IOAgent: a Parallel I/O Workload Generator

Sergio Gómez-Villamor[1], Victor Muntés-Mulero[1], Marta Pérez-Casany[1], John Tran[2], Steve Rees[2], and Josep-L. Larriba-Pey[1] *

[1] DAMA-UPC
Computer Architecture Dept., Universitat Politècnica de Catalunya
Jordi Girona 1-3, Campus Nord-UPC, Modul C6, 08034 Barcelona
{sgomez,vmuntes,larri}@ac.upc.edu, marta.perez@upc.edu
http://www.dama.upc.edu
[2] IBM Toronto Lab.
8200 Warden Ave., Markham, ON L6G1C7, Canada
{jbtran,srees}@ca.ibm.com

Abstract. The purpose of this paper is twofold. First, we present IOAgent, a tool that allows to generate synthetic workloads for parallel environments in a simple way. IOAgent has been implemented for Linux and takes into account different I/O characteristics like synchronous and asynchronous calls, buffered and unbuffered accesses, as well as different numbers of disks, intermediate buffers and number of agents simulating the workload. Second, we propose statistical models that help us to analyze the I/O behaviour of an IBM e-server OpenPower 710, with 4 SCSI drives. The observations used to build the model have been obtained using IOAgent.

keywords: parallel I/O, synthetic workload generator, Linux, performance evaluation, statistical modelling

1 Introduction

The quest for tools that generate workloads for the evaluation of parallel I/O comes from long ago [1]. It is a need both as an aid for the optimum configuration of complex applications on complex computer architectures, and for the evaluation of research on Operating Systems and storage performance. The reason for such tools to exist is that they can emulate the behaviour of complex applications, avoiding the use of such applications for evaluation.

Bonnie [2], *LMbench* [3] and *FileBench* [4] are examples of tools that allow this type of workload emulation for Unix-like environments. However, such tools are restricted from many different points of view, like the number of threads and the type of I/O operating system calls that they can trigger.

* The authors at UPC thank Generalitat de Catalunya for its support through grant GRE-00352. Sergio Gómez-Villamor thanks IBM for its support through a CAS grant.

In order to give access to a generic workload generator for the evaluation of parallel I/O subsystems, we present IOAgent. Our tool sits on top of the Operating System in the same way as an application would do, and allows for the generation of workloads tailored to the needs of the system to evaluate. IOAgent allows users to generate synthetic application-level requests and defines a considerable number of behaviour-variables that allow to mimic an application. Our tool has been tested in large environments with up to 24 disks to emulate the accesses caused by a DBMS with a transactional processing workload [5].

In addition, as an example of the use of IOAgent, we evaluate an OpenPower 710 I/O subsystem. For the evaluation, we generate different workloads that stress the I/O subsystem in different ways and execute them to obtain more than 10 thousand execution time measures. With all those data, we propose a statistical model of the system using the Analysis of Variance (ANOVA). We show that using statistical tools it is easier and more reliable to extract conclusions.

The rest of this paper is organized as follows. In Section 2 we explain IOAgent. Then, in Section 3, we describe the environment and tests performed. In Section 4 we describe the statistical models obtained and in Section 5 we discuss the results with the help of the models. In Section 6 we give a short overview of the literature on the topic and, finally, we conclude.

2 IOAgent

IOAgent offers the possibility to mimic the stress imposed by an application on the I/O subsystem. This is done by simulating processes that exercise read and write predefined patterns on the accessed devices. Thus, IOAgent allows to evaluate parallel environments, both from the processing and the I/O points of view.

Currently, most OSs provide different system calls to execute I/O operations in a *synchronous* or *asynchronous* manner. Also, depending on whether the data blocks involved in I/O operations are mapped onto kernel buffers we can distinguish between *buffered* or *unbuffered* I/O. IOAgent allows for all those possibilities.

At this moment, IOAgent is implemented for Linux but could be ported to any other OS with little effort. IOAgent can be freely accessed in [6].

2.1 Basic Structures

The synthetic workload configuration of IOAgent is set by means of (i) a set of per-thread access-patterns, also called *agents*, and (ii) a set of pseudo-devices, also called *files*.

Agents are responsible for performing the desired stress. Every *agent* is a thread which performs one access pattern. The mixed execution of different *agents* will determine the desired global workload. Each *agent* in a simulation will have specific values for the following fields:

- ***File***. The *file* used to perform the I/O operations.
- **Operation mode**. There are four simultaneously compatible operation modes:
  - **Synchronous or asynchronous**. Synchronous I/O operations block a process until the I/O is performed while asynchronous I/O operations do not block the process.
  - **Number of buffers**. IOAgent associates each I/O operation to a buffer. Synchronous operations are performed sequentially, thus, IOAgent requires only one buffer for synchronous agents. Asynchronous operations can be performed concurrently, thus, asynchronous agents require as many buffers as I/O operations on the fly. The buffers of IOAgent emulate those provided by the application it is emulating.
  - **Buffered or unbuffered**. With buffered I/O, IOAgent uses its own buffers, and indirectly those provided by the OS as an intermediate step for the I/O operations. With unbuffered I/O, IOAgent only uses its own buffers. Unbuffered I/O can be specified using raw devices or the *Direct I/O* mode, which can be associated to a block device interface or a file of some file systems (*i.e.* some file systems do not allow *Direct I/O* mode).
  - **Read or write**. Each agent will perform exclusively read or write operations.
  - **Sequential or random**. Sequential accesses perform a set of consecutive or strided I/O operations. Random accesses perform a number of read or writes over each position of the file following a certain probability distribution. At present, it is possible to choose among *Uniform* or *Poisson* distributions.
- **Operation size**. The size of the I/O operations is fixed for an *agent*.
- **Inter-arrival times**. We fix this per-*agent* value as the number of I/O operations per unit of time that each *agent* must generate.

Files are used in our environment to encapsulate the different storage capabilities of a system. Therefore, every *agent* is associated to a *file* which represents where the *agent* performs its I/O operations. *Files* are devices or common files on which I/O operations are executed. Regarding the storage system support, IOAgent allows for:

- **File systems**. The most common way to access hard disks (or its logical partitions) is through files of a built-in file system (*e.g. ext3*, *ext2*, *reiserfs*, *xfs*, *jfs*, etc.).
- **Block devices interface**. Different block devices (*e.g.* hard disks) of UNIX-type systems can be accessed through the `/dev` interface.
- **Raw devices interface**. A raw device can be bound to an existing block device (*e.g.* a disk) and can be used to perform raw I/O with that existing block device. Such raw I/O bypasses the caching that is normally associated with block devices.

For a comprehensive explanation of other parameters not used in this paper, and how to configure a workload generation, we refer the reader to [5].

Processor	2 Power5 at 1.65 GHz
Memory	4 GBytes, DDR-I ECC at 266 MHz
L1 data cache	4-way set associative LRU
L2 cache	10-way set associative 1.9 MBytes
L3 cache	36 MBytes
Storage	4 146.8 GBytes drives at 10 Krpm 2 channel Ultra320 SCSI controller 320 MBps peak transfer

Table 1. OpenPower 710 configuration.

3 Evaluation Setup

In order to show the use of IOAgent, we perform an analysis of the I/O performance characteristics of an IBM e-server OpenPower 710 [7], with 4 SCSI drives, and Red Hat Enterprise Linux AS v4 for 64-bit IBM Power based on the 2.6 Kernel. The configuration of the OpenPower 710 evaluated in this paper is as shown in Table 1.

We run extensive executions of IOAgent on the system. The executions added up to 10,368 performance measures (two weeks of executions). For the evaluation, we have set up the parameters of IOAgent shown in Table 2.

Buffered and unbuffered I/O tests were both performed on 4 GBytes files, one per disk used. For buffered I/O tests, we used files mounted with an *ext3* file system and to allow buffered file system asynchronous I/O we patched a 2.6.12 kernel version [8]. Asynchronous I/O is a very recent feature in the Linux kernel, therefore we focused our studies on this new feature. For unbuffered I/O we used a 2.6.9 kernel version.

In order to analyze the system under maximum stress, we fix the inter-arrival time to zero for all the agents.

4 Statistical Modelling

We study five different categorical variables (factors) related to the I/O subsystem performance, namely, the use of the OS buffers, the number of disks accessed by the application, the number of agents accessing the disks, the number of intermediate application buffers used to store the data managed during the I/O operations, and the size of those intermediate application buffers. As a response variable, we study the average transfer rate from disk for four different access patterns: sequential reads, sequential writes, random reads and random writes. We propose two models, a general one useful for the four access patterns, and a simplified specific model for random reads, based on the five factors mentioned above.

The models we propose provide a way to analyze the data collected and allow us to find out which parameters are most significant.

Factor	Name	Levels	#levels
Buffered	O	Buffered and Unbuffered	2
#disks	D	1, 2 and 4	3
#agents/disk	A	1, 2, 4, 8, 16 and 32	6
#buffers/agent	B	1, 2, 4, 8, 16 and 32	6
buffer size (KBytes)	S	8, 32 and 128	3

Table 2. Parameters used for the evaluation.

4.1 The Models

We use the Analysis of Variance (ANOVA) because it is the classical statistical technique to describe the behaviour of a response variable as a function of some factors [9]. Conversely, regression plays the same role for continuous variables. The factors considered in the models and their levels are summarized in Table 2. All of them are fixed effect factors, which means that the levels are considered constants. This case is opposed to the random effects case, in which they are considered observations from a random variable.

First, we try to model the transfer rate as a function of the main effects of the factors without interactions. The results are not satisfactory since the errors (difference between the observed and the predicted values) do not satisfy the hypothesis of independence, normality and equal variance, required for the ANOVA. The problem disappears by transforming the response variables by means of a logarithm, and considering some interactions in the model. Following the Principle of Parsimony [9], we have finally accepted the following model for the four access patterns analyzed in this paper:

$$\begin{aligned} y_{ijkml} = \mu + O_i + D_j + A_k + S_l + B_m + \\ +(OD)_{ij} + (OA)_{ik} + (OS)_{il} + (OB)_{im} + (DA)_{jk} + \\ +(AS)_{kl} + (AB)_{km} + (SB)_{lm} + e_{ijklm} \end{aligned} \tag{1}$$

for $i = 1..2$ (2 levels for factor O), $j = 1..3$ (3 levels for factor D), $k = 1..6$ (6 levels for factor A), $l = 1..3$ (3 levels for factor S), and $m = 1..6$ (6 levels for factor B) where,

1. y_{ijkml} is the logarithm of the average transfer rate of 4 executions of the application, that have been run under conditions i, j, k, m, l of the factors.
2. μ is known as the general average. In our case, it represents the mean value of the logarithm of the average transfer rate expected if the conditions under which the observation has been obtained are unknown.
3. O_i, D_j, A_k, S_l and B_m correspond to the main effects of the five factors explained before. Specifically, O_i corresponds to the effect of the ith level of O, D_j corresponds to the effect of the jth level of D, and so on. For being a fixed effects model, the conditions $\sum_i O_i = \sum_j D_j = \sum_k A_k = \sum_l S_l = \sum_m B_m = 0$ must be satisfied.
4. $(OD)_{ij}$ corresponds to the interaction of the ith level of O with the jth level of D. Those constants must verify that $\forall i$, $\sum_j (OD)_{ij} = 0$ and $\forall j$, $\sum_i (OD)_{ij} = 0$. Analogously, $(OA)_{ik}$, $(OS)_{il}$, $(OB)_{im}$, $(DA)_{jk}$, $(AS)_{kl}$, $(AB)_{km}$ and $(SB)_{lm}$ correspond to the different interactions between the

levels of the corresponding factors, and the corresponding analogous restrictions must be verified.

5. e_{ijklm} corresponds to the experimental error and contains the information in the data which is not explained by the considered factors.

However, for random reads, the model above can be simplified since some interactions are not statistically significant. The simplified model is:

$$\begin{aligned} y_{ijkml} = {} & \mu + O_i + D_j + A_k + S_l + B_m + \\ & +(OD)_{ij} + (OS)_{il} + (OB)_{im} + (AB)_{km} + e_{ijklm} \end{aligned} \tag{2}$$

The R-Squares of the four response variables modeled (sequential reads and writes, and random reads and writes) are 0.81, 0.95, 0.99 and 0.99 respectively, using model (1) for all the cases except for random reads, where we use model (2). This means that the models explain the corresponding percentage of the total variability in the data (*i.e.* 81% for 0.81). The error terms for each model are independent and follow a normal distribution with zero mean and constant variance. Therefore we can accept model (1) for sequential I/O activity and random writes and model (2) for random reads.

5 Discussion

In the following paragraphs we dissect the general and specific characteristics of the four types of accesses that we exercised: sequential reads and writes, and random reads and writes. Although the models characterize the logarithm of the transfer rate, we always refer to the transfer rate of the I/O subsystem for simplicity in the text. All the plots show averages obtained from real executions.

5.1 Single Factor Analysis

First of all, it is remarkable that the models show an important difference between the levels of factor O, OS buffered/unbuffered accesses. While OS buffered accesses work better in sequential reads, unbuffered accesses work better in the rest of the cases. This can be understood from the fact that the OS monitors sequential read access patterns and prefetches data blocks in those cases.

A general characteristic for factor A is that, in random accesses, the larger the number of agents, the better, while in sequential accesses, the smaller the number of agents, the better. An explanation to this follows. One of the strategies to maximize throughput is to sort disk accesses to reduce the number of backward and forward disk arm movements [10]. Therefore, in random accesses, a small number of agents may cause the disk arms to move backward and forward constantly. A larger number of random accesses (*i.e.* a larger number of agents) causes more chance to have accesses to be on the route between two far away accesses, improving the usage of the resource. On the other hand, in sequential accesses, there is a trade off between the number of accesses and the randomness introduced by having several sequential accesses along a significant lapse of time.

Factors D and S increase the transfer rate of the I/O subsystem as their values increase. Finally, although factor B behaves in the same way, there are cases where its influence is unnoticeable.

5.2 Multiple Factor Analysis

Now we analyze the most significant interactions between pairs of factors for the different access patterns.

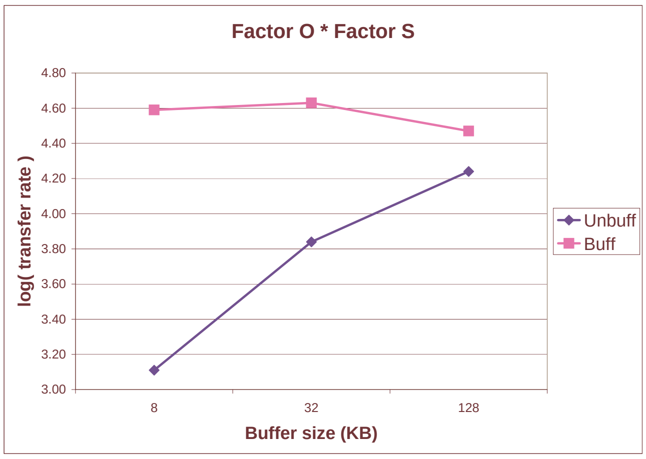
Sequential Reads. Among the interactions modeled for this type of I/O accesses, we found that the most interesting were the two shown in Figure 1: (top chart) between the OS buffered/unbuffered I/O, factor O, and the size of the application buffers, factor S, and (bottom chart) between OS buffered/unbuffered I/O, factor O, and the number of application buffers used, factor B.

As shown in Figure 1, OS buffered I/O always behaves better than unbuffered I/O for sequential reads. Also, the interactions show that both increasing the number of application buffers (factor B) or their size (factor S) benefits unbuffered I/O while, in general, it is less significant for OS buffered I/O throughput. This makes sense since the OS prefetches or reads data ahead when it detects sequential access patterns. On the other hand, unbuffered I/O improves with larger number and size of buffers (factors B and S respectively) because the bandwidth of the I/O is exercised more intensely either with more agents in parallel or with larger data sets accessed sequentially that, in both cases, allow for better throughput.

Sequential Writes. The interactions that we chose in this case for their significance are shown in Figure 2: (top chart) between the number of agents, factor A, and the OS buffered/unbuffered I/O, factor O, and (bottom chart) between the number of agents, factor A, and the number of buffers per agent, factor B.

The first interaction shows how increasing from one agent to two decreases the performance of the I/O subsystem in OS buffered accesses. This is caused by the randomness added by one more agent accessing a different set of data. The performance for two agents is sustained for more agents as shown in the same plot. Note that if we increase the size of the buffer (not shown in the plots), we improve the performance, showing that increasing the sequentiality benefits performance.

Turning to the plot at the bottom, we can see that the number of buffers used per agent saturates the performance at a certain point with 8 or 16 buffers per agent for one agent and more for more agents. This shows that having a larger number of buffers allows for a better planning of the I/O activity when an agent is writing data to the disks. This is also true when we increase the number of agents but, in those cases, the randomness and the number of context switches introduced due to a larger number of agents reduces significantly the performance



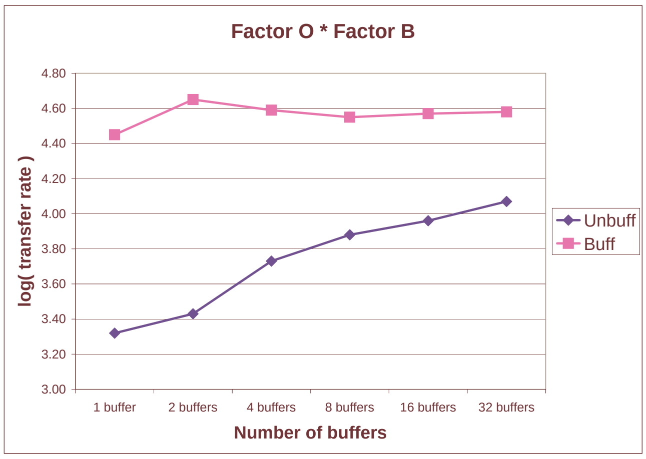
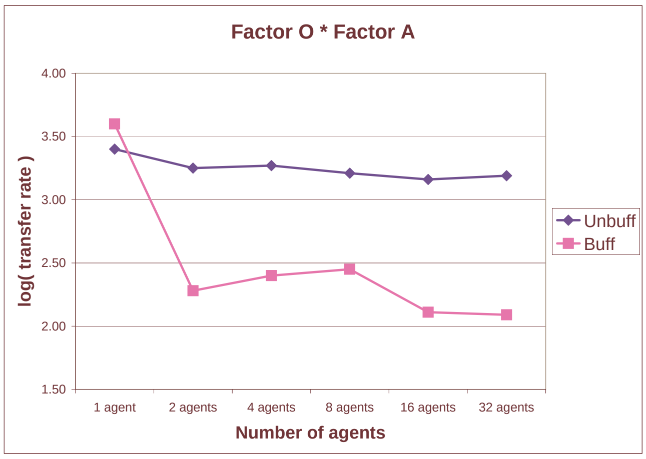


Fig. 1. (top) Interaction between OS buffered I/O, factor O, and the application buffer size, factor S, and (bottom) interaction between OS buffered I/O, factor O, and the number of application buffers, factor B, for sequential data reads.

of the system. Therefore, the performance decreases when the number of agents increases.

In general, we can say that it is beneficial to have a large number of buffers per agent. In particular, the performance starts to saturate at 16 to 32 buffers for the case of one agent.

Finally, not in the plots, we observe that the interaction between the number of buffers per agent (factor B) and their size (factor S) is also significant. In this case, the smaller the size of the buffer, the more beneficial it is to have a larger number of buffers. In any case, the combination of large buffers with large number of buffers is the best, even though it is more important to have large buffers than a large number of them.



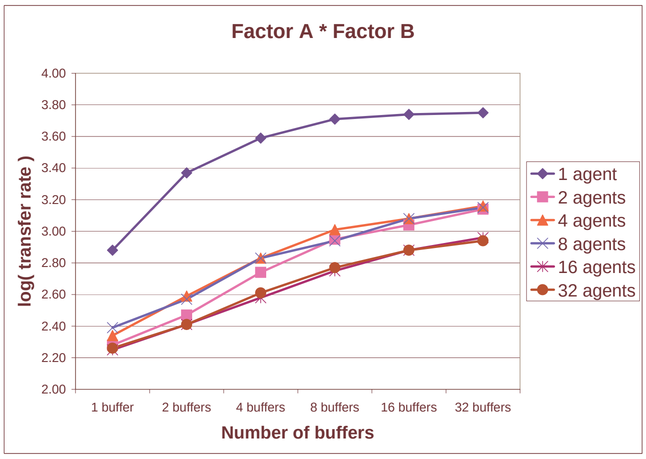


Fig. 2. (top) Interaction between OS buffered I/O, factor O, and the number of agents, factor A, and (bottom) interaction between the number of agents, factor A, and the number of buffers per agent, factor B, for sequential data writes.

Random Reads and Random Writes. As a general observation, in the plot of Figure 3 we observe that a larger number of agents is beneficial for random operations, as opposed to sequential operations (plot at the bottom in Figure 2), where it was better to have less agents.

More specifically, in the plot of Figure 3, we show the interaction between the number of buffers per agent (factor B) and the number of agents (factor A) for random reads (the behaviour for random writes is similar although the model for random reads is simpler). The plot shows a clear tendency to converge to an asymptote, with larger numbers of agents converging faster. This is so because there is a better planing of the resources when the number of agents and buffers is large. However, the size of the buffer (not shown in the plot) does not have a significant interaction neither on the number of agents nor on the number of buffers.

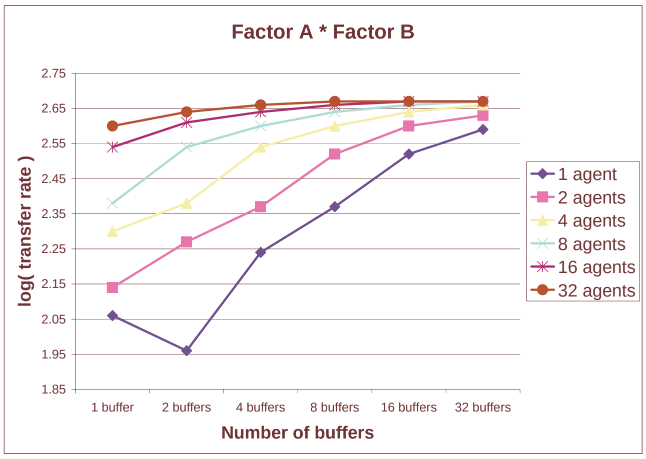


Fig. 3. Interaction between the number of agents, factor A, and the number of buffers per agent, factor B, for random data reads.

6 Related Work

There is a significant amount of work on synthetic workload generation tools for Unix-like systems as *Bonnie* [2], *LMbench* [3] and *FileBench* [4]. However, those softwares do not have all the features IOAgent offers and we have used here, as explained above.

On the other hand, we are not aware of the existence of a comprehensive study of the performance of the Linux asynchronous I/O on parallel devices before, or the Linux I/O performance on Power-based architectures. Moreover, our characterization can be regarded as the first one to use statistical methods for the analysis of the results. However, we want to mention the following pieces of work related to Linux I/O characterization, as examples. Ram Pai *et al.* [11] used *iozone* to study the performance improvement through readahead optimization. Unfortunately, the tool used did not allow to test the optimizations under the Linux native asynchronous I/O. Also, Suparna *et al.* [12] analyzed the performance and robustness of the new Linux asynchronous I/O for enterprise workloads. The main difference between the evaluation done in this paper and that in [12] is that we give a more generic view and make extensive use of statistical infrastructure to validate the model presented.

7 Conclusions

This paper shows the importance of using a solid software and mathematical infrastructure for the evaluation of hardware/software systems. Our most important conclusion is that using a workload generator and a statistical methodology, we can extract solid and sound conclusions about the interaction of a simulated application, the OS and the hardware at use, in particular, the I/O subsystem.

Access pattern	**Buffered mode**	**Number of agents**	**Buffer size**	**Number of buffers**
Sequential reads	Buffered	small	large	*SNS*
Sequential writes	Unbuffered	small	*SNS*	large
Random reads and writes	Unbuffered	large	*SNS*	large

Table 3. Recommended configurations. SNS stands for Statistically Not Significant.

We have generated test cases for the evaluation of the I/O subsystem of an IBM OpenPower 710. With more than 10 thousand execution results, we have built a statistical model that describes and fits accurately the behaviour of the I/O subsystem. Table 1 summarizes the best configurations for the four scenarios analyzed in this paper.

From the point of view of the use of IOAgent in a parallel environment, we show that the benefits obtained with an increase in the degree of parallelism is not straight forward.

References

1. Greg Ganger. Generating Representative Synthetic Workloads: An Unsolved Problem. *Proceedings of the Computer Measurement Group (CMG) Conference*, 1995.
2. Tim Bray. Bonnie Benchmark for UNIX Filesystem Operations. *http://www.textuality.com/bonnie.*
3. Larry McVoy and Carl Staelin. LMbench: Portable Tools for Performance Analysis. *USENIX Annual Technical Conference, San Diego, California, USA*, 1996.
4. Richard McDougall, Joshua Crase, and Shawn Debnath. FileBench: File System Microbenchmark. *http://www.solarisinternals.com/si/tools/filebench.*
5. Sergio Gómez-Villamor, John Tran, Steve Rees, Victor Muntés-Mulero, and Josep-L. Larriba-Pey. IOAgent: Leveraging the Application Analysis of Workload Effects. *Technical Report UPC-DAC-RR-2005-49, Department of Computer Architecture, Universitat Politecnica de Catalunya*, 2005.
6. DAMA-UPC. Data Management group at Universitat Politècnica de Catalunya. *http://www.dama.upc.edu/en/research/ioagent.html.*
7. *IBM e-server OpenPower 710 Technical Overview and Introduction.*
8. Suparna Bhattacharya. Patches for buffered file system AIO in 2.6 Linux kernel. *http://www.kernel.org/pub/linux/kernel/people/suparna/aio.*
9. Douglas C. Montgomery. *Design and analysis of experiments.* John Wiley, New York, fifth edition, 2001.
10. Bruce L. Worthington, Gregory R. Ganger, and Yale N. Patt. Scheduling Algorithms for Modern Disk Drives. *SIGMETRICS Perform. Eval. Rev.*, 1994.
11. Ram Pai, Badari Pulavarty, and Mingming Cao. Linux 2.6 Performance Improvement Through Readhead Optimization. *Proceedings of the Linux Symposium, Ottawa, Canada*, 2004.
12. Suparna Bhattacharya, John Tran, Mike sullivan, and Chris Mason. Linux AIO Performance and Robustness for Enterprise Workloads. *Proceedings of the Linux Symposium, Ottawa, Canada*, 2004.