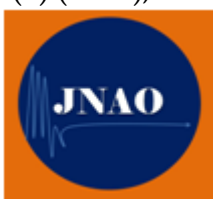
Journal of Nonlinear Analysis and Optimization
Vol. 13(2) (2022), February 2022
https://ph03.tci-thaijjo.org/
ISSN : 1906-9685



J.Nonlinear Anal. Optim

Machine Learning Technique to improve Linux Process Scheduling

DR.G BALAKISHORE, Associate Professor, Department of CSE SHREE INSTITUTE OF TECHNICAL EDUCATION, RENIGUNTA, gbalakishore1993@gmail.com
DR.G SRIKANTH, Associate Professor, Department of CSE DHRUVA INSTITUTE OF ENGINEERING & TECHNOLOGY, HYDERABAD, Srikanth9303@gmail.com.
DR.G THEJA, Associate Professor, Department of CSE DHRUVA INSTITUTE OF ENGINEERING & TECHNOLOGY, HYDERABAD, thejagaddam@gmail.com

.

Abstract—In Linux system, different machine learning tech- niques used to learn the CPU time-Slice utilization behaviour of known program by analysis of certain static and dynamic attributes of processes when learning is done. When they are in running mode our objectives was to minimize a process TaT (turn-around-time) with using the most important static and dynamic attributes of the process that can help best to predict the CPU burst time. To allow scheduling with customized time slices we modify the Linux kernel scheduler. The WEKA tool used to find the most suitable machine learning technique or method to characterize our program. With using different experiment, we find that the C4.5 decision tree algorithm most effectively solved this problem. we find that the turn around time(TaT)reduce in predictive scheduling in the range of 1.4percent to 5.8percent. this is because reduction in the context switches which is used to complete the process execution .our result interesting were operating system presently never make use of program's previous execution history in there scheduling.

Index Terms—Context Switches, Decision Tree Algorithm, Machine Learning, TaT, WEKA.

I. INTRODUCTION

According to scheduling algorithm, process schedulers al- locate CPU time slices to every process that does not use any previous execution history of the process. For better utilization of resources we could recognize a program and predict it's resource requirements. For example, many processes having fixed CPU time is pre empted while having a very small CPU time needed for completion. This increases no. of context switches also called as process switch or task switch [4]. It causes invalidation of pipelines, caches and swapping of buffers and so on. This increases delay between process submission and process completion called as TAT of the pro- gram. So, by recognizing or characterizing program we may understand their previous execution history and predict their resource requirements. In this work, we address problem of using machine learning technique, how to minimize the TAT of program. Using different features of program of characteristics machine learning technique used to predict CPU burst time. To minimize the TAT we use special time slice of STS as the CPU burst time. In section II, we discuss Related work, Proposed System in section III, in section IV, Results and Discussion. In section V, present the conclusion and describe possible future work.

A. Dissertation Idea

In Linux, the CPU scheduler in the Linux kernel has been rewritten multiple times since its conception. This is because Linux has evolved during all these years and has gainedsupport many different kinds of systems and applications. The original goals of Linux were to provide a Unix-like operating system to run on small personal computers. At that time, the applications were very simple and their requirements were not the same as the ones we have nowadays. One of these is quick response for interactive applications, something that has become very important due to Linux being used on desktop machines. Since the kernel has evolved in a lot of areas and now runs on many kinds of different machines from high-end computer clusters to embedded devices, including, as already mentioned, desktop machines. These changes have forced the rewriting of the CPU scheduler multiple times during its existence to adapt to the new requirements.

B. Motivation of Dissertation

In Linux, the CPU scheduler in the Linux kernel has been rewritten multiple times since its conception. This is

because Linux has evolved during all this years and has gained support many different kinds of systems and applications. The original goals of Linux were to provide a Unix-like operating systems to run on small personal computers. At that time, the applications were very simple and their requirement were not the same as the ones we have nowadays.one of these is quick response for interactive applications, something that has become very important due to Linux being used on desktop machines. Since the kernel has evolved in a lot of areas and now runs on many kinds of different machines from high- end computer clusters to embedded devices,including desktop machines. these changes have forced the rewriting of the CPU scheduler multiple times during its existence to adapt new requirements.The latest refactoring went in Linux 2.6.8.1, which introduced a new CPU scheduler with O(1) algorithms. This renewed scheduler can make decisions in a constant finite time, independently of the number of processes running on the machine. In other words, its response is generally deterministic and imposes a small penalty over system performance. So, generally process schedulers allocate CPU time slices to a process according to a scheduling algorithm that does not use any previous execution history of the process. Here we suggest that better resource utilization could be done if we
”recognize” a program and predict its resource requirements. For example, consider a process which exhausted its allocated CPU time and is pre-empted although it needed a small slice of additional CPU time for completion. Such preemption, increases number of context switches (also called as a process switch or task switch). It causes invalidation of caches and

pipelines, swapping of buffers and so on. Thus ultimately this increases TaT of the program. (TaT is the delay between process submission and process completion). Thus we observe that by characterizing or recognizing programs it may be possible to understand their previous execution history and predict their resource requirements.
So, detail objectives of proposed system are as follows,

- The Prime objective is to improve the process scheduling algorithm by seeing past behaviour of processes.
- We will identify the process with their attribute and instances and will find it's ideal values that will be checked by C4.5 Decision Algorithm to minimize their turn-around time.
- We will prove that fusion of process management of the OS and machine learning technique will provide a great optimization in the performance of computer system.

II. RELATED WORK

To remember the previous execution behaviour of certain well known programs, suranauwarat and Taniguchi, presents the approach where they study the process times of these programs in various similarity states.[8] The program flow sequence(PFS) used to extend the CPU time slice of a process from the past history of process PFS is calculated. To decide whether the program executing currently needs additional time, PFS is used. They set a maximum dispatch delay time called Tm which determines the time limit for context switching having multiple of the delay for minimum process switching time. So, either reducing or increasing some scaling features they control the CPU time of a process Tp. Thus, they conclude from experimental results that overall processing time is reduced for known programs. PFS knowledge base used to schedule the process they search and improve it's behaviour.

a) : In the paper by smith [5], they use genetic algorithm to identifying "good template" for a particular workload. Template defines similarity between two applications. The author predicted the application run times using historical information. They present a technique for deriving predictions for the run times of parallel application from the run time of similar application that have executed in the past. To define similarity they use different characteristics of program. Ge- netic algorithm techniques used to determine those application characteristics that yield the best definition of similarity for making predictions. In a machine learning perspective, Genetic algorithm's are expensive in terms of computation[13] and also their results are fragile. Genetic algorithms (GAS) provide a learning method motivated by an analogy to biological evolution. Rather than search from general-to-specific hy- potheses, or from simple-to-complex, GAS generate successor hypotheses by repeatedly mutating and recombining parts of the best currently known hypotheses. At each step a collection of hypotheses called the current population is updated by replacing some fraction of the population by offspring of the fit current hypotheses. The process forms a generate-and- test beam-search of hypotheses, in which variants of the best

current hypotheses are most likely to be considered next. The popularity of GAS is motivated by a number of factors including: Evolution is known to be a successful, robust method for adaptation within biological systems.GAS can search spaces of hypotheses containing complex interacting parts, where the impact of each part on overall hypothesis fitness may be difficult to model. Genetic algorithms are easily parallelized and can take advantage of the decreasing costs of powerful computer hardware. The problem addressed by GAS is to search a space of candidate hypotheses to identify the best hypothesis. In GAS the "best hypothesis" is defined as the one that optimizes a predefined numerical measure for the problem at hand, called the hypothesis Jitness. For example, if the learning task is the problem of approximating an unknown function given training examples of its input and output, then fitness could be defined as the accuracy of the hypothesis over this training data. If the task is to learn a strategy for playing chess, fitness could be defined as the number of games won by the individual when playing against other individuals in the current population. Although different implementations of genetic algorithms vary in their details, they typically share the following structure: The algorithm operates by iteratively updating a pool of hypotheses, called the population. On each iteration, all members of the popu- lation are evaluated according to the fitness function. a new population is then generated by probabilistically selecting the most fit individuals from the current population. Some of these selected individuals are carried forward into the next generation population intact. Others are used as the basis for creating new offspring individuals by applying genetic operations such as crossover and mutation. In the paper of Gibbsons [3] , it uses the statistical Regression Methods used for the prediction. While this method work on numeric data but can't used in nominal data. The application signature model used for predicting application performance on a given set of a grid resources in the paper by Fredrik[2]. In this model, they introduced the notion of application intrinsic behavior to separate the performance effects of the runtime system from the behavior inherent in the application itself. The signature model is used as the basis for performance predictions. So their approach combines the knowledge of application intrinsic behavior with run-time predictions of resources. They also define application intrinsic metrics as metrics that are solely dependent on the application code and problem parameters. Signature Model Achieved application performance reflects a complex interplay of application demands on system re- sources and the response of the resources to those demands. In this section, we outline a new approach to decomposing this interdependence, enabling us to separate of application demands on resources from specification of resource response. We then combine information about application demands and resource capabilities to generate a performance prediction for the application running on a specific system. A metric space is a multidimensional space where each axis represents a single metric (e.g. FLOPS or I/O request size). Consider an applica- tion where we measure N different metrics. These metrics span

out an N-dimensional metric space. Each metric is measured at regular intervals, and the measured values specify a trajectory through the N-dimensional metric space. As the application executes, it traces a trajectory in the metric space. Contract verification consists of comparing the predicted signatures with the signatures based on runtime measurements. Applica- tion Intrinsic Metrics We define application intrinsic metrics as metrics that are solely dependent on the application code and problem parameters. These metrics are independent of the capabilities of the system on which the application is running, such as processor speed or network bandwidth. These metrics express the demands the application places on the resources, or the resource stimuli. Examples of application intrinsic met- rics include number of bytes transferred per communication message and average number of source code statements per floating point operation. As the application executes, it traces a trajectory through the application intrinsic metric space. We call this trajectory the application intrinsic signature. By selecting application intrinsic metrics that capture important resource demands, we can understand the load the application places on the execution environment. The ability of the re- sources to service the load determines the overall performance of the application. Timing issues or data dependencies may cause the application intrinsic signatures to vary between runs. The Tigran [14] author presents the Linux kernel 2.4 internal architecture that used for the implementation of scheduler using Linux kernel. In the paper Garner [7] it used the WEKA tools for identifying the best machine learning technique used for the characterization program. The design of Unix operating system can be represented by the Maurice Bach [1] for the purpose of the system development.

From this brief review, we have following conditions,

- It is possible to predict scheduling behaviour.
- The approach depends on machine learning techniques used to train on previous program execution behaviour.
- For significant prediction, it is important to characterise the program attributes.

III. PROPOSED SYSTEM

The proposed structure mainly focuses on following areas

- Module 1:- User Level and System Level processes.
- Module 2 :- User Space Project Development.
- Module 3:- Kernel Space Project Development.

A.

Module 1:- User Level and System Level Processes The Module 1 consist of A process is a program (object code stored on some media) in the midst of execution. Processes are, however, more than just the executing program code

(often called the text section in Unix).They also include a set of resources such as open files and pending signals, internal kernel data, processor state, a memory address space with one or more memory mappings, one or more threads of execution, and a data section containing global variables. Processes, in effect, are the living result of running program code. The kernel needs to manage all these details efficiently

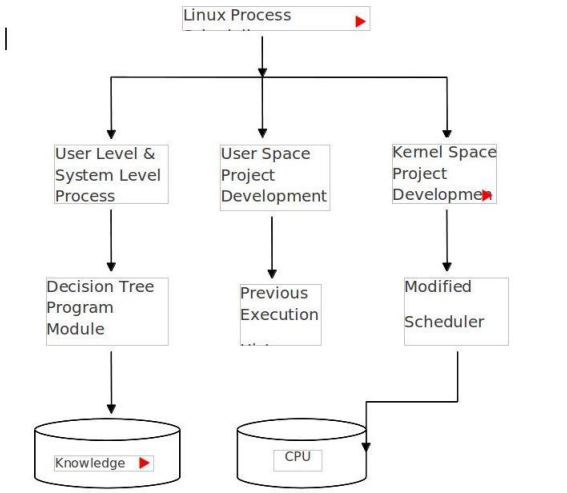


Fig. 1. System Overview

and transparently. Threads of execution, often shortened to threads, are the objects of activity within the process. Each thread includes a unique program counter, process stack, and set of processor registers. The kernel schedules individual threads, not processes. In traditional Unix systems, each pro- cess consists of one thread.

b) : In modern systems, however, multithreaded programs-those that consist of more than one thread- Linux has a unique implementation of threads: It does not differentiate between threads and processes. To Linux, a thread is just a special kind of process. A program itself is not a process; a process is an active program and related resources. Indeed, two or more processes can exist that are executing the same program. In fact, two or more processes can exist that share various resources, such as open files or an address space. A process begins its life when, not surprisingly, it is created. In Linux, this occurs by means of the fork() system call, which creates a new process by duplicating an existing one. The process that calls fork() is the parent, whereas the new process is the child. The parent resumes execution and the child starts execution at the same place: where the call to fork() returns.The fork() system call returns from the kernel twice: once in the parent process and again in the newborn child. Often, immediately after a fork it is desirable to execute a new, different program. The exec() family of function calls creates a new address space and loads a new program into it. In contemporary Linux kernels, fork() is actually implemented via the clone() system call. Finally, a program exits via the exit() system call.This function terminates the process and frees all its resources. A parent process can inquire about the status of a terminated child via the wait4()1 system call, which enables a process to wait for the termination of a specific process. When a process exits, it is placed into a special zombie state that represents terminated processes until the parent calls wait() or waitpid().

B. Module 2:- User Space Project Development

In Module 2, Learning is done by an analysis of certain static and dynamic attributes of the processes while they are being run. we discover the most important static and dynamic attributes of the processes that can help best in prediction of CPU burst times which minimize the process TaT (Turn-around-Time). In high performance environments, a request to execute a program is not serviced immediately but instead serviced only when resources are available. Many grid applications have varying resource requirements and the problem with these applications is that current scheduling systems (as in PBS scheduling for grids)[2] rely on user's estimates of the resource requirements. Scheduling in such systems could benefit by scheduling of resources using the knowledge of previous process execution behaviour. Thus there is a need to characterize the process execution behaviour and resource utilization (like total CPU time). Such knowledge could help to improve the scheduling policy, guide the resource selection and balance the workload distribution. Mainly we use two broad

categories within which we compare programs and find similarities. These are the interactive and non-interactive classes of programs. Since we would like to study the program characterization in general, we include the interactive class of programs also although the practical utility of such programs in batch systems may not be meaningful. In each category, we run the programs with different input Sizes. By using these run traces, we collect data such as system time, user time etc. and prepare a knowledge base. Applying machine- learning techniques on this knowledge base, we extract the most effective parameters that can characterize the process execution behaviour. These isolated parameters were used to predict the resource requirements of similar process. They study the process times of these programs in various states. The knowledge of the program flow sequence (PFS) is used to extend the CPU time slice of a process. PFS of a process is computed from its past execution history. PFS characterizes the process execution behavior and is used to decide whether the program executing currently needs additional time. They set a threshold Tm called as maximum dispatch delay time, which determines the time limit for context switching; and is a multiple of the delay for minimum process switching time. They control the CPU time of a process Tp, by either reducing or increasing a scaling feature. The information for computing the PFS and Tp adjustment thresholds are based on observations from the 4 phases:

- when a process is created.
- when a process uses up its time-slice.
- when a process blocks itself, and
- when a process terminates.

we conclude from experimental results that overall processing time is reduced for known programs. To schedule a process they search for its name in the PFS knowledge base and thus improve its behavior. Then, The program 'X' is given to C4.5 decision tree as an input. The decision tree will classify 'X'

and output the Special Time Slice (STS). Then we send this
STS information to modified scheduler through a system call.

C. Module 3 :- Kernel Space Project Development

In Module 3, we have kernel space project development; in this from user level space we take that special time slice to the modified scheduler. Run the programs with different special time slices with modified O(1) scheduler and find STS (best special time slice) which gives minimum turn- around-time(TaT). We send this STS information to modified scheduler through a system call[10]. The scheduler instructs the CPU such that CPU allocates STS ticks to 'X'. The CPU allocates STS ticks to 'X' and it will run with minimum TaT. The Linux scheduler is defined inkernel sched.c. The scheduler algorithm and supporting code went through a large rewrite early in the 2.5 kernel development series. Consequently, the scheduler code is entirely new and unlike the scheduler in previous kernels. The new scheduler was designed to accom- plish specific goals, implement fully O(1) scheduling. Every algorithm in the new scheduler completes in constant-time, regardless of the number of running processes or any other input. The basic data structure in the scheduler is the runqueue. The runqueue is defined inkernel/sched.c as struct runqueue. The runqueue is the list of runnable processes on a given processor; there is one runqueue per processor. Each runnable process is on exactly one runqueue. The runqueue additionally contains per-processor scheduling information. Consequently, the runqueue is the primary scheduling data structure for each processor. Why kernel/sched.cand not include/linux/sched.h Because it is desired to abstract away the scheduler code and provide only certain interfaces to the rest of the kernel. subsectionMathematical Model When solving problems we have to decide the difficulty level of our problem. There are three types of classes provided for that. These are as follows:

- P Class.
- NP-hard Class.
- NP-Complete Class.

c) P: Informally the class P is the class of decision problems solvable by some algorithm within a number of steps bounded by some fixed polynomial in the length of the input. Turing was not concerned with the efficiency of his machines, but rather his concern was whether they can simulate arbitrary algorithms given sufficient time. However it turns out Turing machines can generally simulate more efficient computer models (for example machines equipped with many tapes or an unbounded random access memory) by at most squaring or cubing the computation time. Thus P is a robust class and has equivalent definitions over a large class of computer models. Here we follow standard practice and define the class P in terms of Turing machines.

d) NP Hard: A problem is NP-hard if solving it in
polynomial time would make it possible to solve all problems in class NP in polynomial time. Some NP-hard problems are also in NP (these are called "NP-complete"), some are not. If you could reduce an NP problem to an NP-hard problem and then solve it in polynomial time, you could solve all NP

problems. Also, there are decision problems in NP-hard but are not NP-complete, such as the infamous halting problem.

e) NP Complete: A decision problem L is NP-complete if it is in the set of NP problems so that any given solution to the decision problem can be verified in polynomial time, and also in the set of NP-hard problems so that any NP problem can be converted into L by a transformation of the inputs in polynomial time.

f) The complexity class NP-complete is the set of problems that are the hardest problems in NP, in the sense that they are the ones most likely not to be in P. If you can find a way to solve an NP-complete problem quickly, then you can use that algorithm to solve all NP problems quickly.

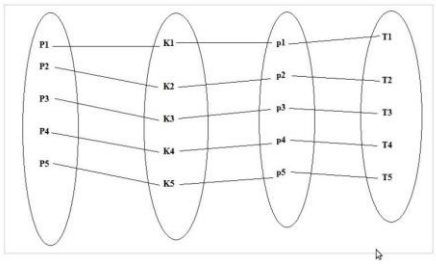


Fig. 2. Venn diagram

D. Dynamic Programming and Serialization

In proposed approach various dynamic programming as- pects are used. Details are as follows:
In user and system level processes, divide and conquer methodology is used for segmentation purpose by considering the same sequence of activity get divided into no of processes, so we divide the processes to characterize the attributes from it. After this it will stored it into the knowledge base data structure. Once all the processes are stored into the knowledge base, when the new process will come then it will first classify with using this structure and then predict the special time slice of it and run this process. If not present the instances of this process in this data structure then it will stored the instances into it.

g) : In the user and kernel space project development, the branch and bound method is used for initializing and restructuring the classifier decision trees.

E. Data independence and Data Flow architecture

Data flow diagram (DFD) is also called as 'Bubble Chart' is a graphical technique, which is used to represent information flow, and transformers those are applied when data moves from input to output. DFD represents system requirements clearly and identify transformers those becomes programs in design. DFD may further partitioned into different levels to show detailed information flow e.g. level0, level1 etc.

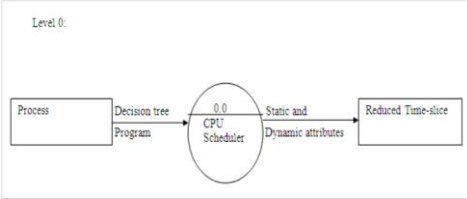


Fig. 3. dfd(level 0)

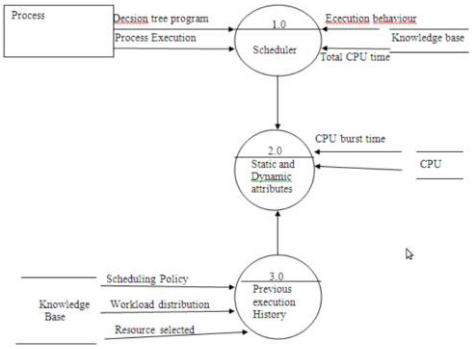


Fig. 4. dfd(level 1)

F. Turing Machine

The large class of applications having following character- istics requires Turing machine:-

- The applications those are driven by the events rather than data.
- The applications those produce control information rather than reports or displays.

IV. RESULTS AND DISCUSSION

The implementation of system consists of implementation of c4.5 decision tree Algorithm. Given the training instances below, use C4.5 and C4.5rules to generate rules as to when to play, and when not to play, a game of golf.

A. Implementation of C4.5 Decision Tree Algorithm

Input - Training Data
Output - Strong Rules

1) Training Data: The column headers - the attribute names - become part of ”golf. names”, the filestem.names file. The subsequent rows - the training instances - are entered into
”golf.data”, the filestem.data file. following evaluation on the training data.
The different verbosity levels:

- golf.names :- Class, attribute, and value names.
- golf.data :- Training data. The resulting decision tree,
- golf.dt, at the default verbosity level.
- golf.dt1, at verbosity level 1.
- golf.dt2, at verbosity level 2

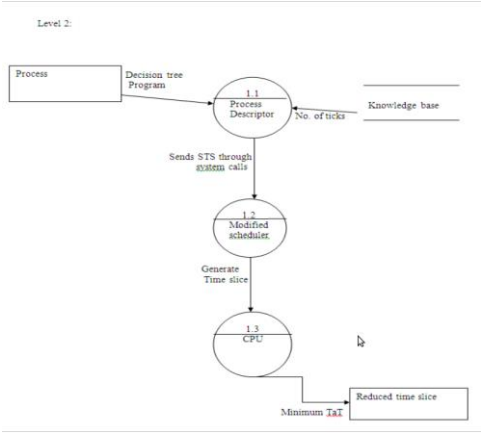


Fig. 5. dfd(level 2)

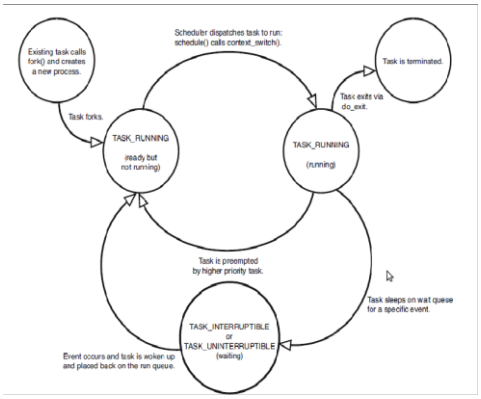


Fig. 6. State Transition Diagram

TABLE I
TRAINING DATA OF GOLF

Outlook	Temperature	Humidity	windy	Play / Dont Play
Sunny	85	85	False	Dont play
Sunny	80	90	True	Dont play
Overcast	83	78	False	play
Rain	70	96	False	play
Rain	68	80	False	play

Rain	65	70	True	Dont play
Overcast	64	65	True	play

- golf.dt3, at verbosity level 3. The resulting

decision rules,

- golf.r, at the default verbosity level.
- golf.r1, at verbosity level 1.
- golf.r2, at verbosity level 2.
- golf.r3, at verbosity level 3.

As you can see, higher verbosity levels provide much more quantitative data than those at the lower levels. Since we are primarily concerned with qualitative results, however, only the generated output at the default verbosity level.

B. Testing

Testing is an integral part of any system or project. The various objectives of Testing:

- To uncover the errors in function logic or implementation for the software.
- To verify that software needs the specific requirement.
- To verify that software has been implemented according to the predefined standard.

Here, I have performed module level testing, checking for each input to be tested. In computer programming, unit testing is a procedure used to validate that individual units of the source code are working properly. A Unit is the smallest testable part of an application. In procedural programming, a unit may be an individual program, function, procedure etc. While in Object Oriented Programming, the smallest unit is a method, which may belong to a base or super class, abstract class or derived/child class. Ideally, each test case is independent from others; mock or fake objects as well as test harness can be used to assist testing a module in isolation. Unit testing is typically done by developers and not by software testers or end users. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a script, return contract that the piece of code must satisfy. As a result, affords several benefits. In continuous unit testing environment, through the inherent practice of sustained maintenance, unit tests will continue to accurately reflect the intended use of the executable and code in the phase of any change. Depending upon established development practices and unit test coverage, up-to-the-second accuracy can be maintained. Unit testing helps to eliminate uncertainty in the units themselves and can be used in a bottom up testing style approach. By testing the parts of a program first and then testing the sum of its paths, integration testing becomes much easier.

C. Test Cases

The different test cases generated by C4.5 at the default verbosity level are interpreted as follows:

i) Firstly, the header. It indicates:

- The name of the file stem being used.(e.g., ”golf”)
- The total number of training instances, or cases, read in the filestem.data file by C4.5.
- The number of attributes per instance.

j) Secondly, one or more ASCII renditions of a generated decision tree:

- The tree consists of a spine of attribute-values that stem from a root attribute test.

In this example, the root is the attribute test ”outlook”. It has three attribute-values: ”sunny”, ”overcast”, and ”rain”.
Two subtrees occur: a ”humidity” subtree below ”sunny”, and a ”windy” subtree below ”rain”.

- The number in brackets following each leaf equals the number of training instances, out of the total number of cases

presented in the header, which belong to that path in the tree.

- This number may be followed by a second number (e.g., 4.0/2.0), in which case the second value (2.0) equals the number of classification errors encountered out of the total number of classifications made from the training data in that particular path of the decision tree (4.0).

The sum of the first series of numbers equals the total number of cases read by C4.5 from the golf.data file.
(e.g., 4.0 + 2.0 + 3.0 + 2.0 + 3.0 = 14.0)
The sum of the second series of numbers equals the total number of errors.
(e.g., 0 for this example).
Two binary files are created during execution: filestem.unpruned: the unpruned decision tree generated and used by C4.5. filestem.tree: the pruned decision tree generated and used by C4.5 which is subsequently required by C4.5rules to generate rules.
Thirdly, the unpruned decision tree and the pruned decision tree are evaluated against the training data instances to test the fitness of each.
The first table illustrates the fitness of the unpruned tree. It has two columns:

- Size: the size of the unpruned tree. That is, the number of nodes of which it is composed.
- Errors: the number of classification errors and their corresponding error percentage from the total number of cases.

The second table illustrates the fitness of the pruned tree. It has three columns:

- Size: the size of the pruned tree. It is either less than or equal to that of the unpruned tree depending upon the extent of the pruning performed by C4.5.
- Errors: the number of classification errors and their corresponding actual error percentage after pruning.
- Estimate: the estimated error percentage of the tree after pruning, useful when comparing with the actual percent- age.

D. Result Analysis

The output generated by C4.5rules at the default verbosity level is interpreted as follows:
Firstly, the header.
1. Same as that of C4.5 Secondly, the set of generated rules.
1. One set of rules is generated for each pruned decision tree.
2. The set of rules usually consists of at least one default rule, which is used to classify unseen instances when no other rule applies.(e.g., Play).
3. Every enumerated rule is composed of attribute-values and a resulting classification, followed by a percentage which represents the accuracy of that rule.
(e.g., Rule 1: if ”outlook = sunny” but ”humidity ¿ 75” then
”Don’t Play”; according to C4.5rules, this rule is accurate 63

TABLE II
TRAINING DATA OF GOLF

(a)	(b)	classified as
9		(a) class play
	5	(b) class Dont play

of the time, and thus has a 37 error margin)
Thirdly, the rules are evaluated against the training data instances to test the fitness of each.
The rule table has six columns:
1.Rule: the number assigned by C4.5rules to each rule.
2.Size: the size of the rule. That is, the number of antecedents of which it is composed.
3. Error: the error margin of the rule.
4. Used: the number of times the rule was used, regardless of correctness, in classifying the training instances. The sum of this column yields the total number of cases.
5. Wrong: the number of times a rule has been misused, and the corresponding percentage of this value from the previous column.

6. Advantage: the difference between the number of times a rule has been used correctly and the number of times it has been used incorrectly.
Fourthly, C4.5rules sums up the number of correct and incorrect classifications in a table. Both the rows and columns have the same headers, but there is a distinction between them:
The rows of the table are the classes available for use in the classification process.
The columns of the table are the classes chosen during classification.
The cell where a particular row and column intersect may either contain a number or not.
1. If the cell does not contain a number, then no tested instances under that cell's row class have been classified by its corresponding column class.
2. Otherwise, the number represents the number of instances of the row class which have been classified as a member of the corresponding column class.
3.Misclassifications occur when the row and column classes of a cell do not match.
For example, in the table below:
9 instances of the known class "Play" were correctly classified using the generated rules as members of class "Play".
5 instances of the known class "Don't Play" were correctly classified using the generated rules as members of class "Don't Play".
0 instances were incorrectly classified.
9 + 5 = 14 = the total number of instances tested.

k) Summary:

Rule 1 suggests that if "outlook = sunny"
and"humidity greater than 75" then "Don't Play".

Rule 2 suggests that if "outlook = overcast" then "Play".

Rule 3 suggests that if "outlook = rain" and "windy = true" then "Don't Play".

Rule 4 suggests that if "outlook = rain" and "windy = false"
then"Play".Otherwise,"Play" is the default class.

REFERENCES

[1] Maurice Bach, The design of the Unix operating system, Pearson Edu- cation Asia, pp: 159-170, 2002.

[2] Fredrik Vraalsen,.Performance Contracts: Predicting and monitoring grid application behavior. In Proceedings of the 2nd International Workshop on Grid computing, November, 2001.

[3]Richard Gibbons, A Historical Application Profiler for Use by Parallel Schedulers, Lecture Notes on Computer Science, Volume : 1297, pp: 58-75,1997.

[4]Hyok-Sung Choi and Hee-Chul Yun, Context Switching and IPC Performance Comparison between uClinux and Linux on the ARM9 based Processor,Linux in embedded applications, Jan., 2005. www.linuxdevices.com/articles/AT2598317046.html.

[5]Warren Smith, Valerie Taylor, Ian Foster, .Predicting Application Run- Times Using Historical Information., Job Scheduling Strategies for Par- allel Processing, IPPS/SPDP'98 Workshop, March, 1998.

[6] Kishore Kumar. P and Atul Negi, Characterizing Process Execution Behavior Using Machine Learning Techniques, In DpROM WorkShop- Proceedings, HiPC 2004 International Conference, December, 2004.

[7]Garner, S .R.WEKA: The Waikato Environment for Knowledge Analysis In Proc. of the NewZealand Computer Science Research Students Conference, pp : 57-64, 1995..

[8]Surkanya Suranauwarat, Hide Taniguchi, The Design, Implementation and Initial Evaluation of An Advanced Knowledge - based Process Sched- uler . ACM SIGOPS Operating Systems Review, volume: 35, pp: 61-81,October, 2001.

[9] Mark A. Hall, Co-rrelation based feature selection for Machine Learning, Master Thesis, Department of Computer Science, University of Waikato, pp: 7-9, 12-14, April, 1999.

[10] Andrew Marks, A Dynamically Adaptive CPU Scheduler, Master Thesis, department of Computer Science, Santa Clara University, pp :5- 9, June, 2003.

[11] Robert Love, Linux Kernel Development, 1sted, the Pearson Education, 2004

[12] Danie P. Bovet, Marc, Understanding the Linux Kernel, 2nded, O' Reilly and Associates, Dec., 2002.

[13] Tom Mitchell, Machine Learning, 1st ed, pp: 52-75, 154-183, 230-244,The Mc-Graw Hill Company.Inc.International Edition,1997.

[14] Aivazian, Tigran, Linux Kernel 2.4 Internals, The Linux Documentation Project, August, 2002.

[15] Luis Paulo Santos, Albert Proenca "Scheduling Under Conditions Of Uncertainty: A Baysian Approach.

[16] Aryabrata Basu, Shelby Funk, "An Optimal Scheme for Multiprocessor Task Scheduling: A Machine Learning Approach.