Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds

MODELING AND SIMULATING THE EFFECTS OF OS JITTER

Elder Vicente
Rivalino Matias Jr.

School of Computer Science
Federal University of Uberlandia
Uberlandia, BRAZIL

ABSTRACT

The phenomenon of operating system (OS) Jitter has been investigated and considered a critical factor in high-performance computing. In this paper we model and simulate the effects of different sources of OS Jitter in the Linux operating system. We adopt the design of experiments approach to conduct experiments statistically planned. Our simulation models corroborate the results obtained experimentally. We conclude that the OS Jitter has a higher impact when the number of the computational phases is high, for any number of computing nodes from 1 to 500. We also observed that in Linux, the highest OS Jitter impacts are caused by managing the shared processor cache and network interrupts, where the second shows the highest sensitivity with respect to the cluster size.

1 INTRODUCTION

Clusters of computers are extensively used for running high-performance computing (HPC) applications (Cheon, Kim, and Im 2003). In a typical HPC cluster, each computing node executes its own operating system (OS). Thus, in addition to the user's application running on the node, there are also OS routines executing on the same node. This means that OS routines such as synchronization of dirty buffer-cache entries, interrupt handlers, kernel timers, and administrative processes, all of them compete with the user application for the node computing resources. It leads to a scenario where during the user application runtime it periodically suffers from interferences caused by the OS internal routines. These interferences have been extensively investigated (Garg and De 2006; De, Kothari, and Mann 2007; Gioiosa et al. 2004; Agarwal, Garg, and Vishnoi 2005; Tsafrir et al. 2005; T.R. Jones and Fier 2003) and reported as OS Jitter effects.

Importantly, HPC cluster-based applications are designed to run in a parallel processing paradigm, where instructions are programmed to execute in many computational phases (Garg and De 2006). Right after all distributed processes finish a given computational phase, they synchronize and start the subsequent phase (Gioiosa et al. 2004; Agarwal, Garg, and Vishnoi 2005; Tsafrir et al. 2005). A new phase only starts after all processes conclude the current phase, so synchronizing the computing time of all processes is critical. The last process that finishes a given phase determines the length of the current phase. Reducing the runtime variability in each node is a major requirement, because the occurrence of unexpected delays in a given node will spread along other nodes involved in the same computational phase, taking a longer time to complete the whole task. Many factors cause the application runtime variability in a cluster environment, specially network and disk I/O latencies, and the OS Jitter (Petrini, Kerbyson, and Pakin 2003).

In this paper, we model and simulate the effects of different sources of OS Jitter in the Linux operating system. We choose the Linux OS because it is used in more than 90% of the HPC clusters listed on the Top500 supercomputer website (Top 500 Supercomputer Sites 2013). We apply controlled experiments in order to collect the experimental data sets used to characterize the effects of each modeled source of OS Jitter. The data sets are analyzed using different parametric and non-parametric statistical

techniques. Since we execute different experiments, we also implement specific instrumentations to compare the control group against the planned experimental groups, considering the evaluated factors individually and in a combined way. The rest of this paper is organized as follows. Section 2 describes related works. Section 3 presents the methodology used in this study. Section 4 shows our experimental plan. Section 5 discusses the experimental results. Section 6 presents the simulation modeling and results. Section 7 present our final remarks.

2 RELATED WORKS

In De, Kothari, and Mann (2007), the authors present an experimental study of OS Jitter in the Linux operating system. They reported that 63% of every observed delay are caused by the timer interrupt, and the remaining 37% came from different OS services and other hardware interrupts. Similarly, In Tsafrir et al. (2005) the authors also concluded that the main source of OS Jitter was the timer interrupt. In Agarwal, Garg, and Vishnoi (2005), a research work that models the effect of OS Jitter over the scalability of parallel applications is presented, which shows that in presence of OS Jitter with exponential distribution, the system under study showed the expected scalability. Gioiosa et al. (2004) discusses the impact of the OS Jitter on parallel applications, where the authors implemented a micro benchmark that executes *n* computational phases calibrated to execute in a certain amount of time. Results for a micro benchmark of 1000µs indicated delays between 0.5µs and 1.4µs. They verified that these delays were related to the programmable interrupt timer, local timer interrupts, and network card interrupts. In Jones and Fier (2003), the impact of the OS Jitter on the scalability and performance of parallel applications in large clusters is investigated. The most important interferences observed were caused by system processes and OS kernel routines. In Fröhlich, Gracioli, and Santos (2011), two strategies for programming the timer interrupt in APIC-based systems are evaluated, which are one-shot and periodic. They implemented an event-driven model for timer interrupts using one-shot timers, where the timer programming is based on the time interval for the next event. Their experiments showed that in terms of accuracy and interferences, the periodic timer interrupt is comparable to one-shot, and in some cases is even better.

Most of the above-mentioned works focused on analyzing the OS Jitter experimentally. In this work we execute experiments to characterize the Jitter phenomenon and then model and simulate it. We consider three different HPC scenarios under the influence of main OS Jitter sources. We simulate all evaluated scenarios varying the cluster size, number of computational phases, and the workload profile per node.

3 METHODOLOGY

For the experimental study, we adopt the design of experiments (DOE) method (Montgomery 2000). We apply it to measure the impact of different sources of OS Jitter on the execution time of a typical HPC application. The HPC application used is a CPU-bound program that performs a matrix multiplication algorithm. Our control group is composed of all treatment executions where the sources of OS Jitter are present, as they originally manifest in a typical HPC environment. The experimental groups are the treatments that we control the presence and levels of investigated sources of OS Jitter. Each treatment test executes according the following protocol: *i*) setup the test bed according to the treatment specification; *ii*) collect the start time (T_1); *iii*) execute the matrix multiplication algorithm; *iv*) collect the end time (T_2); *v*) replicate steps two to four 53 times; *vi*) write all computation times, $(T_2 - T_1)_{i=1..53}$, into a log file. We replicate the treatments in order to have a sample size large enough to ensure a proper estimation of experimental errors. The runtime of step three is approximately 10 minutes in average. To guarantee the independency of each treatment execution, we restart the OS kernel right before starting the execution of a new treatment, making sure that each treatment test starts in a renewed OS environment. An important issue on measuring computation time is the dataset variability. For each treatment, we discard the first three replications considering that their results are more likely to suffer influences from file system and processor caches. Our final dataset per treatment is composed of 50 run times. Another procedure adopted is turning off the automatic CPU frequency regulation. This feature allows the Linux kernel to change dy-

namically the processor frequency, affecting the run time length of the application processes. To analyze the experimental data, first we identify which treatments are statistically different. We do not use parametric approaches, such as analysis of variance (ANOVA) and multiple comparisons, because the dataset obtained does not fit to the necessary assumptions, especially regarding to independent and identically distributed observations. Thus, we use the non-parametric Kruskal-Wallis test (Kvam and Vidakovic 2007), which allows us to use ranks of observations providing statistics equivalent to those obtained with ANOVA and other parametric tests. We compare all treatments and the differences between their response variables are statistically significant if *p-value* is less than 0.05. For the setup of treatment combinations we adopt the signal matrix method (Jain 1991) that is arranged according to the Yates' order (Montgomery 2000). Solving the signal matrix we have a ranking of individual and combined factors that are sorted by their influence degree on the application runtime. Supported by this ranking we can identify the OS Jitter sources with more impact on the test application. For the simulation modeling, we use the experimental data of specific treatments to obtain the density functions used to simulate the delay probability in a computing node for a given computational phase (see Section 6).

4 EXPERIMENTAL STUDY

4.1 Test-bed and Instrumentation

In order to conduct the experiments we use a test bed based on a computer composed of two quad-core sockets (Intel Xeon E5620 2.40GHz), 24 GB memory, and 1 TB SATA disk. The computer microarchitecture has a three-level cache per CPU socket, being the last level (L3) of 12MB and shared by all cores of the same socket. Each core has two individual levels of cache, L1 (32KB) and L2 (256KB). Figure 1 illustrates the processors topology of our test bed machine.

We refer to each core as PU #0 to PU #7, where PU stands for processor unit. The test program runs only on PU #1, where we control the enabling and disabling of OS Jitter sources. The remaining cores are used according to each treatment specification. We encode each evaluated factor using upper case letters (e.g., A, B, …). Each factor assumes two levels represented by symbols (+) and (-). The level (-) means that the OS Jitter source represented by that factor is disabled, and (+) means enabled. Since we adopt a factorial design, if there are k factors, each at two levels, a full factorial design with replication results in 2^k treatments, where each treatment test is replicated r times, given a total of $2^k \times r$ runs. The rest of this section presents the details for each experiment.

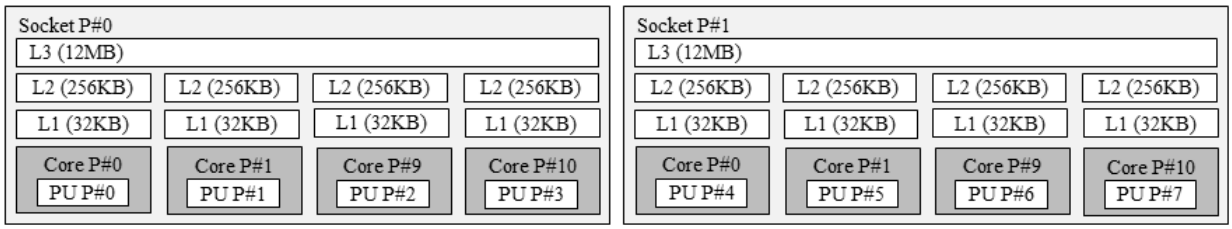

Figure 1: Processor topology of the test-bed machine

4.2 Design of Experiment #1

Table 1 shows the experimental plan for experiment #1. It evaluates five factors, where factor A represents the OS runlevel (Van Vugt 2006) that defines how the OS services are loaded during the system startup. At level (-) the runlevel is 5 (highest number of service loaded), and the level (+) sets a minimal number of services loaded. The Jitter effect of this factor is related to the number of OS services running concurrently with the user application and thus competing for the node's computing resources. Factor B represents the kernel timers, which are used to allow the execution of kernel or user level routines at a given future time. As discussed in Section 2, previous works have considered timers as a source of OS Jit-

ter. At level (-) we disable the execution of timers on the same processor (PU #1) that executes the test application, avoiding possible noises caused by their execution. The level (+) means that kernel or user-level timers can run on the same processor executing the test application. In our experiments, we always move timers from PU #1 to PU #0, where PU #0 is the processor we defined to run all timers from PU #1 when this factor is at level (-). This approach allows us to observe the direct interference of timers on the test application. Factor C represents the hardware interrupt request (IRQ). At level (-) this factor indicates that the processor PU #1 cannot receive interrupt requests (except from the timer interrupt); we redirect all IRQs to PU #0. At level (+) it indicates that all IRQs are handled only by the PU #1. We managed to control this factor by using the functionality of "SMP IRQ affinity" available in the Linux kernel. Factor D represents the processor affinity of the system processes. This factor at level (-) means that processor affinity is disabled, and thus all OS services can be executed in any processor; so they can be scheduled to run on the processor (PU #1) where is running the test application. At level (+) we set all system services to run only on PU #0. This allows us to observe the direct interference of system processes, in a given runlevel, on the test application. Factor E represents the timer interrupt. This factor at level (-) indicates that we disable this interrupt on processor PU #1, where the test application is running. We disable the timer interrupt right before performing the matrix multiplication routine and next we enable it.

Table 1: Factors and levels of first experiment.

			Level (-)	Level (+)
Factors	A	Runlevel	5	1
	B	Kernel Timers	Off	On
	C	IRQ	Off	On
	D	Processor Affinity	Off	On
	E	Timer Interrupt	Off	On

4.3 Design of Experiment #2

It consists of six factors. The first five factors are the same described in Exp. #1, so we introduce the factor F that represents a CPU-bound workload running in background. This background load is also a process running a matrix multiplication program. The factor F at level (-) means that the application performing the background workload is running in a processor (PU #5) that is not sharing L3 cache with the processor PU #1, where the test application is running. This factor at level (+) means the opposite; i.e., no L3 cache sharing between PU #2 and P #1. This allows us to observe the interference of other processes sharing processor cache memory with the test application. For this purpose we carefully control the working set size of each process to make sure that both working sets are large enough to fill out the entire L3 cache (12 MB), which means that when evaluating the scenario with shared cache (level +), both processes compete for the entire L3 cache memory. We consider the influence of cache as a source of OS Jitter because the OS manage the memory cache in various ways (e.g., cache aware scheduling). The signal matrix for Exp. #2 follows the same rationale than for Exp. #1, however the additional sixth factor results in 64 (2^6) treatments. We suppress this table due to the page limits of this paper.

4.4 Design of Experiment #3

It introduces a network workload in addition to the factors evaluated in Exp. #1. This background workload allows us to observe the interference of network interrupts on the test application. For all evaluated treatments, the network workload runs on PU#2. The network workload is based on an application receiving 500-byte UDP datagrams in a continuous way. In this experiment, some treatments tested in experiments #1 and #2 were not evaluated, which are related to the IRQ factor in level (+) and timer interrupt factor in level (-). This is necessary because disabling the timer interrupt on PU#1 makes the kernel routines, responsible for the datagram packet processing, work improperly, which causes the loss of network

packets. It occurs because these routines use kernel timers that require the timer interrupt enabled. The same applies to the IRQ with respect to the network card interrupt handling.

5 EXPERIMENTAL RESULTS

Based on graphical and numerical analyses we identified and removed experimental errors considered outliers. This removal procedure replaces the outlier to another value that is calculated averaging the remaining (non-outlier) values of the same treatment dataset. For all experiments, the number of outliers was very small: Exp. #1 (2.68%), Exp. #2 (2.15%), Exp. #3 (2.33%). Based on the outlier-free datasets, the rest of this section presents the most important findings of our experimental study.

5.1 Result Analysis of Experiment #1

Figure 2 presents the average run time of each treatment. We can observe that the variability of the run times between the 1st and 16th treatments is quite lower than that observed between the 17th and 32nd treatments. After the 16th treatment, there is a raise in the average run time, which is caused by enabling factor E (timer interrupt). This is an evidence of the influence of factor E on the runtime variability. Next, we calculate the percentage that each factor, individually and combined, contributes to the variation of the test application run time. Table 2 shows the percentage of contribution of the three most influential factors. We found that 91.23% of the test application run time variation is caused by factor E (timer interrupt). As can been seen, the other factors did not show important contributions when compared with the timer interrupt. Note that 6.70% are not explainable by any factor or their interactions. This may be due to experimental errors or factors that were not considered in our experimental plan.

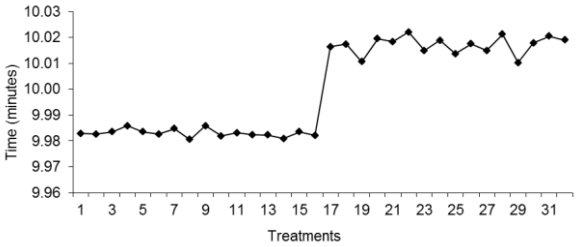

Figure 2: Average runtimes for Exp. #1

Table 2: Factors' influence on the runtime variation for Exp #1

Factor / Interactions	Percentage of Contribution
E	91.2349
AE	0.6046
BDE	0.2707

5.2 Result Analysis of Experiment #2

The analysis conducted for Exp. #2 follows the same procedures than for Exp. #1. Figure 3 presents the average run time of each treatment in Exp. #2. We split the treatments in four groups (G1 to G4). Firstly, we observe that the variability of the run times increases according to the group. G1 and G2 reproduce the treatments evaluated in Exp. #1, so the results are practically the same presented in Section 5.1. The factor F (shared L3 cache) is disabled in all treatments of G1 and G2, and enabled in all treatments of G3 and G4, where in G3 the factor E (timer interrupt) is disabled and in G4 it is enabled. This means that in all treatments of G3 the test application did not suffer influence of timer interrupts, but from sharing the L3 cache. In G4 both influences, timer interrupts and sharing processor cache, are present. Based on the graphical analysis, we conclude that the individual contribution of factors E and F on the application run time are very similar. The numerical analysis corroborates the graphical analysis, showing that the aver-

age run time in G2 (10.0179 minutes) and G3 (10.0235) are very close. Next, we calculate the percentage each factor and its interactions contribute to the variation of the application run time. Table 3 shows the values for the top three causes (responsible for approx. 60%) of variability on the application run time. Note that 32.84% of the total variability could not be explained by our factorial design. This may be due to experimental errors introduced with the activation of factor F. The Kruskal-Wallis test shows that for pairs of treatments in the same group there are no significant differences, except in pairs 33-34, 33-36, and 33-37, from G3. Comparing treatments in G2 to treatments in G3 we obtain only 4% of the comparisons considered as statistically significant. The comparisons among treatments from G1-G2, G1-G3, G1-G4, G2-G4, and G3-G4 show statistically significant differences.

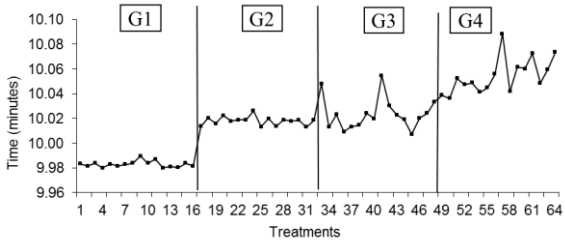

Figure 3: Average runtimes for Exp. #2

Table 3: Factors' influence on the runtime variation for Exp. #2

Factor / Interaction	Percentage of Contribution
F	33.90%
E	24.76%
DF	0.90%

Table 4: Comparison of intergroup treatments for Exp. #2

	G1	G2	G3	G4
G1	0.0%	98.0%	97.3%	100.0%
G2		0.0%	4.3%	75.4%
G3			7.5%	66.8%
G4				0.0%

5.3 Result Analysis of Experiment #3

Figure 4 presents the average run time of each treatment in the third experiment. The group G2 represents the treatments not evaluated, as mentioned in Section 4.4. In G1 the factors C (IRQ) and E (timer interrupt) are disabled in all treatments. Factor E is enabled and the factor C is disabled in all treatments of G3. Finally, in G4 all treatments have both factors (C and E) enabled. We observe that the joint contribution of factors C and E on the application run time is high. However, when the hardware interrupt request and timer interrupt factors are enabled simultaneously on PU#1, shown in G4, the average run time increases significantly (45%). In this case, the average run time of G4 in Exp. #3 is higher than in G4 of Exp. #2; i.e., the network interrupts may have a greater impact than the worst case of sharing cache. Next, we calculate the percentage each factor and its interactions contribute to the variation of the application run time. Table 5 shows the values for the top three causes (responsible for approx. 99%) of variability on the application run time. We observe that factor C and the iteration CE have very close contributions (approx. 46%). In this experiment 0.003% of the total variability could not be explained by our factorial design, probably due to experimental errors. Next, we perform the Kruskal-Wallis test and verified that there are statistically significant differences between the treatments of this experiment. We compare all pairs of treatments and the results are shown in Table 6. All comparisons between groups (G1-G3, G1-G4, G3-G4) are considered statistically significant.

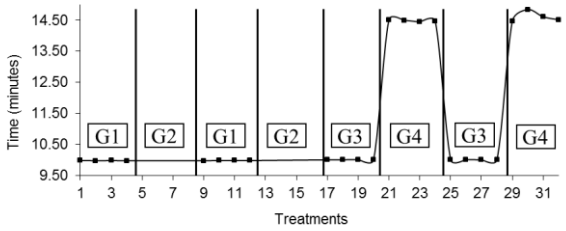

Figure 4: Average run time for Exp. #3

Table 5: Factors' influence on the runtime variation for Exp. #3

Factor / Interaction	Percentage of Contribution
E	46.89%
CE	46.51%
C	6.57%

Table 6: Comparison of intergroup treatments for Exp. #3

	G1	G3	G4
G1	0.0%	93.7%	100.0%
G3		0.0%	91.6%
G4			0.0%

6 SIMULATION RESULTS

6.1 Simulation #1

Based on the datasets obtained in the experiments, we modeled and simulated the effects of the OS Jitter on a HPC application executed as multiple computational phases, and running in a cluster of compute nodes. We used treatments T10 and T23, from Exp. #1, to represent the scenarios with no OS Jitter effect (T10) and fully affected by OS Jitter (T23), respectively. Based on the differences between the T23 and T10 run times, we generated a new dataset to obtain the probability density function (*pdf*) of the run time delay caused by OS Jitter. Based on a Kolmogorov-Smirnov test, with 95% of confidence level, we found that this sample follows a Normal distribution, N(μ=1.91, σ=0.36). Next, we used the estimated *pdf* to simulate the delay occurrences on each computational phase per process running on multiple computing nodes. We vary the number of computational phases per process (1 to 200) and the number of nodes (1 to 500). The simulation results show that when we increase the amount of compute nodes, to any amount of phases, the application execution time grows logarithmically (see Figure 5). For few nodes (e.g., < 20) the growth of the curve is quite sharp; for more nodes the increase in the application time tends to moderate. This happens because with few processes (nodes) taking part at each phase, there is a smaller probability that in a given phase some of these processes suffer from OS Jitter influences whose delay is close to the highest possible delay values. If the amount of nodes rises, then this scenario increases the probability of delays caused by OS Jitter, per phase, to be close to the highest observed delay. Differently, when raising the number of phases the application run time rose linearly. Figure 6 illustrates this behavior. Increasing the number of phases the probability of delays caused by OS Jitter inside of each node also increases. Since the nodes are working in parallel, the summation of these increased probabilities explains this linear behavior. Figure 7 summarizes this result presenting a sensitivity analysis of the runtime delay with respect to the number of nodes and number of phases.

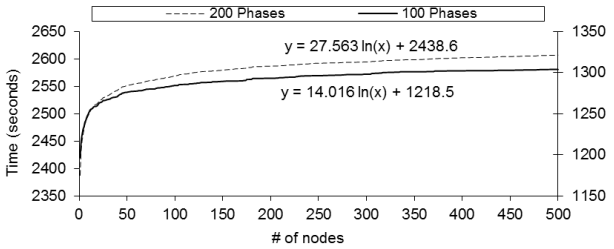

Figure 5: Effect of OS Jitter per # of phases and # of nodes

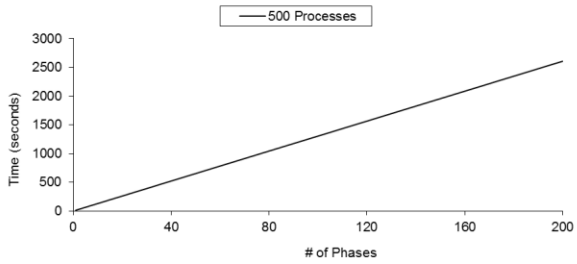

Figure 6: Effect of OS Jitter for 500 nodes and multiple phases

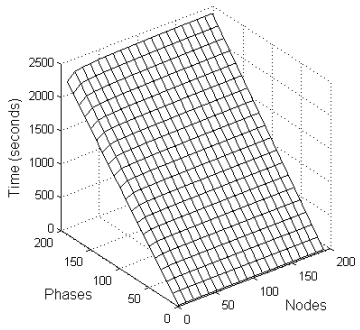

Figure 7: Runtime sensitivity for simulation #1

6.2 Simulation #2

In simulation #1 we consider only one application per node. Now, in addition to the test application we also consider a background workload running in the same node and sharing the L3 processor cache. For modeling this scenario, we select T10 and T55 from Exp. #2. We use the same procedure adopted in Section 6.1 to obtain the *pdf*. This new dataset also follows a Normal distribution according to a Kolmogorov-Smirnov test with 95% of confidence level (see Figure 8). Based on the estimated *pdf*, we simulate the delay occurrences on each computational phase per process running on multiple computing nodes under the influence of the background workload. We also vary the number of computational phases per process (1 to 200) and the number of nodes (1 to 500). Similarly to simulation #1, for different number of computational phases we observe that for few processes the growth of the curve is quite strong (see Figure 9). For more than that, the increase in the application time tends to smooth. Due to the background workload is sharing L3 cache with the test application, we observe that now it is necessary more compute nodes for the delays caused by OS Jitter, per phase, to be close to the highest observed delay. Comparing the results of simulations #1 and #2, we may observe the longer delay in #2 caused by the additional Jitter effect related to the L3 cache sharing between the test application and the background workload, which corroborates the experimental results. This additional Jitter effect represents the influence of the OS managing the successive cache misses during the test application execution. In order to evaluate the amount of

influence of this Jitter effect isolate, we compute the difference between the OS Jitter effects of simulation #1 and simulation #2, for 100 and 200 phases with multiples computing nodes. We observe that according the number of nodes, we obtain a logarithmic variation in the runtime from 13% to 31% (see Figure 10). It means that in a cluster, the effect of factor F (shared L3 cache) increases logarithmically according to the number of nodes.

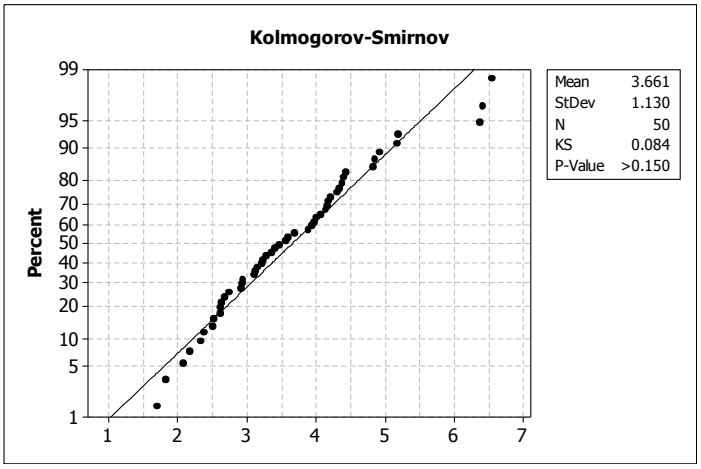

Figure 8: Goodness of fit of Normal distribution (simulation #2)

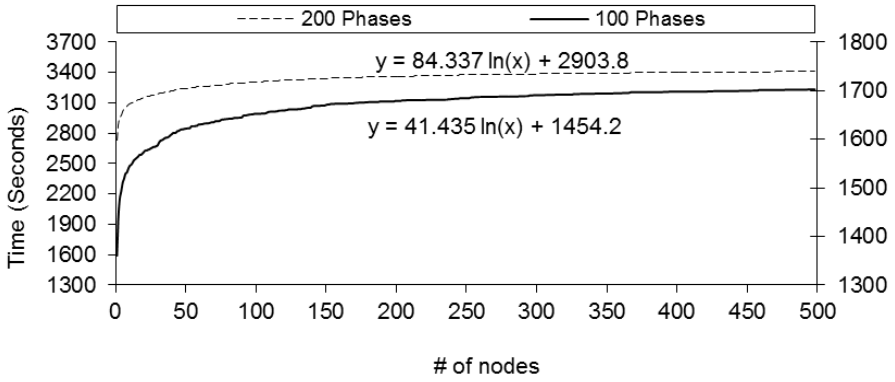

Figure 9: Effect of OS Jitter with background workload sharing L3 cache

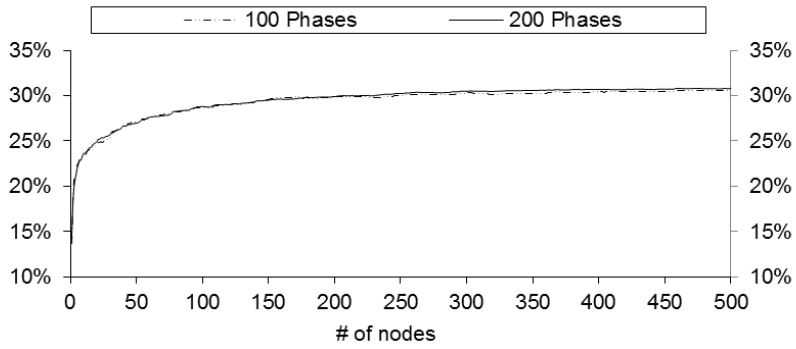

Figure 10: Percentage of influence of "shared cache" on runtime per number of nodes

6.3 Simulation #3

For this simulation scenario we estimate the delay probability density function based on treatments T10 and T23 of experiment #3. It considers the interference of network interrupts on the test application runtime. In terms of OS Jitter, T10 has all factors disabled and T23 all factors enabled including the network interrupts (see subsection 4.4). We also obtained an adequate goodness of fit for the normal distribution (see Figure 11). Hence, we use the estimated *pdf* to simulate the delay occurrences on each computational phase of each application process running on multiple computing nodes. We again vary the number of computational phases per process (1 to 200) and the number of nodes (1 to 500). As in the previous simulations, for different number of computational phases we observe that for few processes the growth of the curve is quite strong (see Figure 12), and it tends to moderate as the number of nodes increase. Comparing this simulation with simulations #2 and #1, it clearly has the longest delay, which corroborates the experimental results presented in Section 5. Considering the three simulation results for 100 phases and 500 nodes, we can clearly observe that the factor network interrupts is more sensitive to the number of nodes than any other Jitter effect evaluated (see Figure 13). This model behavior is desired, given that it represents well scenarios where the number of nodes grows and consequently increasing the number of processes communications, which consequently causes a higher number of network interrupts.

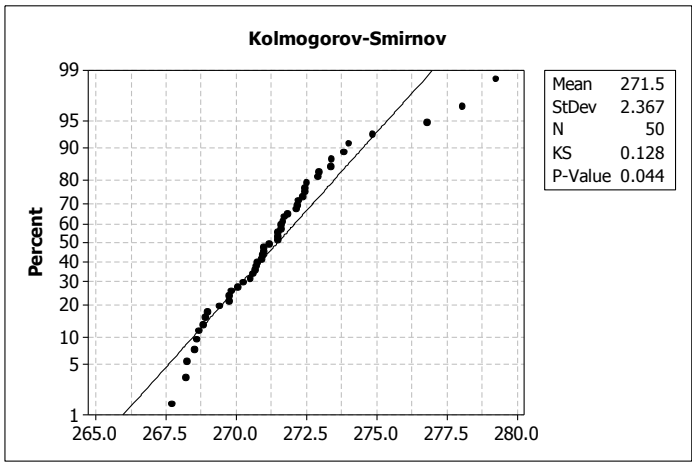

Figure 11: Goodness of fit of Normal distribution (simulation #3)

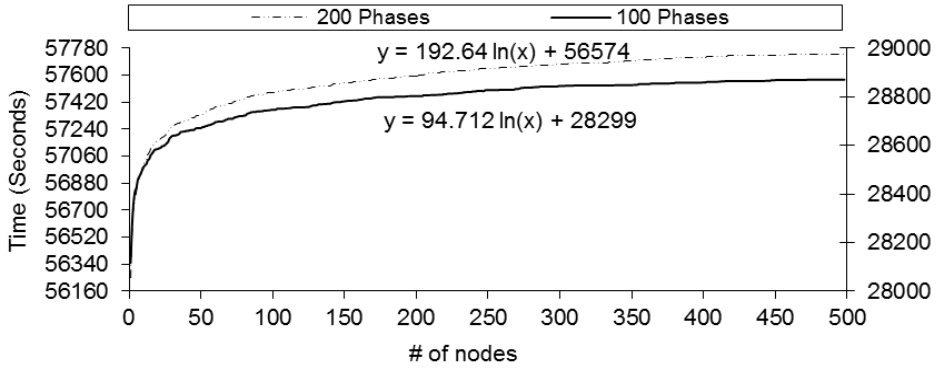

Figure 12: Effect of OS Jitter with background network workload

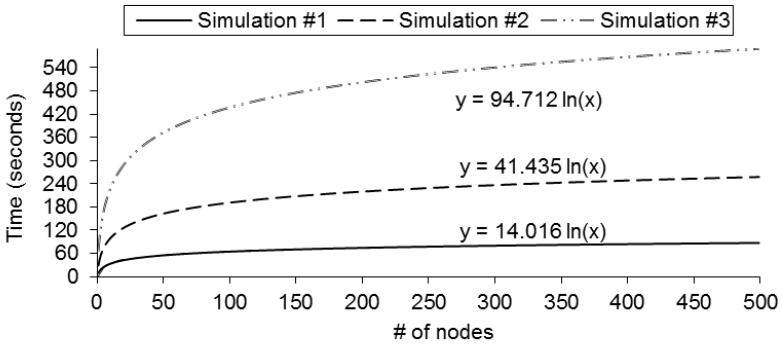

Figure 13: Comparison of simulations #1, #2, and #3 for 100 phases and multiple nodes

7 FINAL REMARKS

The recent advances in areas such as power saving and processor topology have changed the way the operating systems work. These changes consequently affect how the OS routines interfere on the user applications. The combination of different hardware and OS platforms present a challenging scenario to evaluate the impact of different sources of OS Jitter in a given computing scenario. Modeling and simulation allow us to deal with this complexity.

In this paper we present a practical approach to model and simulate the OS Jitter effects in a typical HPC cluster. To illustrate our approach we consider three different simulation scenarios. The simulation results corroborate the experimental data obtained through controlled experiments. The simulation models allowed us to understand the behavior of the evaluated source of OS Jitter under different cluster sizes and for different number of computational phases, which would be unfeasible by experiments.

We conclude that the OS Jitter has a higher impact when the number of computational phases is high, for any number of nodes from 1 to 500. In terms of sources of OS Jitter, we also observed that in Linux the highest OS Jitter impacts are caused by managing the shared processor cache and network interrupts, where the second shows the highest sensitivity with respect to the cluster size.

ACKNOWLEDGMENTS

This work was supported partially by CNPq (National Council for Scientific and Technological Development), CAPES (National Council for the Improvement of Higher Education of Brazil) and FAPEMIG (Research Foundation of Minas Gerais State).

REFERENCES

Agarwal, S., R. Garg, and N. K. Vishnoi. 2005. "The impact of noise on the scaling of collectives: a theoretical approach". In *Proceedings of the 12th international conference on High Performance Computing*, HiPC'05, 280–289. Berlin, Heidelberg: Springer-Verlag.

Cheon, J., S. Kim, and Y. Im. 2003. "Three-dimensional bulk metal forming simulations under a PC cluster environment". *Journal of Materials Processing Technology* 140 (1-3): 36 – 42.

De, P., R. Kothari, and V. Mann. 2007. "Identifying sources of Operating System Jitter through fine-grained kernel instrumentation". In *Cluster Computing, 2007 IEEE International Conference on*, 331–340.

Fröhlich, A. A., G. Gracioli, and J. F. Santos. 2011. "Periodic timers revisited: The real-time embedded system perspective". *Computers and Electrical Engineering* 37 (3): 365 – 375.

Garg, R., and P. De. 2006. "Impact of noise on scaling of collectives: an empirical evaluation". In *Proceedings of the 13th international conference on High Performance Computing*, HiPC'06, 460–471. Berlin, Heidelberg: Springer-Verlag.

Gioiosa, R., F. Petrini, K. Davis, and F. Lebaillif-Delamare. 2004. "Analysis of system overhead on parallel computers". In *Signal Processing and Information Technology, 2004. Proceedings of the Fourth IEEE International Symposium on*, 387–390.

Jain, R. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley Professional Computing*. Wiley.

Jones, T. R., L. B. Brenner, and J. M. Fier. 2003. "Impacts of Operating Systems on the Scalability of Parallel Applications". Technical Report UCRL-MI-202629, Department of Energy by University of California, Lawrence Livermore National Laboratory, California.

Kvam, P., and B. Vidakovic. 2007. *Nonparametric Statistics with Applications to Science and Engineering. Wiley Series in Computational Statistics*. Wiley.

Montgomery, D. 2000. *Design and analysis of experiments*, 3rd edition. John Wiley.

Petrini, F., D. Kerbyson, and S. Pakin. 2003. "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q". In *Supercomputing, 2003 ACM/IEEE Conference*, 55–55.

Top 500 Supercomputer Sites 2013. "Top 500 Supercomputer". Accessed April 3, 2013. http://www.top500.org/.

Tsafrir, D., Y. Etsion, D. G. Feitelson, and S. Kirkpatrick. 2005. "System noise, OS clock ticks, and fine-grained parallel applications". In *Proceedings of the 19th annual international conference on Supercomputing*, ICS '05, 303–312. New York, NY, USA: ACM.

Van Vugt, S. 2006. *The Definitive Guide to SUSE Linux Enterprise Server*. Apress.

AUTHOR BIOGRAPHIES

ELDER VICENTE received his B.S.(2007) in control and automation engineering from the Polytechnic School of Uberlandia, Brazil. He earned his M.S.(2012) degree in Computer Science from the Federal University of Uberlandia. His research interests include techniques for optimizing and evaluating the performance of operating systems. His email address is elder@mestrado.ufu.br.

RIVALINO MATIAS JÚNIOR received his B.S. (1994) in informatics from the Minas Gerais State University, Brazil. He earned his M.S. (1997) and Ph.D. (2006) degrees in computer science, and industrial and systems engineering from the Federal University of Santa Catarina, Brazil, respectively. He is currently an Associate Professor in the Computer School at Federal University of Uberlandia, Brazil. Dr. Matias has served as reviewer for several international journals. His interests include computing systems engineering, software aging theory, and diagnosis protocols for computing systems. His e-mail and web addresses are rivalino@fc.ufu.br and http://hpdcs.facom.ufu.br/, respectively.