Performance issues? Hey DevOps, mind the uncertainty!

Catia Trubiani, *Gran Sasso Science Institute (GSSI), L'Aquila, Italy*

Pooyan Jamshidi, *University of South Carolina - Columbia, US*

Jurgen Cito, *MIT CSAIL - Cambridge, MA, US*

Weiyi Shang, *Concordia University - Montreal, Canada*

Zhen Ming Jiang, *York University - Toronto, Canada*

Markus Borg, *RISE SICS AB - Lund, Sweden*

DevOps is a novel trend that aims to bridge the gap between software development and operation teams. When applied to the performance evaluation process, it brings new challenges since developers need to be aware of the deployment settings and application runtime characteristics. At the operational stage, several uncertainties, e.g., workload fluctuations and resource availability, may affect the performance analysis. The goal of this paper is to identify the uncertain parameters and quantify their propagation to performance analysis results, in order to bring upfront the main system criticisms. To this end, we make use of a popular big data system showing that the sources of uncertainty may span on different characteristics and the performance analysis results can be heavily affected by these uncertainties. The paper contributes as an experience report aiming to better identify performance uncertainties through a case study. It provides a step-by-step guide to practitioners for controlling system uncertainties.

Keywords: Uncertainty; Performance Analysis; DevOps; Software Development.

Introduction

DevOps is a recent trend aimed at integrating development (Dev) and operational (Ops) teams together[5]. One of the needs for such an integration is driven by the requirement to continuously adapt software system designs based on operational uncertainties, such as workload fluctuations and resource availability. Such uncertainties inevitably affect the dependability characteristics of systems (e.g., performance and reliability) which may suffer and produce negative consequences. For instance, a software system can initially perform well with high throughput and low response time, but its performance may suddenly degrade due to reasons like workload fluctuations or software upgrades. To make informed decisions, DevOps teams need to be aware of uncertainties in the whole DevOps life-cycle to be able to interpret data, models, and results accordingly. Support in this direction can be provided by: (i) identifying sources of uncertainty in a performance-aware DevOps scenario; (ii) elaborating how these uncertainties manifest in input data, design models and operational results; (iii) performing a sensitivity analysis to quantify the impact of these sources of uncertainty for results interpretation. Hence, it is necessary to put in place a set of methodologies that model and control these uncertainties so that violations of performance requirements can be detected and thus, bridging the operational performance issues within the decision process of

software designers. In literature, uncertainty embeds the concept of variability (i.e., the natural variation of some parameters) and it is recognized to be very relevant in the analysis process[1,2]. Hence, it is necessary to include some form of uncertainty representation[12] into the engineering process and to identify software characteristics that are not completely known. There exist performance prediction approaches that provide a sensitivity analysis of many parameters by monitoring a system or by considering reasonable guesses by the domain experts. However, such model-based approaches are typically used for predicting performance of the system as a result of system configuration[6] or external uncertainties. However, such estimations are only approximations and may result in low accuracies which could be misleading in the software development process[14].
There are two main contributions of this paper: (1) we make the developers aware of the system uncertainties; (2) we run a sensitivity analysis to highlight the main system criticisms leading to performance issues. This experience demonstrates that it is fundamental to bring the sources of uncertainty up-front in the DevOps process to make the developers aware of such uncertainties, thus to guarantee the stakeholders' performance expectations.

Related work

This section briefly reviews related works that have been defined to model, analyze, and minimize the software system uncertainties. Note that *variability* can be considered a specific case of *uncertainty* since it includes the specification of parameters subject to varying values, whereas uncertainty also considers the lack of knowledge[12].

Modeling and Analyzing Uncertainty

The concept of uncertainty is discussed in many scientific fields. Kennedy and O'Hagan distinguish between six sources of uncertainty for models implemented in source code[8]: 1) *Parameter uncertainty* -- originating in the challenge of calibrating the model, i.e., aiming at finding the actual parameter input values; 2) *Model uncertainty* -- the difference between the real world process and the code output given to the system model; 3) *Residual uncertainty* -- due to an inherently unpredictable real world process. Even under stable conditions, such a process might produce different output when repeated; 4) *Parametric uncertainty* -- introduced when some of the input conditions are not specified by the parameter input, either intentionally or due to an uncontrollable process; 5) *Observation error* -- occurring when actual observations are used to calibrate the system model; 6) *Code uncertainty* -- variations in the output produced from executing a system model on a given platform. Ramirez et al. introduced an uncertainty taxonomy to establish a common vocabulary for the self-adaptive software community[10]. This taxonomy is composed of three phases of the development life-cycle: requirements, design, and run-time. In particular, the authors identified 26 sources of uncertainty, ranging from missing requirements to sensor noise. A taxonomy highlighting three aspects of uncertainty: *location*, *level*, and *nature*[9]. This taxonomy helps to understand i) where uncertainty manifests in the model, ii) what the level of uncertainty is (from deterministic knowledge to not even being certain about being uncertain), and iii) whether the uncertainty is caused by lack of measurement data or by inherent randomness in the model.

Minimizing Uncertainty

Minimizing uncertainty results to be an open research challenge. Research effort has been invested in order to minimize the uncertainty in software engineering experiments. Three main techniques have been adopted in this context. First, a widely adopted resolution of minimizing performance-related uncertainty is through repeated measurement[7]. The instability of performance measurements leads to uncertainties of performance evaluation results. Such measurements may be misleading and incorrect without providing measures of variation[7]. Georges et al.[3] recommended computing a confidence interval for repeated performance measurements when doing performance evaluation. With the knowledge of variation from repeated measurement, rigorous statistical techniques can be used to minimize uncertainty. Second, another attempt to minimize uncertainty is to gain more knowledge about the nature of the system based on extensive modeling and simulation. With models, it is possible to specify the uncertainty of parameters values

through probability distribution functions[13]. In this way, Monte-Carlo based simulation allows to extract samples of parameter values to minimize the uncertainty. Goldsby and Cheng[4] developed a model-based approach that generates a system model to simulate system behavior in complex environments. Developers can use such model to interactively understand the uncertainty in such environment. Third, another way to minimize uncertainty is to provide more information about the subject system, and uncertainties can be reduced. Yuan et al.[15] enriched the monitoring of system by attaching more runtime information.

Decisions in a Performance-aware DevOps Life-cycle Leading to Uncertainty

In this section, we describe the envisioned performance-aware DevOps life-cycle whose high-level illustration is reported in Figure 1. A central element of such life-cycle is represented by the models used for decision making. They may be: formal models to predict the system runtime, e.g., queuing models; runtime and system models to test design changes. For these models there are two types of inputs: *static information* in source code repositories and *dynamic information* instrumented or observed during the system runtime. A sensitivity analysis, i.e., the study of how the uncertainty in the output can be apportioned to different sources of uncertainty in the input[11], closes the cycle: stakeholders use this information base to converge towards an understanding of the system on various levels, e.g., influencing their decisions in the task of changing the running system by altering the source code, the configurations, the infrastructure, or even re-deploying the system.

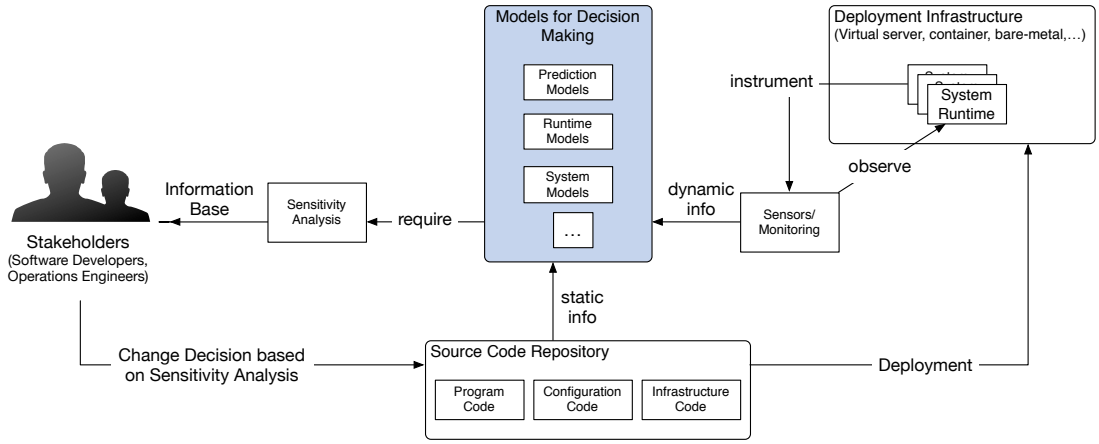

Figure 1 - Performance-Aware DevOps Life-Cycle under Uncertainty.

Design decisions are influenced by the uncertainty of the system under development[9]. To this end, we elaborate on the high-level decisions taken by the stakeholders that may affect the performance analysis results:

- Deployment infrastructure (DI): deciding on which physical or virtual infrastructure to deploy a system can have a tremendous effect on the uncertainty of various performance characteristics, especially in public cloud infrastructures.
- Software versions and code changes (SC): uncertainty in the performance characteristics can also be directly introduced by a code change through a software developer or indirectly by an operator's decision to upgrade to a different version of the software.
- Configuration parameters (CP): small adjustments in configuration parameters of software can have a tremendous effect on the uncertainty of the runtime characteristics of a system, e.g., the SQLite benchmarking has been found to be often incorrect.
- Workload fluctuation (WF): the performance of a system is a function of its workload, and linearly increasing load may have non-linear effects on a system. Wrongfully interpreting future states of a system when higher workloads occur can lead to uncertain decisions affecting configuration parameters.

- Monitoring and sensor accuracy (MS): operators rely on active monitoring, instrumentation, and sensors to observe and retrieve information about the (internal) state of a system, and adjusting the accuracy of sensors (e.g., through sampling) determines an inherent trade-off between the visibility of the state and introduced overhead.

Case Study

We conducted a controlled experiment as a case study to illustrate the effects on uncertainty caused by the DevOps decisions depicted in the previous section. The goal of this case study is to demonstrate the performance impact of various DevOps decisions. We assessed and quantified both the Dev (e.g., code or configuration changes) and the Ops (e.g., hardware and workload changes)-side changes, which may impact the system performance. Specifically, we measured different performance characteristics of Apache Cassandra while changing the deployment infrastructure, source code repository, and workload. For the sake of illustration, we focused on the throughput (i.e., sensors/monitoring, see Figure 1) of the benchmark queries against the Cassandra engine. However, a broader definition of performance can be extended to any quantitative non-functional property (e.g., reliability and security) of the system. We modeled potential actions that could affect uncertainty as a discrete decision space D = {DI, SC, CP, WF}. DI represents the space of underlying hardware (i.e., deployment infrastructure). SC= $\{v_1, \ldots, v_n\}$ depicts the space of software releases (i.e., software versions and code changes). CP= $\{c_1, \ldots, c_n\}$ is a set of configuration options that can be set for a particular version of a software. WF= $\{w_1, \ldots, w_n\}$ models the workload change on the system as a set of relative percentages of read and write operations. Finally, |D| represents the number of combinations of all these decision parameters.

Experimental Setting

We measured the performance characteristics of Apache Cassandra under different environmental changes (see Table 1) that represents the dimensions of concrete instances of the decision space D. For our Cassandra case study, we conducted systematic measurements (for measuring the performance indicators of one configuration for specific system version in specific environments and for specific workloads). We ran the benchmark for 10 minutes with the same operation repeated multiple times. Before the next measurement, the Cassandra database was cleaned and restarted to ensure each measurement started with the same initial state. The Cassandra database was left idle for 10 seconds as the warmup period before the start of each measurement round. We used the Yahoo! Cloud Serving Benchmark (YCSB - https://github.com/brianfrankcooper/YCSB) for generating different workloads and collecting the performance indicators of the system. More specifically, we used YCSB workload generator to first define the dataset and load it into the database; and second, to execute operations against the dataset while measuring performance. YCSB is a standard benchmarking tool that has been extensively used for performance measurements of key-value and cloud-based data engines.

Table 1 -Overview of the case study subject system. |D|: number of all possible decisions; |DI|: number of hardware environments; |SC|: number of analyzed software versions; |CP|: number of configuration options; |WF|: number of different analyzed workloads.

System	Domain	|D|	|DI|	|SC|	|CP|	|WF|
Cassandra	DB	1024	2	3	6	6

Table 2 -Summary of hardware platforms on which configurable software systems were measured; NC - Number of CPUs; IS - Instruction set; CCR - CPU clock rate (GHz); RAM - memory size (GiB).

ID	Type	NC	IS	CPU	CCR	RAM	Disk
h1	NUC	4	x86_64	i5-4250U	1.30	15	SSD
h2	NUC	2	x86_64	Celeron	2.13	7	SCSI

We observed output parameters in all different combinations: (i) hardware change; (ii) workload change; (iii) version change; (iv) workload-version change; (v) hardware-workload-version change. In particular, we measured the system performance considering 6 configuration options (leading to a total of 1024 system configurations) in different environments: 2 hardware environments, 3 software versions, and 6 workloads.

Table 2 provides a summary of the underlying hardware options in this case study. In order to understand the performance with respect to varying incoming request rate behavior, in light of different choices made in the decision space of the Apache Cassandra benchmark system, we ran six different workloads:

- **Workload A** (Update heavy): this workload has a mix of 50/50 reads and writes. An application example is a session store recording recent actions.
- **Workload B** (Read mostly): this workload has a 95/5 reads/write mix. An application example is a photo tagging session where adding a tag is an update, but most operations are to read tags.
- **Workload C** (Read only): this workload is 100% read. An application example is a user profile cache where profiles are constructed elsewhere (e.g., Hadoop).
- **Workload D** (Read latest): new records are inserted, and the most recent inserted records are the most popular. An application example is a user status updates; people want to read the latest.
- **Workload E** (Short ranges): short ranges of records are queried, instead of individual records. An application example is offered by threaded conversations where each scan is for the posts in a given thread.
- **Workload F** (Read-modify-write): the client will read a record, modify it, and write back the changes. An application example is a user database where user records are read and modified by the user.

We considered three different versions of Apache Cassandra for our measurements: SC = v_1 = 1.2.19, v_2 = 2.2.8, v_3 = 3.10. We chose the last release at the time, one closer release (2.2.8), and a very distant version of the system (1.2.19). We also artificially injected white noise of 10% to the sources in order to investigate the performance impact due to measurement noise. More details regarding the configuration options, considered environments, and all measurement data are publicly available: https://github.com/pooyanjamshidi/uncertainty.

Results

We analyzed the percentage of the top/bottom configurations that are common across the environmental changes. Table 3 shows the results, and uncertainty clearly shows a dramatic influence when we consider variations along hardware, workload, version changes or combinations of them. More specifically, we considered throughput as a metric and derived 10% of top (highest throughput) and 10% of bottom (lowest throughput). The results show that the percentage of common configurations is very low. In particular, we found that in ec_6 (ec^* stands for environmental change where the source environment is different from the target), practitioners have a higher chance to achieve top throughput, at the same time, they have a higher change in ec_9 to spot performance issues, if they choose the same configuration. But the chance is still low. Also, the major gap (0.08) between top and bottom values is showed by ec_3 which demonstrates that in this environment change, it is more likely that they find a high performing configuration and not observing a performance issue. This means that, despite of uncertainties in the parameters, ec_6 results the system configuration showing the highest system throughput, ec_9 is the one most likely leading to performance issues, whereas ec_3 is the one whose input uncertainties largely influence performance analysis results. This sensitivity analysis supports system administrators in their task of setting the best configuration of the system using specific hardware, workload, and version of the software or their combinations. In the last two columns of Table 3 we report the Spearman rank correlation values calculated on the original sources and on the artificially injected data (i.e., white noise of 10%), respectively. We observe that the rank correlation, despite being weak, still shows

a decreasing trend. This observation highlights the importance of handling uncertainty in the DevOps process, due to the sources of uncertainty because of environmental changes and measurement noise. For instance, as a developer, if one selected a configuration based on previous measurements in a certain environment, it should be careful to choose a configuration for the system in another environment since some environmental changes are more prone to uncertainty than others.

Table 3 - Results. Top/Bottom: percentage of top/bottom common configurations between source and target.

Decision	ID	Source	Target	Top	Bottom	|Top-Bottom|	Corr	Corr (10%)
DI	ec_1	h2-A-V3	h1-A-V3	0.0980	0.1569	0.0589	0.0364	-0.0078
SC	ec_2	h1-A-V3	h1-A-V2	0.0490	0.0588	0.0098	-0.1266	-0.0527
	ec_3	h1-A-V3	h1-A-V1	0.1176	0.0376	**0.08**	0.1424	0.0696
WF	ec_4	h2-A-V3	h2-B-V3	0.0392	0.0686	0.0294	-0.1732	0.0139
	ec_5	h2-A-V3	h2-C-V3	0.1373	0.1275	0.0098	0.0318	0.0381
	ec_6	h2-A-V3	h2-D-V3	**0.1471**	0.1176	0.0295	0.0088	0.0172
	ec_7	h2-A-V3	h2-E-V3	0.0490	0.0686	0.0196	-0.0704	0.0127
	ec_8	h2-A-V3	h2-F-V3	0.0686	0.1373	0.0687	0.0217	0.0078
SC-WF	ec_9	h1-A-V3	h1-B-V1	0.1078	**0.1765**	0.0687	0.1001	-0.0302
DI-SC-WF	ec_{10}	h2-A-V3	h1-B-V1	0.1078	0.1176	0.0098	-0.0327	0.0192

Our numerical results provide evidence that if uncertainty is not handled properly, performance issues may arise. For example, if a system configuration is selected based on a model trained by measurement data that are collected in an environment with a different workload, it may lead to a sub-optimal configuration. As a result, systems may encounter higher deployment costs or more failures due to larger memory allocations or threads spin up. This can be more problematic in critical domains, such as robotics. A further experience has been matured in the Model-based Adaptation for Robotics Software project (https://github.com/cmu-mars), where power models are used under budget constraints to adapt to perturbations, such as environmental or internal resources changes (e.g., battery level). Pareto optimal configurations are swapped at runtime based on environmental condition of the robot (e.g., its remaining battery level). We found out that when the model is inaccurate, Pareto optimal configurations are chosen incorrectly, and it results in a mission failure, as shown in the following demo: https://youtu.be/ec6BhQp2T0Q.
Given these consequences, if practitioners are aware of the uncertainty, they can opt to: 1) conduct additional experiments to further reduce the uncertainty. For example, repetitive measurements combined with statistical methods are widely used in prior research to reduce uncertainty, 2) identify and handle at the root cause of the uncertainty. As an example, if the deployment infrastructure introduces the uncertainty, one should consider control or leverage a more stable infrastructure, and 3) if the uncertainty cannot be easily reduced or handled, uncertainty quantification approaches (such as forward uncertainty propagation or inverse uncertainty quantification) can be employed to determine how likely certain outcomes are if some aspects of the system (e.g., optimal configurations) are not deterministically known[16].
Similar results have been observed in other systems (including a compiler, a SAT solver, a database engine, and a video encoder) across different environmental changes, see details in https://github.com/pooyanjamshidi/ase17. Note that the considered uncertainties are demonstrated to be relevant for some software systems, but they are far from being exhaustive. Further applications may show other characteristics that have not been evidenced so far, since it is indeed difficult to link performance issues to a finite list of system configuration settings.

Conclusions and Future Work

We conclude this paper by discussing the lessons learned and their implications as a result of our study in the following two dimensions: (1) identifying sources of uncertainties; and (2) modeling and controlling the uncertainties.

Identifying Sources of Uncertainty

The identification of the sources of uncertainty is challenging and two different strategies can be used for this scope: (i) the bottom-up that is based on the knowledge of the possible sources of uncertainty in the given model; and (ii) the top-down which starts from the complete lack of knowledge about possible uncertainties. We followed the bottom-up approach defined in[9], and we first started off by enumerating various decision points in the performance-aware DevOps life-cycle as shown in Figure 1. Then, we further limited the decision points to only those which can lead to performance uncertainty, and we narrowed down with five sources of uncertainty.

Modeling and Controlling Uncertainties

Once we have identified the sources of uncertainties, we want to minimize their impact by: (1) modeling and analyzing the performance variation caused by such uncertainties, then (2) devising approaches to control their impact. In our case study we observed the following sources of uncertainty:

- Deployment infrastructure (DI): our results show that within the same release and workload, changing deployment infrastructure can have a small to large impact on the system performance. For example, for version 3.10, the throughput can be more than doubled when switching between two different hardware platforms. Hence, to limit such uncertainties, it is important to conduct user acceptance testing or closely monitor the performance of the canary deployment[5] before full-fledged infrastructure changes.
- Software versions and code changes (SC): our results show that the optimal configurations for one version of Cassandra would probably not yield the best performance after system upgrade. As shown in Table 3, when switching between versions (ec_2 and ec_3), there is less than 12% top configurations shared among the two versions (before and after system update). Therefore, it is vital to examine the performance impact of various configuration parameters for each version. However, due to the large combination of the configuration space and the rapid changes in DevOps, performance testing reduction techniques (e.g., experimental design or redundancy detection) should be used to efficiently explore the system configuration space.
- Configuration parameters (CP): among different hardware platforms, software versions and workloads, the percentage of common configurations is very low. Within the same kind of setting (same hardware, workload, and release version), the throughput can vary up to 9 times differences among different Cassandra configurations. This clearly shows the importance role that the configuration parameters play in terms of the system performance. However, to minimize and control the uncertainty due to the configuration parameters, it is necessary to isolate and study the most relevant ones.
- Workload fluctuation (WF): similar as the above three aspects, for Cassandra, the optimal configurations do not translate when different workloads are exercised. As the system keeps evolving, the user behavior co-evolves. Hence, it is important to periodically verify and update the performance testing workload.
- Monitoring and sensor accuracy (MS): the measurement noise can impact the validity of the performance results, although its overall effect can be small. Hence, it is helpful to cross-reference the measurement data to ensure their validity.

Acknowledgments

This work is one of the results of break-out group sessions held during the Dagstuhl Seminar 16394 on ``Software Performance Engineering in the DevOps World'' which took place in September 2016. The report from GI-Dagstuhl Seminar 16394 is publicly available: https://arxiv.org/abs/1709.08951.

References

1. M. Autili, V. Cortellessa, D. Di Ruscio, P. Inverardi, P. Pelliccione, and M. Tivoli. “Eagle: Engineering software in the ubiquitous globe by leveraging uncertainty”. In Proceedings of ACM Symposium on the Foundations of Software Engineering (FSE), pages 488--491, 2011.

2. D. Garlan. “Software engineering in an uncertain world”. In Proceedings of the International Workshop on Future of Software Engineering Research (FoSER), pages 125--128, 2010.
3. A. Georges, D. Buytaert, and L. Eeckhout. “Statistically rigorous java performance evaluation”. In Proceedings of the ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications (OOPSLA), pages 57--76, 2007.
4. H. J. Goldsby and B. H. Cheng. “Automatically generating behavioral models of adaptive systems to address uncertainty”. In Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MoDELS), pages 568--583, 2008.
5. M. Httermann. “DevOps for developers”. Apress, 2012.
6. P. Jamshidi, N. Siegmund, M. Velez, C. Kastner, A. Patel, and Y. Agarwal. “Transfer learning for performance modeling of configurable systems: An exploratory analysis”. In Proceedings of the International Conference on Automated Software Engineering (ASE), 2017.
7. T. Kalibera and R. Jones. “Rigorous benchmarking in reasonable time”. In Proceedings of the International Symposium on Memory Management (ISMM)}, pages 63--74, 2013.
8. M. C. Kennedy and A. O'Hagan. “Bayesian calibration of computer models”. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 63(3):425--464, 2001.
9. D. Perez-Palacin and R. Mirandola. “Dealing with uncertainties in the performance modelling of software systems". In Proceedings of the International Conference on Quality of Software Architectures (QoSA), pages 33--42, 2014.
10. A. J. Ramirez, A. C. Jensen, and B. H. Cheng. “A taxonomy of uncertainty for dynamically adaptive systems”. In Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)}, pages 99--108, 2012.
11. A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto. “Sensitivity analysis in practice: a guide to assessing scientific models”. John Wiley & Sons, 2004.
12. R. C. Smith. “Uncertainty quantification: theory, implementation, and applications”, volume 12, Siam, 2013.
13. A. Aleti, C. Trubiani, A. van Hoorn, and P. Jamshidi “An efficient method for uncertainty propagation in robust software performance estimation”. Journal of Systems and Software, 138, 222-235, 2018.
14. C. M. Woodside. “Regression techniques for performance parameter estimation”. In Proceedings of WOSP/ SIPEW International Conference on Performance Engineering, pages 261--262, 2010.
15. D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, Y. Zhou, and S. Savage. “Be conservative: Enhancing failure diagnosis with proactive logging. In Proceedings of the USENIX conference on Operating Systems Design and Implementation (OSDI), pages 293--306, 2012.
16. Smith, Ralph C. Uncertainty quantification: theory, implementation, and applications. Vol. 12. Siam, 2013.