Grand Challenge: Experiment-driven Adaptive Systems

Shivnath Babu
Duke University

The collection, maintenance, and use of instrumentation data from systems is a central theme in autonomic computing research. Instrumentation data, collected through various *sensors*, can be used in many ways to simplify system management. For example, the data can be used to (i) understand and forecast complex system behavior [7], (ii) learn models of system behavior to aid failure diagnosis and capacity planning [4], and (iii) derive controllers that automatically tune system configuration under varying workloads and recover systems upon failure [10].

Figure 1 illustrates the pipeline that collects and processes instrumentation data in a self-adaptive system. Clearly, the collected data plays a critical role. If this data does not represent true system behavior, then most of the techniques built on top will fail, possibly in disastrous and embarrassing ways due to incorrect diagnoses, misconfigured systems, and wrong actions taken by controllers. The phrase "garbage in, garbage out" sums up this situation aptly.

The grand challenge we identify in this paper centers on how the "right" instrumentation data can be collected and made available to the higher levels of the data-processing pipeline. We will first outline some long-standing open problems in database systems and argue that the core challenge in these problems is the generation of appropriate instrumentation data. These problems have counterparts in most other systems like Internet services and storage servers.

Configuration parameters: Commercial database systems ship with hundreds of configuration parameters C_1, C_2, ..., C_n (e.g., buffer pool sizes, number of I/O daemons). Today, reasonable settings for these "tuning knobs" for a complex application depend on the expertise of highly-skilled database administrators or time-consuming trial-and-error steps. To develop automated configuration recommenders, performance models of the form $Perf = F(C_1, \ldots, C_n)$ are needed, where F is a function (e.g., regression) to estimate overall performance given any configuration setting. Learning such functions is the challenging part that requires a number of data points of the form $\langle C_1 = v_1, \ldots, C_n = v_n, Perf{=}p \rangle$, where p is the performance observed when the system runs in the configuration $\langle C_1{=}v_1, \ldots, C_n{=}v_n \rangle$.

Query mixes: The typical workload in a database system consists of a mix of queries of different types, running concurrently and *interacting* with each other. These interactions may be due to hardware resource characteristics (e.g., L2 cache conflicts, buffering in the storage controller), software resource limitations (e.g., locks, buffer pool), data correlations, and so on. Thus, optimizing performance requires reasoning about *query mixes* and inter-query interactions, alongside individual queries [1]. Current database systems have little support in this direction. The space of possible query mixes is large and time-varying, so to capture the effect of inter-query interactions we need performance data from a variety of mixes when they run on the system.

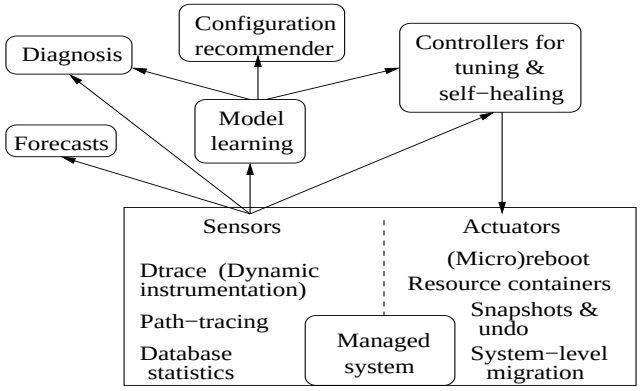

Figure 1: Data-processing pipeline in a self-adaptive system

Query plan selection: Cost models for predicting the performance of query execution plans are a foundational substrate of database systems. For each query, the system will consider a number of different execution plans, use a cost model to predict the execution time of each plan, and then use the plan with minimum predicted execution time to run the query to completion. The rapid evolution of storage systems (e.g., virtualization, smart I/O scheduling) and data patterns is causing estimates from traditional cost models to be increasingly inaccurate, sometimes by many orders of magnitude [3]. Thus, cost models need to be updated online using run-time observations of plan performance. For example, fine-grained monitoring of the difference between estimated and actual performance for a number of plans could help pinpoint the cause of inaccuracy.

There are two predominant ways in which instrumentation data is generated from systems today:

- *Preproduction or offline testing:* Instrumentation data can be collected from runs of the system before it goes into production use, e.g., when the system is subjected to load tests. Furthermore, data could be collected from runs on a (separate) development platform, e.g., runs done as part of application testing and tuning.
- *Production-time monitoring:* Once the system is in production, a variety of products (e.g., HP OpenView, IBM Tivoli) are available for monitoring performance.

We argue that these modes of data collection are inadequate to generate the data needed by potential solutions to long-standing problems like those listed above. It is hard to generate real workloads during preproduction testing. Furthermore, the collected data may become unrepresentative of system behavior if the configuration changes (e.g., the mere addition of an index can change a database's I/O pat-

terns). At the same time, there are limitations on the data that can be collected at production time. For example, we can collect $\langle C_1 = v_1, \ldots, C_n = v_n, Perf = p \rangle$ data points only for the limited number of configurations actually used in the system; which will give a biased representation of the space of configurations. Similarly, only the performance of query plans and mixes used in the system can be observed.

We need a solution where the appropriate instrumentation data can be generated on demand, especially in a production system. The (new) mechanism we identify is the concept of planned, online *experiments* in running systems. For example, in the context of tuning configuration parameters, an experiment will be a run of the system for a chosen configuration to produce a data point of the form $\langle C_1 = v_1, \ldots, C_n = v_n, Perf = p \rangle$. In the case of query interactions, an experiment may schedule queries to create a chosen mix in the system, and observe its performance. Another example experiment may run a plan p that the database would not have normally picked, simply to observe p's performance.

If experiments generate the desired instrumentation data, then why are they not being employed in systems today? The answer is that there are significant technical and psychological barriers that need to be overcome. The rest of this paper illustrates why removing the technical barriers and demonstrating the usefulness of online experiments (to remove psychological barriers) is a 5-10 year challenge.

System-level support: Today, most system administrators will not allow experiments to be conducted on the production system since it is unclear how they will affect the production workload. Experiments could have large startup and running costs. (For example, consider experimenting with different index configurations in a terabyte database system.) Thus, supporting online experiments poses a challenging system design problem. We need systems where an administrator can specify: "Use up to 20% of the system's resources to generate useful data through experiments, but do not harm the production workload by more than 10%".

Promising recent technologies can provide the necessary building blocks, e.g., fine-grained resource control and isolation through OS-level resource containers and virtualization [12], fast system-level snapshots and undo [6], efficient system-level suspension, resumption, and migration, fine-grained and dynamic instrumentation [12], and accurate workload replay. The emerging cloud computing paradigm offers tremendous potential for online experiments. (Search engines are known to experiment online with new services on real users.) Efforts to redesign the Internet could consider grassroot-level support for secure online experiments.

Planning experiments: Choosing the right experiments to conduct is very challenging for a number of reasons.

- The space of experiments is usually very large.
- Conducting experiments creates a cost-benefit tradeoff known as the *exploration-exploitation* tradeoff in machine learning. (For example, should we run a known good plan, or try out another plan that could be better?)
- It is advantageous to pick experiments that can be merged seamlessly with the regular production workload.

A number of theoretical frameworks can be leveraged to attack this problem: statistical design of experiments [9], active (sequential) learning and bandit problems from machine learning [5], and stochastic learning. We have obtained promising initial results for experiment selection in the context of learning cost models [11], scheduling query mixes [1], and diagnosing system failures [8].

Pulling uncertainty up the data pipeline: When experiments are costly and come from a large space, we may never be able to cover the full space of instrumentation data required. Hence, dealing with incomplete data and the resulting uncertainty will have to be a cornerstone of autonomic computing. As one example, it may be desirable to to recommend system configurations that give robust performance given the uncertainty in predicting system behavior, instead of ignoring this uncertainty and going after the (illusional) optimal configuration. Database researchers have made initial attempts towards capturing the uncertainty in plan cost estimates, and using that to pick plans robust to deviations of actual values from estimates [3]. Other recent advances in this context include adaptive query processing [2] and noise-aware learning for automated diagnosis [4].

Pushing accuracy needs down the data pipeline: Similar to percolating the uncertainty in the instrumentation data up the data-processing pipeline, it is desirable to push the accuracy needs of models trained from the data down to the experiment selection (i.e., data generation) phase. For example, if a 70% accurate cost model is sufficient for effective plan selection, then we can reduce the data generation overhead significantly compared to the need for a 90% accurate model. The challenge that arises is how to quantify and estimate the accuracy needs of higher-level modules like controllers, optimizers, and configuration recommenders.

In this paper, we argued that generating the right instrumentation data is key to the effectiveness of many techniques proposed for adaptive systems. While this problem may look deceptively simple, it is worthy of being considered a grand challenge in autonomic computing. We outlined a solution approach that involves mechanisms and policies for conducting experiments on demand in production systems.

1. REFERENCES

[1] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Qshuffler: Getting the query mix right (poster). In *Proc. of the 2008 Intl. Conf. on Data Engineering*, 2008.

[2] R. Avnur and J. Hellerstein. Eddies: Continuously adaptive query processing. In *Proc. of ACM SIGMOD*, 2000.

[3] S. Babu, P. Bizarro, and D. DeWitt. Proactive re-optimization. In *Proc. of ACM SIGMOD*, 2005.

[4] S. Babu, S. Duan, and K. Munagala. Processing diagnosis queries: A principled and scalable approach (poster). In *Proc. of the Intl. Conf. on Data Engineering*, Apr. 2008.

[5] M. Brodie, I. Rish, S. Ma, and N. Odintsova. Active probing strategies for problem diagnosis in distributed systems. In *IJCAI*, 2003.

[6] A. B. Brown and D. A. Patterson. Rewind, repair, replay: three R's to dependability. In *ACM SIGOPS European Workshop*, pages 70–77, 2002.

[7] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *OSDI*, Dec. 2004.

[8] S. Duan and S. Babu. Guided problem diagnosis through active learning. In *Intl. Conf. on Autonomic Computing (ICAC)*, Jun 2008. (To appear).

[9] V. Fedorov. *Theory of Optimal Experiments*. Academic Press, 1972.

[10] P. Padala, X. Zhu, et al. Adaptive control of virtualized resources in utility computing environments. In *Proc. of European Systems Conference (EuroSys)*, Mar 2007.

[11] P. Shivam, S. Babu, and J. Chase. Active and accelerated learning of cost models for optimizing scientific applications. In *Proc. of VLDB*, 2006.

[12] *Dtrace, ZFS, and Zones in Solaris*. http://www.sun.com/software/solaris.