Predicting Data Structures for Energy Efficient Computing

Junya Michanan, Rinku Dewri, Matthew J. Rutherford
Department of Computer Science, University of Denver
Denver, Colorado U.S.A.
jmichana@du.edu, {rdewri, mjr}@cs.du.edu

***Abstract*—Dynamic data structures in software applications have been shown to have a large impact on system performance. In this paper, we explore energy saving opportunities of interface-based dynamic data structures. Our results suggest that opportunities do exist in the C5 Collection, at least 16.95% and up to 97.50%. We propose an architecture for building adaptive green data structures by applying machine learning tools to build a model for predicting energy efficient data structures. Our neural network model can classify energy efficient data structures based on features such as the number of elements, frequency of operations, interface and set/bag semantics. The 10-fold cross validation results show 96.01% accuracy on the training data and 95.80% on the training validation data. Our n-gram model can accurately predict the most energy efficient data structure sequence in 19 simulated and real-world programs—on average, with more than 50% accuracy and up to 98% using a bigram predictor.**

Keywords—Energy; Power; Efficiency; Performance; Machine Learning; Neural Network; N-Gram; Bigram; C5 Collection; Green Data Structure; Software Adaptation

I. INTRODUCTION

The proliferation of global computing devices has pushed IT energy consumption higher and higher, raising concerns among many environmentalists. As reported in [1], information and communication technology ecosystems have approached 10% of the world's electricity generation. The need for reducing IT energy consumption and research efforts in every sector is now necessary, not just for energy saving and extending battery life, but also for the environment. There are many areas in the layers of computer systems that can be optimized for energy efficiency, from hardware [3, 4], operating system [5] to application layers [12, 13]. By making "unpolluted design" of computer systems [2], hardware engineers and software developers can also play a significant part in the fight against climate change. In this study, we focus on the application layer, primarily object-oriented software with "interface-based" implementation [6], where there are multiple choices of classes that implement the same interface. Our goal is to explore how the choice of these classes impacts the energy consumption of software applications and to find ways to intelligently switch between implementations for energy efficiency.

As a case study, we investigate dynamic data structures because they have been shown to have a large impact on performance and are considered key components of many object-oriented software applications [7, 8]; many applications are implemented using interface-based design [6]. By using a "select the right data structure for the right workload" approach, we apply machine learning tools and enhance these classes to be adaptive green data structures that can dynamically adapt for energy efficiency without creating more work for developers. Another objective is to provide an architecture for building easy-to-use, smart, adaptive and green dynamic data structures that can be used in developing energy-efficient software applications. However, in this paper, we concentrate on developing models for predicting energy efficient data structures since these models are essential components in the adaptive green data structures.

The contributions of this paper are threefold: (1) empirical evidence that energy saving opportunities exist in interface-based, object-oriented dynamic data structures; (2) development of a predictive model based on artificial neural networks and n-gram inference to predict energy efficient data structures for use in object-oriented programs; and, (3) an architecture for building an adaptive green data structure.

II. BACKGROUND

A. Interface-Based Dynamic Data Structure

In object-oriented programming, one common way to reduce complexity and increase maintainability, reusability and flexibility of software systems is by using interface-based design [6]. Many modern programming languages, such as Java, C#, SmallTalk and C++, implement their class libraries using interface-based design where there are multiple choices of dynamic data structures with different implementations [8]—e.g.:array-based, linked-list-based, hash-based and tree-based [7]. Dynamic data structures are useful for managing internal data in algorithms of software application (creating, retrieving, updating and deleting data, for example) with the flexibility to grow or shrink memory requirements based on the number of elements in the structure. Different implementations of the data structures have been shown to have different impact on the performance and memory consumption of software applications [8, 9]. Each is designed differently and is intended for different workloads and usage. Thus, it is possible that by putting the right data structure with the right workload, performance and energy consumption of software applications can be improved.

We select the C5 generic collection (version 2.3) for this study because it contains interface-based dynamic data structures and is created using a "code-to-interface" implementation [9]. It is a comprehensive, open-source C# data structure library for the .NET framework, with solid documentation. It also provides a full complement of well-known abstractions such as lists, sets, bags, dictionaries, priority

queues, queues and stacks. For the case study, we explore how the choice of data structure impacts the energy consumption of software applications and what are the energy saving opportunities among these data structures. The predictive model and proposed architecture for building adaptive green data structures are also based on the C5 collection.

In programming, the selection of a data structure implementation is normally done at the development phase. Once a program is completed, the data structure choice is fixed. Dynamic data structure selection and switching are not normally done at runtime. Moreover, programmers tend to choose their favorite data structures [8] for their programs, often without taking performance and/or energy consumption into consideration, or knowing whether there are better choices. As a result, the energy consumption of computer systems when running the application can be higher than necessary. The process of selecting the most energy efficient data structure should be automatic and switching to a different implementation should be dynamic, without creating more work for programmers. Our long-term goal is to develop an architecture for adaptive green data structures that allows programmers to replace their existing data structure with a universal "green" data structure, and expect the programs to function the same, with minimal overhead and configuration.

B. Data Structure Features

In order to create the predictive model for use in the dynamic selection and switching processes, our study looks at data structure features that can impact performance and limit the selection choice. Based on [7], there are several features that influence the performance of dynamic data structures: number of existing elements, or size, and types and frequency of data structure operations. There are also features that can limit the selection choice: interface and data structure properties such as bag and set semantics, for example. For a program or algorithm that requires data structures with a sort method, the choice can only be made on some data structure classes that implement the interface with a sort method, limiting the number of applicable data structures. Moreover, if a requirement is to allow or not allow duplication of data elements, the data structures with bag or set semantic properties can also limit the selection.

TABLE I. THE SELECTED DATA STRUCTURE FEATURES

Features that can impact the performance of data structures	**Features that can limit the choice of data structures**
Number of elements in the data structure at the time of an operation (N)	Interface
Frequency of data structure operations	Bag/set semantics

Table I shows the selected data structure features in this study. For the features that impact the performance, we select number of elements in the data structure at the time of an operation (labeled as N), and frequency of insertion, deletion, query and update operations. We focus on the four operations because they are fundamental to a data structure. Many other operations, such as the sort operation, can be made up from combining these common operations together [7]. For features that limit the selection choice, we select interface, bag and set properties of dynamic data structures.

We also select only those C5 dynamic data structures that implement the ICollection interface in the C5 interface hierarchy (the full interface hierarchy can be found in [9]). ICollection implements the four common data structure operations. Under this interface, we select 9 (out of 12) dynamic data structures, as listed in Figure 1, grouped by interface, set and bag semantics. We call them data structure groups, and denote them by G. There are 6 data structure groups: ICollection, ICollectionBag, ICollectionSet, IList, IListBag, and IListSet. For example, in the ICollectionBag group, there are four dynamic data structures that implement the ICollection interface and have bag semantic property. Similarly, there are two and three selection choices in the IListBag and IListSet groups, respectively. The groupings demonstrate how selection choices can be different and vary based on application requirements. Our predictive model for energy efficiency also makes predictive data structure selection within each of these groups.

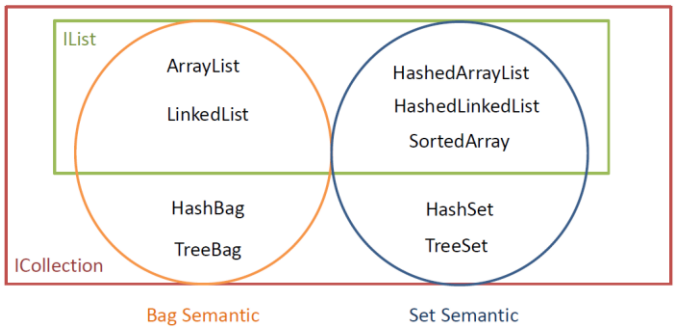

Fig. 1. C5 Data Structure Groups by Interface and Set/Bag Semantics.

Create, Retrieve, Update and Delete operations (also known as CRUD operations) are four basic functions of persistent storage and database-driven applications [10]. We consider insertion and addition of elements as the same operation, analogous to the create operation. We use percentage numbers, instead of counts, as the frequency of CRUD operations. These numbers are labeled as %C, %R, %U, and %D respectively. Since non-CRUD operations are ignored, these numbers always add up to 100. These numbers reflect the workload that a data structure is subjected to. Altogether, the group G, the size N, %C, %R, %U and %D are combined as input features to our model, and are the independent variables of our training and validation datasets.

C. Related Work

There are several studies related to our research. A closely related study is the Smart Data Structures project [11]. Their research is aimed at creating a new class of parallel data structures that leverage online machine learning to adapt automatically. However, the approach has many differences with our study. First, the objective there is to create a new set of data structures for parallel computing, while we attempt to apply adaptation techniques to select and transform the right data structure for the right workload from existing objected-oriented data structures, mainly for energy efficiency. As such, [11] concentrates on using online learning and self-adaptation techniques to improve speed in order to gain the energy savings.

The main technique is self-tuning of internal algorithms by changing parameters to get good performance. Instead, the knowledge for adaptation and online/offline learning of our models is based on actual energy consumption data of computer systems and done using statistical machine learning techniques.

In another related study, Edgar et al. introduce systematic, high-level, data structure transformations in the context of memory-efficient and low-power, embedded software design for dynamic multimedia applications [12]. Instead of using predictive models, the decision-making for the transformation is performed based on a Pareto tradeoff analysis between the data accesses and memory footprints. The speed and power consumption improvements are the gains from the decision results. The adaptation approach and methodology rearranges internal data structures for better memory footprint and data accesses. The power consumption data for the analysis are derived from a memory chip model, while our models are based on the energy consumption data measured by a power meter.

Our training and validation datasets are created from a custom energy profiler using a power meter to measure power consumption at the system level. However, modern processors have started embedding power measurement components in their design (e.g. Intel Power Gadget [23]) that can potentially be used to estimate processor power consumption of software applications at a finer granularity. We expect to explore this avenue in future, primarily to create training datasets, and provide real-time energy information to the adaptive green data structure for online learning.

There are other studies related to software adaptations for performance and/or energy efficiency. For example, the study in [13] extends the Odyssey platform to guide mobile applications to adapt for battery life. By monitoring energy supply and demand, the platform is able to select the correct tradeoff between energy conservation and application quality. Also, research on dynamic adaptive data structures for monitoring data streams focuses on changing a specific data structure representation for accuracy, speed of response and memory requirements [14]. And lastly, the study in [15] presents a method for auto-tuning programs with algorithmic choice. The main differences with our research are that these research projects either use different adaptation or learning techniques, or are not data structure related.

III. Adaptive Green Data Structure

Our vision is to see a new generation of software applications composed of smart/green objects and components. We envision that these adaptive objects and components have the ability to intelligently adapt themselves to the workloads and environments for energy efficiency, and become the main building blocks in developing green software applications. The green objects are smart because they can learn, classify and predict their workload, and can decide when and how to dynamically adapt for energy efficiency. This section explains the architecture of our adaptive green data structure.

A. Green CRUD-Based C5 Collection

The sought Green CRUD-based C5 Collection is an example of an adaptive green data structure. Figure 2 displays the high-level components of our proposed Green CRUD-based C5 Collection. It is an enhanced version of the C5 Generic Collection that wraps the 9 data structures into one, and adds the Green component to make it smart and energy-aware. There are two main components in our proposed adaptive green data structure—the CRUD-based C5 Collection and the Green component. The first is a wrapper/factory class of the 9 C5 data structures. It contains the public interface of the C5 data structures and has the ability to transform itself to different implementations at runtime, as directed by the Decision Maker of the Green component.

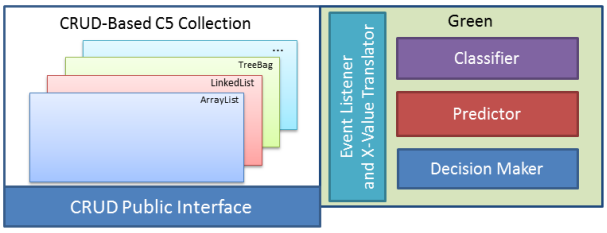

Fig. 2. A Component Architecture of an Adaptive Green CRUD-Based C5 Collection

The second component, the Green component, is composed of four main sub-components—Event Listener and X-Value Translator, Classifier, Predictor, and Decision Maker. These sub-components add the ability for the C5 Collection to learn workloads, classify, predict energy efficient C5 data structures, and make decisions based on the current workload, environment and requirements. The Event Listener and X-Value Translator component acts as a utility component for observing activities, operation execution and states of the CRUD-based C5 Collection component, and translating them into meaningful feature values for the Classifier. The Classifier acts as a virtual power measurement tool inside the green object. It guesses the most-likely energy efficient C5 data structure for an observed sequence of operations, based on prior knowledge gained during training.

Therefore, the output from the Classifier component is a sequence of data structures per instance of the data structure being executed in a program. This sequence is input to the Predictor component to learn (in real time) and predict the next data structure most likely to appear in the sequence. The Decision Maker then uses this prediction to analyze, decide and instruct the CRUD-based Collection when to switch to the new data structure implementation. The main functions of the Decision Maker include: data structure transformation cost analysis and decision making of when to make a switch to a different data structure for energy efficiency. The costs of transformation analyzed by the Decision Maker can include costs of instantiation of a new data structure and the cost of copying existing data over to the new data structure. The details of the Decision Maker component is not included here and left for future work.

In more detail, Figure 3 displays a complete process of the Green CRUD-based C5 Collection, depicting how the green data structure is trained and how it operates at runtime.

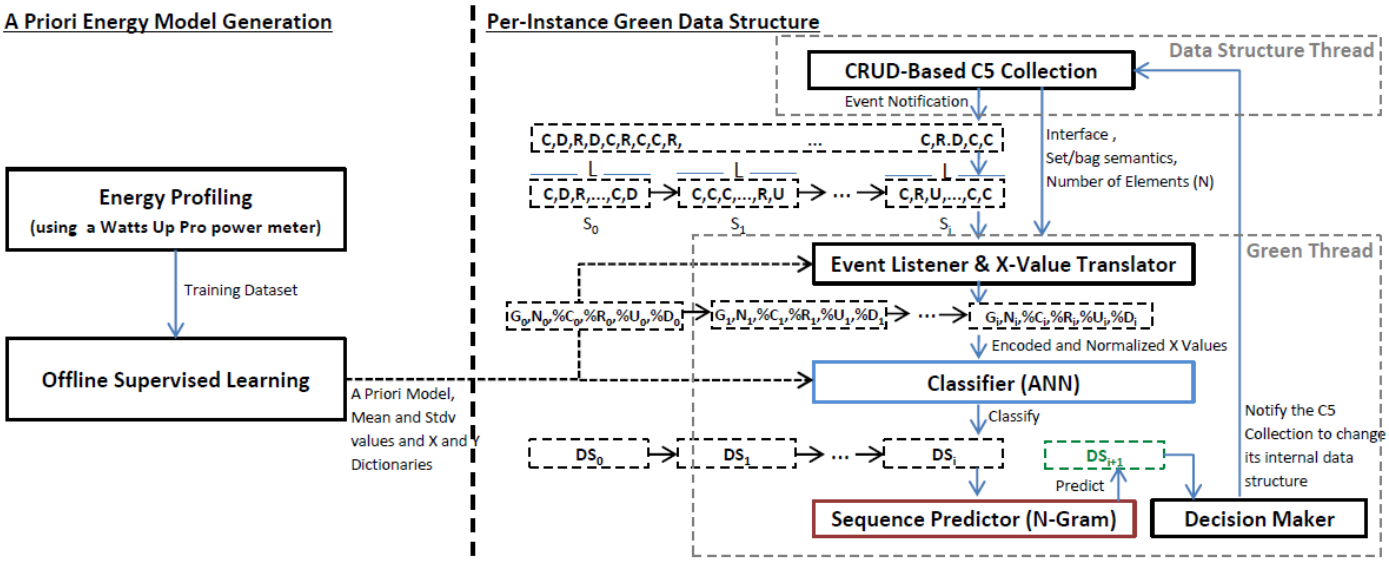

Fig. 3. Adaptive Green CRUD-based C5 Collection Process

The complete process in the figure can be summarized into the following sections and sub-sections:

1) A Priori Energy Model Generation—a one-time process for producing knowledge for the Classifier.
   a) Energy Profiling—for collecting energy data and creating a training dataset for the Classifier and all ground truths for model validations in the experiment.
   b) Offline Supervised Learning—for training and validating the Classifier.
2) Per-Instance Green Data Structure—for tracking how each instance of an adaptive green data structure works at runtime.

The final output from the left half of Figure 3 is a priori knowledge to be embedded in the Classifier. We use an Artificial Neural Network (ANN) as the Classifier. Therefore, this knowledge contains weights and biases for the ANN edges, along with other values for data normalization, encoding and decoding purposes. In the right half, for each instance, there are two threads running asynchronously: the Data Structure thread and the Green thread. The CRUD-based C5 Collection component is instantiated in the Data Structure thread while the Green component is executed in the Green thread. When a CRUD operation is performed on a data structure instance, a notification event, such as *ItemAdded* and *ItemRemoved*, is raised. The Green component listens to these notifications, and observes other data structure states such as the interface and set/bag semantics (collectively the data structure group G), and the current number of elements in the data structure (N). The Green component sees a sequence of CRUD operations as the data structure's workload.

During execution, an instance of the data structure in the program can be alive for a period of time and sometimes indefinitely if a program is always on and running. As a result, the Green component can observe a long sequence of data structure operations. In the figure, this long sequence is divided into subsequences, S_0, S_1, S_2, ..., each of length L. Together with the other observed information (G and N), each subsequence S is then input to the X-Value Translator to be translated into a meaningful feature vector for the Classifier. This feature vector is in the <G,N,%C,%R,%U,%D> format. The ANN-based Classifier maps this feature vector to a data structure DS, and outputs it. Recall that DS is one of the 9 data structures depicted in Figure 1–ArrayList, LinkedList, HashBag, TreeBag, HashedArrayList, HashedLinkedList, SortedArray, HashSet, and TreeSet. Hence, for every subsequence S_i, the Classifier outputs a data structure DS_i, effectively producing a sequence DS_0, DS_1, DS_2,..., of data structures. This sequence is fed to the Predictor in real time, where it is used for online learning and prediction of the next data structure that is likely to appear in the sequence. The predicted data structure is input to the Decision Maker to be used for cost analysis and decision making to instruct the CRUD-based C5 Collection to switch if needed.

Note that we do not directly use the features (G, N and percentage of CRUD operations) to perform prediction of the data structure to use for the next L operations. Instead, we make a best guess for the data structure that would have been the most energy-efficient, and use the guesses as input in the prediction. This method allows us to discard feature vectors once they are processed, and also reduces the dimensionality of the input space for the prediction. Further, training a model that uses a history of feature vectors for prediction is not straightforward, and can easily become prone to issues related to biased sampling in the training set.

B. Energy Profiling

The energy profiling step is crucial to produce a dataset for training the Classifier and create ground truths for model validations. All datasets in this experiment are created and validated using a Watts Up Pro [16], a plug load power meter. The power meter can measure power consumption data at about 1 sample/second. It is a cost effective tool that can produce accurate power consumption data of computer systems. For this study, we have developed an energy profiler to read power data

by sending a command to the device via a USB port. We conduct the profiling on a HP Envy h8 PC (model h8-1520t) system, with a Intel Core i7 CPU@3.4 GHz, 10GB RAM and 64-bit Windows 8.1 Pro. The active power consumption of the base PC ranges from about 35 to 100 watts.

```
L := 10000
start WattsUpPro asynchronously
for each N do
   for each C5DataStructure do
     for each Workload := {%C, %R, %U, %D}  do
       while PowerReadCount < = 5 do
          create an instance of C5DataStructure
          fill instance with N elements
          start Timer and capturing of power data samples
          perform C operation (%C×L)  times
          perform R operation (%R×L)  times
          perform U operation (%U×L)  times
          perform D operation (%D×L)  times
          stop Timer
       end while
       save average power/execution time data to a file
     end for
   end for
end for
stop WattsUpPro
```

Fig. 4. Energy Profiling Algorithm

In the energy profiling process, energy consumption data are collected for creating the datasets of the Classifier. The exact process is detailed in Figure 4. For each of the 9 C5 data structures, the instance is first filled with N elements. Then the operations corresponding to each workload (%C, %R, %U and %D) are performed on the data structure instance. The data structure is profiled for energy consumption while performing the CRUD operations. During the energy profiling operations, a *WattsUpPro* instance runs asynchronously and a power reading event is raised every second. The loops in the pseudo-code are controlled by the *WattsUpPro* instance. The inner loop breaks after the power read count reaches 5 and five power values are collected. We ignore the first read sample to reduce noise due to potential delays of the USB port. The average power consumption data and execution times are measured and recorded to a file. The data are then analyzed to determine the most energy-efficient data structure (called the Y value) for each feature value set explored in the profiling. The Y values, together with the X values (independent variables), become a ground truth dataset.

There are a total of 21 datasets created by this process—one training dataset, one validation dataset, and 19 program validation datasets. The training and validation datasets contain 37,098 and 7,392 observations, respectively. The training and validation datasets uniformly explore the space of possible X values. The 19 program validation datasets are from 10 simulated programs and 9 real-world programs. The 10 simulated programs execute a random CRUD sequence of 400K size. Each CRUD sequence is equally divided into 40 10K-size subsequences. The 10-K size (also in Figure 4, the initial value of L) is selected because it is an appropriate number for our energy profiler to capture power consumption data. Note that the CRUD percentage numbers of these programs may not exist in the training dataset. The number of elements N of each subsequence is calculated based on the prior CRUD operations. Each simulated-program validation dataset contains 40 observations.

The program validation datasets of the 9 real-world programs are created from the actual CRUD operations generated by 3 real-world, open-source C# programs—A* Path Finder [17], Huffman Encoder [18] and Genetic Algorithm [19]. Each real-world program contains at least one .NET data structure. In generating the CRUD sequences of the real-world programs, we replace the original data structures with an enhanced one that can trace and map the add, insert, access, delete and update operations to CRUD operations. The length of the generated CRUD sequences range from 120K to 1.52 million. Like the simulated programs, each such CRUD sequence is divided into 10K-size subsequences and translated into program validation datasets. The last subsequence with length less than 10K operations is ignored.

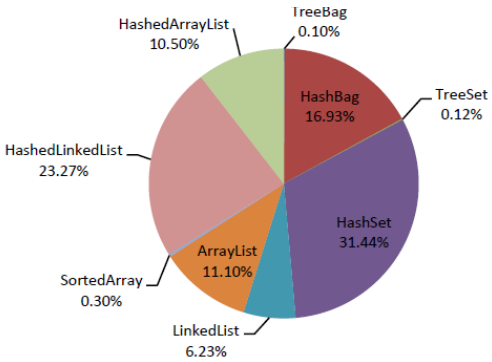

Fig. 5. Distribution of Most Energy-Efficient C5 Data Structures in the Training Dataset

	TreeBag	HashBag	TreeSet	HashSet	LinkedList	ArrayList	SortedArray	HashedLinkedList	HashedArrayList
ICollection (6,183 obs.)	0.18%	3.44%	0.21%	92.80%	0.13%	0.86%	0.13%	0.89%	1.36%
ICollectionBag (6,183 obs.)	0.44%	98.14%			0.15%	1.28%			
ICollectionSet (6,183 obs.)			0.53%	95.83%			0.42%	1.20%	2.02%
IList (6,183 obs.)					0.13%	1.46%	0.19%	68.64%	29.58%
IListBag (6,183 obs.)					37.00%	63.00%			
IListSet (6,183 obs.)							1.04%	68.91%	30.05%
Total (37,098 obs.)	**0.10%**	**16.93%**	**0.12%**	**31.44%**	**6.23%**	**11.10%**	**0.30%**	**23.27%**	**10.50%**

Fig. 6. Distribution and Ranking Table of Most Energy-Efficient C5 Data Structures by Data Structure Group

C. Energy Analysis Results of the Training Dataset and Potential Energy Savings

As a feasibility study, the energy data collected for creating the training dataset are further analyzed to see how the Y values, the most energy-efficient C5 data structures, are distributed in the training dataset and how much energy saving opportunity there is among the C5 data structures. The pie chart in Figure 5 is the analysis result displaying the distribution of the most energy-efficient C5 data structures in the 37,098 observations of the training dataset. As we can see, HashSet is a preferable energy efficient C5 data structure, and covers 31.44% of the training dataset. Overall, all 9 data structures are well represented in the training dataset. For more detail, Figure 6 displays an energy efficiency ranking table of the data structures by group. The higher percentage values indicate the more

preferable energy efficient data structure in each group. For example, in row 5, the IListBag group has two choices of applicable data structures: LinkedList and ArrayList. ArrayList is more preferable and should more likely be selected since it covers 63% of the workloads, while LinkedList only covers 37%. The last row of the ranking table is what is represented in the pie chart in Figure 5. This table can be used as a guide to select an energy efficient data structure within a given data structure group. One main capability of the adaptive data structure that we seek is the automatic selection of the most energy-efficient C5 data structures online, without using this ranking table.

The potential energy savings are calculated using actual energy consumption data (in Joules) of each workload (observation). For each observation in the training dataset, we calculate the energy difference between the most energy-efficient and the least energy-efficient data structures, ignoring the ones in between. The percentage difference tells us how much energy can be saved by selecting the best data structure, compared to the worst one. This analysis (based on the data points in the training dataset) shows that the overall potential energy saving is about 81.33%. By data structure group, the IListBag group produces the smallest energy saving opportunity of about 16.97%. This implies that if our requirement is to select only C5 data structures in the IListBag group (only two choices, ArrayList and LinkedList), the potential energy saving by selecting the right data structure is about 16.97%. In contrast, the largest potential energy saving is revealed by data structures in the ICollection group, at about 97.50%.

IV. Predicting Energy Efficient Data Structures

This section discusses in more detail our implementation of the two predictive components of the adaptive green data structure, the Classifier and the Predictor, along with some prediction accuracy and validation results. The implementations represent possible machine learning based solutions to the two problems highlighted in our adaptive green data structures approach—classification and sequence prediction problems. We have not yet tested the comparative effectiveness of our choices with that of other possible machine learning approaches.

A. Classifier

The problem of identifying the data structure to which a new feature vector maps to is a classification problem. We choose to solve the problem by implementing the Classifier component with an ANN. Our C# implementation of the network is based on a book by James McCaffrey [20]. Our hand-tuned ANN consists of 10 input, 15 hidden, and 9 output nodes. Among the 10 input nodes, there are five nodes for the data structure group, G; one node for N; and the other four for %C, %R, %U and %D. The feature G is encoded into 5 input nodes using the 1-of-(c-1) effects encoding method, where c is the number of data structure groups. The 9 output nodes are derived from encoding the 9 C5 data structures using the dummy encoding method. The 15 hidden nodes are the result of hand tuning that produced the highest accuracy results. The activation functions in this implementation include the hyperbolic tangent function on the input-to-hidden nodes and the SoftMax function for hidden-to-output nodes.

Offline supervised learning is used for training the Classifier. The incremental backpropagation method is the training algorithm in our implementation. The result from the training is an a priori energy model that consists of 309 weights and biases, the knowledge for the classification model. The numbers are hardcoded in the Green C5 Collection to be used at runtime, along with means, standard deviation values, and X and Y dictionaries—the means and standard deviation values are used for X-value data normalization, and the X and Y dictionaries are used for translating encoded numeric values to/from the data structure group and C5 data structures respectively. The other default parameter values are also derived from the hand-tuning process—maximum epochs = 3000, learning rate = 0.02, momentum = 0.01 and stop error = 0.04. The classification method used in the implementation is the multi-layer feed-forward method. More detail and definitions of the terminology can be found in the James' book [20].

To reduce bias and over fitting, we use a 10-fold cross validation method [21] to train the Classifier. The original training dataset is randomly partitioned into ten equal size subsamples. A single subsample, 10% of the training set, is retained as the training validation data, and the remaining 90% is used as the training data. This process is repeated 10 times, with each of the 10 subsamples used exactly once as the training validation data. The average accuracy result from all 10 subsamples is observed to be 95.80%. The most accurate model for classification is selected as the final model to be used in the Classifier. This model is tested with other remaining unseen datasets, the 20 remaining datasets (one validation dataset and 19 program validation datasets). We also test the accuracy with the training data to see how the Classifier performs on the already seen data. The accuracy result on the training data is 96.01%. The accuracy on the validation dataset is 82.40%, and averages 76.52% on the 19 programs (ranges from 59.71% to 99.25% accuracy). The numbers show that the neural network Classifier is adequately accurate.

B. Predictor

The purpose of the Predictor is to add the ability to predict the next most-likely energy efficient data structure in the already seen data structure sequence produced by the Classifier. This is a sequence prediction problem. We solve the problem with a well-known probabilistic language model, the n-gram model [22]. This model is used for predicting the next item in a sequence in the form of a (n-1)-order Markov process. Our C# implementation of the n-gram Predictor is based on the string matching pseudo code found in the book by Ian Millington [22].

Refer to Figure 3, our n-gram Predictor component uses incremental online learning to infer the distribution of data structures conditioned on observed data structure sequences, and make online predictions based on this distribution. The n-gram sequences used for learning come from the Classifier's output. The prediction is continuously made once an n-gram is registered. The output will be "Unknown" if there is not enough knowledge for the prediction, and is considered a misprediction. The final outcome of each program execution is a sequence of predicted most energy-efficient data structures, to be compared

with the ground truths. We validate and test the accuracy of the model with the 19 validation programs.

To validate the n-gram Predictor, both the Classifier and Predictor components are used. CRUD operation sequences of the 19 programs are input to the Classifier to produce data structure sequences for the Predictor to learn and predict. Each program is set to run in loops for multiple iterations simulating that real-world programs/algorithms can be executed repeatedly multiple times. The graphs in Figure 7(a) display prediction accuracy results of a trigram predictor from running 4 of the 19 simulated and real-word programs. The y-axis of the graphs represents the accuracy in percentage and the x-axis shows the first four iterations of the looped execution of a program. The accuracy numbers indicate how many data structures in each predicted sequence match with the corresponding ground truth. The graphs display the accuracy results by data structure group as indicated by the graph legend at the top of the figure. As we can see, the predictions on C5 data structures in ICollection, ICollectionBag and ICollectionSet are more accurate than that of ones in IList, IListBag and IListSet. However, there are some programs, such as the simulated program #2 and Huffman Encoder, for which the trigram Predictor produces very accurate predictions for all groups.

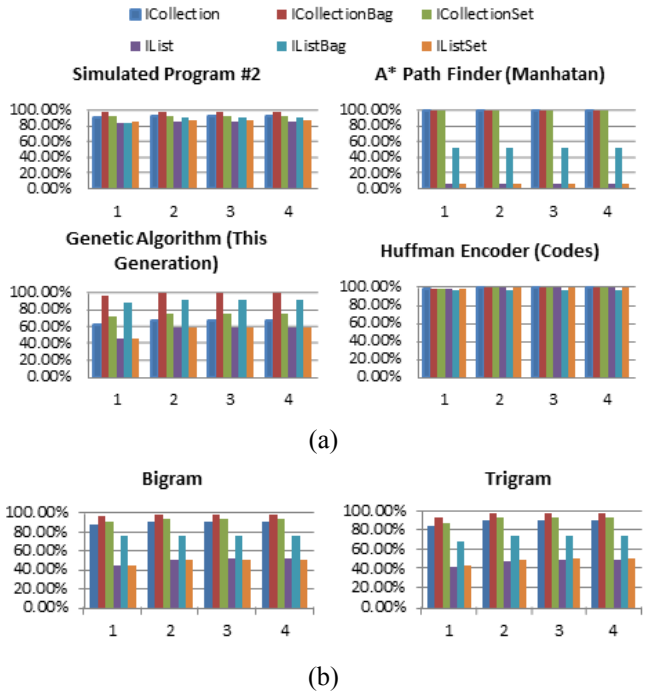

Fig. 7. Prediction Accuracy Results: (a) Accuracy Results by Programs using a Trigram (b) Averaged Accuracy Results of All 19 Programs using Bigram and Trigram

We also notice some low accuracy in the IList and IListSet groups when executing the A* Path Finder programs. After further investigation, we found that the inaccuracy is due to the limited size of the training dataset, and the uniform coverage therein. As a result, the workloads represented in the training dataset do not adequately sample those produced by these particular programs. We validate this by also training the Classifier with two of the five A* Path Finder programs, and then testing the Predictor with the other three remaining programs. The accuracy of the test programs in these data structure groups improves significantly, to more than 90%. Therefore, to improve the accuracy of the Predictor, it is crucial that the Classifier is trained with either more granular data points, or with a data set that contains data points that are representative of the programs expected to execute in the system.

We also conduct some misprediction analysis. One interesting result revealed in the mispredictions of a bigram Predictor is that, even if the prediction is incorrect, the energy savings from the mispredicted data structure is still more than 56%, compared to the worst data structure choice. Therefore, absolute accuracy is not always necessary to gain advantages from the approach, especially when data structure choices are made in an uninformed manner.

Overall prediction accuracy results are displayed in the graphs in Figure 7(b). The graphs display average prediction accuracy of the bigram and trigram predictors when executing all 19 programs. For each interface group, while incremental online learning is in progress, the accuracies of the two predictors are increasing in every iteration of program execution, and start to converge at iteration 4. The values stay unchanged after iteration 4, implying a steady state in the underlying model. Overall, the prediction for data structures in the ICollectionBag group gives the highest accuracy, converging at about 98%. On the other hand, ones in the IList and IListSet groups give the lowest prediction accuracy, primarily due to the poor performance in the A* program. For the top 4 interface groups (ICollection, ICollectionBag, ICollectionSet and IListBag), the average accuracy converges above 70%. Since the performance is not significantly different, a bigram Predictor should be sufficient. A bigram Predictor will predict the next energy efficient data structure by only looking at the current data structure in the sequence.

C. Threats to Validity

There are many other aspects that need to be considered in order to develop a fully functional adaptive green data structure. This section explains several validity threats to the design of our study. First, our study is limited to the C5 dynamic data structures that implement the ICollection interface, and is based only on the selected data structure features. Because the training dataset does not cover all possible workloads, our predictive model has some limitations. First, the predictive model is limited to common operations that map to the CRUD operations. Other operations are ignored. Second, the feature N is also limited to 50K elements and the length of CRUD subsequences is fixed at 10K. The workloads, %C, %R, %U and %D, are also limited at some level of granularity.

The energy profiling is a key process in creating the a priori knowledge for our predictive model. The energy data collection process is done with a power meter that can read power consumption at 1 sample per second. The result can be more accurate if we can measure power consumption data at a finer grain and on different parts of the system, such as CPU, memory, display, etc. Our power measurement is at the system level, and may have overhead and noise. The energy collection process is designed to collect energy data on a fixed setting as described in the algorithm presented in Figure 4.

We do not consider whether the CRUD workloads in the training dataset are feasible. For example, we include a workload with 90% of delete operations and 10% of add operations; such a scenario will never realize.

Our study explored the energy saving opportunities resulting from automated choice of a data structure. One aspect that we did not explore is the performance impact of such a choice. It may be argued that a low energy data structure is probably fast too; however, we do not have conclusive evidence to support it. We seek to explore the performance impact of low energy data structures, and whether it needs to be addressed while making the decision to switch between data structures. Performance overhead of the architecture is also significant and needs to be studied in order to make the green data structure practical. As indicated from our results, the predictive model seems to converge at a steady state. We expect that the cost and overheads from online learning, decision making and switching to be high only at the initial stage. This is also an open question needed to be investigated in our future research.

Finally, the adaptive green data structure is not complete without the Decision Maker module. Even if the Predictor never mispredicts, the process of switching between the data structures may dissolve the potential energy savings. Being able to predict more than just the next data structure could be useful in this decision making. For more accurate models, more studies and models can also be done using other machine learning methods such as SVM, Logistic Regression, etc. These are also left for future research.

V. CONCLUSION

With a vision to see smart and adaptive green objects and components be part of green software programs in the future, we have detailed a concept and an architecture for building adaptive green data structures that can intelligently adapt for energy efficiency. We provide empirical evidence that there exists energy saving opportunities in C5 dynamic data structures, which may be present in other interface-based, object-oriented dynamic data structures as well. Using a "select the right data structure for the right workload" approach, we demonstrate how the C5 data structure selection process can be automated with machine learning tools such as Artificial Neural Networks and n-gram based predictors. The validation results show that the models can accurately classify and predict data structures for energy efficient computing. The work can also be applied in other interface-based objects and is an essential groundwork for building fully functional adaptive green data structures in the future.

REFERENCES

[1] M. P. Mills. *The cloud begins with coal. Big data, big netowrks, big infrastructure,and big power—an overview of the electricity used by the global digital ecosystem*, Digital Power Group, August 2013, http://www.tech-pundit.com/wp-content/uploads/2013/07/Cloud_Begins_With_Coal.pdf?c761ac

[2] GreenPeace, "The iPad, internet, climate change link in the spotlight" , Greenpeace International, 30 March 2010, http://www.greenpeace.org/international/en/news/features/ipad-cloud-climate-change-290310/

[3] J. Michanan, R. Dewri, and M. J. Rutherford. "Understanding the power-performance tradeoff through Pareto analysis of live performance data." In *International Green Computing Conference (IGCC)*, November 2014.

[4] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction." In *ACM/IEEE MICRO*, pp. 81–92, 2003.

[5] K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, and B. Jacob. "The performance and energy consumption of embedded real-time operating systems." In *IEEE Transactions on Computers*, vol.52, no.11, pp. 1454-1469, November 2003.

[6] F. Steimann and P. Mayer. "Patterns of Interface-Based Programming." In *Journal of Object Technology*, vol. 4, no. 5, July-August 2005, pp. 75-94, http://www.jot.fm/issues/issue_2005_07/article1

[7] T. King. *Dynamic data structures: Theory and applicaiton*. Academic Press, 1992.

[8] R. Horvick. *Data Structures Succinctly Part 1*, Technology Resource Portal, Syncfusion Inc., 2012.

[9] N. Kokholm and P. Sestoft. "The C5 Generic Collection Library for C# and CLI." *Technical Report ITU-TR-2006-76*, IT University of Copenhagen, January 2006, https://www.itu.dk/research/c5/latest/ITU-TR-2006-76.pdf

[10] S. Millett and N. Tune. *Patterns, Principles, and Practices of Domain-Driven Design*, John Wiley & Sons, Apr 20, 2015.

[11] J. Eastep, D. Wingate and A. Agarwal. "Smart data structures: an online machine learning appraoch to multicore data structures." In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, pp. 11-20, 2011.

[12] E. G. Daylight, D. Atienza, A. Vandecappelle, F. Catthoor and J. M. Mendias. "Memory-access-aware data structure transformatins for embedded software with dynamic data accesses." In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.12, no.3, pp. 269-280, March 2004.

[13] J. Flinn and M. Satyanarayanan. "Energy-aware adaptation for mobile applications." *ACM SIGOPS Operating Systems Review*, vol. 34, no. 2, pp. 13-14, 2000.

[14] J. Aguilar-Saborit, P. Trancoso, V. Muntes-Mulero and J.L. Larriba-Pey. "Dynamic adaptive data structures for monitoring data streams." *Data & Knowledge Engineering*, ScienceDirect, vol.66, pp. 92-115, March 2008.

[15] J. Ansel. "Autotuning programs with algorithmic choice." *Doctoral Dissertation*, Massachusetts Institute of Technology, 2014.

[16] Watts Up? Meters—Watts Up? Plug Load Meters. https://www.wattsupmeters.com/secure/products.php?pn=0

[17] G. Franco. "A-Star (A*) Implementation in C#", *CodeGuru*, 6 September 2006, http://www.codeguru.com/csharp/csharp/cs_misc/designtechniques/article.php/c12527/AStar-A-Implementation-in-C-Path-Finding-PathFinder.htm

[18] S. Natav. "HuffMan Encoding - From Implementation to Archive." *CodeProject*, 18 May 2009, http://www.codeproject.com/Articles/36415/HuffMan-Encoding-From-Implementation-to-Archive-Pa

[19] B. Lapthorn. "A Simple C# Genetic Algorithm." *CodeProject*, 21 August 2003, http://www.codeproject.com/Articles/3172/A-Simple-C-Genetic-Algorithm

[20] J. McCaffrey. *Neural Networks Using C#*, Technical Resource Portal, Syncfusion Inc., 2014.

[21] R. Kohavi. "A study of cross-validation and bootstrap for accuracy estimation and model selection." In *Proceedings of the 14th international joint conference on Artificial intelligence*, vol. 2, pp. 1137-1143, 1995.

[22] I. Millington. *Artificail Intelligence for Games*. Morgan Kaufmann Publisher, Elsevier Inc., 2006.

[23] J. D. Vega, "Intel Power Gadget." *Intel Developer Zone*, 7 January 2014, https://software.intel.com/en-us/articles/intel-power-gadget-20