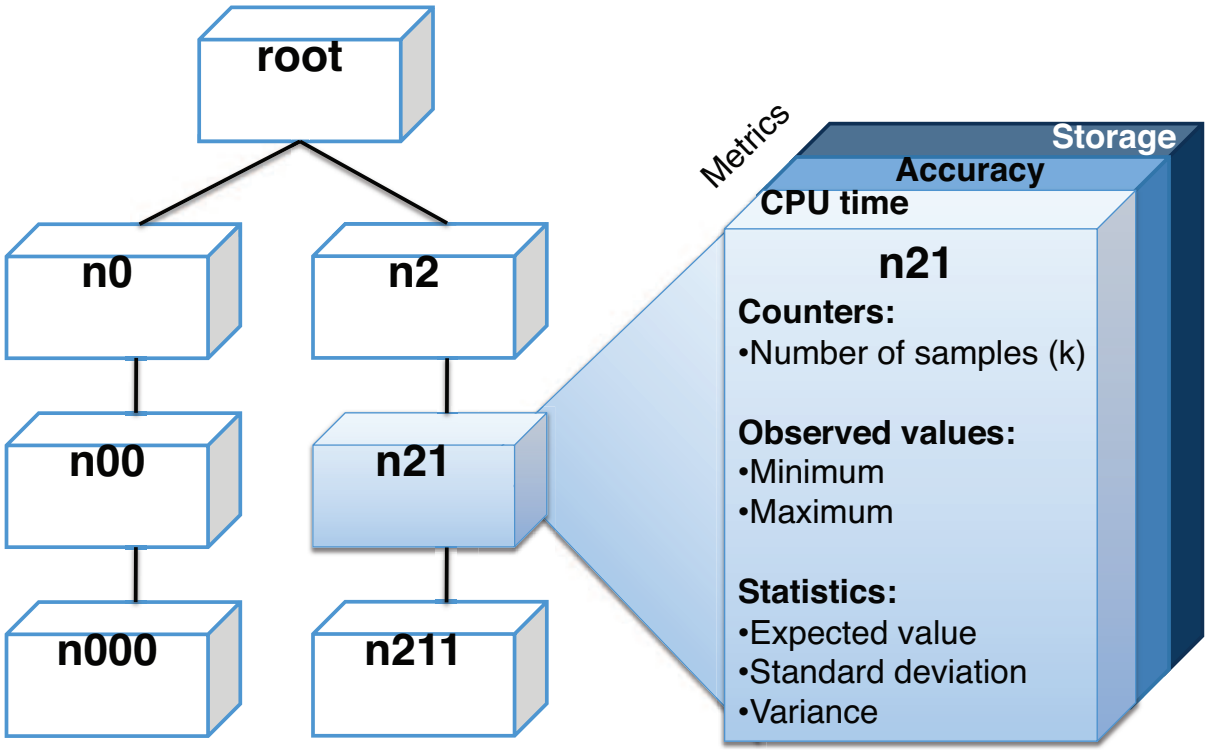
Tree organization



KOTree is parametric and requires that the user inputs the height of the tree (ρ) and N sets of parameter ranges.

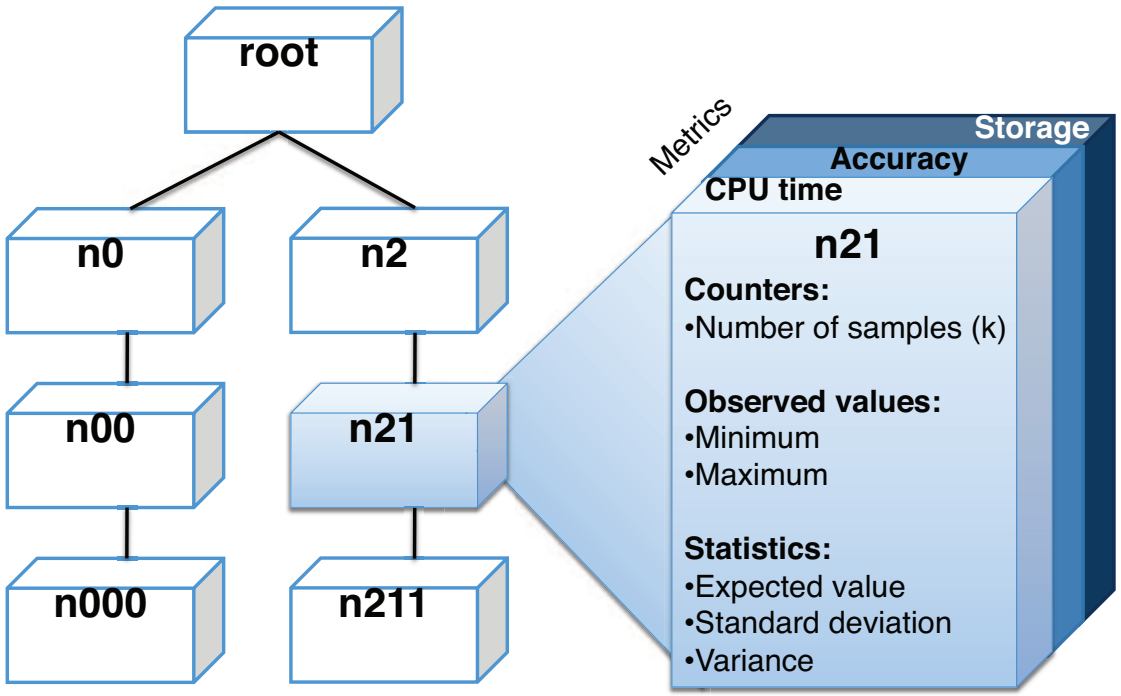
Knowledge organization trees (KOTrees)

Tree organization



Every node has a set of statistics per metric

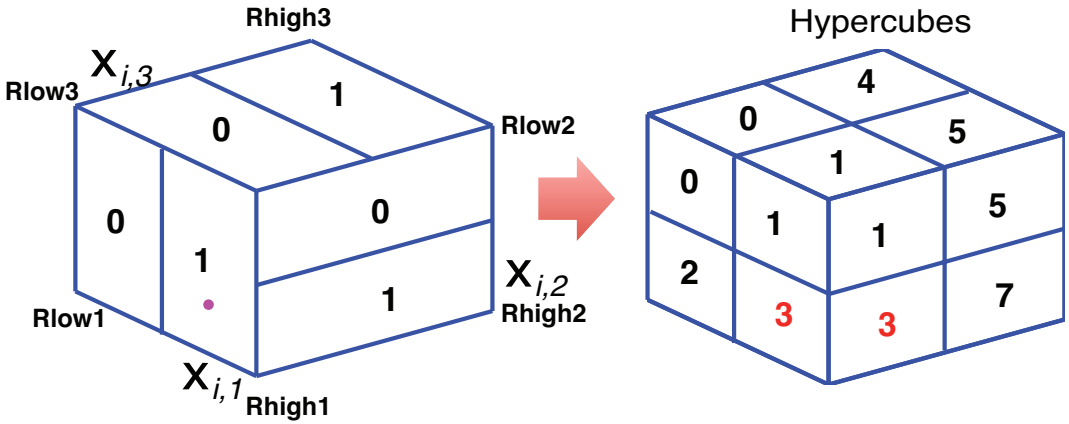
Requirement 1: One-pass learning



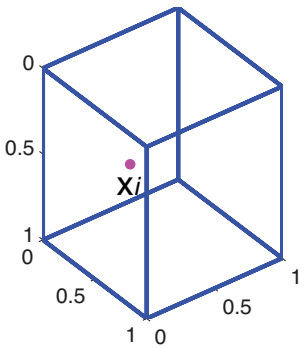
We use the Welford algorithm for the running variance and mean of each node. This allows us to aggregate information of the samples without actually storing them.

Requirement 2: Building a KOTree at run time

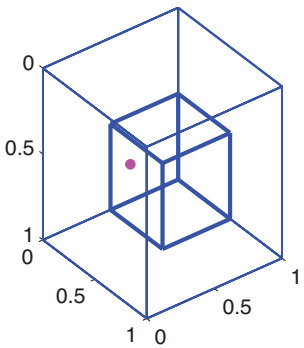
Calculating the path of a job in the tree



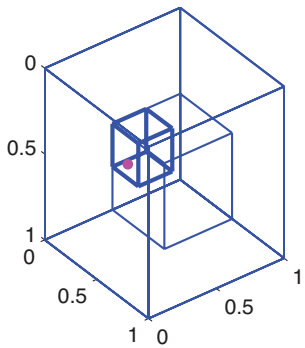
$$id_{hc}: 2^0\delta(x_{i,1}) + 2^1\delta(x_{i,2}) + 2^2\delta(x_{i,3}) = 1\,(1) + 2\,(1) + 4\,(0) = 3$$



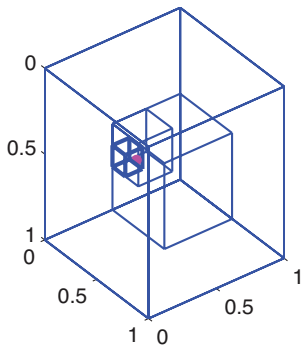
key: "602"



id_{hc1}: 6



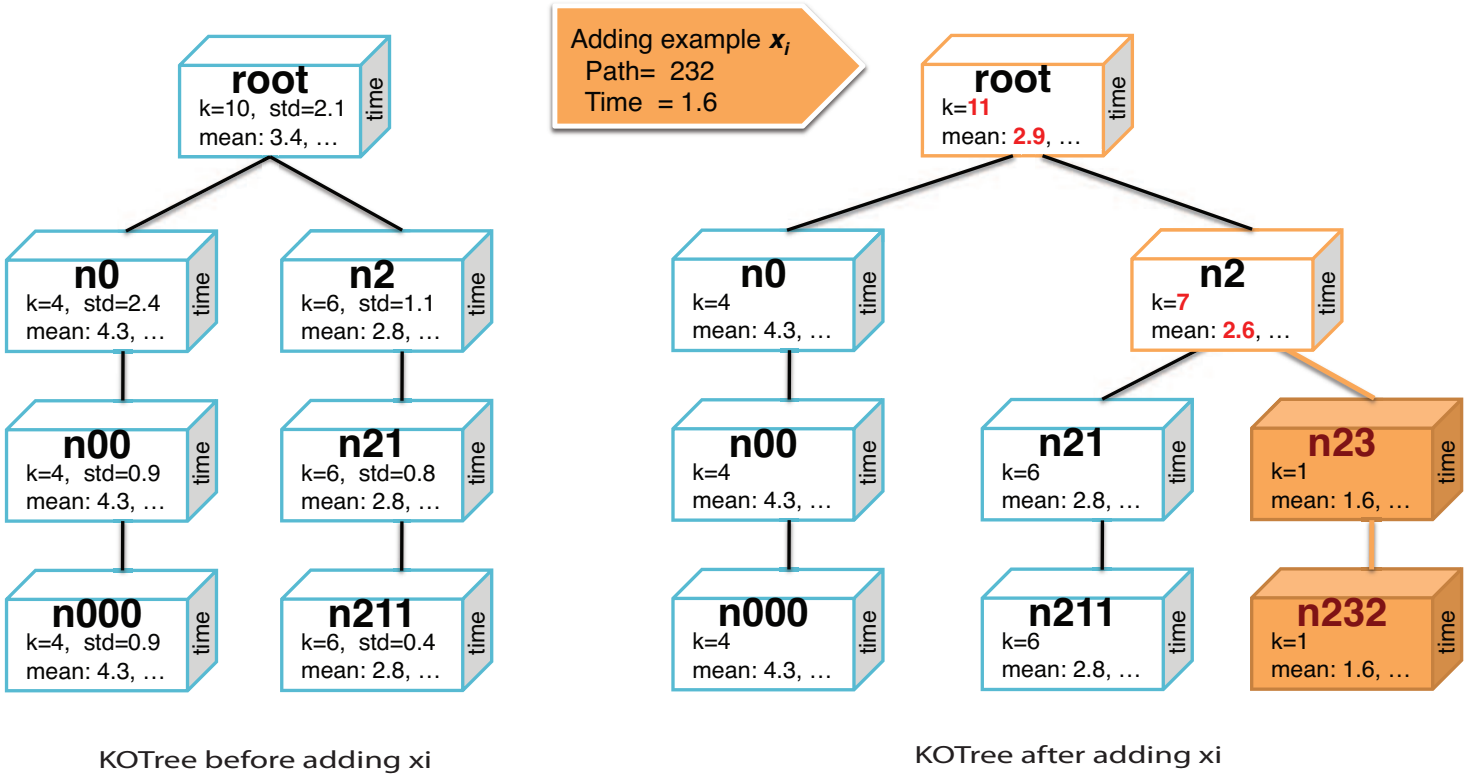
id_{hc2}: 0



id_{hc3}: 2

Requirement 2: Building a KOTree at run time

Online training

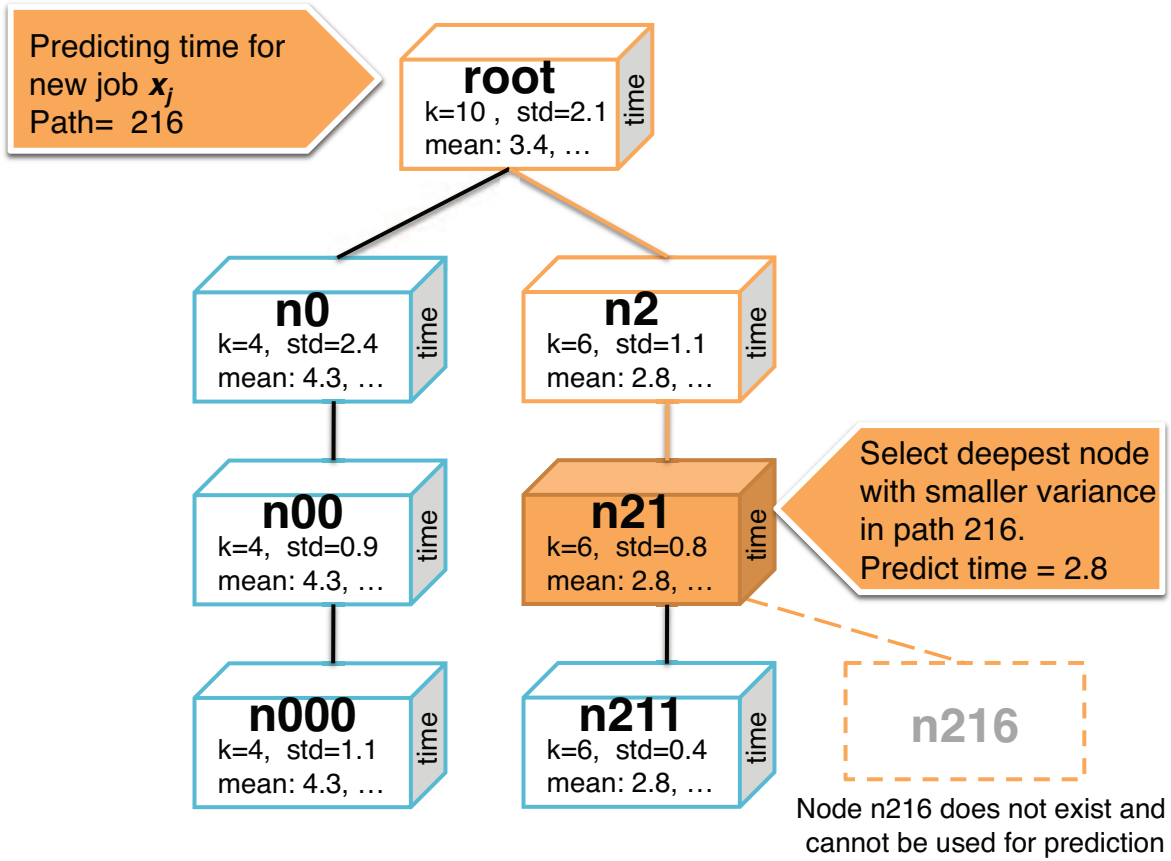


KOTree before adding xi

KOTree after adding xi

The computational cost of updating a tree of height ρ with a new sample is $O(\rho + 1) \approx O(1)$

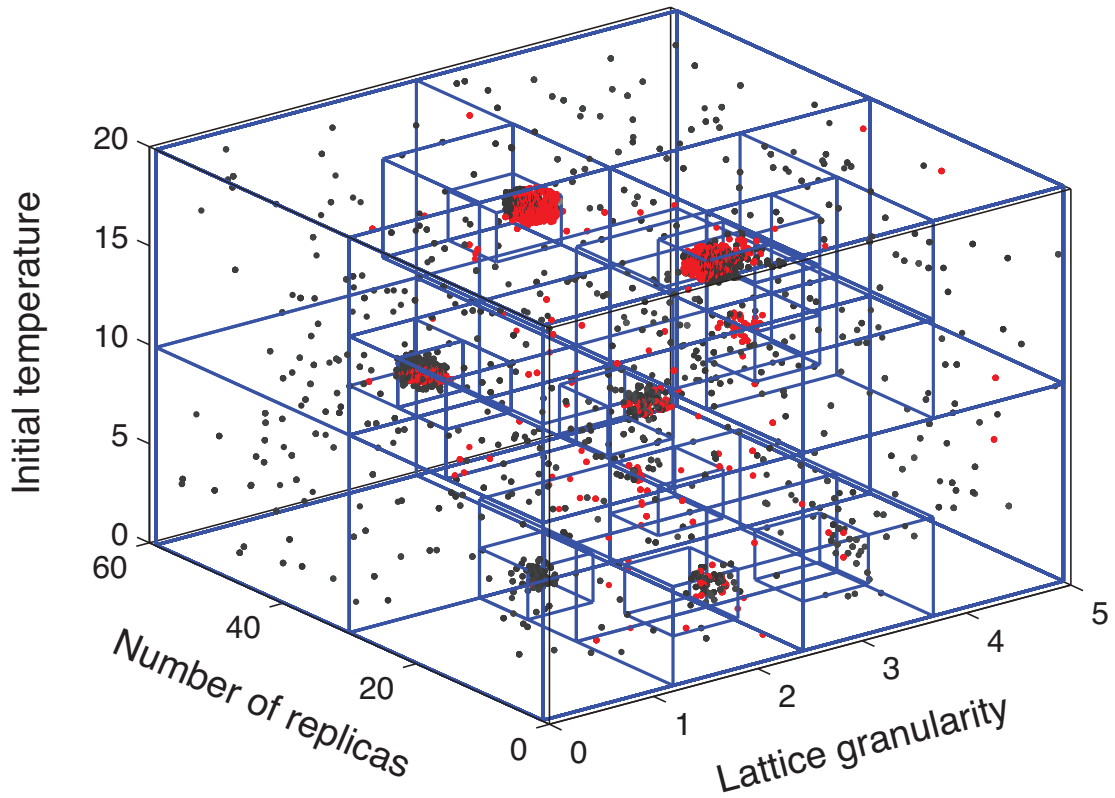
Requirement 3: Metric prediction in near-real time



The computational cost of making a prediction for a tree of height ρ is $O(\rho + 1) \approx O(1)$

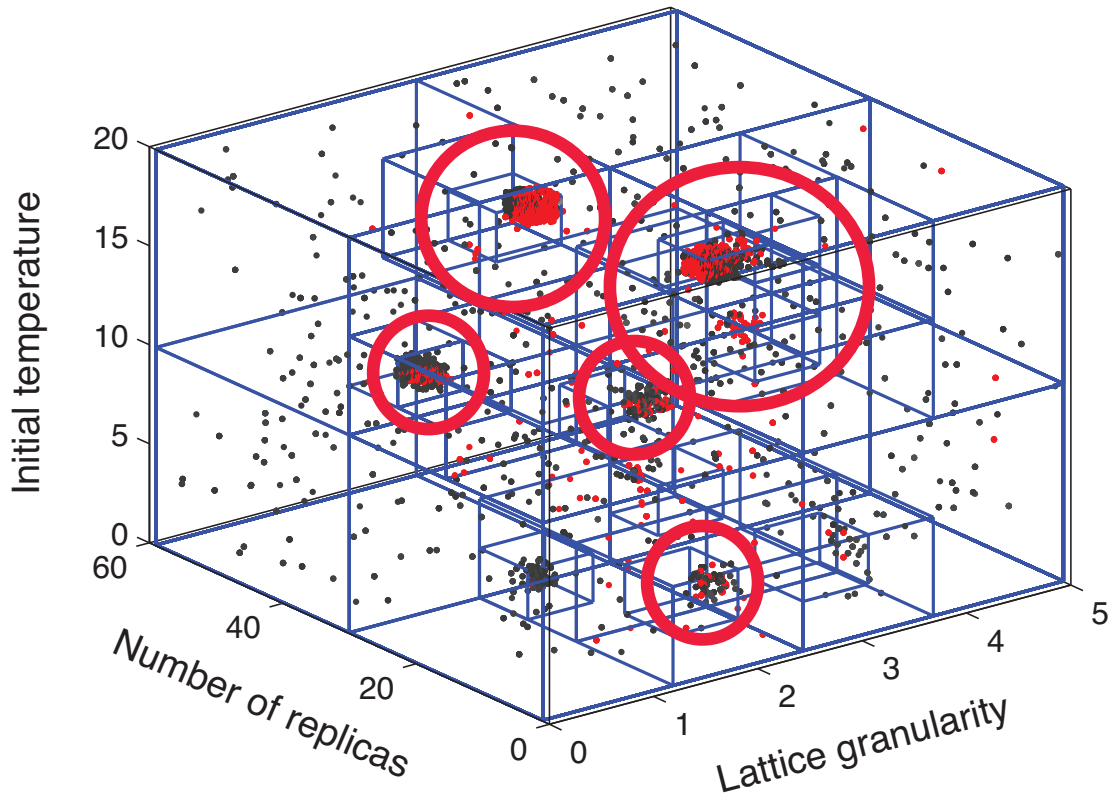
Requirement 4: Parameter space exploration

The statistical structure in KOTree can find sweet-spots of parameters whose jobs can potentially advance application goals



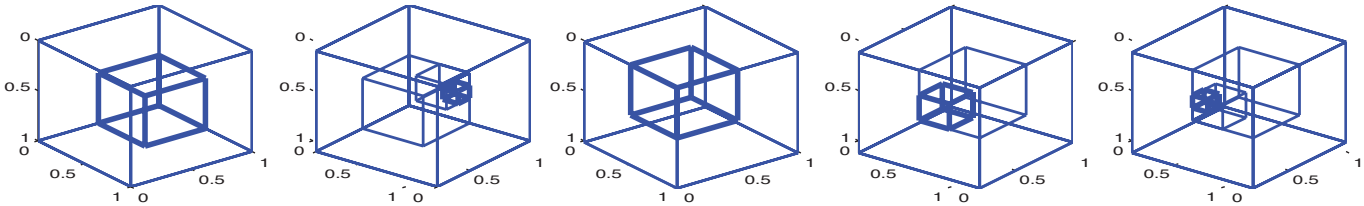
Requirement 4: Parameter space exploration

The statistical structure in KOTree can find sweet-spots of parameters whose jobs can potentially advance application goals



Requirement 4: Parameter space exploration

Selection of hypercubes to be explored



1/1.0 + 1/1.0 + 1/1.2 + 1/1.4 + 1/1.6 = 4.16

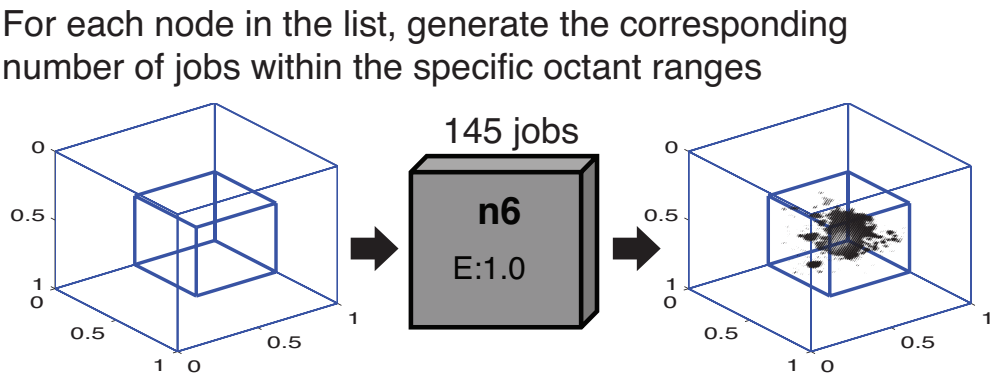
n6	n657	n1	n62	n120
E:1.0	E:1.0	E:1.2	E:1.4	E:1.6
100/(1*4.16)	100/(1*4.16)	100/(1.2*4.16)	100/(1.4*4.16)	100/(1.6*4.16)
24	24	20	17	15

100 Jobs

- Keep a sorted list of nodes whose values optimize application goal
- Generate jobs within the node ranges proportionately to the node score

Requirement 4: Parameter space exploration

Generation of job parameters within hypercubes



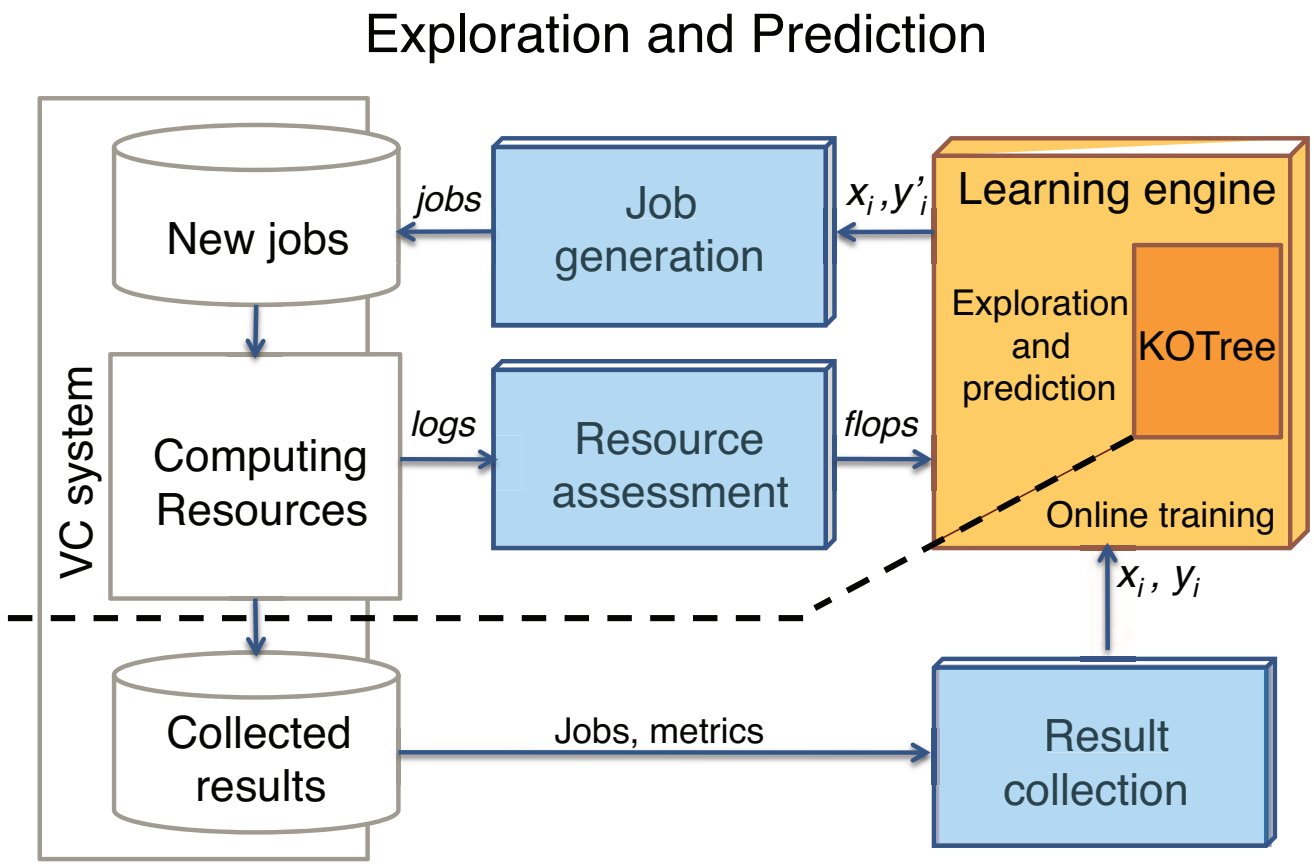
For each node, we generate job parameters in one of three ways:

Top promotes exploitation of a parameter that has proved to advance application goal

Uniform promotes exploration of new regions and avoids getting trapped in local minima

Chebyshev promotes a more extensive exploration near to the edges of an hypercube

Modular framework



More implementation details on the paper

1 Introduction

2 Motivation
- Challenges of parametric scientific applications
- Need for application-aware self-managed systems

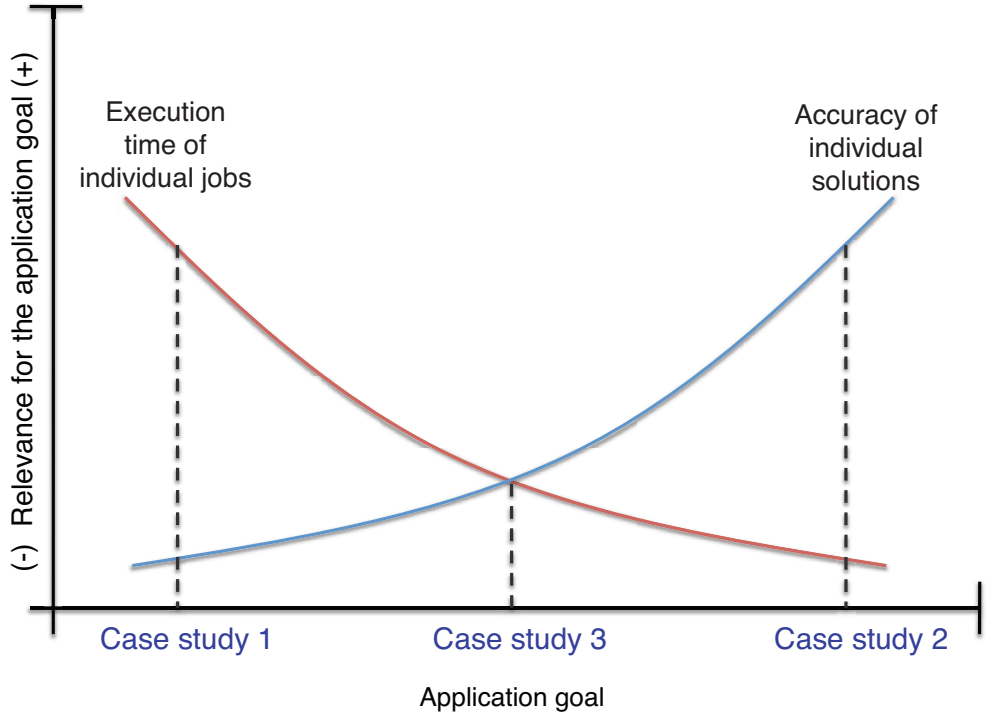
3 Method
- Towards a general application-aware self-managed VC system
- Using KOtrees for parameter prediction and exploration
- Integrated modular framework

4 **Evaluation**
- Description
- Results

5 Discussion
- Related work
- Conclusion

Evaluation

3 case studies



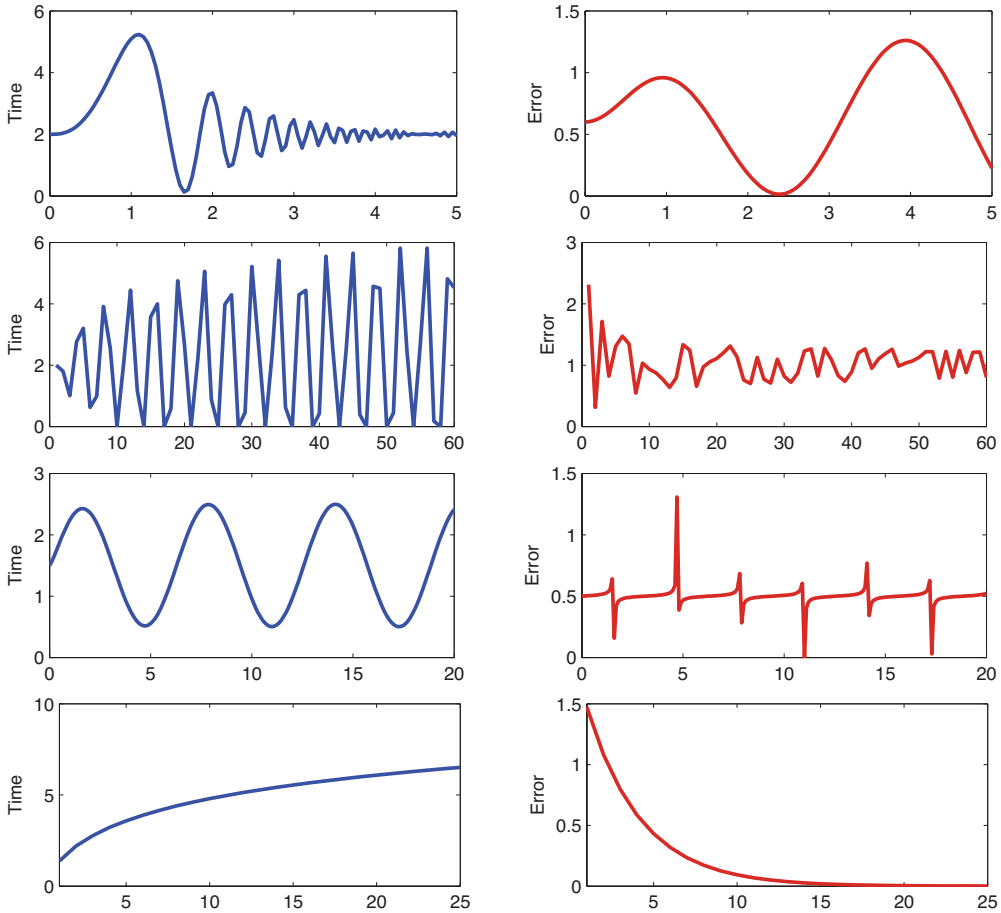
Case study 1 Assumes a latency-oriented application.

Case study 2 Assumes an accuracy-oriented application.

Case study 3 Assumes a convergence-oriented application.

Evaluation

14 implementation scenarios



Using 4 building blocks per metric (CPU time and error), we constructed 14 functions representing 14 different application implementations with 1 to 4 parameters each

Evaluation

Comparing KOtrees vs. other methods

KOTrees

KOM Generation of parameters per job using a KOTree driven by minimum values.

KOE Generation of parameters per job using a KOTree driven by expected values.

Other

RND Generation of parameters per job using a random value within specified ranges per parameter.

SAN Generation of parameters per job using a simulated annealing approach

Evaluation

Experimental set-up

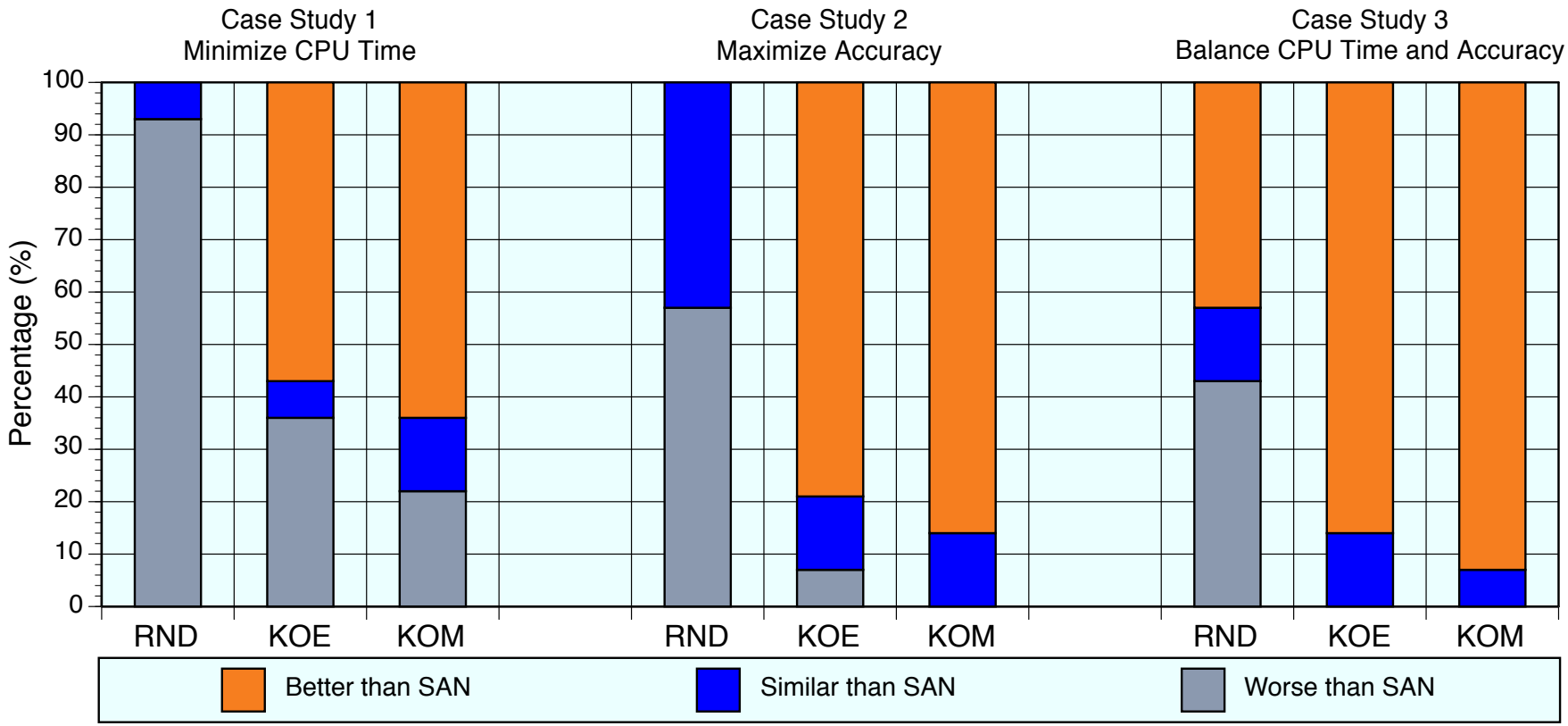
- BOINC server (version 6.11.1) - default scheduling policy and default daemons for generation and validation of jobs.
- EmBOINC (version v.1.2)
- Same set of 12,470 hosts obtained from traces of Docking@Home
- 168 simulated hours (1 week).

Total:

3 case studies * 14 scenarios * 4 algorithms per scenario * 5 simulations per algorithm = 840 simulations

Comparison of scenarios

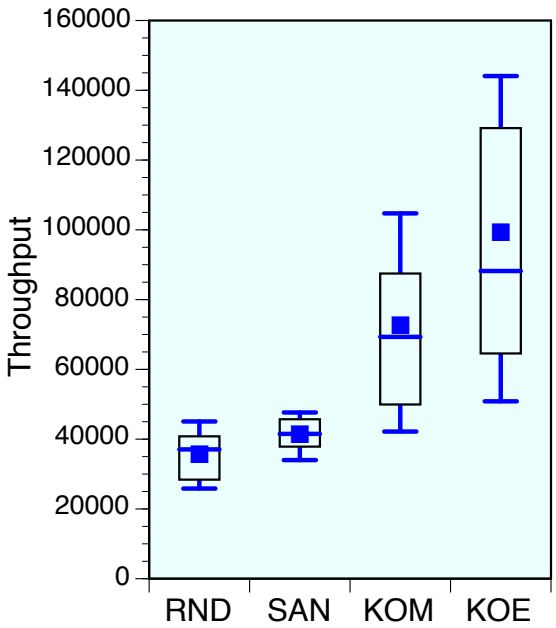
Kolmorgorov-Smirnov test for $dist(X) < dist(SAN)$, where X is *RND*, *KOM*, and *KOE* respectively.



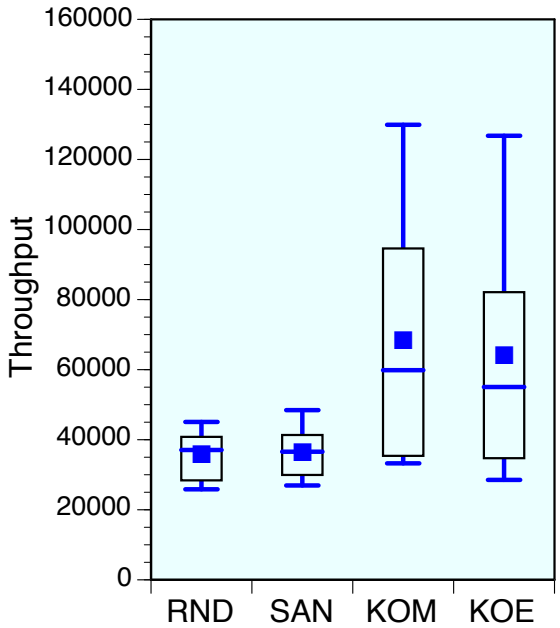
Comparison of throughput

Normalized throughput with respect to *SAN*.

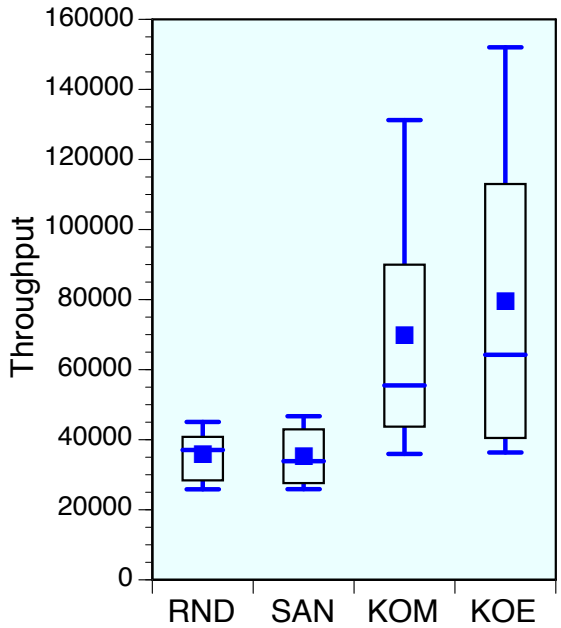
Case Study 1



Case Study 2

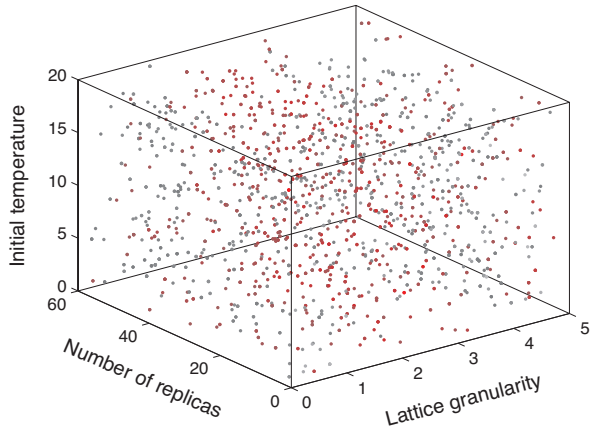


Case Study 3

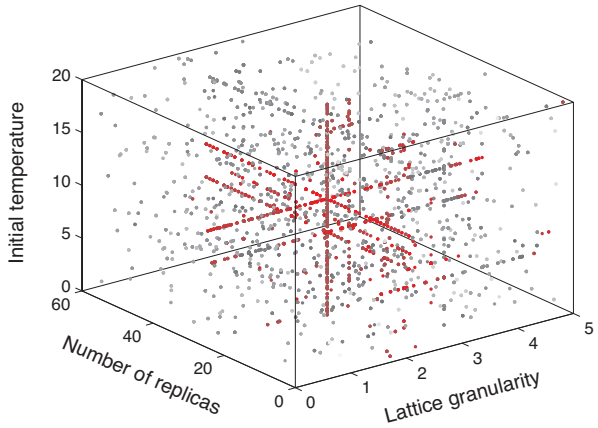


Analysis

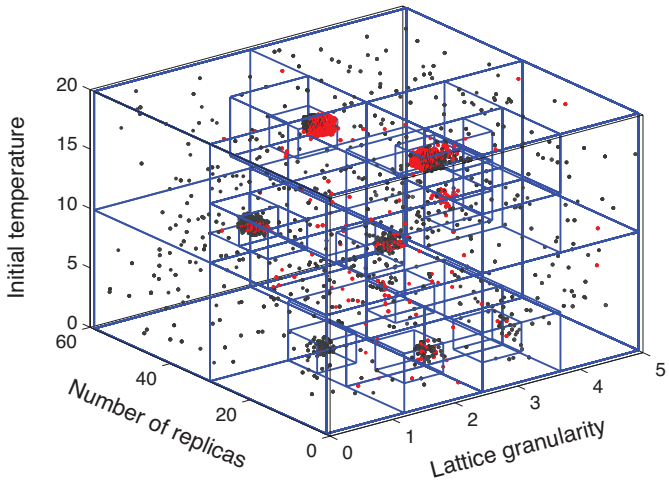
Random



Simulated Annealing



KOTree



KOTree highlights

- Space exploration
- Relevant metric prediction (expected job length)

1. Introduction
2. Motivation
   - Challenges of parametric scientific applications
   - Need for application-aware self-managed systems
3. Method
   - Towards a general application-aware self-managed VC system
   - Using KOtrees for parameter prediction and exploration
   - Integrated modular framework
4. Evaluation
   - Description
   - Results
5. **Discussion**
   - Related work
   - Conclusion

Related work

- Build and update a tree-like model at runtime, which is able to learn from observed data in a single pass, can be used to predict multiple application metrics and explore parameter spaces efficiently.
  - Stream mining algorithms [Guha et.al., Zhang et.al., Yang et.al., Ueno et.al., He et.al., Leng et.al., Raahemi et.al., Kawashima et.al., Qing et.al., Machot et.al., Domingos et.al.]
- Build a modular framework allowing integration of application-aware self-management in VC.
  - MindModeling@Home propose the Cell mechanism to explore parameter space [Moore Jr. et.al.]

Conclusion

We present an autonomic, modular framework for providing application-aware self-management for VC applications

KOTree is a fully automatic method that can be built and updated at runtime. At any point in time, we have an organized data structure that can predict multiple *metrics of interest* and explore the N-*dimensional* space of parameters effectively.

- This framework can effectively provide application-aware self-management in VC systems.
- The KOTree algorithm is able to predict expected length of new jobs accurately, resulting in an average of 85% increased throughput with respect to other algorithms.

Acknowledgements

Global Computing Lab - http://gcl.cis.udel.edu/

This work was supported by the **NSF IIS #0968350** entitled Collaborative Research: SoCS - ExSciTecH: An Interactive, Easy-to-Use Volunteer Computing System to Explore Science, Technology, and Health.

Questions

Contact

- Trilce Estrada, estrada@udel.edu
- Michela Taufer, taufer@udel.edu

Future work

- Adding a range expansion mechanism that allows just a rough estimate of the initial parameter space.
- Extending our application-aware self-management framework to other distributed systems.
- Extending KOTrees to perform multi-classification in the context of a general stream mining algorithm

Limitations

Parametric nature of KOTree

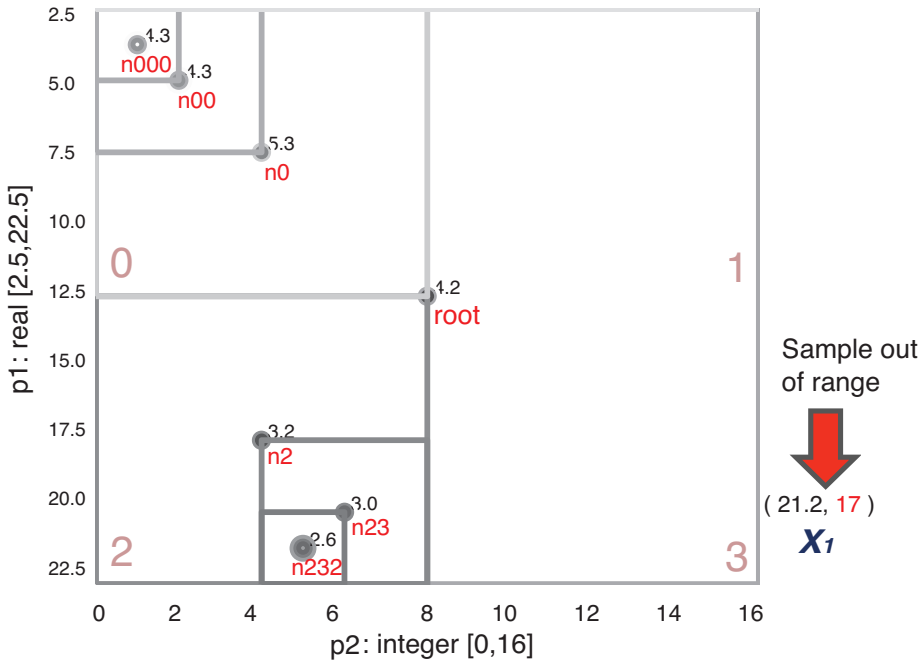
- Space requirements. For a KOTree with N dimensions and height ρ, the maximum number of nodes is:

$$total\_nodes = O(2^{N\rho}) \tag{1}$$

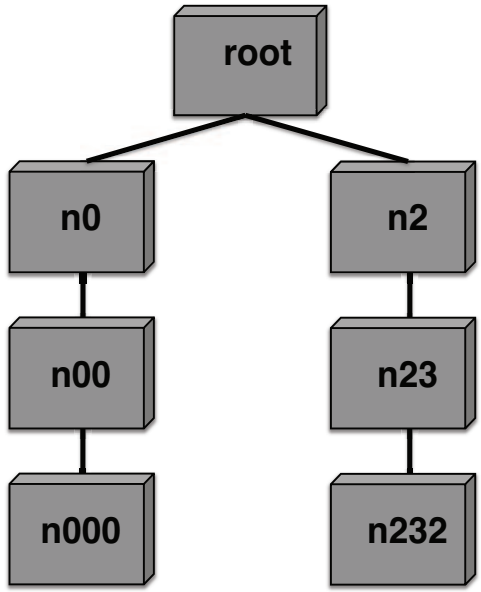
  - Few parameters $N \leq 10$
  - Height of the tree $\rho \leq 6$
- Parameter ranges
  - Runtime structural modification, allowing parameter space expansion

Range expansion

(a) 2-dimensional representation of KOTree

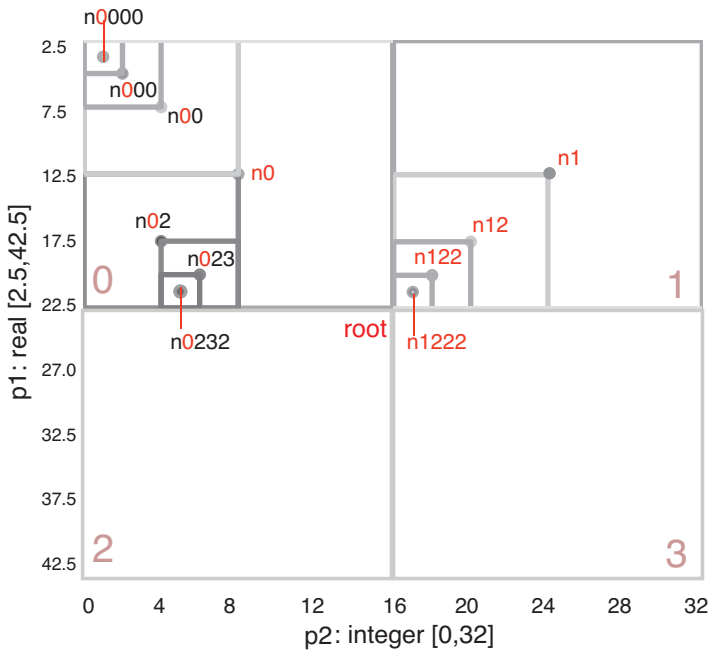


(b) KOtree structure

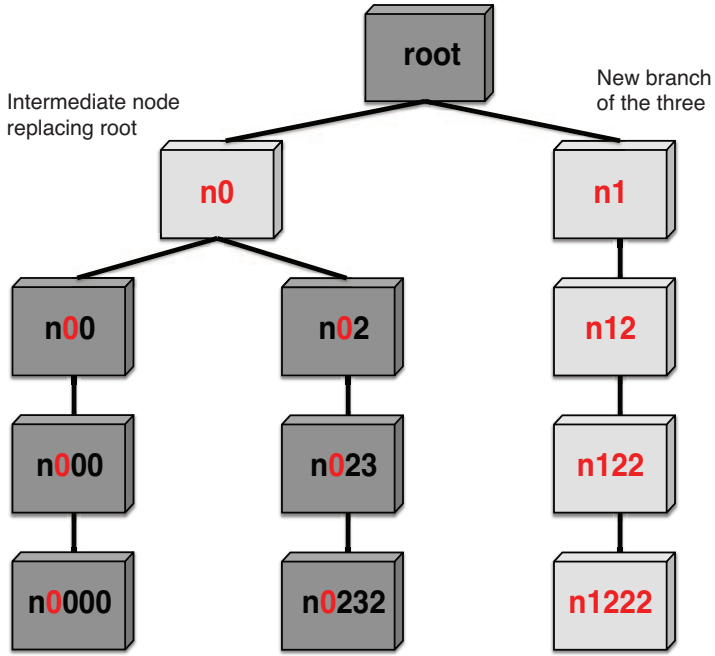


Range expansion

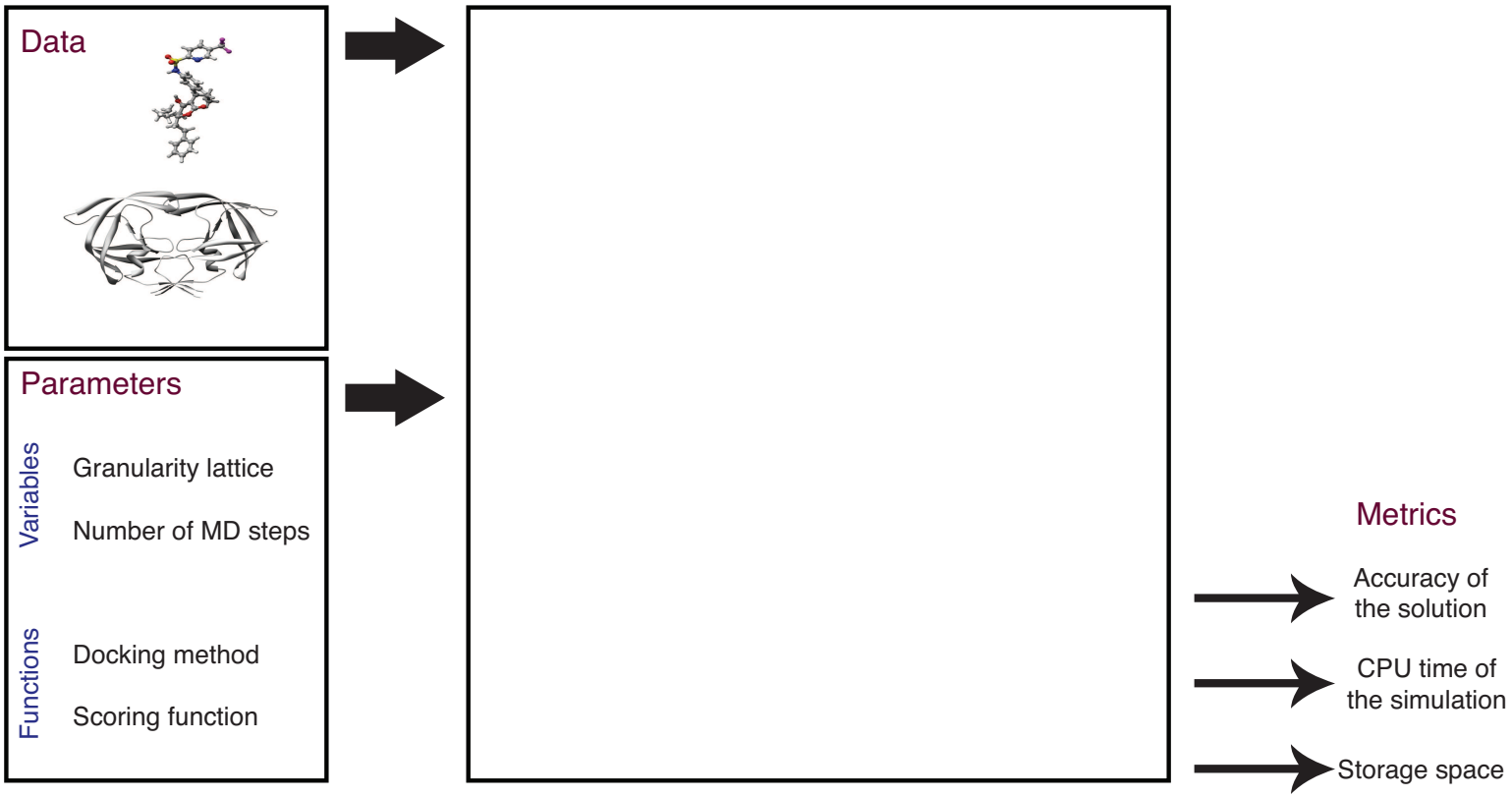
(a) Updated 2-dimensional representation of KOTree



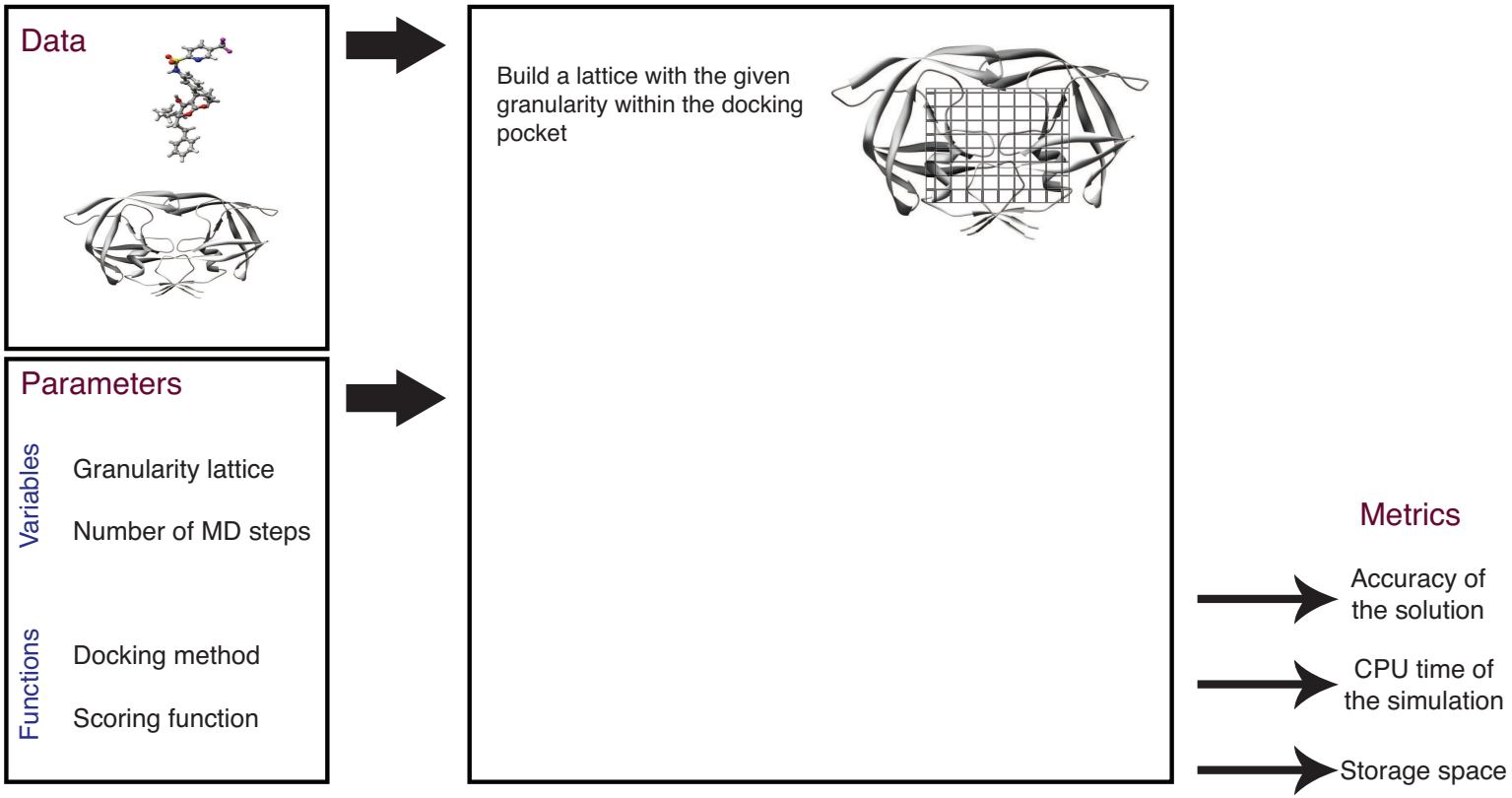
(b) Updated KOtree structure



Parametric application of protein-ligand docking



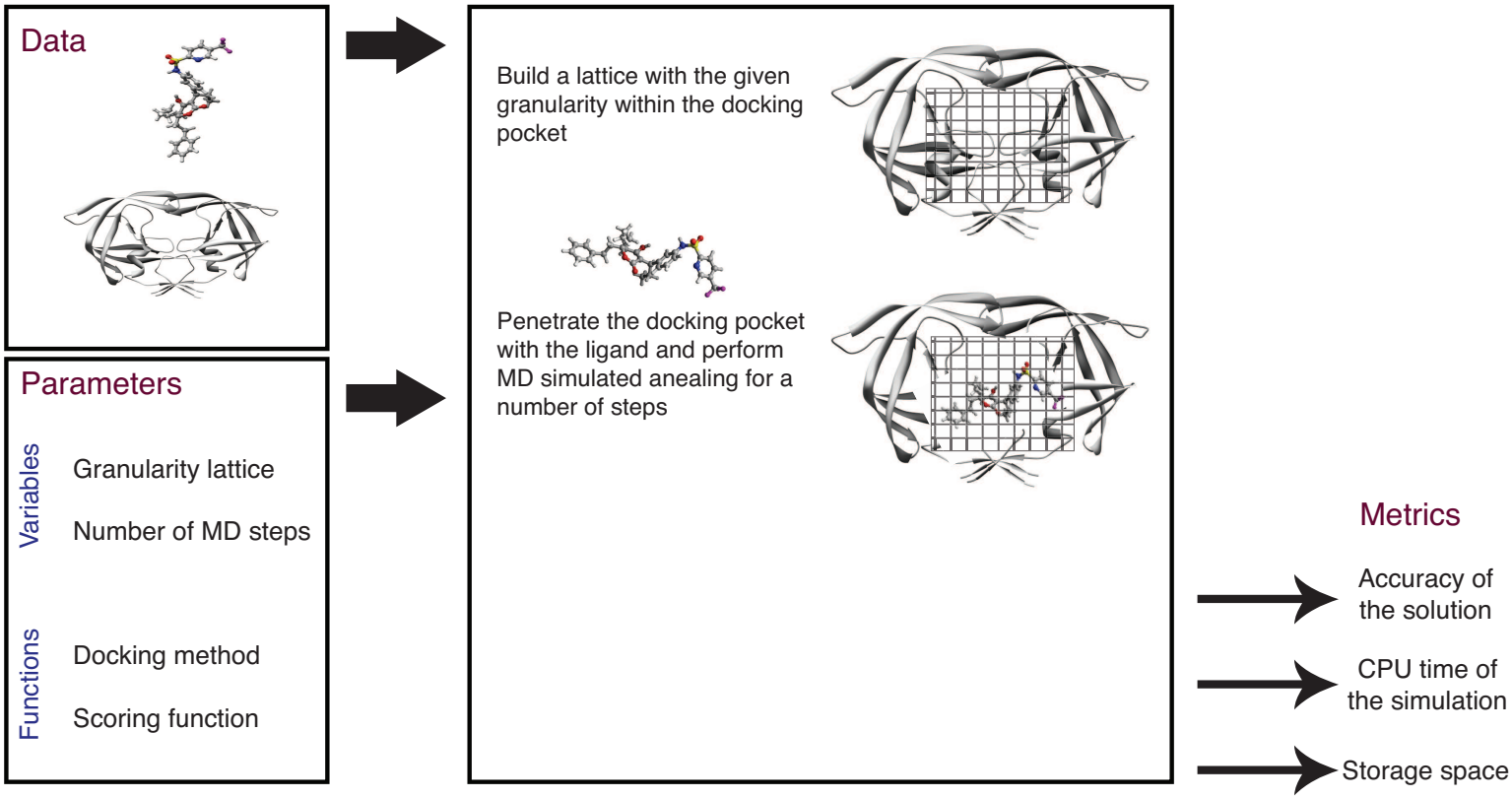
Parametric application of protein-ligand docking



Finer lattices:

- Increase storage space of the application
- Increase accuracy of the solution, but just up to a point

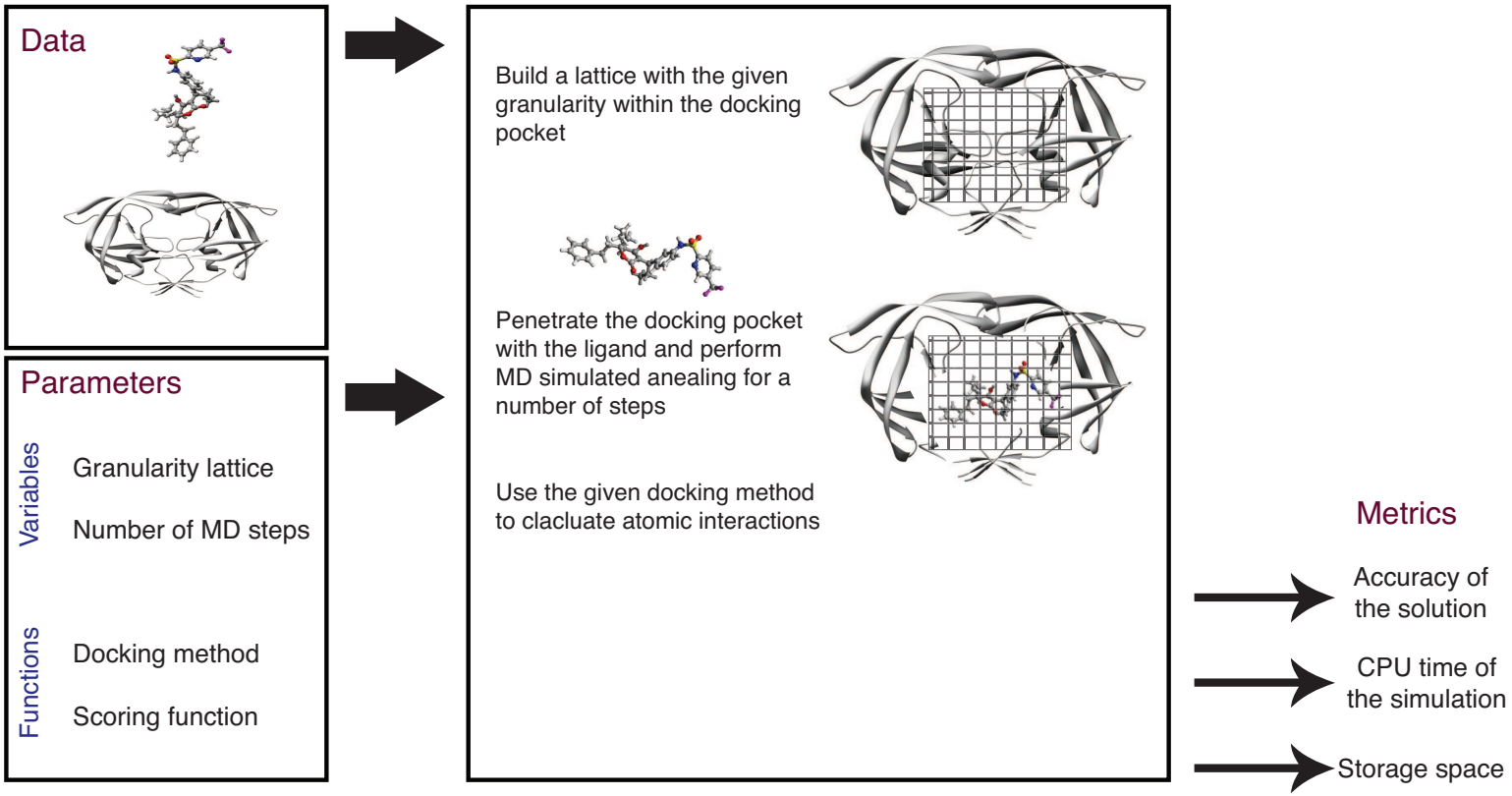
Parametric application of protein-ligand docking



Number of MD steps:

- Increase accuracy of the solution non-monotonically and just up to a point

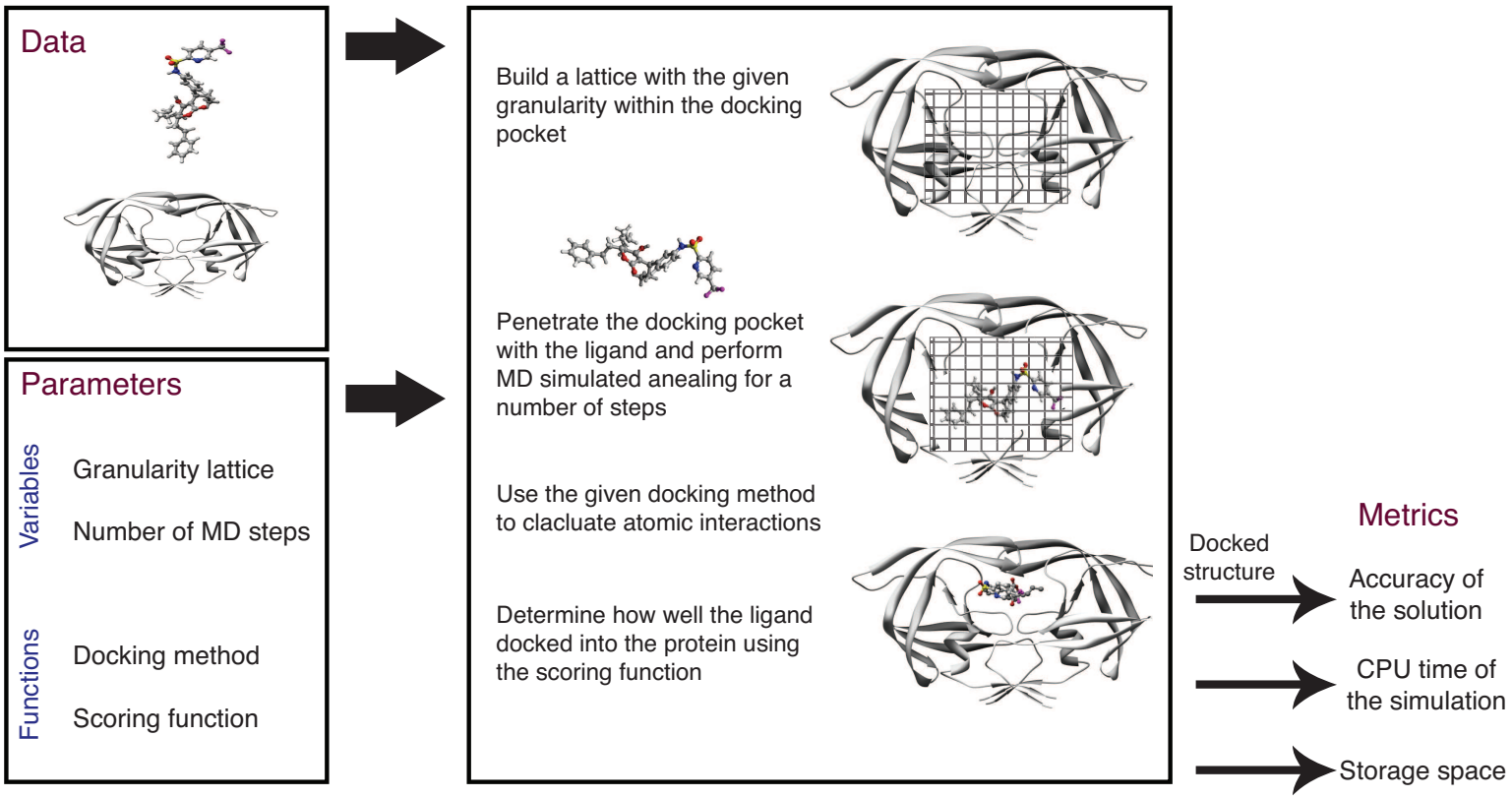
Parametric application of protein-ligand docking



Docking methods:

- Produce more or less accurate solutions
- Take different amounts of CPU time

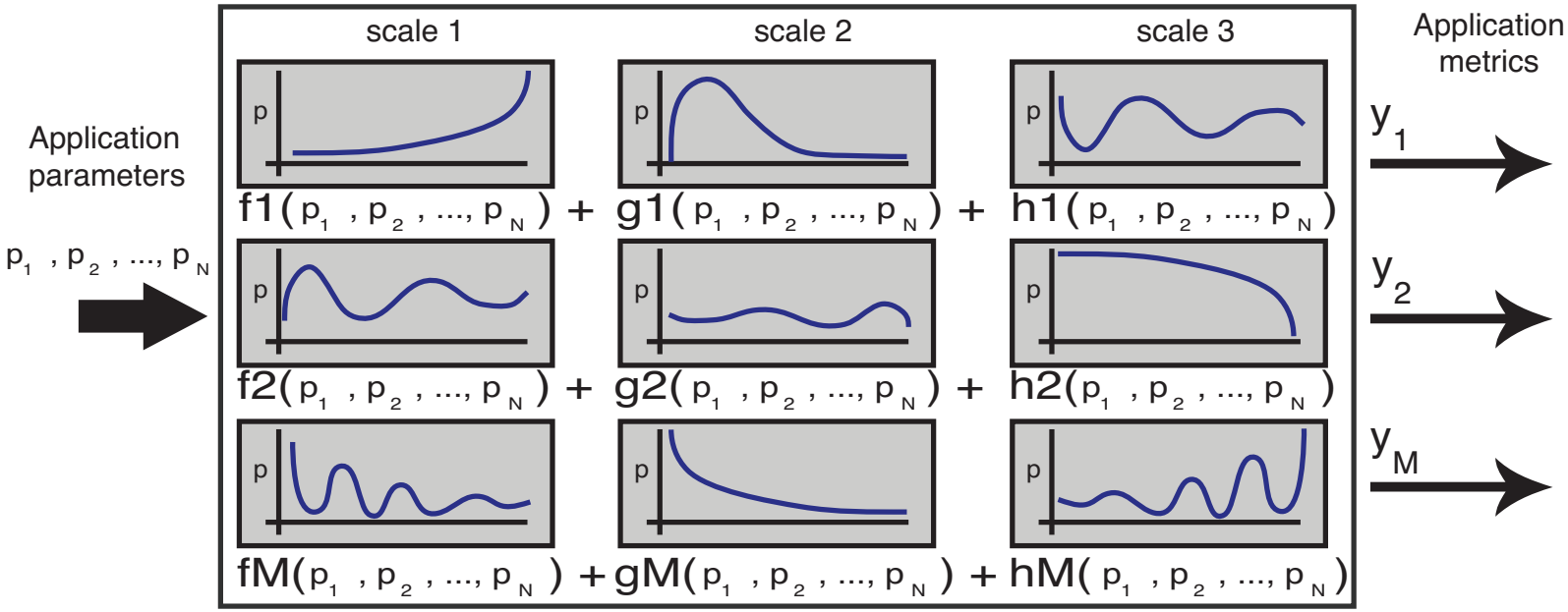
Parametric application of protein-ligand docking



Scoring functions:

- Have different sensitivity to rank correctly accurate solutions
- Take different amounts of CPU time per simulation

Simulating multiscale applications with EmBOINC



Given a specification provided by the user, our framework parses this specification and builds the functions into a Perl module that is used at runtime to provide information of each job to EmBOINC.

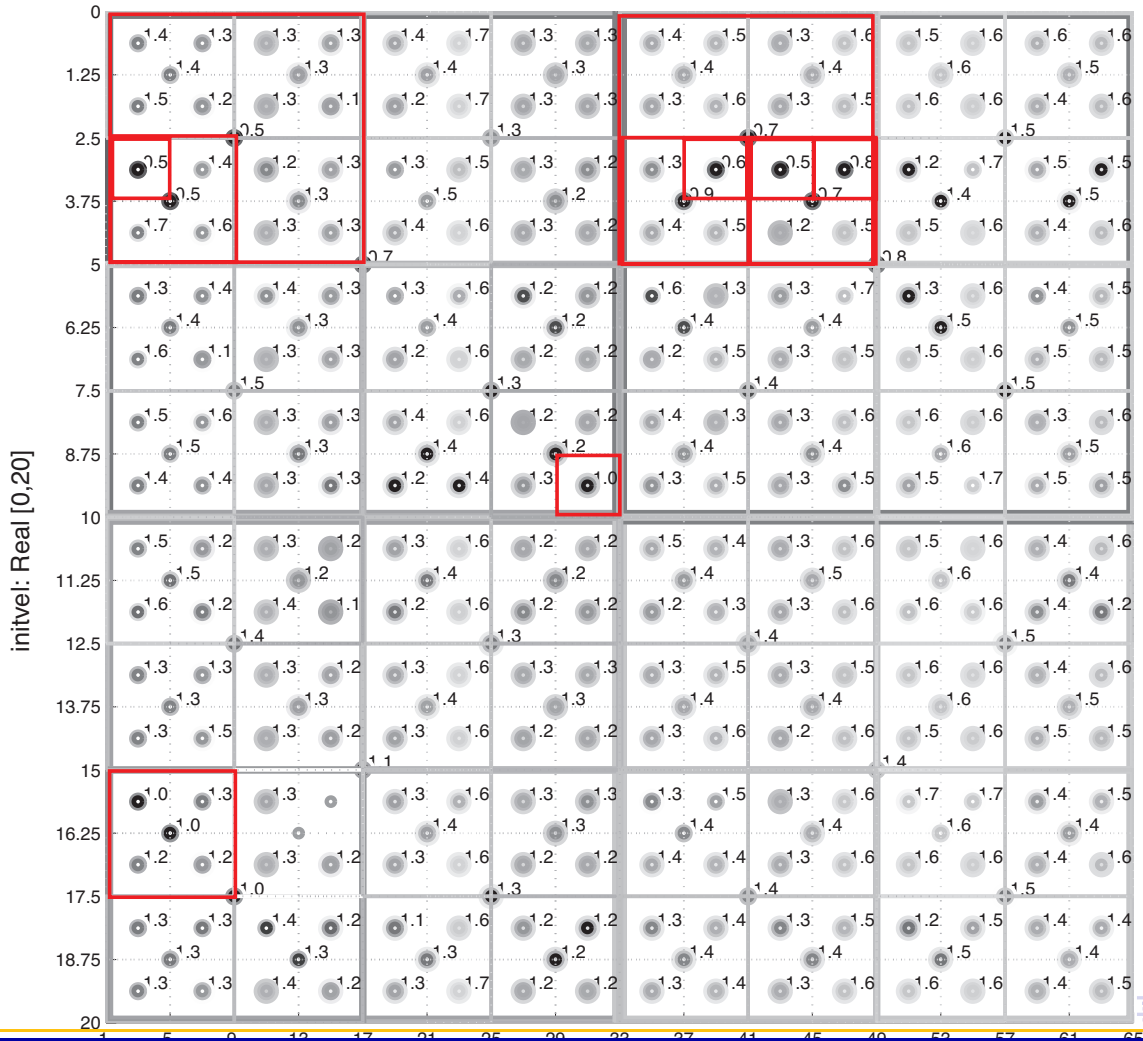
Simulating multiscale applications with EmBOINC

An example of an EmBOINC specification file looks like this:

```
@application replica_exchange
@metric time
@metric accuracy
@parameter number_replicas
   $time=2+(sin($number_replicas**3);
   $accuracy=0.6+sin($number_replicas*2);
@parameter initial_velocity
   $time=0.5+sin($initial_velocity);
   $accuracy=0.5+tan($initial_velocity)/100;
@parameter exchange_temperature
   $time=log($exchange_temperature+1)*2;
   $accuracy=(exp(-$exchange_temperature)*2;
```

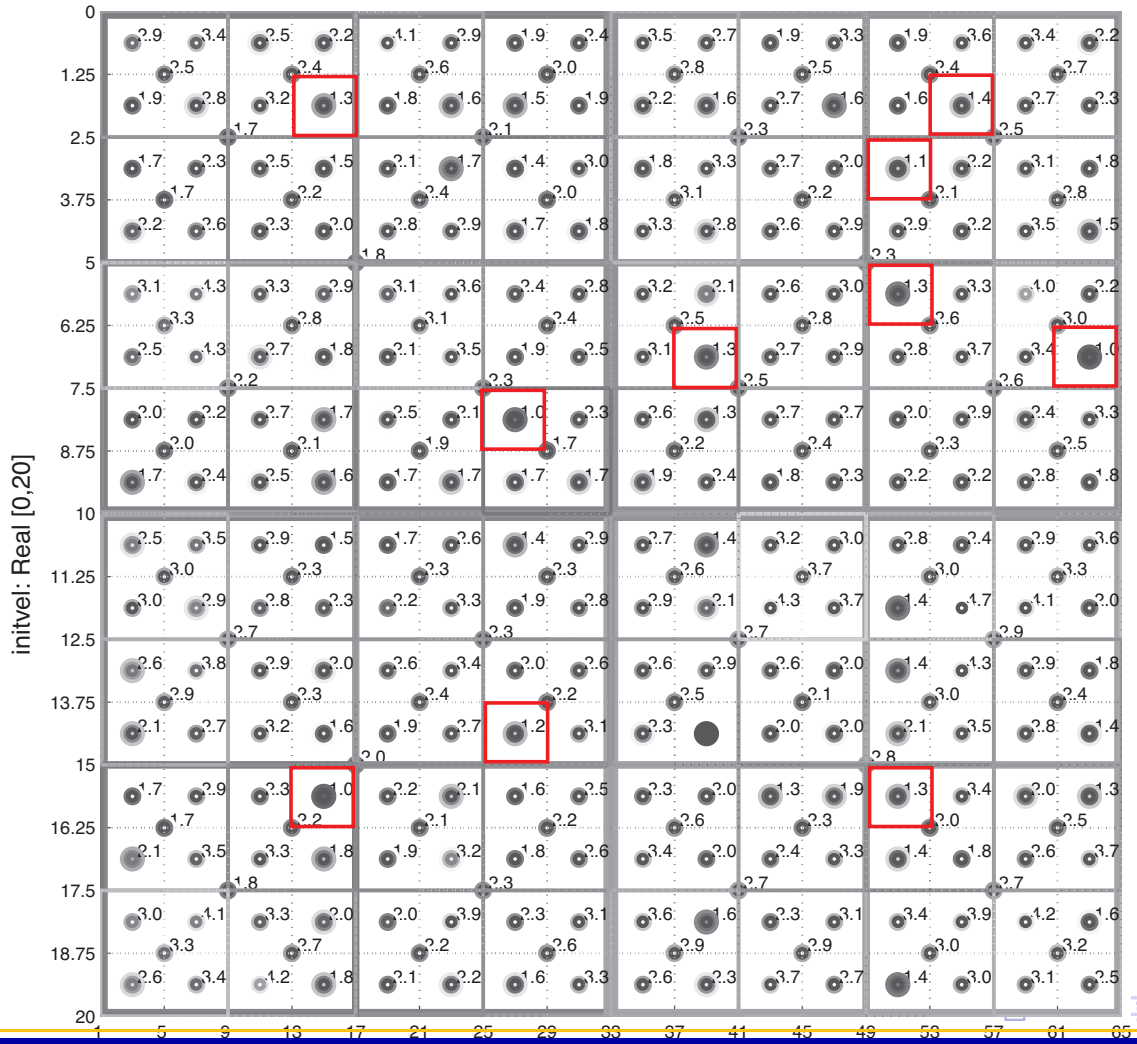
Parameter space exploration example 1

Goal: minimum expected error



Parameter space exploration example 2

Goal: minimum expected time



Modular framework

Job generation module

- Provides a specification for parameter generation to the learning engine.

A Replica Exchange (RE) simulation can be expressed as follows:

```
@application replica_exchange
@parameter num_replicas integer [512 1024]
@parameter init_temp integer [1000 10000]
@metric specific_heat real
@metric total_time integer
@metric expected_flops
@goal var(specific_heat)*exp(total_time)
@predict exp(expected_flops)
```

Modular framework

Job generation module

- Communicates parameters in a format that is understandable by the application.
- Provides the application with specifications of the workload such as number of replicas to be executed, and quorum.

Communication with the application is done through XML files

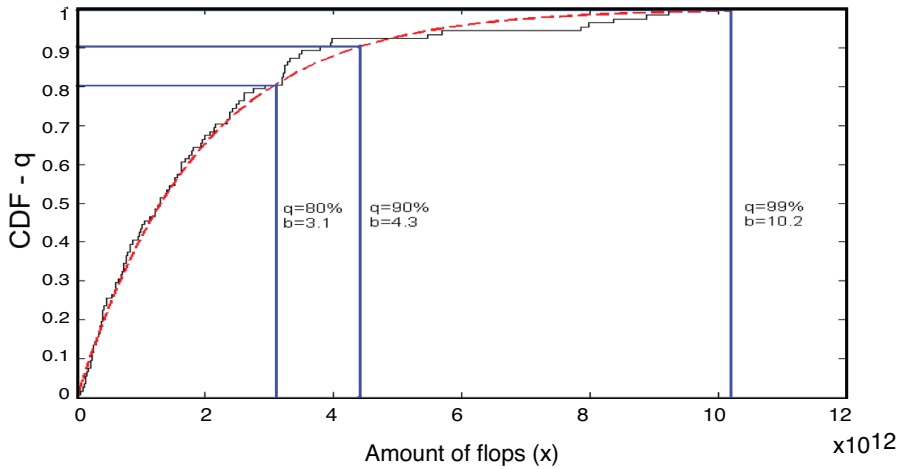
```
<params> 64, 3000, 5000 </params>
<expected_flops> 2155683199 </expected_flops>
<quorum> 3 </quorum> ...
```

Modular framework

System assessment module

Determines the expected CPU time (CPU_t) that the resources can successfully process based on:

- the 85*th* quantile of distributed jobs (in flops)
- the number of unsatisfied requests times the average assigned workload per request
- the 85*th* quantile of distributed jobs whose execution latency has exceeded a time-out bound



Modular framework

System assessment module

This module receives three files from the distributed system:

- **Log 1** : Time of request, amount of flops requested, amount of flops assigned
- **Log 3** : Job id, flops, CPU time, distributed time, collected time
- **Log 3** : Timed-out job id, estimated flops, distribution time, time-out bound

Modular framework

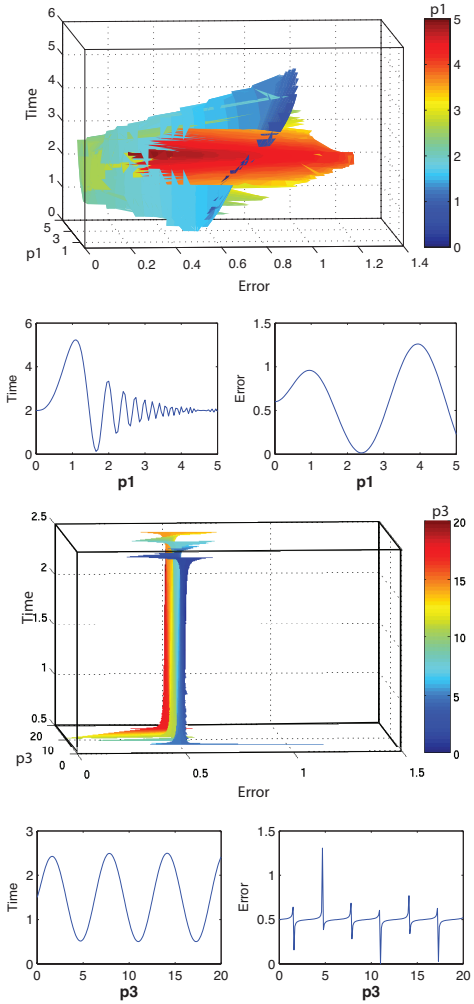
Result evaluation module

Extracts and formats metrics from collected results, then communicates the output to the learning engine.
Following with our RE example, an output file looks like this:

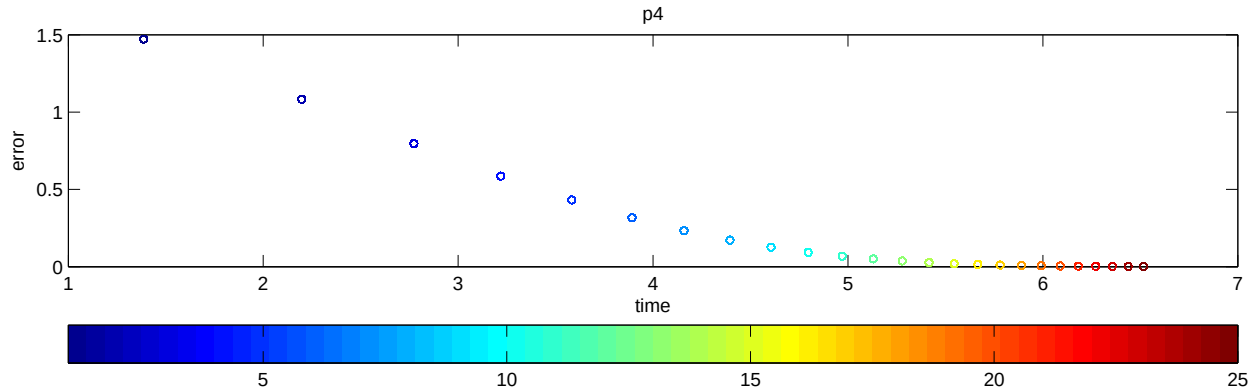
```
<out params="64, 3000"> 3456.78, 986, 24563</out>
```

Evaluation

14 scenarios

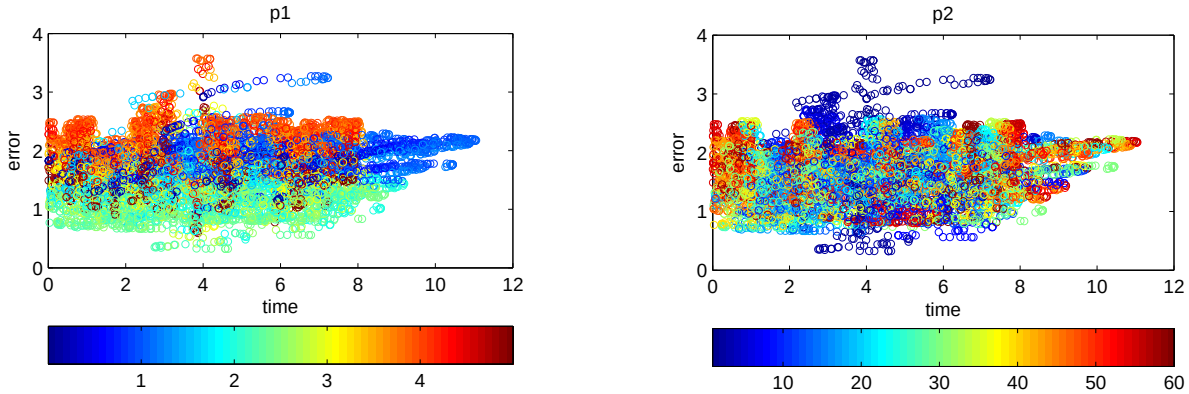


SAN is better than KOtree when



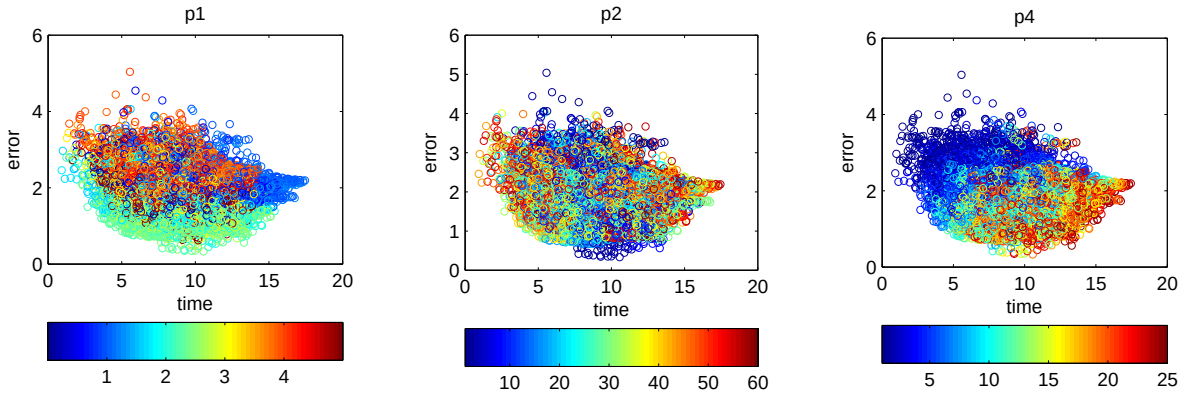
The application has a single parameter with a small domain, and the application has a well defined global minimum, such as in p4

SAN is better than KOtree when



There are only two parameters, one of them dominates the metric of interest and has a quasi-random behavior, such as in p1p2

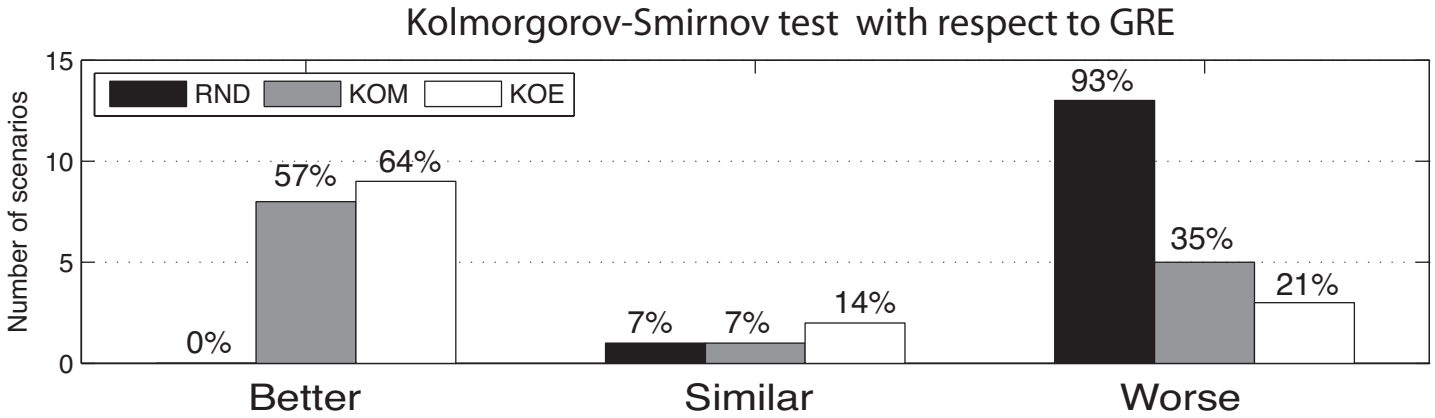
SAN is better than KOtree when



Similar parameter values do not cluster naturally, and appear to be scatter all over the landscape , such as in p1p2p4

Comparison of scenarios

Case study 1 - minimizing time

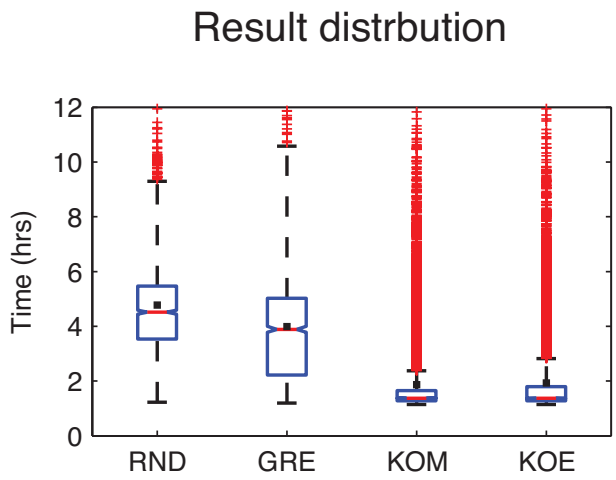


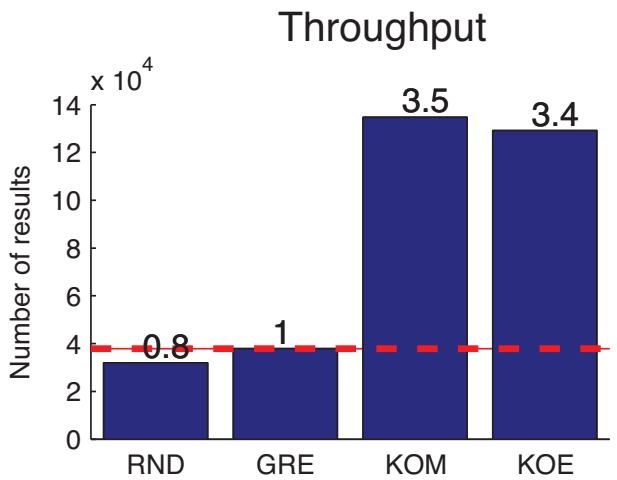
- *KOM* is better than *SAN* in 57% of the cases and increases throughput in average 75%.
- *KOE* is better than *SAN* in 64% of the cases and increases throughput in average 132%.

Comparison of scenarios

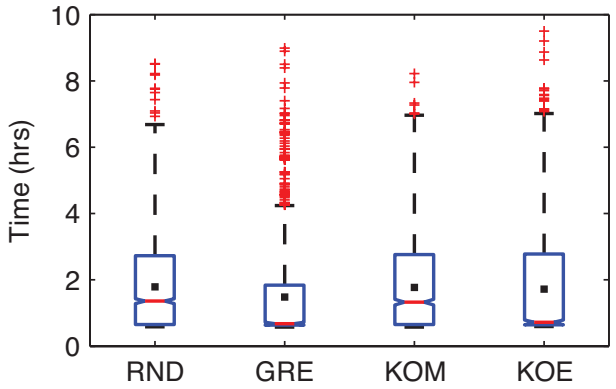
Case study 1 - minimizing time

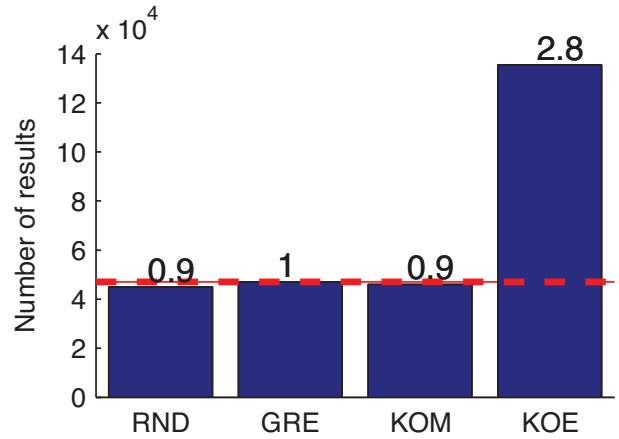
(a) p3p4





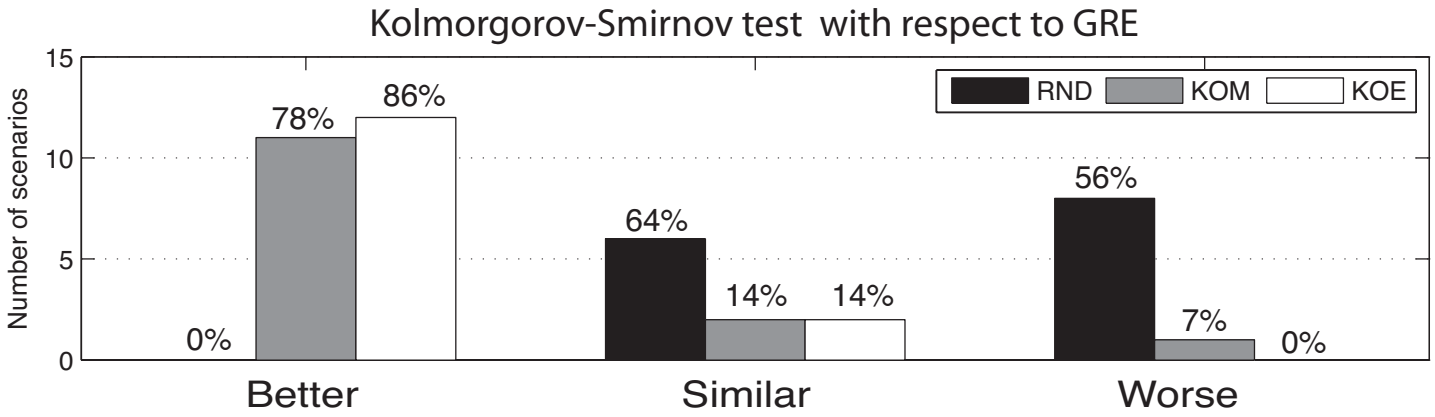
(c) p2





Comparison of scenarios

Case study 2 - maximizing accuracy

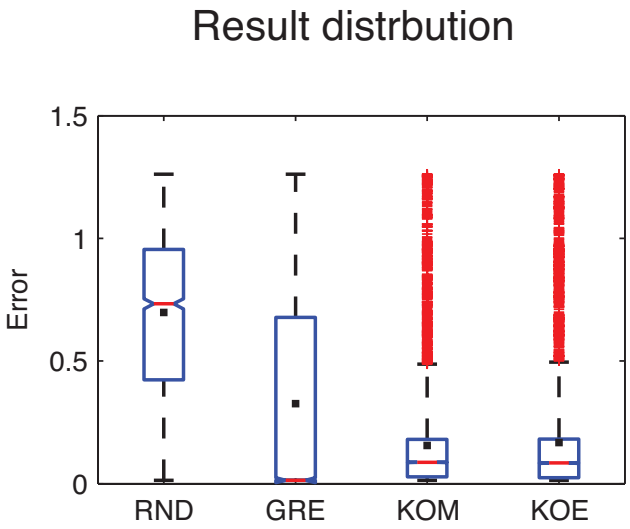


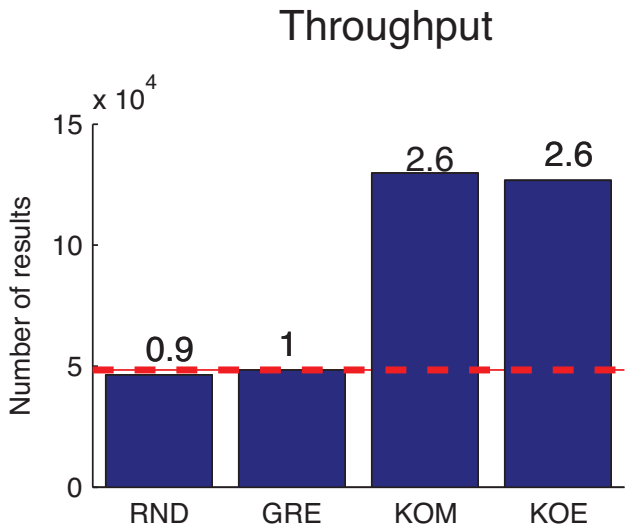
- *KOM* is better than *SAN* in 78% of the cases and increases throughput in average 73%.
- *KOE* is better than *SAN* in 86% of the cases and increases throughput in average 61%.

Comparison of scenarios

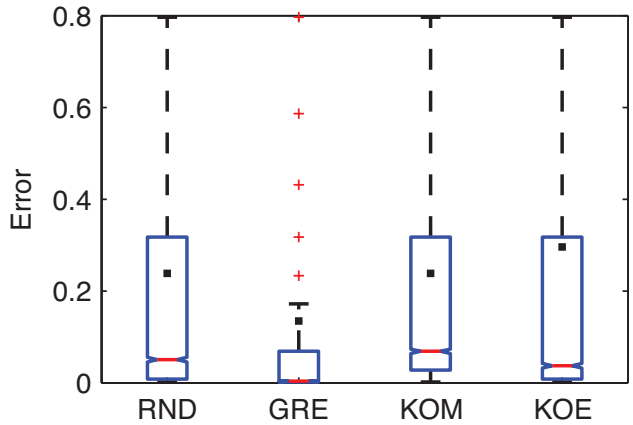
Case study 2 - maximizing accuracy

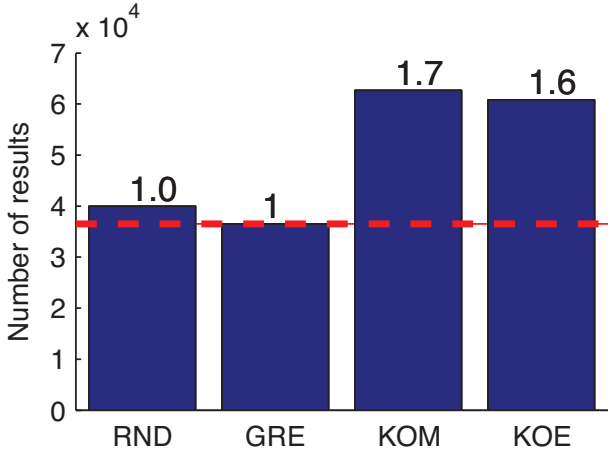
(a) p1





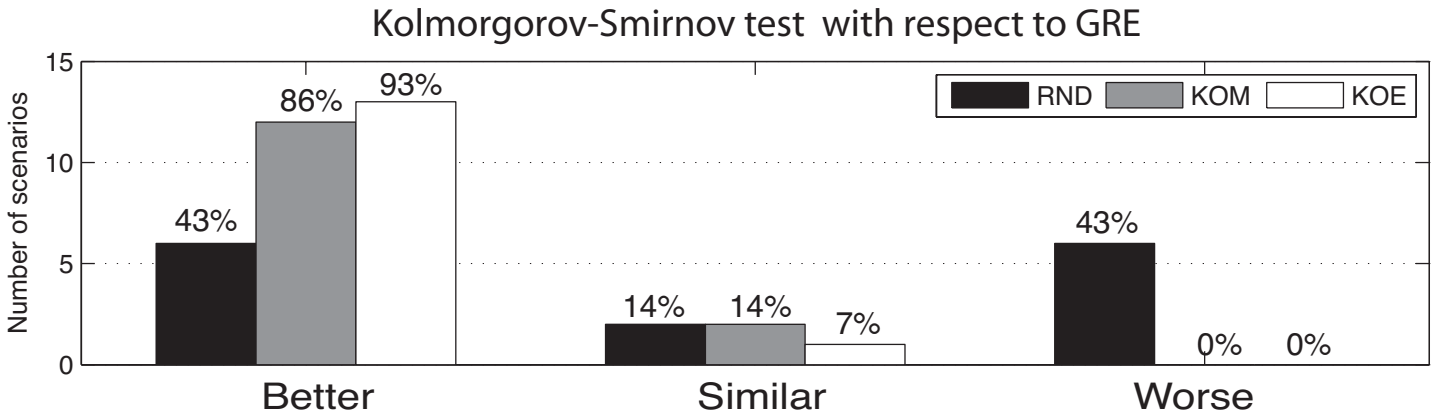
(b) p4





Comparison of scenarios

Case study 3 - balancing time and accuracy

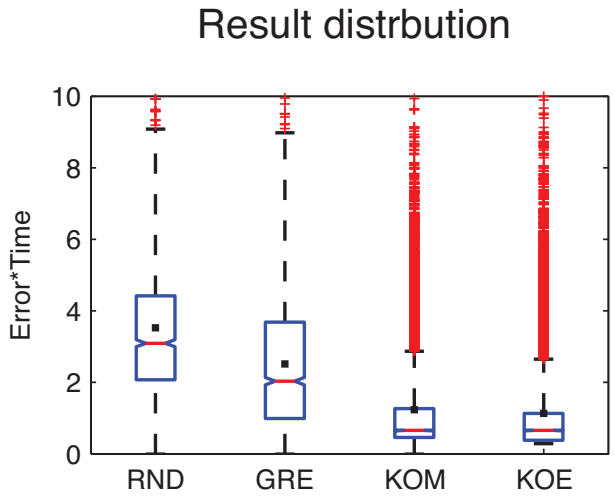


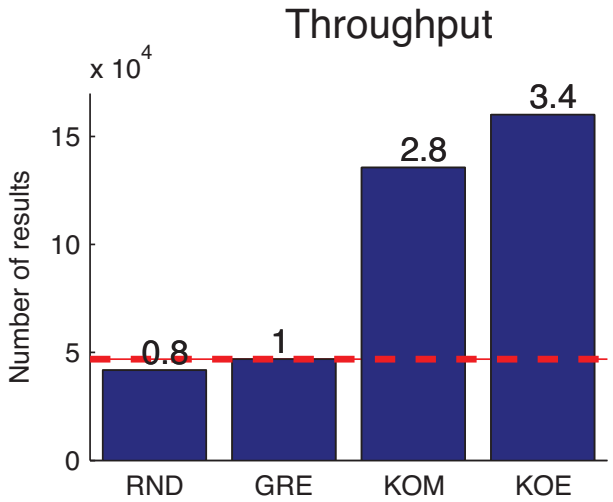
- *KOM* is better than *SAN* in 86% of the cases and increases throughput in average 85%.
- *KOE* is better than *SAN* in 93% of the cases and increases throughput in average 107%.

Comparison of scenarios

Case study 3 - balancing time and accuracy

(a) p1p3





(b) p1p2