

Using Fractal Analysis to Model Software Aging

Jonathan Crowell, Mark Shereshevsky, Bojan Cukic
Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV 26506-6109
{crowell, smark, cukic}@csee.wvu.edu

May 3, 2002

1 Introduction

1.1 Software Aging

A phenomenon known as “software aging” has been identified and described in recent literature [22, 19, 17]. It is important to distinguish between two varieties of software aging that are discussed. The first use of the term was introduced by David Parnas [14] and refers to the degradation of a software system over many years as maintenance of the system takes it in new directions and as the requirements of the organization change. The second use of the term, which is the one we are concerned with here, refers to the degradation of a software system over a period of hours to months as errors accumulate while the system is running. This use of the term “software aging” is applicable mainly to software systems that are designed to run for an extremely long period of time, such as a server in a client-server application, or a software system on a long-term space mission.

Computer users everywhere are familiar with the phenomena of operating system crashes and software freezes. These unexpected events usually result in a loss of time and data. The first and most important way to counteract system crashes is to design and build better software. Large software systems, however, are extraordinarily complex entities, and achieving perfection in their design and implementation is unlikely if not impossible. The next line of defense is to take proactive measures to anticipate crashes so that data can be saved and the system can be shut down, cleaned, and restarted in a rejuvenated state. This process has been described as “software rejuvenation” [22, 18, 16].

The causes of software aging are the accumulation of numerical rounding errors, the corruption of data, the exhaustion of operating system resources, unreleased file locks, memory leaks, and other similar minor malfunctions [6]. These errors are due to bugs in the system described by Gray [4] and Huang et al. [22] as *Heisenbugs* because they are non-deterministic and could remain inactive for a long time. (*Bohr bugs*, on the other hand, are deterministic and predictably lead to a software failure). *Heisenbugs* are difficult if not impossible to detect during system testing due to their non-deterministic character. Indeed, the term was first used to describe those bugs in a system that cease to occur whenever debugging or tracing flags are used during software compilation.

Since these errors are by definition extremely difficult to predict directly, we must look for other ways of predicting when a system is in a vulnerable state. Vaidyanathan and Trivedi [6] propose a semi-Markov reward model based on system workload and resource usage to estimate the time of failure of a system. The data they collected tends to fluctuate a great deal and appears chaotic rather than linear, however. The goal of this study is to suggest a method for determining whether an operating system has begun to age and is likely to crash. Our approach is based on fractal analysis of patterns of resource use in a software system.

1.2 The Fractal Approach to Software Rejuvenation

Fractal geometry is a relatively new tool within mathematics, having been developed in the 1960s and 70s by Benoit Mandelbrot [9], which can be used to model many of the complex, chaotic phenomena that appear throughout nature. It also appears that many objects of human derivation such as network traffic patterns [21] and the stock market [15, 8] display fractal characteristics. In addition, fractal analysis has been employed with success in the analysis and

synthesis of speech signals [10, 2], in the analysis of human heartbeats [13, 12], and in image compression algorithms [1, 5]. If it can be established that the patterns of resource use in operating systems display fractal characteristics, then it is hoped that these subtle patterns will betray information about the age of the operating system and can be exploited to predict the best times to engage in operating system rejuvenation.

1.3 Structure of the Report

The remaining sections of this report will be devoted to explaining the mathematical theory that underlies our approach, discussing the operating system resources we measured and their fractal characteristics, explaining the design of our experiment, suggesting possible methods for developing a model for the detection of software aging and the prediction of imminent operating system crashes, and suggesting directions for further work.

2 Fractality in Functions and Signals

A fractal function is a function whose graph displays many of the characteristics of a fractal (Falconer [3]): self-similarity, irregularity, fine structure, and fractional dimension. A function that displays variable local scaling at different points is known as multifractal. The data dealt with in this study is a time series of values for three different parameters within an operating system. The extent to which the values of these time series display fractal and multifractal characteristics will be elucidated through calculation of the Hölder exponents at each point.

2.1 The Hölder Exponent

The Hölder exponent of a function contains information about the fractality of a function at a local point, and is calculated from the degree of fluctuation in the function at that point. A highly irregular and chaotic point in a function will have a lower Hölder exponent, and a smoother portion of a function will have higher Hölder exponent. A function may have different Hölder exponents at different points due to a variation in the amount of local fractality that the function displays, thereby manifesting multifractality.

A continuous function $f : \mathcal{R} \rightarrow \mathcal{R}$ is said to be a Hölder function with exponent α if

$$|f(x) - f(y)| \leq c|x - y|^\alpha$$

for some constant c [3]. Clearly, a differentiable function is always Hölder, with exponent $\alpha = 1$. In general, if this inequality holds for α_0 , then it will hold for all $\alpha \leq \alpha_0$. The Hölder exponent of a function, then, is the upper bound of a given α .

Specifically, a function has a Hölder exponent α at point t_0 if and only if:

1. for every real $\gamma < \alpha$:

$$\lim_{h \rightarrow 0} \frac{|f(t_0 + h) - P(h)|}{|h|^\gamma} = 0$$

and

2. if $\alpha < +\infty$, for every real $\gamma > \alpha$:

$$\limsup_{h \rightarrow 0} \frac{|f(t_0 + h) - P(h)|}{|h|^\gamma} = +\infty$$

where P is a polynomial whose degree is less than or equal to α [7].

When the Hölder exponent of a function $f(t)$ at a certain point t_0 is greater than or equal to 1, the function $f(t)$ is differentiable, or locally smooth, at t_0 . The extreme case is that of a locally constant function: the definition of the Hölder exponent clearly implies that for such a function $\mathcal{H}_f(t_0) = \infty$ (since $\log(0) = -\infty$). Because the graph of the constant function behaves geometrically similarly to the linear function $f(t) = at + b$, which (for $a \neq 0$) has Hölder exponent 1, we have adopted the convention that $\mathcal{H}_f(t_0) = 1$ when f is constant in a neighborhood of t_0 .

In most of our data, however, the Hölder exponent stays within the range $(0, 1)$ which characterizes a non-differentiable, or fractal, signal.

2.2 Calculating the Hölder Exponent

Véhel and Daoudi have shown [20] that the Hölder exponent at a point t of a function $f(t)$ can be expressed by the formula

$$\mathcal{H}_f(t) = \liminf_{h \rightarrow 0} \frac{\log(|f(t+h) - f(t)|)}{\log(|h|)}. \quad (1)$$

We have developed an algorithm for computing the Hölder exponent of a signal based on equation 1: Suppose $\{y_0, y_1, y_2, \dots, y_n\}$ is a time series of length n representing some quantity measured at uniform intervals. The idea is that the degree to which the function is smooth or chaotic (and therefore the strength of its Hölder exponent) at a certain data point y_i is a function of a number of data points preceding and following it. This number, s , is called the window width and is not fixed but is rather a parameter set by the user when running the algorithm. The weight that each of the $2s$ values (s values on each side of the data point) has on the ultimate calculation of the Hölder exponent is controlled by another parameter, λ . λ is called the weighted regression coefficient and must satisfy $0 < \lambda < 1$. Adjusting the strength of λ will allow us to adjust the degree to which the data points further away from the point we are interested in influence the calculation of the Hölder exponent at that point. Obviously nearby points should have a greater weight than distant points.

To calculate the Hölder exponent at the i th point in a data set, we begin by choosing a window width, s , and a value for λ . We have used $\lambda = 0.5$ and $s = 10$ for all the Hölder exponent calculations appearing in this study.

Next, we calculate a value, $R_{i,k}$, for each integer $k \neq 0$ from $-s$ to s in the following manner:

$$R_{i,k} = \frac{\log |y_{i+k} - y_i|}{\log(\frac{|k|}{1000})}$$

In the above case, k must satisfy $0 \leq i+k \leq n$. For data points at the very beginning or end of the time series for which s values to the left or right are not available, s is shrunk to the appropriate size.

Then, in the second step, for each integer j , $1 \leq j \leq k$, calculate

$$h_{i,j} = \min\{R_{i,k} : |k| \leq j\}$$

(This corresponds to taking the \liminf in the equation for the Hölder exponent).

In the third step, calculate \mathcal{H}_i , the estimate of the Hölder exponent at the i th data point by computing the weighted average of the approximations $h_{i,j}$:

$$\mathcal{H}_i = \frac{1 - \lambda}{1 - \lambda^k} (h_{i,1} + \lambda h_{i,2} + \lambda^2 h_{i,3} + \dots + \lambda^{k-1} h_{i,k})$$

where more weight is given to terms with a smaller k (i.e., h closer to 0 in equation 1).

2.3 Accuracy of the Hölder exponent Calculation

Our algorithm for calculating the Hölder exponent of a function can be tested by running it on a function whose Hölder exponent is known. A generalized Weierstraß function is a function whose Hölder exponent can be predefined. The formula for this class of functions is

$$f(t) = \sum_{k=0}^{\infty} 3^{-ks(t)} \sin(3^k t),$$

where $s(t)$ is the seed function ranging between 0 and 1 [7]. It is known that $s(t) = \mathcal{H}_f(t)$ for all t .

Figure 1 shows the plot of our Hölder exponent estimation compared to the known Hölder exponent. In this case, the known holder exponent is the function $s(t) = |\sin(5\pi t)|$, the generalized Weierstraß function with its Hölder exponent specified at every point by $s(t)$ is shown next to the seed function, and our estimation of the Hölder exponent along with an alternative estimation of the Hölder exponent are shown in the bottom half of the figure.

Our estimation is very close and is better than some existing algorithms. It is obviously not perfect, however, the main drawback being that it fluctuates a great deal in a seemingly chaotic manner. Unfortunately, this characteristic is magnified for functions from the “real world,” such as the time-series data we deal with later in this study. This fluctuation sometimes causes plots of the Hölder exponent to appear at first glance to be intractably chaotic, but in fact trends are discernable by applying standard averaging or denoising techniques.

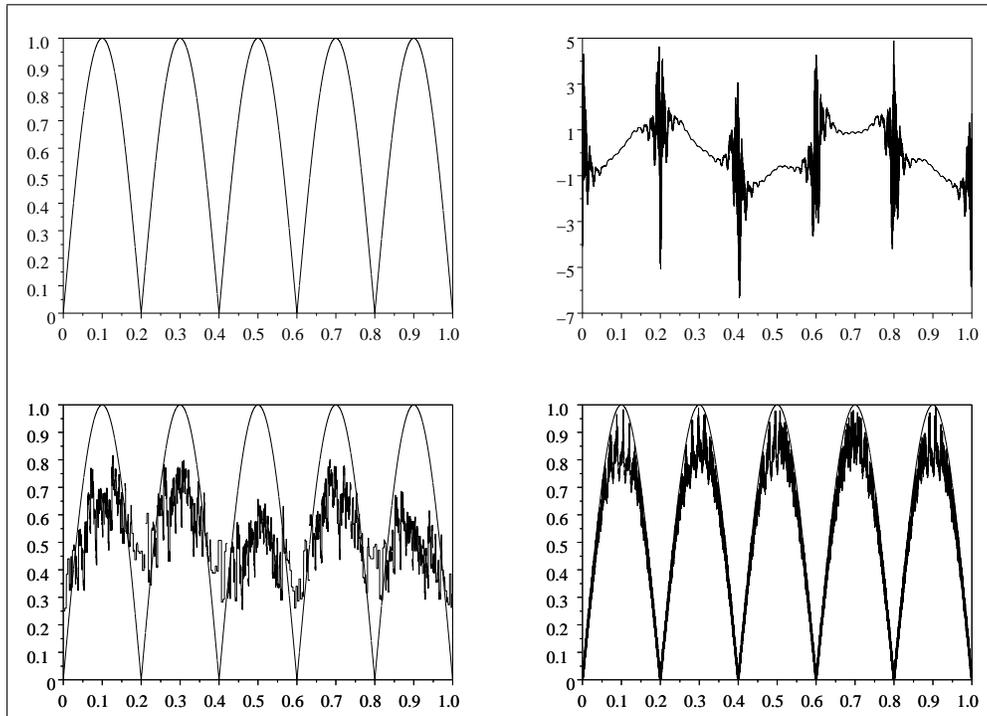


Figure 1: Our estimation of the Hölder exponent, on the lower right, compared with Daudi's estimation on the lower left. The plot in the upper left is the plot of the seed function, and the plot in the upper right is the resulting Generalized Weierstrass function.

3 Fractality of Operating System Resources

Current operating systems have a great number of parameters that can be measured. Not all of these parameters lend themselves to fractal analysis, however. The Hölder exponent calculation that we have developed is only suited to *time-series* of non-additive data, for instance. This means that each data point must be a measurement of an internal state of the operating system at an instant in time. This is as opposed to *measures*, which are a function of an interval of time and which are additive.

It turns out that the majority of data collected by standard operating system data collection processes are measures. Examples are the number of page requests per second, the number of system calls per second, the number of interrupts per second, and so on. The most obvious and easily appraisable parameter of an operating system that is not a measure is the memory use. This is the parameter that we have focused on in our experiments.

Our first attempt at collecting and studying operating system data focused on “Naur,” a Unix server in the computer science and electrical engineering department at West Virginia University. Naur has 256 MB of RAM, dual 333 MHz processors, runs SunOS 5.5.1, and is a heavily used server. We collected data once per second from the machine using the *sar* (system activity reporter) utility. We focused exclusively on the following 6 parameters of time-series data:

1. **sml_mem** - the amount of memory the kernel memory allocator has reserved for small requests.
2. **sml_alloc** - the amount of memory allocated to satisfy small requests.
3. **lg_mem** - the amount of memory the kernel memory allocator has reserved for large requests.
4. **lg_alloc** - the amount of memory allocated to satisfy large requests.
5. **freemem** - average pages available to user processes.
6. **freeswap** - disk blocks available for page swapping.

We found that a couple of the parameters displayed marked fractal characteristics, a couple were less fractal, and the two parameters dealing with the amount of reserved memory (`sml_mem` and `lg_mem`) were quite stable and showed no indication of being fractal.

Figure 2 shows a plot of the the `freemem` parameter with a plot of its Hölder exponent beneath it. In this case, the free memory resource displayed little fractality for the first 4900 seconds or so, and the bulk of the Hölder exponents for that time period stayed between 0.6 and 1. This is followed by a burst of chaotic behavior, which is captured by the Hölder exponent when it falls to between 0.3 and 0.6. When the period of fractality is over, the Hölder exponent rises again.

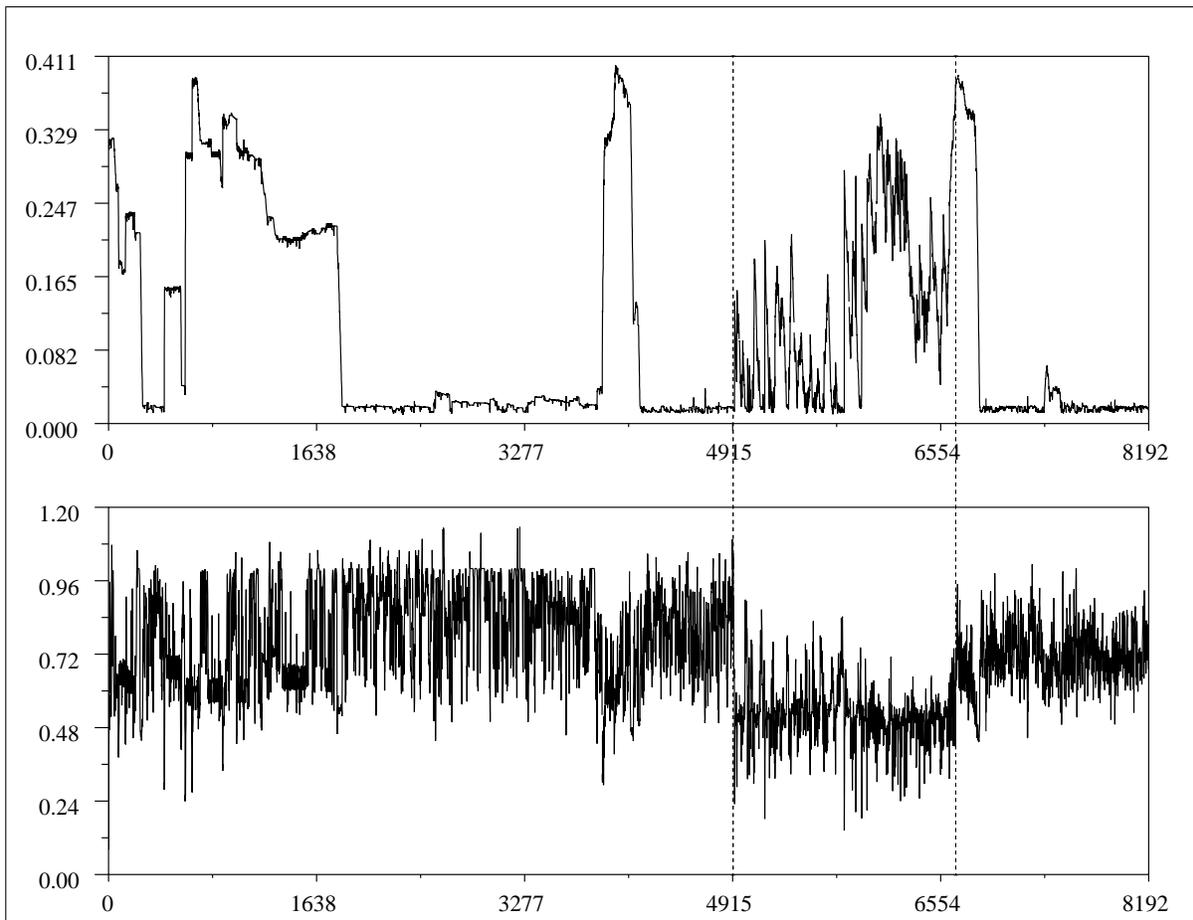


Figure 2: 8192 seconds of the `freemem` parameter from the Naur experiment along with its Hölder exponent.

A more intriguing interplay of data and Hölder exponent is seen in figure ???. In this case, a sharp drop in the `lg_alloc` parameter is immediately preceded by an increased level of fractality. If this increase in fractality is in fact a predictor of the coming drop in the availability of the resource, then the shift in the Hölder exponent could be used to give an alarm that a crash of some sort may be imminent. Measuring the `lg_alloc` parameter directly in this case would be difficult since upon the onset of fractality the resource does not display a very pronounced shift in magnitude — which would be easy to detect with an automated process of some sort. The Hölder exponent displays a marked change in magnitude, however, thereby enabling detection of the increased fluctuation in the original resource.

None of the data we collected from the Naur server actually preceded a system crash, however, so in order to test our hypothesis that a crash could be predicted through fractal analysis of a system’s memory use, we set up an experiment specifically designed to crash a computer. In addition, we needed to more accurately model the behavior of the system and come up with an algorithm that would issue an alarm when an operating system crash became likely. Our experiment and predictive analysis are described in the next two sections.

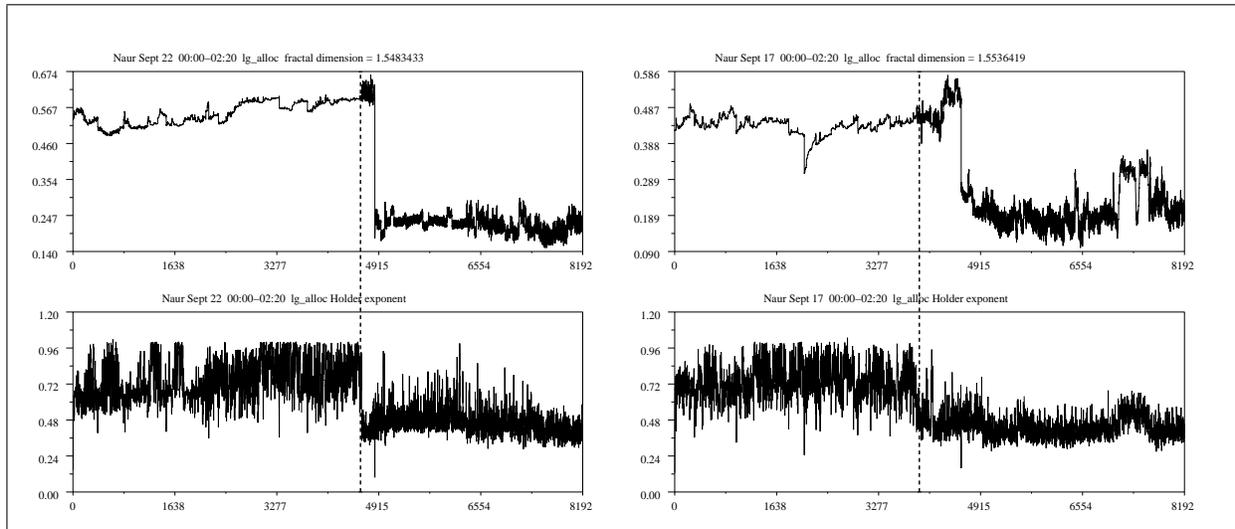


Figure 3: Availability of the lg_alloc resource falls soon after the onset of fractality.

4 Design of Experiment

4.1 Setup

Our “crash experiment” used the ‘System Stress for Windows NT 4.0 and Windows 2000’ software that is included in a subscription to the Microsoft Developer Network (MSDN). Two computers were set up next to each other and networked together with a crossover cable, thus forming a LAN of only the two machines. The System Stress software was installed on one computer (a Dell machine with 600 MHz processor, 64 MB of RAM, running Windows 2000) and configured to contact the other computer and attempt to crash it. The computer that was subject to the stress was a home-grown machine with a 750 MHz AMD processor, 256 MB of RAM, and running Windows 2000.

The System Stress utility gives the user a large number of stress programs that may be used in any particular trial. The selection of which additional stress program to add to the machine was made by a few lines of java code that output a random number. Every few minutes the administrator of the experiment would randomly generate a short list of additional programs and add them to the current stress test. Once a large enough number of programs were added to the stress test, the stress would reach a high enough level and the system would crash.

Prior to beginning the stress utility, a performance log would be set up to record data every second for the duration on the trial. The utility used for the collection of data was the Performance Logs and Alerts utility available in the Windows 2000 Operating System. During the experiment we collected data on 68 different parameters of the operating system, but winnowed this number down to 20 for our subsequent analysis. This number was eventually reduced to the three parameters that displayed the most fractal behavior and were also not strongly correlated with each other. The crash of the operating system naturally disrupted the collection of data, but the process was robust enough to collect data until the penultimate second before the crash.

The experiment was run 20 times, with a crash sometimes occurring with 5 minutes and sometimes occurring only after several hours.

4.2 The Data

The data we collected and analyzed confirmed our hypothesis that an increase in fractality precedes a crash. Figure 4 shows the plots of three parameters with the multidimensional Hölder exponent of the three data sets combined into a vector plotted below them.

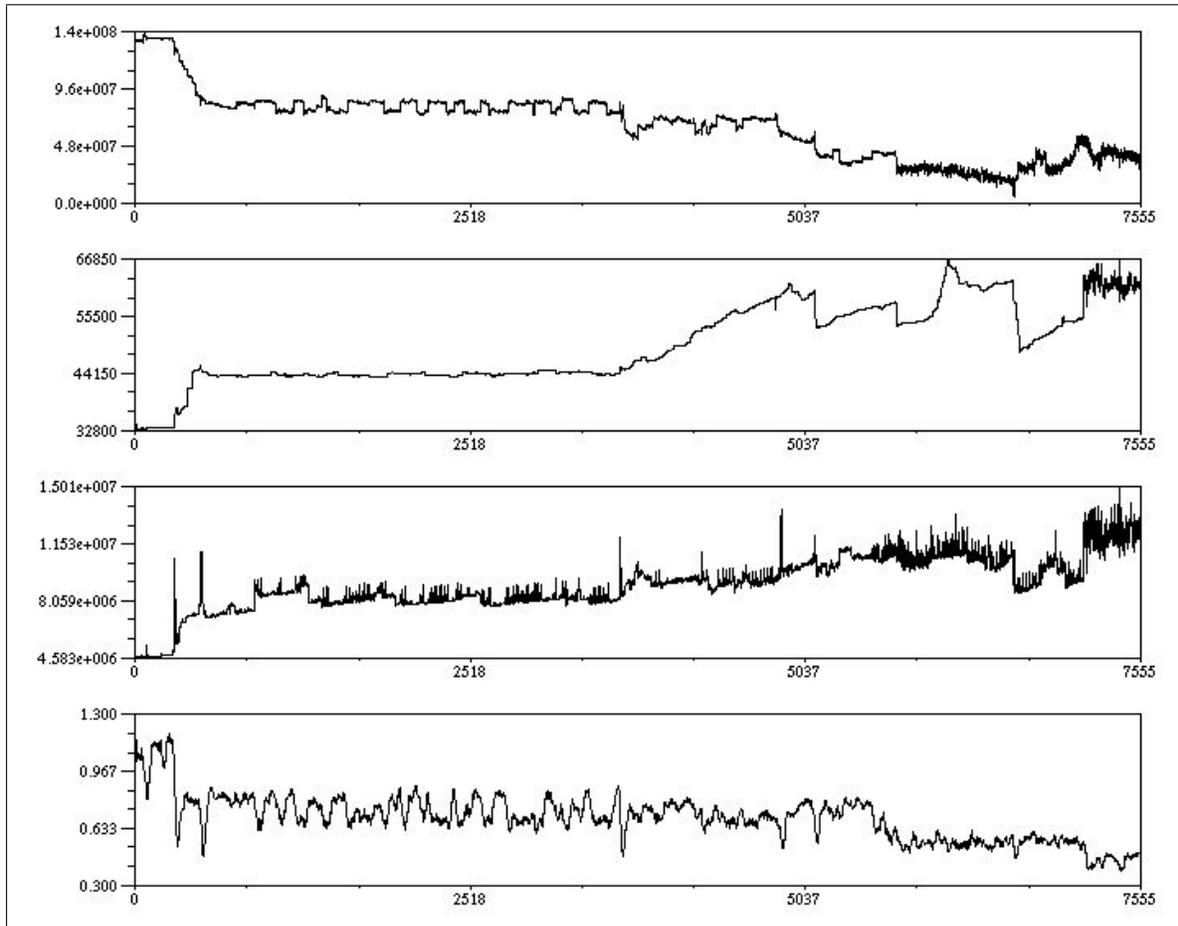


Figure 4: Available Bytes, Pool Paged Allocs, System Cache Resident Bytes, and their multidimensional Hölder exponent (with a 30-second moving average applied to the Hölder exponent).

5 Fractal Aging: Analyzing Hölder Exponent Plots

Even a cursory glance at the Hölder exponent plots depicting the dynamics of the operating system resources shows that, typically, a gradual increase in fractality (i.e. decrease of the Hölder exponent value) of the resource dynamics is observed as the stress on the system increases before the system crashes (see figures 5 and 6).

It is our belief, therefore, that *the process of resource exhaustion (“aging”) is represented by the decrease of the value of Hölder exponent of the systems resources considered as functions of time.* However, to be of any practical value this intuitive observation has to be converted into a precise quantitative indicator. This kind of problem is always difficult and requires a long and thorough investigation, both experimental and theoretical. In this report we present only the first step in this direction, which suggests the possibility of creating a useable ‘early warning’ system for operating system based on the multi-fractal analysis of its resources.

We started with selecting a small group of resources that we want to base our analysis on. The selection criteria have been:

1. The resource should represent “continuous” quantity, and it should not represent “per unit of time” kind of measurements. This criterion disqualifies such resources `system_driver_total_bytes`, because of their discrete, discontinuous nature, as well as `system_calls_per_sec` and such, since they represent quantities measured per unit of time (second). Remark: It is worth noting that the per-unit-of-time quantities may be of considerable interest for the multi-fractal analysis. However, they should be treated as multi-fractal measures, as opposed to multi-fractal signals. Therefore the computational methods involved in it would be rather different. We will

address the multi-fractal analysis of this type of resources in our future studies.

2. The resources whose plots exhibit flat or smooth behavior or consist of several intervals of such behavior. In our experiments the resources `system_code_total_bytes` and `system_driver_resident_bytes` demonstrated such behavior and, therefore, were excluded from the analysis.
3. The group of resources selected for analysis should not have high mutual correlations.

Proceeding in accordance with the above guidelines, we have selected for our analysis the following three system parameters:

1. `available_bytes`
2. `pool_paged_allocs`
3. `system_cache_resident_bytes`

(note that the choice is not uniquely determined by the guidelines). All these parameters represent various kinds of memory resources, they demonstrate fractal behavior and have rather low mutual correlations. The following table gives the correlations for the three parameters we have chosen:

	<code>avail_bytes</code>	<code>pool_page_allocs</code>	<code>sys_cache_res_bytes</code>
<code>avail_bytes</code>	1.0	-0.66	-0.85
<code>pool_page_allocs</code>	-0.66	1.0	0.5
<code>sys_cache_res_bytes</code>	-0.85	0.5	1.0

We wanted to incorporate into our analysis the fractal behavior of all three of these parameters. To this end, we considered the three parameters as a 3-dimensional memory resource vector. We sample this vector as it changes in time resulting in a three-dimensional time series, to which we apply the multi-fractal analysis. By computing (online) its Hölder exponent at every point of observation. The formula for the Hölder exponent is slightly modified in the case of multidimensional functions:

$$\mathcal{H}_f(t) = \liminf_{h \rightarrow 0} \frac{\log(\|f(t+h) - f(t)\|)}{\log(|h|)}.$$

The algorithm we used to estimate the Hölder exponent of the multidimensional time series is the same as described in Section 2.2, only instead of the difference between two points, we use the Euclidean distance.

Figures 5 and 6 show the plots of the Hölder exponent (figure 5) and the moving average of the Hölder exponent (figure 6) of the *3-dimensional memory resource vector* in 8 experiments ending with the system crashing. Since the Hölder exponent series obtained are very noisy (this is typical for experimentally estimated Hölder exponent values), we use simple 30 second moving average to somewhat smooth out the plots.

Based on the visual inspection of the plots, we observe that the plots have *long intervals during which the Hölder exponent fluctuates around a certain level*. This quasi-constant behavior is sometimes (infrequently) interrupted by moments of abrupt significant changes (usually drops) in the average value of Hölder exponent. We call a sudden sustained drop in the average value of Hölder exponent a *fractal breakdown*. It appears that most of the plots contain exactly two such breakdowns.

Conjecture. The *second fractal breakdown* observed since the system starts working signals a dangerous level of resource exhaustion, but leaves enough time to shut the system down before the crash occurs.

In validating the above conjecture and implementing the emergency shutdown strategy based on it we face the following problems:

1. *Detecting fractal breakdown.* We view the problem as that of detecting a change in the mean value of a noisy time series (see e.g. [11]). Here the time series in question is the one formed by values of the Hölder exponent of the memory resource vector computed during the experiment. We model the behavior of the Hölder exponent as a piece-wise constant function of time with a white Gaussian noise added to it. Though several good algorithms exist, the most reliable ones presuppose the a priori knowledge of the post-change. We are making good progress in this area.

2. *Determining the optimal shutdown time after the second breakdown has been detected.* In order to infer the optimal strategy for choosing shut-down time to, on one hand, let the system run as long as possible, while, on the other hand, not allowing it to crash before the shutdown, we plan to study the statistics of the time intervals between the second breakdown and the crash. We will continue working on this problem.

References

- [1] Michael Barnsley. *Fractals Everywhere*. Academic Press, 1988.
- [2] Khalid Daoudi and Jacques Lévy Véhel. Speech signal modeling based on local regularity analysis. *IASTED IEEE International Conference on Signal and Image Processing*, 1995.
- [3] Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. John Wiley and Sons, 1990.
- [4] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.
- [5] E. Tosan J. Thollot, C.E. Zair and D. Vandorpe. Modeling fractal shapes using generalizations of IFS techniques. In *Fractals in Engineering*, pages 65–80. Springer, 1997.
- [6] K. Trivedi K. Vaidyanathan. A measurement-based model for estimation of resource exhaustion in operational software systems. *Proceedings of the 19th International Symposium on Software Reliability Engineering*, 1998.
- [7] Yves Meyer Khalid Daoudi, Jacques Lévy Véhel. Construction of continuous functions with prescribed local regularity. *Journal of Constructive Approximation*, 14(3):349–385, 1998.
- [8] Benoit Mandelbrot Laurent Calvert, Adlai Fisher. Large deviations and the distribution of price changes. Technical Report 1165, Cowles Foundation, Sep 1997.
- [9] Benoit Mandelbrot. *Fractals: Form, Chance, and Dimension*. W.H. Freeman and Company, 1977.
- [10] Petros Maragos. Modulation and fractal models for speech analysis and recognition. *Proceedings of COST-249 Meeting*, Feb 1998.
- [11] Igor V. Nikiforov Michele Basseville. *Detection of Abrupt Changes : Theory and Application*. Prentice Hall, April 1993.
- [12] R. Morin A.L. Goldberger L.A. Lipsit N. Iyengar, C.K. Peng. Age-related alterations in the fractal scaling of cardiac interbeat interval dynamics. *American Journal of Physiology*, 40(4):1078–1084, 1996.
- [13] A. Goldberger S. Havlin M. Rosenblum Z. Struzik H. Stanley P. Ivanov, L. Amaral. Multifractality in human heartbeat dynamics. *Nature*, 399:461–465, Jun 1999.
- [14] David Parnas. Software aging. *Proceedings of the 16th Intl. Conference on Software Engineering*, pages 279–287, 1994.
- [15] Edgar E. Peters. *Fractal Market Analysis*. John Wiley and Sons, 1994.
- [16] A. Van Moorsel S. Garg. Towards performability modeling of software rejuvenation.
- [17] K. Vaidyanathan K. Trivedi S. Garg, A. Van Moorsel. A methodology for detection and estimation of software aging. *Proceedings of the 9th International Symposium on Software Reliability Engineering*, pages 282–292, 1998. Paderborn, Germany.
- [18] K. Trivedi T. Dohi, K. Goševa-Popstojanova. Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule. *Proceedings of the 2000 Pacific Rim International Symposium on Dependable Computing*, pages 77–84, 2000.
- [19] P. Heidelberger S. W. Hunter K. S. Trivedi K. Vaidyanathan W.P. Zeggert V. Castelli, R.E. Harper. Proactive management of software aging. *IBM J. Res. and Dev.*, 45(2):311–332, Mar 2001.

- [20] Jacques Lévy Véhel and Khalid Daoudi. Generalized IFS for signal processing. *IEEE DSP Workshop*, Sep 1996.
- [21] Walter Willinger Daniel Wilson Will Leland, Murad Taqqu. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, 2:1–15, 1994.
- [22] N. Kolettis N. D. Fulton Y. Huang, C. Kintala. Software rejuvenation: Analysis, module and applications. *Proceedings of the 25th Intl. Symposium on Fault-Tolerant Computing*, 1995.

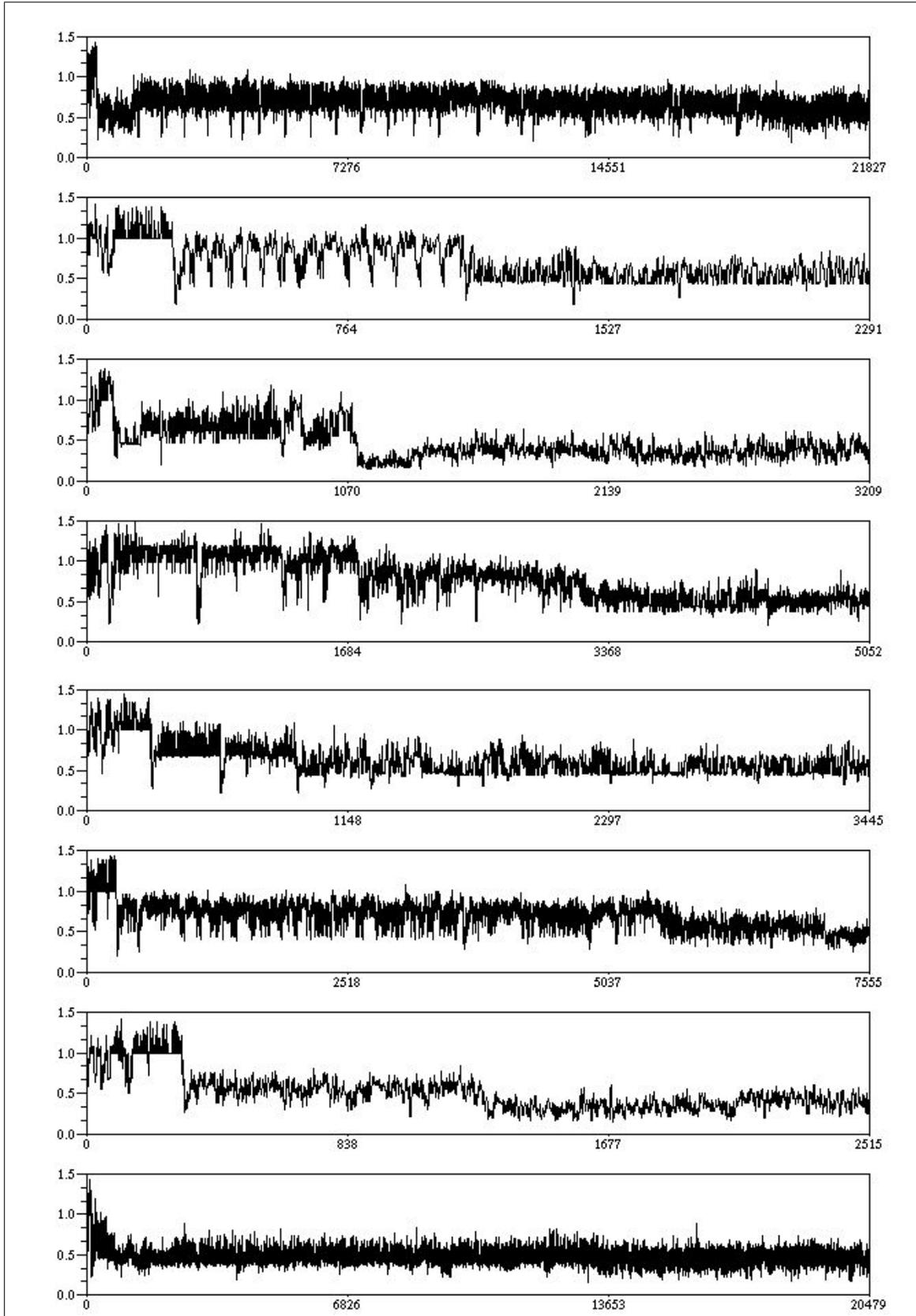


Figure 5: Multidimensional Hölder exponent for eight different sets of crash data.

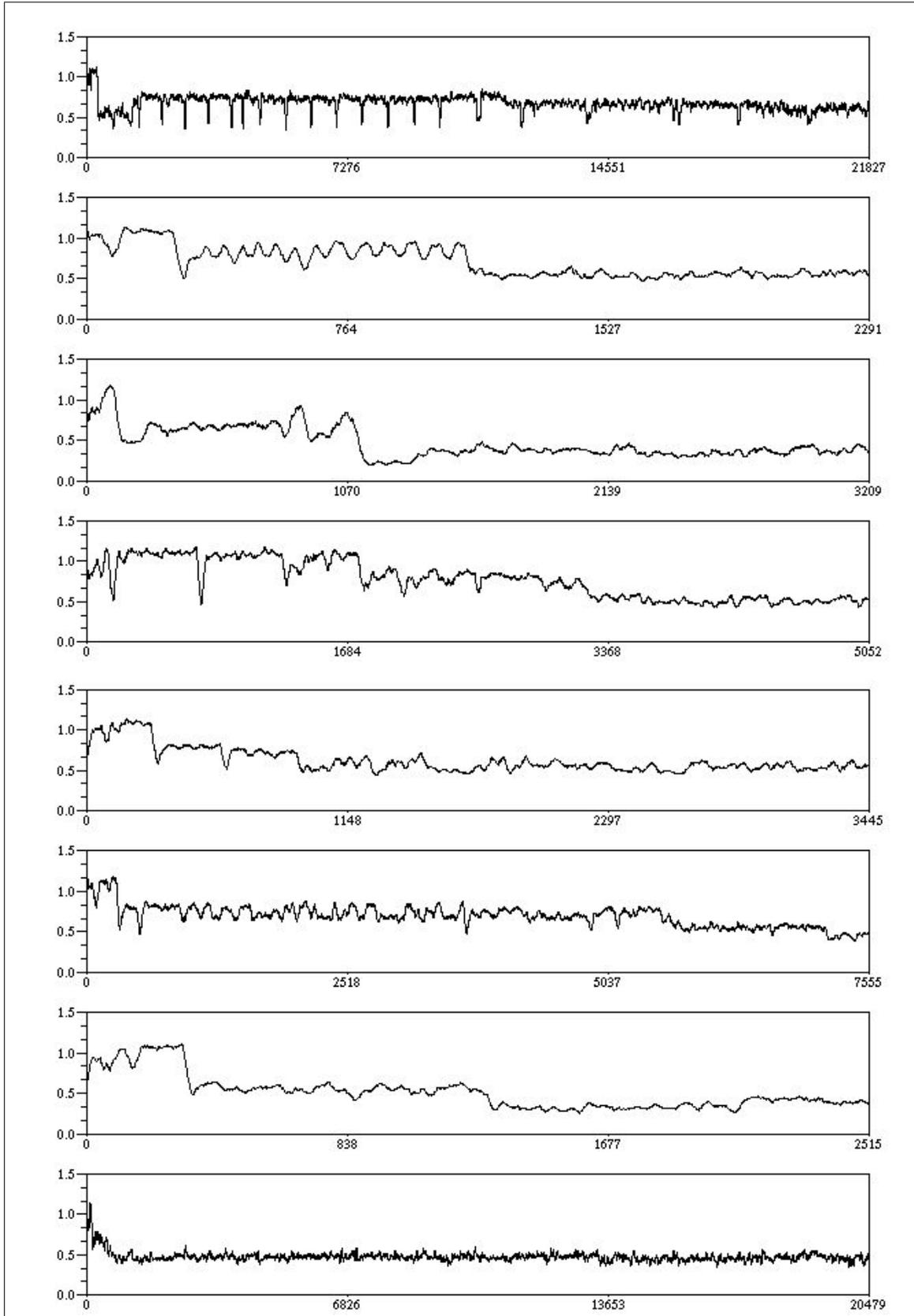


Figure 6: Moving average of the multidimensional Hölder exponent for eight different sets of crash data.