ACTS in Need: Automatic Configuration Tuning with Scalability Guarantees*

Yuqing Zhu, Jianxun Liu, Mengying Guo, Wenlong Ma, Yungang Bao
Advanced Computer Systems Research Center
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
zhuyuqing,liujianxun,guomengying,mawenlong,baoyungang@ict.ac.cn

ABSTRACT

To support the variety of Big Data use cases, many Big Data related systems expose a large number of user-specifiable configuration parameters. Highlighted in our experiments, a MySQL deployment with well-tuned configuration parameters achieves a peak throughput as **12 times** much as one with the default setting. However, finding the best setting for the tens or hundreds of configuration parameters is mission impossible for ordinary users. Worse still, many Big Data applications require the support of multiple systems co-deployed in the same cluster. As these co-deployed systems can interact to affect the overall performance, they must be tuned together. Automatic configuration tuning with scalability guarantees (ACTS) is in need to help system users. Solutions to ACTS must scale to various systems, workloads, deployments, parameters and resource limits. Proposing and implementing an ACTS solution, we demonstrate that ACTS can benefit users not only in improving system performance and resource utilization, but also in saving costs and enabling fairer benchmarking.

ACM Reference format:
Yuqing Zhu, Jianxun Liu, Mengying Guo, Wenlong Ma, Yungang Bao. 2017. ACTS in Need: Automatic Configuration Tuning with Scalability Guarantees. In *Proceedings of APSys '17, Mumbai, India, September 2-3, 2017,* 8 pages.
https://doi.org/10.1145/3124680.3124730

1 INTRODUCTION

The Big Data industry is estimated to be worth more than hundreds of billions of dollars and still growing [5, 32]. Along with the Big Data phenomenon, many systems emerge to fulfill the tasks of collecting, processing and analyzing the huge amount of data, e.g., Hadoop [2] and Spark [3]. To support the variety of Big Data use cases, many Big Data related systems are designed and developed with a large number configuration parameters (or knobs) [45]. For example, Hadoop [2] has more than 180 knobs, while the database system MySQL [7] has more than 450 knobs. These tunable configuration parameters control nearly all aspects of system runtime behaviors [21].

On the one hand, these configuration parameters are highly correlated with the system performance [1, 11, 12]. Take MySQL for instance. Changing the configuration setting can result in more than **11 times** performance gain for MySQL (§5.1). On the other hand, the large number of configuration parameters lead to an ever-increasing complexity of configuration issues that overwhelm users, developers and administrators. As multiple systems can be involved in a task of Big Data management, tuning multiple systems' configuration parameters that intrinsically interact in an application has surpassed the abilities of humans [17].

Automatic configuration tuning (ACT) can help users tune the large number of configuration parameters towards a better overall performance. ACT involves solving the performance optimization problem in the high-dimensional space of configuration parameters. Previous attempts to automate configuration tuning have used search-based [44, 46], model-based [21, 31] or control-based [19, 40] methods. Some of these methods assume manually constructed models [31] or the existence of system simulators [26, 29, 46]. These assumptions are feasible for a specific system with a limited number of parameters [19, 48]. Some methods assume the existence of a large sample set [18]. Many have not considered the influence of workloads on tuning [42], and hardly any related work considers the deployment environment as a factor that affects configuration tuning.

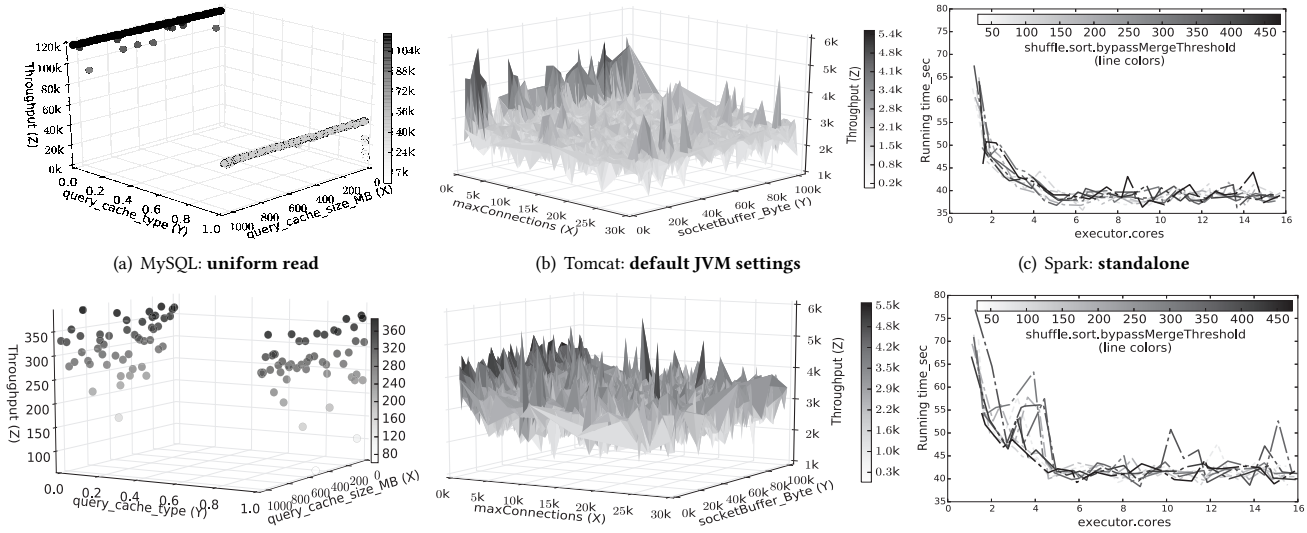
Now, the problem of automatic configuration tuning is facing three new challenges not studied before (§2). These challenges come from the large number of configuration parameters, the dynamicity and complexity of the performance model, and the expensive sample collection process. The unprecedentedly large number of configuration parameters have complicated the performance model of a system. Worse still, the complex performance model is also related to factors like workloads and deployment environments; hence, a system must be tuned in a deployment environment similar or the same as the real system deployment. This fact makes the collection of samples very expensive.

Automatic configuration tuning with scalability guarantees (ACTS) is in need to address these challenges. In this work, we motivate the study of the ACTS problem: automatically tuning the system configuration parameters while addressing the above challenges by guaranteeing five scalability requirements (§3). The five scalability requirements involve the scalability regarding SUT (system under tune), workload, deployment environment, parameter set, and sample set size.

*Yuqing Zhu is the corresponding author.

APSys '17, September 2-3, 2017, Mumbai, India

ACM ISBN 978-1-4503-5197-3/17/09…$15.00
https://doi.org/10.1145/3124680.3124730



(a) MySQL: **uniform read** (b) Tomcat: **default JVM settings** (c) Spark: **standalone**

(d) MySQL: **zipfian read-write** (e) Tomcat: **tuned JVM settings** (f) Spark: **cluster**

Figure 1: Diverging performance surfaces of *MySQL, Tomcat* and *Spark* under different workloads and deployments.

Despite the difficulty of the ACTS problem, we propose a preliminarily feasible solution (§4). This solution has a flexible system architecture different from previously proposed architectures. This flexible architecture can adapt to different SUTs, workloads and deployment environments. It also allows the easy integration of scalable sampling methods and scalable optimization algorithms to solve the ACTS problem. We have implemented the solution to demonstrate the benefits of ACTS.

Solving the ACTS problem can lead to great benefits for users (§5), including but not limited to facilitating the system usage, improving the system performance, increasing the system utilization, saving labor costs, fairer benchmarking results, identifying system bottlenecks, etc. As only sporadic efforts are found to study the problem of automatic configuration tuning, our goal here is mainly to motivate the study of the ACTS problem, as well as demonstrating the feasibility and the great benefits of solving ACTS.

2 NEW CHALLENGES OF ACT

The problem of automatic configuration tuning (ACT) is becoming more challenging. On the one hand, ACT involves an optimization problem which is previously known to be difficult to solve. On the other hand, new challenges are emerging, thanks to Big Data.

2.1 A Large Number of Knobs

The number of configuration parameters is now tens, hundreds or more. Previously, the number of variables is multiple in the optimization problem of configuration tuning [26, 29, 46]. As Big Data systems are targeting a wide range of use cases, they can provide tens or hundreds of configuration parameters [45] for users in various use cases and with various deployment environments. Moreover, co-deployed systems might need to be tuned together for one use case such that the number of parameters to tune can further increase. For example, the tuning guides for Hadoop suggest tuning the configuration parameters of both the Hadoop (with more than a hundred knobs) and the JVM (Java Virtual Machine, with tens of knobs) [14, 28]. Thus, the performance optimization problem of configuration tuning must be solved in a high-dimensional space that rarely occurs in previous optimization problems' settings.

The number of parameters can hardly be reduced in configuration tuning. The impacts of configuration parameters on a system's performance are intrinsic and complicated. Sometimes, a configuration parameter can have impacts unfound even by system developers [21]. Besides, configuration parameters are non-stochastic variables. Hence, the dimension reduction methods commonly used in machine learning cannot be applied to reduce the number of configuration parameters, e.g., factor analysis [6] or principal component analysis [8]. Even though the more impacting knobs can be tuned first [42], the less impacting knobs cannot be neglected in tuning, because it is likely that the combined impact of all the less impacting knobs exceeds that of the more impacting knobs.

2.2 Dynamicity and Complexity

We have carried out **thousands of experiments** to study the dynamicity and complexity of performance models. The performance model of an SUT is highly dynamic and complex, because it is related to not only the SUTs, but also the varying workloads and deployment environments. The impacts of the deployment environment can come from the hardware and the software [16, 47]. Such impacts make it infeasible to decompose the SUT and the deployment environment into subcomponents for tuning. These facts also make it very difficult for human beings to manually construct models or simulators for general systems.

Different SUTs have different performance models. Take the widely used database system MySQL [7], Web server system Tomcat [4] and big data processing system Spark [3] for example. We plot in Figure 1(a), 1(b) and 1(c) their performance functions projected in low-dimensional spaces respectively. For MySQL, the projection is two lines, while Tomcat's is a irregularly bumpy surface. Spark's is a relatively smooth surface, which can be depicted as smooth lines when projected to a 2-D space.

Different workloads also lead to different performance models. For the same deployment of MySQL, we apply the different workloads of uniform read and zipfian read-write. The differed workloads result in the diverging plots in Figure 1(a) and 1(d). For the uniform read workload, the *query_cache_type* is the configuration parameter dominating the system performance. But for the zipfian read-write workload, the value of *query_cache_type* has no such dominant influence. The impacts of configuration parameters are workload related.

The hardware of the deployment environment influences the performance model. A system can be deployed on a single server or a server cluster. The system deployed on a single server generally behaves differently from that deployed in a cluster. To demonstrate the hardware impact from the deployment environment, we deploy Spark in the standalone mode and the cluster mode. Applying the same workload, we get two differed performance functions as plotted in Figure 1(c) and 1(f). As compared to the smooth performance function of the standalone mode, that of the cluster mode rises up sharply at some points, e.g., when the value of *executor.cores* equals to four.

The co-deployed software also has intrinsic impacts on the SUT's performance. For Big Data applications, multiple systems might need to be deployed together to accomplish one task. For example, we might need to deploy the Hadoop file system for using Spark; and, running Java-based systems requires the running of JVM (Java Virtual Machine). Co-deployed software systems can interact with and influence each other, as they might share hardware resources like CPU cycles, memory and network bandwidth. For instance, Figure 1(e) differs from Figure 1(b) only in that we change the JVM setting *TargetSurvivorRatio* when generating Figure 1(e). Although the performance surface remains as bumpy, the maximum performance is achieved at different areas.

2.3 Costly Sample Collection

Only a limited number of samples can be collected for tuning. Because of the impacts from the deployment environment and the workload, the performance-configuration samples can only be generated in tests that apply the workload on the deployed system. Thus, collecting a large set of tuning samples is too expensive to be practical. As performance models or simulators can hardly be constructed due to complexity, no arbitrary number of samples can be generated for configuration tuning. It is not practical at all to collect thousands of samples as required by existing solutions to configuration tuning [29]. Rather, users might expect a solution exploiting only hundreds or tens of samples, considering that Big Data workloads generally take time to run [24, 27]. In sum, configuration tuning must restrain the overhead of sample collection.

3 THE NEW PROBLEM: ACTS

The new challenges in fact call for solutions to a new problem. The new problem is the problem of *automatic configuration tuning with scalability guarantees (ACTS)*. The ACTS problem is to find, within a given **resource limit**, a **configuration setting** that can optimize the performance of a given **SUT**'s **deployment** under a specific **workload**.

Resource limit can be represented as the time or the number of tests allowed for tuning. Different measures for resource can be transformed into and represented by each other. For convenience, we consider in this paper the resource limit as the number of allowed tests, which is equal to the number of samples to be collected.

The solution to the ACTS problem must guarantee scalability with regard to **resource limit**, **configuration parameter set**, **SUT**, **deployment environment** and **workload**. When the resource limit is relaxed, the solution to ACTS is expected to output a configuration setting with a better performance. The solution must also be able to find new best configuration settings when new configuration parameter sets, SUTs, deployment environments and workloads are provided. It must adapt to the changes of these factors. Besides, the integration of evolved SUTs, deployment environments and workloads must be facilitated.

The problem of ACTS and the scalability requirements invalidate the assumptions of related works on automatic configuration tuning. First, preconstructing models or simulators [31] has surpassed the capabilities of humans due to the large number of configuration parameters in the overall system, as well as due to the complicated interactions between the SUT, the workload and the deployment environment. Second, the sample collection becomes very expensive as the tuning samples can only be collected for a specific deployment of system, instead of being reused across deployments [42]. The costs of sample collection must be taken into account in configuration tuning, rather than assuming a large sample set [29]. Third, assuming strong conditions for tuning is impractical as the performance model of an SUT is correlated with the varying workloads, hardware settings, co-deployed software, and the set of configuration parameters.

4 A PRELIMINARY ACTS SOLUTION

Although the ACTS problem is difficult, we demonstrate that it is solvable by presenting a preliminary ACTS solution in this section.

4.1 Design Rationale

To solve the ACTS problem, we must allow the tuning system to collect samples directly from the SUT in the target deployment environment and under the real workload. The sample collection process requires changing the configuration settings of the SUT, which must be restarted to allow the new configuration setting to take effect. Therefore, the tuning system must be able to control the SUT and run the workload. To fulfill this purpose, we design the architecture of the tuning system with the components of system manipulator and workload generator. To avoid the interference with the real applications on tuning, the design of this architecture takes advantage of the staging environment that commonly exists

and that is the same as the actual deployment environment. The resulting architecture is plotted in Figure 2. Section 4.2 presents a brief overview of the architecture and the interactions between system components.

While the problem about how to collect samples with scalability guarantees is solved by the flexible architecture, the problem about which samples to collect remains to be addressed. Due to the large number of configuration parameters and their wide ranges, it is impossible to try every possible combinations. In fact, only a very limited number of configuration settings can be tested and sampled, because of the resource limit in the ACTS problem. Here, there exists a subproblem of sampling.

The subproblem of sampling must handle all types of parameters, including boolean, enumeration and numerics. The resulted samples must have a wide coverage of the solution space. To guarantee scalability, the sampling method must also guarantee better coverage of the whole solution space if more samples are allowed by the users. Thus, the sampling method must produce sample sets satisfying the following three conditions: (1) the set has a wide coverage over the high-dimensional space of configuration parameters; (2) the set is small enough to meet the resource limit and reduce test costs; and, (3) the set can be scaled to have a wider coverage, if the resource limit is expanded. We propose to use the LHS (Latin Hypercube Sampling) [36] method to solve the sampling subproblem, as it meets all the three conditions (detailed in §4.3).

There also exists the second subproblem, which is to maximize the performance metric based on the given number of samples. It is required that the output configuration setting must improve the system performance than a given configuration setting, which can be the default one or one manually tuned by users. To optimize the output of a function/system, two general methods exist, i.e., model-based and search-based. Whichever method is used, the optimization must satisfy the following conditions: (1) it can find an answer even with a limited set of samples; (2) it can find a better answer if a larger set of samples is provided; and, (3) it will not be stuck in local sub-optimal areas and has the possibility to find the global optimum, given enough resources. As model-based methods generally require a large sample set, we consider search-based methods. Thus, we propose to use, along with LHS, the recursive random search (RRS) algorithm [46] that satisfies all the three conditions (detailed in §4.3).

4.2 A Flexible Architecture

The flexible architecture is depicted in Figure 2. It mainly consists of three components, i.e., a tuner, a system manipulator, and a workload generator. Abiding by the ACTS problem definition, the tuner accepts the resource limit (typically the number of allowed tests) from the user. It extracts the configuration parameter set and their ranges from the SUT. The tuner allows different sampling and optimization methods to be used, because the SUT, the deployment environment and the workload are decoupled from the tuning process by the other two components. The workload generator allows the easy integration of various workloads for tuning, thus satisfying the workload scalability. The system manipulator can

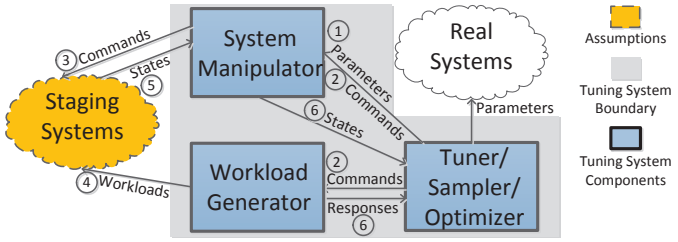


Figure 2: Architecture: automatic configuration tuning for general systems. (Best view in color)

easily integrate with different SUTs in different deployment environments.

Existing ACT solutions cannot solve the ACTS problem partially because their architecture designs are based on assumptions violating the scalability requirements. We can group the architectures of ACT solutions into three categories, i.e., the simulation-based architecture [29], the large-sample-set-based architecture [18] and the deployment-irrelevant architecture [42] that reuses samples collected from different system deployments. These architectures are illustrated in Figure 3. Explained in Section 2, the new challenges invalidate the assumptions that underlie these architectures.

Compared to the architectures in Figure 3, three major differences exist for the flexible architecture. First, the SUT, workload and deployment scalability is considered in the design of the workload generator and the system manipulator, with the tuner controlling these two components. Second, the tuner has not reused samples collected from other system deployments. As explained in Section 2, performance models are deployment-related, thus samples for other deployments cannot be reused.

Third, the tuning tests are run in a staging environment, instead of real systems or simulators. The staging environment is a mirror of the production environment, having the same actual deployment settings (e.g. hardware, clustering, software, etc.) [33, 39, 41]. Using live data, it is mainly for a final test of the system before production [23, 37]. And, implementing the real application workload in the workload generator is possible for the system in the staging environment, e.g., by log replay [20, 22, 34]. Our architecture exploits the staging environment such that samples can be collected without affecting applications on the real system deployment.

4.3 Subproblem Solutions: LHS + RRS

Following the analysis for solving the two subproblems (§ 4.1), we adopt the LHS (Latin Hypercube Sampling) [36] method and the recursive random search (RRS) algorithm [46].

LHS is a classic method for experimental design. Assuming that we need to collect m samples. LHS divides the range of each parameter into m intervals. It combines one interval of each parameter to form a subspace, in which LHS randomly chooses a sample. Repeating this process for m times, m samples get chosen. It is required that every interval of each parameter is used exactly once in the process.

LHS is a scalable sampling method. First, it has a wide coverage over the high-dimensional space because it considers every interval of each parameter. Second, it can sample by setting m equal to

the sample set constraint. Third, if m is increased, it can be scaled to have a wider coverage, because the sampling process of LHS is based on m.

The RRS algorithm has the exploitation and exploration structure commonly seen in search-based algorithms. In the exploration stage, RRS searches in a sample set that is taken from the whole parameter space and finds a promising sample that has the best performance. Then, it starts an exploitation stage by searching around the promising sample in the local parameter subspace. The exploitation stage is for locally searching the best point. When no improvement is made in the exploitation stage, RRS reenters the exploration stage to search globally to avoid local suboptimal results.

RRS algorithm is a scalable optimization algorithm. First, with the sample set size constraint, we can actually tune the optimization problem into one for finding a configuration setting **better** than a known setting. As a search-based method, RRS works for a sample set of any size. Second, RRS will find a better answer if a larger set of samples is provided, as it can search locally around the best sample. Third, RRS will not be stuck in local sub-optimal areas, because it has the exploration stage.

5 HOW ACTS BENEFITS USERS

We have implemented LHS and RRS with the flexible ACTS architecture, as well as trying other sampling and optimization algorithms. We apply the ACTS implementation to MySQL and Tomcat to demonstrate how ACTS can benefit users.

ACTS can bring about the benefits of manual configuration tuning by improving system performance and increasing system utilization. Besides, due to its objectivity, ACTS also brings about the extra benefits such as enabling fairer system comparisons and identifying system bottlenecks.

5.1 Improving System Performance: *11 Times Better*

Configuration tuning can improve the system performance, thus manual configuration tuning before using a system is in fact a common practice. General rules of "best practices" can be found on the Web for many popular systems, but they do not always provide the best results in many cases. Besides, some rules are difficult for common users to follow. As a result, although manual configuration tuning can improve system performance, users cannot always tune a system to the system's best potential.

ACTS only changes the configuration settings of a system, but the possible performance gain can be as much as **11 times**. In the example of MySQL, the best configuration setting suggested by ACTS can reach a throughput of 118184 ops/sec, while that for the default setting is only 9815 ops/sec. In comparison, many systems implementing new designs can only improve the system performance by *a limited percentage or multiple times*. That is, an easy change of the configuration settings can benefit the user much more than laboriously implementing new designs. Moreover, as many workloads are repetitive and recurring [13, 25], this performance gain can actually be highly significant to users.

5.2 Improving System Utilization: *Eliminating 1 from every 26*

ACTS can also improve system utilization by reducing the demands of virtual machines. Nowadays, it is common that many systems are deployed on the virtual machines in the cloud. Improving the throughputs of a single virtual machine can in turn reducing the number of virtual machines in need.

In a use case of Tomcat, we apply ACTS to Tomcat servers deployed on virtual machines, which run on physical machines equipped with ARM CPUs. Each virtual machine is configured to run with 8 cores, among which four are assigned to process the network communications. Under the default configuration setting, the utilizations of the four cores serving network communications are fully loaded, while the utilizations of the other four processing

Table 1: ACTS improving performances of a fully-utilized Tomcat server.

Metrics	Default	BestConfig	Improvement
Txns/seconds	978	1018	**4.07%** ↑
Hits/seconds	3235	3620	**11.91%** ↑
Passed Txns	3184598	3381644	**6.19%** ↑
Failed Txns	165	144	**12.73%** ↓
Errors	37	34	**8.11%** ↓

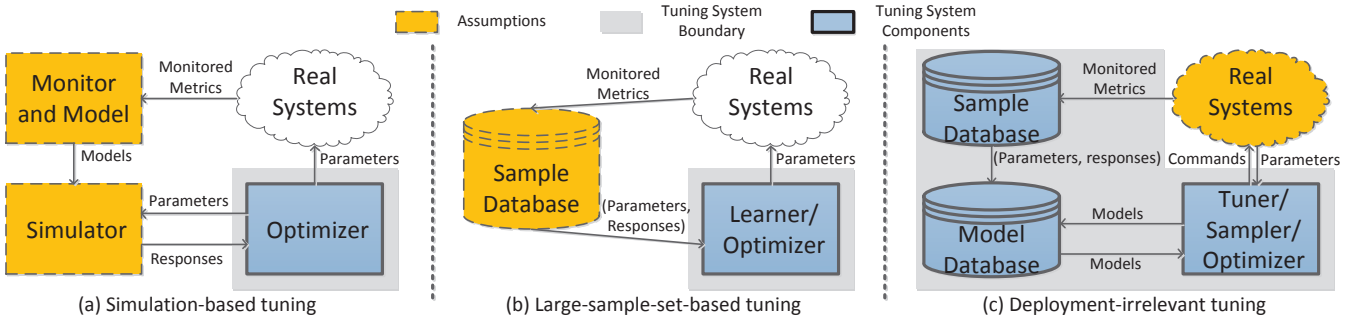


Figure 3: Common assumptions and architectures for configuration tuning in related works. (Best view in color)

cores are about 80%. By automatic configuration tuning, a better configuration setting is found to improve the performance of the deployment by 4%, while the CPU utilizations remain the same. The performance results of the tuned and the default configuration settings are presented in Table 1. We can observe improvements on every performance metric by the tuned configuration setting. With this improvement on throughput, we can eliminate 1 virtual machine from every 26 virtual machines, if the tuned configuration setting is used instead.

5.3 Saving Labor Costs: *Machine-Days vs. Man-Months*

Configuration tuning is highly time-consuming and laborious. It requires the users: 1) to find the heuristics for tuning; 2) to manually change the system configuration settings and run workload tests; and, 3) to iteratively go through the second step many times till a satisfactory performance is obtained. Sometimes, the heuristics in the first step might misguide the users, as some heuristics are correct for one workload but not others; then, the latter two steps are in vain.

In our experience with MySQL tuning, it has once taken five junior employees about half a year to find an appropriate configuration setting for a cloud application workload. We have also exploited our ACTS system to tune the same system deployment. A better performance is achieved within two days. Automatic configuration tuning not only saves labor costs, but also shortens the tuning time from months to days. Even if system experts might tune a system much better and faster than common users, they are very expensive to hire [35]. In comparison, automatic configuration tuning almost involves no labor costs.

5.4 Fairer Benchmarking and Comparison of Systems

Benchmarking is a well-established method for comparing the performance of various hardware or software systems [15, 30], e.g., running SPEC for hardware comparison [9] or TPC benchmarks for database systems [10].To enable an *apples-to-apples* comparison between systems, it is required that the only changed factor on benchmarking is the system under test. Besides, to get a good benchmarking result, the system under test must be well tuned [38, 43]. Configuration tuning is part of the performance tuning process. However, the performance tuning process is highly subjective and depends heavily on the tuning experts.

ACTS enables an objective tuning process and enables fairer starting points for benchmarking. As demonstrated by the MySQL case, a simple change of parameters can lead to more than 11 times performance improvement. As many new system designs can only improve system performances by some percentage or multiple times, it is more relevant that any improvement on system be tested on an optimized system state. Without a proper configuration tuning process, the benchmarking results can be highly suspicious or misguiding. As previous configuration tuning is usually manual work, there is no way to define how a system is in a state ready for benchmarking. To tap the performance potential of a system, system users need help in configuration tuning.

5.5 Identifying System Bottlenecks

In the use case of Big Data, it is common that multiple systems are deployed simultaneously for an application. For example, we might need to deploy the Hadoop file system for using Spark, or run a workload balancing system to distribute requests to the backend database system. Among the co-deployed systems, we might need to find out which system is the bottleneck in order to improve the overall performance.

ACTS can help identify system bottlenecks by (1) tuning the subsystem to its best performance; and, (2) combine systems to tune for the best performance. Take the database system for example. Database is usually deployed along with a front-end caching and load balancing system. Once we have tuned a database system by itself and improved the performance by 63%. Then, we apply the same workload to the tuned database system through a front-end caching and load balancing system. Even after a long time tuning, we found that the performance remaind at the untuned level for the database system co-deployed with the front-end system. By such, we located the bottleneck to be the front-end caching and load balancing system. Without automatic configuration tuning, we would not be able to make sure whether the reason is configuration setting or systems themselves.

Furthermore, by automatically tuning each system or the system combination to its best performance, we can also identify the bottleneck to be a specific system, if the system has the worst performance among all systems and system combinations; or, if the system combination has the worst performance, the bottleneck is the specific system combination. When a combination of systems has the worst performance, it indicates that the member systems are having interactions affecting the overall performance. This bottleneck identification can help users decide whether to improve the design of a specific system or to reduce the influences between systems.

6 CONCLUSION

In this paper, we comprehensively investigate the challenges and analyze the characteristics of the ACTS problem. The solution to the ACTS problem must guarantee scalability with regard to resource limit, configuration parameter set, SUT, deployment environment and workload. We propose and implement a preliminary ACTS solution. This solution features a flexible architecture, which enables the easy integration of various SUTs, deployment environments and workloads, as well as scalable sampling methods and optimization algorithms. The scalable sampling method and optimization algorithm adopted in the preliminary solution are LHS and RRS respectively. Based on the initial experimental results, we demonstrate that ACTS can benefit users in facilitating the system usage, improving the system performance, increasing the system utilization, saving labor costs, fairer benchmarking results, system bottleneck identification, etc.

Systems are becoming more complex nowadays. We believe that ACTS will become more beneficial or even indispensable to users. As a result, we believe that future systems should be equipped with automatic configuration tuning. We have only proposed a preliminary solution to the ACTS problem to demonstrate that

ACTS is solvable. Great research opportunities exist in devising better solutions to ACTS and equipping systems with ACTS.

ACKNOWLEDGMENTS

We would like to thank our shepherd, Cheng Li, and the anonymous reviewers for their constructive comments and inputs to improve our paper. This work is in part supported by the National Natural Science Foundation of China (Grant No. 61303054), the State Key Development Program for Basic Research of China (Grant No. 2014CB340402) and gifts from Huawei.

REFERENCES

[1] 2017. 17 KEY MYSQL CONFIG FILE SETTINGS (MYSQL 5.7 PROOF). http://www.speedemy.com/17-key-mysql-config-file-settings-mysql-5-7-proof/. (2017).
[2] 2017. Apache Hadoop Website. http://hadoop.apache.org/. (2017).
[3] 2017. Apache Spark Website. http://spark.apache.org/. (2017).
[4] 2017. Apache Tomcat Website. http://tomcat.apache.org/. (2017).
[5] 2017. Data, data everywhere. http://www.economist.com/node/15557443. (2017).
[6] 2017. Factor Analysis. https://en.wikipedia.org/wiki/Factor_analysis. (2017).
[7] 2017. MySQL Website. http://www.mysql.com/. (2017).
[8] 2017. Principal component analysis. https://en.wikipedia.org/wiki/Principal_component_analysis. (2017).
[9] 2017. Standard Performance Evaluation Corporation (SPEC). http://www.spec.org/. (2017).
[10] 2017. Transaction Processing Performance Council (TPC). http://www.tpc.org/. (2017).
[11] 2017. Tuning Spark. http://spark.apache.org/docs/latest/tuning.html. (2017).
[12] 2017. Tuning YARN. https://www.cloudera.com/documentation /enterprise/5-6-x/topics/cdh_ig_yarn_tuning.html. (2017).
[13] Sameer Agarwal, Srikanth Kandula, Nicolas Bruno, Ming-Chuan Wu, Ion Stoica, and Jingren Zhou. 2012. Re-optimizing data-parallel computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 21–21.
[14] AMD. 2017. Hadoop Performance Tuning Guide. https://developer.amd.com/wordpress/media/2012/10/Hadoop_Tuning_Guide-Version5.pdf. (2017).
[15] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. 2006. *The landscape of parallel computing research: A view from Berkeley*. Technical Report. UCB/EECS-2006-183, EECS Department, University of California, Berkeley.
[16] Theophilus Benson, Aditya Akella, and Aman Shaikh. 2011. Demystifying configuration challenges and trade-offs in network-based isp services. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 302–313.
[17] Phil Bernstein, Michael Brodie, Stefano Ceri, David DeWitt, Mike Franklin, Hector Garcia-Molina, Jim Gray, Jerry Held, Joe Hellerstein, HV Jagadish, et al. 1998. The Asilomar report on database research. *ACM Sigmod record* 27, 4 (1998), 74–80.
[18] Josep Lluís Berral, Nicolas Poggi, David Carrera, Aaron Call, Rob Reinauer, and Daron Green. 2015. Aloja-ml: A framework for automating characterization and knowledge discovery in hadoop deployments. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1701–1710.
[19] Xiangping Bu, Jia Rao, and Cheng-Zhong Xu. 2009. A reinforcement learning approach to online web systems auto-configuration. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*. IEEE, 2–11.
[20] Yanpei Chen, Archana Sulochana Ganapathi, Rean Griffith, and Randy H Katz. 2010. Towards understanding cloud performance tradeoffs using statistical workload analysis and replay. *University of California at Berkeley, Technical Report No. UCB/EECS-2010-81* (2010).
[21] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning database configuration parameters with iTuned. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1246–1257.
[22] George W Dunlap, Samuel T King, Sukru Cinar, Murtaza A Basrai, and Peter M Chen. 2002. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 211–224.
[23] Steve Evans. 2010. Windows Azure Staging Model. http://www.loudsteve.com/2010/10/10/windows-azure-staging-model/. (2010).
[24] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. Popescu, A. Ailamaki, and B. Falsafi. 2012. Clearing the Clouds: A Study of Emerging Workloads on Modern Hardware. *Architectural Support for Programming Languages and Operating Systems* (2012).
[25] Andrew D Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. 2012. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 99–112.
[26] Adem Efe Gencer, David Bindel, Emin Gün Sirer, and Robbert van Renesse. 2015. Configuring Distributed Computations Using Response Surfaces. In *Proceedings of the 16th Annual Middleware Conference*. ACM, 235–246.
[27] Ahmad Ghazal, Minqing Hu, Tilmann Rabl, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. In *Proc. of SIGMOD 2013*. ACM.
[28] Vitthal Gogate. 2017. Hadoop configuration & performance tuning. https://www.slideshare.net/vgogate/hadoop-configuration-performance-tuning. (2017).
[29] Bilal Gonen, Gurhan Gunduz, and Murat Yuksel. 2015. Automated network management and configuration using Probabilistic Trans-Algorithmic Search. *Computer Networks* 76 (2015), 275–293.
[30] Jim Grey. 1993. The Benchmark Handbook for Database and Transaction Systems. (1993).
[31] Herodotos Herodotou, Fei Dong, and Shivnath Babu. 2011. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 18.
[32] Manyika James, Chui Michael, Brown Brad, Bughin Jacques, D Richard, R Charles, and H Angela. 2011. Big data: the next frontier for innovation, competition, and productivity. *The McKinsey Global Institute* (2011).
[33] CHRIS LEMA. 2015. Why do I need a staging environment? Let me tell you. http://chrislema.com/staging-environment/. (2015).
[34] Ang Li, Xuanran Zong, Srikanth Kandula, Xiaowei Yang, and Ming Zhang. 2011. Cloudprophet: towards application performance prediction in cloud. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 426–427.
[35] Marc Linster. 2014. Best practices for becoming an exceptional postgres dba. https://www.slideshare.net/EnterpriseDB/dba-best-practices-webinar-slides-final. (2014).
[36] Michael D McKay, Richard J Beckman, and William J Conover. 2000. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* 42, 1 (2000), 55–61.
[37] Peter Murray. 2017. Traditional Development/Integration/Staging/Production Practice for Software Development. http://dltj.org/article/software-development-practice/. (2017).
[38] G.P. Musumeci, M. Loukides, and M.K. Loukides. 2002. *System Performance Tuning*. O'Reilly Media.
[39] Konrad Ohms. 2015. Best practices for developing and organizing multi-stage app deployments in Bluemix. https://www.ibm.com/blogs/bluemix/2015/12/best-practices-for-multistage-app-deployments-in-bluemix/. (2015).
[40] Pradeep Padala, Kang G Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. 2007. Adaptive control of virtualized resources in utility computing environments. In *ACM SIGOPS Operating Systems Review*, Vol. 41. ACM, 289–302.
[41] Quora. 2013. DevOps: How do companies like Facebook, Twitter, and Tumblr handle their stage environments? https://www.quora.com/DevOps-How-do-companies-like-Facebook-Twitter-and-Tumblr-handle-their-stage-environments. (2013).
[42] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1009–1024.
[43] VMware. 2017. Performance Tuning and Benchmarking Guidelines for VMware Workstation 6. https://www.vmware.com/pdf/WS6_Performance_Tuning_and_Benchmarking.pdf. (2017).
[44] Bowei Xi, Zhen Liu, Mukund Raghavachari, Cathy H Xia, and Li Zhang. 2004. A smart hill-climbing algorithm for application server configuration. In *Proceedings of the 13th international conference on World Wide Web*. ACM, 287–296.
[45] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*. ACM, 307–319.
[46] Tao Ye and Shivkumar Kalyanaraman. 2003. A recursive random search algorithm for large-scale network parameter configuration. *ACM SIGMETRICS Performance Evaluation Review* 31, 1 (2003), 196–205.
[47] Jiaqi Zhang, Lakshminarayanan Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, Tianyin Xu, and Yuanyuan Zhou. 2014. Encore: Exploiting system environment and correlation information for misconfiguration detection. *ACM SIGPLAN Notices* 49, 4 (2014), 687–700.

[48] Wei Zheng, Ricardo Bianchini, and Thu D. Nguyen. 2007. Automatic Configuration of Internet Services. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. ACM, New York, NY, USA, 219–229.