Self-Tuning of Disk Input-Output in Operating Systems

A. Santos[a], J.J. Romero[a], J. Taibo[b], C. Rodriguez[c]

University of A Coruña, A Coruña. SPAIN

[a] *Artificial Neural Networks and Adaptive Systems LAB*
[b] *VideaLAB*
[c] *Computing and Communications Service*

Abstract

One of the most difficult and hard to learn tasks in computer system management is tuning the kernel parameters in order to get the maximum performance. Traditionally, this tuning has been set using either fixed configurations or the subjective administrator's criteria. The main bottleneck among the subsystems managed by the operating systems is disk Input/Output (I/O). An evolutionary module has been developed to perform the tuning of this subsystem automatically, using an adaptive and dynamic approach. Any computer change, both at the hardware level, and due to the nature of the workload itself, will make our module adapt automatically and in a transparent way. Thus, system administrators are released from this kind of task and able to achieve some optimal performances adapted to the framework of each of their systems. The experiment made shows a productivity increase in 88.2% of cases and an average improvement of 29.63% with regard to the default configuration of the Linux operating system. A decrease of the average latency was achieved in 77.5% of cases and the mean decrease in the request processing time of I/O was 12.79%.

Keywords: operating system, genetic algorithms, evolutionary computation, IO optimization, kernel optimization

1. Introduction

Computer system performance depends on three main factors: hardware, operating system and applications. The system administrator cannot usually modify applications, so if he needs to increase the global system performance he will have to improve some of the other two factors. Purchasing new

hardware is generally expensive and sometimes unnecessary, because a smart operating system tuning may often achieve a satisfactory increase in the system performance. Therefore, this option should be the first one to be considered by a responsible administrator, since it is feasible and does not require any additional investment.

Tuning a system means making the most efficient use of the available resources according to the workload supported. On the one hand, unnecessary tasks must be avoided and on the other hand, all the available options must be set for an optimal performance [16]. The general way in which an operating system configuration is tuned consists of making a performance measurement in the different system modules in order to spot the system's bottleneck, that is, the point in which performance is limited. This limitation is usually caused by a resource demand greater than its availability. Once the bottleneck is located, it must be eliminated, either by increasing the availability of that resource or by reducing the demand. This process continues until it reaches a satisfactory performance or else, a deadlock.

There are several subsystems that can be tuned within an operating system: process management and Central Processing Unit (CPU) scheduling, system caches and memory, buses and Input/Output (I/O) devices, file system and network. There are specific performance measurement tools for every subsystem (i.e. vmstat for virtual memory statistics), in addition to some of general use, such as the SAR utility (System Activity Reporter) from UNIX systems. Besides, most operating system manufacturers provide software packages to ease the collection and analysis of performance data, showing them in graphical format and including Graphical User Interfaces (GUIs); for example SE Toolkit, by Solaris, that has been released under General Public License (GPL), or Windows Performance Analyzer by Microsoft [15]. Solaris has also developed a language named SymbEL [24] devised to simplify the access to the data Kernel, both performance statistics and configuration parameters.

Although all these tools make tuning easier, success in this task still depends on the personal administrator's skills. There are several studies about automatic methods for specific parameter tuning, such as the Transmission Control Protocol (TCP) buffer size [23, 3, 20], or Data Base Management System (DBMS) optimization [21]. The development of new tools that automate the whole tuning process in a dynamic and adaptive way that depends on time and architecture will provide our operating systems with a certain degree of intelligence.

Standard tuning techniques are not able to obtain the maximum performance from the system, since they are based almost exclusively on the administrator's experience and they are hard to adapt to the different software and hardware configurations of each system. They do not take into account that there are some factors that vary through time, such as the workload.

We propose an automatic intelligent tuning module for system optimization. Due to the great amount of subsystems involved in tuning, it would be desirable to initially treat them separately. We have chosen the disk I/O subsystem because it is usually the main bottleneck in our systems.

The paper has been structured as follows: section 2 describes the disk I/O subsystem; section 3 includes a sensibility study of the Linux I/O disk management subsystem. The parameters which can be modified together with their impact on the system performance are analyzed. The Linux mechanisms for subsystem monitoring are included, as well as some of the tools available. Section 4 introduces the approach put forward for subsystem optimization, while section 5 tackles the implementation of our IOPerf self-tuning module for the disk I/O subsystem and analyzes the elements managing the functioning of the Genetic Algorithm (GA). Finally, sections 6 and 7 present those aspects related to the experiment and the results achieved, as well as the work conclusions.

2. Disk I/O System Description

The hard disk is the main non-volatile information storage element in an information system. It plays a key role in our computer architectures [7] and this may directly impact the system's global performance. The development of storage devices has constantly aimed at improving performance. One of the latest innovations enhancing accountability and performance in recent years were the disk arrays.

The relevance of the performance of the disk I/O subsystem is determined to a great extent by the way in which the system is used, the applications running, as well as their hard-disk usage. Hard disks and their controllers can be compared by checking the numerous technical specifications provided by their manufacturers. Knowing and analyzing hard-disk specifications is vital in order to understand its performance and to be able to evaluate and optimize it. Among them, we should highlight those related to positioning and transfer, as well as factors depending on memory management, the file

systems or the disk interface [27]. Therefore, the performance of the disk I/O subsystem is ruled by a variety of factors. Some of them are technological; others depend on the operating system and its management strategies.

Quantifying the performance of the disk I/O subsystem is a complex task. There are several parameters in order to evaluate whether the storage system function at the level required by the remaining system. One of our goals could be the optimization of the whole set of parameters, bearing in mind the I/O subsystem specifications (positioning, transfer, memory management, files system, disk interface, etc.). Nevertheless, some of these attributes are contradictory depending on the requirement type of the applications running in the system. Determining and focusing on those attributes which make a greater impact on performance is of essence. The following parameters must be highlighted among those allowing the measurement of the disk I/O:

- Throughput: the amount of work completed within a period of time. It can be measured according to the number of data which can be moved through the system in a given time, or according to the I/O operations completed per time unit. The context will determine which one is the fittest. For instance, in a system oriented towards the transfer of big files, the most appropriate thing will be measuring the number of transferred bytes. In case the system performs a great number of small independent access operations, then the number of operations per time unit will be more relevant.

- Response or latency time: total amount of time needed in order to complete a particular task. In the I/O subsystem, it reflects the time of a specific request. In environments with extremely big I/O requests, the response time will basically depend on the transfer speeds. In other contexts, with many small access operations, it will be marked by the requests management and their access times.

- Fairness: ability to process the tasks-requests uniformly.

The throughput and the response time are two hugely important metrics in a disk I/O system. Usually, some compromises are reached between them. For instance, the response time is minimized, processing the request as soon as possible, while productivity may be increased if those requests accessing near positions are grouped together. In the latter case, the response time would increase, since they must wait for a longer time.

I/O schedulers will usually try to maximize productivity. With that goal, they will re-organize requests by keeping in wait those which have not been served for the longest time, and they process new requests soliciting disk blocks that minimize the access time. This system enhances the system performance by sacrificing the fairness of the disk requests set. Current I/O schedulers do take this circumstance into account and they usually assign life times to requests so as to avoid their eventual inanition. In order to maintain a good system performance, it is basic to keep a certain fairness.

The huge variety of existing hardware components has helped operating systems to evolve from monolithic environments to module-based models. The purpose is to obtain a kernel that is as light as possible. New functions can be added thanks to the loading of new modules. Linux operating systems are a clear example of that evolution.

The Asynchronous I/O (AIO) request support is one of the most interesting aspects integrated in the 2.6 kernel. Thus, the issuing of multiple I/O requests is enabled with a single system call, as well as the ability to issue one request by a given process without waiting for its completion. As a result, the I/O operations of the application's main loop become independent, which optimizes CPU time and increases productivity, since extra processes are eliminated and the number of context switches is reduced [1].

The Linux operating system facilitates information export from the kernel space to the user one, as well as the other way round. The file system /proc (ProcFS) constitutes a system information point for the user's space. It makes it possible to modify dynamically the determining parameters in the management of the various modules in the operating system, simply by the administrator's writing in the files [1]. Thus, the disk I/O system can be tuned by a set of parameters, which can be modified by the system's administrator or by a dynamic and adaptive automatic tuning module.

3. Linux I/O Disk Management Subsystem

Before developing the automatic tuning system, we performed a sensibility study of the Linux I/O disk management subsystem and the parameters that can be modified. The study was divided into three of its elements: the disk I/O schedulers, the virtual memory management and the file systems. The study of the adjustable parameters, as well as the associated configuration files, will provide other researchers with the basic technical information

in order to develop new works in the field. This section is completed with a description of some of the subsystem monitoring tools.

The first element analyzed is the disk I/O scheduler. In Linux, the I/O scheduler generates the interface between the generic block layer (BIO layer) and the device driver [12, 11]. The BIO layer integrates a series of functions used by the file system and the virtual memory manager in order to issue the I/O requests of the block devices. The I/O scheduler transforms these requests so that they can be processed by the driver. It collects the requests and sends them to the device hardware controller for their processing.

Several lacks were detected in the single disk scheduler of the 2.5 kernels (Elevator or SCAN [26]). This algorithm could sometimes cause the inanition of disk requests for a long period of time. The 2.6 kernels integrate some new I/O schedulers so as to avoid that problem. Users have the chance to select among a variety of them: Anticipatory (AS), Deadline, Noop and Completely Fair Queuing (CFQ) [11]. Choosing the ideal one will depend on each system's scenario.

The Deadline scheduler operates with 5 I/O queues and it links a maximum life time to each request. It re-organizes requests with the goal of improving the I/O performance, always making sure that none of the requests undergoes inanition. The nature of the Deadline [12, 11] queue and request management focuses on minimizing the average response time of reading operations, thus damaging disk productivity and mean response time to the total number of requests. Its goal is the attention of reading requests within a given time, while writing requests have no associated life time. This scheduler is oriented to servers trying to minimize the waiting time of a request.

The AS scheduler aims at decreasing the reading response time for each thread. A controlled delay is included in the equation determining the next I/O request to be assisted [9, 17]. Once each I/O reading request is completed, the scheduler starts a short waiting time allowing the thread which made the last access to the disk to issue a new reading request that can be immediately assisted. Thus, positioning times between requests are shortened, while the spatial location between disk access operations is the aim. This scheduler is oriented to applications that quickly generate another I/O request which could be served before the scheduler selects another task, thus avoiding the deceptive idleness [9]. The fact that the disk wastes that period of time does not necessarily entail a decrease in I/O performance. The balance between the decrease in positioning time and disk productivity is managed according to a cost-benefit analysis. The heuristics used in this

analysis uses mainly estimates of the positioning and access times.

The CFQ scheduler may be considered as an improved extension of Stochastic Fair Queuing (SFQ) [14, 25]. Both schedulers are based on the concept of a fair distribution of the I/O bandwidth for every process making access to the disk. While the SFQ uses a fixed number of queues for the I/O requests, normally 64, the CFQ uses as many queues as existing I/O processes. The current CFQ Linux implementations use a fixed number of queues, similarly to the SFQ, though modifying the queue hashing function in order to avoid the serious issue of SFQ collisions. This scheduler focuses on maintaining the process fairness and it provides a good performance in those systems requiring a low latency and demanding a high productivity.

The Noop scheduler, integrated in Linux 2.6, uses a very simple algorithm. It serves the next request without ordering the requests at all. Its main application field is found in those devices not based on blocks, as well as the specialized software-hardware integrating its own I/O policy, and it requires minimum kernel participation [25]. This scheduler may yield good results in big I/O subsystems with RAID controllers.

One of the parameters to be highlighted among those related to I/O requests processing is the one regarding the selection of the scheduler. The file `/sys/block/<device>/queue/scheduler` provides information about the scheduler used in a given device. One scheduler may be selected for each disk from kernel 2.6.10 on, and it can be replaced at any time. One of the key tasks consists of choosing the fittest scheduler for each work environment. Linux allows tuning the functioning of its schedulers by adjusting some of the parameters involved. This is a practical function in those environments where the disk access profile does not vary greatly through time. Tuning some of the parameters involved may enhance performance without requiring a scheduler replacement, with the associated costs.

The configuration files including the parameters analyzed are grouped in the `/sys/block/<device>/queue` directory. Here are the modifiable parameters involved in the schedulers:

- Inside the deadline scheduler:
  - read_expire: maximum life time of every reading request. The lower its value, the sooner the requests will be assisted, at the expense of decreasing productivity, given that the trend towards processing "consecutive" reading requests diminishes. When its

value increases, then the scheduler may organize reading requests better.

  - write_expire: maximum life time of every writing request.
  - fifo_batch: the number of requests in each batch sent for immediate servicing once they have expired. Increasing their value enhances productivity, at the expense of decreasing the response time of other requests.
  - front_merges: requests next to the existing request in the scheduler are merged to the front of the queue, instead of the back. This is an interesting option in case of sequential access.

- Inside the AS scheduler:

  - read_expire: same as in deadline.
  - write_expire: same as in deadline.
  - read_batch_expire: the time spent servicing reading requests before servicing pending writing requests. If its value increases (the default value being 500 msg), then the priority assigned to reading requests will increase.
  - write_batch_expire: it is equivalent to read_batch_expire but with regard to writing requests.
  - antic_expire: the time in milliseconds that the scheduler pauses while waiting for a follow-on request from an application before servicing the next request in the queue. If 0 value is assigned to it, then the anticipation mechanism will remain off and the functioning of the AS scheduler will be equivalent to the deadline.

- Inside the CFQ scheduler:

  - fifo_batch_expire, fifo_batch_async, fifo_batch_sync: set of parameters defining the treatment of the FIFO queue. fifo_batch_expire specifies the time elapsed until request selection is allowed to be processed from a request queue. fifo_batch_async is the life time of asynchronous requests. fifo_batch_sync determines the life time of synchronous requests.

  - key_type: the key to be used in order to link the inputs of hash tables. There are 4 possibilities (pgid, process group identifier; tgid, thread group identifier; uid, user identifier; gid, group identifier). Selecting the key type will impact the number of inputs in the hash table used by the scheduler.
  - quantum: the number of internal queues from which the requests are taken in one cycle and moved to the dispatch queue for processing.
  - queued: the maximum number of requests allowed in a given queue.

- no parameters can be tuned inside the noop scheduler due to its simpleness.

- nr_request: size of each disk requests queue.

- read_ahead_kb: amount of data requested to the driver for each block required in a reading request. The data found in the disk are loaded in the memory subsequently to those actually requested. The performance of sequential reading of big files is enhanced. In those systems with a majority of random access, a small read_ahead_kb value will usually yield better results.

- max_sectors_kb: maximum size of every I/O request. An increase in this parameter entails an improvement in disk productivity, at the expense of an increase in average latency, particularly in environments with a majority of sequential access.

Another relevant issue which is directly linked to our system's performance is the virtual memory management. Linux uses the demand paging technique which only loads the pages needed for processes in the system's physical memory. In order to replace pages, Linux uses a LRU algorithm (Least Recently Used). For performance reasons, the kernel memory cannot be paged. Given that the kernel may use a variable amount of memory, it will be necessary to balance the memory in order to determine the amount to be used by the kernel as well as the amount reserved for the remaining processes. For that purpose, Linux incorporates a set of heuristics in the memory balancing process. The kswapd daemon is in charge of this task [6].

The directory `/proc/sys/vm` holds the files storing the values of the adjustable parameters regarding the management of the virtual memory. The following ones can be highlighted among them: dirty_background_ratio, dirty_ratio, dirty_writeback_centisecs, dirty_expire_centisecs, min_free_kbytes, nr_hugepages, overcommit_memory, overcommit_ratio, page_cluster, swappines and vfs_cache_pressure.

The swappines parameter controls the system's trend towards using the swap memory. Its value may range between 0 (no swap is used) and 100 (swapping whenever possible). In case of intermediate values, the option executed will depend on certain factors, such as the memory occupied at each time. Athorough review of each of these parameters can be found in the Linux kernel version 2.6.29 documentation [22].

The third element, the file systems, defines the structures used at top level in order to organize data in disks. Their selection may cause a tangible impact on performance. Linux supports a great amount of file systems. In order to guarantee the system modularity, it implements an upper abstraction layer called Virtual File System (VFS). This layer defines a general file model capable of representing every file system supported.

The directory `/proc/sys/fs` contains all sort of information regarding the file systems mounted in the system. The following parameters may be highlighted among all of them:

- aio_max_nr: maximum number of asynchronous I/O requests that can be supported by the kernel at any given time. The current number of asynchronous requests issued and pending execution is updated in the file `/proc/sys/fs/aio-nr`.

- file_max: maximum number of file handles that the linux kernel will allocate. It specifies an upper limit for the number of files open at the same time. Its parameter file is `/proc/sys/fs/file_max`. The three values in `/proc/sys/fs/file-nr` denotes the number of allocated file handles, the number of allocated but unused file handles, and the maximun number of file handles at any given time.

Operating systems usually perform an acceptable resource management, though it is advisable to provide them with adaptation facilities. Tuning tools are needed because of the lack of information about the disk traffic required in the future or the performance expected by users. The Linux

kernel 2.6 allows the tuning of a great amount of parameters related to its various modules, including hard disks management. The fact of being able to make changes in the operating system status clearly favors computer tuning.

Modifying any of the subsystem parameters will cause a certain variation in the I/O performance. Evaluating performance requires a monitoring process. As of kernel version 2.4.20, Linux has introduced a disk statistics system in order to help measure its activity. Tools such as iostat or sar interpret that information which is also directly accessible by means of any proprietary development. In kernel 2.6, that information is located in the file `/proc/diskstats` and in the sysfs file system which is generally mounted in the `/sys`. These files include one input per each file system with 11 values, some of which are the number of reading and writing requests completed, the amount of data read and written in the disk, the latencies of each access type, as well as the number of requests currently in progress. All these fields are cumulative from the machine init, except for the last one [10].

The kernel stores the statistics related to memory management and its pages in the file `/proc/meminfo`, as well as the swap space management. This information allows the determination of the RAM and swap memory in use, the amount of dirty memory and the data related to the overcommit strategy. The information available in the `/proc/meminfo` [18] is detailed in the kernel documentation.

There are various freeware tools in order to measure the I/O disk performance. These benchmarks are based on the creation of files in the hard disks and the subsequent generation of requests to them, trying to overload disk access. They will later analyze the performance achieved and they will yield results such as the productivity or mean latency. Tools such as Iozone [19] allow different measurements, such as productivity or latency, of disk I/O workloads. The benchmark generates and measures a variety of file operations. Tiobench is another available benchmark measuring productivity (reading and writing) and mean latency (reading and writing) of a set of sequential reading/writing operations, random reading/writing and re-read [13]. We should also highlight other benchmarks such as postmark, lmbench or bonnie++.

4. Approach to the performance tuning of the disk I/O system

The performance tuning of the disk I/O system must be tackled from the point of view of the resources control. Using a proactive method through

a control module carrying out preventive actions will avoid eventual negative states in disk I/O. Identifying some significant and sufficient indicators leading to believe that those states will arise makes our solution execute a series of actions producing a more positive state. The goal is minimizing the chance of a non-desirable situation in the I/O subsystem, instead of waiting for the worst-case scenario in order to act up.

In the face of such an issue, we should be able to predict from time to time the approximate workload of our disks in the near future. Thus, we may determine the conditions under which the I/O system will operate. Tuning the I/O system working conditions entails a clear optimization problem.

Heuristic techniques are capable of finding solutions closet to the optimal one. They also provide an appropriate performance and reduce the high costs associated to finding the best solution. Optimization techniques cannot simply be evaluated by their capacity to locate the optimal solution, but also according to their cost. GAs are a very competitive alternative in terms of solution quality compared to cost. We used GAs due to their adequacy for solving the problem to tackle [2, 5, 8].

The approach that we have developed obtains a population of individuals (chromosomes) characterized by the parameters defining the various patterns of the disk I/O subsystem. The initial population is integrated by a set of randomly generated disk access prototypes. Benchmarking the individuals in the population will yield a whole set that is characteristic of the performance results of the I/O subsystem. Evaluating the results achieved and applying the crossover and mutation operators will generate new chromosomes for the population that will be treated in the search for new solutions.

One of the most important decisions is the chromosome coding criterion. Building chromosomes is made by uniting the genes representing the characteristic parameters of disk I/O access, both those of the scheduler and those of the virtual memory management and file systems. The algorithm's optimization capabilities are closely linked to data representation. Besides, convergence time is strongly influenced by the representation.

Based on these reflections, two types of parameters are chosen to be integrated in the chromosomes: those characterizing the nature and status of disk access, together with those integrating the solution itself. The first type of parameters can be deemed as key in order to determine the nature of disk access. Assigning clearance factors to these parameters will determine whether a new disk access pattern is significantly similar to that solution. The second group of parameters will determine the new solution values-in

case the solution is applicable which will define the behavior-tuning of the disk I/O subsystem.

Determining the number of parameters of the disk I/O pattern raises several questions. Considering a great number of parameters may seem attractive, since one can control all of the I/O system variations. This scenario would create greater population variability and an increase in convergence times. Moreover, it would force one to increase the sampling frequency in order to achieve a bigger chromosomes population that would guarantee reaching some valid solutions. Characterizing disk access by a huge set of parameters generates a very wide space of solutions, which in turn renders finding a solution more complex. We have decided to select a smaller set of significant parameters, although we have risked not capturing all the dynamism and complexity of disk I/O system.

The parameters characterizing the disk access patterns must reflect the nature of the existing requests. Among all the possible parameters, we have chosen the ones that represent those elements which may generate a greater variation in disk access performance:

- Process number: number of processes that issue requests simultaneously to the disk being monitored. The clearance factor for a huge number of processes should not be small, given that the workload varies less according to the increase or decrease in the number of processes. As the number of processes decreases, the influence will be greater in case the amount changes in one of them.

- Percentage of reading vs. writing access operations: it regards the chance for a reading request issued by a process. As readings are processed faster, the clearance factor of this parameter must be small.

- Percentage of random vs. sequential access operations: it regards the chance for the occurrence of a disk request of a process not being subsequent to the last one issued by it. Its clearance factor must be small, given that a minimum change in this parameter will entail a significant change in the number of movements of the disk reading and writing heads.

- Size of the disk requests: amount of data requested by each reading and writing request. In the case of big requests, access times increase and productivity is enhanced, given that the data requested is located

in adjacent positions. Wide clearance factors should be allowed, given that similar sized requests will not entail major variations in disk performance.

As regards clearance factors, small values will be assigned to those parameters in which a smaller variation entails a significant change in disk performance. Thus, their values should bear a closer resemblance to the solution.

The second group of parameters determines the new solution values that will define the tuning of the disk I/O subsystem.

One chromosome in our population is integrated by the four disk access parameters already selected, together with a set of parameters characterizing the management-behavior of the disk I/O subsystem. This second group of parameters must have a determining impact on performance, as well as a certain relation to the already selected parameters characterizing disk access. These parameters will determine the solution space and, as a consequence, the parameter selection or tuning of the disk I/O system allowing a certain enhancement of performance. We have chosen the following ones:

- Scheduler: the scheduler is in charge of managing every I/O disk request. Choosing the scheduler among the possible ones will be determined by the system's needs. A dynamic and correct scheduler selection will provide a considerable increase in performance.

- Read_ahead_kb: amount of spare data brought to memory at each reading request made to the disk. Those sectors which are consecutive to the requested ones are loaded. It establishes the assurance level that subsequent accesses will refer to neighboring disk positions. In those cases, information would be readily available in memory.

- Swappiness: it controls the tendency of the kernel to move processes out of physical memory onto the swap disk. Setting the right swappines value may avoid a high and unnecessary exchange of pages between memory and disk. It was decided that this parameter should be added due to the strong impact of virtual memory management in hard disk use, as well as its significant impact on the performance expected by our systems applications.

Another relevant point when selecting the parameter set is the way in which the performance achieved by each of our population's individuals is

evaluated. The tools Iozone [19] and tiobench [13] are used in order to generate an on-going disk workload and in order to analyze the performance achieved.

The process is as follows: for each of the chromosomes in an initial population, a workload is generated which is determined by those parameters characteristic of disk I/O requests (number of processes, percentage of reading vs. writing access operations, percentage of sequential vs. random access operations and disk request sizes). Previously, the kernel will have been modified with those parameters characterizing I/O disk management (scheduler, read_ahead_kb and swappiness). The performance achieved sets the evaluation function of each of the chromosomes in the population.

A performance measurement method is used oriented towards determining the amount of information transmitted per time unit, also with the purpose of minimizing the average access time of requests. For that purpose, the indicators used are provided by the benchmarks (Iozone and tiobench) selected for workload generation. These tools may determine the productivity obtained and the mean latency of the simulation, knowing first-hand the size of the data dealt with and the number of requests. An increase in the transfer rate associated to a reduction of the average access time clearly indicates an enhanced performance.

5. IOPerf.: Self-Tuning Module of Disk I/O

IOPerf is a module for tuning automatically and dynamically the parameters characterizing disk access and its requests. It aims at improving significantly the global performance of the disk I/O system. Its modular design facilitates extending its functions to later uses and enhancements. It was developed on a software architecture based on the following elements:

- A converter (EntityAccess) defining the codification of the chromosome structure into the integrating parameters and vice versa. It also allows the modification of kernel parameters. It will constitute the interface between the IOPerf and the kernel. If our module was integrated in another operating system, then the only change to be made would be related to these functions.

- A simulator (Simulator) that translates the chromosomes in a statement for the execution of selected benchmark. It is the only interface with the benchmarking tool used. It is also in charge of hiding to the

remaining application the mechanism for generating the disk workload. Its main goal is getting to know the information related to the benchmarks used and generating the appropriate statements from the key disk access parameters.

- A disk I/O workload generator (Monitor) which generates random I/O workloads in order to evaluate the chromosome populations. It generates random I/O workloads, it selects the solution chromosome and it evaluates the results.

- A disk I/O optimizer (Optimizer) using the GAUL library (Genetic Algorithm Utility Library) [4] in order to manage the chromosome populations and to obtain new generations. It also collects the simulation data in the disk access, using them as source for the GA, and thus obtaining some new solutions that can be applied to future disk access operations.

- A library (Tools) with implementation of several methods which are common to several application modules. It has access to a log file storing the sequence of all the events generated.

- IOPerf defines the algorithm flow indicating, among other things, the number of tests to be carried out for each generation or the time at which a new generation must be obtained. This module allows configuring and tuning several elements that have an impact on the GA's evolutionary mechanism.

Table 1: Chromosome coding.

Parameter	n^o of bits	Minimum	Maximum	Clearance factor
n^o of processes	3	1	8	0.8
%reading acess	7	0	100	1.0
%random access	7	0	100	1.0
Request size	11	0	2047	0.7
Scheduler	2	noop, AD, deadline, CFQ		-
read_ahead_kb	10	4	1023	-
swappiness	7	0	100	-

A binary coding has been used for the chromosomes in the population. Table 1 shows the number of bits assigned, the range of values they may take on and the clearance factor of the characteristic parameters.

The population size determines the maximum number of chromosomes that may integrate a stable population ready to be used as a basis for solutions. It must be big enough to allow a wide variety of solutions. Nevertheless, the greater the number of chromosomes is, the longer the computing time, which may render the technique inefficient.

5 ranges were defined for each of the following parameters: percentage of reading access and percentage of random access. We worked with 25 possible ranges and tried to maintain a minimum number of 2 individuals per range, therefore there will be a minimum of 50 chromosomes in the population. Every range is assigned a fixed length, except for the margins which have been reduced, given that a small variation in the parameter value will have a greater impact on disk I/O performance. The ranges selected for both parameters were: [0,4], [5,34], [35,64], [65,94], [95, 100].

The initial population is generated randomly, bearing in mind the specifications already mentioned in the previous paragraph. The evaluation function is applied to every chromosome from this population. This function determines the performance linked to each chromosome by means of the following equation:

$$\text{fitness} = (\text{total bytes transferred}/\text{mean latency}) * factor \tag{1}$$

where the factor is ruled by the following equation:

$$\text{factor} = (((\%\text{readings} + 1)/100)^2) * (((100 - \%\text{random} + 1)/100)^2) \tag{2}$$

Thus, within a small range of variations in the percentages of reading and random access operations, the influence of the adjustable parameters of the kernel will decide which is the solution providing the best performance.

Using ranges in the characteristic parameters and integrating the factor in the evaluation function will guarantee a certain diversity in the population. The performance obtained by each chromosome as the productivity achieved between the average access latency was initially used as evaluation function. Nevertheless, if we only consider both variables, this will lead us to one of the usual problems with GAs. For instance, an access pattern with 100% readings compared to one where writings are predominant shows a clear example of enhanced productivity and decrease of access time. In both scenarios, regardless of the disk access parameters, the evaluation function will always obtain the best results when readings are predominant. The same scenario is true with regard to the remaining characteristic parameters, such as the

percentage of random access operations and the number of processes. Thus, as new generations arise, the population will become more homogeneous regarding these parameters. As a result, the predominant chromosomes will have high percentages of reading and sequential access operations and a high number of processes. This scenario would not allow the population to cover the different possibilities for accessing the disk, and those patterns which are not represented approximately in the population would not achieve a good performance. This problem is solved by keeping a minimum number of individuals within the ranges that are characteristically defined as fostering diversity. Moreover, each chromosome's fitness is penalized according to a factor depending on the value of the characteristic parameters. As regards the number of processes, it has been considered that productivity should not be evaluated globally, but based on the average number of bytes transferred per process. Therefore, the fitness will be multiplied for the $1/n^o$ factor of processes. Selecting those chromosomes with the highest fitness while keeping a minimum for each of the existing ranges will guarantee population diversity.

A new population is generated from the initial one using crossover, mutation and selection operators. Crossover is made using the best chromosomes at a random point, while gene mutations occur to a small proportion of them. A random selection function was used for choosing those chromosomes to be reproduced. This pattern aims at increasing the chance for crossover between chromosomes that are as different as possible. A selection function fostering crossover between those chromosomes with highest fitness will probably make them similar between them and their offspring. The selection between chromosomes with the purpose of mutation is also made at random.

Another important factor consists of choosing the crossover and mutation ratios. The former specifies the percentage of chromosomes in the total population that will constitute the set of parents of the next generation, while the latter indicates the chance for a mutation taking place in a given chromosome. Generally speaking, mutation causes diversity in the population; therefore it is applied less frequently, while crossover is applied more often in order to foster the exchange of genetic material between chromosomes. Initially, a crossover ratio of 0.9 is used and a 0.1 ratio is applied to mutation. Since the population tends to converge generation after generation, the mutation factor has been gradually increased until reaching the value of 0.25, while the crossover ratio has been decreased until 0.7. As the population becomes homogeneous, a crossover between two chromosomes which are

genetically similar does not bring anything new. That is why mutation aims at exploring new solutions.

The best chromosomes resulting from the application of the evaluation function, which contemplates the criteria of ranges of characteristic parameters, will give rise to a new population. This new set of solutions will determine the possible tuning of the disk I/O system from then on. Compiling data on the disk access of the production computer goes on indefinitely. A new population is periodically created by uniting the chromosomes arising from the new piece of information with the already existing solutions. There are some solutions that will survive generation after generation, while others will die in order to open the path to new solutions adapting to the eventual changes in the system. Once a new population has been obtained, a benchmark checking its success or failure with different workloads randomly generated is used.

Our servers evolve quickly, both as regards workload and its nature. We could pinpoint some critical scenarios, such as the change of task in a system which moves on from acting as an e-mail server to providing web services, or as a database. There are several populations that will support our systems, evolving and adapting to them and to the disk I/O needs at any time. The IOPerf module has been designed in order to allow being used in any operating system. The only requirement is changing the means to access the operating system parameters in order to obtain or to modify their values.

6. Experimentation and results

The work environment in the experimentation phase is integrated by an Athlon XP2800 PC with 1 GB of RAM and an ST340014A hard disk of 40GB, 7200 rpm, 2MB of buffer-cache, internal transfer speed of 683 Mbps, external transfer speed of 100 MBps and a positioning time of 8.5 ms. The operating system used is a debian with a 2.6.29 kernel. The file system mounted is an ext3 and the swap space assigned in the disk is 1 GB. Tiobench-0.3.3 [13] and iozone3 [19] have been used in order to measure disk I/O performance.

There are two different stages in the experiment: optimization and monitoring. The crossover, mutation and evaluation operators are used during the optimization stage in the search for a new population. The simulation time for each chromosome's evaluation is 30 seconds. During that time, it is possible to carry out a sufficient number of disk requests in order to evaluate the corresponding pattern without generating a high computational cost.

During the monitoring stage, a simulation of randomly generated workloads is performed. Workloads are simulated and monitored continuously for 300 seconds until a decision is made to obtain a new population of solutions. This phase generates a new set of chromosomes which can be reproduced in the next optimization stage in the search for a new population. The IOPerf module analyzed in section 5 is responsible for alternating both phases.

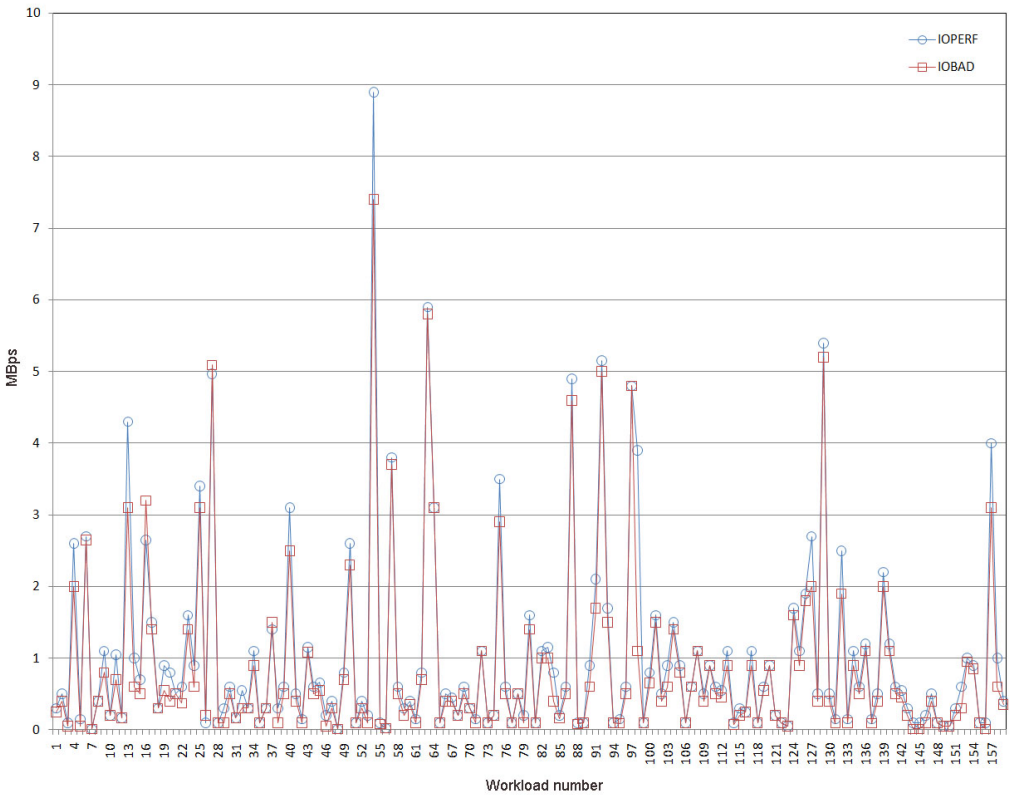

Figure 1: Mean productivity per process.

Each new population is tested with 1000 workloads of disk access in order to analyze the results. Each of them runs for 300 seconds and it is defined by the characteristic parameters (number of processes, percentage of reading access, percentage of random access and request sizes). Workloads are randomly generated. The same simulations are also made, although using the default values of the kernel parameters, as they are executed in most real work environments. An IOBad module was developed in order to read the

benchmark sentences executed at this stage which have been stored in a log file. This application issues the corresponding sentences without modifying the kernel parameters. Taking into account these test conditions, various analysis are made about our module's behavior. The graphs used only show the first 160 tests out of the 1000 that were made, with the purpose of enhancing comprehension and visual analysis.

The first result obtained is average performance for the 1000 cases, measuring the mean productivity per process and the mean access latency. The results achieved by the IOPerf module with an adaptive tuning in kernel are compared with IOBad module, without kernel tuning.

Figure 1 shows the productivity in MBps of the initial 160 cases. The IOperf graph represents productivity under the adjustments made by our module to the kernel. The IObad graph shows the results with the default kernel, without parameter tuning.

The other goal of our work consists of decreasing the response time of the I/O requests. Therefore, the purpose in this case is decreasing the mean latency. Figure 2 shows the average access time of the first 160 cases. Both the IOperf and the IObad graphs can be observed.

A significant performance difference is obtained in favor of the IOPerf module. Productivity increases at 88.2% of cases, with an average improvement of 29.63%. The goal of decreasing the mean latency was also achieved in 77.5% of the cases, with an average reduction in the I/O request processing time of 12.79%.

The goal of trying to jointly increase productivity per process and decreasing the request response time was achieved at 77.5%, with an average improvement of 26.1% in productivity and 12.79% in latency.

There are a minority of situations which do not yield the expected result. Productivity decreases while latency increases in 11.8% of the cases, which is exactly the opposite of the desired effect and entails an obvious loss of performance. In most cases, the assigned scheduler was not AS, the kernel's default one, and there is considerable variation among the various patterns. We should bear in mind that each scheduler's specific parameters are not being adjusted.

An intermediate situation is one in which productivity per process is enhanced at the expense of increasing the mean latency. This happens in 10.7% of cases and, although it is not the ideal scenario, it is possible that in those work environments where maximizing productivity is the main goal, it could be considered as a good solution. There is a common pattern regarding

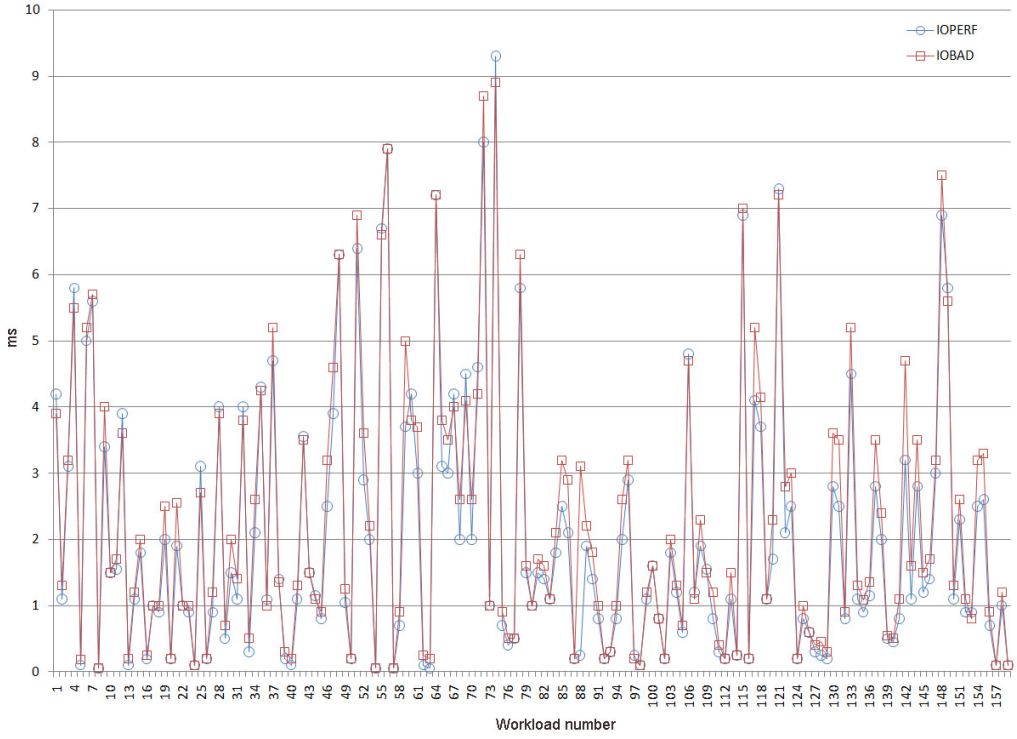

Figure 2: Mean latency per request.

these access operations, with a high number of processes linked to a higher chance for random access requests which causes a higher movement in the disk reading and writing head, thus increasing latency. Nevertheless, productivity may be enhanced, given that a greater number of processes will stop the disk from idling.

Another factor which has been analyzed is GA convergence towards optimizing disk access. Figure 3 shows the percentage of variation in average productivity per process achieved by the IOPerf module vs. IOBad. Figure 4 shows the percentage of variation in average response time per request. In both cases, a new generation is used each 20 workloads. Let us note that each randomly generated workload runs for 300 second. Thus, the evolution of the algorithm in searching for better solutions is checked. In the case of the average productivity, the goal is increasing it and it appears above 0% in the graph. In the case of the average latency, the goal is to decrease it, with

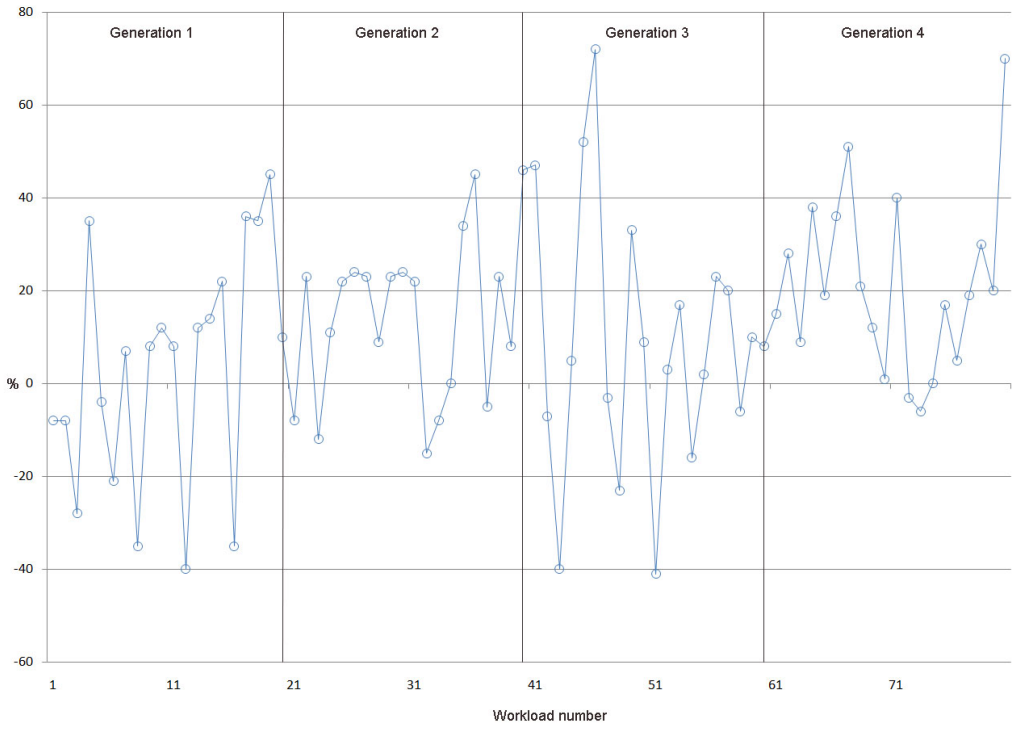

Figure 3: GA Evolution. Percentage of improvement of the mean productivity per process.

a percentage below 0%.

Regarding the results of the different disk access types, sequential readings, random readings, sequential writings and random writings, we should note that IOPerf generally manages to reduce the latency of writing operations, while the opposite is true as regards reading operations. This result should not be striking, given that the default value of the Linux read_ahead_kb parameter is relatively low, compared to the possibilities allowed by our module IOPerf. In order to foster an increase in productivity, IOPerf tends to assign a value to read_ahead_kb which is superior to the default one. Therefore, an increased latency is normal, given that the amount of information read by each reading request is higher.

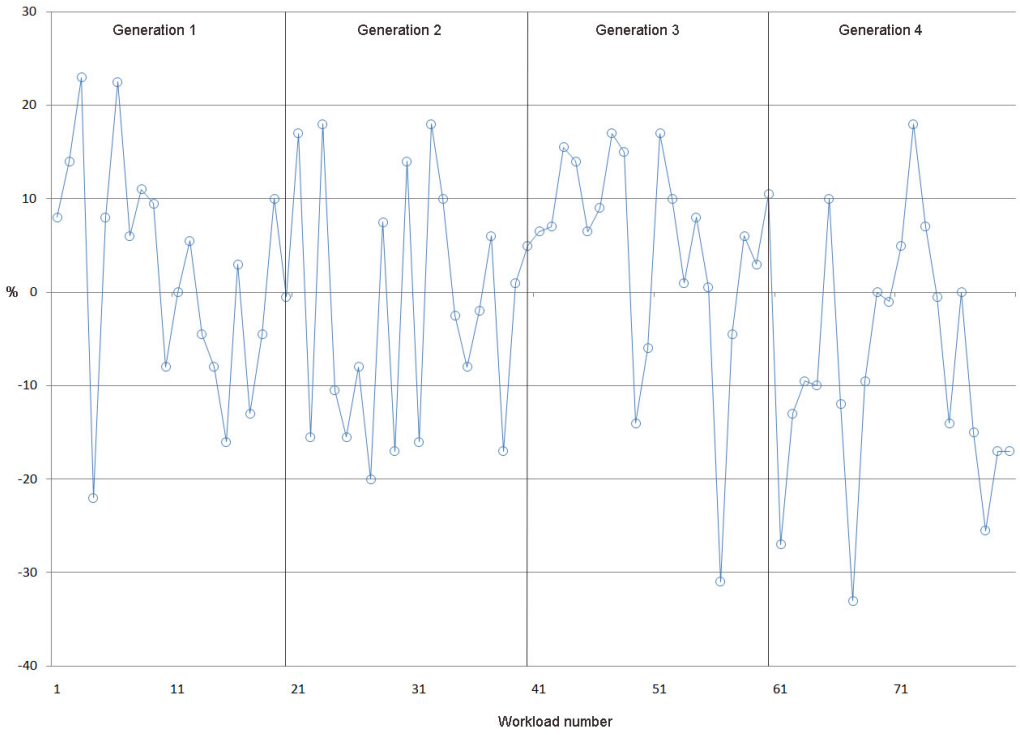

Figure 4: GA Evolution. Percentage of improvement of the mean latency per request.

7. Conclusions

One of the most complex tasks in the context of operating systems is getting an optimal performance of the kernel parameter tuning, because traditional techniques depend on the administrator's experience and skills. Our IOPerf module performs the automatic and dynamic tuning of these parameters, focusing on the disk I/O management subsystem. Adaptive system techniques have been applied to the development of the module, in particular GA.

The parameter sensibility study has contributed to determine which parameters are the most relevant in the global system performance, so that the most important ones are prioritized.

The final module (IOPerf) automates the tuning process, eliminating the subjectivity introduced by manual configurations. The experiments carried out show a considerable increase in the productivity of the disk I/O subsys-

tem. In 88.2% of cases, productivity increases, with an average enhancement of 29.63%. The average decrease in response time is 12.79% in 77.5% of cases. A prolongued runtime of the module could improve these results, obtaining some new generations.

In the final system there will be two working modes: a normal mode, which will use the machine as usual; and a learning mode, in which all workload is eliminated from the system and it is used just for the tuning process. In the normal mode, first of all, the workload is periodically evaluated, then the most similar chromosome in the optimal solution population is located and, finally, the kernel parameters are tuned according to the values of this chromosome. Besides, the system performance is evaluated from time to time, and so new chromosomes are generated (inerts) which will be added to the population when the working mode changes.

When the system is not being used (nights, weekends), it will change to learning mode, which consists of creating new generations and evaluating each new chromosome by simulating its workload. Thus, when the system changes back to normal mode, it will have not only a new and improved solution set, but also a set dinamically adapted to the eventual changes of the environment.

The final model will be able to perform an automatic, dynamic and real-time tuning of the I/O management subsystem in order to get the maximum system performance. The use of adaptive systems provides a dynamism that is not possible to achieve with other techniques.Thus, the model is capable of adapting dynamically and in real-time to the different configurations and workloads that could appear.

Tuning many of the kernel parameters of the operating systems requires a certain degree of experience. Integrating automatic tuning modules in operating systems will allow a substantial improvement in the performance of the various subsystems. Moreover, the adaptive nature of these modules will provide good solutions, regardless of the working environment architecture and its evolution, both at hardware and software levels.

References

[1] Bovet, D., Cesati, M., 2005. Understanding The Linux Kernel. O'reilly & Associates Inc.

[2] Davis, L.D., Mitchell, M., 1991. Handbook of Genetic Algorithms. Van Nostrand Reinhold .

[3] Fisk, M., chun Feng, W., 2000. Dynamic Adjustment of TCP Window Sizes. Technical Report. Los Alamos Unclassified Report (LAUR) 00-3221, Los Alamos National Laboratory.

[4] Gaul, 2009. Personal Home Page. `http://gaul.sourceforge.net`.

[5] Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[6] Hagen, W.V., 2002. Linux Filesystems. Sams, Indianapolis, IN, USA.

[7] Hennessy, J.L., Patterson, D.A., 2007. Computer Architecture, A Quantitative Approach. Elsevier.

[8] Holland, J.H., 1992. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA, USA.

[9] Iyer, S., Druschel, P., 2001. Anticipatory Scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous i/o, in: SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles, ACM, New York, NY, USA. pp. 117–130.

[10] Lind, R., 2003. Linux Kernel Documentation/iostats.txt. Accessible via `http://www.mjmwired.net/kernel/Documentation/iostats.txt`.

[11] Love, R., 2004. Kernel Korner: I/O Schedulers. Linux J. 2004, 10. Accessible via `http://www.linuxjournal.com/article/6931`.

[12] Love, R., 2005. Linux Kernel Development (2nd Edition) (Novell Press). Novell Press.

[13] Manning, Kuoppala, 2003. Tiobench Benchmark. Accessible via `http://sourceforge.net/projects/tiobench/`.

[14] McKenney, P., 1990. Stochastic Fairness Queueing.

[15] Microsoft, 2009. Windows Performance Analyzer. Accessible via `http://msdn.microsoft.com/en-us/performance/cc825801.aspx`.

[16] Musumeci, G.P.D., Loukides, M., 2002. System Performance Tuning, 2nd Edition (O'Reilly System Administration). O'Reilly Media, Inc.

[17] Nagar, S., Franke, H., Choi, J., Seetharaman, R., Kaplan, S., Singhvi, N., Kashyap, V., Kravetz, M., 2003. Class-Based Prioritized Resource Control in Linux, in: In Proc. 2003 Ottawa Linux Symposium, p. 03.

[18] Nerin, Feng, 2009. Linux Kernel Documentation/filesystems/meminfo. Accessible via `http://www.mjmwired.net/kernel/Documentation/filesystems/proc.txt`.

[19] Norcott, W.D., 2006. IOzone Filesystem Benchmark. Accessible via `http://www.iozone.org`.

[20] Oak, T.D., Dunigan, T., Mathis, M., Tierney, B., 2002. A TCP Tuning Daemon, in: in Proceedings of SuperComputing: High-Performance Networking and Computing.

[21] Oracle, 2008. Using Automatic Memory Management, Oracle Database Administrator's Guide. Accessible via `http://download.oracle.com/docs/cd/B28359\_01/server.111/b28310/memory003.htm`.

[22] Riel, R., Morreale, P., 2008. Virtual Memory, Linux kernel version 2.6.29 documentation. Accessible via `http://www.mjmwired.net/kernel/Documentation/sysctl/vm.txt`.

[23] Semke, J., Mahdavi, J., Mathis, M., 1998. Automatic TCP Buffer Tuning, in: SIGCOMM, pp. 315–323.

[24] Setoolkit, 2009. The SymbEL Language Reference Manual. Accessible via `http://www.setoolkit.org/cms/node/3`.

[25] Shakshober, D., 2005. Choosing an I/O Scheduler for Red Hat Enterprise Linux 4 and the 2.6 Kernel.

[26] Silberschatz, A., Galvin, P.B., Gagne, G., 2004. Operating System Concepts. John Wiley & Sons.

[27] Tanenbaum, A.S., 2007. Modern Operating Systems. Prentice Hall Press, Upper Saddle River, NJ, USA.

Antonino Santos del Riego received the B.S. degree in Computer Science from A Coruña University (Spain) in 1992 and the Ph.D. degree in Computer Science from A Coruña University in 1998. At present I am professor at University of A Coruña (Spain). Since 1991 I have worked with several research groups in Artificial Neural Networks, Genetic Algorithms and Internet servers and services. Dr. Santos has authored and edited more than 25 articles, 7 books, and participated as researcher in 12 funded research proposals concerning to Artificial Intelligence, Adaptive Systems and Internet Security.

Juan Romero received the B.S. degree in Computer Science from A Coruña University (Spain) in 1996 and the Ph.D. degree in Computer Science from A Coruña University in 2002. He is associate professor at University of A Coruña. He edited a "Natural Computing" Springer book, published 6 papers in international ISI journals and chaired 5 events published as Springer LNCS. He directed and participated in 10 European and Spanish research projects and research contracts with firms such Microsoft Spain.

Francisco Javier Taibo Pena received the B.S. degree in Computer Science from University of A Coruña (Spain) in 1998, and the Ph.D. degree in Computer Science in 2010. He is currently an assistant professor in this University. He has collaborated in several courses about Computer Science, System Administration, Multimedia, Computer Generated Imagery and Interaction. Since 1995 he has collaborated in several research projects. His research interests are oriented towards Computer Graphics, working in the Architecture, Engineering and Urbanism Visualization Group (VideaLAB) since 1998.

Carlos Rodríguez Díaz received the B.S. Degree in Computer Science from A Coruña University (Spain) in 2005. I am currently a software developer at Information Systems at University of A Coruña. I've been developing several applications using the lastest .NET technologies for 5 years and building a Java-based Content Management System for a significant printed Spanish media.