Scheduling Model for Symmetric Multiprocessing Architecture Based on Process Behavior

Ali Mousa Alrahahleh[1], Hussein H. Owaied[2]

**[1] Department of Computer Science, Middle East University
Amman, Jordan**

**[2] Department of Computer Science, Middle East University
Amman, Jordan**

***Abstract*—**this paper presents a new method for scheduling of symmetric multiprocessing (SMP) architecture based on process behavior. The method takes advantage of process behavior, which includes system calls to create groups of similar processes using machine-learning techniques like clustering or classification, and then makes process distribution decisions based on classification or clustering groups. The new method is divided into three stages: the first phase is collecting data about process and defining subset of data is to be used in further processing. The second phase is using data collected in classification or clustering to create classification/clustering models by applying common techniques similar to those used in machine learning, such as a decision tree for classification or EM for clustering. System training classification should be done in this phase, and after that, classification or clustering models should be applied on a running system to find out in which group each process belongs. The third phase is using process groups as a parameter of scheduling on SMP (sympatric Multi Processor) systems when doing distribution over multi-processor cores. Another advantage can be achieved by letting the end user train the system to classify a specific type of process and assign it to a specific process core, targeting real-time response or performance gain. The new method increases process performance and decreases response time based on different kinds of distribution.

***Keywords*:** Scheduling Algorithm, Symmetric Multi Processors Architecture, Processor Behavior, Processor Behavior Classification.

I. Introduction

Computer systems play an important role in modern life: from work to entertainment, they make things easier, faster, and more enjoyable. Computer System consists of system software and hardware. While hardware serves an important role in a computer system, but the system cannot do any work without the system software and specially the operating system.

One of the main goals of operating systems is to provide a fair share of resources among different types of programs running on the same machine. One of the main resources which operating systems try to allocate in an efficient way is the time of Central Processing Unit (CPU). The operating system tries to schedule programs and allocate CPU based on different scheduling algorithms; some of these algorithms are priority-based algorithms, where the highest priority program will run before the lowest priority program for a definite period of time. There is a different mechanism to avoid starvation like decreasing the priority of program that uses the allocated time.

Priority algorithms have many advantages: it is easy to change the process priority by the user or by the operating system itself [1]. While priority scheduling is very successful on a single-CPU system, it achieves the same result on a multiple-CPU system with a little modification to the old scheduling algorithm. Modern operating systems now support a multi-CPU system, including Symmetric Multiprocessor (SMP) [2].

Symmetric Multiprocessor (SMP) architecture is dominating home PCs and servers. It offers high performance with minimal cost, because it uses a single shared memory. When SMP architecture applies to cores, each one is treated as a separate processor: they are connected together using buses and crossbar switches, switches that connect multiple inputs to multiple outputs in a matrix manner. Examples of systems that use SMP architecture are as follows: Intel’s Xeon, Pentium D, Core Duo, Core 2 Duo, AMD’s Athlon64 X2, Quad FX or Opteron 200 and 2000 series, Sun Microsystems UltraSPARC, Fujitsu SPARC64 III SGI MIPS, Intel Itanium, Hewlett Packard PA-RISC, Hewlett-Packard DEC Alpha, IBM POWER, and Apple Computer PowerPC [3].

Modern Operating provides support of SMP Architecture: Linux, for example, starts from kernel 2.6 and treats each core as a separate processor, maintaining a running queue for each one. Once processed, an instance of a computer program in execution is created and it is added to one of these queues while waiting to be executed. Kernel will choose the next task to run from the queue based on the priority attached to each process [4].

Preemptive Priority Scheduling is used on Windows [5], Linux [6], Solaris [7], and BSD [8] Operating systems. It classifies each process based on priority; the scheduler chooses the process with the highest priority to run next after another process is finished, blocked, waiting for system call,

IJCSI
www.IJCSI.org

or preempted by the scheduler because it takes all time quantum.

There are different types of priority scheduling. One of these types, which are used by Solaris, is a multi-level feedback queue [9]. Each queue contains processes with the same priority; if there are processes that are waiting in the high priority queue, the scheduler chooses and runs these processes. If not, the scheduler checks each queue from the highest-priority queue to the lowest one for processes to run. If one of these queues contains multiple processes, the multi-level scheduler runs each one of them in round-robin manner, providing fair sharing for processes that share the same priority level.

One of notable issues solved by priority scheduling is starvation, and the priority scheduler avoids starvation by degrading the priority of processes that take too much time executing on the same processer. Most modern operating systems distribute processes on different cores based on the load assigned for each instance/core [10]. It applies the same uni-processer scheduling algorithm on each core. While this may provide good results in some cases, it may be better to group processes that share some characteristics or behaviors and assign them to one of these cores.

II. PROPOSED METHODOLOGY

Classification of processes can be done based on the memory requested, the type of I/O operating performed (which can be tracked through a requested system call), CPU usage, and many other criteria by monitoring real-life processes such as database, web browser, word document, and other applications used on desktop or servers; one can provide grouping for each criterion. If a specific combination of these groups is assigned to one processer, it should provide highly-optimized scheduling and efficient use of each core while maintaining load balancing when there are no processes that match the criteria used for classification.

Because the solution space is very large, one can use genetic algorithms to achieve the best number of combinations (solutions), and then apply one of them. There is a lack of policies or rules in open source operating systems FreeBSD, Linux, and Solaris which are used to classify processes dynamically-based on their behavior, therefore the methodology is based on:

- The identification of processes' behavior
- The classification of the processes' behavior
- The distribution of the processes' behavior to the associated processors

The proposed methodology for the system presented as seen in Fig.1 can be sub-divided into four main sections, collecting information related to the subject of research. This information includes any information that assists in classification and making decisions: the second phase is pre-processing collected information to make it usable and clean it from any noise, classification or clustering comes in the third place to identify any possible patterns between data, and the final phase is experimenting and comparing results of previous phases to find out if the goals of research can be achieved. This methodology achieves a good distribution of processes based on their behaviors. Achieving a smart operating system which can make a high level decision without much help from the end user involves the same process in any machine learning solution with little modification. These phases will be explained in next section in detail, the following sections describe proposed work phases.

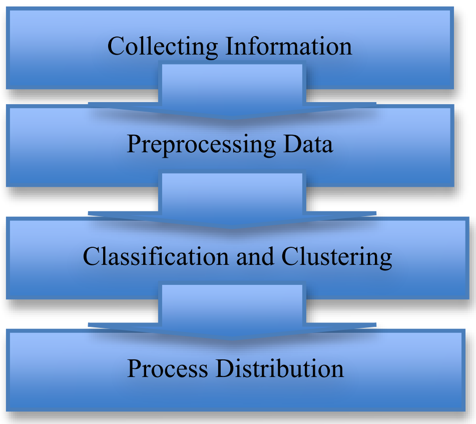

Fig. 1 Proposed the Steps for Scheduling on SMP Systems

A. *Collecting Information*

There are many tools for collecting information about processes in operating system such as Ktrace, Strace, Dtrace. While every one of these tools has advantages and disadvantages, each one is known to be working with a set of operating systems. For example, Strace only works for Linux operating system [11] and Ktrace only works on FreeBSD and early Mac OS [12].

Strace is a Linux tool used to collect information about processes in general; Strace should be attached to processes before the process starts. It provides a run time list of system calls occurred during process operation or execution. Strace affects the performance of running process, and has hard-to-process output if compared to other monitoring tools. Strace output includes the system call name and its parameter; output is similar to normal function call in C programming language [13].

While Strace provides useful information, it doesn't meet the requirements of this research because of the following disadvantages:

1. It is only supported under Linux, making any programs that have Strace dependent and not portable to other operating systems.
2. It doesn't provide formatting options, making output processing a long and hard task, unlike other tools, which provide a good support for data processing like aggregation.

3. It must be attached to a process before it starts. It is hard to attach every program to Strace and if that is possible, it may take a long time to change all the current scripts or executables to append Strace to the environment of execution.
4. It affects program performance, making it unusable for Benchmarking [14], and it may affect the result of research.
5. It generates a single output for each process, introducing a lot of files, and decreasing system performance by flooding desk queues.

Ktrace is another system tool used to collect information about processes; it is only supported under FreeBSD and Mac OS until version 10.5. Unlike Strace, it is much faster and it produces its output in a special format, which is not readable by humans, and the output needs another tool called Kdump to convert it to a readable format [15].

While Ktrace is lighter and faster than Strace, it still has the following disadvantages:

1. It should be attached on each individual process.
2. It produces a lot of files, which means a lot of processing work.
3. Aggregation is not supported, which is useful for large data analysis.
4. It is only supported by FreeBSD and early versions of Mac OS X.

To cover these issues, a framework called Dtrace has been developed by Solaris OS development team and has been ported to other operating systems such as FreeBSD, Mac OS X, and NetBSD. This framework provides comprehensive dynamic tracing for processes on the running system. Besides tracking processes, it provides a complete overview of running system internals such as scheduling activity, file systems, and network usage.

Dtrace is an open source project, which makes it easier to be adapted and integrated with any operating system. Dtrace framework has very good analysis support by providing augmentation and aggregation functions such as summations, averages, and counts; these functions are very helpful for collecting data and tracking processes over a period of time. It also has good support for timed tracking (start and end script based on user configuration). Another important advantage of Dtrace is that a user of Dtrace has complete control over the script output, which means very little post-processing [16].

When combining augmentation functions with output formatting functions, useful statistics can be gathered and analyzed. It provides a complete overview of the system, as well as help in finding bottlenecks. For example, a Dtrace script of one line: “io:::start { printf("%d %s %d",pid,execname,args[0]->b_bcount); }” can print disk transaction live with transaction size and the requester process name. Another sample of Dtrace script: “syscall::open*:entry { printf("%s %s",execname,copyinstr(arg0)); }” lists all opened files by running programs.

Because of all the previously-listed advantages, Dtrace is a useful tool to do most of the research related to operating system performance. It can do all the work starting from data collection and ending with most pre-processing work, while being light on the host system.

Dtrace works using a concept called probes, in which is a simple function call is distributed over a program in special places where users are expected to track. To make Dtrace work on a host operating system, there should be a special function call on each monitored point. Dtrace can also work for user programs, not only system program; these functions are light on the running system and don’t seem to have a heavy affect on system performance.

Dtrace also provides virtualization of data, like data quantization, which make data readable by humans and easy to analyze, as seen in Fig. 2.

```
lint
value  ------------- Distribution ------------- count
65536 |@@@@@@@@@@@@@@@@@@@@@                     74
131072 |@@@@@@@@@@@@@@@@@                   59
262144 |@@@                          14

acomp
value  ------------- Distribution ------------- count
8192 |@@@@@@@@@@@@@                  840
16384 |@@@@@@@@@@@@@                 750
131072 |@@@@@@@               446
524288 |                 0
```

Fig. 2: A Quantization Output Shows System Call Time on Running System

Because of the previously listed advantages, Dtrace have been used in this research as a main source of data. Dtrace has been used for: -

1. Timely and accurate data collection, which reduces the collected sample size and provides periodical data samples.
2. Collect system calls statistical data, which is one of the main targets of this research. System calls inherit its importance from being the only way to request operation or dates from the hardware or operating system. It gives information about resource consumption, I/O operation, and network usage. System calls may vary from one operating system to

IJCSI
www.IJCSI.org

another, which means sample data collected for one operating system is not necessarily the same as other operating systems, making the training data for one operating system only suitable to that system or a similar system.

Only the following data is going to be collected about system call:

1- System call name
2- Number of calls
3- Caller name

Using a small set of data makes the data collected minimal and easy to parse, which can provide fast decision-making. This data is high-level enough to make training data not over fit (only suitable for decision-making on the same training set or similar data).

The Dtrace script handles the preprocessing of data, placing each piece of collected information in one column.

Dtrace has been used outside the machine learning process to find out if there is any easy visually-identified pattern or any enhancement that can be made to Dtrace script to make data more accurate by increasing or decreasing the scope of collected data.

B. *Data Preprocessing*

Most preprocessing is handled by Dtrace, but data should be converted to a special file format called ARFF which classifiers can parse. This format imposes a special naming convention for each column; any volition of these rules may result in process failure, as this is beyond the capability of Dtrace and any addition to fixing these issues may affect the generic modular design of the final program.

ARFF is a special file format developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato. It is used to describe elements that share common definitions or attributes. ARFF files has two sections: one for meta data like attribute names, types of attributes, names of relation (as seen in Fig. 3), and another one for data – data should conform to the description in meta section.

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall
(MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris
@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-
virginica}
```

Fig. 3: Sample ARFF Header

ARFF is useful for representing different types of data like numeric, string, enumeration, and special data format (Example: date); this makes it fit to represent data collected in different contexts and is used as an input for machine learning processes [17]. ARFF data sections contain comma-separated data; each column is mapped to a definition found in the header as seen in Fig. 4.

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

Fig. 4: Sample ARFF Data

There are many text processing libraries and tools for text processing: one of these tools is AWK. AWK is a special utility developed specially for text processing/extraction; a simple AWK script can extract useful information from text files or standard inputs and export information in other formats. This may take a lot of time if done in a traditional complied programming language like C or C++. AWK processes a text file by sequentially finding patterns defined in AWK script files and triggers corresponding actions if there are any. An action can be a mathematical notion like addition, subtraction, etc.; or it can be a normal text substitution where the results are printed directly to a screen unless redirected to a normal file [18].

AWK script can take a raw data produced by a program like Dtrace and transform it to a valid ARFF file. AWK can be found on most of Unix and Unix-like operating systems, such as Linux, FreeBSD, OpenBSD, NetBSD, Mac OS X, and Solaris, making AWK scrip portable over a wide variety of operating systems.

While AWK script provides a simple and fast way to process data, it can be replaced in the future with flex. Flex is a program which takes a pattern/action definition file similar to the one used in AWK script and generates self-contained C language files that can be compiled to a native executable capable of processing input and producing the desired output [19].

C. *Classifications and Clustering*

There are two regularly-used methods for extracting knowledge from data: classification and clustering. Both of them try to find a pattern in data and group data according to these patterns, and each one of them provides different results when applied and has a different set of algorithms; only the algorithm related to the research is going to be discussed here.

Classification tries to assign a set of elements to correct groups based on training data. Training data is a sample of data collected from same environment where the machine learning is going to be conducted and has correct groups or predicted value assigned to each element. The classifier tries to build a model based on training data which will be used in further classification.

Classification is a supervised learning technique, where there is a clear vision of groups and elements that belong to them, and which provides great control over the classification output such as a group's number and group element's specification, but it prevents identifying any new unrevealed pattern in data.

The decision tree is one of the used techniques in classification: it creates a hierarchical tree model or graph which consists of nodes and edges. Each node contains a rule used in branching: where there is any entry in question to other nodes, the leaf of edges is determines the final decision. A sample decision tree looks like figure 3-9 [20][21].

J48 is one of the decision tree algorithms that are fully implemented in WEKA. It builds a model based on attributes of training data that helps in predicting values. J48 algorithm has good accuracy and it is fast enough to be used in critical operating. J48 will be used in this research to classify processes based on system call types and counts [21].

Clustering is the process of identifying similar elements, which can be grouped together. Clustering is an unsupervised learning technique where the number of clusters and their location is unknown before conduction the clustering process, which means less control on the output of clustering. However, it provides a ready-to-use solution when there is a need to identify patterns in data while these patterns are not visible and there is not enough information to build a classification model for the classifier [22].

Clustering has many algorithms; one of the known algorithms is simple EM clustering and Weka has a good support for EM clustering. It will be used in this research as a replacement for classification when there are no predefined groups.

D. Process Distribution

After building the classification/clustering model, which can be used to predict new elements groups, these groups will assist the scheduler in decision-making.

Different kinds of distribution based on user preferences or system usage can be made: for example, if the user needs real-time processes to run on a separate processor for maximum response time, a system can be trained to identify these processes and allocate them to a separate processor. System usage can affect the process distribution: for example, a system can start at 50-50% for two process groups, and if the load or response time is increased, the operating system can make adjustments to the distribution to stabilize the system again.

Bounding processes or a group of processes to one processor can be done using a user land command like "cpuset" in FreeBSD and Linux. This tool helps in setting process affinity to one or more processors.

In the future, the entire scheduling process should be integrated with the kernel because other parameters can incorporate into scheduling process. These parameters include system load, waiting time, and utilization, which is not accessible in user land.

E. Benchmarking and Measuring Results

Benchmarking is "to take a measurement against a reference point." [23]. This helps identify any possible improvement in software and hardware.

Measuring utilization and response time can be indirectly represented by benchmarking processes throw by comparing benchmarking before and after applying the new scheduling technique. Different kinds of open source benchmarking tools can be found, UNIX bench is an example of advance benchmark tools. UNIX bench provides a comprehensive set of benchmarks engineered too quickly and accurately measure processor and operating system performance. Designed to make benchmarks easy to run and easy to understand, UNIX bench takes the guesswork out of producing robust and reliable benchmark results. UNIX bench has many advantages over other tools, these are:

1. UNIX bench benchmark is multi-core aware.
2. UNIX bench is cross-platform running on FreeBSD, Linux, and Mac.
3. UNIX bench returns overall scores compiled from all tests.
4. UNIX bench tries to test different kinds of algorithms on operating systems and reports back the result as a score; these scores can be compared to find out if there are any improvements that have been made after doing modifications to scheduling in an operating system.
5. UNIX bench score is result of comparisons between systems in question and SPARCstation 20-61 performance. For example, if the score is 10, then the machine in question is 10 times faster than SPARCstation 20-61. This helps in comparison and makes the result of benchmark more readable.

UNIX bench does the following test by defaults as listed on the UnixBench website [24]:

- Dhrystone is a test developed by Reinhold Weicker in 1984. This benchmark is a good indicator of the computer's performance. It focuses on string manipulation, with zero floating-point operations. Hardware and software design, compiler and linker options, code optimization, cache memory, wait states, and integer data types have a strong influence on Dhrystone.
- Whetstone: This test focuses on speed and efficiency of floating-point operations. This test does a set of operations typically performed in scientific applications which include a wide variety of C functions like sin, cosine, square root, exp,

and log. They are used as well as integer and floating-point math operations, array accesses, conditional branches, and procedure calls. This test measures both integers and floating-point arithmetic.

- Execl Throughput: This test measures the number of execl calls that can be performed per second. Execl is function call to replace the current process image with a new process image. It is only a front-end for the function "execve ()".
- File Copy: This measures the rate at which data can be transferred from one file to another, using various buffer sizes. The file read, write, and copy tests capture the number of characters that can be written, read, and copied in a specified time (the default is 10 seconds).
- Pipe Throughput: A pipe is the simplest form of communication between processes. Pipe throughput calculates the number of times a process can write 512 bytes to a pipe and read them back per second. The pipe throughput test has no real counterpart in real-world programming.
- Pipe-Based Context Switching: This test measures the number of times two processes can exchange an increasing integer through a pipe. The pipe-based context-switching test is more like a real-world application. The test program spawns a child process with which it carries on a bi-directional pipe conversation.
- Process Creation: This test measures the number of times a process can fork and reap a child that immediately exits. Process creation refers to actually creating process control blocks and memory allocations for new processes, so this applies directly to memory bandwidth. Typically, this benchmark would be used to compare various implementations of operating system process creation calls.
- Shell Scripts: The Shell Scripts Test measures the number of times per minute a process can start and reap a set of one, two, four, and eight concurrent copies of shell scripts where the shell script applies a series of transformation to a data file.
- System Call Overhead: This estimates the cost of entering and leaving the operating system kernel, i.e. the overhead for performing a system call. It consists of a simple program repeatedly calling the getpid (which returns the process id of the calling process) system call. The time to execute such calls is used to estimate the cost of entering and exiting the kernel.

GTKPerf is another benchmarking tool to test desktop widgets; this tool tries to create different kinds of controller-like scroll bars, tabs, selection menus, and drawing, and it helps in testing UI (interactive programs). Comparing results should give an indirect overview of interactive operating system response time.

III. RESULT

Benchmark provides a measurement of improvement after applying the new scheduling policy; in these sections, two types of benchmark are going to be conducted. Unix Bench is used to measure a wide variety of performance metrics and GTKPerf is used to measure interactive application performance.

A. Benchmark for string and mathematical operation

This benchmark focuses on string and mathematical manipulation as seen in Fig. 5; it emulates operations done in compiler and linkers, and it also measures the efficiency of cache memory.

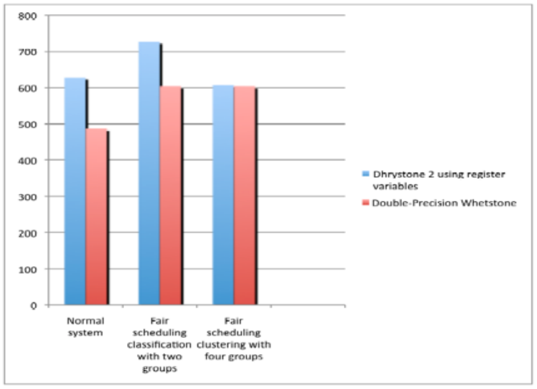

Fig. 5: Benchmark for Whetstone and Dhrystone 2 (More is better).

B. Benchmark for File Operation

The second graph (Fig. 6) shows benchmark for file operation with different buffer sizes; it measures the data rate when copying from one file to another and the number of characters that can be written to the file system per second.

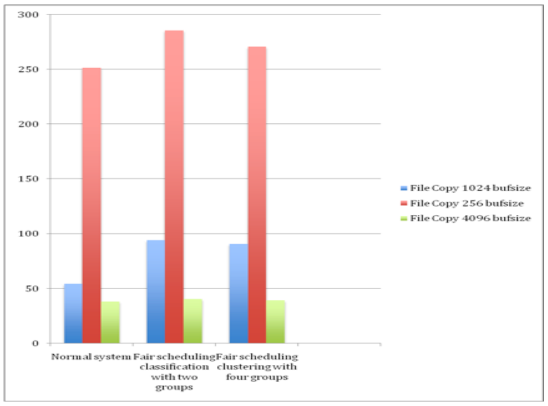

Fig. 6: Benchmark for File Operation with Different- Buffer Size (More is Better)

C. Benchmark of Process Intercommunication

The third benchmark is for inter-process communication using pipes; it measures the number of time processes can write to pipe and read from it per second, and it also tests the speed of communication as seen in Fig. 7.

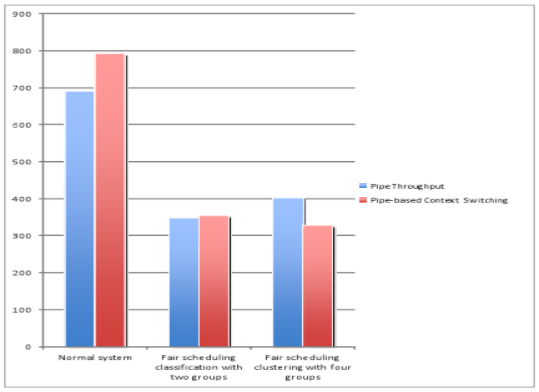

Fig. 7: Benchmark for Pipes Inter-process Communication

D. Process Creation and Image Replacement

This benchmark measures the process creation and image replacement rate, which means the number of time processes, can be created and that an image gets replaced with another process within specific time range as seen in fig. 8.

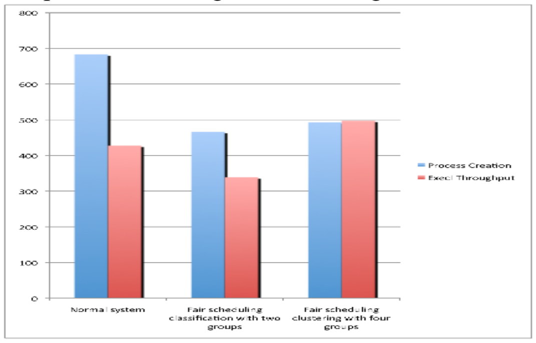

Fig. 8: Benchmark Process Creation and Image Replacement.

E. Shell Script Execution

It measures shell script execution time; these shell scripts do a common file transformation: first, it tests a single shell script, and after that, it tests eight concurrent shell scripts as seen in fig 9.

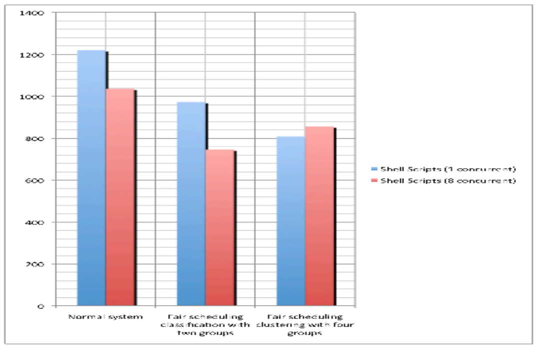

Fig. 9: Benchmark Process Creation and Image Replacement.

F. System Call Overhead

This measures system call overhead by calling the same system calls multiple times within a defined period of time and counts the number of calls, which indicates overhead to enter and leave the kernel as seen fig. 10.

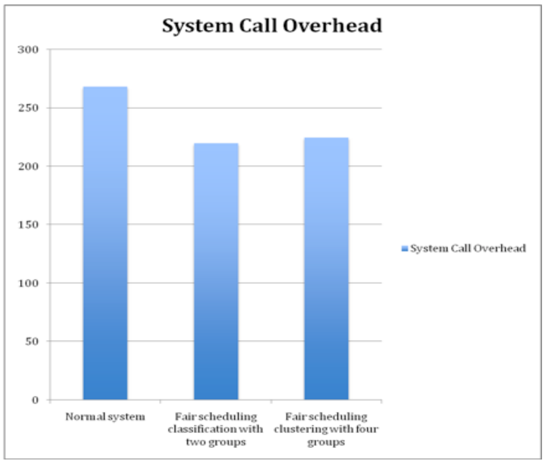

Fig. 10: System Call Overhead.

G. Interactive Application Benchmark

Interactive benchmark tools try to create a different kind of user interaction component like menus, compo boxes, listings, graphical drawings, as well as stimulating user action on these components to measure response time.

GTKperf Tool was used to create benchmarking graphs seen in fig. 11.

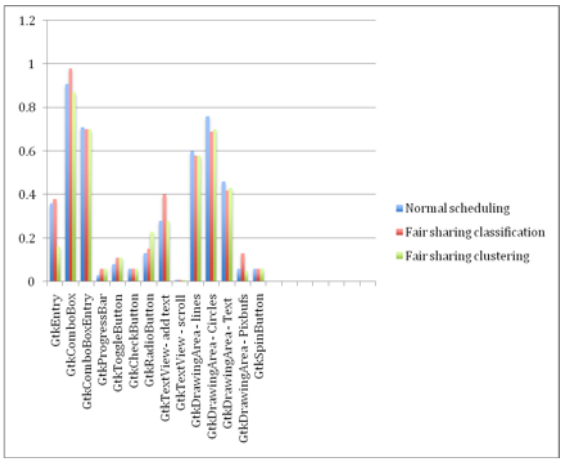

Fig. 11: Interactivity Benchmarks (Less is Better).

IV. Conclusion and Future Works

A. Conclusion

Process has a lot of feature or attributes that can be tracked or logged, and used in further processing. System calls are a good high-level process behavior, and it used in this research to collect information about processes. Tools like Dtrace supports live tracking of process. Aggregating the tracked information and output them in readable format with low overhead on host systems.

Classification and clustering can be implemented in user land, and the distribution of processes can be done from user land, while this implementation has limitation because not all the scheduling data are available.

Classification and clustering can be applied on different kind of processes, and give good result in grouping processes, based on similarity or based on user training data.

A fair scheduling of classified process groups achieves a good result in process performance, while it has drawbacks when it is comes to inter-process communication especially pipes, because processes that are communicating using pipes are allocated to different processors core.

A fair scheduling of clustered process groups achieves similar result to classification, while it has same draw backs when in it comes to inter-process communication.

A fair scheduling clustering or classification has poor handling of fast process creation; reducing classification and clustering time or decreasing time between each run can do a fix. Interactivity benchmark achieve the same result for normal, clustering, and classification with different in seconds.

B. Future works

The research can be improved in the future through the Implementation of the process as part of scheduler, and use data available to scheduler as parameter to process distribution. Also assign Inter-communicating processes to same processor, reducing the communication over head.

References

[1] Haldar, S., & Aravind, A. A. (2010). Operating systems. Pearson Hall.
[2] Tanenbaum, A. S., & Woodhull, A. S. (2009). Operating Systems Design and Implementation. Pearson Prentice-Hall.
[3] Choi, L. (2007). Advances in computer systems architecture. Seoul, Korea: Choi, Lynn; Paek, Yunheung; Cho, Sangyeun.
[4] Bradford, E., & Mauget, L. (2002). Linux and Windows Interoperability Guide. Prentice Hall Professional.
[5] Hailperin, M. (2007). Operating Systems And Middleware: Supporting Controlled Interaction. Max Hailperin.
[6] Love, R. (2007). Linux system programming. O'Reilly Media.
[7] Crowley, C. P. (1997). Operating systems: a design-oriented approach. Irwin.
[8] McKusick. M. K., Neville-Neil G. V. (2004). The Design and Implementation of the FreeBSD Operating System. Addison-Wesley Professional.
[9] Rodriguez, C. S., Fischer, G., & Smolski, S. (2005). The Linux kernel primer: a top-down approach for x86 and PowerPC architectures. Prentice Hall Professional.
[10] Douglas, S., & Douglas, K. (2004). Linux Timesaving Techniques For Dummies. Wily.
[11] Robbins, A. (1999). Larger Cover UNIX in a Nutshell, 3rd Edition. O'Reilly Media.
[12] Jepson, B., & Rothman, E. E. (2005). Mac OS X Tiger For Unix Geeks. O'Reilly Media.
[13] Fusco, J. (2007). The Linux Programmer's Toolbox. Prentice Hall.
[14] Cheney, S. (1998). Benchmarking. ASTD .
[15] Foster, J. C. (2005). Buffer Overflow Attacks: Detect, Exploit, Prevent. Syngress.
[16] Mauro, J., & Gregg, B. (2011). DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD. Prentice Hall.
[17] Witten, I. H., Frank, E., & Hall, M. A. (2011). Data Mining: Practical Machine Learning Tools and Techniques. Elsevier.
[18] Dougherty, D., & Robbins, A. (1997). Sed & awk. O'Reilly Media.
[19] Levine, J. (2009). Flex & Bison. O'Reilly.
[20] Rokach, L., & Maimon, O. Z. (2008). Data mining with decision trees: theroy and applications. World scientific.
[21] Lin, T. Y., Xie, Y., & Wasilewska, A. (2008). Data mining: foundations and practice. Springer.
[22] Janert, P. K. (2010). Data Analysis with Open Source Tools. O'Reilly Media.
[23] Cheney, S. (1998). Benchmarking. ASTD.
[24] byte-unixbench. (2012, 5 8). byte-unixbench. Retrieved 5 8, 2012, from code.google.com: http://code.google.com/p/byte-unixbench/

Authors

Ali Mousa Alrahahleh is an M.Sc Student in the Dept. of Computer Science, faculty of information technology, at the Middle East University.

Hussein H. Owaied is the faculty member and supervisor of the student in the department of Computer Science, Faculty of Information Technology at the Middle East University. Dr. Owaied is the corresponding author can be reached at the address: Dr. Hussein H. Owaied, Department of Computer Science, Faculty of Information Technology, Middle East University P.O.Box 383 Amman 11831 Jordan. Email: howaied@meu.edu.jo