Improving Scheduling on Symmetric Multiprocessing (SMP) Architecture Based on Process Behavior

By
Ali Mousa Alrahahleh

Student Number
400920056
Supervisor
Dr. Hussein H. Owaied

Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer Information Systems

Faculty of Information Technology
Middle East University

Amman-Jordan

May 2011/2012

Authorization Statement

I, Ali Mousa Abed Alrahaman Alrahahleh, authorize Middle East University to supply hard and electronic copies of my thesis to libraries, establishments, bodies, and institutions concerned with research and scientific studies upon request, according to university regulations.

Name: Ali Mousa Abed Alrahaman Alrahahleh

Date: 9/6/2012

Signature:

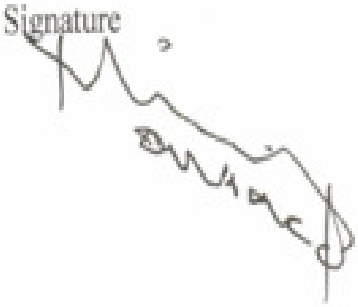
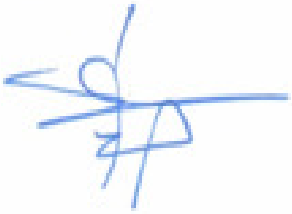
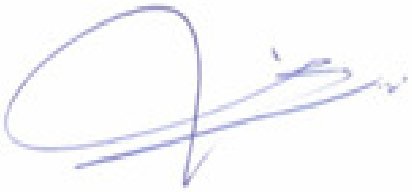
جامعة الشرق الاوسط
اقرار تفويض

انا علي الرحاحله افوض جامعة الشرق الاوسط بتزويد نسخ من رسالتي للمكتبات او المؤسسات او الهيئات او الافراد عند طلبها.

التوقيع: ٢٠١٢/٦/٩

Examination Committee Decision

This is to certify that the thesis entitled "Improving Scheduling on Symmetric Multiprocessing (SMP) Architecture Based on Process Behavior" successfully defended and an approved on June 2012

Examination committee Member	Signature
Supervisor Dr Hussein H. Owaied Associate Professor in the Department Of Computer Information Systems (Middle East University)	
External Examiner Dr. Abd Ellatif Abu Dalhoum Associate Professor in the Department Of Computer Science (University of Jordan)	
Internal Examiner Dr. Sherif Mahrous Gad Rezk Associate Professor in the Department Of Computer Information Systems (Middle East University)	

DEDICATION

I dedicate this dissertation to my parents, who first planted the seeds of knowledge and wisdom in me. From my birth and throughout the development of my life, they have encouraged me with love and care to seek out knowledge and excellence. They challenged me to pursue my dreams, which led me to the completion of this endeavor.

ACKNOWLEDGEMENTS

I wish to acknowledge my supervisor Dr. Hussein, who offered me guidance and assistance throughout this process. I must credit Mamoon who gave me help, hope, and encouragement along the way. I wish to thank all my friends and family, who helped me by contributing in many ways, big and small.

Abstract

This thesis presents a new method of scheduling for systems 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 which 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 to which group each process belongs. The third phase is using process groups as a parameter of scheduling on SMP 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. The fair distribution method was used in this research, where group elements (processes) are divided equally on processor cores, and it has a good result in terms of performance and response time, but it has limitations where there is inter-process communication.

الملخص

يقدم هذا البحث طريقة جديدة لجدولة على أنظمة المعالجة المتعددة المتماثلة **(SMP)** القائمة على سلوك عملية. الأسلوب يأخذ مزايا سلوك عملية التي تشمل مكالمات نظام لإنشاء مجموعات من عمليات مماثلة باستخدام تقنية اليات التعلم مثل تجميع أو تصنيف، ومن ثم القيام بعملية توزيع القرار على أساس تصنيف أو المجموعات. الطريقة جديدة مقسمة إلى ثلاث مراحل، المرحلة الأولى هي جمع البيانات عن العمليات وتحديد أي مجموعة فرعية من البيانات لاستخدامها في مزيد من المعالجة. المرحلة الثانية يتم استخدام البيانات التي تم جمعها في تصنيف أو تجميع لخلق النماذج من خلال تطبيق تقنية شائعة مماثلة لتلك المستخدمة في التعلم الآلي، مثل شجرة قرار لتصنيف أو **EM** لتجميع. وينبغي أن يتم تدريب ونظام التصنيف في هذه المرحلة، بعد وينبغي تطبيق تلك النماذج أو تصنيف المجموعات على نظام تشغيل لمعرفةالى اي عمليه تنتمي كل مجموعه. المرحلة الثالثة يتم باستخدام مجموعات عملية كمحدد للجدولة على أنظمة المتعددة النواه. ويمكن تحقيق ميزة أخرى عن طريق السماح للمستخدم النهائي لتدريب نظام لتصنيف نوع معين من العمليات، و توزيعها على معالجات محددة، لتقليص زمن الاستجابة أو كسب الأداء.. تم استخدام أسلوب التوزيع العادل في هذا البحث، حيث عناصر المجموعة (العمليات) مقسمة بالتساوي على نوى المعالج، ولها نتائج جيدة في زيادة الأداء و تقليص زمن استجابة ولكنها لم تحقق نتائج جيده عند التعامل مع عمليات تحتاج الى اتصال مع عمليات اخرى.

Content

List of Tables

List of Figures

List of Abbreviations

SMP	Symmetric Multi Processor
CPU	Central Processing Unit
EM	Expectation-Maximization
HALD	Hardware Abstraction Layer Daemon
PowerD	Power Daemon
PID	Process Identification
RTOS	Real-Time Operating System

Chapter 1

Introduction

1.1 Overview

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

Software tells hardware what to do by providing a list of instructions to be executed. These instructions manipulate data to provide useful information. While software needs access to hardware most of the time, it can't do this directly because this may result in data loss. If other software is trying to access the same hardware, it may abuse the hardware, resulting in physical damage or it may prevent other software from using the same hardware by holding it for a long time.

These issues introduce software called the operating system, which handles all low-level operations by providing a layer between software and hardware. 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 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 (Haldar & Aravind, 2010).

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) (Tanenbaum, Modern Operating Systems, 2008).

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 (Choi, 2007).

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 (Bradford & Mauget, 2002).

Preemptive Priority Scheduling is used on Windows (Hailperin, 2007), Linux (Love, 2007), Solaris (Crowley, 1997), and BSD (Marshal 2004) OSs. 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, or preempted by the scheduler because it takes all time quantum.

There are different types of priority scheduling. One of these types which is used by Solaris is a multi-level feedback queue (Rodriguez, Fischer, & Smolski, 2005). 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 (Douglas & Douglas, 2004). 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.

For example: if we assign a mix of interactive tasks and background tasks to each instance, it may provide a high response time and high throughput at the same time.

A process behavior is a type of requested resource and the frequency of request that can be tracked by monitor input/output (I/O) requests or system calls initiated by process.

Classification is supervised learning technique used to classify data and put it in meaningful groups. It is supervised because it depends on user to train the classification system by supplying a sample of real data with known groups for each element.

Another machine learning technique is clustering, while classification needs user to train the system, clustering works directly on data and generates groups based on similarity between individual items.

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.

While each core has it is own cache setting affinity of process may reduce cache misses and improve system performance.

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.

1.2 Problem Definition

There are 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 and provide recommendations to a scheduler in order to achieve good distribution of processes over processors, therefore the following problems have been identified:

1. The identification of processes' behavior
2. The classification of the processes' behavior
3. The distribution of the processes' behavior to the associated processors

1.3 Objectives

The objectives of this thesis are the following:

1. Design a methodology for increasing the performance and reducing the response time on SMP architecture by defining a new parameter to improve existing scheduling algorithm, which balances between different cores based on process behavior.
2. Collect data related to system calls in order to identify the processes' behavior.
3. Classify the identified processes behavior.
4. Devise a method for distribution of the processes' behavior to the associated processors according to the running processes.

1.4 Motivation

Because SMP is so public these days and any improvement on performance is going to be critical for real-life application on servers such as web servers, mail servers, and file sharing servers, and on desktop such as web browser and text editing, the result of this research can be applied to any multi-processor system and may achieve an increase in performance and a reduction in response time.

Chapter 2

Literature Review

In recent years, there has been an abundance of research and surveys presented in the context of SMP scheduling. This section presents a literature survey and an overview of recent research.

2.1 Literature Surveys

The following section introduces the latest work done on open source operating system schedulers (Linux, Solaris and FreeBSD):

2.1.1 Solaris

Solaris is a UNIX operating system. Solaris supports multiple levels of threads, a program in execution, and these processes: user level thread, lightweight process, and kernel thread. A process has its own resources and it represents user-executed programs or programs started by the system automatically; each process contains a thread or multiple threads, a lightweight process is only a mapping between user threads and kernel threads, which are physically existing threads created by kernel and swapped between different lightweight threads. A lightweight process should have at least one thread (Godbole, 2005).

Solaris uses a Multi-Level Feedback Queue (MLFQ) scheduling approach; MLFQ tries to address multiple problems:

1. Optimize turnaround time, which can be achieved normally by running small task first; it is very hard to depict process future run time.
2. Improve interactivity of OS, which can be achieved normally by using round-robin scheduling, but this results in a long turnaround time.

MLFQ maintains a different set of queues: each one of these queues is mapped to a priority level. When a process is submitted to the system, the scheduler places this process based on its priority in one of these queues.

Once the processor is idle, the scheduler selects a process from the top priority queue. If there are many processes in the queue, the processor uses round-robin to select among them; if the queue is empty, then the scheduler selects a process from the queue which has a priority less than the current queue.

MLFQ observes process behavior, and then makes decisions based on this behavior. For example, if a process block is waiting for I/O, the scheduler increases its priority by moving it to higher priority scheduler to increase response time in the next run. Or, if a process tries to use all allocated time in each turn, the scheduler decreases the process priority by moving it to lower priority scheduler to avoid starvation (H, Arpaci-Dusseau, & C., 2011).

From Version #2, Solaris started supporting scheduling classes. There are four scheduling classes shipped by default with Solaris operating system:

1- System class (SYS): It is used for operating system threads and guarantees fixed priority and unlimited time of execution for all system threads unless a high-priority process interrupts these system threads.
2- Interactive class (IA): This is the same as TS, but it increases priority of interactive applications, like a windowing system.
3- Time-sharing class (TS): This is the old traditional scheduling technique as mentioned in MLFQ: process priority calculated periodically.

4- Real time class (RT): This gives a high fixed priority to some of user processes; these processes have a higher priority than other user land processes, so RT processes can stop other processes when there is an action which RT processes need to process.

Solaris (versions 2.6 and later) has the ability to define a process set, which is only allocated to one processor, and the user can disable interrupt on these processors to meet real-time application requirements (Jim Mauro, 2001).

2.1.2 Linux

As other operating systems, Linux process scheduler tries to achieve a set of goals: fast process response time, good throughput, avoiding process starvation, and many others.

Linux scheduler is based on time sharing, which means the process time is sliced down and assigned for each process and every process on the system has a priority that increases the process' chance to be selected in next round. Priority is dynamic and it is derived using different kinds of complex algorithms.

Process in general can be classified into different groups:

A. Interactive Processes: These processes interact with the user for most of their execution time. They need user interaction to complete their task: these interactions include blocking to read from the keyboard or mouse (or other user interaction method) and these types of processes need acceptable response time ranging from 50 to 150 ms.
B. Batch Processes: These processes run in the background with few or no interactions with the user. Examples are database systems and web servers.
C. Real-Time Processes: These types of processes need a very short response time, such as sound and video.

Linux scheduler favors interactive processes over batch processes. On every round, Linux scheduler calls a function in order to determine the goodness of process. If the process uses the entire time quantum, it will not be selected when there are other processes with the same priority that did not finish their last time quantum, or are real-time processes.

Priority scheduler has some problems, and Linux tries to solve some of these issues:

A- The algorithm does not scale well: If there are a huge number of processes, computing the goodness for each one for every second is not efficient. Linux scheduler solves this issue by only computing the goodness of processes when it finishes its time quantum.

B- The predefined quantum is very large for a large system: If the quantum is very large, this may affect response time when there are a large number of processes.

C- Support of real-time system is weak: Since kernel is non-preemptive in Linux, this is not suitable for real-time systems because it needs a restricted time frame to execute within and kernel processes may take a long time.

SMP scheduling on Linux operating systems did not anticipate many changes to scheduling algorithms; the only thing that is important is to maintain the process affinity, which means to assign the same process to the same processor to maintain high cache hit (Bovet & Cesatí, 2005).

The Linux kernel 2.6 has a new scheduler which can perform the scheduling of different processes on constant time O(1), another important feature introduced by 2.6 kernel is kernel thread preemption, which means that kernel can stop kernel running threads and run another thread from user land (Rodriguez, Fischer, & Smolski, 2005).

2.1.3 FreeBSD

FreeBSD is one of several open source operating systems. It is derived directly from 4.3BSD and there are a lot of similarities between FreeBSD and Linux scheduler such as:

A. They both use a fair share scheduler.

B. They use process behavior as scheduling parameters to find out which processes to run.

C. They favor interactive processes over non-interactive ones.

One of the interesting features of FreeBSD scheduler is priority propagation, which means that if child processes start to use CPU intensely, its priority will be decreased and the priority of its parent will be decreased also because most background jobs try to fork child processes to increase performance (Example: make).

The FreeBSD scheduler is also effected by memory usage: if the system starts running low on memory and new processes request memory allocation, the scheduler tries to block recent running processes from running again to give a chance for the memory manager to swap pages from memory to secondary storage, and thus reduce the effect of memory thrashing.

There are different types of process classes in FreeBSD: the bottom half of kernel, the top half of kernel, real-time thread, and time sharing. The only difference between these classes and classes found on Solaris is the introduction of a new class called bottom-half class, which defines a set of threads that run in kernel mode and can be interrupted; the default class of user processes is time-sharing class (McKusick & Neville-Neil, 2004).

A new scheduler called ULE scheduler was introduced lately to replace the old scheduler on FreeBSD system, because the old scheduler doesn't take full advantage of

the new SMP architecture. SMP borrows some ideas from the Linux 2.6 scheduler to reduce scheduling time from O (n) to O (1), and uses different mechanisms to select a process to run. One of these mechanisms is interactive scoring, used to find an interactive task that is used in the computation of process priority. The interactive scoring is computed based on process-interactive process behavior; as an interactive process sleeps for a long time waiting for user input and wakes up for short period of time to do computation (Roberson).

The following section outlines some works related to scheduling algorithms.

2.2 Related Works

There is much research that has been done in the area of SMP scheduling algorithms, and the following are some works related to scheduling**:**

A. Operating systems maintain a set of thread queues; when any processor becomes available, one of these threads is assigned to it. In most of cases, this achieves high response time, but when the number of threads is increased, threads may experience some synchronization delays if they didn't execute in the right order or in parallel as the software designer/developer expected. This can be solved by using a special type of scheduling called gang scheduling, which groups programs and parallel threads of execution into a gang and it concurrently schedules an independent processor to each thread in the gang. This reduces spin, wait, and solve problems related to synchronization (IPPS).

B. An algorithm for scheduling parallel programs for execution is on a parallel architecture based on dynamic SMP processer cluster with data transfer on the fly. It takes a program graph and decomposes it into sub-graphs and treats each one as a

moldable task (Tchernykh & Trystram, 2003) .It determines which task is a parallel task that can be executed using an arbitrary number of parallel processors; they use penalty function to determine task execution efficiency versus an ideal parallel on given number of processors.

The scheduling algorithm is defined as set of steps:

1- Definition of a Moldable Task (MT) inside program graph: the main target of this step is dividing each graph into sub-graphs, and each sub-graph represents a Moldable Task which should be the smallest possible task. All external reads can only be performed at the beginning of the task, and all external rights can be performed at the end of the task; it is then the responsibility of algorithm to merge each small moldable task into a larger task in order to reduce communication and improve performance.

2- Determining the penalty function for each moldable task: this also includes the scheduling of each MTs for a range of available resources. The main goal of this step is to determine the penalty function for each MT created in the previous step. This is going to be used in determining the best scheduling of the MT and comparing it with the proposed one.

3- Assignment of each resource to each MT and their scheduling: in this step, every MT obtained in previous steps is mapped to a number of resources (Masko, Dutot, Mounie, Trystram, & Tudruj, 2006).

C. Many extensions were introduced for operating systems to improve scheduling on SMP architecture: one of these extensions is called ARTiS, which exploits the SMP

architecture to guarantee the preemption of a processor when the system has to schedule a real-time task.

The main work of ARTiS is to assign a number of processor to a real time operation and to provide a migration mechanism of non-preemptable tasks to reduce latency. These processes are assigned real-time process and are not exclusively for that type of process to ensure that there is no waste of resources.

ARTiS was targeting a response time below 200us; the key for achieving such a goal is a good load-balancing algorithm and effective migration processes, which are implemented by most RTOS (real time operating systems) and not all GPOS (general purpose operating systems). ARTiS tried to mix functionality of GPOS and RTOS in new SMP platform called Asymmetric Real-Time Scheduler.

ARTiS partitions processes and processors into two types (classification): NRT CPUs (Non-Real-Time) and an RT CPU set (Real-Time). Each one has a particular scheduling policy that guarantees that real-time process can be preempted by any other process (Non-Real Time) on RT CPU. This reduces latency, increases efficiency, and provides a mechanism to move NRT from RT processors to prevent starvation.

Migration happens when processes try to disable interrupts on an RT processor or try to improve load balancing between processors.

Measuring of Performance was done by measuring the elapsed time between the hardware generation of an interrupt and the execution of code concerning the interrupts, ARTiS results in reducing latency for real-time process; the only limitation that was faced is when there are many RT assigned to one RT processor (Piel, Marquet, Soula, & Dekeyser, 2007).

D. Another solution was using dynamic performance metrics when making scheduling decisions; it uses hardware performance counters to make better scheduling decisions. The data stored on these counters is subject to offline analysis which determines scheduling thresholds and reports them back to kernel. Using these thresholds can help the scheduler to make more efficient decisions, but this research only focuses on cache affinity (Mulvihill & Grobman, 2012).

E. Genetic Algorithms were used to solve problems with large solution spaces in scheduling. Real-time scheduling was one of these problems where genetic algorithms are applicable because real-time scheduling has a very large search space and it has dynamically-changing problems. For example, new high-priority tasks may arrive which need to be scheduled immediately. This may change the problem definition or a resource may malfunction, which affects scheduling decisions as well, and real-time systems have a variety of constraints to be achieved (both hard and soft constraints). The result of incorporating genetic algorithms to fix scheduling problems was incredible; it is now used on production lines (Montana, Brinn, Moore, & Bidwell, 2001).

Chapter 3

Proposed Work

3.1 Overview

The proposed methodology for the system presented as seen in figure-3.1 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. The following sections describe proposed work phases:

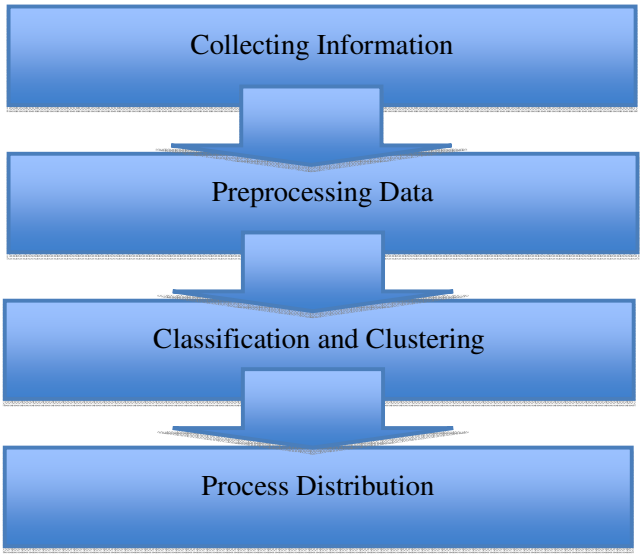


Figure 3-1: Proposed Steps for Scheduling on SMP Systems

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. Figure 3-1 show these phases (a proposed model); each phase will be explained in next section in detail:

3.2 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 (Robbins, 1999), and Ktrace only works on FreeBSD and early Mac OS (Jepson & Rothman, 2005).

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 (Fusco, 2007).

Figure 3-2; shows a sample of Strace output where process tries to open and file and close it after doing a set of operation on it .

```
open(".",
O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY|O_CLOEXEC)
= 3
fstat64(3, {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
fcntl64(3, F_GETFD)                 = 0x1 (flags FD_CLOEXEC)
getdents64(3, /* 18 entries */, 4096)   = 496
getdents64(3, /* 0 entries */, 4096)    = 0
close(3)                            = 0
```

Figure 3-2: Sample Strace Output

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 (Cheney, 1998), 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 (Foster, 2005).

```
845 AppleFileServerCALL  open(0xbffff1f0,0,0x1b6)
845 AppleFileServer NAMI
"/Library/Preferences/com.apple.AppleFileServer.plist"
```

Figure 3-3: Sample Ktrace Output after Being Processed by Kdump.

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. FreeBSD and early versions of Mac OS X only support it.

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 (Mauro & Gregg, 2011)

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 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 figure 3-4.

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

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

Figure 3-4: 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 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. 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.

```
2005 Jul 19 13:33:15, load: 0.24, disk_r:  95389 Kb,  disk_w:      0 Kb
UID    PID   PPID CMD            DEVICE  MAJ MIN D          BYTES
0     99      1 nscd           dad1    136   8 R          16384
0   7036   7033 dd               sd5      32  42 R        15826944
```

Figure 3-5: Sample of Dtrace Collected Data

3.3 Data Preprocessing

Dtrace handles most preprocessing, 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 figure 3-4), and another one for data – data should conform to the description in meta section.

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 (Witten, Frank, & Hall, 2011).

```
% 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}
```

Figure 3-6: Sample ARFF Header

ARFF data sections contain comma-separated data; each column is mapped to a definition found in the header as seen in figure 3-7.

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

Figure 3-7: 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 (Dougherty & Robbins, 1997).

```
for(i=1;i<=length($0);i++) {
# get the character to be checked
char=substr($0,i,1);
# is it an upper case letter?
j=index(UC,char);
if (j > 0 ) {
# found it
out = out substr(LC,j,1);
} else {out = out char;}}
printf("%s\n", out);}
```

Figure 3-8: Sample AWK Script.

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 script 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 (Levine, 2009).

3.4 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 determines the final decision. A sample decision tree looks like figure 3-9 (Rokach & Maimon, 2008) (Lin, Xie, & Wasilewska, 2008).

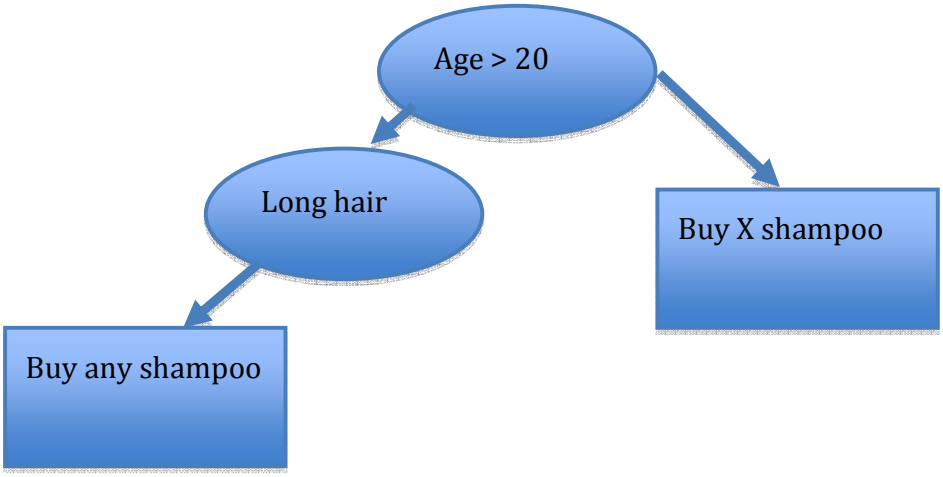


Figure 3-9: Sample Tree Model.

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 (Lin, Xie, & Wasilewska, 2008).

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 (Janert, 2010).

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.

3.5 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.

3.6 Benchmarking and Measuring Results

Benchmarking is "to take a measurement against a reference point." (Cheney, 1998). 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 to 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:

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.

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.

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 (byte-unixbench, 2012):

- 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, cos, sqrt, 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.

Chapter 4

Implementation and Results

4.1 Introduction

Implementation was done in several phases. Phases were similar to the proposed work and output of each phase was an input to the next phase. Here is a list of these phases:

4.2 Information Collection

The first step in implementation was determining what kind of process behavior should be collected; data to be collected should meet the following rules:

1- Easy to be collected.
2- Easy to process.
3- Covers all the interaction between software and hardware.
4- Gives a great indication about resource allocation.

System calls meet all the previous rules. First, it is easy to be collected and processed, because there are many user land-tracking tools running on different kinds of operating systems for system call tracking, and these tools have options to increase or decrease the level of details and cover all the interactions between hardware and software because the only way for accessing hardware or asking for the operating system it serves is a system call. This means that system calls give statistics about resource allocation – especially hardware.

There is a lot of data that can be collected about system calls, such as system call name, parameters, return data, and error code. While including them all can increase end

system accuracy, it may affect the flexibility of the end system and make it only suitable for a small subset of software which was running when training data had been collected.

A small subset of statistical information about system call has been chosen, which includes only system call name, number of calls, and the process name because they are shared among all processes and result in a flexible module.

Dtrace was used to monitor running processes. The Dtrace scripts run for a defined period of time (30 seconds) and collect the following information about the process:

1- Process name
2- System call name
3- System call frequency

The result of Dtrace script is a raw data. Table 4-1 shows a sample of collected data, with each row representing a map between a system call, process, and number of calls.

Table 4-1: Sample of System Call Data**.**

System call	Process name	Number of calls
fcntl	mds	603
lseek	Opera	617
recvfrom_nocancel	Opera	633
lstat64	sudo	640
geteuid	RealPlayer Down	720
write_nocancel	Opera	732
select_nocancel	dbfseventsd	760
close	tcsh	762
__sysctl	iTerm	784

Dtrace script is programmed to collect information for a specified period of time (30 seconds) and it will report data back once the 30 seconds have finished.

4.3 Data Preprocessing

After collecting system call information, a transformation process begins to transform data from the raw format as collected by Dtrace script to a format that a clustering or classification tool can understand. This has been handled using AWK script which adds the appropriate ARFF headers including: -

1- Relation name
2- Data definition, which includes attribute names and attribute types

Data definition in the ARFF file includes the following attribute system call, number of calls, and process name. Figure 4-1 shows a sample of the ARFF format where data definition includes a column called number_of_calls and the data section includes a couple of records showing system call names, process names, and number of system calls.

```
@attribute number_of_calls numeric
@data

access,RealPlayer_Down,1
access,SymantecALS,1
access_extended,mdworker32,1
audit_session_join,launchd,1
audit_session_self,UserEventAgent,1
```

Figure 4-10: Data after Being Transformed

Transformation also includes removing spaces and other special characters and replaces them with underscores to prevent any crashes when doing the classification process.

Before doing classification or clustering, the system should be trained using sample data: this should be done for one time before running the system; training requires a modified sample of data which includes the classification groups.

Classification groups should be added manually by hand, the current implementation has two classes/groups for processes, which run in the background, and are interactive. Figure 4-2 show modified sample data (training data) after adding the group's name.

```
@attributenumber_of_calls numeric
@attribute type  {INTERACTIVE,BACKGROUND}
@data
access,RealPlayer_Down,1,BACKGROUND
access,SymantecALS,1,BACKGROUND
audit_session_join,launchd,1,BACKGROUND
audit_session_self,UserEventAgent,1,BACKGROUND
bsdthread_create,Finder,1,BACKGROUND
bsdthread_create,Mail,1,BACKGROUND
```

Figure 4-2 : Classification Group to Training Data.

4.4 Classification and Clustering

The classification model can be built based on training data; this classification model will be used in the further classification of the request. WEKA has a command to build the classification model and save it as a file for further use. The classification model should have good accuracy; the classification model in this implementation has the

following matrices as reported directly from WEKA tools, using decision tree classifications:

1- Decision tree has 19 leaves
2- Decision tree size is 20
3- Time taken to build the model is 0.05 seconds
4- Time taken to test model on training data is 0.03

Once the classification model is ready, it can be used to classify processes. The classification was done using a WEKA decision tree because it has a good accuracy.

After the data has been collected and all preprocessing has been done, classification can be applied by passing this data to the classifier beside the model which has been built before. The column which the user needs to predict should be passed as well: in this case, the column which needs to be predicated is the group column. WEKA reports back a prediction for each row. Predication contains redundant prediction since the process name is repeated with each system call statistics row. A sample of prediction can be seen in Table 4-2: the first column shows the instance number, the second one shows the default value for this instance which has been set in the preprocessing phase, the third column shows the prediction, and the forth column shows if the default value mismatches with the prediction.

An ID in classifier prediction output identifies a process; each ID matches the order in the collected raw data about process. A second round of processing is needed to remove any redundant information to make each record unique, referring only to one process, and to attach each ID with a process name. This has been handled by AWK (a text processing tool).

Table 4-2: Sample Prediction as Reported by Decision Tree Classifier

=== Predictions on Test Data ===			
inst#	actual	predicted	error
1	1:INTERACT	1:INTERACT	
2	1:INTERACT	1:INTERACT	
3	1:INTERACT	1:INTERACT	
4	1:INTERACT	2:BACKGROUND	+
5	1:INTERACT	2:BACKGROUND	+

The prediction will be used in the next phase to do distribution of processes. Table 4-3 shows the output of post-processing scripts: the data includes process groups and PID (process identified), which are used by operating system to track processes. This can be retrieved from process name by using operating system utilities. PID helps in further processing because most of operating system utility only accepts PID. As seen in Table 4-3, the following programs have been classified:-

1- Opera is a web browser.
2- Abiword is a text editor.
3- Firefox is a web browser.
4- Xorg is a window manager.
5- Hald (Hardware Abstraction Layer Daemon) is daemon which runs in background providing a live database of devices connected to the system.
6- Gconf-d is daemon running in the background.
7- Sendmail is a mailing server.
8- Hald-addon-storage is a plug-in for hald.
9- Httpd is a web server (apache web server)
10- Powerd is daemon running in the background that is responsible for power management.

A daemon is an application running in the background and providing services periodically without the end user's involvement like web servers, mail servers, etc.

Table 4-3: A Sample Process Classification

Process Name	Group Prediction	PID
opera	Interactive	1670
abiword	Interactive	1692
firefox-bin	Interactive	1658
Xorg	Interactive	1636
hald	background	1603
gconfd-2	Background	1691
sendmail	Background	1474
hald-addon-storage	Background	1603
httpd	Background	1712
powerd	Background	1323

Clustering can be used instead of classification, but the weakness of clustering is that the user has no control over the results of clustering. This is unlike classification, where the user can train the system to identify groups based on preset specifications; in clustering, the output is based on similarity between data, resulting in unpredicted and changing numbers of groups and patterns that appear.

The clustering process is similar to classification: first, a model should be built based on sample data. The only difference in this phase is that cluster doesn't depend on an attribute to make groups; it only makes groups based on similarities between processes system calls. The result of building the model is the number of clusters, as shown in Table 4-4; the number of clusters varies based on the nature of the data and the algorithm used.

Table 4-4: EM Clustering Results

Cluster ID	Number of Element	Percentage
0	44	31%
1	82	58%
2	3	2%
3	13	9%

The model will be used in the next phase to cluster processes; output of the second phase, as in classification, is the prediction of process group or cluster. These groups will be used in further phases, especially process distribution, because the output of clustering predication is similar to classification. The prediction output needs post processing to remove any redundant information, make each record unique, refer only to one process, and to attach each id with process name. Table 4-5 shows a sample of prediction for each row:

Table 4-5: Clustering Predication Results.

Row ID	Cluster ID
122	0
123	0
124	3
125	3
126	3
127	3
128	3
129	3
130	3
141	2
142	2

After post-processing, process groups will be cleared and can be used in process distribution. Table 4-6 shows data after post-processing: it includes process name, PID, and process group. For more information about each process, please refer to the classification phase, since the same process data has been used.

Table 4-6: A Sample Clustering Data after Post-Processing.

Process Name	Process ID	Process Group
Firefox (web browser)	1658	3
Xorg (Windows manager)	1636	2
Powerd (power manager)	1323	0
Abiword (text editor)	1692	0
Send mail (mail server)	1474	1
Httpd (web server)	1712	1
Dwm (windows manager)	1638	0

As seen in the clustering result, clustering may result in unclear pattern and grouping, which may not useful except for experimenting. For example, Abiword is a text editor and it is in the same group which includes power daemon; they do not share much in common from the end user point, but they share the same system calls with same call frequency when the clustering model is applied.

Classification has a superior result when it comes to controlling the system and provides a clear decision when the training data is based on a solid operating system user or designer experience, whereas clustering is good at finding similar processes, requesting the same system calls, and same resources – this will help in distributing the work load among processors if fair distribution is used, or it can help to isolate processes which request high frequency of system calls (block too much) and allocates them to one processor, reducing context switching on other processors.

4.5 Distribution of Processes Groups

Different kinds of distribution schemes can be done; the one in use in this research is fair distribution, where each processor core has same number of processes from each group. This one of the simplest distributions: other distributions can be made based on user requirements; for example, an advanced user can train the system to recognize real time processes and allocate them to one processor to achieve real time response for these processes, or the system can be trained to do complex allocation: for example, 30% of group A to processor 1 and 70% of group B to processor 1.

Clustering can be used for load balancing because it groups similar processes in one group, this makes it efficient in distributing load because grouping is done mostly based on system call request frequency. This means high processing from the kernel side: when group instances are distributed on different cores, this leads to distributing kernel processing load to other cores as well.

Fair scheduling has also been used for cluster group instances distribution, which means every processor core has the same share of the same group.

Operating systems can be shipped with a default distribution, which can achieve a good utilization, and provides high response time. Operating systems also should provide a flexible way for an advanced user to train the system to accept new classification criteria and a new distribution model.

After distribution has been completed, each process will be bound to one processor core. Table 4-7 shows the classification of the distribution of processes on dual-core processors using fair scheduling; each core has nearly half of interactive and background processes in the system.

Table 4-7: Fair Share Distributions on Dual-Core Hardware.

Process Type/Group	Process Name	Process ID	Processor ID
Interactive	Pidgin(Yahoo messenger)	1966	0
Interactive	Firefox(web browser)	1948	0
Interactive	Opera(web browser)	1960	1
Interactive	Xorg(window manager)	1584	1
Background	Hald (hardware abstraction layer)	1545	0
Background	Syslogd (system log daemon)	1051	0
Background	Sendmail (mail server)	1411	0
Background	Httpd (web server)	2214	1
Background	Powerd (power manager deamon)	1265	1

Fair scheduling using clustering is different from classification since the cluster total number of groups is different from one model to another; Table 4-8 shows fair scheduling clustering and distribution, which results in four groups:

1- Group zero has 2 elements.
2- Group one has 12 elements.
3- Group two has 0 elements.
4- Group three has 2 elements.
5- Table 4-8: A Sample of Clustering and Distribution
6- Table 4-9: A Sample of Clustering and Distribution

Table 4-10: A Sample of Clustering and Distribution.

Process ID	CPU ID	Process Name	Group ID
1948	0	Firefox	3
414	0	wpa_supplicant	1
1418	0	cron	1
1960	1	opera	3
1966	0	pidgin	1
1584	0	Xorg	0
1265	1	powerd	0
3632	0	sh	1
1545	0	hald	1
1609	1	dbus-daemon	1

4.6 Benchmarking and Result

Benchmark provides a measurement of improvement after applying the new scheduling policy; in these sections, two types of benchmarks 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.

4.6.1 Unix Bench

This section will include the result reported back by the Unix bench tool for:

- A system with normal scheduling
- A fair scheduling classification with two groups (interactive, background)
- A fair scheduling clustering with four groups

Benchmark results will be grouped based on their nature and type; each graph will show the final score of the bench mark, then the score is computed based on an index value: an increase in score value means better results.

Benchmark was done on same machine with same number and type of processes, and has repeated for a number of times to make sure that the result is stable.

Here is a list of benchmark conducted using Unix Bench:

A- Benchmark for string and mathematical operation

This benchmark focuses on string and mathematical manipulation as seen in Figure 4-3; it emulates operations done in compiler and linkers, and it also measures the efficiency of cache memory.

As seen in Figure 4-3; fair share distribution based on classification groups has achieved the highest score in Dhrystone, while fair share distribution based on clustering groups has the second highest score in Dhrystone benchmark, normal system has the lowest score in the same benchmark.

For Double-Precision Whetstone, fair share distribution based on Clustering groups has the same score as fair share distribution based on Classification groups, which is near 600, while normal scheduling achieved less than 500.

The result of this benchmark shows that fair share distribution based on classification groups and fair share distribution based on clustering groups have a high score in mathematical and string operation, which is used a lot in compiler and text manipulation programs, giving similar application a boost and increase in performance.

For more information about Double-precision Whetstone and Dhrystone 2, please refer to the proposed work section.

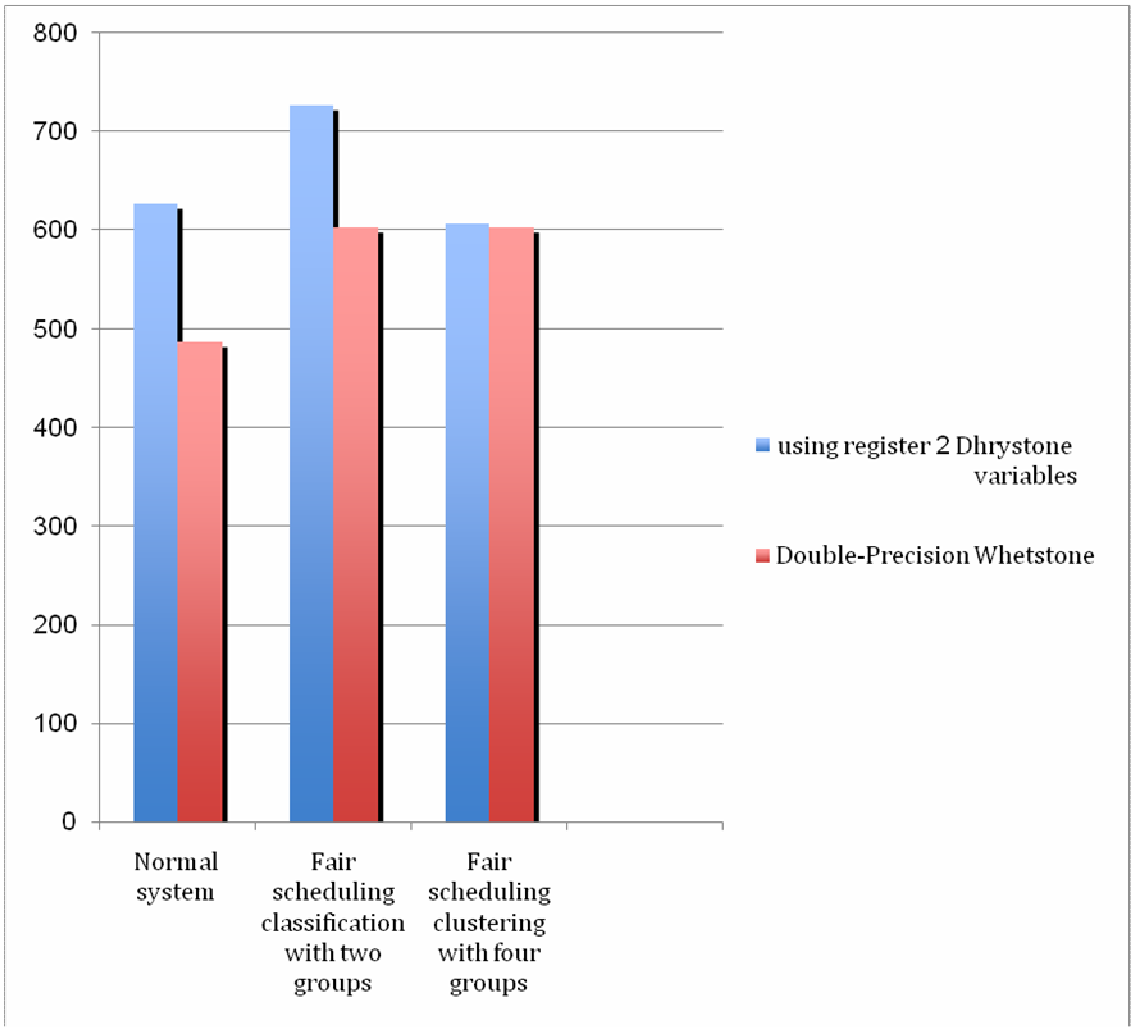


Figure 4-3 : Benchmark for Whetstone and Dhrystone 2 (More is better).

B- Benchmark for File Operation

The second graph (Figure 4-4) shows benchmark for file operation with different buffer sizes; it measures the data transfer rate when copying from one file to another and the number of characters that can be written to the file system per second.

As seen in Figure 4-3; fair share distribution based on classification groups has achieved the highest score for file operation (copying from one file to another), when the

buffer size is 265, while fair share distribution based on clustering groups has the second highest score in the same test, on other side normal system becomes last in score.

When the buffer size increased to 1024, fair share distribution based on classification groups has achieved a score of 91, while fair share distribution based on clustering groups has a score of 89, which is nearly the same as classification results. The normal system has the lowest score, which is near 50 points.

Increasing in buffer size reduces the score of all kinds of distribution including normal system distribution, For example increasing the buffer size from 265 to 1024 reduces the score of fair share distribution based on classification groups from 290 to 91, and for fair share distribution based on clustering groups from 280 to 90, which is nearly a 200 score points for both test.

Using huge buffer size 4096, makes all the tests including fair share distribution based on classification groups, fair share distribution based on clustering groups and normal distribution achieve the same result which is 40 scoring points.

Using fair share distribution based on classification and clustering groups leads to a good performance gain when dealing with file operation; especially when small buffer is used.

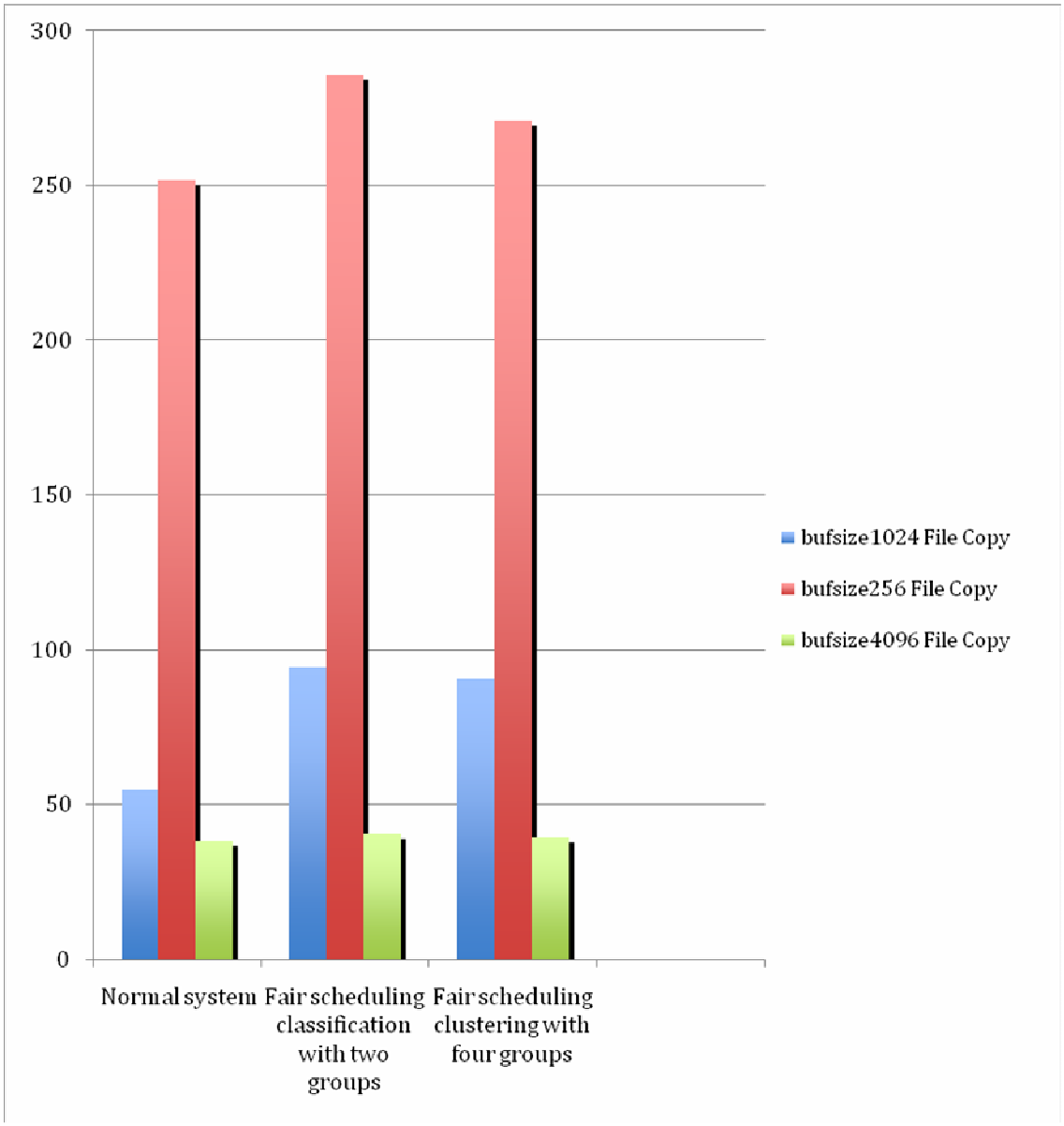


Figure 4-4: Benchmarks for File Operation with Different Buffer Size (More is Better)

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 Figure 4-5.

Normal system has the highest score for pipe throughput and pipe-based context switching, which is 800 for pipe throughput and 700 for pipe-based context switching,

while fair share distribution based on process classification and clustering groups has a low score for pipe throughput and pipe context switching due commendation overhead between different CPU cores.

Fair share distribution of processes based on Clustering groups has a better result pipe throughput than fair scheduling based on classification groups. The difference between both is 50 scoring points, while classification depends on user to train the system, clustering has a heuristic nature, and do clustering based on process similarities.

The probability of having inter-commentating process on the same core increases for fair share distribution of processes based on cslustering, since clustering has more groups than classification in current case.

Pipe-context switching score is higher than pipe throughput for normal distribution and fair share distribution of processes based on classification, except for fair share distribution of processes based on clustering, where pipe throughput score is 400 and pipe based context switching is 320.

The result of this test shows that fair share distribution based on clustering groups and classification groups doesn't has good score, when inter-communication is not an effective parameter in scheduling decision.

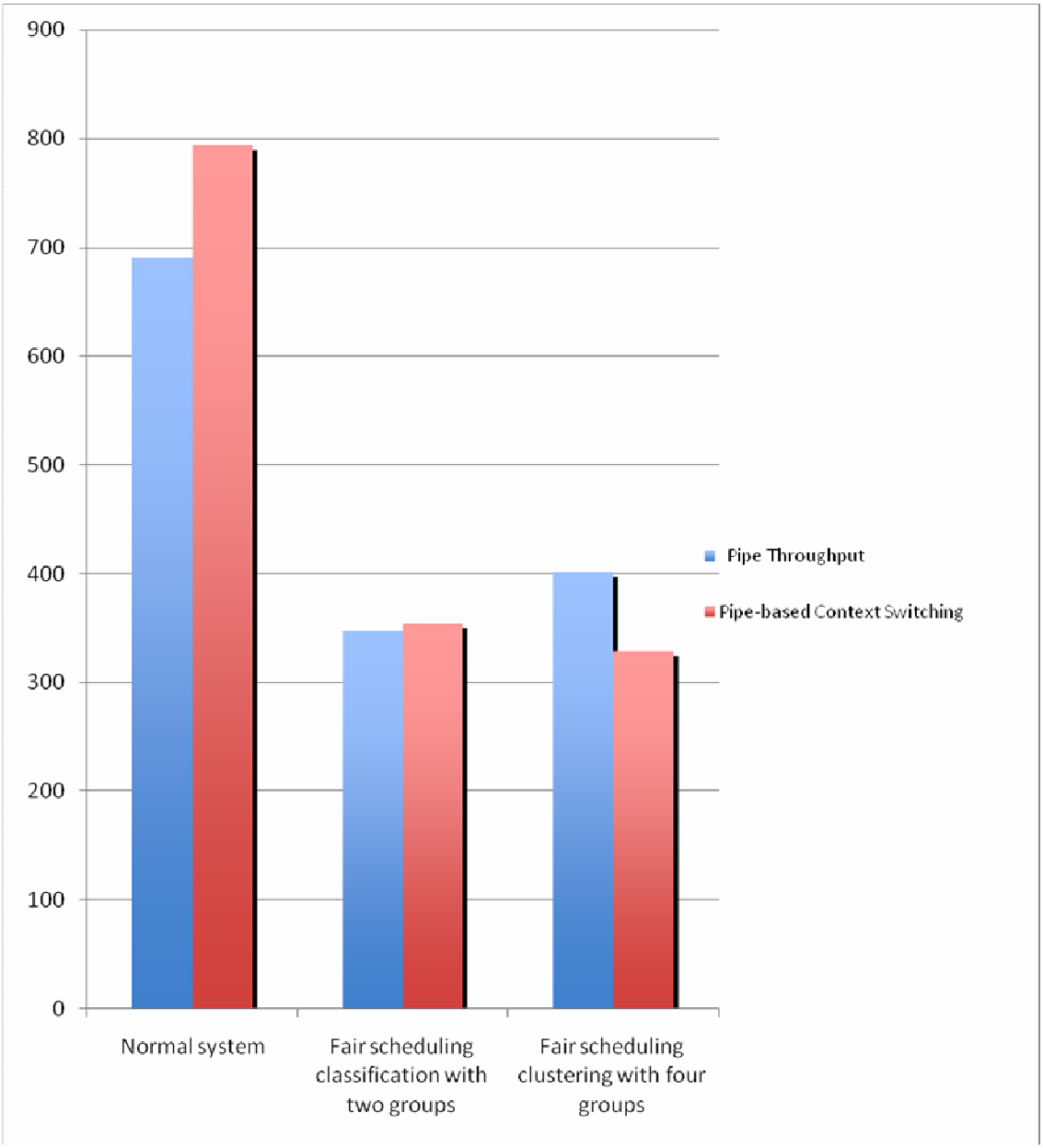


Figure 4-5: Benchmark for Pipes Inter-process Communication (More is better).

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.

This benchmark is not related directly to fair share distribution using clustering or classification, but it is a good indicator of system flexibility, when handling a huge number of newly created processes.

Normal system has the highest score for creating processes, while fair share distribution of processes based on clustering groups become next in score, and the least score is for fair share distribution of processes based on classification groups, with a difference of 20 scoring points.

Fair share distribution based on clustering groups has a score of 500 for process image replacement. Fair share distribution based on classification groups has a score of 330 for the same operation, and normal system has a score in between, which is 430 scoring points.

This benchmark shows; that all kinds of distribution including normal distribution, fair share distribution based on clustering, and fair share distribution based on classification has close scores, except for process creation in normal system, because it has the highest score among all other distributions.

Fair share distribution based on clustering and classification can be used without mush loss in process creation and image replacement performance, and sometime good distribution may result in increasing of performance of previous operation.

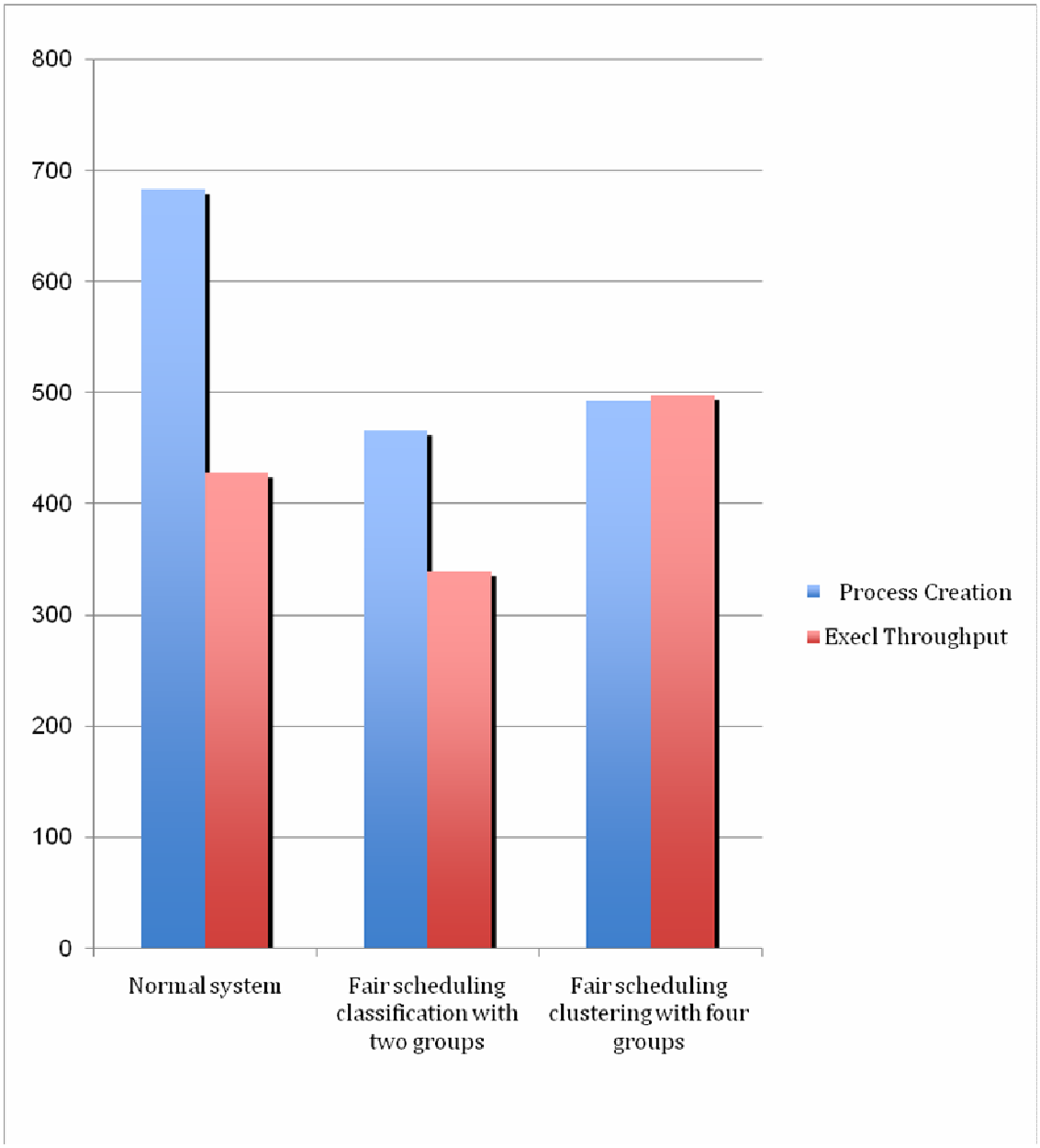


Figure 4-6: Benchmark Process Creation and Image Replacement (More is Better).

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.

Normal distribution has the highest score among the entire test for running one shell script, while fair share distribution using classification group has second highest score, and fair share distribution using clustering groups has the lowest score.

Figure 4-7; shows that normal system achieves highest score, while fair distribution has satisfactory result.

Benchmark for eight concurrent shell scripts has the following results: -

1- Normal distribution has a score of 1100, which is the highest score.
2- Fair share distribution based on clustering groups has better result than fair share distribution based on classification groups, achieving 900 score points, which is less than normal distribution by 200 score points.
3- Fair share distribution based on classification groups has lowest score, achieving 700 scoring point.

All of distribution tested in this benchmark achieved better score for single shell script, except fair share distribution based on clustering which has a better score when the number of concurrent shell scripts increased.

This benchmark shows that normal system has superior result due to high optimization, and load balancing. If this optimization applied to other distribution, it may increase scoring points.

Fair share distribution based on clustering groups achieves high score for concurrent scripts, making it suitable for high parallel independent processes, while fair share distribution based on classification needs a little modification to increase performance for concurrent shell scripts.

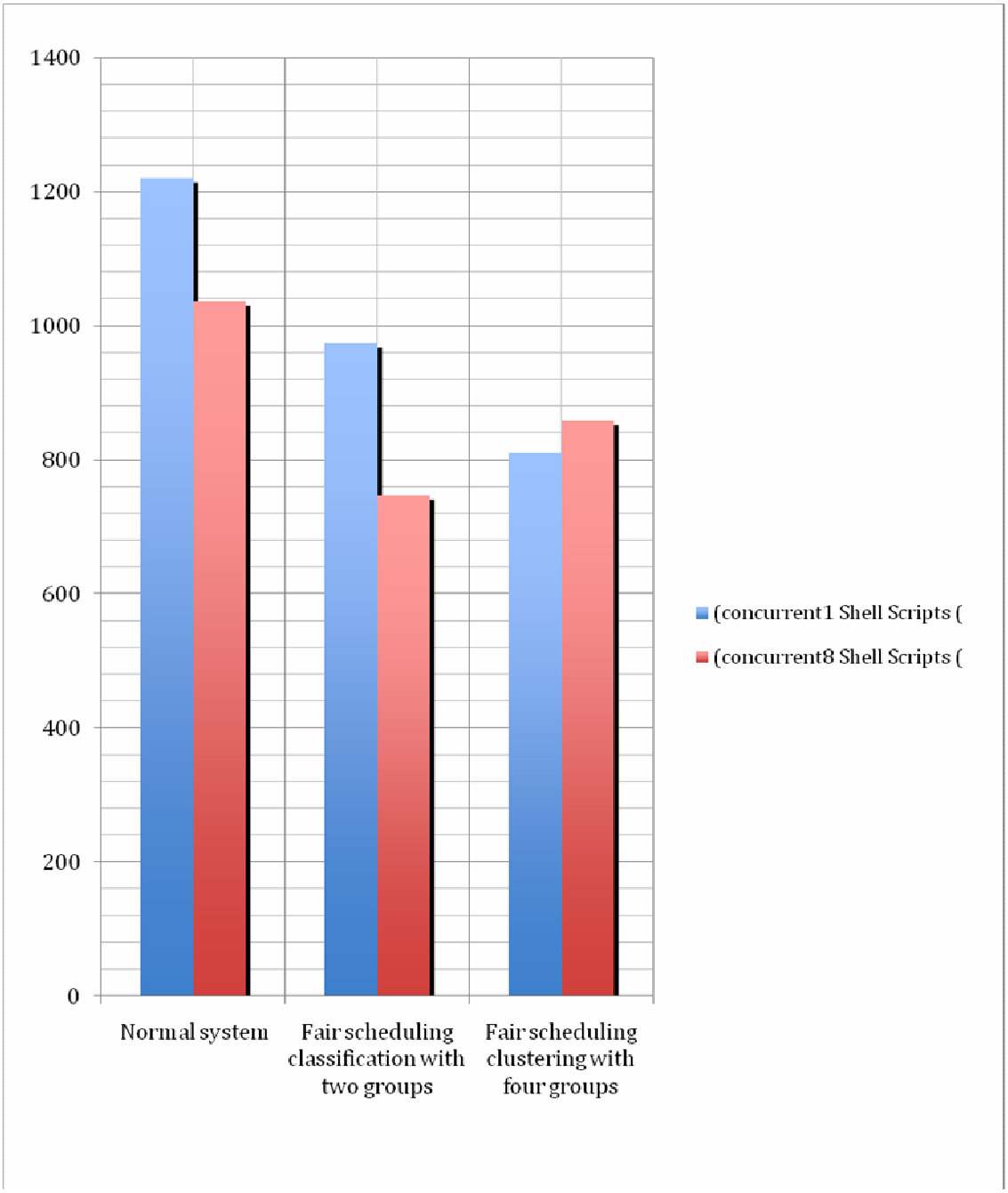


Figure 4-7: Benchmark Process Creation and Image Replacement (More Is Better).

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.

Figure 4-8; shows that fair share distribution using classification and clustering, achieves nearly same result, while normal system has the lowest over head when doing system calls.

Normal system has the lowest system call overhead, while both fair share distribution based on clustering and fair share distribution based on classification have nearly the same cost when doing the same system call.

System call overhead in this benchmark is not a good indicator for overall system call overhead, since only one system call has been used in this benchmark, while there is a wide variety of system calls, which depends on external or internal devices.

External dependence like networking latency or I/O overhead, makes it hard to take other system call in consideration, when measuring system call overhead.

This benchmark shows that normal distribution, fair share distribution based on classification and fair share distribution based on clustering have nearly the same result for one system call overhead, while both fair share distribution based on classification and fair share distribution based on clustering should be modified or replaced with other distribution to increase system call performance and reduce total overhead.

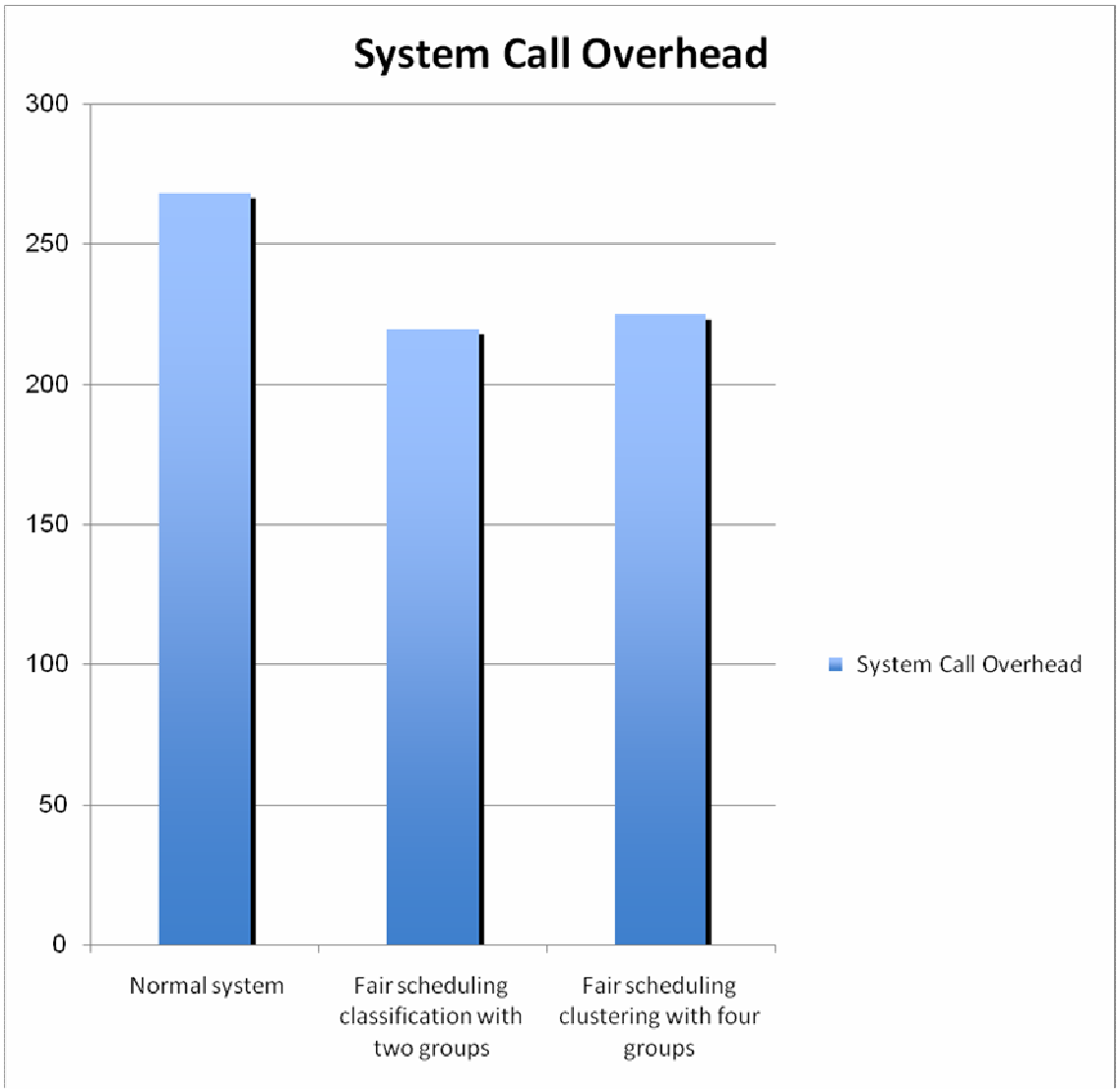


Figure 4-8: System Call Overhead (More is better).

4.6.2 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.

Figure 4-10; shows, that fair share distribution based on classification or clustering doesn't add much performance gain for interactive application.

This benchmark uses millisecond to measure response time, making a little different in response time an advantage of one approach over another, while this difference exists, it won't be visible to the end user.

This test shows that all types of distribution used in this benchmark have acceptable response time, and any of them can be used without affecting response time for interactive application.

GTKperf Tool was used to create benchmarking graphs seen in figure 4-10.

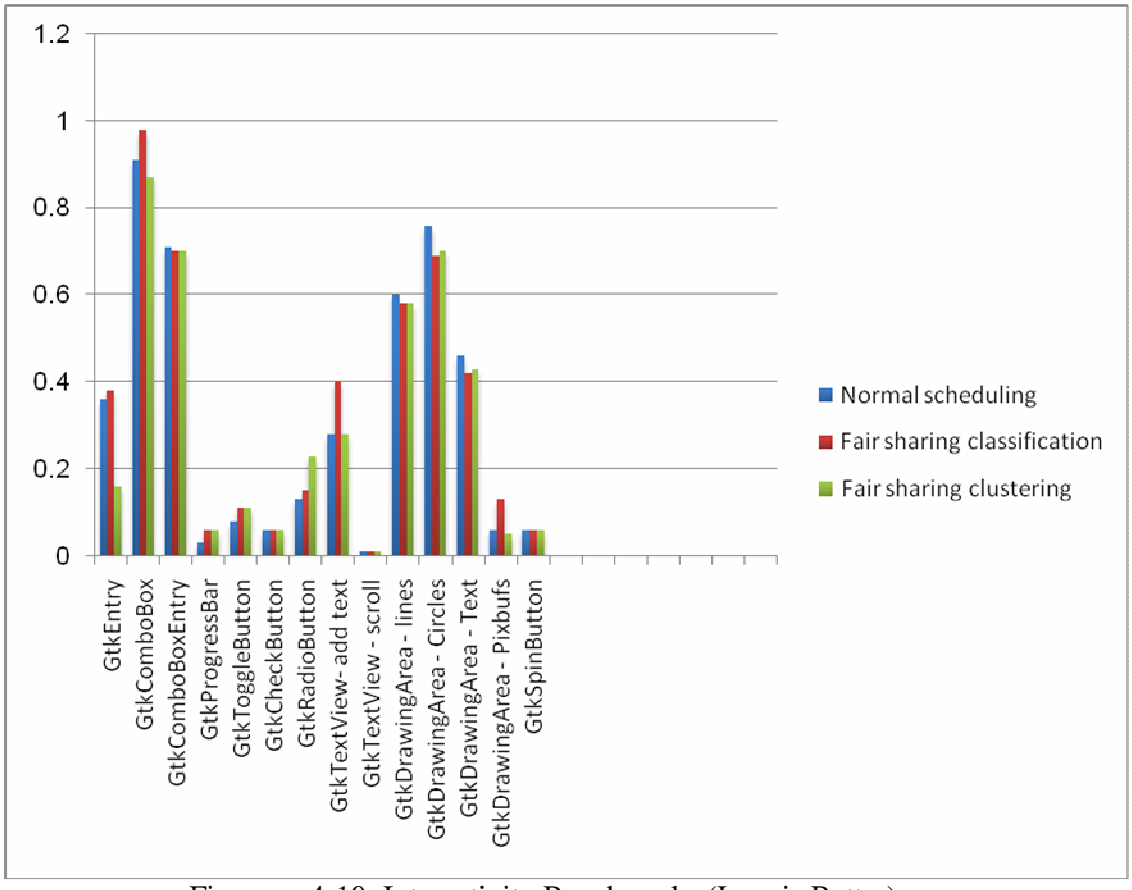


Figure 4-10: Interactivity Benchmarks (Less is Better)

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Process has a lot of features or attributes that can be tracked or logged and used in further processing. System calls are good high-level process behaviors, and they are used in this thesis to collect information about processes.

Tools like Dtrace support live tracking of process. Aggregating the tracked information and creating output for them in a readable format results in 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 too, although this implementation has limitations because not all the scheduling data is available.

Classification and clustering can be applied on different kinds of processes, and give good results 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, but it has drawbacks when it comes to inter-process communication, especially pipes, because processes that are communicating using pipes are allocated to a different processor core.

A fair scheduling of clustered process groups achieves a similar result to classification, while it has the 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 fix this.

Interactivity benchmark achieves the same result for normal clustering and classification with a small difference in seconds.

5.2 Future Works

The following items can be improved in the future:

1- Implement the new method in operating system schedulers, and use data available to schedulers as a parameter to process distribution.
2- Assign inter-communicating processes to the same processor, reducing the communication overhead.
3- Find issues which make process creation and system call overhead high while using classification and clustering.

References

Bovet, D. P., & Cesatí, M. (2005). *Understanding The Linux Kernel.* O'Reilly Media.

Bradford, E., & Mauget, L. (2002). *Linux and Windows Interoperability Guide.* Prentice Hall Professional.

byte-unixbench. (2012, 5 8). *byte-unixbench*. Retrieved 5 8, 2012, from code.google.com: http://code.google.com/p/byte-unixbench/

Cheney, S. (1998). *Benchmarking.* ASTD .

Choi, L. (2007). *Advances in computer systems architecture.* Seoul, Korea: Choi, Lynn; Paek, Yunheung; Cho, Sangyeun.

Crowley, C. P. (1997). *Operating systems: a design-oriented approach.* Irwin.

Dougherty, D., & Robbins, A. (1997). *Sed & awk.* O'Reilly Media.

Douglas, S., & Douglas, K. (2004). *Linux Timesaving Techniques For Dummies.* Wily.

Foster, J. C. (2005). *Buffer Overflow Attacks: Detect, Exploit, Prevent.* Syngress.

Fusco, J. (2007). *The Linux Programmer's Toolbox.* Prentice Hall.

Godbole, A. S. (2005). *Operating Systems.* Tata McGraw-Hill Education.

H, R., Arpaci-Dusseau, & C., A. (2011). *Operating Systems: Four Easy Pieces.* Remzi H; Arpaci-Dusseau ; Andrea C.

Hailperin, M. (2007). *Operating Systems And Middleware: Supporting Controlled Interaction.* Max Hailperin.

Haldar, S., & Aravind, A. A. (2010). *Operating systems.* Pearson Hall.

IPPS. (n.d.). Symposium on parallel and distributed processing.

Janert, P. K. (2010). *Data Analysis with Open Source Tools.* O'Reilly Media.

Jepson, B., & Rothman, E. E. (2005). *Mac OS X Tiger For Unix Geeks.* O'Reilly Media.

Jepson, B., & Rothman, E. E. (2005). *Mac OS X Tiger For Unix Geeks.* O'Reilly Media.

Jim Mauro, R. M. (2001). *Solaris Internals core kernel architecture.* Prentice Hall.
Levine, J. (2009). *Flex & Bison.* O'Reilly.

Lin, T. Y., Xie, Y., & Wasilewska, A. (2008). *Data mining: foundations and practice.* Springer.

Linux® Kernel Primer, T. A.-D. (2005). *Claudia Salzberg Rodriguez; Gordon Fischer; Steven Smolski.* Prentice Hall .

Love, R. (2007). *Linux system programming.* O'Reilly Media.

Masko, L., Dutot, P.–F., Mounie, G., Trystram, D., & Tudruj, M. (2006). Scheduling Moldable Tasks for Dynamic SMP Clusters in SoC Technology. Retrieved from hal.inria.fr/docs/00/05/74/09/PDF/mdmtt_ppam05.pdf.

Mauro, J., & Gregg, B. (2011). *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD.* Prentice Hall.

McKusick, M. K., & Neville-Neil, G. V. (2004). *The Design and Implementation of the FreeBSD Operating System.* Addison-Wesley Professional .

Montana, D., Brinn, M., Moore, S., & Bidwell, G. (2001). Genetic Algorithms for Complex, Real-Time Scheduling.

Mulvihill, D., & Grobman, I. (2012). Multiprocessor Scheduling Using Dynamic Performance Measurement and Analysis. Madison: University of Wisconsin. Retrived from http://pages.cs.wisc.edu/~mulvihil/736_paper.pdf.

Piel, E.´., Marquet, P., Soula, J., & Dekeyser, J.-L. (2007). Asymmetric Scheduling and Load Balancing for Real-Time on Linux SMP. Retrived from http://pieleric.free.fr/PMSD05.pdf.

Primate labs. (2012 йил 8-5). *Geekbench*. Retrieved 2012 йил 8-5 from Primate labs: http://www.primatelabs.ca/geekbench/

Robbins, A. (1999). *Larger Cover UNIX in a Nutshell, 3rd Edition.* O'Reilly Media.
Roberson, J. (n.d.). ULE: A Modern Scheduler For FreeBSD. San Mateo .

Rodriguez, C. S., Fischer, G., & Smolski, S. (2005). *The Linux kernel primer: a top-down approach for x86 and PowerPC architectures.* Prentice Hall Professional.

Rokach, L., & Maimon, O. Z. (2008). *Data mining with decision trees: theroy and applications.* World scientific.

Tanenbaum, A. S. (2008). *Modern operating systems.* Pearson Prentice Hall.

Tanenbaum, A. S., & Woodhull, A. S. (2009). *Operating Systems Design and Implementation.* Pearson Prentice-Hall.

Tchernykh, A., & Trystram, D. (2003). On-line scheduling of multiprocessor jobs with idle regulation. Retrieved from http://usuario.cicese.mx/~chernykh/papers/PPAM2003_idle_regulation.pdf

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques.* Elsevier.