Clockwork: A new movement in autonomic systems

by L. W. Russell
S. P. Morgan
E. G. Chron

Statically tuned computing systems may perform poorly when running time-varying workloads. Current work on autonomic tuning largely involves reactive autonomicity, based on feedback control. This paper identifies a new way of thinking about autonomic tuning, that is, predictive autonomicity, based on feedforward control. A general method, called Clockwork, for constructing predictive autonomic systems is proposed. The method is based on statistical modeling, tracking, and forecasting techniques borrowed from econometrics. Systems employing the method detect and subsequently forecast cyclic variations in load, estimate the impact on future performance, and use these data to self-tune, dynamically, in anticipation of need. The paper describes a prototype network-attached storage system that was built using Clockwork, demonstrating the feasibility of the method, and presents key performance measurements of the prototype, demonstrating the practicality of the methods.

Large computing systems, especially those running time-varying workloads, are difficult to keep tuned. Dozens of interacting parameters may need to be understood and adjusted. Even if a system is tuned well at one point, because of changing workloads it may end up being poorly tuned at some other point. Badly tuned systems not only perform poorly, they also waste resources and frustrate users.

There is substantial and growing interest in autonomic systems, that is, systems that dynamically self-regulate. A key aspect of self-regulation is self-tuning. Current work on autonomic tuning is only slightly more advanced than static tuning; largely, such work revolves around primitive notions of *reactive autonomicity*, based on feedback control. Reactive autonomic systems reconfigure on the basis of instantaneous need or, at best, on the basis of short-term historical measurements. As with any techniques involving feedback control, reactive autonomic systems carry with them the well-known problems of potential instability or slow response to change.

In the next section of this paper, we propose a new approach to the problem. We introduce the concept of *predictive autonomicity*, based on feedforward control. We outline a general method, which we call *Clockwork*, for constructing predictive autonomically tuned systems. Using statistical modeling, tracking, and forecasting techniques borrowed from econometrics, systems employing the Clockwork method detect and forecast cyclic variations in their load, estimate the impact of the variations on future performance, and use these data to reconfigure themselves, in anticipation of need.

The third section describes a prototype, scalable network attached storage (NAS) system that we built using Clockwork, demonstrating the feasibility of the method. A *network attached store* is a network file server that processes requests sent to it using a pro-

Table 1 The Clockwork method

Step	Electric Utility	Retail Chain	NAS Plex
1. Establish system objective	Reliability (by rate class)	In-stock ratio (by sales class)	Response time (by file or client class)
2. Establish measure of demand	Electricity, as it is being consumed	Sales, as they are being made	Requests, as they are being processed
3. Track objective with demand	Reliability, as electricity is being consumed Generator spin-up times Instantaneous capacity	In-stock ratio, as items are being sold Product distribution times —	Response time, as requests are being processed — —
4. Forecast demand	Use autoregressive time series analysis		
5. Adjust controllable parameters	Buy or sell electricity or options to buy or sell Bring generators on or off line Activate or deactivate spinning reserve	Issue store orders to distribution centers Issue purchase orders to vendors Liquidate excess inventory	Assign files to stores Copy or move files between stores Bring stores on or off line

tocol such as Network File System (NFS),[1] over a medium such as Ethernet, by one or more clients. NFS, layered in turn on the Transmission Control Protocol/Internet Protocol (TCP/IP) suite, uses a remote procedure call architecture, in which every request from a client to a server engenders a response from the server to the client. Typical NFS requests are to create a file, to write data to a file, to read data from a file, and to delete a file. A response indicates whether the corresponding request was processed without error and, if so, contains request-specific data, for example, file contents from a read.

An NAS acts as a central repository for data shared among clients. With an NAS, clients need not each store the data, reducing cost. Clients need not coordinate updates to the data, simplifying their workings. Data management may be centralized, simplifying management and reducing costs. Small computers may be deployed widely; alternatively, large systems may be scaled further. It is desirable to have a powerful NAS to support more clients or to process more work from the same number of clients. For this paper, we prototyped one with a scalable architecture, integrating multiple stores into a single, virtual NAS. Requests are sent to the virtual NAS and are spread among the individual stores. The advantage of the architecture is that systems of various capabilities, including a very powerful system, may be built from relatively inexpensive components. The disadvantage is that the overall performance of a system will be only as good as that of its worst performing store. Although a virtual NAS could be massively overprovisioned to minimize the effect of one poorly performing store, that would reduce the advantage of the architecture. Alternatively, autonomic tuning could be used to balance load among the stores. We chose the latter approach.

Key performance measurements of the NAS prototype, demonstrating the practicality of the method, are presented in the fourth section. Finally, in the fifth section, directions for future work are suggested.

The Clockwork method

Clockwork is a general method, analogous to those already in wide industrial use by electric power utilities and retail chains, for example. It enables a predictive autonomic system to be implemented following five simple steps, summarized in Table 1. The first two are configuration steps. They establish a system objective and a means to track it with load. The remaining three are operational steps. They automatically and continually track, forecast, and control the system.

A system that cannot be measured cannot be managed. Clockwork first establishes a simple, quantifiable objective, comprising a performance objective and a confidence level. For an electric utility, an appropriate performance objective would be to meet the instantaneous demand for electricity reliably. A potential performance objective for a retail chain

would be to achieve a certain in-stock ratio, a measure of how much product is in stock at a given time. For an NAS, achieving a certain average response time would be a suitable performance objective. The confidence level measures how closely the system must meet its performance objective. For example, the electric utility might need to meet demand 99.99999 percent of the time, the retail chain might need to achieve the in-stock ratio 90 percent of the time, and the NAS might need to achieve the average response time 66 percent of the time.

Often, objectives are subclassed. Some electric utility customers may be willing to trade decreased reliability for lower cost, some retail chains may require tighter controls on in-stock ratios for more profitable products, and some NAS clients may be willing to trade increased response time for lower cost. Although for brevity, the present discussion ignores subclassed objectives, Clockwork can handle them.

Clockwork, in the second step, establishes a simple, quantifiable measure of demand. An appropriate measure for an electric utility would be electricity being consumed; for a retail chain it would be sales being made; and for an NAS, it would be requests being processed.

Tracking the objective (and its variance) in relation to demand is the third step. An electric utility would track how reliably it met electricity demand, the time it took (or would take) for generators to be spun-up, and instantaneous capacity, as electricity was being consumed; a retail chain would track product in-stock ratios and product distribution times, as sales were being made; and an NAS would track response time, as requests were being processed.

In the fourth step, demand is forecast, along with uncertainty, using autoregressive time series procedures. This technique projects future values of a variable based on the history of that variable alone, which simplifies forecasting considerably. A key contribution of Clockwork is that *the same procedure would be used by the utility, the retail chain, and the NAS*.

Fifth and finally, the controllable parameters of the system are adjusted to meet the objective. In anticipation of forecast demand: the electric utility would bring its generators on or off line, would buy or sell electricity or options to do the same, or would activate or deactivate its spinning reserve; the retail chain would issue store orders to its distribution centers, would issue purchase orders to its vendors, or would liquidate its excess inventory; and the NAS would reassign files to stores, would replicate files among or migrate files between stores, or would bring stores on or off line.

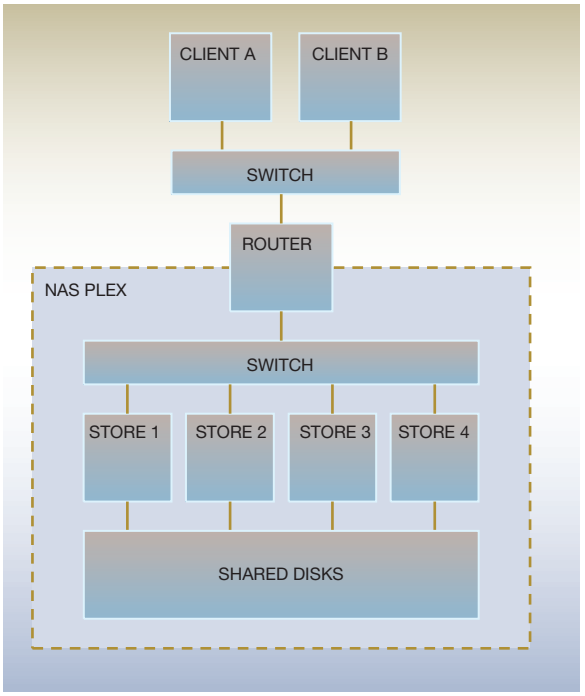
The prototype

In this section, we describe how we used Clockwork to prototype a scalable, autonomically tuned NAS. Our purpose in building the prototype was to determine whether the method is feasible and practicable, rather than to achieve optimal performance. Nevertheless, as the measurements in the next section show, the prototype performs well. For proof of concept, and because we were able to operate in a shared-disk environment, we implemented file reassignment, but not file replication (copying a file to multiple stores) or migration (moving a file between stores). Had we been faced with a serially shared disk or a shared-nothing environment, we would have had to have implemented replication and migration.

The prototype comprises three main components: a set of stores, or storage servers, that process requests for files kept in a cluster file system, a request router that spreads requests among the stores, and an autonomic control program that directs the router, following the Clockwork method. We call the overall system an *NAS plex*, as it integrates multiple, otherwise independent systems. The prototype NAS plex is depicted within the dashed-line area of Figure 1. It includes four stores, a router, an internal network, and shared disks. Two clients are connected to the plex via an external network.

The clients, the router, and the stores are computers with an Intel architecture. With the exception of the router, all computers run the Linux** operating system. The router runs a real-time operating system to minimize latency and runs the Clockwork control program. The stores share files via the General Parallel File System (GPFS)[2] cluster file system, which manages fibre channel disks. The prototype is interconnected via Fast Ethernet. Clients access files via NFSv3/UDP (Network File System version 3/User Datagram Protocol). Although the clients are configured identically, the stores deliberately are not, so that the prototype is inherently unbalanced (see below). The stores contain processors of various speeds. Some stores have one processor, whereas others have two. Stores have different amounts of memory. We used GPFS[2] because it is a robust IBM product that supports the hardware and software

Figure 1 The prototype NAS plex, as built



used in the prototype.[3] GPFS implements a scalable, shared disk architecture. Although the prototype used GPFS, the IBM Storage Tank*[4] storage area network (SAN) file system was a viable alternative.[5]

The prototype works as follows. A client sends a request to the router, which forwards it to a store for processing. Any store may access any file, since files are managed by a cluster file system, which coordinates accesses to them. Which store will process a given request is a decision made by Clockwork based on the type of the request, the file to which it refers, and the state of Clockwork. The decision process is described in more detail below. The architecture enables the load of the NAS plex to be shared among its stores.[6] Load balancing, or intelligently sharing load, has two main benefits. First, as with any modern computer system, performance is nonlinear. Past a saturation point, a linear increase in load causes a much greater increase in response time. Load balancing can keep the plex operating within a linear performance region. Second, assigning related requests to the same store can take advantage of data caching, thereby keeping the number of I/Os, and the amount of computation, low.

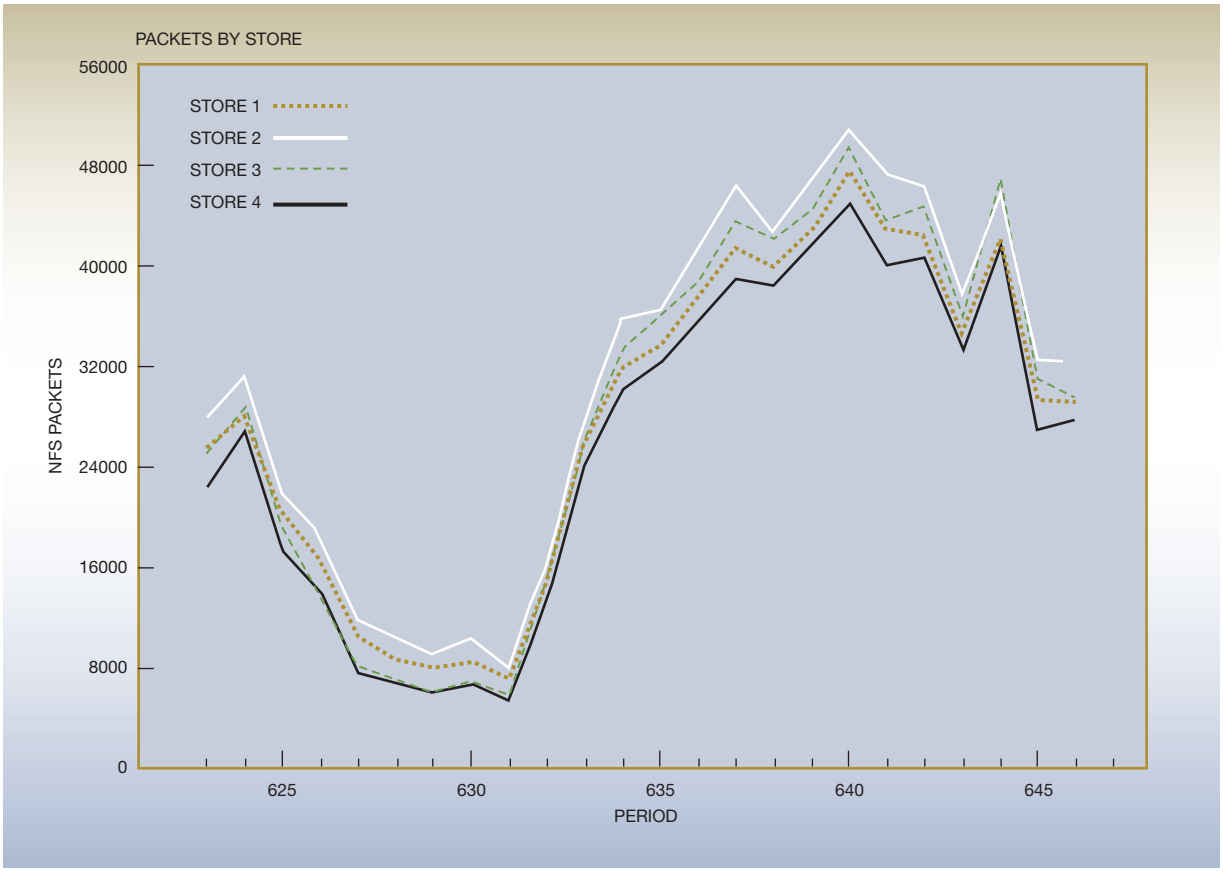
For this architecture, load could be balanced statically, that is, files could be assigned to stores following a fixed schedule, or it could be balanced dynamically, with file assignment changing over time. In reality, static assignment would prove a poor choice. Requests arriving clustered in time tend to be related, load tends to include multiple cyclical components, and load tends to vary substantially over time. The prototype is dynamically balanced using feedforward control.

The router is NFS request- and response-aware. It analyzes and routes requests and responses at network speeds. The router records statistics on a per-file, per-request basis, as well as on a per-store, per-response basis. It forwards requests to appropriate stores using a default rule and an exception set. The default rule—the prototype uses a simple hash of the NFS file handle (or file identifier) to choose a store—has several characteristics. It spreads load more or less uniformly among the stores. It repeatably assigns a given file to the same store. It is simple to compute. Given these characteristics, the rule takes advantage of store data (and cluster file system token) caching; however, it assigns files to stores statically, ignoring store load and file heat, that is, the extent to which the file contributes to load.

Using the statistical data gathered by the router, the Clockwork control program periodically: tracks and forecasts store response time at a given load; tracks and projects per-file heat; estimates the effect on response time of reassigning hot files to stores; decides which files to reassign, and updates the exception set of the router to reassign the files. On the assumption that there are cyclical components to access patterns, the projections and assignments of Clockwork are refined over time, as its statistical database grows. Clockwork detects and adjusts rapidly to any fundamental changes in access patterns.

Clockwork projects the expected load of each file using well-known time series analysis methods borrowed from econometrics. In particular, Clockwork models load using an Autoregressive Integrated Moving Average (ARIMA) model[7] from which it extracts cyclical components. Clockwork applies Geweke's Spectral Forecasting procedure[8] to the components to forecast future load from present load. In essence, the number of requests per period is viewed as an infinite moving average, a Fourier transform of the time series is estimated, a corresponding time-domain model is computed, and the model is used to forecast load. As the same model

Figure 2 Default distribution of NFS requests, by store



applies to all such series, the procedure can be automated.

Using the load forecast, Clockwork determines which stores, if any, are likely to be overloaded in the next period. It iteratively proposes a reassignment of files from overloaded to underloaded stores. Files are proposed for reassignment in descending order of heat. Iteration terminates when the performance objective of the plex is achieved or, if the objective cannot be achieved because of plex overload, when the load is balanced.

Given a proposed assignment, Clockwork estimates the response time of a store using Hannan's Efficient Estimator,[9] a spectral procedure for estimating generalized least squares. This procedure is applicable assuming that all factors taken together, other than the number of requests processed in a period, follow a stationary ARIMA process. In practice, this assumption has proven reasonable. Because the same model applies to all data series, the procedure again can be automated.

Rather than use a default rule and iteratively proposing reassignment of a few hot files, the prototype could have computed an optimal assignment following a stochastic optimization model, with Benders decomposition and Lagrangian relaxation. See Dentcheva and Romisch[10] for examples of such computations. The model is completely specified, both from a mathematical standpoint and in terms of statistical estimation procedures. The procedures require historical data on load and response time, which the router gathers and records. In reality, such a computation would be highly complex and slow. Given the existence of a simple default rule, Clockwork has an adequate starting point from which to

Figure 3 Measured average response time by store, without reassignment

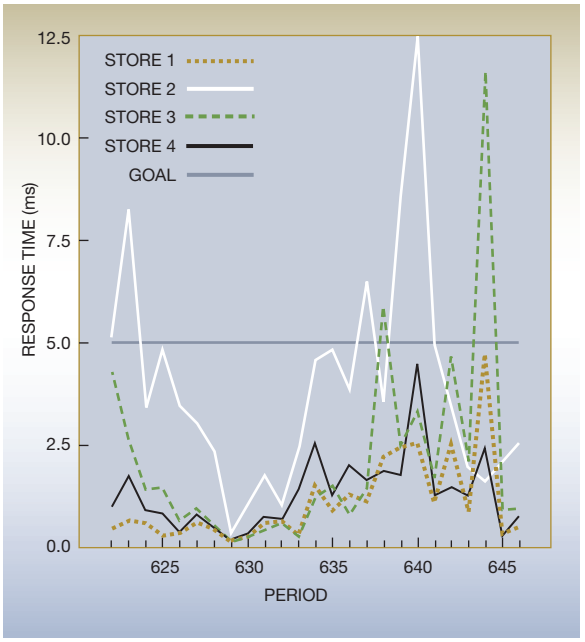
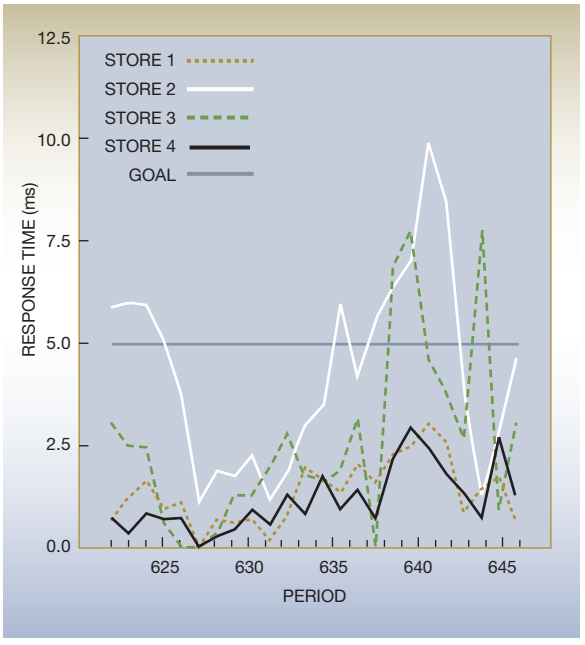


Figure 4 Forecast average response time by store, without reassignment



iteratively apply incremental changes, which quickly leads to good results. There is no need to apply a more complex procedure.

Measured results

There are no generally accepted, long-running NFS traces suitable for evaluating the prototype. For predictive systems, synthetic workloads are inappropriate, because they invariably contain artificial cycles or are highly random, leading either to perfect results or perfectly useless ones. Lacking real workloads, yet desiring independently reproducible results, we generated NFS workloads from four essentially different HyperText Transfer Protocol (HTTP) traces we downloaded from HitBox.[11] We chose those from a "fantasy" soccer site on which users create virtual teams with which they play virtual matches, a memorabilia site on which users trade sports and other memorabilia, a name definition site, which expectant parents use to choose a name for their baby, and an MP3 download site.

We used Fstress,[12] an NFS benchmarking tool, to generate the actual workloads from the HTTP traces. For each, we constructed an appropriate set of files, numbering over 1100, total. We determined a base load, at which all four workloads were issuing requests at a heavy rate, and under which the plex was stressed; that is, its response time was changing nonlinearly as a function of load. We evaluated the system with the workloads running simultaneously and independently. First, we ran the workloads with file reassignment disabled, and again with it enabled. The results given below correspond to a representative 24 hours of the trace, starting 622 hours into it.[13] Each period corresponds to one hour of the trace.

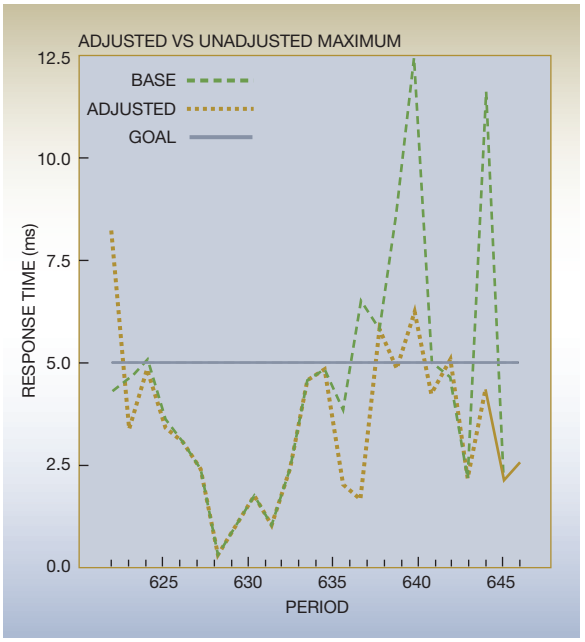
Figure 2 shows the distribution of NFS requests by store following the default rule. The rule tends to spread requests evenly among the stores. Figure 3 shows the measured average response time, by store, without file reassignment. Clearly, the stores perform very differently. Figure 4 shows a forecast average response time, by store. In a comparison of Figures 3 and 4, the projections seem acceptably close. Notably, Clockwork detects the differences among the stores and projects them forward.

Next, we reran the traces through the prototype with the file reassignment of Clockwork enabled, and with an appropriate objective: to achieve a 5 ms or better average response time at a 66 percent confidence level. The goal was chosen to demonstrate the prac-

ticality of the method, not to achieve the best possible results. Other goals could have been chosen that also would have demonstrated practicality, for example, reducing average response time by 10 percent. With the chosen goal, files were reassigned in 11 of the 24 periods. In 10 periods, files were reassigned from Store 2; in three periods, files were reassigned from Store 3; and in two periods, files were reassigned from both Stores 2 and 3. In all cases, the files were reassigned to Store 4, the best performing store.

Figure 5 shows the effect of the reassignments on response time. The graphs show the maximum of the average per-store response times, where base indicates the measured times without reassignment, and adjusted indicates the times with reassignment. The prototype achieved the performance component of its goal, or came very close, in nearly all periods. It missed by more than the calibration error only in periods 637 and 639. Given the confidence level chosen, it achieved its overall goal. It is notable that, during Periods 639 and 643, including one of the periods in which it missed its performance goal, the prototype shaved the maximum average response time of the plex nearly in half.

Figure 5 Measured maximum average response time, with and without reassignment



Future work

We are continuing this work in several areas. We have extended the router to translate incoming NFS/TCP connections to NFS/UDP inside the plex to balance a connection-oriented NAS protocol. We have projected per-file workloads multiple periods out, with encouraging results. Given the multiperiod results, we believe it will be possible to balance load by file replication and migration, extending the method to serially shared-disk and shared-nothing environments.

Re-examining Figure 5, we see that the prototype incorrectly forecast an overload in periods 624 and 641, which led to slightly worse response times. Although we argue against feedback control as the sole method for autonomic tuning, integrating some form of feedback control with Clockwork may improve the method. In the noted periods, a real-time monitor could have detected that actual load was deviating from the forecast, and might temporarily have overridden, or perhaps canceled, reassignment. It is unclear what steps should be taken in general. See Wang and Morris[14] for a comprehensive study of load monitoring and balancing techniques, including some that may be appropriate for integration with Clockwork.

Conclusions

In this paper, we proposed a new approach to autonomic systems. We introduced the concept of a predictive autonomic system, which regulates its behavior in anticipation of need, using statistical modeling, tracking, and forecasting procedures. We proposed the Clockwork method for autonomic systems. We demonstrated the feasibility of the method, using it to prototype a self-tuning NAS plex. We presented measurements of the prototype under substantial workloads. The measurements demonstrate the practicality of the method. Finally, we discussed future work.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Linus Torvalds.

Cited references and notes

1. B. Callaghan, B. Pawlowski, and P. Staubach, *NFS Version 3 Protocol Specification*, Request for Comments 1813, Internet Engineering Task Force, Mountain View, CA (June 1995).

2. F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, USENIX Association, Monterey, CA (January 2002), pp. 231–234.
3. GPFS also runs on IBM RISC System/6000® SP™ parallel systems.
4. Storage Systems Group. IBM Almaden Research Center, Storage Tank Web page, see http://www.almaden.ibm.com/cs/storagesystems/stortank/.
5. GPFS and Storage Tank originated at the IBM Almaden Research Center, where the present work also was done.
6. The architecture can also conceal certain system administrative tasks such as adding, removing, or upgrading a store, and can mask certain hardware or software failures.
7. G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis, Forecasting and Control,* Third Edition, Prentice-Hall, Upper Saddle River, NJ (1994).
8. J. Geweke, *Priors for Macroeconomic Time Series and Their Application*, Discussion Paper No. 64, Federal Reserve Bank of Minneapolis, Institute for Empirical Macroeconomics, Minneapolis, MN (1992).
9. E. J. Hannan, "Regression for Time Series," in *Time Series Analysis*, M. Rosenblatt, Editor, John Wiley & Sons, Inc., New York (1963), pp. 17–37.
10. D. Dentcheva and W. Romisch, "Optimal Power Generation Under Uncertainty via Stochastic Programming," in *Stochastic Programming: Numerical Techniques and Engineering Applications*, K. Marti and P. Kall, Editors, *Lecture Notes in Economics and Mathematical Systems* **458**, 22–56, Springer-Verlag, Berlin (1997).
11. HitBox, WebSideStory, San Diego, CA, available from http://www.hitbox.com/.
12. D. C. Anderson and J. S. Chase, *Fstress: A Flexible Network File Service Benchmark*, in preparation; available from Duke University, Department of Computer Science at http://www.cs.duke.edu/ari/fstress/download/paper.pdf.
13. There is no significance to this particular choice of starting period.
14. Y.-T. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems," in *ACM Transactions on Computing Systems* **34**, No. 3, 204–217 (March 1985).

Accepted for publication October 10, 2002.

Lance W. Russell *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: lancerus@almaden.ibm.com).* Mr. Russell is a research staff member in the Storage Systems Group at the Almaden Research Center. While at the IBM Advanced Workstations Division, he led the early AIX® operating system architecture and implementation efforts for TCP/IP, sockets, streams, and netware. He received an IBM Outstanding Technical Achievement Award for his work on AIX 3.1. At Hewlett-Packard Laboratories, he worked as a principal scientist on server blade architecture and dynamic resource allocation. He holds over two dozen patents and has numerous patent filings. In addition to systems architecture, his research interests include forecasting and system planning under uncertainty. He has testified on the latter subjects before the Public Service Commission of Colorado and the Federal Energy Regulatory Commission.

Stephen P. Morgan *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: smorgan@almaden.ibm.com).* Mr. Morgan is a senior software engineer in the Storage Systems Group at the Almaden Research Center. He has worked on a variety of systems projects since joining the IBM Thomas J. Watson Research Center in 1981. He was a member of the 801 Group and helped transfer its technology into the AIX 3.1 operating system. He is a coinventor of the AIX Logical Volume Manager. Mr. Morgan helped jump-start the Open Software Foundation. He developed High Availability for NFS. He was principal investigator on a Department of Defense secure workstation project, for which he received an IBM Outstanding Innovation Award. He holds more than a dozen patents and has published papers at conferences ranging from VLDB to USENIX. He is primarily interested in storage systems and computer networking. In 1982, Mr. Morgan received an S.B. degree in computer science and engineering from the Massachusetts Institute of Technology.

Edward G. Chron *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: echron@almaden.ibm.com).* Mr. Chron is a senior software engineer in the Storage Systems Group at the Almaden Research Center. He worked on hardware architecture-specific operating system technology and system software development, and on CAD and CAE (computer-aided design and computer-aided engineering) tools for chip layout and design, while at Intel Corporation. At the former IBM Palo Alto Scientific Center, which he joined in 1991, Mr. Chron developed advanced versions of the AIX operating system and of the Mach microkernel, for the IBM RS/6000 and for IBM mainframe systems. Since moving to the Almaden Research Center in 1993, Mr. Chron has worked on numerous systems projects. His research interests include operating systems, systems architecture, distributed and networking systems, and storage-related applications. He received a B.S.E.E. degree from the University of Illinois, Urbana Champaign in 1981.