The Journal of Systems and Software 85 (2012) 77–86

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Self-tuning of disk input–output in operating systems

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

[a] Artificial Neural Networks and Adaptive Systems LAB, University of A Coruña, A Coruña, Spain
[b] VideaLAB, University of A Coruña, A Coruña, Spain
[c] Computing and Communications Service, University of A Coruña, A Coruña, Spain

ARTICLE INFO

*Article history:*
Received 18 October 2010
Received in revised form 23 May 2011
Accepted 13 July 2011
Available online 26 July 2011

*Keywords:*
Operating system
Genetic algorithms
Evolutionary computation
IO optimization
Kernel optimization

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%.

© 2011 Elsevier Inc. All rights reserved.

## 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 (Musumeci and Loukides, 2002). 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 (Microsoft, 2009). Solaris has also developed a language named SymbEL (Setoolkit, 2009) 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 (Semke et al., 1998; Fisk and chun Feng, 2000; Oak et al., 2002), or Data Base Management System (DBMS) optimization (Oracle, 2008). 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.

* Corresponding author.
*E-mail address:* jjrnasa@gmail.com (J. Romero).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.07.030

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 Linux operating system has been chosen for the experimentation tasks, due to the great number of existing platforms and its huge versatility. Implementing our environment in other operating system will only entail its adaptation to the appropriate adjustable parameters.

Our work is an attempt to set the foundations so that those corporations and bodies developing operating systems start considering the benefits posed by the inclusion of adaptive systems.

The paper has been structured as follows: Section 2 describes the disk I/O 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 3 presents the state of the art on automatic tuning of IO systems. Section 4 describes the proposed GA system, while Section 5 tackles the implementation of our IOPerf self-tuning module for the disk I/O subsystem. 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 (Hennessy and Patterson, 2007) 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 (Tanenbaum, 2007). 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 essential. 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 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 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 (Bovet and Cesati, 2005). 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.

We performed a 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 parameters involved in our system are presented next. A thorough description of the total parameters is available in Love (2005). This section is completed with a description of some of the subsystem monitoring tools.

The first element analyzed is the disk I/O scheduler. It collects the I/O requests and sends them to the device hardware controller for their processing.

The 2.6 kernels integrate four I/O schedulers: Anticipatory (AS), Deadline, Noop and Completely Fair Queuing (CFQ) (Love, 2004). Linux tends to use by default the CFQ. Users have the chance to select among them manually. A different one may be selected

for each disk, while the I/O scheduler can be changed on hot swapping. 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 (Love, 2005, 2004) 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 (Iyer and Druschel, 2001; Nagar et al., 2003). 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 (Iyer and Druschel, 2001). 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) (McKenney, 1990; Shakshober, 2005). 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. 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 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 (Shakshober, 2005). This scheduler may yield good results in big I/O subsystems with RAID controllers.

One of the parameters involved in the set of I/O schedulers is the item read ahead kb. It determines the 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.

Another relevant issue which is directly linked to our system's performance is the virtual memory management (Hagen, 2002; Lind, 2003). Among all the adjustable parameters the swappiness 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.

Modifying any of the subsystem parameters will cause a certain variation in the I/O performance. Evaluating performance requires a monitoring process. Linux has a disk statistics system in order to help measure its activity. Some of the information it provides consists of 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 (Lind, 2003).

In the case of virtual memory, the information provided includes the RAM and swap memory in use, the amount of dirty memory and the data related to the overcommit strategy (Nerin, 2009).

There are various freeware tools in order to measure the I/O disk performance. Tools such as "iostat" or "sar" interpret that information which is also directly accessible by means of any proprietary development. 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 (Norcott, 2006) allow different measurements, such as productivity or latency, of disk I/O workloads. The benchmark generates and measures a variety of file operations. Tiobench (Manning and Kuoppala, 2003) 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. We have used the libraries in the latter tool for our development.

## 3. State of the art

Several papers present new I/O Schedulers based on GAs (Moilanen and Williams, 2005), self-learning methods (Zhang and Bhargava, 2009), FIFO queue-based planners (Kim et al., 2009) or optimizations of already-existing planners in virtualized systems (Kesavan et al., 2010). There have also been approaches focusing on Solid State Disks (SSD) (Dunn and Reddy, 2009).

Moilanen and Williams (2005) took an approach using GAs for configuring the kernel. That work focuses on the Zaphod and the Anticipatory I/O Scheduler.

The general genotype contains: read_expire, write_expire, read_batch_expire, antic_expire, and max_thinktime y nr_request. They also use the so-called "placeholder for fitness measures" genes, with the aim of "making sure all workloads are considered, and not favoring one type of workload over another".

They perform a series of comparisons between their system for the Anticipatory I/O Scheduler and the default planners. They generate various workloads by means of a Flexible File System Benchmark Flexible (2008) (FFSB) and a system of ext3 files. They use an OpenPower 710, with 2 CPUs and 2 GB RAM. Tests are made with an SUSE Linux Enterprise Server 9 with a 2.6.11 kernel.

An 8.72% improvement was obtained in the Anticipatory Scheduler, taking into account the number of transactions-per-second and the throughput. The highest increase was observed in the random 256K block writing which achieved a 23.22% improvement. On the other hand, the sequential reading performance suffered a worsening of 0.74%.

Zhang and Bhargava (2009) propose universal self-learning disk I/O scheduling schemes for automating the configuration of a disk planner. Its main goal is adapting to different workloads so as to make optimal decisions at any time as regards planning. For that purpose, they take into account workloads, file systems, disk systems, tunable parameters, CPU systems, and user preferences. They possess four self-learning disk scheduling schemes: Two-layer Learning, Change-sensing Round-Robin, Feedback Learning and Per-request Learning. One of them, the Two-layer Learning Scheme, achieves good results as compared to the other four Linux default schedulers. "It integrates the workload-level and request-level learning algorithms. It employs five feedback learning techniques to analyze workloads, change scheduling policy,

and tune scheduling parameters automatically": C4.5 Decision tree algorithm, Logistic regression, Naive Bayes, ANN and SVM.

Experiments are carried out with a Pentium4 3.2 GHz with 1 GB RAM fitted with a Western Digital Caviar SE 250-Gbyte hard disk. They use Linux Kernel 2.6.13 and a Ext3 file system.

They generate several workload scenarios with Intel Iometer (Intel Iometer, 2008) under the heavy-loaded multithreaded workloads, the self-learning scheduler outperforms the second best scheduler, the Anticipatory scheduler, by 14.5%, considering the average response time. Under the "maximum throughput" workload, which is generated to measure the maximum possible throughput of the system, the self-learning scheduler outperforms the second best scheduler by 3.5%.

## 4. GA 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 (Davis and Mitchell, 1991; Goldberg, 1989; Holland, 1992).

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 individuals for the population that will be treated in the search for new solutions.

One of the most important decisions is the individual coding criterion. Building individuals 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.

Two types of parameters are chosen to be integrated in the individuals: (i) those characterizing the nature and status of disk access (characterizing the problem), and (ii) 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.

The second group of parameters will determine the new solution values provided there is a solution applicable which will define the tuning behavior of the disk I/O subsystem.

Another option that could be explored for optimizing the I/O access disk would entail using an independent GA for each status or problem nature. However, that option would not allow moving information from one status to the next. But our approach, being based on a single GA coding both the problem and the solution does allow this transfer. Thus, with a given solution for a particular status, this solution or an evolution of it could be useful to others. A similar approach was the one used by Moilanen and Williams (2005).

The first group of parameters will be assigned a clearance factor (CF) to determine whether a new disk access pattern is similar to that solution. Small values will be assigned to those parameters in which a smaller variation entails a significant change in disk performance.

This first group of parameters is used to select the individual in the population which is the most similar to a given situation. The following equation defines the distance between a given situation and an individual:

$$\text{Distance} = \sum_{i=0}^{\text{params}} \frac{\text{CF}_i \cdot |\text{Access Value}_i - \text{Individual Value}_i|}{\text{Maximum}(i) - \text{Minimum}(i+1)}$$

The clearance factor may range between 0 and 1. The greater the clearance factor, the greater the weight if that distance for calculating similitude.

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 individual's 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. 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.
- Percentage of reading vs. writing access operations: it regards the chance for a reading request issued by a process.
- 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.
- 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.

The second group of parameters defines the tuning of the disk I/O subsystem. These parameters will determine the solution space and the parameter selection or tuning of the disk I/O system. We have chosen the following ones:

- Scheduler: the scheduler is in charge of managing every I/O disk request. 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.

**Table 1**
Individual coding.

| Parameter | No. of bits | Minimum | Maximum | Clearance factor |
|---|---|---|---|---|
| No. of processes | 3 | 1 | 8 | 0.8 |
| %Reading access | 7 | 0 | 100 | 1.0 |
| %Random access | 7 | 0 | 100 | 1.0 |
| Request size | 11 | 0 | 2047 | 0.7 |
| Scheduler | 2 | | Noop, AS, deadline, CFQ | |
| Read_ahead_kb | 10 | 4 | 1023 | – |
| Swappiness | 7 | 0 | 100 | – |

- Swappiness: it controls the tendency of the kernel to move processes out of physical memory onto the swap disk. Setting the right swappiness 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.

A binary coding has been used for the individuals 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.

### 4.1. Population

The population size determines the maximum number of individuals 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 individuals is, the longer the computing time, which may render the technique inefficient.

The coding selected may lead us to an obvious problem of lack of diversity. 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 the percentage of random access operations. Thus, as new generations arise, the population will become more homogeneous regarding these parameters. As a result, the predominant individuals will have high percentages of reading and sequential access operations.

This problem is solved by keeping a minimum number of individuals within the ranges that are characteristically defined as fostering diversity. 5 ranges were defined for each of the following parameters: percentage of reading access and percentage of random access. 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].

We worked with 25 possible combinations of ranges and tried to maintain a minimum number of 2 individuals per range, therefore there will be a minimum of 50 individuals in the population. EA population size of 100 individuals has been used for the experiments described in the present paper.

The initial population is generated at random considering the already mentioned specifications.

The evaluation function is applied to every individual from this population. This function determines the performance linked to each individual by means of the following equation:

$$\text{fitness} = \frac{\text{total bytes transferred}}{\text{mean latency}} \cdot \frac{\text{factor}}{\text{number of process}}$$

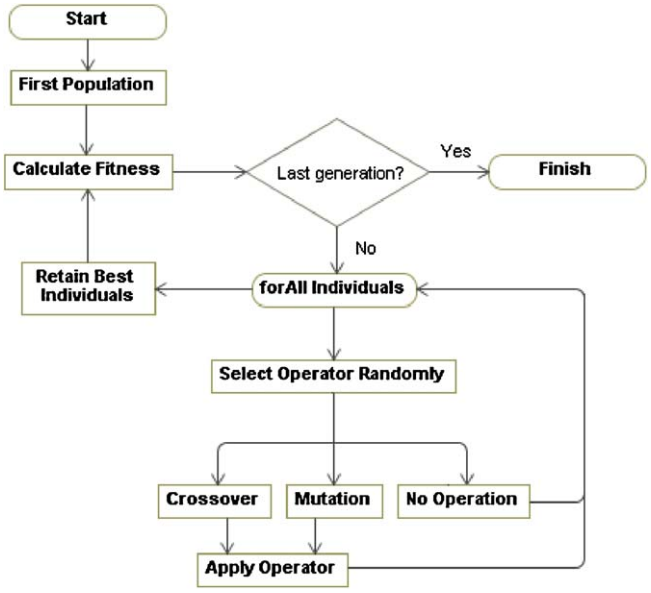


**Fig. 1.** Genetic algorithm.

where the factor is ruled by the following equation:

$$\text{factor} = \left(\frac{\%\text{readings} + 1}{100}\right)^2 \cdot \left(\frac{100 - (\%\text{random} + 1)}{100}\right)^2$$

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, as show in the equation, the fitness will be divided for the number of processes.

Moreover, each individual's fitness is penalized according to a factor depending on the value of the first group of parameters. Selecting those individuals 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. One-point crossover is used between randomly selected individuals. This pattern aims at increasing the chance for crossover between individuals that are as different as possible. Mutation consists of the deletion of one random bit. The selection between individuals with the purpose of mutation is also made at random (see Fig. 1).

These operators will add new individuals to the previous population. A selection operator is applied using the fitness to determine the best individuals to integrate the next population. This operator will guarantee the already explained diversity criteria.

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. An increase/decrease of 0.02 has been used in both cases.

The former specifies the percentage of individuals 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 individuals. 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 individuals.

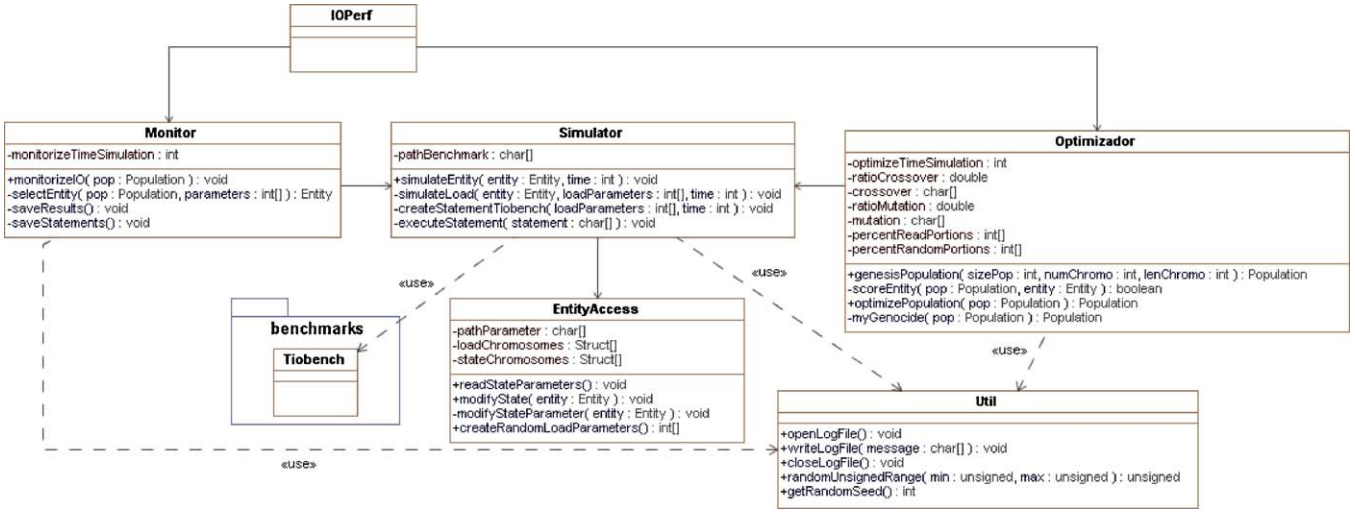


**Fig. 2.** IOPerf architecture.

## 5. IOPerf: self-tuning module of disk I/O

The present section analyzes the software architecture of the developed module, which is called IOPerf. Its modular design facilitates extending its functions to later uses and enhancements. Fig. 2 shows the architecture design.

The modules are described next:

- A converter (Entity Access) defining the codification of the individual 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 individuals in a statement for the execution of selected benchmark. It is the only interface with the benchmarking tool used. In the present case, Tiobench (Manning and Kuoppala, 2003). It is also in charge of hiding to the remaining application the mechanism for generating the disk workload.
- A disk I/O workload generator (Monitor) which generates random I/O workloads in order to evaluate the individual populations. It generates random I/O workloads, it selects the solution individual and it evaluates the results.
- A disk I/O optimizer (Optimizer) using the GAUL library (Genetic Algorithm Utility Library) (Gaul, 2009) to manage the individual populations and to obtain new generations integrates the GA core.
- 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.

The process is as follows: for each of the individuals in an initial population, a workload is generated which is determined by the first group of parameters (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 in the second group or solution (scheduler, read_ahead_kb and swappiness). The performance achieved sets the evaluation function of each of the individuals in the population. An increase in the transfer rate associated to a reduction of the average access time clearly indicates an enhanced performance.

The source code of IoPerf is available at http://www.tic.udc.es/nino/ioperf/ioperf.zip.

## 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 40 GB, 7200 rpm, 2 MB 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 (Manning and Kuoppala, 2003) has been used to measure disk I/O performance.

A 30 s simulation time is used to evaluate each individual in the GA. 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.

With the goal of comparing the GA performance with that of the default kernel, each new population is tested with 1000 workloads of disk access. Each of them runs for 300 s 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. An IOBad module was developed for that task. This application issues the corresponding sentences without modifying the kernel parameters. Various analyses are made about our module's behavior. The graphs of Figs. 3 and 4 used only show the first 160 tests out of the 1000 that were made, with the purpose of enhancing comprehension and visual analysis.

Fig. 3 shows the productivity in MBps of the initial 160 cases of the eighth population. 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. Fig. 4 shows the average access time

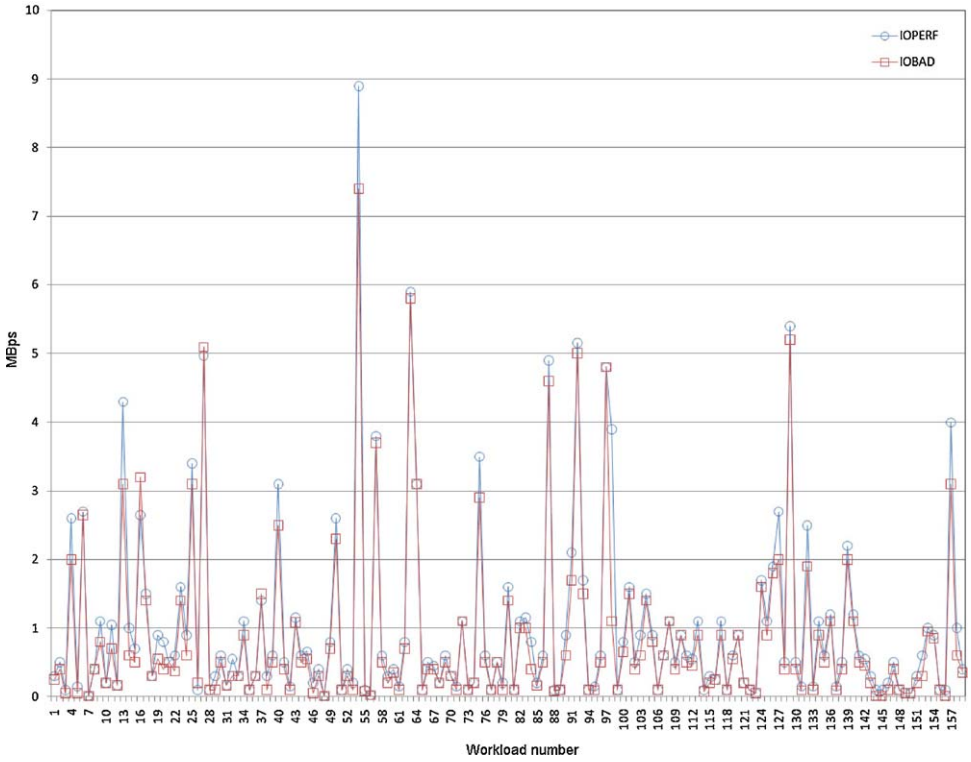


**Fig. 3.** Mean productivity per process.

of the first 160 cases of the eighth population. 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%.

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

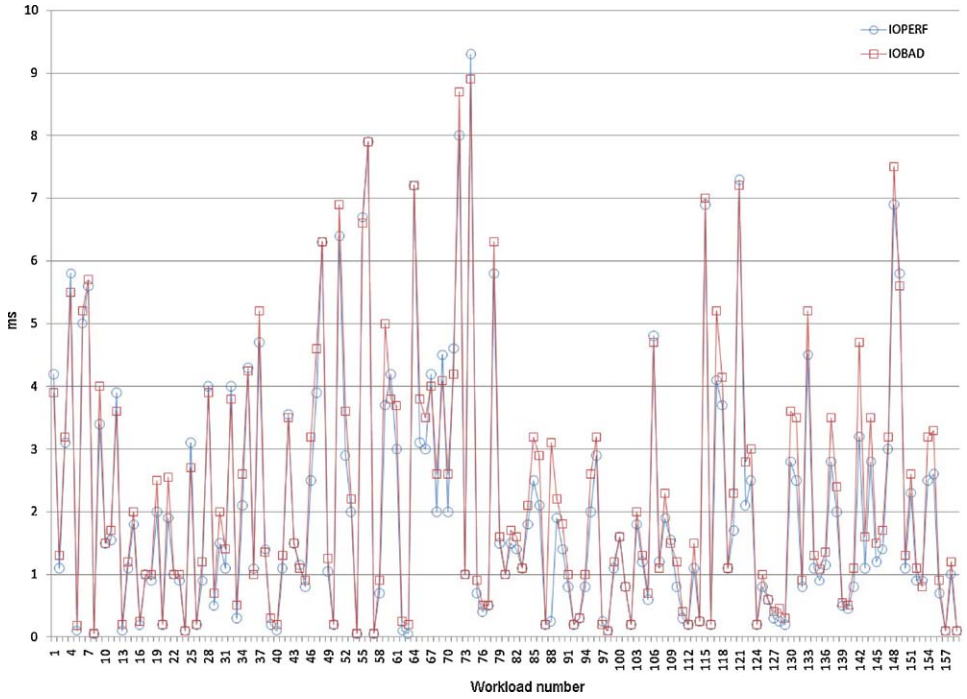


**Fig. 4.** Mean latency per request.

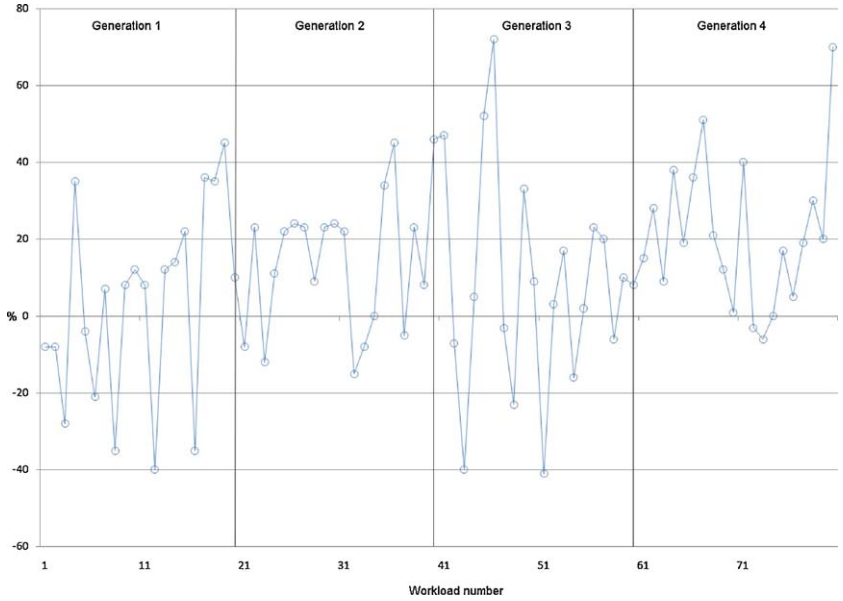


**Fig. 5.** GA evolution. Percentage of improvement of the mean productivity per process.

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 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. Fig. 5 shows the percentage of variation in average productivity per process achieved by the IOPerf module vs. IOBad. Fig. 6 shows the percentage of variation in average response time per request. The first 20 workloads of each generation are shown. Each randomly generated workload runs for 300 s. 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 a percentage below 0%.

The experimentation stage has also used another environment integrated by an Intel Core2 6320 1.86 Ghz with 2 GB of RAM and an ST380215AS hard disk of 80 GB, 7200 rpm, 2 MB of buffer-cache, with internal transfer speed of 930 Mbps, external transfer speed of 300 MBps and a positioning time of 4.16 ms. The operating system used is a Debian 6.0.1 with a 2.6.32 kernel. The file system mounted is an ext3 and the swap space assigned in the disk is 4 GB.

Under this new experimental configuration, a significant performance difference is also obtained in favor of the IOPerf module,

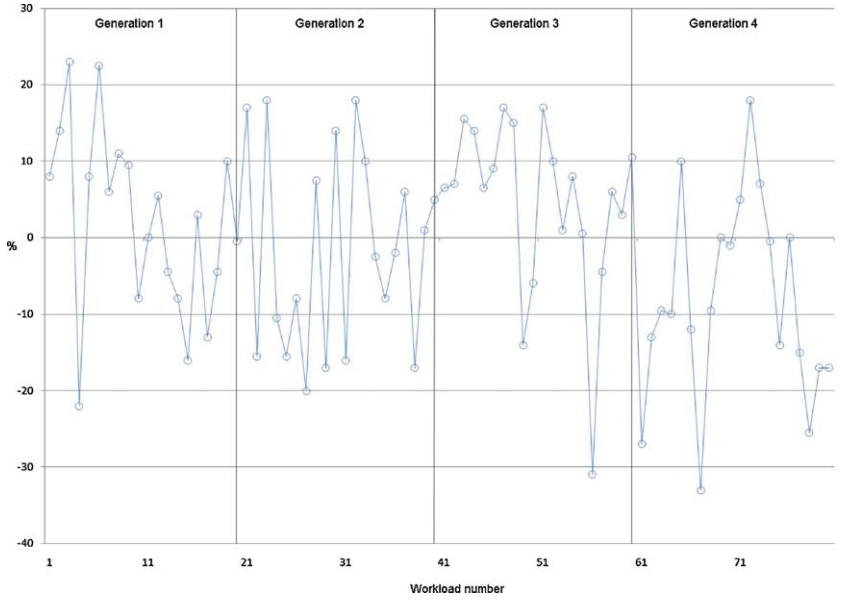


**Fig. 6.** GA evolution. Percentage of improvement of the mean latency per request.

taking into account the initial 20 cases of the eighth population. Productivity increases in 95% of the cases, with an average improvement of 39.8%. The goal of decreasing the mean latency was also achieved in 75% of the cases, with an average reduction in the I/O request processing time of 17.4%.

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, and not to reduce it in reading ones. 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.

## 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 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 subsystem. 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. These results are considerably better than those obtained in the works corresponding to the state of the art. Moilanen and Williams (2005) achieves an average enhancement of throughput of 8.72%. Zhang and Bhargava (2009) achieve a latency enhancement of 14.5% in the best case scenario.

Besides, the GA provides a population of solutions, each of them adapted to a workload scenario. Thus, the system is capable of adapting dynamically and in real-time to the different configurations and workloads that could appear.

This paper proves the advantages posed by integrating GAs in the management subsystem of disk I/O. 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. GAs can quickly and automatically adapt themselves to these changes by means of an endless evolution scheme. According to this approach, a new population would be generated from time to time. The current population would be used for its generation, together with those individuals achieved from a periodical system monitoring.

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.

## Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive comments, suggestions and criticisms. This research is partially funded by: the Spanish Ministry for Science and Technology, research project TIN200806562/TIN; Xunta de Galicia, research project XUGAPGIDIT10TIC105008PR.

## References

Bovet, D., Cesati, M., 2005. Understanding The Linux Kernel. O'reilly & Associates Inc.
Davis, L.D., Mitchell, M., 1991. Handbook of Genetic Algorithms. Van Nostrand Reinhold.
Dunn, M., Reddy, A.L.N., 2009. A New I/O Scheduler for Solid State Devices. Technical Report. Texas A & M University.
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.
Flexible File System Benchmark, 2008. http://sourceforge.net/projects/ffsb/.
Gaul, 2009. Personal Home Page. http://gaul.sourceforge.net.
Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA.
Hagen, W.V., 2002. Linux Filesystems. Sams, Indianapolis, IN, USA.
Hennessy, J.L., Patterson, D.A., 2007. Computer Architecture. A Quantitative Approach. Elsevier.
Holland, J.H., 1992. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA, USA.
Intel Iometer, 2008. http://www.iometer.org/.
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.
Kesavan, M., Gavrilovska, A., Schwan, K., 2010. On disk i/o scheduling in virtual machines. In: Proceedings of the 2nd Conference on I/O Virtualization, USENIX Association, Berkeley, CA, USA, p. 6.
Kim, J., Oh, Y., Kim, E., Choi, J., Lee, D., Noh, S.H., 2009. Disk schedulers for solid state drivers. In: Chakraborty, S., Halbwachs, N. (Eds.), EMSOFT. ACM, pp. 295–304.
Lind, R., 2003. Linux Kernel http://Documentation/iostats.txt. Accessible via http://www.mjmwired.net/kernel/Documentation/iostats.txt.
Love, R., 2004. Kernel korner: I/O schedulers. Linux J. 2004, 10. Accessible via http://www.linuxjournal.com/article/6931.
Love, R., 2005. Linux Kernel Development, 2nd edition. Novell Press.
Manning, J., Kuoppala, M., 2003. Tiobench Benchmark. Accessible via http://sourceforge.net/projects/tiobench/.
McKenney, P., 1990. Stochastic Fairness Queueing.
Microsoft, 2009. Windows Performance Analyzer. Accessible via http://msdn.microsoft.com/en-us/performance/cc825801.aspx.
Moilanen, J., Williams, P., 2005. Using genetic algorithms to autonomically tune the kernel. In: Proceedings of the Ottawa Linux Symposium, Linux Symposium, Inc, pp. 327–338.
Musumeci, G.P.D., Loukides, M., 2002. System Performance Tuning, 2nd edition (O'Reilly System Administration). O'Reilly Media Inc.
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: Proceedings of the 2003 Ottawa Linux Symposium, p. 03.
Nerin, Feng, 2009. Linux Kernel Documentation/Filesystems/Meminfo. Accessible via http://www.mjmwired.net/kernel/Documentation/filesystems/proc.txt.
Norcott, W.D., 2006. IO Zone Filesystem Benchmark. Accessible via http://www.iozone.org.
Oak, T.D., Dunigan, T., Mathis, M., Tierney, B., 2002. A TCP tuning Daemon. In: Proceedings of SuperComputing: High-Performance Networking and Computing, pp. 1–16.
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.
Semke, J., Mahdavi, J., Mathis, M., 1998. Automatic TCP buffer tuning. In: SIGCOMM, pp. 315–323.
Setoolkit, 2009. The Symb EL Language Reference Manual. Accessible via http://www.setoolkit.org/cms/node/3.
Shakshober, D., 2005. Choosing an I/O Scheduler for Red Hat Enterprise Linux 4 and the 2.6 Kernel.
Tanenbaum, A.S., 2007. Modern Operating Systems. Prentice Hall Press, Upper Saddle River, NJ, USA.
Zhang, Y., Bhargava, B.K., 2009. Self-learning disk scheduling. IEEE Trans. Knowl. Data Eng. 21, 50–65.

**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.

**Adrian Carballal Mato** received the B.S. degree in Computer Science from the University of A Coruña (Spain) in 2009, and the Ph.D. degree in Computer Science in 2011. He is currently working as research associate at the Department of Information Technologies and Communications and participates in several Spanish research projects.