The Knowledge Engineering Review, Vol. 21:3, 261–267. © 2006, Cambridge University Press
doi:10.1017/S0269888906000920 Printed in the United Kingdom

Towards self-predicting systems: What if you could ask 'what-if'?

ENO THERESKA[1], DUSHYANTH NARAYANAN[2] and GREGORY R. GANGER[1]

[1]*Carnegie Mellon University, Pittsburgh, PA, USA;*
e-mail: eno@andrew.cmu.edu, ganger@andrew.cmu.edu
[2]*Microsoft Research, Cambridge, UK;*
e-mail: dnarayan@microsoft.com

Abstract

Today, management and tuning questions are approached using *if. . .then. . .* rules of thumb. This reactive approach requires expertise regarding system behavior, making it difficult to deal with unforeseen uses of a system's resources and leading to system unpredictability and large system management overheads. We propose a *What. . .if. . .* approach that allows interactive exploration of the effects of system changes, thus converting complex tuning problem into simpler search problems. Through two concrete management problems, automating system upgrades and deciding on service migrations, we identify system design changes that enable a system to answer *What. . .if. . .* questions about itself.

1 Introduction

As distributed systems become more complex, so does their management. There are challenging management issues at every stage of a system's life-cycle, including initial configuration decisions, periodic hardware upgrades, and continuous tuning for changes in workload. We focus here on two difficult management problems: deciding when and how to upgrade a system and when and where to migrate system services in an environment with heterogeneous components.

For concreteness, here are some typical questions whose answers could help administrators or the system itself, in making decisions.

- *What* would happen to the throughput/latency of tasks from application A *If* the storage system is upgraded by doubling the devices' bandwidth?
- *What* would happen to system throughput *If* the system shifts the workload of user B from device X to Y?
- *What* would happen to user-perceived response lag *If* the system runs the Quake game server on a different machine with different resource capabilities?
- *What* would be the response time of requests from each of 10 virtual machines *If* machine 2 steals 256 MB of memory from machine 5?

Accurate answers to such questions are important for several reasons. First, they would allow an administrator to evaluate different upgrade options, pick the best one, and justify the cost to their superior. Second, they would guide internal system tuning decisions such as service placement.

In an era where there is much talk about autonomous systems, it is surprising that systems provide no support for answering such basic questions. Instead, most management decisions rely on *If. . .then. . .* rules of thumb. Administrators, for example, use rules of thumb to deal with the

https://doi.org/10.1017/S0269888906000920 Published online by Cambridge University Press

upgrade problem. An example rule is 'if the disk queues are large, add more buffer cache memory'. System self-tuning also currently relies on pre-programmed rules, for example, 'if service X loads the CPU beyond 90%, migrate it to a machine with a faster CPU'.

Managing and tuning a large system using rules of thumb is hard for several reasons. First, using rules of thumb requires a skilled administrator with a deep knowledge of the system, which drives the total cost of ownership of a system far beyond the initial purchase cost (Gartner Group ,2000). Furthermore, it is hard to verify the quality of the decisions made by these expert administrators. Second, an administrator using rules of thumb to manage a system cannot predict the effect of those management decisions. Estimating the effect of upgrading resources, for example, depends on the demand placed on the system by the workload, as well as the system's internal decisions on how to handle the workload and how to integrate the new resources. Hence, without the system's help, the administrator can only speculate on the outcome of a change. Third, a system built using rules of thumb is reactive and will tune itself only when certain criteria or thresholds are met, thus missing opportunities for pro-active knob tuning and exploration for better performing configurations.

The net effect of the above limitations has been system over-provisioning. We argue that money can be better spent and we have started experimenting with new ways of building self-predicting, self-managing systems. In this paper, we argue that such systems should incorporate the ability to answer *What. . .if. . .* questions about their own performance. *What. . .if. . .* interfaces convert complex tuning decisions into simpler search-based approaches. With them, iterating over a defined search space can produce a near-optimal answer.

A main contribution of our research is to identify key system design changes that will allow a system to be self-predicting. In this paper, we start by identifying operational laws as building blocks towards this solution. Operational laws require detailed monitoring of system resources and end-to-end request tracking, but do not depend on any assumptions of workload or service patterns. Using these data, they can estimate the performance impact of workload or device changes. Two real management problems, system upgrades and service migration, are used to make concrete the insights we provide on how a system should be built to enable self-prediction.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 details how a system should be designed for predictability. Section 4 discusses some preliminary experiences with predicting the effect of resource upgrades. Section 5 summarizes the paper and discusses future directions, including planned deployment in a large-scale storage service.

2 Common approaches

Choosing the best way to upgrade a system and determining where a system service should run are two faces of the same coin. Service migration is an internal system activity aimed at improving the way existing resources are used and maximizing the performance of services running in the system. System upgrades are externally induced and are aimed at identifying what new resources are needed to reach desired performance levels. This section surveys the common approaches to these two problems.

Over-provisioning: From talking to real system administrators, it is clear that the most common approach in practice is over-provisioning. This approach has several drawbacks. First, for multi-million dollar systems, over-provisioning is expensive. Second, the effects of over-provisioning on workload performance cannot be quantified. Third, over-provisioning does not help in internal system tuning decisions.

Human experts: System administrators can be considered as an external expert system paid to act on their rules of thumb. An example of such a rule is 'if the disk queues are large, then upgrade the amount of buffer cache'. The administrator is expensive because that person needs to have a deep understanding of the system and workload, and past experiences with similar cases. The administrator gets little help from the system itself. The state of the art in the kind of 'help' that comes from commercial database and operating systems is the exporting of hundreds of

https://doi.org/10.1017/S0269888906000920 Published online by Cambridge University Press

performance counters to the administrator (e.g., see systems like the ones in IBM (2004); Microsoft (2005); Oracle (2004)), who has to filter out unimportant information.

Internal expert systems: In self-tuning system designs, the human expert is usually replaced by a computer expert, which today uses the same *If... then...* approach in its decision making. This approach has been used for resource partitioning across virtual machines (Waldspurger, 2002) as well as general server resource management (Sullivan & Seltzer, 2000). Such expert systems use static or naively dynamic resource partitioning rules and do not attempt to quantify the resulting change in performance. For example, they might observe high paging rates in one virtual machine and respond by giving it memory stolen from other virtual machines. However, if the paging I/O largely overlapped with useful CPU work, this will not improve application performance. In addition, rule-driven expert systems require the designer to come up with arbitrary thresholds that would trigger action. These thresholds, and the rules themselves, may be incorrect or outdated (Williams, 1990).

What...if... explorations: A few previous approaches have successfully used *What...if...* explorations to optimize performance, especially in the area of capacity planning. Ergastulum by Anderson *et al.* (2001a) computes a good initial configuration of a storage system by iterating over the space of possible workload characteristics and storage device models. Indy by Hardwick *et al.* (2001) identifies performance bottlenecks in a running system and attempts to predict the bottleneck shifts resulting from a resource upgrade. Although steps in the right direction, these approaches treat the system as a black box, hence the help they get from the system is limited. These approaches still require an expert who knows what kinds of workloads the system should be able to support and who can provide required system models, all from an external view of the system. We propose building systems with self-prediction at their core, sidestepping the need for this external expert.

3 Designing for predictability

A fundamental assumption in system design is that the past is a good indicator of the future. This has not only allowed traditional optimizations such as caching and prefetching to work well, but also implicitly underlies the usage of simple off-the-shelf machine learning tools to predict file usage Mesnier *et al.*, (2004), isolate faulty components (Chen *et al.*, 2002), or identify component bottlenecks (Aguilera *et al.*, 2003).

We combine the assumption of *stability* over time with that of *predictability* over variation in inputs. In other words, we expect that at a macro level, system performance will be some stable function of the available resources and the workload. The goal of prediction is then to approximate this function as closely as possible. In this section, we discuss some insights about different approaches to building predictive models and describe new design requirements for a self-predicting system.

3.1 What we are and are not targeting

Our predictive infrastructure will not help with the initial purchase decision. To do so is difficult because it requires specifying up-front the characteristics of workloads that may run on a new system, which, as (Ganger (1995) shows, is unrealistic and difficult to do. However, once observations from continuous runs of a live system are made, we want to quantitatively determine the performance effect of upgrades, for example, replacing the disks in the RAID-5 box with ones with double the bandwidth.

We are not targeting accurate predictions in the face of drastic unseen-before changes in workload patterns. In that case, the past history no longer helps in predicting. For example, if last month the workload from application X was I/O-intensive, the predictive module will most likely suggest and quantify the change in the workload's throughput from buying faster disks. However,

https://doi.org/10.1017/S0269888906000920 Published online by Cambridge University Press

if the workload from that application all of a sudden becomes CPU-bound for the next month, then the suggested upgrade will not be useful. We believe that identifying permanent qualitative workload changes and re-evaluating the system when that happens is the way to go. Machine learning techniques could prove useful in detecting such changes, however this topic is beyond the target of this paper. Part of our experience, however, will be understanding if and when workloads change.

Our goal is prediction across variation in system resources and configuration. We also target prediction of the effects of simple quantitative workload changes, for example speed-up factors. In addition, we also want to predict the effects of existing workload or service migration across devices with different resources.

3.2 *Operational laws to black-box models*

All the questions mentioned in Section 1 can be answered at some level of accuracy by using operational laws (Denning & Buzen, 1978). One example of a commonly used operational law is Amdahl's law, which bounds the potential performance improvement from speeding up any part of the system.

In general, operational laws mean *measurement laws*. They depend on detailed live system measurements and their power comes from their predictive capability without the need for assumptions about workload or service patterns. Using them, the system can continuously track the bottleneck resources and provide estimates on what would happen to performance metrics of a service if those resources were to be upgraded (or equivalently if services were to move to new machines with different resources). From those estimates, the *What. . .if. . .* modification question can be answered.

There are system components that are not possible to analyze using operational laws. For example, buffer cache hit rates depend on the eviction algorithm, which is not easily captured by operational laws. In those cases, our predictive framework uses simple simulation modules that we advocate should be part of these components. For example, to predict the effect of the buffer cache size on request latency, together with operational laws that monitor the storage system and CPU utilization, a simulation module is needed to predict the miss rates with the new memory. We expand on this example in Section 4.

For components that are built using legacy software, and that do not allow source modification, operational laws can only treat them as a black-box, identify bottlenecks at a coarser-grained level and make aggregate predictions (Aguilera *et al.*, 2003).

Hence, there is a spectrum of methods to be used for making a system self-predictive. Operational laws should be used when a new system is built, together with simulation of internal algorithms. Black-box machine learning should be used for legacy systems as a last resort because of its coarseness in prediction.

3.3 *System design requirements*

This section identifies some essential design principles for systems to answer *What. . .if. . .* questions.

System load characterization: We move away from attempts at characterizing workloads towards characterizing the load workloads place on systems. Workload characterization is an area that has attempted, for a long time, to identify key parameters of workloads that are representative and use them in place of the real requests the system saw. This area has been largely unsuccessful as Ganger (1995) shows. To predict the effect of resource changes on workloads, the system traces the full activity generated by a workload on the system resources. This enables realistic predictions, by allowing separation of the demand process, which depends on the workload, and the service process, which depends on system resources and configuration.

https://doi.org/10.1017/S0269888906000920 Published online by Cambridge University Press

End-to-end activity tracking: We move away from aggregate statistics towards end-to-end tracking, like in Magpie (Barham *et al.*, 2004), because aggregate statistics cannot answer two kinds of important questions. The first relates to fine-grained predictions. For example, out of a workload with many users' the administrator might only be interested in the performance of one 'premium' user with a service-level agreement. End-to-end tracking allows these fine grained predictions by monitoring the behavior of individual requests. The second kind of question that aggregate statistics cannot answer relates to finding the true bottlenecks in the system. The classic question of 'the disk queues appear large, should I buy more memory or faster disks?' cannot be answered using aggregate statistics, which are simply observations that the disk queue size is large. End-to-end tracking allows proper accounting of resource utilization and concurrency in the system. For example, if most I/O requests overlap with useful CPU work, increasing the amount of memory or buying faster disks will not affect throughput (although it will affect request latency).

We advise aggressive instrumentation of all potential resource usage points: in our experience, disabling existing instrumentation at runtime is far easier than retrofitting missing instrumentation to a deployed system.

Co-operation with system components: Our goal is to integrate a predictive framework in the core of every system. As mentioned in Section 3.2, components that cannot incorporate operational laws naturally, need to use simple simulation methods to integrate with the rest of the predictive system. Such components include software algorithms, for example, the buffer cache manager's eviction decisions, which may change the internal workload depending on the resources used, for example, if more memory was available. Such components should be built with these simulation methods in them.

What. . .if. . . interfaces: Existing systems do not have the administrator interfaces that we need to enable exploration. With the development of the systems we are proposing, there will be a need for new, simple interfaces that allow administrators to ask *What. . .if. . .* questions. For self-tuning, the system itself will need to have these interfaces internally; for upgrades, these interfaces will be visible to the administrator, with the system taking an active part in making pro-active *suggestions* as well.

Resources for monitoring and analysis: Spare CPU and storage resources will be needed to collect and analyze end-to-end traces. Preliminary evaluations show that at least for collecting the data the CPU overhead is small (Barham *et al.*, 2004). However, managing the amount of traces generated becomes a challenge. Research on how to manage traces without losing vital information will be necessary.

4 Preliminary evaluation

We are building a large self-managing system which will be the testbed for the ideas presented in this paper (see Section 5). However, we are still at the early stages with that system, and are not yet in a position to deploy or measure its performance. As a proof-of-concept for the ideas presented here, we built a prediction infrastructure for a commercial database manager (DBMS) running on a single machine. The DBMS uses a sophisticated set of algorithms to manage the same set of resources (CPU, memory, and storage) as a larger system would, albeit on a smaller scale.

The *What. . .if. . .* question that drove this evaluation is an upgrade question: '*What* happens to the overall throughput, and the latency of each transaction type, *If* the amount of memory available to the buffer cache manager is varied?' We answer this question with a simple operational model of system performance. We predict the mean response time for transaction type X using the following equation, which is derived from well-known operational laws:

$$t_X = t'_{X/\mathrm{cpu}} + b_{X/\mathrm{io}} d'_{\mathrm{io}} \frac{1 - U'_{\mathrm{io}}}{1 - U'_{\mathrm{io}}(N_{\mathrm{io}}/N'_{\mathrm{io}})}.$$

Here the parameters derived from end-to-end tracking of the live system are $t'_{X/\mathrm{cpu}}$ (the mean critical-path computation time for type X), d'_{io} (the overall mean blocking time per blocking I/O),

https://doi.org/10.1017/S0269888906000920 Published online by Cambridge University Press

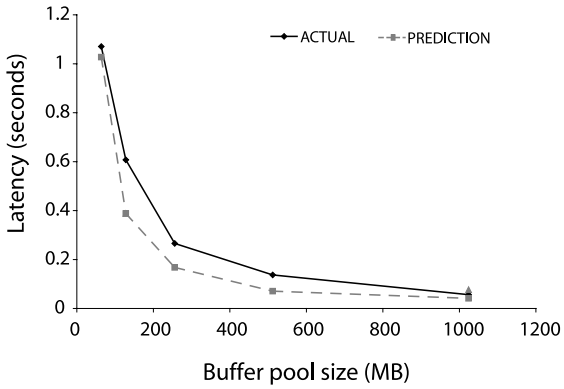


Figure 1 Predicting response time of "new order" transactions in a concurrent OLTP environment

U'_{io} (the I/O subsystem utilization), and N'_{io} (the overall number of I/Os issued per transaction). To predict $b_{X/\text{io}}$ (the predicted number of blocking I/Os per transaction of type X) and N_{io} (the predicted overall number of I/Os per transaction), we modeled the DBMS's buffer cache manager using a simple eviction simulator that predicts the hits and misses for a given reference trace as a function of available buffer cache size.

This simple combination of operational analysis and simulation was effective at answering *What. . .if. . .* questions. Figure 1 shows the accuracy of predicting response time for the main transaction type ('new order') of an On-line Transaction Processing (OLTP) workload, similar to the TPC-C benchmark (Transaction Processing Performance Council, 2005), as the available buffer cache memory is varied. Although the prediction accuracy is not perfect, the error is small compared to the change in the quantity predicted: across the full range of memory sizes measured, response time changes by more than an order of magnitude. More detailed results are provided in Narayanan *et al.* (2005).

The model presented above required a small amount of additional instrumentation in the DBMS source: 189 lines of code in six source files. These lines generated 17 different event types of interest, marking transaction stop/start, CPU scheduling events, buffer cache activity, and disk I/O requests and completions. The *What. . .if. . .* predictor consists of 1150 lines of code, of which around 350 are glue code interfacing to the instrumentation/event system. Our results show that even a legacy system can be made self-predicting with a small amount of carefully chosen instrumentation and simple models based on knowledge of internal system algorithms. We believe this approach will scale to larger systems designed from the beginning for predictability.

5 Summary and continuing work

Systems that can answer *What. . .if. . .* questions convert complex tuning problems into simpler search-based problems, thus simplifying considerably both internal system optimizations and administrator management.

To fully explore the *What. . .if. . .* approach, we are building support for it into a large-scale distributed storage system, which is being built with self-management in mind (Ganger *et al.*, 2003). The system will be deployed at Carnegie Mellon University (with approximately 1 PB of storage) and we will use the *What. . .if. . .* support both for internal optimization and hardware acquisition decisions.

We are designing our system using the guidelines mentioned in Section 3.3. An activity tracking library is part of every system component and it tracks tasks as they go through the system and monitors the resources they utilize. Activity will be stored in databases for ease of online querying by the internal tuning components.

The initial modules we are instrumenting are the storage manager and the metadata service. The instrumentation of the storage manager will allow insight on whether upgrades of storage devices

https://doi.org/10.1017/S0269888906000920 Published online by Cambridge University Press

or memory hierarchy are beneficial for different applications. The instrumentation of the metadata service will allow initial experience with migrating services to the most appropriate physical resources. Through this experience, we hope to learn the types of *What. . .if. . .* questions that are useful to administrators or for internal tuning.

Acknowledgements

We thank the members and companies of the PDL Consortium (including EMC, Engenio, Equallogic, Hewlett-Packard, HGST, Hitachi, IBM, Intel, Microsoft, Network Appliance, Oracle, Panasas, Seagate, Sun, and Veritas) for their interest, insights, feedback, and support. This material is based on research sponsored in part by the National Science Foundation, via grant #CNS-0326453.

References

Aguilera, M. K. *et al.* 2003 Performance debugging for distributed systems of black boxes. In *ACM Symposium on Operating System Principles*. New York: ACM Press, pp. 74–89.

Anderson, E. *et al.* 2001 Ergastulum: an approach to solving the workload and device configuration problem. Technical Report, HP Labs.

Barham, P. *et al.* 2004 Using Magpie for request extraction and workload modelling. In *Symposium on Operating Systems Design and Implementation*. Usenix Press.

Chen, M. Y. *et al.* 2002 Pinpoint: problem determination in large, dynamic Internet services. In *International Conference on Dependable Systems and Networks (DSN'02)*, pp. 595–604.

Denning, P. J. & Buzen, J. P. 1978 The operational analysis of queueing network models. *ACM Computing Surveys* **10**(3).

Ganger G. R. 1995 Generating representative synthetic workloads: an unsolved problem. In *International Conference on Management and Performance Evaluation of Computer Systems*, pp. 1263–1269.

Ganger, G. R. *et al.* 2003 Self-* Storage: brick-based storage with automated administration. Technical Report, Carnegie Mellon University, Pittsburgh, PA.

Gartner Group 2000 Total cost of storage ownership — a user-oriented approach. Research note, Gartner Group.

Hardwick, J. *et al.* 2001 Modeling the performance of E-commerce sites. In *27th International Conference of the Computer Measurement Group*, vol. 105:3.

IBM 2004 DB2 Performance Expert. `http://www-306.ibm.com/software/data/`.

Mesnier, M. *et al.* 2004 File classification in self-* storage systems. In *IEEE First International Conference on Autonomic Computing (ICAC-04)*. Piscataway, NJ: IEEE Press.

Microsoft 2005 Performance counters reference for Windows Server 2003. `http://www.microsoft.com/windowsserver2003`.

Narayanan, D. *et al.* 2005 Continuous resource monitoring for self-predicting DBMS. In *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. Piscataway, NJ: IEEE Press.

Oracle 2004 Oracle database manageability. `http://www.oracle.com/technology/products/`.

Sullivan, D. and Seltzer, M. 2000 A resource management framework for central servers. In *USENIX Annual Technical Conference*. Usenix Press, pp. 337–350.

Transaction Processing Performance Council 2005 TPC Benchmark C Specification. `http://www.tpc.org/tpcc/`.

Waldspurger, C. A. 2002 Memory resource management in VMWare ESX server. In *Symposium on Operating Systems Design and Implementation*. New York: ACM Press.

Williams, J. 1990 When expert systems are wrong. In *ACM SIGBDP Conference on Trends and Directions in Expert Systems*. New York: ACM Press, pp. 661–669.

https://doi.org/10.1017/S0269888906000920 Published online by Cambridge University Press

https://doi.org/10.1017/S0269888906000920 Published online by Cambridge University Press