ACIC: Automatic Cloud I/O Configurator for HPC Applications

Mingliang Liu[†,‡] liuml07@mails.thu.edu.cn
Ye Jin[§] yjin6@ncsu.edu
Jidong Zhai[†] zhaijidong@tsinghua.edu.cn
Yan Zhai[*,¶] yanzhai@cs.wisc.edu
Qianqian Shi[†] shiqq11@mails.thu.edu.cn
Xiaosong Ma[§,♮] ma@ncsu.edu
Wenguang Chen[†,‡] cwg@tsinghua.edu.cn

[†] Department of Computer Science and Technology, Tsinghua University
Tsinghua National Laboratory for Information Science and Technology
[‡] Research Institute of Tsinghua University in Shenzhen
[§] Department of Computer Science, North Carolina State University
[¶] Department of Computer Sciences, University of Wisconsin-Madison
[♮] Computer Science and Mathematics Division, Oak Ridge National Laboratory

ABSTRACT

The cloud has become a promising alternative to traditional HPC centers or in-house clusters. This new environment highlights the I/O bottleneck problem, typically with top-of-the-line compute instances but sub-par communication and I/O facilities. It has been observed that changing cloud I/O system configurations leads to significant variation in the performance and cost efficiency of I/O intensive HPC applications. However, storage system configuration is tedious and error-prone to do manually, even for experts.

This paper proposes ACIC, which takes a given application running on a given cloud platform, and automatically searches for optimized I/O system configurations. ACIC utilizes machine learning models to perform black-box performance/cost predictions. To tackle the high-dimensional parameter exploration space unique to cloud platforms, we enable affordable, reusable, and incremental training guided by Plackett and Burman Matrices. Results with four representative applications indicate that ACIC consistently identifies near-optimal configurations among a large group of candidate settings.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage*; D.4.2 [**Operating Systems**]: Performance—*Modeling and prediction, Measurements*

General Terms

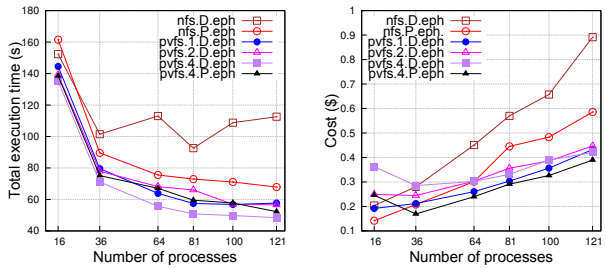
Performance, Measurement, Design, Management

Keywords

Storage, Modeling, Performance, Cloud Computing

1. INTRODUCTION

More and more HPC users today are beginning to explore running their applications in the cloud [20, 3, 12, 55]. Emerging cloud resources targeting HPC usage, such as the Amazon CCIs [3], have largely improved the outlook for HPC in the cloud. Clouds offer many advantages over traditional HPC platforms: elastic resource allocation, elimination of queue waiting, no up-front hardware investment or hosting/maintenance/upgrades, and convenient pay-as-you-go pricing models. By closing on the performance gap between cloud instances vs. in-house clusters [55], public clouds have become a cost-effective choice to many scientific application users and developers.



(a) Execution time (b) Total cost

Figure 1: The execution time and monetary cost of BTIO under selected I/O system configurations, in terms of file system type, number of I/O servers, and placement strategy

Unfortunately, cloud platforms amplify the growing performance gap between the I/O subsystem and other system components long existing in conventional HPC environ-

*Yan took part in this work at Tsinghua University.

SC13 November 17-21, 2013, Denver, CO, USA

http://dx.doi.org/10.1145/2503210.2503216.

ments [24]. Leading cloud platforms such as Amazon interconnect the compute instances with commodity networks instead of dedicated high-speed interconnection, such as InfiniBand. Also, multi-tenant cloud resources deliver inferior and sometimes highly variable performance [5].

On the flip side, clouds empower users with full, a-la-carte configuration of the I/O subsystem, which is impossible on traditional HPC clusters. For example, users can choose important I/O parameters such as the file system type, the number of I/O servers, the type and number of I/O devices to use, etc. Previous study revealed that the in-cloud performance of representative HPC applications is highly sensitive to such I/O system configurations [32]. Figure 1 demonstrates this impact on both performance and monetary cost of running the NPB BTIO application (more information in Section 5), shown to vary dramatically with different I/O system configurations. It also shows that even for a single HPC application, its performance/cost behavior across different I/O configurations varies with different problem/job sizes, and no single configuration excels in all cases. The cloud enables users to setup optimized I/O configurations for *individual* application upon its execution, instead of forcing all applications to use a pre-configured solution.

However, taking advantage of this uniquely available configurability and deriving optimized per-application I/O configuration are very challenging and potentially very expensive. Several factors, including the lack of one-size-fits-all parameter choices, the complexity from both the system and the application side, and the obscureness of I/O system hardware/software details due to virtualization, make white-box modeling and analysis unrealistic. Meanwhile, the high-dimensional cloud I/O configuration parameter space makes learning-based, black-box approaches quite costly, in terms of both time and monetary overhead. Furthermore, as I/O configuration has been shown to be application- and even scale-dependent, knowledge and training data obtained from one application may not apply to another.

There are many tools that evaluate and configure storage systems for traditional clusters [2, 4, 16, 30] (more discussions in Section 7). However, some of them [16, 30] focus on the storage devices only and hence are not able to address the complex, high-dimensional cloud I/O configuration problem. Some others (such as Minerva [2]) are extremely complicated for non-expert users, requiring expertise with advanced tools and a large number of experiments. Moreover, none of them covers the complicated cost-performance tradeoff unique to the cloud.

To address this problem, we propose ACIC (**A**utomatic **C**loud **I**/O **C**onfigurator), the first tool to optimize the I/O system for HPC applications in the cloud. Given an application to run on a given cloud platform, ACIC automatically searches for optimized I/O system configurations from many candidate settings. Our approach takes advantage of a black-box model to learn the relationship between influential I/O system configurations and the optimization objective (cost or performance). After training the model on the target cloud platform, ACIC automatically extracts the given application's I/O characteristics, evaluates candidate I/O configurations, and recommends an optimized configuration according to user's selected objective.

Though learning-based performance modeling/prediction has long been explored, including for parallel applications [44, 54], ACIC's originality lies in the cost-saving mechanisms that make such approaches affordable on clouds:

1. We explore a crowdsourcing service model for automatic, per-application cloud system configuration, where community members build and share a public performance/cost database. The service may not rely on, but can benefit from continuous training data contributions, which improve its configuration accuracy, as well as its adaptivity to system upgrades. We describe our proof-of-concept ACIC tool using parallel I/O as a case study, yet the service model applies to other configurable systems.
2. Rather than case-by-case learning/prediction, we enable *reusable training* by adopting a generic synthetic I/O benchmark and systematically sampling the parameter space.
3. To tackle the large training space that renders the model training prohibitively expensive, we perform dimension reduction by evaluating parameters' impact on performance using PB matrices [38].

We implemented ACIC, trained it with the synthetic yet expressive parallel I/O benchmark IOR [42, 49] on Amazon EC2, and evaluated it with four real-world data-intensive parallel applications. Our results indicate that ACIC consistently provides optimized configurations that improve performance (total execution time) by a factor of 3.0 on average and the cost saving of 53% on average under the baseline configuration (see Section 5).

We have recently released the ACIC tool, plus all our training data collected from EC2 [26]. Currently, users can download the shared training data, build the prediction model, use our provided tool to obtain I/O characteristics from their applications, run the prediction, and configure EC2 to deploy the recommended I/O configuration with our provided scripts. In the future, we plan to also provide full web-based services to enable online configuration queries.

2. APPROACH OVERVIEW

Figure 2 illustrates the ACIC architecture. Its central component is a black-box prediction model, which can be bootstrapped with a certain amount of initial training data. ACIC takes both the cloud system I/O configuration parameters (such as file system type, storage device type, number of I/O servers, etc., to be described in Section 3.1) and application I/O characteristics (such as major operation type, read/write block size, read/write count, etc., to be described in Section 3.2). Concatenated together, these parameters constitute a 15-D exploration space for ACIC's training and prediction. To reduce the time overhead and monetary cost associated with training, ACIC employs a dimension reducer using Plackett-Burman (PB) matrices [38], with more details discussed in Section 4.1.

Generally, there are several ways to collect the training data, such as application case studies, benchmarks, and trace replays. ACIC chooses the IOR [42] synthetic benchmark as it is generic, highly configurable, and open-source. It carries out the initial training by running the synthetic IOR benchmarks on the target cloud system, systematically sampling the concatenated parameter space across the dimensions selected through PB matrices. For each training run, ACIC collects the performance (cost) metric with the candidate cloud I/O configurations. With the sampled data

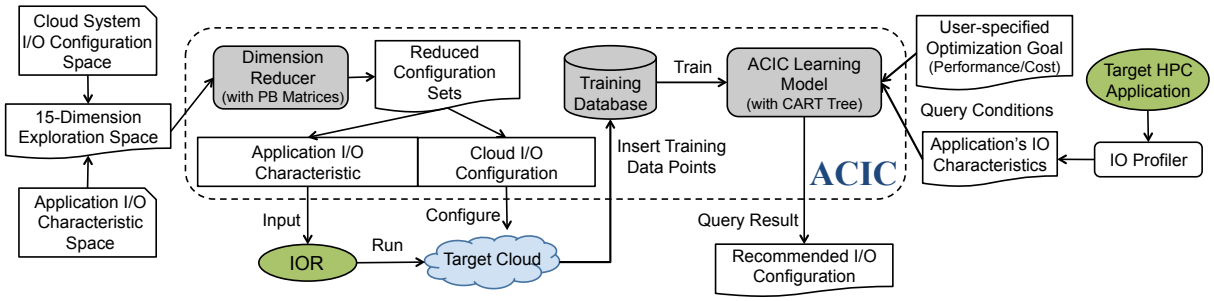


Figure 2: ACIC architecture

points fed into a training database, ACIC can use different machine learning algorithms to train its black-box prediction model. In our implementation, we use the popular classification and regression trees (CART) [35].

Given a target HPC application, users can either directly provide values of relevant I/O characteristics, or use a simple profiling tool (included as part of ACIC) to extract such application-specific parameters. Both approaches are feasible, as HPC applications, especially parallel simulations, are known to have periodic, relatively well-defined I/O behavior. Based on the user-specified optimization goal, currently either the performance (application execution time) or the monetary cost of execution, ACIC outputs the predicted optimal I/O configuration. Note that the monetary cost of a certain application execution is not proportional to the execution time here, as I/O servers can be placed at didicated instances or part-time ones.

One major advantage of ACIC is its reusability. It is worth pointing out that even with its dimension reducer, the initial training of ACIC may cost dozens to hundreds of hours (and dollars). However, we argue that such expense is reasonable considering that the application-independent IOR training results can be reused. Therefore, the training cost is to be amortized over many different applications and different executions of the same applications.

Another chief advantage of ACIC is its expandability. First, it benefits from continuous, incremental training. With more user-contributed IOR training data points, ACIC achieves higher prediction accuracy. This allows it to bootstrap with sparse sampling in its initial training. The additional training may even come at no extra monetary cost, as public clouds like Amazon EC2 typically charge users at a hourly billing granularity. Users can fit one or more short IOR training runs into the "residual" time allocation, after completing their application runs. Second, with continuous, incremental training, the ACIC training database can effortlessly deal with cloud hardware/software upgrades with common data aging methods. Third, ACIC can easily handle new I/O configurations or characteristic parameters by adding more dimensions into its prediction model, though the open-source IOR benchmark may need to be expanded if an application has I/O features that it does not test.

Finally, although the training and prediction are cloud-dependent, ACIC makes no assumptions on the cloud I/O configurations or application I/O characteristics and can be applied to any platform-application combinations.

3. EXPLORATION SPACE

3.1 I/O Configuration Options

Figure 3 depicts the configurable I/O system stack in the cloud, using Amazon EC2 terms. At the lowest level is the storage hardware, where users can choose between two forms of virtual disk devices: Elastic Block Store (EBS) and local ephemeral disks (standard or SSD). Multiple device instances can further be aggregated with configurable software RAID. Above the storage layer are the shared file systems, such as PVFS2 [9] and widely used NFS [8]. For each selected file system, there are also configurable parameters such as the number and placement of I/O servers, plus internal settings like stripe size and buffer sizes. Finally, between the file system and the applications, scientific codes often perform I/O through parallel I/O or middleware such as the MPI-IO and HDF5 [14] library, though some codes may directly utilize the universal POSIX interfaces. In this paper, we choose to leave the EBS QoS level and file system internal buffer sizes with default configurations, as the IOPS (Input/Output Operations Per Second) metric used by the former is not that relevant to HPC applications, and our empirical study did not find the latter with significant impact on performance or cost.

Application	**Application file interface** (MPI-IO vs. POSIX)	
File System	**File system internal parameters** (Stripe Size: 64KB/4MB)	
	I/O server number (1/2/4)	**I/O server placement** (Dedicated vs. Parttime)
	File system (NFS vs. PVFS2)	
Storage Device	**Software RAID** (RAID 0 vs. No RAID)	**Device number** (1/2)
	Cloud storage device type (EBS vs. Ephemeral)	

Figure 3: Configurable I/O system stack in the cloud

Figure 3 shows all configurable layers in the cloud I/O stack, from I/O library all the way to storage device hardware. In contrast, on traditional shared parallel platforms users typically can only configure the top layer. Therefore, in this paper we focus on the layers opened up by cloud platforms. Below we briefly describe the I/O configurations found relevant to parallel applications' performance/cost in the cloud [32].

Storage device and organization Cloud platforms typically provide multiple storage choices, with different levels of abstraction and access interfaces. E.g., with EC2 CCIs, applications have access to: *1*) the local block storage ("ephemeral") with 4×840GB capacity, where user data does

not persist across instance reservations, *2*) off-instance, persistent Elastic Block Store (EBS), and *3*) SSD disks. Apart from data persistence, the ephemeral and EBS devices possess different performance characteristics, usage constraints, and pricing policies. Finally, a cloud HPC user can easily scale up the aggregate I/O capacity and bandwidth, e.g., by aggregating multiple disks into a software RAID 0 partition.

File system selection and configuration Typically, supercomputers or large clusters have parallel file systems such as Lustre [41], GPFS [1], and PVFS [9], while smaller clusters tend to choose shared file systems such as NFS [8]. Cloud users can choose between the two categories based on individual applications' demands, and switch between selections quite easily and quickly, unlike with traditional HPC resources. Once selected, a parallel/shared file system itself has many internal knobs and is non-trivial to configure. Most conventional parallel platforms adopt fixed default settings as it is impossible to cater to individual applications.

In this proof-of-concept work, we focus on two important and highly application-dependent parameters, which configure the file system servers. Parallel file systems can use different numbers of I/O servers. In addition, one can choose to have *dedicated* vs. *part-time* I/O servers. With the former, I/O servers run on separate cloud instances, while with the latter, they share physical instances with a subset of the compute nodes. Due to the obvious impact of I/O server provisioning in both performance and cost, it is important to optimize such server placement for better resource utilization and cost-effectiveness.

3.2 Application I/O Characteristics

I/O workload characterization has remained an active problem [13, 39, 24]. Meanwhile, though applications have varying concrete I/O patterns, they also share high-level I/O behaviors common to most HPC scientific codes, especially the periodic checkpoint/restart output activities.

To enable reusable training, ACIC chooses to measure cloud I/O performance with sampled system configurations using synthetic benchmarks created via IOR [42]. IOR is a flexible and expressive parallel I/O benchmark that can be configured to mimic different applications' I/O behavior. Also, its open-source nature allows easy extension to test additional I/O features when the need arises.

Currently ACIC considers the following I/O characteristics parameters in creating IOR test cases. Note that these do not include access spatiality (random vs. sequential), as most modern HPC applications perform sequential I/O, dominated by append-only writes [42]. The range of parameters is selected based on the real-world applications used in our evaluation, and can be expanded with additional training, without invalidating the collected data.

- **Number of processes:** total number of processes running the application in parallel
- **Number of I/O processes:** number of processes performing the I/O operations simultaneously
- **I/O interface:** POSIX, MPI-IO [31], or high-level libraries such as HDF5 [14] and netCDF [40]
- **I/O iteration count:** number of I/O iterations within the application execution
- **Data size:** amount of data each I/O process reads and writes within each I/O iteration (e.g., the size of the 3-D array partition assigned to each process)
- **Request size:** amount of data transferred in each I/O function call (I/O request size)
- **Read and/or write:** I/O operation type
- **Collective on:** whether I/O processes adopt collective I/O [47] to cooperatively read/write shared files
- **File sharing on:** whether the I/O processes access a single shared file, or per-process private files

Although IOR covers most important aspects of HPC I/O parameters, it does make certain simplifications. For example, the request sizes for different variables a parallel simulation writes out may not be uniform. In our future work, we plan to assess the impact of such simplification on our model prediction accuracy and investigate ways to allow more detailed characteristics specification if necessary.

To extract parameters representing application's I/O characteristics, one can use existing profiling/tracing tools [22, 7, 45] to instrument I/O primitives of the application, followed by trace collection/analysis. We include a simple tool for collecting ACIC-relevant application I/O characteristics encompassing a tracing library and scripts for parsing and statistically summarizing I/O traces [26].

3.3 Defining Exploration Space

Name	Value	Rank
Disk device	{EBS, ephemeral}	10
File system	{NFS, PVFS2}	5
Instance type	{cc1.4xlarge, cc2.8xlarge}	12
I/O server number	{1, 2, 4}	3
Placement	{part-time, dedicated}	7
Stripe size	{64KB, 4MB}	6
Num. of all processes	{32, 64, 128, 256}	14
Num. of I/O processes	{32, 64, 128, 256}	4
I/O interface	{POSIX, MPI-IO}	9
I/O iteration count	{1, 10, 100}	13
Data size	{1, 4, 16, 32, 128, 512 (MB)}	1
Request size	{256KB, 4MB, 16MB, 128MB}	8
Read and/or write	{read, write}	2
Collective	{yes, no}	11
File sharing	{share, individual}	15

Table 1: The variables affecting performance and cost. The top 6 variables are I/O system options in cloud, while the other ones are workload characteristics.

Table 1 summarizes the system I/O configurations and application I/O characteristics considered in this ACIC prototype. We set the range of the values according to our real-world application test cases with different job scales (32 to 256). For each parameter, we sample its value range in our training. For example, the compute-node-to-I/O-server ratio typically varies between 4 : 1 and 64 : 1 on a HPC cluster, which differs a lot from distributed file systems like GFS [15] and HDFS [43]. Since there are at most 16 instances in our testbed, we select 1, 2 and 4 as sampled values of the "I/O server number" parameter. For continuous (numerical) domain parameters, such as *data size* and *request size*, we select samples from their value ranges that form evenly spaced vectors in log space. Such training is used in our study to bootstrap ACIC's auto-configuration. Again, this design allows users to constantly contribute training data points to the ACIC training database.

In Table 1, the "rank" column gives their *relative importance* determined by the PB matrices, as to be discussed in Section 4.1. Though we have left out a number of parameters

and sampled the numerical parameter space rather sparsely, the concatenated exploration space combining system configurations and application characteristics is still daunting. Even considering that not all sample parameter value combinations are valid (e.g., NFS does not have *Strip size*; *request size* cannot be greater than *data size*), the 15 parameter dimensions create roughly *a million valid training data points.*[1] The next section presents how ACIC tackles this high-dimensional training space challenge.

4. PERFORMANCE/COST PREDICTION

4.1 Exploration Space Reduction

Row	*PBM*					Perf.
	A	**B**	**C**	**D**	**E**	
1	+1	+1	+1	-1	+1	19
2	-1	+1	+1	+1	-1	21
3	-1	-1	+1	+1	+1	2
4	+1	-1	-1	+1	+1	11
5	-1	+1	-1	-1	+1	72
6	+1	-1	+1	-1	-1	100
7	+1	+1	-1	+1	-1	8
8	-1	-1	-1	-1	-1	3
Effect	40	4	48	152	28	
Rank	**3**	**5**	**2**	**1**	**4**	

Table 2: Sample PB design working with $N = 5$ and $N' = 8$

To tackle the aforementioned high-dimensional parameter space, ACIC employs a statistical technique called *Plackett and Burman (PB) design* [38]. It helps ACIC identify the relative importance of the parameters, each constructing one dimension of the concatenated *system configuration + application characteristics* space.

Proposed originally for purposes such as agricultural crops experiment design and quality control in manufacturing, PB design screens combinations of N parameters (factors) with N' *runs*, where N' is the smallest multiple of 4 above or equal to N. For each run, the value for each parameter is set according to one row of the PB Matrix, whose elements are assigned with binary values (either "+1" or "-1") based on pre-specified PB design rules. More specifically, given a PB Matrix A, $A_{i,j}$ determines the value of the jth parameter in the ith run. This parameter will use a "high" value if $A_{i,j}$ is "+1", and a "low" one if otherwise. The "high" and "low" values are selected to be at the two ends of the parameter value range. After the runs are completed, the importance ("effect") of the jth parameter is calculated as the dot product of the jth column in A (the "+1" and "-1" setting across the runs for this parameter) and the result column (e.g., performance measurement from the N' runs). The sign of the result is meaningless when ranking the parameters in order of their perceived impact.

Table 2 illustrates the construction of a small PB Matrix, where there are 5 parameters ($N = 5$) and 8 runs ($N' = 8$). Compared to other statistical tools, PB design has the advantage of requiring only a small set of experiments (around N, the total number of parameters/dimensions) [53]. Specifically for our cloud performance training purpose, it allows us to find out parameters that are most influential to our optimization goal(s) with relatively small cost. It also ranks the parameters, enabling a fast (though less accurate) training to bootstrap the ACIC prediction. This way, ACIC populates its training database by sampling the top-ranked parameters first (adopting default settings for the other parameters), then gradually expands training data collection to the lower-ranked dimensions.

Like in the prior work by Yi et al. [53], we adopted in ACIC the improved variation called *foldover PB design* [33]. Foldover PB design further examines the effects of interactions between parameters, at the cost of doubling the number of runs. In this proof-of-concept study, we built the ACIC foldover PB Matrix (PBM) for the 15-dimensional exploration space given in Table 1, with $N = 15$ and $N' = 16$, requring only $N' \times 2 = 32$ runs. For non-binary (numerical) parameter value ranges, we selected the high and low values for all surveyed applications. We carried out the 32 test runs with IOR on the cloud storage system configured according to the PBM rows. The rightmost column in Table 1 gives the importance ranking. The results show that the most important three parameters are "I/O data size", "I/O operation type", and "number of I/O servers", while the least important ones are "whether file sharing is on", "number of all processes", and "I/O iteration count". Such ranking enables ACIC to explore the most influential parameters first. Our evaluation results in the next section will discuss the tradeoff between prediction accuracy and training data collection cost, as guided by the PB design results.

4.2 CART-based Prediction Model

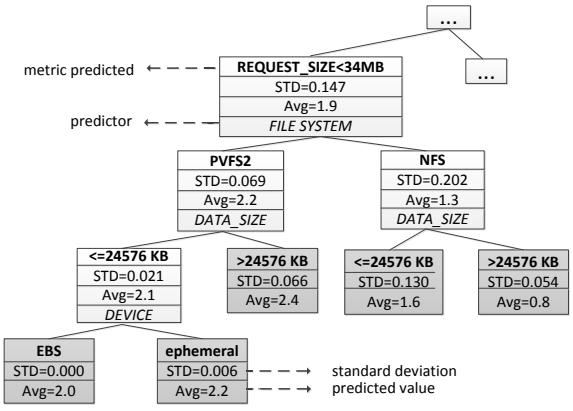


Figure 4: Sample tree built by ACIC using CART

Given the data points collected from IOR training runs guided by PB design, ACIC can then employ different black-box prediction tools. Many machine learning algorithms can help ACIC learn the mapping between I/O system/application parameters and the optimization goal. This problem falls under the general scope of *supervised learning*, and further under *regression*, as the prediction results are continuous numeric values. As supervised learning itself is a quite mature field and is beyond the scope of this paper, we adopt a well known technique to assess the feasibility of ACIC's reusable training. Meanwhile, ACIC is implemented in the way that different learning algorithms can be easily plugged in.

The current ACIC prototype uses CART (Classification and Regression Trees) [35] for its simplicity, flexibility, and interpretability. It is a decision tree based approach, requir-

[1] 2 * 2 * 2 * 3 * 2 * 2 * 4 * 4 * 2 * 3 * 6 * 4 * 2 * 2 * 2 = 1,769,472.

ing no knowledge about the prediction target, with trees built top-down recursively. At each stop in the recursion, the CART algorithm determines which predictor parameter in the training data best splits the current node into leaf nodes, then continues recursively within each subtree. The optimal split minimizes the difference (e.g., root mean square) among the samples in the leaf nodes. The error for each sample is the difference between it and the average of all samples in the leaf node. Therefore, each internal node contains a "best" predictor, while each leaf node gives a predicted target result. Eventually, the optimal decision tree is pruned to avoid over-fitting. To make a prediction, the tree takes a set of parameter values as input, and outputs the predicted target value dictated by the destination leaf node as it follows the path dictated by a sequence of internal nodes.

With ACIC, we face the problem of performance reporting mismatch between IOR and the target application requesting I/O configuration optimization. It is unrealistic to assume that the applications can be modified to report I/O performance in a way consistent with IOR. We solve this problem by adopting *performance/cost improvement* (over a *baseline configuration*) as the predicted target rather than using absolute values. The idea is similar to the "relative" notion in storage performance modeling [30]. In our implementation and experiments, we set the baseline configuration as "single dedicated NFS server, mounting two EBS disks with a software RAID-0", which is indeed the cloud version of a highly common shared storage setup with small- to medium-scale clusters [20, 12, 55].

Figure 4 shows a portion of the tree that models the I/O operation cost built by ACIC. The light-shaded nodes are internal nodes while the darker ones are leaves. Each level of the tree (composed by nodes with the same depth) examines the value of one dimension in the parameter space. For internal nodes, the first field contains the current-level parameter value range (such as "<=24576KB"), automatically calculated by CART to guide the decision making given the input parameter. The second field contains the standard deviation of the target value of all of its children and the third field contains the average value. The last field indicates the next-level parameter for branching its children into two subtrees. The leaf nodes report the predicted target value (both average and standard deviation).

Note that CART also arranges the ordering of parameters, by placing the ones it considers more "important" to decision making higher up (closer to the root). However, this is not redundant with the PB design generated ranking, as the former can only create such ranking based on collected training data, while the latter gives direction to training data collection itself.

In our cloud storage configuration context, given the target application, ACIC joins the application's I/O characteristics with all candidate I/O system configurations considered, as the input to the CART model. As the prediction overhead is negligible compared to the training data collection cost, a full exploration of system configuration space is affordable here. The candidate configurations are then sorted by their relative improvement over the baseline configuration, based on the CART prediction. ACIC can be configured to report the top k predicted optimized candidates. When $k > 1$, the application user has a better opportunity to identify an optimal or near-optimal solution, at the cost of more benchmarking runs trying out the top k configurations.

4.3 PB-guided Space Walking

Although PB design is able to help ACIC to reduce the parameter space to a rather practical level, we'll see in Section 5.3 that to bootstrap the CART based prediction model we still need to collect a substantial number of training data points. This could happen when ACIC starts collecting data on a new cloud platform or there is a major hardware overhaul rendering most of the collected data points obsolete. It's desirable to avoid the time and monetary cost of such bootstrapping, or the relatively inaccurate prediction by ACIC before having a properly populated training database. Therefore, we designed an alternative *PB-guided space walking* approach that can quickly return prediction results to application users. This alternative approach also allows us to further examine the trade-off between training cost and prediction accuracy in evaluating ACIC.

PB-guided space walking reuses the parameter ranking results generated by PB design experiments. The basic idea is to "walk" the I/O configuration space, given application I/O characteristics parameters, by selecting an optimized configuration one dimension at a time. More formally, the PB-guided space walking process can be expressed as a triple $\langle S,\ s_0,\ \delta \rangle$, as defined in *state space exploration*, one of the key techniques for computer-aided verification [23].

S, the "space", comprises the set of all possible points (configuration candidates in our scenario) the walk might reach. Note that here this space only contains the I/O system configuration parameters. Just like in the case of CART, certain parameter combinations are invalid, such as NFS with multiple I/O servers. s_0, the walking start point, is the *baseline* I/O configuration (see Section 4.1 for details). It is also used as a reference configuration in assessing the performance/cost improvement achieved by ACIC. Finally, δ, the heuristic walking strategy, determines how we advance from one parameter dimension to the next (the "walking direction"). ACIC adopts a greedy search algorithm, walking through the I/O configuration dimensions according to the PB design generated parameter ranking by iteratively selecting the current dimension parameter value. In each step, ACIC will run IOR tests that sample the current parameter dimension. Based on the results, it fixes the parameter value at the one that delivers the best target result (execution time or cost). The walk then continues to the next I/O configuration dimension, eventually reaching a heuristic solution as indicated by the selected walking path.

Obviously, the PB-guided space walking explores a much-trimmed parameter space, delivering prediction to application users with low training requirement. The walking-based prediction itself is application-specific: one application's training data collected through the walking may not be of much use to another application if they diverge early in their walking process. However, the IOR training data points collected are of generic interest to the ACIC database, and can be used later in either the CART-based or walking-based prediction. This way, the PB-guided space walking approach nicely complements the CART-based prediction.

5. EVALUATION

5.1 Experiment Setup

Platform All our experiments are performed on Amazon EC2 Cluster Computing Instances (CCIs), with node type *cc2.8xlarge* [3]. Each such instance has two 8-core Intel Xeon processors and 60.5GB of memory. The CCIs are inter-connected with 10-Gigabit Ethernet. Regarding OS and system software, we use the Amazon Linux OS 201202, Intel compiler 11.1.072 and Intel MPI 4.0.1. The compiler optimization level is $O3$.

Name	Field	CPU	Comm.	R/W	API
BTIO	Physics	H	H	W	MPI-IO
FLASHIO	Astro	L	L	W	MPI-IO
mpiBLAST	Biology	M	M	R	POSIX
MADbench2	Cosmology	L	M	RW	MPI-IO

Table 3: Test applications' resource usage and I/O type (H=High, M=Medium, L=Low, R=Read, W=Write)

Applications It is highly time and money consuming to run I/O-intensive parallel applications to evaluate ACIC. This is not due to ACIC's own overhead, but the fact that we perform *exhaustive* evaluation of all candidate configuration settings to evaluate its optimization effectiveness. In addition, we run each experiment several times, with cache content cleared in between. Given such time/cost constraints, we select four representative applications with different I/O characteristics, from different scientific computing domains. Table 3 shows their major I/O characteristics and computation/communication intensity levels.

BTIO is an I/O-enabled version of the BT benchmark in the NAS NPB suite [50], solving 3-D Navier-Stokes equations. The BT problem size used in our experiment is class `C` for all tests, with collective I/O turned on. With its default step size (200 steps) and I/O frequency (every 5 steps), each test run generates a shared output file of about 6.4GB.

FLASHIO is an I/O kernel derived from the full parallel FLASH simulation, a modular adaptive mesh astrophysics code [56]. It uses the parallel HDF5 I/O library to a single checkpoint file around 15GB into disk periodically.

mpiBLAST [11] is a parallel implementation of the widely used NCBI BLAST tool [34], for protein or DNA sequence search. In our tests, the 84GB *wgs* database is partitioned into 32 segments and there are around $1K$ query sequences sampled from itself. Unlike parallel simulations (most common scientific applications), mpiBLAST has a rather read-intensive I/O pattern [25]. We use the `use-virtual-frags` and `replica-group-size` settings to tune the number of processes reading the database (called *I/O processes*).

MADBench2 is a "stripped-down" version of the MADspec code, used in analyzing the Cosmic Microwave Background (CMB) radiation datasets [10]. A matrix is written to disk once after each computation step and read back when it is required in a demand-driven fashion, creating both read and write workloads. In our experiments, the output file is up to 32GB, accessed four times throughout the execution.

5.2 Optimal I/O Configurations

To evaluate ACIC, we need to actually measure the above applications' performance (depicted with total run time) and monetary cost running on EC2, using each of the candidate I/O configurations. Table 4 shows the optimal I/O configurations we found, with performance (overall execution time) as the optimization goal. The results showcase the lack of one-size-fits-all I/O configurations, with 7 unique optimal I/O configurations for 9 application runs. This means that even for the same application, different job sizes (numbers of processes) will call for different I/O system settings. Taking mpiBLAST as an example, the optimal configuration for 32-process runs adopts part-time I/O servers, while the one for 128-process runs adopts dedicated. One possible reason is that with a smaller number of processes, the locality effect brought by the part-time I/O servers outweighs other I/O system options. This is less likely to happen on today's in-house clusters, whose interconnect often use dedicated high performance network like InfiniBand. Even with a moderate 5-D configuration space, it is hard for users to manually explore the impact of parameter values and their interplay, as demonstrated in our user study (Section 6). Due to space limit, we omit the best configurations for cost optimization, where the results show similar behavior and in many cases the best configuration for performance does not agree with that for cost optimization.

Application	NP	Device	P/D	FS	IOS	SS
BTIO	64	EBS	P	NFS	1	NA
	256	eph.	P	PVFS2	4	4MB
FLASHIO	64	eph.	D	NFS	1	NA
	256	eph.	P	NFS	1	NA
mpiBLAST	32	eph.	P	PVFS2	4	64KB
	64	eph.	D	PVFS2	4	4MB
	128	eph.	D	PVFS2	4	4MB
MADbench2	64	eph.	D	PVFS2	4	4MB
	256	EBS	D	PVFS2	4	4MB

Table 4: Optimal performance configurations for different applications with different scales. Column names: **NP** - Number of I/O processes; **Device** - Disk device; **P/D** - I/O server placement, part-time (P) or dedicated (D); **FS** - File system; **IOS** - Number of I/O servers; **SS** - Stripe size for PVFS2; eph. - ephemeral disk

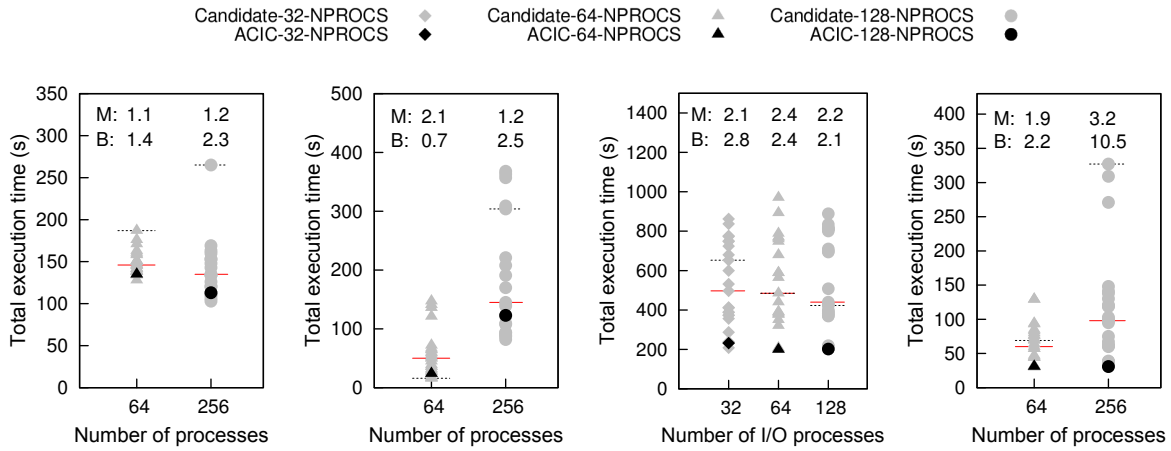
5.3 ACIC Auto-Configuration Effectiveness

Figure 5 and Figure 6 show the execution time and cost distribution, respectively, for the evaluated 9 application executions. The monetary cost for each cloud execution is:

$$cost = execution\_time \times num\_instances \times unit\_price \quad (1)$$

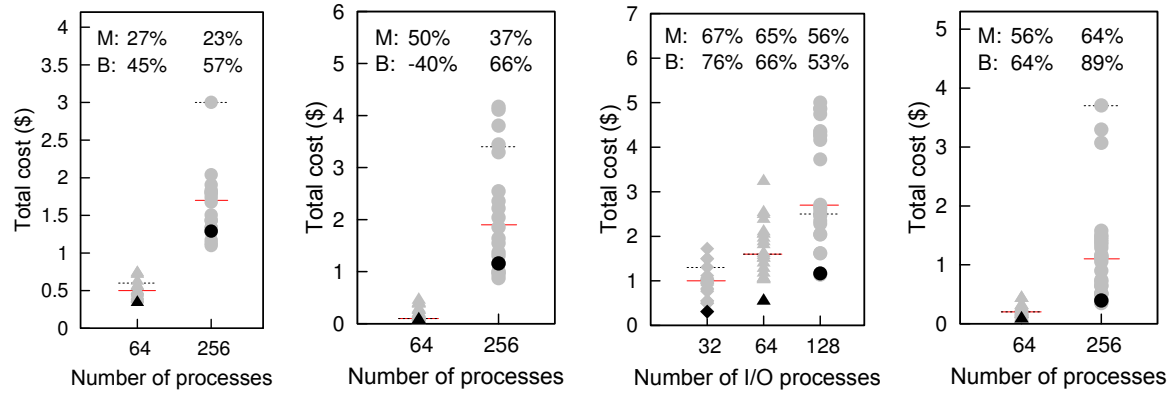
As mentioned earlier, we exhaustively tested *all* candidate configurations, each indicated by a gray dot, whose vertical span depicts the range of performance/cost measurement for the entire configuration space. The lowest dot in each figure is the measured optimal configuration. The black data points highlight the target measurement achieved under the ACIC recommended I/O configuration. The first 10 parameters are used in the training, according to the PB design experiment results. When the CART model gives several configurations as co-champions, we report the median results using these configurations. For each application setting, the solid (red) line indicates the median performing I/O configuration's position among the gray dots and the dashed (black) line marks the performance of the baseline I/O configuration. As described Section 3.3, the baseline we used is "dedicated NFS server mounting two EBS disks with a software RAID-0", a configuration similar to the baseline setup of many small- to medium-sized in-house clusters.

First, these figures clearly demonstrate the potentially large difference, caused by different I/O system configurations, in *overall execution time* (not total I/O time) and monetary cost of running data-intensive applications in the



(b) BTIO (c) FLASHIO (d) mpiBLAST (e) MADbench2

Figure 5: Total execution time of test applications. In each set of application run, the black dot indicates the ACIC predicted best configuration's performance and the gray dots indicate performance of all candidate configurations. The solid (red) line marks the median (M) performance among all configuration candidates, while the dashed (black) line marks the performance of the baseline (B) I/O configuration. Speedup ratios achieved by ACIC over the median and baseline performance are shown at the top of each figure.



(a) BTIO (b) FLASHIO (c) mpiBALST (d) MADbench2

Figure 6: Total monetary cost of running the test applications. Costa saving percentages are listed at the top of each figure.

cloud. More specifically, we see the performance difference ranging between $1.4x$ and $10.5x$, and cost difference between $2.2x$ and $10.5x$. Second, at a glimpse, ACIC is able to identify near-optimal I/O configurations in almost all situations, as the black points are located near the bottom of the gray "spectrum". At the top of each chart, we note the improvement achieved by the ACIC-recommended configuration over the median ("M"/solid line) and the baseline configuration ("B"/dashed line). For performance, we used *speedup*, calculated as:

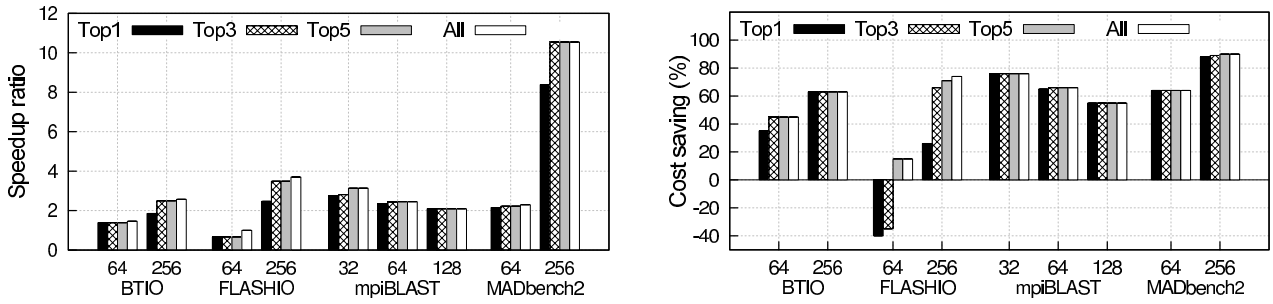
$$speed\_up = \frac{time_{baseline/median}}{time_{ACIC}}. \quad (2)$$

For cost, we report

$$cost\_saving = \frac{cost_{baseline/median} - cost_{ACIC}}{cost_{baseline/median}} \times 100\% \quad (3)$$

In all cases, the ACIC-recommended configuration outperforms the median configuration, by a factor of 1.1-3.2 in execution time, while delivering a cost saving of 23%-67%. It also beats the baseline configuration most of the time. There is an exception of FLASHIO using 64 processes, where the baseline configuration happens to be near-optimal itself. Moreover, the absolute values of execution time (and hence cost) are relatively small, leading to a substantial negative cost saving in this case.

Next, we examine the potential difference made by verifying a larger ACIC recommendation set, an optional effort users can make by running their applications with not one, but the top-k recommendations. As mentioned earlier, users may have "residual resource" left from their hourly cloud instance rentals and can piggy-back verification runs at no extra cost. Figure 7 shows the execution time and cost improvement achieved by the best configuration among the top 1, 3, and 5 recommendations and eventually all I/O configurations (the true optimal). The results reveal that actually the top recommendation (median if there are co-champions) works fairly well, though considering more top candidates does help with several cases (eg. 256-process FLASHIO).



(a) Execution time (over baseline)

(b) Total cost (under baseline)

Figure 7: Accuracy enhancement from examining top-k ACIC recommendations

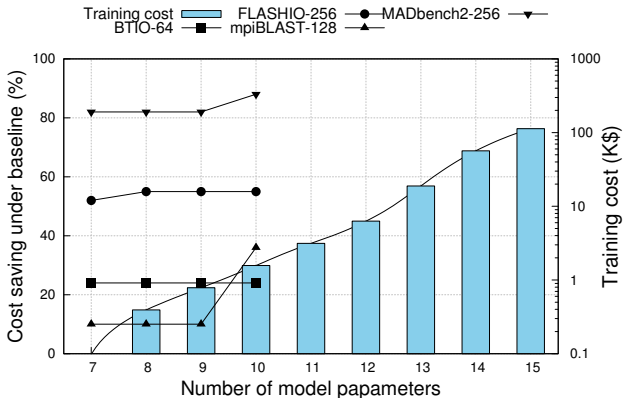


Figure 8: Impact on prediction performance using different numbers of top ranking model parameters

In particular, in almost all cases, little further gain can be achieved by checking beyond the top 3 recommendations.

5.4 Training Cost Analysis

The ACIC overhead includes three types of cost, caused by its profiling, training data collection, the actual prediction. Among them, the most significant item is definitely the training data collection through IOR runs on the cloud, which incurs time overhead larger than the other two by orders of magnitude, and could be expensive money wise. More training data points, however, typically lead to higher prediction accuracy. To investigate this tradeoff, we experimented with CART-based prediction using different numbers of configuration parameters (dimensions), as guided by PB design results.

Figure 8 presents the results of this sensitivity study using four sample runs, one for each application. The x axis indicates the number of top ranking parameters used in model training as ordered by PB matrix. For each parameter count, the y axis on the left measures the performance of the ACIC top recommendation in terms of cost saving under the baseline, while the y axis on the right measures the cost of training data collection. Note that the left axis is linear scale and the right is log scale. When using 10 parameters, the total training data collection cost is around \$1K.

The results here show that we can still achieve considerable cloud application execution cost saving, with only the top 7 parameters (which requires a training data collection cost of only \$108). Meanwhile, we do observe higher optimization effectiveness when considering more parameters (by collecting more training data points), though the gain appears to be heavily application-dependent. As expected, the estimated training data collection cost continues to grow exponentially beyond 10 parameters, reaching \$100K when exploring the full 15-D space. Due to time/funding constraints, we did not perform more training than the top 10 dimensions, and do not expect such additional exploration will bring significant gain, as shown in Figure 7.

5.5 Comparison with PB Space Walking

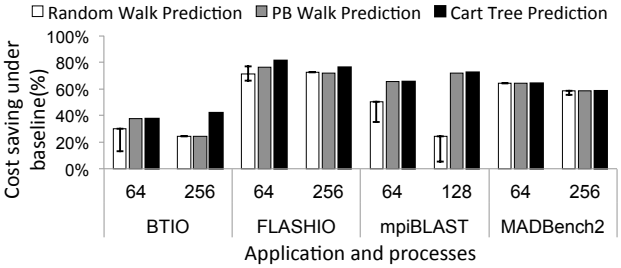


Figure 9: Comparing alternative prediction approaches

Finally, we compare the auto-configuration capability of the CART-based and the PB-guided space walking prediction, again in terms of cost saving over the baseline configuration. Here we compare three prediction methodologies. The first is *random walk*, which randomly selects the ordering of the I/O configuration parameters in its dimension-by-dimension training and prediction. For this approach, we report the average results from 10 predictions with different random parameter orderings, with the y error bars depicting the range of cost saving distribution. The second is the PB-guided walk, as proposed in Section 4.3. The third in black is the CART-based prediction.

Figure 9 shows the CART-based prediction delivers the best optimization results consistently. The PB-guided space walking closely follows in most cases, benefiting from the guidance of PB designs and application-specific training. The random walking approach, on the other hand, generates significantly inferior as well as less predictable optimization performance in half of the cases. The results confirm that PB-guided walking is an appealing approach when the ACIC training database has not been sufficiently populated.

5.6 Observations From Training Experience

In addition to releasing our ACIC tool, here we share the major observations based on our extensive initial training with roughly 10K data points from EC2:

1. It is more cost-effective to use part-time than dedicated I/O servers for applications with I/O aggregators, where each communication group has a root process that collects data and writes them locally. In particular, data locality can be much enhanced when placing the part-time I/O servers on the same physical instances as the aggregators.
2. For parallel file system like PVFS2, having more I/O servers improves performance of both cost and time perspective. Across all four applications, we found few cases where one PVFS2 I/O server performs better than four ones.
3. Ephemeral disks usually perform better than EBS when there is more than one I/O server deployed.
4. NFS often works better for applications performing small amounts of I/O using POSIX API.
5. It is important to tolerate server connection failures on a cloud platform for production runs. We experienced lost connections to the I/O server, causing data corruption, in around 1‰ of experiments during training.

6. USER STUDY

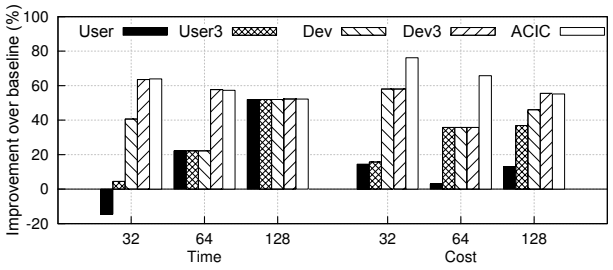


Figure 10: Comparing manual configurations with ACIC

To further verify ACIC's benefit of automated I/O configuration optimization, we performed a small-scale subject study. We used one of our test applications, mpiBLAST, as we obtained consent from one of its core developers [25] ("Dev"), plus one of its skilled users [51] ("User"), to participate in our evaluation. It is challenging to do a larger study due to the difficulty in finding (expert) users/developers of I/O-intensive parallel applications, simultaneously with cloud execution experience and time to participate. We provided the participants with sufficient information regarding the executions (such as input and job scale) and the platform (such as pricing policy and device performance). Based on their knowledge and experience, the participants each gave the optimal configurations manually selected. E.g., the user gave a configuration of "Eph.-P-NFS-1-4MB" for cost minimization of 32-process runs, while the developer gave a configuration of "Eph.-D-PVFS2-2-4MB" for performance optimization of 64-process runs.

Figure 10 shows the improvement of ACIC's predicted configuration and the manually selected ones. Across all execution scales and both optimization goals, ACIC consistently provides better suggestion than the experienced human participants, beating the user by an average of 37.43% and the developer by 17.8%. In addition, both developer and user agree with each other in three out of the six test groups, confirming the impact of common knowledge. However, in two of the rest three test groups, their selections generate highly contrasting results, indicating the limitation and unreliability of manual configurations. We also invited them to give 3 configurations for each test group provided with the insights in Section 5.6, and then compared the ACIC with the top-3 manual configurations (denoted as "Dev3" and "User3"). While the execution time of the top-3 manual configurations by the developer can match the ACIC performance, the manual top-3 configurations visibly lag behind ACIC (36% for user and 17% for developer on average).

7. RELATED WORK

In this section, we briefly discuss several lines of prior work related to the ACIC approach.

Parameter space reduction PB design has been applied to computer systems. For example, Yi et al. [53] employ it to identify key processor parameters for massive simulations. Actually, CART models have also been used as attribute filters to prune the similarity search space [48]. The novelty in this work, however, lies in the combination of PB-based space reduction with multiple machine learning approaches (including CART) to enable cost-effective, reusable model training for black-box performance/cost prediction.

Cloud system configuration Recently several approaches have been developed to optimize cloud platform configurations. Gideon et al. [20] study the impact of different data sharing options for scientific work-flows on Amazon EC2. *Elastisizer* [17] selects the proper cluster size and instance types for MapReduce workloads running in the cloud. DOT [18] is a model analyzing large data analytic software and offering optimization guidelines. Most of these existing efforts assume certain knowledge on the application/middleware internals, while ACIC is based on black-box prediction and can assist many applications with diverse I/O behaviors. Also, ACIC offers the flexibility and expandability that allow it to work across cloud platforms and across hardware updates. The recently proposed *Scalia* [37] is a cloud storage brokerage solution that continuously adapts data placement based on the access pattern and optimization objectives (e.g. storage costs). It focuses on cross-cloud placement and estimates cost using longer-term access statistics. In contrast, ACIC, while capable of multi-cloud optimization, is designed specifically to address the high-dimensional space optimization problem for individual application.

Storage provisioning tools There are tools aiming at reducing the human effects involved in storage system provisioning and management. For instance, Hippodrome [4] and Minerva [2] perform automatic block-level cluster storage tuning. *scc* [28] automates cluster storage configuration based on formal specifications of application behavior and hardware properties. Our work complements such prior work by addressing the unique storage system configuration space opened up by cloud and the training cost challenge brought by the high-dimensional configuration space.

Prediction model Many studies exist on performance

modeling for HPC applications and/or I/O systems [19, 46, 52, 54]. Some models were proposed for multi-platform performance prediction [29, 21, 6, 27]. Nikolaus [19] et al. demonstrated that the Palladio Component Model can predict the performance of industry workload with system using virtual storage. One of the most closely related projects is by Osogami et al. [36], who optimized web system performance by heuristically searching the configuration space to automatically predict the performance based on the model measured similar configurations [36]. In addition, there is *Pesto* [16], a unified storage performance management system that automatically constructs approximate black-box performance models of storage devices. Compared to these studies, our work focuses on the unique high-dimensional black-box modeling of cloud performance and the associated training cost challenge.

8. CONCLUSION AND FUTURE WORK

In this paper, we demonstrate that cloud I/O system configurations have considerable impacts on both the performance and cost efficiency of I/O intensive parallel applications. We further propose ACIC, an automatic cloud I/O system configuration tool for HPC applications. ACIC combines several statistical and machine learning techniques to enable application-dependent, incremental model training and black-box performance/cost prediction. In particular, we have found that the PB design approach, which effectively trims the parameter exploration space and reduces the high-dimensional model training to a feasible task, works well in conjunction with regression tree and space walking. Our evaluation results demonstrate that accurate I/O configuration can be predicted with a significantly reduced exploration dimension, without requesting users to perform application-specific manual tuning or benchmarking.

In the future, we plan to explore web-based ACIC query service. We also hope to assess the extensibility of ACIC to support incrementally new I/O configurations or application characteristics parameters, as well as additional cloud platforms.

Acknowledgments

We sincerely thank the anonymous reviewers for their valuable comments and suggestions. We also thank Frans Kaashoek and Xianhe Sun for their useful early feedback on our work. Special thanks goes to Heshan Lin and Ruini Xue for taking the time and effort to participate in our user study on mpiBLAST. In China, this work has been partially supported by the National High-Tech Research and Development Plan (863 project) 2012AA01A302, as well as NSFC project 61133006 and 61103021. In the US, the work has been partially sponsored by multiple NSF awards (CNS-0546301, CNS-0915861, and CCF-0937908), an IBM Faculty Award, and Xiaosong Ma's joint appointment between ORNL and NCSU.

9. REFERENCES

[1] GPFS: A shared-disk file system for large computing clusters.

[2] G. Alvarez, E. Borowsky, and S. e. a. Go. Minerva: An Automated Resource Provisioning Tool for Large-scale Storage Systems. *ACM Transactions on Computer Systems (TOCS)*, 19(4):483–518, 2001.

[3] Amazon Inc. High Performance Computing (HPC). http://aws.amazon.com/ec2/hpc-applications/, 2011.

[4] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running Circles Around Storage Administration. In *FAST*, 2002.

[5] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[6] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A Scalable Cross-platform Infrastructure for Application Performance Tuning Using Hardware Counters. In *SC*. IEEE, 2000.

[7] S. Byna, Y. Chen, X.-H. Sun, R. Thakur, and W. Gropp. Parallel I/O Prefetching Using MPI File Caching and I/O Signatures. In *SC*. IEEE, 2008.

[8] B. Callaghan. *NFS Illustrated*. Addison-Wesley Longman Ltd., Essex, UK, 2000.

[9] P. Carns, W. L. III, R. Ross, and R. Thakur. PVFS: A Parallel File System For Linux Clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, 2000.

[10] Computational Research Division. Madbench2. `http://crd-legacy.lbl.gov/~borrill/MADbench2/`.

[11] A. Darling, L. Carey, and W. Feng. The Design, Implementation, and Evaluation of mpiBLAST. In *Proceedings of the ClusterWorld Conference and Expo*, 2003.

[12] C. Evangelinos and C. Hill. Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. *ratio*, 2(2.40):2–34, 2008.

[13] M. Fahey, J. Larkin, and J. Adams. I/O performance on a massively parallel Cray XT3/XT4. In *IPDPS*. IEEE, 2008.

[14] M. Folk, A. Cheng, and K. Yates. HDF5: A File Format and I/O Library for High Performance Computing Applications. In *SC*, volume 99, 1999.

[15] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. In *SOSP*. ACM, 2003.

[16] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, and M. Uysal. Pesto: Online Storage Performance Management in Virtualized Datacenters. In *SOCC*, page 19. ACM, 2011.

[17] H. Herodotou, F. Dong, and S. Babu. No One (cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics. In *SOCC*. ACM, 2011.

[18] Y. Huai, R. Lee, S. Zhang, C. H. Xia, and X. Zhang. DOT: A Matrix Model for Analyzing, Optimizing And Deploying Software for Big Data Analytics in Distributed Systems. In *SOCC*. ACM, 2011.

[19] N. Huber, S. Becker, C. Rathfelder, J. Schweflinghaus, and R. H. Reussner. Performance Modeling in Industry: A Case Study on Storage Virtualization. In *ICSE*. ACM, 2010.

[20] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling. Data Sharing Options for Scientific Workflows on Amazon EC2. In *SC*, 2010.

[21] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini,

H. J. Wasserman, and M. Gittings. Predictive Performance And Scalability Modeling of A Large-scale Application. In *SC*. ACM, 2001.

[22] A. Konwinski, J. Bent, J. Nunez, and M. Quist. Towards An I/O Tracing Framework Taxonomy. In *PDSW*. ACM, 2007.

[23] L. M. Kristensen and L. Petrucci. An Approach to Distributed State Space Exploration for Coloured Petri Nets. In *ICATPN*. Springer, 2004.

[24] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock. I/O Performance Challenges at Leadership Scale. In *SC*. ACM, 2009.

[25] H. Lin, X. Ma, W. Feng, and N. Samatova. Coordinating Computation and I/O in Massively Parallel Sequence Search. *IEEE Transactions on Parallel and Distributed Systems*, 22(4):529–543, 2011.

[26] M. Liu, Y. Jin, J. Zhai, Y. Z. Q. Shi, X. Ma, and W. Chen. ACIC Homepage. `http://hpc.cs.tsinghua.edu.cn/ACIC`, 2013.

[27] X. Ma, M. Winslett, J. Lee, and S. Yu. Improving MPI-IO Output Performance with Active Buffering Plus Threads. In *IPDPS*. IEEE, 2003.

[28] H. Madhyastha, J. McCullough, G. Porter, R. Kapoor, S. Savage, A. Snoeren, and A. Vahdat. scc: Cluster Storage Provisioning Informed by Application Characteristics and SLAs. In *FAST*. USENIX, 2012.

[29] G. Marin and J. Mellor-Crummey. Cross-architecture Performance Predictions for Scientific Applications Using Parameterized Models. In *SIGMETRICS*. ACM, 2004.

[30] M. Mesnier, M. Wachs, R. Sambasivan, A. Zheng, and G. Ganger. Modeling the Relative Fitness of Storage. In *SIGMETRICS*. ACM, 2007.

[31] Message Passing Interface Forum. The Message Passing Interface (MPI) standard. http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf.

[32] Mingliang Liu and Jidong Zhai and Yan Zhai and Xiaosong Ma and Wenguang Chen. One Optimized I/O Configuration per HPC Application: Leveraging The Configurability of Cloud. In *APSys*. ACM, 2011.

[33] D. Montgomery. *Design and analysis of experiments*. John Wiley & Sons Inc, 1991.

[34] National Center for Biotechnology Information. NCBI BLAST. http://www.ncbi.nlm.nih.gov/BLAST/.

[35] L. Olshen and C. Stone. Classification and Regression Trees. *Wadsworth International Group*, 1984.

[36] T. Osogami and S. Kato. Optimizing System Configurations Quickly by Guessing at The Performance. In *SIGMETRICS*, 2007.

[37] T. Papaioannou, N. Bonvin, and K. Aberer. Scalia: An Adaptive Scheme for Efficient Multi-Cloud Storage. In *SC*, 2012.

[38] R. Plackett and J. Burman. The Design of Optimum Multifactorial Experiments. *Biometrika*, 33(4):305–325, 1946.

[39] A. Purakayastha, C. Ellis, D. Kotz, N. Nieuwejaar, and M. Best. Characterizing Parallel File-access Patterns on a Large-scale Multiprocessor. In *IPDPS*. IEEE, 1995.

[40] R. Rew and G. Davis. NetCDF: An Interface for Scientific Data Access. *Computer Graphics and Applications, IEEE*, 10(4):76–82, 1990.

[41] P. Schwan. Lustre: Building A File System for 1000-node Clusters. In *Proceedings of the 2003 Linux Symposium*, volume 2003, 2003.

[42] H. Shan, K. Antypas, and J. Shalf. Characterizing and Predicting the I/O Performance of HPC Applications Using a Parameterized Synthetic Benchmark. In *SC*. IEEE, 2008.

[43] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[44] J. Sim, A. Dasgupta, H. Kim, and R. Vuduc. A Performance Analysis Framework for Identifying Potential Benefits in GPGPU Applications. In *PPoPP*. ACM, 2012.

[45] N. R. Tallent, J. M. Mellor-Crummey, and M. W. Fagan. Binary Analysis for Measurement and Attribution of Program Performance. In *PLDI*. ACM, 2009.

[46] V. Taylor, X. Wu, and R. Stevens. Prophesy: An Infrastructure for Performance Analysis And Modeling of Parallel And Grid Applications. In *SIGMETRICS*. ACM, 2003.

[47] R. Thakur, W. Gropp, and E. Lusk. Data Sieving and Collective I/O in ROMIO. In *FRONTIERS*, 1999.

[48] E. Thereska, B. Doebel, A. Zheng, and P. Nobel. Practical Performance Models for Complex, Popular Applications. In *SIGMETRICS*. ACM, 2010.

[49] L. William, M. Tyce, and M. Christopher. IOR HPC Benchmark. `https://asc.llnl.gov/sequoia/benchmarks`, 2003.

[50] P. Wong and R. der Wijngaart. NAS Parallel Benchmarks I/O Version 2.4. *NASA Ames Research Center Tech. Rep. NAS-03-002*, 2003.

[51] R. Xue, W. Chen, and W. Zheng. CprFS: A User-level File System to Support Consistent File States for Checkpoint and Restart. In *ICS*. ACM, 2008.

[52] L. T. Yang, X. Ma, and F. Mueller. Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution. In *SC*. IEEE, 2005.

[53] J. Yi, D. Lilja, and D. Hawkins. A Statistically Rigorous Approach for Improving Simulation Methodology. In *HPCA*. IEEE, 2003.

[54] J. Zhai, W. Chen, and W. Zheng. Phantom: Predicting Performance of Parallel Applications on Large-scale Parallel Machines Using a Single Node. In *PPoPP*. ACM, 2010.

[55] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen. Cloud Versus In-house Cluster: Evaluating Amazon Cluster Compute Instances for Running MPI Applications. In *SC*. ACM, 2011.

[56] M. Zingale. FLASH I/O Benchmark Routine Parallel HDF5. `http://www.ucolick.org/~zingale`, 2001.