Evaluating Machine-Independent Metrics for State-Space Exploration

Vilas Jagannath, Matt Kirn, Yu Lin, and Darko Marinov
Department of Computer Science, University of Illinois at Urbana-Champaign
Urbana IL, 61801, USA
Email: {vbangal2,kirn1,yulin2,marinov}@illinois.edu

***Abstract*—Many recent advancements in testing concurrent programs can be described as novel optimization and heuristic techniques for exploring the tests of such programs. To empirically evaluate these techniques, researchers apply them on subject programs and capture a set of metrics that characterize the techniques' effectiveness. From a user's perspective, the most important metric is often the amount of real time required to find a error (if one exists), but using real time for comparison can be misleading because it is necessarily dependent on the machine configuration used for the experiments. On the other hand, using machine-independent metrics can be meaningless if they do not correlate highly with real time. As a result, it can be difficult to select metrics for valid comparisons among exploration techniques.**

This paper presents a study of the commonly used machine-independent metrics for two different exploration frameworks for Java (JPF and ReEx) by revisiting and extending a previous study (Parallel Randomized State-Space Search) and evaluating the correlation of the metrics with real time both on a single machine and on a high-performance cluster of machines. Our study provides new evidence for selecting metrics in future evaluations of exploration techniques by showing that several machine-independent metrics are a good substitute for real time, and that reporting real time results even from clusters can provide useful information.

I. Introduction

With the prevalence of multicore processors, more concurrent programs are being developed and used. However, concurrent programs are difficult to develop and notorious for having hard-to-find and hard-to-reproduce errors. It is challenging for developers to avoid such errors due to the complexity of reasoning about an enormous set of possible interleavings that are not immediately intuitive from the code itself. Concurrent programs are also difficult to test as it is necessary to check how a program behaves not only for different inputs but also for different interleavings for a given input. As such, it is important to develop and evaluate techniques that test concurrent code effectively.

Concurrent programs are often tested using tools that systematically explore some or even all possible interleavings of a given program for a given input—this approach is known as *state-space exploration* [1]. Conceptually, the exploration begins from a start state, executes the program up to a point where a set of non-deterministic choices $c_1, \ldots, c_n$ are possible (e.g., n threads are enabled so either one of them could proceed first), selects one of the choices c_i according to some strategy, continues exploration based upon that choice until a particular criteria is satisfied, and then backtracks to c_i in order to explore the choices extending from c_{i+1}. This approach can result in a huge number of interleavings to explore (e.g., for n enabled threads, there could be up to $n!$ different results of execution based on the orderings of these threads)—this problem is known as state-space explosion.

Since state-space exploration is essentially a search over the state space, tools can adopt different *search strategies* (e.g., depth-first, random, best-first) to perform the exploration. Tools can also employ different methods to restore states from which other choices were available, to explore those remaining choices. Some tools *checkpoint* encountered states and restore states using the checkpoints. Other tools store the choices that lead to encountered states and *re-execute* those choices to restore states. While checkpointing states can be memory and time intensive, hashes of state checkpoints can be used to efficiently remember previously explored states. Tools that remember previously explored states are called *stateful*, and they can reduce exploration time by avoiding re-exploration of states. Tools that do not remember previously explored states are called *stateless*, and they can reduce exploration time by avoiding costly hash comparisons for all encountered states.

Developing new techniques for faster state-space exploration has received much attention in research, e.g., [2]–[8]. The techniques can be categorized in many different ways. One category of techniques consists of strategies that prioritize or select the exploration of certain parts of the state space by leveraging additional information or heuristics [4], [6]–[12]. A recent, promising strategy in this category is known as *Iterative Context-Bounding (ICB)* [4], which prioritizes exploration according to the number of preemptive context switches between threads. The main idea is to explore the parts of the state space that consist of a smaller number of context switches first, because many concurrency errors often manifest in such schedules. We use an implementation of the ICB strategy in the ReEx framework [8], [13] for some of our experiments. Other techniques include those that parallelize a single exploration by partitioning the exploration into non-overlapping subspaces [14], [15] and those that exploit diversity among different (potentially overlapping) complete explorations by performing them in parallel [3], [5].

To empirically evaluate these techniques, researchers apply them on subject programs, capture a set of metrics, and compare the values to assess the techniques' effectiveness. Effectiveness usually has two dimensions. One dimension is the level of assurance (soundness/completeness) of the technique with respect to the detection of a particular property (e.g., deadlock). The second dimension is whether a technique finishes quickly. A user will have to allocate precious resources that will be consumed to perform the state-space exploration, so a technique that provides high assurance in a short amount of real time will consume less of a user's resources. However, real time is only a valid measurement for a given machine configuration (e.g., CPU/memory/disk/network speed, task scheduling algorithm, garbage collection/disk paging, etc.). As such, research studies that provide evaluations of real time are necessarily machine-dependent, and thus their results may not apply across different machine configurations, may be hard to repeat, and may be difficult to compare with other techniques. Studies that perform large experiments for comparing techniques on clusters of machines are further impacted by this concern since the results from clusters may not necessarily allow comparisons of real time.

The issues with real time lead researchers to use machine-independent metrics that allow comparison of techniques across different machine configurations. Empirical studies in the literature often vary widely in their selection of *machine-independent metrics* (e.g., number of states, transitions, or paths) and *machine-dependent metrics* (e.g., real time or amount of memory). When machine-independent metrics are used, they do not provide much utility for evaluating the efficiency of techniques if they are not good indicators of real time. Hence, machine-independent metrics that correlate highly with real time are the most desirable.

Dwyer et al. performed an important study on the controlling factors in evaluating state-space exploration techniques [2] and found that three factors—the search strategy used, the error density of the state space, and the number of threads—greatly affect comparisons among techniques. Based on this study, they selected a set of programs, with errors which were the most challenging to find (i.e., with low error density), to be used for comparing techniques. Moreover, they proposed and evaluated a new technique called Parallel Randomized State-Space Search (PRSS) [3] to reduce the time required to find these errors. PRSS was evaluated for the Java PathFinder (JPF) tool for stateful exploration of Java programs [16], [17] (specifically, an older JPF version, 3.1.2), and it was shown that PRSS can be very effective at finding errors quickly by exploiting the diversity among different random but potentially overlapping parallel explorations of a state space.

In this paper, (i) we revisit and extend the PRSS study by evaluating it with new contexts: additional subject programs, an additional state-space exploration tool, an additional search strategy, and a new version of JPF; and (ii) we additionally evaluate the correlation of machine-independent metrics with real time for two different state-space exploration tools for Java, stateful JPF (the latest version 6.0) and stateless ReEx (version 1.0).

In summary, we provide new evidence for selecting metrics in future evaluations of state-space exploration techniques. We find that several machine-independent metrics correlate highly with real time for many programs across multiple machine configurations and exploration tools. Additionally, all of the currently widely used machine-independent metrics for state-space exploration appear to be equally good proxies for real time, so researchers can measure and report metrics that have the lowest measurement overhead without much loss in confidence. Lastly, real time measured on clusters can correlate reasonably well with machine-independent metrics and so should be reported by studies that perform experiments on clusters.

II. Background

In this section we provide more information about PRSS and review metrics commonly used for evaluation of state-space exploration techniques in recent literature.

A. Parallel Randomized State-Space Search

The Parallel Randomized State-Space Search (PRSS) technique consists of performing multiple parallel randomized state-space explorations, each with a unique random seed on a separate machine in a cluster, and stopping all the explorations when one of them detects an error. The intuition behind the technique is that different randomized explorations will exercise diverse regions of the state space and hence detect an error (if one exists) faster than just performing a single non-randomized (default) exploration.

To evaluate PRSS [3], the authors performed the following steps. (i) They ran 5000 randomized depth-first explorations of each artifact using JPF. (ii) They sampled 50 random simulations of various PRSS configurations including 1, 2, 5, 10, 15, 20, and 25 parallel computers for each artifact. For example, the simulations for the PRSS configuration with 2 parallel computers involved randomly choosing 50 pairs of explorations from logs of the 5000 explorations recorded for an artifact and retaining the fastest exploration within each pair. (iii) They plotted the distribution of the results of the 50 simulations for each PRSS configuration and artifact. The mean of the distribution was considered the expected performance. (iv) They determined the point of diminishing return (PDR), i.e., the configuration where adding more parallel computers did not yield substantial performance benefits compared to the cost of utilizing additional computers.

Through this evaluation, PRSS was shown to be effective (some speedups more than 100x) for exploring various concurrent programs using JPF version 3.1.2 across a reasonably small number (5-20) of parallel machines. However, it is not

Paper	Metric(s)	Search Type	Used Randomness	Machine Configuration
CAPP [8]	Transitions	Stateful	Yes	Cluster
CAPP [8]	Schedules	Stateless	Yes	Cluster
Controlling Factors [2]	States	Stateful	Yes	Cluster
CTrigger [18]	Time	Stateless	No	Single machine
Depth Bounding [11]	States, Transitions, Time, Memory	Stateful	No	Not specified
Distributed Reachability [15]	Time	Stateless	Yes	Cluster
Gambit [7]	Time, Memory	Hybrid	Yes	Not specified
ICB [4]	States, Time	Hybrid	No	Not specified
PENELOPE [19]	Time	Stateless	No	Not specified
PRSS [3]	States	Stateful	Yes	Cluster
Random Backtracking [12]	States, Transitions	Stateful	Yes	Not specified
Swarm [5]	States, Time, Memory	Stateful	Yes	Single machine

Table I
CHARACTERISTICS OF EVALUATION FOR A SAMPLE OF RECENT PAPERS ON STATE-SPACE EXPLORATION

clear whether similar results could be obtained for (i) other concurrent programs, (ii) stateless exploration, (iii) different search strategy, or (iv) the latest version of JPF which incorporates many new optimizations. We revisit PRSS with these four additional contexts in Section III.

The authors of the PRSS study also considered using stateless search with JPF but ultimately did not use it because the version of JPF that they used in their experiments could not find an error in orders of magnitude more time than stateful searches that could in a few minutes. In our evaluation, we perform stateless search with ReEx and find that stateless search can be used effectively for PRSS, and that its effectiveness may depend on the implementation of the tool used.

B. Metrics

Researchers developing techniques for improving state-space exploration use various metrics to evaluate their techniques and present their results. Table I shows a sampling of recent papers presenting techniques for state-space exploration and the metrics used in their evaluation. Researchers often use machine-independent metrics to allow for analysis of results across machine configurations. The machine-independent metrics used vary based on the type of exploration tool used. In this paper, we consider both stateful (JPF) and stateless (ReEx) exploration tools.

Common machine-independent metrics for stateful exploration (e.g., JPF) include the following. *States:* The total number of unique program states encountered during the exploration. *Transitions:* The total number of transitions performed during the exploration; a transition progresses the execution from one program state to another. *Instructions:* The total number of instructions executed during the exploration; each transition consists of one or more instructions. *ThreadCGs:* The total number of thread choices generated during exploration; a thread choice is generated when the exploration encounters a state with multiple enabled threads.

Common machine-independent metrics for stateless exploration (e.g., ReEx) include the following. *Schedules:* The total number of schedules encountered during the exploration. Since the most common form of stateless exploration performs backtracking via re-execution, each schedule involves the re-execution of the program being explored for a unique sequence of choices. *Choices:* The total number of choices encountered during the exploration. Each choice is a point where more than one thread is available to be scheduled, and one of them is chosen to be scheduled. This is equivalent to the ThreadCGs metric for JPF. *Events:* The total number of events encountered during the exploration. Each event denotes the execution of a scheduling-relevant bytecode instruction (e.g., lock/unlock, field read/write). Choices are created when two or more threads in the program are all about to perform an event. *Threads:* The total number of threads created during the exploration across all schedules. The threads that are part of the program are re-created and counted during each schedule.

In Section III, we investigate the correlation of these commonly used machine-independent metrics with real time, which is the metric that a user of a tool eventually experiences and cares most about.

III. STUDY

A. Study Goals

The goal of our study is two-fold. First, we reinvestigate PRSS with the new contexts: different version of JPF (the latest version, 6.0), stateless exploration (using ReEx), different search strategy (ICB), and different concurrent programs. Second, we utilize the results of these and additional experiments to answer questions about the correlation of various machine-independent metrics with real time both on a compute cluster and on a dedicated desktop computer. As such, we revisit the same three research questions from PRSS [3] and address two more for correlation of real time with machine-independent metrics. More specifically, we address the following five questions:

RQ1 - Cost Reduction: Does there exist a feasible PRSS configuration that performs better than the default exploration? A feasible configuration is one with a reasonable number of parallel machines that could be available to a testing organization.

Subject	Source	Error	#Threads	#Classes	SLOC	Evaluated with JPF	Evaluated with ReEx
Airline	[20]	Assertion violation	6	2	136		X
BoundedBuffer	[20]	Deadlock	9	5	110	X	X
BubbleSort	[20]	Assertion violation	4	3	89	X	X
Daisy	[20]	Assertion violation	3	21	744	X	
Deadlock	[20]	Deadlock	3	4	52		X
DEOS	[20]	Assertion violation	4	24	838	X	
Elevator	[20]	ArrayIdxOOBExcptn	4	12	934	X	
PoolOne	[21]	Assertion violation	3	51	10042	X	X
PoolTwo	[22]	Assertion violation	3	35	4473	X	X
PoolThree	[23]	Deadlock	2	51	10802	X	X
RaxExtended	[20]	Assertion violation	6	11	166	X	
ReplicatedWorkers	[20]	Deadlock	3	14	432	X	
RWNoDeadLckCk	[20]	Assertion violation	5	6	154	X	

Table II
STUDY ARTIFACTS

RQ2 - Parallel Speedup: Does the performance of PRSS improve with the number of parallel machines used? If so, is there a point of diminishing returns?

RQ3 - Error Detection: Can PRSS be used to detect an error in programs where the default exploration runs out of time or memory?

RQ4 - Metrics Correlation: Do machine-independent metrics for state-space exploration correlate with real time for exploration? Does the correlation differ on compute clusters and dedicated machines?

RQ5 - Metrics Selection: Which metrics should be reported in future studies on state-space exploration?

B. Study Setup

1) Artifacts: We conducted our study with the thirteen concurrent Java programs shown in Table II. The programs have diverse characteristics and include benchmarks obtained from the Software-artifact Infrastructure Repository (SIR) [20], [24] and real-world test cases obtained from the Apache Commons Pool project [21]–[23].

We performed our JPF-based experiments with the eleven programs shown in Table III, which includes all seven programs used in the original PRSS study with the same inputs. The only exception is BoundedBuffer for which the latest JPF runs out of 8GB of memory for all explorations with the (3,6,6,1) input used in the original PRSS study. We have reported this as a regression to JPF developers, who have confirmed the error and are investigating it. Therefore, in our experiments we used the default input of (1,4,4,2). Airline and Deadlock were not used for the JPF experiments since the default exploration found the error too quickly to warrant the use of PRSS.

We performed our ReEx-based experiments with the seven programs shown in Table IV. The remaining six programs are reactive programs with cyclic state spaces that cannot be explored with ReEx (or another stateless tool that performs no state matching or fair scheduling to make progress when cycles are possible), because any single execution of such a program could be infinitely long.

2) Experiments: We conducted six sets of experiments to answer our research questions:

- **JPF-Cluster**: We performed default depth-first exploration and 1000 randomized depth-first explorations using the latest JPF for each of the eleven programs used for the JPF-based experiments. To perform the randomized depth-first explorations, we used the `cg.seed` property with the `DFSHeuristic` search class and set the `cg.randomize_choices` property to `path`. Using the results of these experiments we were able to *revisit PRSS with a different version of JPF* and reconsider the PRSS-related research questions.
- **ReEx-Cluster (DFS and ICB)**: We performed both default depth-first and default ICB exploration using ReEx for each of the seven programs used for the ReEx-based experiments. For both DFS and ICB, we additionally performed 500 randomized explorations with the `reex.exploration.randomseed` property using the `RandomDepthFirst` and `RandomIterativeContextBounding` search strategies, respectively, on the same seven programs. For both the default and randomized ICB explorations, we set the `reex.exploration.preemptionbound` to `2`. Using the results of these experiments we were able to *revisit PRSS with a stateless exploration tool*. Using the results of the ICB experiments we were able to *revisit PRSS with a different search strategy*.
- **Desktop**: We repeated a subset (50 seeds) of each of the cluster experiments on a dedicated desktop machine to measure real time more precisely and contrast the results with the cluster experiments.

The cluster experiments were performed on our departmental compute cluster, which uses Condor [25], and has 375 machines with CPU configurations that were either an Intel Xeon CPU X5650 @ 2.67 GHz or Intel Xeon CPU L5420 @ 2.50 GHz, memory configurations ranging from 1GB to 8GB, and 64-bit Sun JVM v1.6.0_10 on Linux 2.6.18. This was the only cluster available to us, but we repeated some of the experiments to account for

the diversity of machines and found that the difference in measured real time was negligible. The desktop experiments were performed on a dedicated machine with an Intel Core2 Duo E8400 @ 3.0 GHz, 2.00GB RAM, 64-bit Sun JVM v1.6.0_29 on Microsoft Windows 7; each seed was re-explored for 3 samples, and the real times were averaged to account for the potential system fluctuations. Each exploration in all the experiments had a one hour time limit.

We used the cluster experiments to compute the effectiveness of PRSS (described in Section II-A) for the new contexts. Then, as in the original PRSS study, we decided on the PDR values by making reasonable averages from several researchers that mostly agreed on common values for the number of computers. To measure the correlation of various machine-independent metrics with real time, we built linear regression models of each common machine-independent metric (described in Section II-B) versus real time for each of the six experiments.

C. Study Design

Independent Variables: The independent variables for our study include the artifacts used, type of exploration tool used (stateful JPF or stateless ReEx), type of search performed (depth-first or ICB), and machine configurations used. Another independent variable for the simulations performed to evaluate PRSS effectiveness is the number of parallel computers in the simulated configurations.

Dependent Variables: The measured results of our experiments in terms of both machine-independent and machine-dependent metrics form the basis for the dependent variables of our study. For RQ1 and RQ2, the dependent variable is the performance of the various PRSS configurations in terms of States for JPF and Schedules for ReEx and also the PDR in terms of the number of parallel computers. For RQ3, the dependent variable is the detection of an error by PRSS configurations in cases where the default exploration was unable to do so. For RQ4 and RQ5, the dependent variable is the coefficient of determination (R^2) of the linear regression models built to fit the measured real time to various machine-independent metrics.

D. Study Results

In this section we present the results of our experiments in terms of the research questions posed in Section III-A. RQ1, RQ2, and RQ3 address the effectiveness of PRSS under new contexts. Figure 1 and Table III show PRSS results for the latest JPF version. Figure 1 shows the distribution of results for the various PRSS configurations. Table III shows the exploration costs for the default exploration, comparing it with the exploration costs for the PRSS configuration indicated by the PDR. The table also shows the minimum and maximum exploration costs across all the 1000 random explorations performed for the experiment. Figure 2 and Table IV show the PRSS results for stateless DFS exploration with ReEx. Figure 3 and Table IV show PRSS results for ICB exploration performed with ReEx. RQ4 and RQ5 address the correlation of machine-independent metrics with real time. Table V shows results for the machine-independent metrics collected during the (stateful) JPF based experiments. The table presents R^2 values for the linear regression models built to fit millisecond-precision real time to the machine-independent metrics. The models were built both for the experiments performed on the compute cluster and dedicated desktop machine. Table VI shows the same results for machine-independent metrics collected during (stateless) ReEx based experiments.

RQ1 - Cost Reduction: We discuss this question with respect to the new contexts under which we revisited PRSS.

Latest JPF (version 6.0): Comparing Table III with the original PRSS [3], the results indicate that the performance of JPF has improved substantially since the original PRSS study. For most of the programs, the default exploration with the latest JPF finds the error faster than reported in the original PRSS study. This is most evident with Rax-Extended, ReplicatedWorkers, and RWNoDeadLckCk where the default exploration with the latest JPF is orders of magnitude faster than what was reported earlier. Despite this improvement, all cases have a feasible PRSS configuration for every program, which could find an error substantially faster in terms of states than the default exploration (Table III, Maximum and Minimum columns). For Daisy, all the PRSS configurations are able to find the error faster.

Stateless Exploration Tool: The results indicate that PRSS can be applied successfully even with a stateless exploration tool. All programs had a feasible PRSS configuration that reduced exploration costs compared to the default exploration. For BoundedBuffer, Deadlock, and PoolThree all PRSS configurations were faster than or as fast as the default exploration. For all programs, the fastest random search (Table IV, Minimum column) was orders of magnitude faster than the default exploration. Also, the slowest random search (Table IV, Maximum column) that was slower than the default exploration was orders of magnitude slower than the default exploration. This diversity among the random explorations is key to the success of PRSS.

ICB Search Strategy: The results for the ICB search strategy show that the effectiveness of PRSS is *dependent* on the search strategy that is used. While each program (except Deadlock) had a feasible PRSS configuration that improved upon the default exploration, the diversity among different configurations was minimal compared to the JPF and ReEx random depth-first explorations. This can be observed from the difference between the Minimum and Maximum values in Table IV and the means of the various PRSS configurations in Figure 3. For example, for Airline, the costs of the random explorations ranged from 44287 schedules to 44314 schedules, while the cost of the default

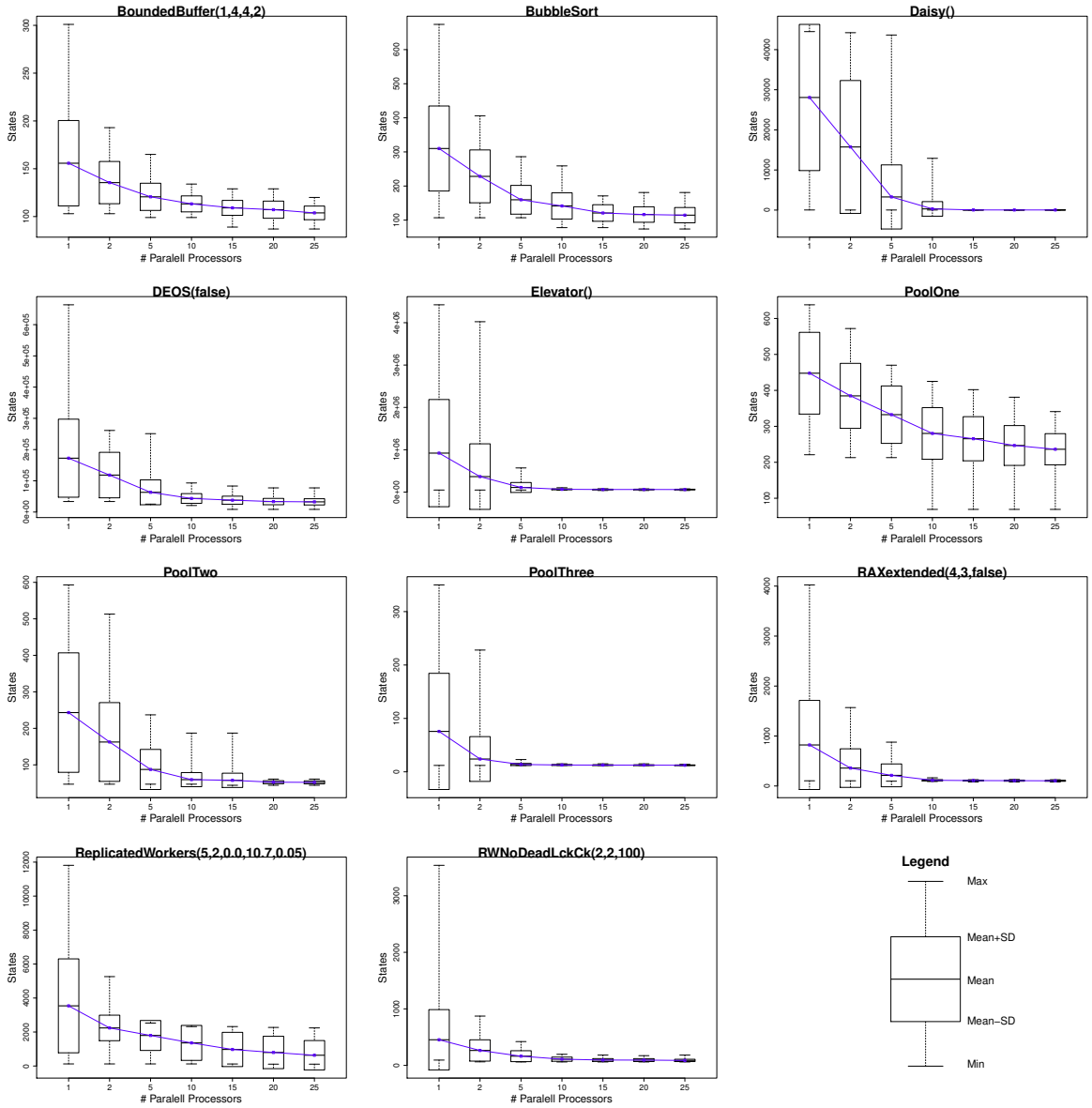

Figure 1. PRSS results for JPF

Subject	States			PDR		
	Default	Minimum	Maximum	Nodes	Mean	Speedup
BoundedBuffer	235	87	314	5	121	1.9
BubbleSort	258	57	748	5	160	1.6
Daisy	45712	19	44570	10	295	155.0
DEOS	28523	8509	663529	10	43486	*0.7
Elevator	468064	38040	4558268	10	65381	7.2
PoolOne	490	69	1470	5	332	1.5
PoolTwo	603	44	2029	5	87	6.9
PoolThree	218	12	377	2	24	9.0
RaxExtended	4331	80	12546	10	114	38.0
ReplicatedWorkers	1960	108	29833	15	973	2.0
RWNoDeadLckCk	396	61	6071	10	110	3.6

Table III
JPF PRSS RESULTS (* INDICATES A SLOWDOWN)

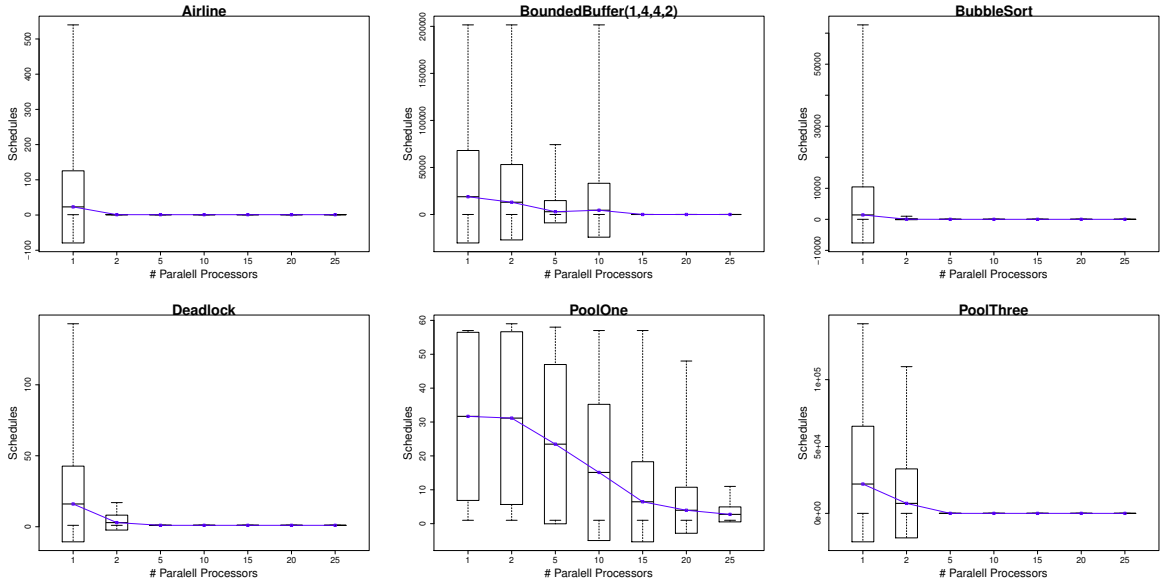

Figure 2. PRSS results for ReEx DFS

Subject	Schedules (DFS)			PDR (DFS)			Schedules (ICB)			PDR (ICB)		
	Default	Min	Max	Nodes	Mean	Speedup	Default	Min	Max	Nodes	Mean	Speedup
Airline	48	1	1783	2	1	48.0	44312	44287	44314	5	44287	1.0
BoundedBuffer	1166902 (TO)	1	1140436	15	1	≥1166902.0	1329	1	40	10	1	1329
BubbleSort	109	1	2426308	2	28	3.9	5179	4644	4712	15	4646	1.1
Deadlock	298	1	298	2	3	99.3	6	9	9	1	9	*0.7
PoolOne	444908 (TO)	1	968282	15	6	≥74151.3	6463	1094	1153	15	1097	5.9
PoolTwo	540355 (TO)	1	734014	1	1	≥540355.0	123	107	108	10	107	1.2
PoolThree	434097 (TO)	1	156921	5	1	≥434097.0	2	1	2	1	1	2.0

Table IV
REEX PRSS RESULTS (* INDICATES A SLOWDOWN)

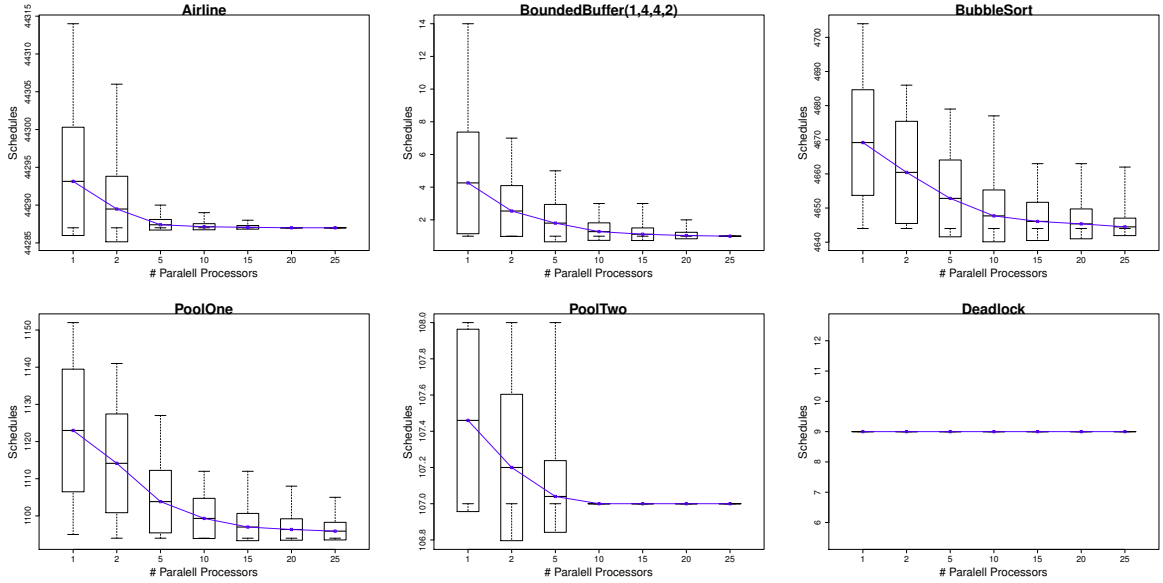

Figure 3. PRSS results for ReEx ICB

exploration was 44312 schedules. This reduction in diversity is a result of the nature of our randomized ICB exploration where randomization only happens among choices with the same number of preemptions.

Different Programs: All the additional programs had feasible PRSS configurations that could provide speedup compared to their default explorations. This demonstrates that PRSS generalizes across various types of programs.

RQ2 - Parallel Speedup: We discuss this question with respect to the new contexts under which we revisited PRSS.

Latest JPF (version 6.0): The improvement in the default performance for the latest JPF has resulted in a reduction in the magnitude of performance improvement random explorations can achieve, and also reduced the frequency of random explorations that perform better than the default exploration. While PRSS was able to achieve more than 100x speedup for three of the programs with the older JPF, only Daisy achieved more than 100x speedup with the latest JPF. PRSS even resulted in a *slowdown* for DEOS with the latest JPF, while it had obtained a 1.8x speedup with the older JPF. Also, while the speedup reduced, the number of computers in the PDR PRSS configuration remained relatively the same for the latest JPF. To summarize, *PRSS obtained lesser speedup with the latest JPF. Moreover, the reduced speedup still required the same number of parallel computers as with the older JPF.*

Stateless Exploration Tool: The PDR configurations for stateless exploration achieve significant speedups compared to the default exploration, the lowest speedup being 3.9x for BubbleSort, and the highest speedup being $\geq$1166902.0x for BoundedBuffer. The second highest speedup is $\geq$540355.0x for PoolTwo, which is still several orders of magnitude. However, note that for PoolTwo, it is not really necessary to perform PRSS since the PDR configuration has only one parallel computer, i.e., any one random depth-first exploration is expected to find the error so much faster than the default exploration. *While PRSS achieves much higher speedups for stateless exploration compared to stateful exploration, it does so with a relatively similar number of parallel computers. For all the programs, the PDR configuration has 15 or fewer (mostly fewer than 5) parallel computers.*

ICB Search Strategy: As can be seen from Table IV, PRSS achieves much lesser speedup (Deadlock even incurs a slowdown) for the ICB search strategy compared to depth-first search. This can be attributed, as described earlier, to the lack of diversity in the randomized ICB explorations.

Different Programs: PRSS was able to achieve speedups for all the additional programs that we used for evaluation. The speedups ranged from 1.5x for PoolOne with stateful JPF to $\geq$434097.0x for PoolThree on stateless ReEx.

RQ3 - Error Detection: Unlike in the original study, none of the default explorations timed out with the latest JPF. So we were unable to address this question with respect to those experiments. However, four of the seven programs used in the stateless ReEx experiments did time out for the default exploration (indicated by TO in Table IV). For all four of these programs, PRSS was able to find the error and achieve significant speedup even compared to the costs at the point of timeout. There were no default explorations that timed out for the ICB experiments so this question could not be addressed for that context.

RQ4 - Metrics Correlation: We discuss this question with respect to the various exploration tools and strategies used.

Stateful JPF: The Desktop section of Table V shows that for almost all the programs, all the machine-independent stateful exploration metrics considered correlate highly with real time, with R^2 values generally more than 0.85. The main exceptions are PoolOne and PoolTwo for which all the metrics have lower R^2 values. For both these programs, the real time values were low (1 second or less) and not diverse. The imprecision in measurement may explain why their R^2 values were lower. It is interesting to note that all the considered metrics correlate equally well with real time, i.e., *metrics that measure exploration costs with a finer granularity (e.g., instructions), which could result in a higher measurement overhead, do not provide benefits in terms of higher correlation with real time.*

The Cluster section of Table V shows that the metrics considered do not correlate with real time as well as in the Desktop experiments. However, surprisingly, for the majority of programs, all the stateful exploration metrics do correlate reasonably well with real time, with R^2 values generally more than 0.75. Note that BoundedBuffer, Bubblesort, PoolOne, PoolTwo, and PoolThree each have relatively small state spaces (indicated by Maximum/Minimum ranges in Table III), and this smaller amount of diversity may explain their lower correlations. Also, as observed in the Desktop experiments, all the metrics considered correlate equally well (or equally badly) with real time.

Stateless ReEx: Table VI shows that for almost all the programs, all the machine-independent stateless exploration metrics considered correlate highly with real time. This is the case for both the Desktop experiments and *surprisingly* the Cluster experiments as well. The only exception is PoolTwo for the Desktop experiments in which all 50 seeds finished in 1 schedule, so this small correlation can be explained by lack of seed diversity.

ICB ReEx: Due to the lack of diversity in the random ICB explorations and their fast completion times, we were unable to build meaningful linear regression models comparing the measured real time from these experiments with machine-independent metrics. Hence, we do not consider these results in our discussions.

RQ5 - Metrics Selection: The experimental results show that all the commonly considered machine-independent metrics correlate equally well (or equally badly) with real time.

Subject	Desktop				Cluster			
	States	Transitions	Instructions	ThreadCGs	States	Transitions	Instructions	ThreadCGs
BoundedBuffer	0.943	0.935	0.887	0.908	0.231	0.234	0.220	0.231
BubbleSort	0.866	0.873	0.755	0.780	0.313	0.319	0.297	0.303
Daisy	0.995	0.995	0.999	0.995	0.935	0.935	0.936	0.935
DEOS	0.998	1.000	1.000	0.848	0.916	0.918	0.918	0.785
Elevator	0.999	0.999	0.998	0.999	0.929	0.929	0.929	0.929
PoolOne	0.581	0.566	0.511	0.570	0.286	0.282	0.228	0.284
PoolTwo	0.694	0.614	0.787	0.692	0.568	0.522	0.612	0.571
PoolThree	0.889	0.892	0.875	0.889	0.683	0.684	0.677	0.684
RaxExtended	0.972	0.948	0.746	0.961	0.894	0.853	0.597	0.873
ReplicatedWorkers	0.995	0.993	0.994	0.995	0.895	0.891	0.894	0.895
RWNoDeadLckCk	0.990	0.981	0.959	0.991	0.780	0.758	0.741	0.780

Table V
JPF DFS METRICS TIME CORRELATIONS (R^2 VALUES)

Subject	Desktop				Cluster			
	Schedules	Choices	Events	Threads	Schedules	Choices	Events	Threads
Airline	0.887	0.894	0.891	0.893	0.951	0.955	0.952	0.952
BoundedBuffer	1.000	0.998	0.999	1.000	0.992	0.991	0.993	0.992
BubbleSort	1.000	1.000	1.000	1.000	0.988	0.995	0.994	0.988
Deadlock	0.941	0.941	0.941	0.941	0.884	0.883	0.884	0.884
PoolOne	0.801	0.799	0.801	0.801	0.953	0.950	0.955	0.953
PoolTwo	0.000	0.018	0.000	0.000	0.963	0.961	0.965	0.963
PoolThree	0.884	0.886	0.884	0.884	0.994	0.981	0.994	0.994

Table VI
REEX DFS METRICS TIME CORRELATIONS (R^2 VALUES)

Thus, researchers can continue to use the metrics that are currently used in the literature. In fact, *counter intuitively*, metrics that measure exploration cost more accurately—like instructions for stateful exploration and events for stateless exploration—do not provide substantial benefit in terms of correlation with real time.

It is commonly acknowledged in the research community that real time measured from experiments performed on compute clusters may not be reliable. Depending on the configuration used, cluster job management systems may schedule one or more processes on each processor of a single SMP compute node [26], [27], so the time measurements for a process may be affected by the execution of other processes. Also, compute nodes on clusters may be heterogeneous, i.e., have different configurations, so time measurements may not be comparable across processes. Hence when experiments are performed on compute clusters, the authors usually do not report the time measurements. However, *surprisingly*, our results indicate that real time measurements from clusters can be useful since they correlate well with machine-independent metrics. So in the future, researchers should consider reporting real time measured from cluster experiments, but they should exercise caution while aggregating results from heterogeneous clusters.

E. Threats to Validity

Internal Threats: We performed our experiments with the default settings in JPF and ReEx, with a time bound of one hour. Changing the settings or increasing the time bound could affect our findings. To the best of our knowledge, there are no errors in JPF and ReEx (and our implementations of randomized depth-first search and randomized ICB search algorithms in ReEx) that affected our results. However, we did encounter a regression error in JPF that we reported to the JPF developers.

External Threats: We used a diverse set of thirteen artifacts in our experiments as shown in Table II. However, these programs do not necessarily form a representative set of concurrent programs. Also, the programs that we used exhibit a diverse set of errors including deadlocks, atomicity violations, and data races that lead to various assertion violations. However, these errors may not form a representative set of errors found in all concurrent programs.

To mitigate the effects of using a single type of exploration tool or a single search strategy, we used both stateful JPF and stateless ReEx for both DFS and ICB search in our experiments. However, this does not cover all types of exploration tools or search strategies that may be used, hence there remains a threat that our results may not generalize to other types of tools or search strategies, or combinations of search strategies [5].

Our experiments were performed across two different types of machine configurations, including a desktop machine and a heterogeneous cluster, using two different operating systems and two different JVM versions. However, these machine configurations may not necessarily form a representative set of all machine configurations. More combinations of both desktop and cluster machines with configurations including different operating system parameters, such as disk swapping, and different JVMs, possibly

with alternate parameters such as heap size, would be needed to mitigate this threat.

Construct Threats: While designing our study, we chose metrics that are used most commonly to report results for stateful and stateless state-space exploration experiments. However, using different metrics or new combinations of them may shed a different light on our results.

Conclusion Threats: We used 1000 random seeds for the JPF-based experiments performed on the cluster, 500 random seeds for both the ReEx-based experiments performed on the cluster, and a subset of 50 of those seeds for the three sets of experiments performed on the dedicated desktop machine. While these numbers of seeds are reasonably large, there is a threat that they may not have sufficiently represented the actual distribution of all possible random explorations.

While our cluster experiments were performed at different times of the day across the span of a few weeks, the real time results obtained could be a consequence of the load patterns prevalent at those times. Controlled experiments with different load patterns would have to be performed to rule out this threat.

IV. Conclusions

Researchers have developed many new techniques to address the increasingly important problem of state-space exploration of concurrent code. It can be hard to compare these techniques because empirical evaluations of them use various machine-independent and machine-dependent metrics. We present a study of various common machine-independent metrics for two different exploration frameworks for Java (JPF and ReEx) by evaluating their correlation with real time across a variety of machine configurations. We find that all of the machine-independent metrics we studied (i) correlate reasonably well with real time, particularly on programs with larger state spaces, (ii) correlate equally with real time, so lower granularity metrics can be used without loss in confidence, (iii) correlate reasonably well with real time measured on the cluster, so researchers should also report real time measured on clusters. Our results also indicate that PRSS is still effective for the latest JPF and also effective for stateless search in ReEx.

Acknowledgments

We would like to thank Mladen Laudanovic for discussions on statistical analysis, Qingzhou Luo for discussions about state-space exploration, and Peter Mehlitz for help with JPF. This material is based upon work supported by the National Science Foundation under Grant Nos. CCF-1012759, CNS-0958199, CCF-0916893, and CCF-0746856.

References

[1] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. The MIT Press, Cambridge, MA, 1999.

[2] M. B. Dwyer, S. Person, and S. G. Elbaum, "Controlling factors in evaluating path-sensitive error detection techniques," in *FSE*, 2006.

[3] M. B. Dwyer, S. G. Elbaum, S. Person, and R. Purandare, "Parallel randomized state-space search," in *ICSE*, 2007.

[4] M. Musuvathi and S. Qadeer, "Iterative context bounding for systematic testing of multithreaded programs," in *PLDI*, 2007.

[5] G. J. Holzmann, R. Joshi, and A. Groce, "Tackling Large Verification Problems with the Swarm Tool," in *SPIN*, 2008.

[6] N. Rungta, E. G. Mercer, and W. Visser, "Efficient Testing of Concurrent Programs with Abstraction-Guided Symbolic Execution," in *SPIN*, 2009.

[7] K. Coons, S. Burckhardt, and M. Musuvathi, "Gambit: Effective Unit Testing for Concurrency Libraries," in *PPoPP*, 2010.

[8] V. Jagannath, Q. Luo, and D. Marinov, "Change-aware preemption prioritization," in *ISSTA*, 2011.

[9] M. Gligoric, V. Jagannath, and D. Marinov, "MuTMuT: Efficient exploration for mutation testing of multithreaded code," in *ICST*, 2010.

[10] G. Yang, M. B. Dwyer, and G. Rothermel, "Regression model checking," in *ICSM*, 2009.

[11] A. Udupa, A. Desai, and S. Rajamani, "Depth bounded explicit-state model checking," in *SPIN*, 2011.

[12] P. Parizek and O. Lhoták, "Randomized backtracking in state space traversal," in *SPIN*, 2011.

[13] "ReEx home page," http://mir.cs.illinois.edu/reex/.

[14] G. Ciardo, Y. Zhao, and X. Jin, "Parallel symbolic state-space exploration is difficult, but what is the alternative?" in *PDMC*, 2009.

[15] R. H. Carver and Y. Lei, "Distributed reachability testing of concurrent programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 18, 2010.

[16] W. Visser, K. Havelund, G. P. Brat, S. Park, and F. Lerda, "Model checking programs," *Automated Software Engineering*, vol. 10, no. 2, 2003.

[17] "JPF home page," http://babelfish.arc.nasa.gov/trac/jpf/.

[18] S. Park, S. Lu, and Y. Zhou, "CTrigger: Exposing atomicity violation bugs from their hiding places," in *ASPLOS*, 2009.

[19] F. Sorrentino, A. Farzan, and P. Madhusudan, "Penelope: weaving threads to expose atomicity violations," in *FSE*, 2010.

[20] University of Nebraska Lincoln, "Software-artifact Infrastructure Repository," http://sir.unl.edu/portal/index.html.

[21] Apache Software Foundation, "POOL-107," https://issues.apache.org/jira/browse/POOL-107.

[22] ——, "POOL-120," https://issues.apache.org/jira/browse/POOL-120.

[23] ——, "POOL-146," https://issues.apache.org/jira/browse/POOL-146.

[24] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, 2005.

[25] Condor Team, "Condor Project Homepage," http://research.cs.wisc.edu/condor/.

[26] T. A. El-Ghazawi, K. Gaj, N. A. Alexandridis, F. Vroman, N. Nguyen, J. R. Radzikowski, P. Samipagdi, and S. A. Suboh, "A performance study of job management systems," *Concurrency: Practice and Experience*, vol. 16, no. 13, 2004.

[27] Condor Team, "Condor Manual Section 3.13.9: Configuring The Startd for SMP Machines."