An analysis of the I/O performance of the OpenPower 710 on Linux

S. Gómez-Villamor\textsuperscript{1} J. B. Tran\textsuperscript{2} Steve A. Rees\textsuperscript{2} V. Muntés-Mulero\textsuperscript{1} J. L. Larriba-Pey\textsuperscript{1}

\textsuperscript{1}DAMA-UPC
Computer Architecture Dept.
Universitat Politècnica de Catalunya
Jordi Girona 1-3, Campus Nord-UPC, Modul C6, 08034 Barcelona
e-mail: \{sgomez,vmuntes,larri\}@ac.upc.edu

\textsuperscript{2}IBM Toronto Lab.
8200 Warden Ave., Markham, ON L6G1C7, Canada
e-mail: \{jbtran,srees\}@ca.ibm.com

ABSTRACT
The importance of a well tuned I/O subsystem is beyond doubt. In many cases, I/O may be the real bottleneck of an application and understanding its limitations and capabilities may be of interest both from the research and development points of view.

In this paper we analyze the I/O subsystem of an IBM e-server OpenPower 710 with 4 SCSI disks for asynchronous and OS unbuffered I/O. The analysis is done with the aid of IOAgent, a tool that allows to generate synthetic I/O workloads for Linux.

With the performance measures we build a statistical model of the I/O subsystem behavior. The most relevant results show that the number of disks being accessed has a positive influence on the absolute performance of an I/O bound simulated application. However, in general, our results also show that sequential I/O activity benefits more from a small number of processes and application buffers, while random I/O activity benefits more from a large number of processes and application buffers.

keywords: I/O, Linux, performance evaluation

1. INTRODUCTION
The tuning of an application has many parameters to be taken into account. In general, the architecture of the computer (from processor to permanent storage), the operating system and the application itself are to be considered, and their interaction may lead to many unexpected outcomes.

In any case, the importance of I/O and the interaction between the hardware involved and the operating system calls issued by the application put those aspects in the focal point of the analysis. However, large applications are usually difficult to set up, taking a large amount of time to configure and test on different environments, and making the whole process too long for a productive cycle.

In order to understand the behavior of the I/O subsystem of the OpenPower 710 under Linux, we have executed, modeled and analyzed different I/O stresses. Instead of using a specific application, we have used IOAgent \cite{IOAgent} to mimic the behavior of different situations exercised by an application. IOAgent has been designed as a general tool for the analysis and research of I/O subsystems. Our tool allows to exercise the I/O subsystem under different parameterizable variables like synchronous/asynchronous or buffered/unbuffered accesses among others as explained in detail in \cite{IOAgent}. This way an application does not need to be configured but modeled in such a way that IOAgent emulates its I/O access patterns.

In particular, we run a set of experiments and model
them statistically. Our statistical models are sound and explain all the cases executed. For instance, the models allow us to state that the transfer rate achieved with 1, 2 and 4 disks increases significantly as the number of disks grows, as expected. However, our models also show that the influence of an increase on the number of processes and application buffers for a fixed amount of disks is, in general, negative for sequential I/O while it is positive for random I/O.

The paper is structured in the following way. We start giving a short account of IOAgent in Section 2. Then, in Section 3 we describe the analysis that we performed. In Section 4 we present an statistical model. Finally, we give a short overview of the literature on the topic and conclude.

2. IOAGENT

IOAgent allows to generate synthetic workloads to analyze the I/O behavior of an emulated application when executed on specific environments. It offers the possibility to mimic the stress imposed by the application on the I/O subsystem by simulating the process workloads and by exercising the read and write patterns on the accessed devices.

To do that it is possible to configure IOAgent after modeling the application so that different configurations of devices and threads accessing them can be tested without having to configure the application every time a test is run.

IOAgent is implemented for Linux at this moment but could be supported for any other operating system with little effort.

2.1 Parametrization of IOAgent

IOAgent allows the user to define different per-thread access-patterns, also called agents, each being exercised on a set of pseudo-devices, also called files which encapsulate the different storage possibilities of a system. A per-thread access-pattern is determined by a large set of properties (i.e. size and frequency of its I/O operations). The mixed execution of different threads (each with its own access-pattern) will determine the global workload of an emulated application.

A file in IOAgent can support any file of a well supported file system, block devices and raw devices.

An agent can perform I/O operations of a predefined size on a file. The operations of an agent can be parameterized in terms of: (i) read or write, (ii) buffered or unbuffered, where buffered stands for operations with intermediate operating system buffers, and (iii) synchronous or asynchronous. The I/O operations performed by an agent can be random or sequential. A probability distribution can be defined for random operations so that some positions are accessed more frequently than others. At this moment, either Uniform or Poisson distributions are available but other distributions can be included. Also, a stride can be defined for sequential operations where a fixed distance between data accessed is forced. Finally, a frequency can be defined by means of setting a delay between I/O operations by means of holding the CPU or blocking the agent and leaving the CPU for another agent to take over.

A comprehensive explanation of how to configure a workload generation and fix different properties about agents and files is given in [3].

3. EVALUATION SETUP

We perform an analysis of the I/O performance characteristics of an IBM e-server OpenPower™ 710 [1], with 4 SCSI drives under a Red Hat Enterprise Linux AS v4 for 64-bit IBM POWER based on the 2.6.9 Kernel. The configuration of the OpenPower 710 evaluated in this paper is as follows:

- Two Power5 processors at 1.65GHz.
- 4GBytes of main memory, DDR-I ECC at 266MHz.
- 4-way set associative LRU L1 data cache.
- 10-way set associative 1.9MBytes L2 cache.
- 36MBytes L3 cache.
- 4 146.8GBytes SCSI drives at 10Krpm with a peak transfer rate of 320MBps.
- Two channel Ultra320 SCSI controller.

We run extensive executions of IOAgent on the system in such a way that we were able to fit the statistical models. The executions took more than one week obtaining more than 1000 performance measures. For the evaluation, we have set up the following parameters of IOAgent:

- Only asynchronous I/O.
- Only OS unbuffered I/O.
- Random reads and writes with uniform distribution.
- Sequential reads and writes with no stride between data items.
• 1, 2, 4, 8, 16 and 32 agents.

• 1, 2, 4, 8, 16 and 32 application buffers.

• No delay between consecutive I/O operations.

8KBytes, 32KBytes and 128KBytes buffers.

4. STATISTICAL MODEL AND RESULTS

We analyze four different factors related to the I/O subsystem performance, namely 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 buffers. On the other hand, we study the average transfer rate from disk for four different applications characterized by the following access patterns: sequential reads, sequential writes, random reads and random writes. For each scenario we propose a statistical model based on the four main factors mentioned above and their interactions if necessary.

4.1 Statistical Model

Based on the experimental results we perform an analysis of variance (ANOVA) [2]. The levels considered in the model for each main factor are summarized in Table 1. All of them are fixed effect factors.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Name</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>#disks</td>
<td>D</td>
<td>1, 2 and 4</td>
</tr>
<tr>
<td>#agents/disk</td>
<td>A</td>
<td>1, 2, 4, 8, 16 and 32</td>
</tr>
<tr>
<td>buffer size (KB)</td>
<td>S</td>
<td>8, 32 and 128</td>
</tr>
<tr>
<td>#buffers/agent</td>
<td>B</td>
<td>1, 2, 4, 8, 16 and 32</td>
</tr>
</tbody>
</table>

Table 1: Levels for the main factors used in the statistical models.

We first try to model the transfer rate as a function of the bare factors. The results are not satisfactory since the error does not satisfy the hypothesis of independence and equal variance. The problem disappears by applying the logarithm on the observations. Following the Principle of Parsimony [2] that states that one should always choose the simplest explanation of a phenomenon, in our case, we choose models capable of describing data behavior but simple enough to be handled and interpreted. We have finally accepted the following for models:

\[ y_{ijkl}^{SR} = \mu + \tau_i + \beta_j + \delta_k + \gamma_l + e_{ijkl} \]  

(1)

\[ y_{ijkl}^{SW} = \mu + \tau_i + \beta_j + \delta_k + \gamma_l + (\tau \beta)_{ij} + (\beta \gamma)_{jl} + (\tau \beta \gamma)_{ijl} + e_{ijkl} \]  

(2)

\[ y_{ijkl}^{RR} = \mu + \tau_i + \beta_j + \delta_k + \gamma_l + (\tau \beta)_{ij} + (\beta \gamma)_{jl} + e_{ijkl} \]  

(3)

\[ y_{ijkl}^{RW} = \mu + \tau_i + \beta_j + \delta_k + \gamma_l + (\beta \gamma)_{jl} + e_{ijkl} \]  

(4)

where,

• \( y_{ijkl}^{SR} \) is the response corresponding to the observations of the logarithm of the average transfer rate for 5 executions of an application performing sequential reads.

• \( y_{ijkl}^{SW} \) is the response corresponding to the observations of the logarithm of the average transfer rate for 5 executions of an application performing sequential writes.

• \( y_{ijkl}^{RR} \) is the response corresponding to the observations of the logarithm of the average transfer rate for 5 executions of an application performing random reads.

• \( y_{ijkl}^{RW} \) is the response corresponding to the observations of the logarithm of the average transfer rate for 5 executions of an application performing random writes.

• \( \mu \) is known as the general average. It is the common expected value and it is independent from the levels in the factors. 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.

• \( \tau_i, \beta_j, \delta_k \) and \( \gamma_l \) correspond to the main effects of the four factors explained before. Specifically:

  - \( \tau_i \) corresponds to the effect of the \( i \)th level of \( D \)
  - \( \beta_j \) corresponds to the effect of the \( j \)th level of \( A \)
  - \( \delta_k \) corresponds to the effect of the \( k \)th level of \( S \)
  - \( \gamma_l \) corresponds to the effect of the \( l \)th level of \( B \)

Given the fixed effects of the four factors, it makes sense to assume that \( \forall i, \forall j, \forall k, \forall l \beta_i, \beta_j, \delta_k \) and \( \gamma_l \) are constants such that:

\[ \sum \beta_i = \sum \beta_j = \sum \delta_k = \sum \gamma_l = 0 \]  

(5)
• \((\tau_\beta)_{ij}\) corresponds to the interaction of the \(i\)th level of \(D\) with the \(j\)th level of \(A\). Analogously \((\tau_\gamma)_{il}\), \((\beta_\delta)_{jk}\), \((\beta_\gamma)_{jl}\) and \((\delta_\gamma)_{kl}\) correspond to the different interactions between the levels of the corresponding factors.

• \((\tau_\beta\delta)_{ijk}\) corresponds to the second-level interaction of the \(i\)th level of \(D\) with the \(j\)th level of \(A\) and the \(k\)th level of \(S\).

• \(e_{ijkl}\) corresponds to the experimental error and contains the information in the data which is not explained by any of the different factors.

The R-Squares for the four models are 0.78, 0.99, 0.99 and 0.99 respectively. The error terms for each model, i.e. the difference between the observed values and the fitted values (predicted by the model) are independent and follow a normal distribution with mean zero and constant variance equal to \(\sigma^2\). Therefore we can accept the four models described above.

4.2 Discussion
From the statistical model we can extract some interesting results for the data collected. We use Duncan's test to determine the significant differences between group means in the analysis of the variance setting.

Sequential Reads
Sequential reads are important in situations like scans of tables in data warehouse workloads. This is the case of the first model, and from it we can extract several obvious conclusions that, in general also hold for the other scenarios:

• The average of the logarithm of the transfer rate is statistically different depending on the number of disks. Therefore, the transfer rate increases with as we add disks to our system.

• The average of the logarithm of the transfer rate is statistically different depending on the number of buffers. Specifically, the more buffers in the system, the higher the transfer rate.

• The average of the logarithm of the transfer rate is statistically different depending on the size of the buffers. The transfer rate increases with the size of system buffers.

Not so obvious is the fact that as the number of agents increases the transfer rate decreases. This effect is due to the fact that since read operations are sequential we cannot obtain any benefit from having more than one agent reading from disk. On the contrary, the existence of more than one agent overloads the system and causes the transfer rate to decrease. This effect cannot be observed in applications performing random I/O as we will see in further sections.

Sequential Writes
Sequential writes are important when large intermediate files have to be materialized because the application runs short of memory. In this case, the four observations from the previous subsection hold. We can also say that the number of agents per disk is not statistically different for 2 or 4 agents and for 8, 16 or 32 agents.

Figure 1 shows a noticeable interaction between the number of agents per disk and the number of disks. We can observe that while the system benefits from increasing the number of agents when we use 4 disks, it is affected negatively when we use a single disk. This effect is caused by the fact that all the agents are attempting to write on the same disk overloading the device.

![Figure 1: Plot of the interaction between the number of disks and the number of agents per disk for sequential writes.](image-url)

The plot in Figure 2 shows the interaction between the number of agents per disk and the number of buffers per agent for sequential writes. There, it is possible to see that one agent benefits from incrementing the number of buffers. However, as the number of buffers increases, this benefit is reduced and even decreases when we change from 16 to 32 buffers. As the number of agents increases, it is more beneficial to have the largest number of buffers, although, globally, having more agents reduces the performance of the I/O subsystem.

In general, we can say that it is beneficial to have
as many buffers as possible per agent, although the performance starts to saturate at 16 to 32 buffers. Also, we can say that it is more beneficial to have a significant number of buffers per agent in absolute terms.

Random Reads
Random reads, as well as random writes, are the case for transaction processing where, there are many processes accessing different areas of the disk simultaneously to update, delete and insert records of the database. We discuss this type of applications in this section.

As a general observation, we can say that a larger number of agents is more beneficial for random read operations as opposed to sequential reads, where it was better to have less agents. As for the rest of principal factors, we observe similar tendencies than for sequential reads.

Random Writes
In general, the trends for random writes are similar to those for random reads. There is only one significant interaction as shown in Figure 4, between the number of agents per disk and the number of buffers per disk.

The plot in Figure 4 shows an asymptotic behavior of the performance for each number of buffers per agent, as the number of agents per disk grows. Also, the asymptote seems to converge although for 1 buffer, there is a significant distance to the 32 buffer plot. In this case, the behavior is significantly different than that of sequential writes for the same combination of factors (Figure 2). In that case, there was no convergence to an asymptote whatsoever.

5. RELATED WORK
There is a significant amount of work on workload generation for Unix-like systems as explained in [3]. However, most of those pieces of work do not have all the features IOAgent offers and we have used here.
Also, we have used our tool to perform a more extended performance analysis in [3]. There, we use two systems larger and closer to highly stressed commercial environments.

On the other hand, the Linux AIO interface is very recent. Therefore there are not studies of the impact of this new feature as far as we know. Moreover, we are not aware of any study of the Linux AIO performance on Power-based architectures.

6. CONCLUSIONS

In this paper, we have built an statistical model of the performance of the I/O subsystem of an OpenPower 710 architecture. The model has taken into account four factors and their interactions. The four factors we have taken into account are (i) the number of disks used, (ii) the number of processes or agents issuing I/O operations, (iii) the number of application buffers used for each of the agents and (iv) the size of those buffers.

We have used IOAgent to build a set of four workloads forcing asynchronous read and write activity. The workloads and factors tested have provided a total of more than 1000 performance measures that allowed us to get some interesting conclusions.

First, the number of agents has a significant impact on the final performance depending on the type of I/O activity performed. In the case of sequential activity, as the number of agents grows the transfer rate decreases. In the case of random activity, as the number of agents grows the transfer rate increases. Second, the interaction between the number of agents and number of buffers is significant. For sequential accesses, a growing number of agents implies a moderate decrease in the performance for a fixed amount of buffers. Also, the as the number of buffers grows, the performance tends to saturate which is rather stable no matter the number of agents. On the contrary, the interaction of those factors on random operations is rather different, showing a convergence for all the number of buffers as the number of agents grows.

In general, we can say that the use of a statistical model helped us to be more precise in understanding the behavior of the I/O subsystem of the OpenPower 710 under Linux.

7. ACKNOWLEDGEMENTS

Thanks to the IBM Toronto Lab Center for Advanced Studies (CAS) and the Departament d’Arquitectura de Computadors at Universitat Politècnica de Catalunya (UPC). This work has been possible thanks to a Linux on Power Faculty Award received from IBM by Josep L. Larriba-Pey. We want to thank Marta Pérez-Casany for her teachings in statistics.

8. BIOGRAPHIES

Sergio Gómez-Villamor is a Ph.D. student in the Department of Computer Architecture at the Universitat Politècnica de Catalunya (UPC) and a CAS student at the IBM Toronto Lab. His interests include Power-based architectures, performance of DBMS and Linux OS.

John B. Tran is an experienced developer of DB2 UDB at the IBM Toronto Lab.

Steve A. Rees is an experienced developer of DB2 UDB at the IBM Toronto Lab.

Victor Muntés-Mulero is a Ph.D. student in the Department of Computer Architecture at the Universitat Politècnica de Catalunya (UPC). His interests include optimization of large join queries, performance of DBMS and non-deterministic algorithms.

Josep L. Larriba-Pey is Associate Professor at Universitat Politècnica de Catalunya since 1996. His interests are in the area of DBMS performance for sequential and parallel systems, parallel and memory hierarchy conscious non-numeric algorithms and sequential and parallel record linkage.

9. REFERENCES

