Spring 2012

A failure index for high performance computing applications

Clayton F. Chandler
Louisiana Tech University

A FAILURE INDEX FOR HIGH PERFORMANCE COMPUTING APPLICATIONS

by

Clayton F. Chandler, B.S., M.S.

A Dissertation Presented in Partial Fulfillment

Of the Requirements for the Degree

Doctor of Philosophy

COLLEGE OF ENGINEERING AND SCIENCE

LOUISIANA TECH UNIVERSITY

March 2012

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

November 15, 2011

Date

We hereby recommend that the dissertation prepared under our supervision by Clayton F Chandler

entitled

A FAILURE INDEX FOR HIGH PERFORMANCE COMPUTING APPLICATIONS

be accepted in partial fulfillment of the requirements for the Degree of

Doctor of Philosophy

[illegible]

Supervisor of Dissertation Research

[illegible]

Head of Department

Computational Analysis and Modeling

Department

Recommendation concurred in:

[illegible]

[illegible]

[illegible]

[illegible]

Advisory Committee

Approved:

[illegible]

Director of Graduate Studies

[illegible]

Dean of the College

Approved:

[illegible]

Dean of the Graduate School

GS Form 13a
(6/07)

ABSTRACT

This dissertation introduces a new metric in the area of High Performance Computing (HPC) application reliability and performance modeling. Derived via the time-dependent implementation of an existing inequality measure, the Failure index (FI) generates a coefficient representing the level of volatility for the failures incurred by an application running on a given HPC system in a given time interval. This coefficient presents a normalized cross-system representation of the failure volatility of applications running on failure-rich HPC platforms. Further, the origin and ramifications of application failures are investigated, from which certain mathematical conclusions yield greater insight into the behavior of these applications in failure-rich system environments.

This work also includes background information on the problems facing HPC applications at the highest scale, the lack of standardized application-specific metrics within this arena, and a means of generating such metrics in a low latency manner. A case study containing detailed analysis showcasing the benefits of the FI is also included.

APPROVAL FOR SCHOLARLY DISSEMINATION

The author grants to the Prescott Memorial Library of Louisiana Tech University the right to reproduce, by appropriate methods, upon request, any or all portions of this Dissertation It is understood that "proper request" consists of the agreement, on the part of the requesting party, that said reproduction is for his personal use and that subsequent reproduction will not occur without written approval of the author of this Dissertation Further, any portions of the Dissertation used in books, papers, and other works must be appropriately referenced to this Dissertation

Finally, the author of this Dissertation reserves the right to publish freely, in the literature, at any time, any or all portions of this Dissertation

Author [illegible]

Date 11/15/11

GS Form 14
(5/03)

DEDICATION

To my wife for giving me the ability to see this through to the end, and to my parents for their never-ending support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would foremost like to acknowledge Dr. Chokchai Leangsuksun for his continued guidance in my research and professional development. He has been a great friend as well as a great advisor. I would also like to thank Dr. Nathan DeBardeleben of Los Alamos National Laboratory for his guidance over the years, as well as Dr. Mihaela Paun for lending me her knowledge and assistance whenever asked. I would also like to thank Dr. Bernd Schroeder for his great assistance in completing the final revisions of the dissertation. A special thank you is also warranted for the other members of my advisory committee: Dr. Weizhong Dai and Dr. Dexter Cahoy. Lastly, I would like to thank the faculty and staff members in the Computer Science, Mathematics, and Graduate Studies offices for their patience in handling my never-ending list of questions and predicaments.

CHAPTER 1

CONTRIBUTIONS AND AN INTRODUCTION TO HPC RESILIENCE

1.1 Contributions

The novel contribution of this work is the introduction of a normalized metric for measuring the time-dependent failure volatility of a High Performance Computing system. This metric – the Failure Index (FI) – is based on existing inequality measures such as the Gini [86], Atkinson [91] and Theil [92] indices and is formulated, specifically, as a time-dependent implementation of the Atkinson index.

The FI generates coefficients based on the downtime resulting from all failures incurred by a given system in a given time interval. These coefficients represent the differences in downtime resulting from individual system failures in the time interval using a 0-to-1 system, with FI coefficients closer to 1 representing higher levels of volatility amongst failures in the given time interval than FI coefficients closer to 0. This volatility is not captured by existing metrics such as Mean Time between Failure (MTBF) and system uptime percentage. Figure 1.1 illustrates the information captured by the FI relative to these measures.

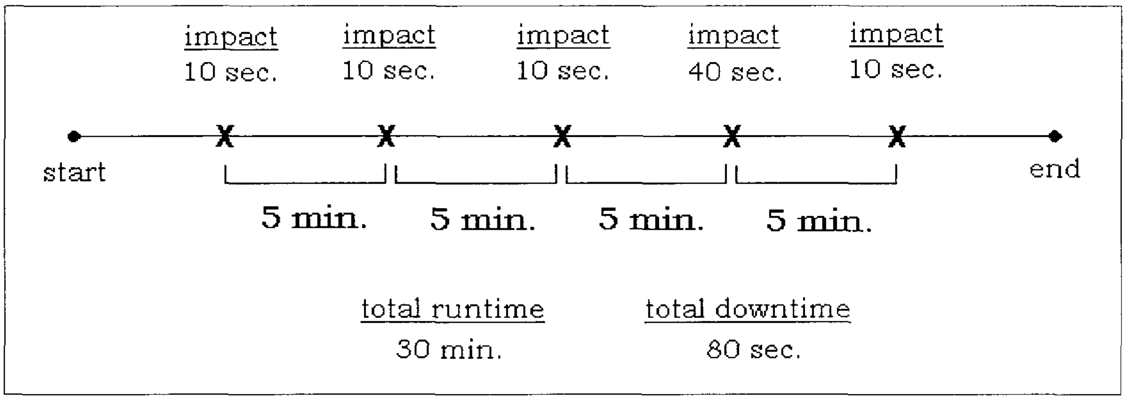


Figure 1.1 Information Captured by the Failure Index

The HPC system represented in Figure 1.1 runs for a total of 30 minutes. Its MTBF is 5 minutes. Likewise, the total uptime percentage of this system is 1720 / 1800 seconds = 0.955 = 95.5%, or "one nine". However, neither metric captures that the system exhibits four failures that cause a downtime of 10 seconds each and one failure resulting in a downtime of 40 seconds. A Failure Index coefficient would represent this inequality. Specifically, given parameter $\varepsilon = 0.20$ the FI coefficient for this example system is 0.2098798, representing a modest amount of downtime volatility. If each of the five failures resulted in a downtime of 10 seconds, the FI coefficient would be 0, meaning that every failure's impact on the system was equal. Likewise, in the case that four failures resulted in no downtime and a single failure caused all 80 seconds of downtime, the resulting FI coefficient would be 1. A mathematical introduction to the Failure Index is given in Chapter 4.

The scale and location-invariant nature of FI coefficients allows us to compare the failure volatility of multiple systems regardless of size. This will be demonstrated in Chapter 5 when constructing FI coefficients for 23 different machines housed at Los Alamos National Laboratory, all of which contain different numbers of compute nodes.

The FI coefficient is a unitless value. As will be discussed in Chapter 2, many existing system reliability metrics are not standardized across the entire HPC arena, which allows for manipulation and corruption of system performance metrics. However, the unitless nature of the FI prevents this, as, given an assumption of globally defined failure events, FI coefficients represent similar levels of volatility across multiple machines. This allows for a true comparison of system failure volatility using a normalized scheme and should find a home as a quality of service measure for HPC system performance and resilience.

1.2 An Overview of the Dissertation

Chapter 1 first summarizes the novel contributions of this work. It then provides an overview of the dissertation and an introduction to the resilience issues facing today's large scale HPC machines and applications. It covers a general introduction to the field of resilience and its importance in today's HPC arena, a look specifically at the issues facing applications computing on the world's fastest machines, and a review of existing approaches for combating such problems. Further, the current lack of application-specific metrics in the HPC research and development community is examined. This includes discussion on the lack of information stemming from large-scale systems as well as the need for standardization in HPC measures and definitions across the landscape, including government, academic and industrial entities. This chapter also represents the motivation for the work outlined in subsequent chapters in this dissertation.

Chapter 2 introduces a module for generating relevant resilience-specific information from a given HPC application. This application, Gilgamesh [83], was developed by the author as a means for combating the application information gap discussed in Chapter 1. An existing HPC dataset is then examined using traditional statistics.

Chapter 3 begins the discussion of inequality indexing by demonstrating that such approaches are both possible and practical when applied in an HPC context. The Gini, Atkinson, and Theil indices are mathematically introduced and Gini and Atkinson coefficients are generated using an existing HPC performance dataset containing reliability information from multiple machines. Results are then discussed and conclusions pertinent to the use of such indices in examining HPC application behavior are drawn.

Chapter 4 introduces the Failure Index (FI) from a theoretical standpoint, including its definition and mathematical construction. This chapter contains the bulk of the novel work undertaken in the research and as such it should be read carefully.

Lastly, Chapter 5 demonstrates the utility of the FI by conducting a case study consisting of multiple models and analysis. In this analysis FI coefficients are generated in conjunction with the same HPC performance dataset utilized in Chapter 3. The scale- and location-invariant nature of the metric allows us to obtain novel results related to all machines contained in the dataset.

Chapter 6 contains conclusions drawn from the entirety of the work as well as proposed future studies in resilience, HPC applications and metric generation.

1.3 The Current State of HPC Resilience

High-end parallel computing is relying increasingly on large clusters with thousands or even tens of thousands of processors. Further, many of today's large scale systems such as the Road Runner machine at Los Alamos National Laboratory include heterogeneous architectures, increasing the complexity of such machines even further. With so many nodes and with such complicated architectures, system and application failures are becoming increasingly commonplace on these machines.

As an example, one of today's fastest systems – the BlueGene/L (BG/L) machine housed at Lawrence Livermore National Laboratory (LLNL), which contains 65,536 compute nodes and 131,072 cores – was failing once nearly every 48 hours during its initial deployment in 2005 [1]. Each time one of those nodes failed, a 1024-processor midplane had to be temporarily shut down in order to replace a dual-processor compute card. This is clearly unacceptable from a quality of service standpoint, especially considering that each of these machines costs millions (or in the case of the afore-mentioned Road Runner, hundreds of millions) of dollars. Table 1.1 illustrates that the existing reliability of larger HPC clusters is currently constrained by a mean time between failures (MTBF) in the range of 1.2 - 351 hours depending on the age of the machine.

Table 1.1 Publicly Available HPC System Reliability Statistics

Installed	System	Processors	MTBF	Measured	Source
2000	ASCI White	8,192	40.0 hours	2002	[2]
2001	PSC Lemieux	3,016	9.7 hours	2004	[3]
2002	NERSC Seaborg	6,656	351.0 hours	2007	[4]
2002	ASCI Q	8,192	6.5 hours	2002	[5]
2003	Google	15,000	1.2 hours	2004	[6]
2006	Blue/Gene L	131,072	47.8 hours	2006	[7]

Existing work [2] similarly suggests a system mean-time to failure (SMTTF) constraint of 5-6 hours, or 4 failures per day, for current HPC systems. The most common causes of failure were processor, memory and storage errors. Extrapolating from current system performance levels, a study by Los Alamos National Laboratory (LANL) estimates a MTBF of 1.25 hours for a petaflop machine [3]. Commercial installations such as Google experience an interpolated MTBF of just over one hour for an equivalent number of nodes (see Table 1.1) as the BG/L system. However, Google's fault-tolerant middleware hides such failures, leaving user services completely intact [4].

It must be noted that parallel applications still maintain reduced completion times in comparison to their single threaded counterparts. However, when substantially increasing the number of nodes located within an HPC system and assuming theoretical linear scalability for applications, application completion times do not necessarily decrease proportionally. In fact, based on results obtained in [6], the opposite is true – while application completion time initially decreases as more nodes share the work, at some critical point this value begins to rise substantially due to the increased likelihood of reliability issues stemming from addition of computational units. Figure 1.2 illustrates this.

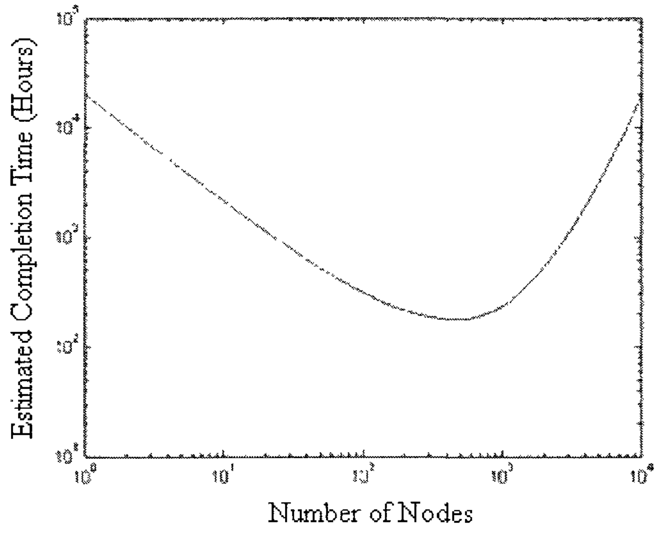


Figure 1.2 Application Completion Time vs. Number of Nodes

Recent accomplishments in providing an insight into HPC resilience showed that some HPC system failures can be anticipated by detecting deteriorating system health through hardware monitoring [5]. Further recent work focused on capturing the availability of large-scale systems using combinatorial and Markov models and subsequently comparing these results with statistics from large-scale U.S. Department of Energy (DOE) installations [6, 7].

However, the health data collection and processing algorithms outlined in these studies do not efficiently perform on large-scale HPC systems. Furthermore, fair and meaningful reliability comparisons between systems are impossible due to different hardware and software architectures, failure modes, and system health and failure reporting mechanisms. Others have suggested [6] that reliability and availability metrics standard for HPC were needed, as well as scalable, non-intrusive system health data collection and processing algorithms. Both of these topics are discussed at length in the body of this dissertation.

To prevent lost computation time due to failure, checkpoint/restart (C/R) algorithms have become a requirement for most long-running HPC jobs. Current C/R mechanisms commonly allow checkpoints to be written to a global file system. This allows an entire MPI (Message Passing Interface) job to be restarted from its last checkpoint in the event of failure. One example of such a solution is LAM (Local Area Multicomputer)/MPI's [8] C/R support through Berkeley Labs C/R (BLCR) [9].

C/R, like many other existing techniques, is a reactive scheme. Such techniques allow a computation to recover once a failure has occurred. The Los Alamos study leveraged these techniques to estimate the checkpoint overhead requirements of a petaflop machine. It found that based on current techniques, a 100 hour, failure-free job will be prolonged by an additional 151 hours in petaflop systems.

Some recent studies suggest collecting data from existing machines and using that information in a reactive manner to derive a checkpoint interval that trades off checkpoint cost against restart cost [10]. Instead of a reactive scheme for fault tolerance, others are promoting a proactive approach that migrates processes away from unhealthy nodes and onto healthy ones. Such an algorithm has the advantage that checkpoint frequencies can be reduced as sudden, unexpected faults should become less common [11].

Further still, failure prediction has become a relevant and highly researched topic due to the substantial growth in size and scope of HPC deployments and the corresponding increase in system failure rate [12]. Predicting and proactively treating failures via the use of appropriate resilience mechanisms such as the ones previously mentioned substantially reduces the amount of wasteful re-computation time required in the triage period following a failure. However, most contemporary failure prediction techniques involve MTTF approximation and post-event analysis of system logs [13, 14]. Others suggest a need for analysis based on individual compute node reliability in conjunction with system health [15].

The feasibility of health monitoring at various levels has recently been demonstrated for temperature-aware monitoring via the utilization of ACPI [16] and more generically by critical-event prediction [17]. Such approaches in systems with thousands of processors such as BG/L range from application runtime-level techniques to the level of operating system (OS) schedulers [18].

While process-level C/R has received much attention in the HPC arena, recent results on OS-level C/R show that OS virtualization is a viable alternative. More specifically, experiments were conducted with process-level BLCR [19] and Xen [20] to assess the overhead of saving and restoring the image of an MPI application on a faulty node. For BLCR this comprises the process of an MPI task, while for Xen the entire guest OS is saved. Tests for NAS PB programs under Class C inputs show an overhead of 8-10 seconds per one-minute run of BLCR and 15-23 seconds per one-minute run of Xen on the same experimental platform [21]. Variations are mostly due to the memory requirements of specific benchmarks. These memory requirements also dominate those of the underlying OS, which explains why Xen remains competitive in these experiments.

1.4 Preliminary Studies and Related Work

Detailed analysis was performed on system event records generated by the four Advanced Scientific Computing (ASC) machines (White, Frost, Ice, and Snow) housed at LLNL [2]. Using Markov modeling, time to failure (TTF) data for the 512-nodes ASC White machine covering a four-year operational period [22] was generated. The average per-node SMTTF was found to be approximately 3293 hours, or around four months.

This failure information was then taken and compared to a variety of statistical distributions, using the Kolmogorov-Smirnov (K-S) test to obtain the most appropriate fit. The K-S test yields a best fit between the empirical data and four theoretical distributions – namely the exponential, Weibull, Gamma and Log-normal distributions. K-S testing was performed on 512 different nodes and time period permutations assimilated from the ASC White logs. The data suggested that the failure rate varies over time. It was found that, as such, Weibull, Gamma, and Log-normal distributions, all of which suggest variable response in conjunction with time change, provide a more accurate basis for reliability modeling than an exponential distribution, which implies constant failure behavior, as shown in Table 1.2.

Table 1.2 Best Fit Cases, ASC White

Distribution	Number of Best Fit Cases
Exponential	185
Weibull	334
Gamma	328
Log-normal	318

In a separate investigation on raw LLNL Blue Gene/L system logs, the XCR team at Louisiana Tech University [23] found that both hardware and software failures could not be easily obtained from the log file. The team also introduced a novel approach for deriving software interrupts from raw system failures using the system time to repair (TTR). It was suggested that this data be used for higher-level knowledge discovery such as reliability analysis or failure prediction.

Additionally, the XCR team presented optimistic and pessimistic approaches for estimating a system's failure behavior, representing best-case and worst-case behaviors respectively. Results showed that the failure behavior bounds – both best-case and worst-case – vary greatly due to a multitude of undetermined events that are flagged as fatal system failures. Problems arose in attempting to determine if such events were truly fatal behaviors or false positives. This problem confirmed the importance of failure identification mechanisms such as system monitoring and logging.

We have also developed a reliability-aware resource allocation model for parallel programs [24] and an optimal checkpoint/restart model [25]. The reliability-aware resource allocation model aims to minimize performance loss due to failure. Results indicate that applying a reliability-aware resource allocation technique reduces the overall waste time of parallel jobs by as much as 30%. The improved checkpoint model optimizes wasted time (checkpoint overhead, recovery time, and re-computational time) by balancing both checkpoint overhead and re-computational time.

In addition, a fault tolerance framework [26] was developed that enables an HPC system to self-heal/self-clone in order to tolerate a system failure by using checkpoint/restart mechanisms. This framework was implemented on Linux-based HPC systems and integrated an optimal checkpoint placement model. This model was further extended to act as a feedback control loop – a fundamental part of the resilience framework.

Additionally, the author assisted in the formation of a Resilience Consortium consisting of top researchers in the field of HPC resilience. Stated goals include the standardization of terms, methods, and algorithms encountered in resilience research and the development of fault tolerant large-scale computing systems. This consortium is an open community of HPC leaders from industry, academia, and research institutions. Preliminary studies concluded that better data acquisition coupled with improved reliability and prediction models as well as continued enhancement to the feedback control loop will lead to an improved resilience model that increases application productivity.

As in this dissertation, most contemporary studies which conduct reliability analysis on large-scale computing systems assume that failures transpiring within various components or nodes are independent [27-29]. However, other studies exist suggesting some inter-constituent dependencies in failure origin, especially at the application level [30]. Dependencies reported by these studies occur mostly in system configuration and within the operation environment [31]. Such dependencies are not accounted for in this work.

To improve reliability model accuracy, a relaxation in the assumption of independent failure behavior was suggested, as the model should represent the failure probability of each node as well as of the cross-application runtime environment. Another important discovery derived from reliability analysis of HPC systems is that for a given parallel architecture and problem size, application completion time will not continuously decrease due to a constant introduction of new processors and higher core counts into the computing system. Such results rest at the heart of the resilience issues facing large-scale HPC, as it is that victim-of-scale phenomenon that is most often studied. Various aspects of scalability were examined in [32, 33].

In application distribution, Shatz et al. [34] modeled tasks and communication links as a directed graph, using it to intelligently allocate jobs within heterogeneous distributed systems. Other groups studied application allocation through checkpointing mechanisms [35]. There are also existing works which propose checkpoint scheduling optimization [36, 37]. Geist et al. [38] and Wong et al. [39] presented Markov availability models and obtained an optimal checkpoint placement that maximizes system availability. Ling et al. [40] presented optimal checkpoint scheduling models for an infinite horizon time by using a calculus of variations technique. They concluded that, theoretically, a fixed checkpoint interval is optimal if and only if application failure follows a Poisson distribution. Ozaki et al. [41] extended the calculus of variations concept introduced in [42] to apply to a system with a finite horizon time and incomplete failure information. However, both works define re-computation time as a linear function demonstrating model applicability – a metric that in practice should depend upon failure behavior.

Plank et al. [43] discussed the importance of processor count as a performance attribute in checkpointing applications. The now-popular concept of incremental checkpointing was introduced as a means of reducing checkpoint/restart mechanism overhead by saving the state of only those application pages that have been changed [44-46]. Palaniswamy et al. [47] observed that, given a minimal number of increments, the incremental checkpointing approach has the potential to surpass traditional checkpointing algorithms in efficiently utilizing computational resources. The challenge in achieving minimum overhead while using incremental checkpointing schemes is to find a maximum number of incremental checkpoints while maintaining lower system costs than traditional algorithms.

In addition to the utilization of checkpointing mechanisms, some researchers also studied the costs and benefits associated with job migration. Halchor-Balter and Downy [48] used process aging to present a migration cost model. They concluded that process migration most benefits those processes which exhibit long execution times. Mello and Senger [49] proposed a model which evaluates process load and lifetime in order to analyze the effects of job migration within an environment composed of heterogeneous computers. That study assumes that the load balancing frequency of each compute node indicates the occupation level of that node. The proposed model then makes a decision regarding which application processes should be migrated.

Other groups proposed algorithms which combine checkpoint/restart mechanisms and process migration. For example, Cao et al. [50] presented a process migration strategy based on the coordinated checkpointing of message passing interface (MPI) applications.

This technique modifies process location-related information in the checkpoint file and reintroduces the application with respect to the modified location. Sun et al. [51] proposed the Fault-aware ENabled Computing Environment (FENCE) system for high end computing – a unified computing framework with both reactive and proactive mechanisms. As in this dissertation, most existing optimal scheduling frameworks [52-54] assume that system failure follows a Poisson distribution. However, many also assume a constant failure rate, which in most cases is not a good representation of actual system failure characteristics [55, 56].

In HPC failure prediction research, Xue et al. [57] surveyed five major prediction techniques – a statistic-based threshold method, time series analysis, rule-based classification, Bayesian network models and semi-Markov process models. The survey concluded that Bayesian network models and semi-Markov process models are the best approximations of system failure due to precision and recall. This conclusion led to the utilization of naïve Bayesian classification in the construction of the FI.

Semi-Markov processes were used in [58] to model the system reliability of computational grids, and then further used in forming the probability of failure for an individual system within in the grid. By filtering out periodic failures, the relative error of predicting system reliability in relation to empirical reliability is relatively small (less than 0.05). However, this reliability metric directly infers only the probability of failure, rather than actually predicting system failure.

Lead time ($|T_1|$) is defined as the interval between the time a failure prediction takes place and the time that predicted failure is expected to occur. The predicted failure interval is similarly denoted $|T_P|$. Salfner et al. presented a failure prediction method called Similar Event Prediction (SEP) [61], which exploited a semi-Markov model by using groups of events to form states and predicting a failure if the probability of failure obtained from the model exceeded a predefined threshold. The model predicted failures with a bounds of $|T_1| = |T_P| = 1$ minute and yielded results illustrating high precision (0.8) and recall (0.923). Precision and recall are measures of accuracy defined in relation to true or false positive or negative resulting values, with values closer to 1 indicating more accurate results than values closer to 0. Precision (formula 1.1) and recall (1.2) are mathematically defined below:

$$Precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{1.1}$$

$$Recall = \frac{true\ positive}{true\ positive + false\ negative}\,. \tag{1.2}$$

This analysis was performed on legacy telecommunication deployments varying greatly in design and purpose from high-end HPC systems. Further, the assumed one minute lead time may be too short to allow the system to take adequate responsive action.

In addition to Markov models, association rule discovery [59] has also been applied to HPC failure prediction. The researchers form sets by grouping events that appear close to each other chronologically and apply an association rule discovery algorithm, predicting failures using a set of rules inferring either critical or fatal events. Sahoo et al. [60] applied this approach utilizing a variable time interval, from 100 to 800 seconds. They claimed that the resulting model provided up to 70% accuracy in predicting failures. Gujrati et al. [61] similarly applied this technique using a varying $|T_P|$ – from 5 minutes to 1 hour – and $|T_1| = 0$. However, because $|T_1| = 0$, an implementation of this model is unobtainable. Additionally, meta-learning applied in [62] can help boost recall from 0.22-0.55 up to more than 0.65.

Bayesian network models have not been explored to their full potential in HPC resilience study. This is due to the high computational costs of network topology construction and structural learning. As in this dissertation, Hamerly and Elkan [63] exploited a naïve Bayes scheme: a Bayesian methodology which assumes independence among variables – to predict disk drive failures, achieving a 0.33 true positive rate and a 0.0067 false positive rate. Causality of variables can also be studied with Bayesian analysis [64]. Sahoo [65] used Bayesian analysis to study causes and effects amongst system variables, such as CPU usage and occurrence of system events.

Analyzing system failure behavior is not a trivial matter, and as such, a variety of statistical methods must be exploited in the development of a reliability-aware runtime framework for the modeling of large-scale systems, specifically in analyzing the behavior of large-scale HPC applications. Bayesian analysis serves as a good alternative to conventional statistical analysis methods in that it allows researchers to use all available current and prior information in creating statistical models, as opposed to conventional methods which restrict analysis to current data. The Bayesian approach models parameters as random quantities and uses existing information to construct an antecedent distribution model for the values.

There is little to no existing work in the utilization of inequality indices in modeling HPC application behavior, let alone the development of a new index specifically targeted at creating such models.

1.5 A Lack of Standardization Amongst Metrics

When one examines existing work in the areas of performance and reliability analysis, behavioral modeling, quality of service estimation and failure prediction, it becomes immediately clear that there exists substantial need for standardization in both the explicit definitions and mathematical derivations utilized in such studies. Concepts such as an application's MTBF, checkpoint/restart latency or overhead, failure prediction costs or application process migration overhead are defined in radically different ways depending on the individuals or organizations performing the analysis in question.

One such example is the definition of failure itself. While on its surface a very simple idea requiring what should be an analogously simple definition, failure is instead defined in radically variant ways across the HPC resilience community. Entities focusing strictly on HPC systems, for example, define failure strictly as a system-centric construct – that is, power or network outages are labeled as failures, while the unexpected termination of one or more applications running on those systems are not labeled as such. Examples of such entities are telecommunication data centers and large storage facilities.

However, more application-focused individuals or organizations would certainly consider unexpected job termination a failure. Included in this group are pure scientists, government laboratories, or any entity utilizing its HPC assets for capability computing. Even at this level – which, it should be noted, is the point of view taken in this dissertation – there exists much debate on the specific details included in the definition of application failure. Certainly it would be considered a failure if one submits a job and it terminates before reaching its expected outcome. However, what if only one application process terminates unexpectedly, while the remainder successfully finish? What if the application enters an infinite loop? In this case the application certainly hasn't terminated, yet it will never finish executing.

There is no existing, widely adopted answer to any of these questions. The Resilience Consortium has begun the process of tackling issues such as standardization, but no industry-wide consensus has been reached and there remains much work to be done in defining even the simplest of terms. All entities performing high performance computing are aware of the issue and acknowledge the utility of arena-wide metric standardization, but all bring their own biases in regard to such terms and little progress has been made in this area.

This lack of standardization along with a strong need for application-specific reliability metrics spawned the work outlined in this dissertation. The latter issue is covered explicitly in the following chapter, while the entirety of this work is devoted to providing one solution for the former. Although one-hundred percent adoption of these concepts is an unrealistic goal, it is hoped that industry leaders and specifically the Resilience Consortium give this idea of a unified Failure Index for HPC application volatility a strong look and seriously consider pressing for the adoption of such a metric. Even ignoring its utility as a location- and scale-invariant measure of application volatility (those concepts will be discussed in depth in Chapters 3 through 5) the FI as proposed in this dissertation can be easily implemented on live HPC systems, making it a lightweight solution.

CHAPTER 2

HPC METRICS: GENERATION AND EXAMINATION

2.1 Generating Metrics for HPC Applications

The HPC community is presently encountering substantial proliferation in the number of computational units used in its computing platforms. Although this increase in size (and, subsequently, computing power) has both raised the bar for what high performance machines can do and brought mainstream attention to the field via its ascension beyond the petaflop barrier, it has also led to an increase in application downtime [66].

Traditionally, the HPC research community has attempted to alleviate such issues via the research and development of solutions geared toward increasing the reliability of these machines. These hardware-centric solutions carry a single objective: maximize the system uptime of a given HPC distribution. However, as extreme-scale HPC platforms enter the petaflop age via the dissemination of ten and hundred-thousand core architectures, application failures are encountered at rates that pragmatically prevent fully reliable systems [67]. Thus, there is much work to be done in providing HPC applications with the ability to run *through* failure. That is, solutions must be devised that endow these programs with the capability to be resilient to failures encountered by the systems upon which they execute.

We define a fully resilient HPC application as one that continues an acceptable level of execution in the event of any variant form of non-catastrophic failure encountered by its host system (catastrophic failures being wholly unavoidable events such as natural disasters and center-wide power outages). Resilience, then, is a metric denoting how close a given HPC application comes to realizing full resilience. This measure will, in time, become as important in measuring the 'worth' of a given HPC platform as contemporary benchmarks such as peak FLOPS, due to the increasing power and cost demands of computing at an extreme scale.

The question, then, becomes "how does one provision HPC applications with resilience-focused capabilities?" The answer begins with investigating how and why contemporary extreme-scale machines are encountering such a startling number of performance interruptions. This, however, is no simple task, due to the severe lack of existing public information originating from high-end HPC applications. Los Alamos National Laboratory collected and published data regarding the failure and usage of more than 20 of their supercomputing clusters [68] and this information has been analyzed by Schroeder et. al. from CMU in an attempt to study the root cause of the reported failures [69]. This data consists primarily of failure data and system administrator notes but does not include machine logs. This dataset will be examined in the following chapter.

As outlined in the previous chapter, work undergone by others within the HPC research community has concluded that the inherent lack of structure exhibited by supercomputer system log files prevents the ability to perform efficient analysis on the reliability of such machines. Thus, to further resilience-centric study, there exists the need for a novel method of extracting performance and reliability data directly from extreme-scale applications. This chapter outlines a solution that, via a combination of dynamic instrumentation and autonomous application correction, provides and handles this information in an efficient manner.

Resilience, as a research and development field, can be defined as the radically application-centric study of HPC reliability. Recapping from the previous chapter, those within this field began preliminary investigations into the Mean Time to Interrupt (MTTI) exhibited by contemporary high-end HPC applications. A LANL study [70] extrapolated current system performance and subsequently estimated a 1.25 hour application MTTI for a petaflop system – a very alarming and unacceptable value especially given the time required to take a full-system memory snapshot (checkpoint). This investigation concludes by suggesting that novel applications aimed at extending the functional inter-failure lifespan of HPC applications be devised.

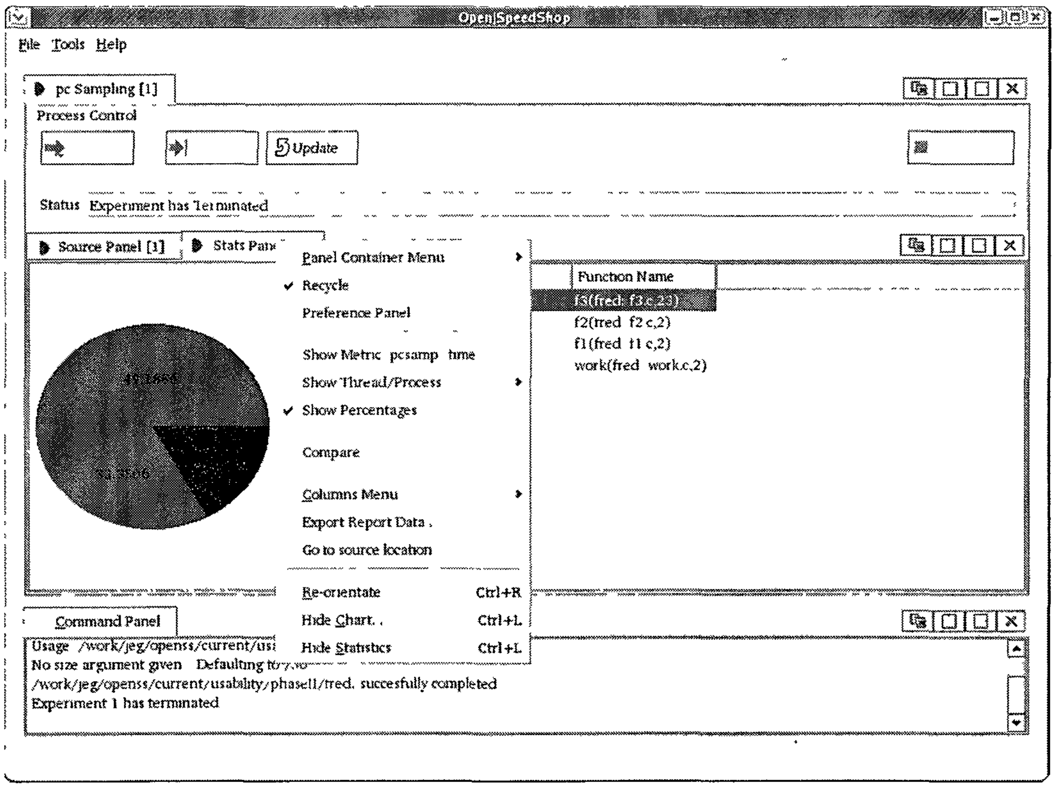
In many of these studies, the authors encounter tremendous difficulty in extracting relevant and legible reliability information from extreme-scale application log files [71]. Most concluded that the labyrinthine nature and enormous size of these documents greatly hinders both the speed with which one can perform root cause analysis and the efficient real-time monitoring and modeling of application performance. Sisyphus [72], a web-based log file analysis tool developed at Sandia National Laboratory, aims to resolve this conflict by filtering such log files via latent-semantic indexing. This process utilizes a network of regular expression algorithms to parse and display relevant information located within the log.

The health data collection processes outlined in existing studies, however, are both reactive and substantially system-centric. Furthermore, these approaches do not efficiently scale when large numbers of nodes are added to an HPC system.

As such, an efficient solution for the real-time generation of reliability metrics from highly-scaled HPC applications must be designed and implemented to facilitate further study in the resilience field. Further, these metrics should be handled autonomously, allowing the application to self-correct resilience issues without live human interaction. These goals, then, are the focus of the work described in this chapter. Here Gilgamesh is presented, developed by the author as a plug-in for the Open|SpeedShop performance analysis suite. Gilgamesh utilizes the dynamic instrumentation of binary source to efficiently collect program information for jobs run on extreme-scale HPC distributions. It then uses a scripting and database interface to handle the generated information and autonomously provision application reliability.

Open|SpeedShop (OSS) is an open source performance monitoring tool funded by the U.S. Department of Energy Tri-Labs at Los Alamos, Sandia, and Lawrence Livermore National Laboratories. OSS was initially developed by Silicon Graphics, Inc. (SGI) and later adopted as a community effort by the Krell Institute. Per its website, OSS is "an open source multi platform Linux performance tool which is initially targeted to support performance analysis of applications running on both single node and large scale IA64, IA32, EM64T, and AMD64 platforms. It is explicitly designed with usability in mind and targets both application and computer scientists" [73].

Open|SpeedShop contains four different user interfaces: graphical, command line, Python scripting, and batch. The graphical user interface (GUI) provides real-time visual representation of system and application performance. Likewise, the command line interface (CLI) enables a less costly method for quickly and easily viewing performance information. Lastly, the batch interface (utilized by Gilgamesh) serves as a means for external scripts and applications to call upon OSS to perform a specified function. Figure 2.1 showcases the Open|SpeedShop GUI in action – in this case, displaying program counter sampling information.



```
Usage /work/jeg/openss/current/usi
No size argument given  Defaulting to
/work/jeg/openss/current/usability/phaseII/fred. succesfully completed
Experiment 1 has terminated
```

Figure 2.1 Open|SpeedShop GUI

OSS also supports the development of user-created plug-ins (called ‘experiments’ in the OSS lexicon) which allow one to leverage the software’s mechanics to capture user-defined metrics from the target application. There are a number of application properties that OSS can extract and analyze without any modification or addition to its source. Among these are the following: floating point exceptions, hardware counter data, input/output information, MPI tracing, program counter sampling, and user time metric collection [74], most of which cannot be found in system log files. The Open|SpeedShop portion of Gilgamesh cherry picks collectors for resilience-pertinent information and establishes an interface through which the external storage and correction processes can access this data in real-time.

Prior research within the HPC resilience research community has concluded that there exists a need for the development of software which provides extreme-scale HPC applications with the ability to successfully continue computation in the event of non-catastrophic performance interruptions [75, 76]. The logical first step in this software's development cycle, then, is the creation of a module by which resilience-pertinent information is extracted from an HPC application. That is, in order to successfully provision fully resilient high performance programs, one must first gather the information to determine how and why the application is failing.

Gilgamesh leverages Open|SpeedShop to accomplish this via the dynamic instrumentation packages located within the OSS framework. Developed as an OSS plug-in, this portion of Gilgamesh utilizes the DPCL [77] and DynInst [78] packages that drive OSS's dynamic instrumentation capabilities to quickly generate resilience-pertinent metrics from running HPC applications. As the application executes, user-determined function calls trigger DynInst and DPCL to comb the program's source and generate a snapshot of its current state, dynamically instrumenting the application. These parameters are then collected by Gilgamesh's scripting framework and stored in an SQL database, where the autonomous correction module analyzes the data and resolves reliability-threatening issues before they impact application execution.

By default Gilgamesh is configured to capture specific metrics from a user application. This initial set of metrics constitutes a general class of parameters that are often useful to a wide range of applications, but, through the Gilgamesh graphical user interface, the user can define others.

Gilgamesh utilizes Open|SpeedShop's batch interface to intelligently attach its data generation module to the job queuing and execution processes without being explicitly launched from within a given user space. When a user submits a job to the batch scheduler, Gilgamesh recognizes this and prepares to attach itself to the application once it is scheduled and begins to execute. The interaction between Open|SpeedShop and Gilgamesh is displayed in Figure 2.2.

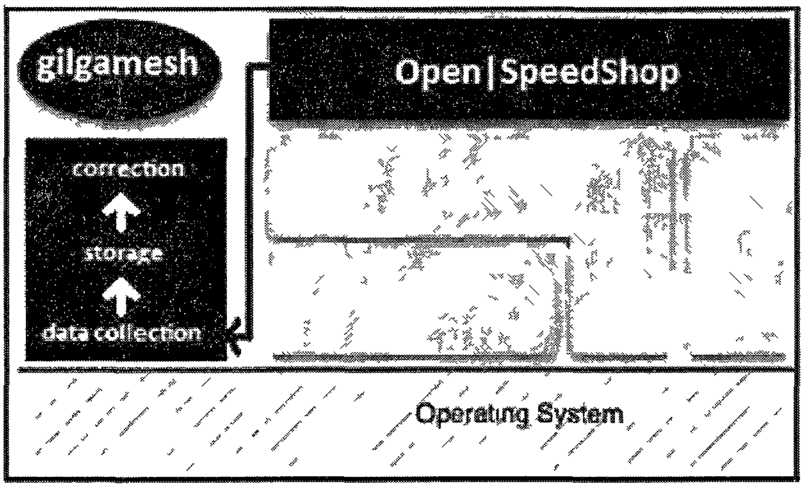


Figure 2.2 Interaction between Open|SpeedShop and Gilgamesh

As a plug-in for Open|SpeedShop, Gilgamesh's data collection module works by executing revised copies of the runtime.c (application-side) and collector.cxx (OSS-side) source files that initialize the various dynamic instrumentation processes. These files, then, work together to generate Binary Large Objects (blobs), which contain the actual information extracted from the application. The rate at which Gilgamesh produces blobs may be explicitly determined by the user via the Open|SpeedShop GUI. This dissertation will focus more on the data generation and storage aspects of Gilgamesh, as its analysis and correction routines have since been replaced by the failure indexing algorithm introduced in Chapter 4 and used in Chapter 5.

Gilgamesh contains a script which brokers storage of the reliability information captured by OSS. Launched in conjunction with the data collection processes as the target application leaves the queue, this script periodically enters OSS's batch interface and dumps the generated metrics into a SQL database which exists completely outside of the OSS realm. This greatly expands its flexibility – one could connect any number of additional applications to this database and, so long as it remains formatted in the Gilgamesh schema, the metric generation and storage processes will execute seamlessly. The rate at which this script interacts with OSS can be easily modified by editing an associated configuration file.

As application reliability information is stored, Gilgamesh then evaluates the data, determines if there exist any potential problems, and attempts to preemptively correct these issues via a number of available techniques. Once the storage script has completed execution, it instantiates a correction script that accesses the SQL database and statistically analyses the data captured by Open|SpeedShop. It is important to note that these scripts exist completely outside of the OSS framework – any database containing application reliability information can be analyzed by this correction module, so long as it is a SQL database with the Gilgamesh schema.

Unlike most contemporary approaches, the health determination algorithm used by Gilgamesh does not utilize a threshold-crossing algorithm. Instead, Gilgamesh determines if the application is trending towards failure. That is, instead of comparing the application information in the database to a collection of associated values and determining if the value exceeds the listed threshold, Gilgamesh's failure anticipation algorithm works by comparing all values of a given metric and the associated times at which these values were generated by the application. It then analyses the slope of this comparison and signals if the given attribute is trending towards a potential failure. This idea of an anticipation algorithm played a big role in the eventual creation of the Failure Index.

For example, if CPU usage is enabled in the Gilgamesh GUI, Gilgamesh will, when dynamically instrumenting the application's binary, extract per-node CPU usage from the application and store this information in the database. The correction script is then called and, instead of comparing this CPU usage value to a predetermined threshold, it takes this value and all previous recordings of this value for this run of the application and differentiates this data with respect to time. If it determines that that this parameter's slope is getting steeper at a sufficiently significant rate, the correction algorithm is triggered. Differentiating data in this manner is analogous with the concept of rate of occurrence of failure (ROCOF), discussed later in this dissertation.

This approach goes beyond contemporary threshold-related schemes. Should an application always run at 95% CPU usage yet show no symptoms of potential failure, Gilgamesh does not detect an anomaly. However, if an application typically utilizes 20% of the CPU, but has escalated to 30%, 50%, and 70% CPU usage over the past 5 minutes, then the tool identifies this as a potentially serious change of events.

Once Gilgamesh determines that some aspect of the application is trending towards failure, the autonomous correction module is alerted and begins work on mitigating the anticipated failure before it occurs. At this point, Gilgamesh has a multitude of options at its disposal: Schemes such as process re-instantiation or migration, node rejuvenation, and checkpointing/restarting are all available as potential remedies for the deteriorating application. Many of the best ideas discussed in the previous chapter were cherry picked in Gilgamesh's implementation of fault mitigation.

The remedy selected by Gilgamesh, of course, depends on the specific metric acting as catalyst for the failure. In general, node-related parameters such as CPU usage trigger process migration and node rejuvenation, while application-centric metrics such as floating point exceptions trigger process re-instantiation. Every attempt is made to avoid application-wide checkpoint/restart, as this form of fault-tolerance should be reserved for recovering from catastrophic failures. In scenarios where a proactive response to impending failure is possible, Gilgamesh chooses that approach.

Current research and development on Gilgamesh focuses on instrumenting a number of HPC applications with a full-capacity build of this software, in an effort to generate a wealth of information related to the failures encountered by applications running at an extreme scale. This information, then, will be used to retool the software to perform its duties in a substantially less costly manner, while increasing its ability to accurately anticipate failure. Future work includes further decreasing the software's overhead requirements and specifically increasing the intelligence of the autonomous correction component via the implementation of the FI. Real time monitoring using a combination of Gilgamesh and the FI is proposed as future work in Chapter 6.

2.2 Examining HPC Application Metrics

A common goal in studies done on resilience provision within HPC distributions is the development of an effective reliability, availability, and serviceability (RAS) logging and monitoring framework for detecting, circumventing, and quickly recovering from system and application failures. However, before this development can begin, it is both necessary and critical to characterize potential failure scenarios. This includes creating working definitions of these failures (as will be covered later in this document) and developing a sound understanding of the structural semantics and dependencies located within the target architecture. This section develops a means to characterize system- and application-level failures encountered within the HPC environment as a general architectural paradigm. Specifically, this section introduces the general concepts and procedures for describing an HPC dataset as used in the remainder of this dissertation.

To develop a working dataset that was both tangible and rich in failure and performance information, a number of system logs from existing large- and extreme-scale HPC deployments were obtained. The findings obtained by mining and modeling these records, combined with an in-depth examination and subsequent understanding of requisite HPC architectural components drive the bulk of this section.

The multiple processors that serve as nodes within an HPC environment, much like those found in any other architectural permutation, output various information via system- and application-level log files. However, as discussed in the previous section, there are a number of hurdles encountered when attempting to mine this information in an effort to better understand failure characteristics and capture system health indicators.

As discussed in the first chapter, there is no standardized design method for culling performance information from these files, nor is there a standard means of arranging the data. In multi- and heterogeneous-core environments, in particular the dominant cluster computing model, there is the potential for any number of divergent processor technologies. Various manufacturers and clock speeds, amongst other discrepancies, lead to diverse, sometimes radically different log formats amongst nodes. This is a major obstacle in identifying overall environmental health, as, without another level of abstraction above the various node-specific reporting modules, performing any detailed comparisons between individual system components proves very cumbersome.

Also, in most systems, only performance metrics pertaining to individual nodes or components within the individual machines are logged, and, as such, no multi-component or system-wide performance indicators are currently used in formulating log data. These are often crucial values that must be taken into consideration when formulating a given system's overall health and viability.

Briefly discussed in the first chapter, the Blue Gene/L (BG/L) log file is used as our test dataset in this section. BG/L, a well-known IBM-developed supercomputer, is an extreme-scale HPC deployment which strictly adheres to many of the core design principles of high performance computing and, as such, serves as a worthy target architecture for the study of HPC failure activity. Contained in its 710 MB, 4,747,963 line log file was performance and error information covering a six month period, from

June 3rd, 2005 to January 4th, 2006. The information outlined in this section is the result of analysis into the behaviors and trends located within this rich performance dataset.

According to the Top500 Supercomputer website, BG/L – located at Lawrence Livermore National Laboratory – was at one time the largest supercomputer in existence. The system is comprised of 106,496 dual-processor compute nodes, of which 67% are 512 MB and 33% are 1 GB. It also contains 1,664 I/O nodes, 212,992 IBM PowerPC CPUs, and a total disk space of 1.89 PB. Within the Blue Gene/L architecture, each rack is divided into two parts: a top midplane and a bottom midplane. Each part contains 16 node cards, 1 service card and 4 link cards, and there are 32 compute nodes and 4 optional I/O nodes on each node card [79].

During the time interval covered in the log file, there were 4,747,963 messages sent to the log. Each message contains the *time*, *location*, *RAS* or *NULL*, *facility*, *severity*, and *event description* information shown in Figure 2.3. Locations are denoted by codes which represent a particular hardware component.

```
2005-08-02-17.18.18.545821 NULL RAS BGLMASTER FAILURE mmcs_server exited normally with exit code 1
2005-08-02-18.05.32.652047 NULL RAS BGLMASTER FAILURE idoproxy exited normally with exit code 0
2005-08-02-18.05.42.661358 NULL RAS BGLMASTER FAILURE mmcs_server exited normally with exit code 15
2005-08-02-18.05.47.668332 NULL RAS BGLMASTER FAILURE ciodb exited normally with exit code 15
2005-08-03-13.34.51.000398 NULL RAS BGLMASTER FAILURE mmcs_server exited normally with exit code 15
2005-08-10-09.09.58.139632 NULL RAS BGLMASTER FAILURE mmcs_server exited normally with exit code 15
2005-08-12-07.20.27.921676 NULL RAS BGLMASTER FAILURE ciodb exited abnormally due to signal: Aborted
```

Figure 2.3 An Example of BlueGene/L's Log Format

The facility variable, found in every log entry, indicates the hardware or service affected by the corresponding reported event [80]. This value can be characterized into one of 10 types: *MMCS*, *APP*, *KERNEL*, *LINKCARD*, *DISCOVERY*, *MONITOR*, *HARDWARE*, *CMCS*, *BGLMASTER*, and *SERV_NET*. *MMCS* represents the system's midplane management and control service, while *CMCS* stands for core management and control system. *KERNEL* indicates events related to hardware instruction and data manipulation. *DISCOVERY* is a service that monitors hardware changes. *MONITOR* is another control system component which provides various hardware status metrics such as temperature. *BGLMASTER* is a service that controls the *MMCS*. Also within the log messages are six severity levels: *INFO*, *WARNING*, *SEVERE*, *ERROR*, *FATAL*, and *FAILURE*.

As outlined in Tables 2.1 and 2.2, 91% of messages contained in the log are generated by the *KERNEL* facility, and 79% of messages are of the *INFO* severity level. Also, 2% of all messages were generated by the *DISCOVERY*, *MONITOR*, and *HARDWARE* facilities – all indicating hardware abnormality – and 4% of all log data was generated by *APP*.

Table 2.1 Number of Messages per Facility, BG/L

Facility	Number of Messages	Facility	Number of Messages
MMCS	88,930	LINKCARD	1,170
APP	228,536	DISCOVERY	97,172
KERNEL	4,324,967	MONITOR	1,681
CMCS	211	HARDWARE	5,148
BGLMASTER	145	SERV_NET	3

Table 2.2 Number of Messages per Severity Level, BG/L

.Severity Level	Number of Messages	Severity Level	Number of Messages	Severity Level	Number of Messages
INFO	3,735,823	SEVERE	19,213	FATAL	855,501
WARNING	23,357	ERROR	112,355	FAILURE	1,714

The *FAILURE* severity level contained messages produced exclusively by failing system components. It was also observed that many event descriptions in the *FATAL* level represented failure activity, such as "*panic: -stopping execution*". It was consequently assumed that all failure events were located in the *FATAL* or *FAILURE* severity levels and, given that the objective of this initial study was the isolation and analysis of failure information from a given BG/L log file, messages other than those of *FATAL* and *FAILURE* severity were discarded. This resulted in a collection of 855,501 *FATAL* and 1,714 *FAILURE* messages, which are further detailed in Tables 2.3 and 2.4 respectively.

Table 2.3 Time between *FATAL* Messages, BG/L

Time Between FATAL Messages	
Number of Messages	855,501
Minimum	0 seconds
Maximum	303533 seconds
Mean	21.683 seconds
Median	0 seconds

Table 2.4 Time between *FAILURE* Messages, BG/L

Time Between FAILURE Messages	
Number of Messages	1,714
Minimum	0 seconds
Maximum	3382246 seconds
Mean	8,193 seconds
Median	0 seconds

The mean values found in these results suggest that *FATAL* and *FAILURE* messages are generated at an incredibly high frequency. Further, a median value of nearly 0 seconds for both the time between *FATAL* messages and the time between *FAILURE* messages confirms that more than half of all failure-related messages are generated almost simultaneously. After further analysis, it was found that there existed a number of *FATAL*-tagged messages that did not, in fact, suggest legitimate system failures. For example, "*guaranteed data cache block touch*", "*store operation..................1*", or "*instruction address space......0*" are all tagged as *FATAL*, but do not result in the system entering a compromised state. However, many of them – such as "*Power deactivated*", "*kernel terminated*" and "*Lustre mount FAILED*" – strongly suggested traceable system failures.

It was discovered that in many cases a single *FATAL* or *FAILURE* message may be repeatedly and massively reported. For example, 346 *FAILURE* messages, containing the message "*Temperature over Limit on link card*" were repeatedly reported from 2005-11-07-12.28.58 to 2005-11-07-12.37.20. Put into perspective, that single message was generated almost every second for close to ten minutes. Because of this phenomenon, two datasets, *U* and *L*, were constructed.

The first dataset *U* contains all *FATAL* and *FAILURE* messages, less those that do not infer actual, legitimate system failures. *U*, then, represents an upper bound for all system failure behavior that has been reported in the log, as it contains all reported system failures. Certain message patterns, such as those containing only numeric codes, were filtered from the data as non-failure messages, and are not included in *U*. The second dataset *L* contains only those *FATAL* and *FAILURE* messages inferring system failures. Similarly, *L* represents a lower bound for reported system failure behavior, as it is a subset of the actual failures. Like in *U*, particular message patterns were used to form *L* via the extraction of messages that suggested system- or application-affecting failures.

Repeated messages stemming from a single failure were removed from both datasets, as the target of this study is less the messages in the log file and more the actual failure behavior of the system. This process is conducted inductively with a time interval of 60 seconds. As an illustration, suppose that there are three identical messages from the same source appearing at times 1, 30, and 100 seconds after log initialization. Because the time between messages one and two is 29 seconds, message two is removed from both datasets. However, because the time between messages two and three is 70 seconds, message three is not removed. The time between the individual messages contained in both datasets is summarized in Table 2.5.

Table 2.5 Time between Messages, Datasets *U* and *L*

Time Between Messages, L		Time Between Messages, U	
Count	72208	Count	185102
Minimum	0	Minimum	0
Maximum	460,510 seconds	Maximum	330,072 seconds
Mean	254,101 seconds	Mean	100.0569 seconds
Median	0	Median	0

Although the mean time between the messages contained in both datasets appears longer than the mean found in the original data, the value remains extremely low. In researching the genesis for this abnormality, it was discovered that many of the logged messages stemming from a single incident and occurring during similar time windows were reported by a large group of nodes. For example, 2,048 compute nodes reported an "*rts internal error*" within a very small timeframe – from 2005-06-14-11.15.09 to 2005-06-14-11.16.15 (a window of only one minute and six seconds).

However, because this dissertation is most concerned with failure information suggesting application interruption (recall from the previous chapter our working definitions for both failure and resilience), the time between messages data was further processed by making an assumption that the system has a certain amount of time to repair and that a given application uses all nodes in the system. If the time between two failures is less than time to repair, the latter failure can no longer affect the application because the system is being repaired and is no longer in production mode. Thus, any time between failures less than or equal to the time to repair those failures could be removed from the datasets. For the purposes of this section, the results of this removal are examined using four different time to repair values—1, 5, 10, and 20 minutes. The results are summarized in Table 2.6 and Table 2.7.

Table 2.6 Time between Messages, *L*, After Repeated Message Removal

Time to Repair (L)	1 min	5 min	10 min	20 min
Count	476	453	436	375
Minimum	10.08 min	10.52 min	10.6 min	20.25 min
Maximum	5.3 day	5.3 day	5.3 day	5.3 day
Mean	10.69 hrs	11.24 hrs	11.67 hrs	13.53 hrs
Median	3.96 hrs	4.94 hrs	5.6 hrs	8.26 hrs

Table 2.7 Time between Messages, *U*, After Repeated Message Removal

Time to Repair (U)	1 min	5 min	10 min	20 min
Count	1023	912	872	741
Minimum	1 min	5.23 min	10.05 min	20.1 min
Maximum	3.8 day	3.8 day	3.8 day	3.8 day
Mean	5.02 hrs	5.63 hrs	5.89 hrs	6.97 hrs
Median	1.81 hrs	2.41 hrs	2.57 hrs	3.56 hrs

By using the datasets *L* and *U*, which represent the best and worst case system failure behavior respectively, the results form Table 2.6 and Table 2.7 suggest that the mean time to interrupt for a BG/L application plausibly falls between 5 and 14 hours, depending on the utilized time to repair.

In addition, when a single failure generates multiple messages, the first messages generated will typically have a larger amount of time between them while their duplicates will have a much smaller time between messages. This suggests that for duplicate messages a large time between messages should change very little after the TTR becomes larger than time between messages. Time between message measures for both U and L of more than 100 minutes are very stable for all TTR of one minute or more, indicating that one minute is sufficient to remove the vast majority of duplicate messages for a single root cause.

In conclusion, results suggested that the BG/L system has a mean time to failure (MTTF) of 5.89 hours – or roughly 4 times a day – for an application with a time to repair of ten minutes. This is also assuming a full, evenly distributed system load, which is of course not always the case. Failures may occur as often as 10.6 minutes apart, or, likewise, one could go days (in the case of the 10 minute MTTR assumption – 3.8 days) without observing a system breakdown.

This initial study proved that BG/L's log files provide a wealth of information, but much of it is of no use to those interested in provisioning resilient applications for it and similar technologies without first filtering the data and performing appropriate statistical analysis to arrive at correct and logical values. Similarly, the results gathered from observing BG/L are not comparable to those of other systems without first developing a way to normalize the data.

The remainder of this dissertation and, in fact, the novel contribution of this work, will be devoted to developing such a measure. By first conceptually introducing inequality indices and then demonstrating their effectiveness in describing the failure behavior on an HPC system reliability dataset, the following chapter demonstrates the utility of such a measure.

CHAPTER 3

INEQUALITY MEASURES FOR HPC RELIABILITY DATA

3.1 Introduction and Justification

A needed element in the study of resilient High Performance Computing (HPC) applications is the creation and widespread adoption of a normalized metric representing an application's failure behavior [81, 82]. Such a metric opens the door for enhanced quality of service provision and a set of standard operating expectations for large-scale HPC application development and execution.

The contemporary expansion in HPC system size and complexity has generated a lack of quantitative expectations in application performance due to the volatile failure activity encountered when computing at such a high scale [83]. A standardized and widely adopted failure metric expedites the process of informing research, development and administration personnel of unexpected application behavior and acts as notification that application processes are failing to maintain an adequate level of performability [84].

In this chapter, inequality indices are introduced from a statistical point of view and applied to an existing HPC dataset. These metrics serve as the initial step in the long-term research and development of new resilience-related values, as their scale- and location-invariant nature and normalized representation allow HPC researchers to adequately compare inequality in failure behavior across various systems and applications. Specifically, the Gini, Atkinson and Theil indices are introduced in this chapter, with the final sections detailing the application of the Gini and Atkinson indices to an HPC system reliability dataset.

3.2 The Lorenz Curve

Inequality indices are based on the Lorenz curve, introduced in 1905 by Max Otto Lorenz as a graphical representation of a distribution's level of equality, wherein observed events are compared to distributions with perfect equality [85]. The Lorenz curve is based on a convex function, and has been widely adopted by economists for use in comparing income distributions [86].

Figure 3.1 shows an example Lorenz curve. The plot is a 45 degree line with the *y-axis* representing the percent of event occurrence and the *x-axis* representing, as a percentage, an increase in population. Logically, then, the upper limit for both scales is 100 – one hundred percent of a population earns, for example, one hundred percent of all income and, analogously, zero percent of the population earns zero percent of all income.

When income is distributed evenly – that is, 10% of the population earns 10 percent of the income, 50% of the population earns 50 percent of the income, and so on – the Lorenz curve represents what is called the egalitarian line, or the line of absolute equality. The egalitarian line is represented by the 45-degree line in Figure 3.1.

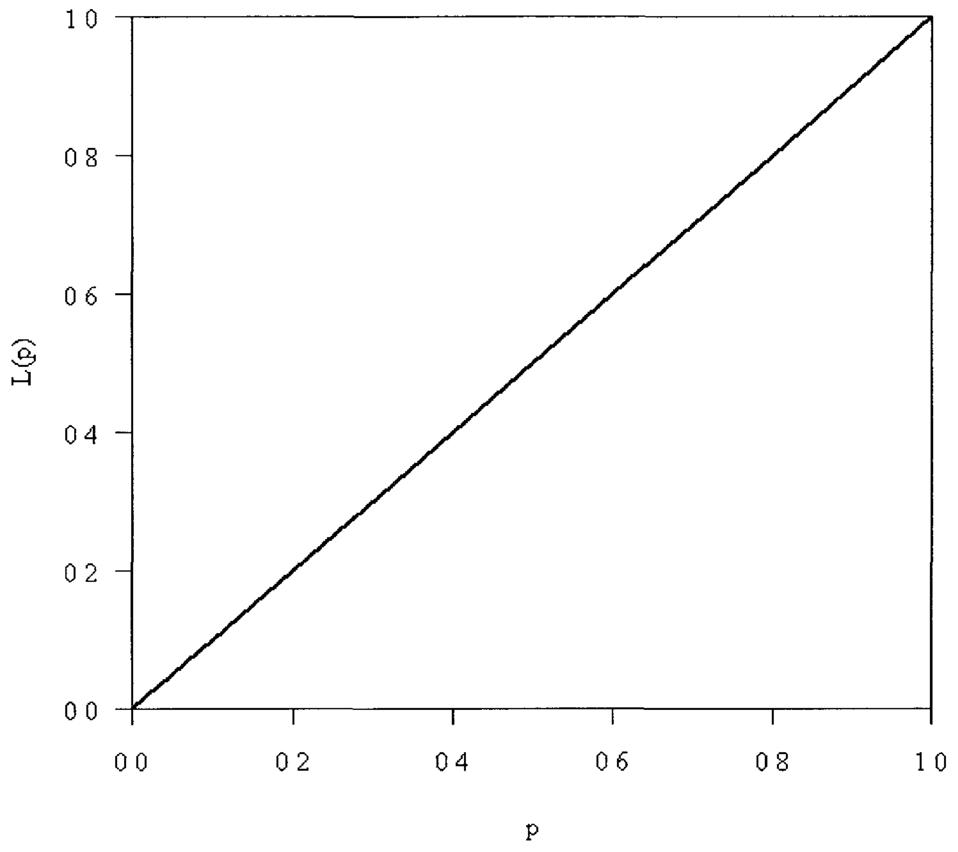


Figure 3.1 Lorenz Curve Given Total Equality

The representation of unequal distributions results in an empirical Lorenz curve (L) which is well defined on the interval $[0,1]$. Note that $L(0) = 0,\ L(1) = 1$. If all event values followed a uniform distribution, then, the empirical Lorentz curve would equal the egalitarian Lorenz curve as shown in Figure 3.1. However this is typically not the case and the empirical Lorenz curve L rests below the diagonal, as shown in (3.1). The Lorenz curve is defined as

$$L(p) = \frac{1}{E[X]}\int_0^p F^{-1}(u)du,$$

$$F^{-1}(s) = \inf\{u: F(u) \geq s\}\,; 0 < s < 1\,. \tag{3.1}$$

Here, F is the distribution's cumulative distribution function. One can compare the difference in inequality between two distributions by comparing their Lorenz curves, with the curve that most deviates from the line of absolute equality representing the distribution containing the higher amount of inequality. This situation is represented in Figure 3.2, with the Lorenz curve closer to the right and bottom of the graph representing a distribution containing more inequality than the distribution represented by the Lorenz curve closest to the line of absolute equality.

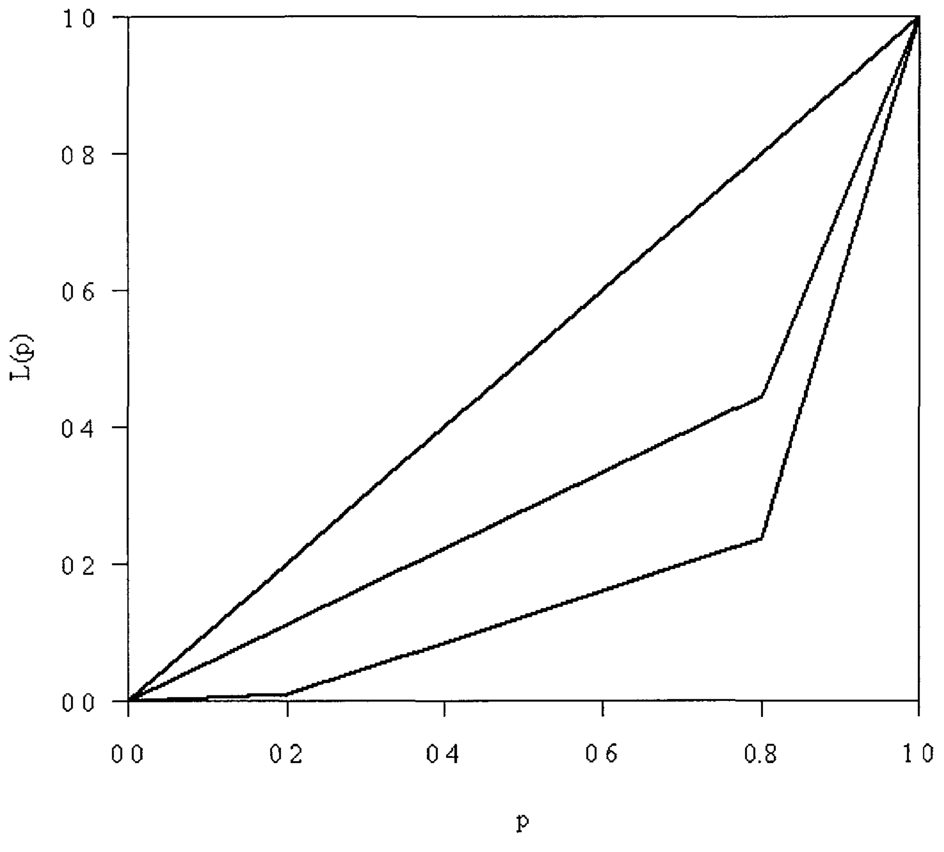


Figure 3.2 Lorenz Curves for Two Distributions with Varying Levels of Inequality

The following three sections introduce various types of inequality indices – means of representing the inequality in a distribution via the creation of location- and scale-invariant coefficients. All three are based on the Lorenz curve.

3.3 The Gini Index

The Gini index is a normalized measure of the statistical inequality of a given dataset which generates a coefficient based on a 0-to-1 system, with a value of 0 measuring total equality and a value of 1 measuring total inequality. Using an example illustrating this metric's use in economics, the Gini index is often used to express the level of inequality in national income distributions, with as of 2009 Sweden having the world's lowest Gini coefficient (0.23) and South Africa having one of the highest (0.67) [100]. This implies that Sweden has the most equal distribution of wealth amongst the world's nations, and South Africa has one of the most unequal. Taking this to an extreme, a nation reporting a Gini coefficient of 1 in such a study would indicate that one individual controls *all* of that nation's wealth, with the remaining members of the population each having no income. Likewise, a Gini coefficient of 0 would indicate that all individuals in a given country earn the exact same income.

Statistically, the Gini Index measures the ratio of the area between the Lorenz curve ($L(p)$) and the line of absolute equality (I), to the area under that line, which is 1/2. More easily stated, the Gini Index (3.2) represents twice the area between I and $L(p)$, which can be mathematically expressed as

$$G_F \coloneqq 2 \int_0^1 | I - L(p)| \, dt \, . \tag{3.2}$$

When one uses the Gini index, a Gini coefficient is generated which represents the level of inequality measured. As mentioned, a perfectly uniform distribution would generate a Gini coefficient of zero, as mathematically there would be no area between the line of absolute equality and the Lorenz curve. Likewise, in the event of complete inequality, the Gini coefficient would be 1, coinciding with straight lines at the lower and right boundaries of the curve. The general definition for the Gini index (3.3) , where X_1 and X_2 are independent random variables with mean μ, is

$$G_F = \frac{1}{2\mu} E(|X_1 - X_2|) = \frac{1}{2\mu} \int_R \int_R | x_1 - x_2 | \, dF(x_1) dF(x_2) \,. \tag{3.3}$$

In addition to economics, the Gini Index has applications in biodiversity [87], chemistry [88], agriculture [89], and engineering [90]. In general, it can be used to measure any subject involving a distribution.

3.4 The Atkinson Index

The Atkinson index is similar to the Gini index, with the notable inclusion of a coefficient that allows it to examine movement within different sections of a distribution [91]. The Atkinson index becomes more sensitive to changes at the lower end of the distribution as this coefficient approaches 0. Conversely, as this coefficient approaches 1, the Atkinson index becomes more sensitive to changes at the upper end of the distribution. Like the Gini index, the Atkinson index generates a coefficient which measures a distribution's departure from uniformity using a 0-to-1 system, again with 0 representing total equality and 1 representing total inequality. The Atkinson index given multiple events $y_1, \dots, y_n$ with mean μ (3.4) is properly defined as

$$A_\varepsilon(y_1, \dots, y_n) = \begin{cases} 1 - \dfrac{1}{\mu}\left(\dfrac{1}{N}\displaystyle\sum_{i=1}^{N} y_i^{1-\epsilon}\right)^{\frac{1}{1-\epsilon}}, & 0 < \varepsilon < 1 \\ 1 - \dfrac{1}{\mu}\left(\displaystyle\prod_{i=1}^{N} y_i\right)^{\frac{1}{N}}, & \varepsilon = 1 \end{cases} \,. \tag{3.4}$$

The distinguishing feature of the Atkinson Index is its ability to measure movements in different segments of a given distribution. Like the Gini index, the

Atkinson index has seen uses largely in economics, but also in biodiversity and chemistry, amongst other fields. The inclusion of the parameter makes it particularly useful when applied to large data sets.

3.5 The Theil Index

The Theil index is a measure of inequality closely related to the Atkinson index – in fact, a Theil coefficient can be transformed into an Atkinson coefficient and vice-versa. Specifically, the Theil index is a measure of entropy, where maximum entropy occurs when there is perfect equality [92].

Like the Gini and Atkinson indices, the Theil index generates a coefficient based on a 0-to-1 scheme. Here, however, the coefficient measures the level of entropy in the distribution, rather than the level of inequality. It is important to note than an increasing Theil index does *not* indicate increasing entropy. It instead indicates an increasing level of redundancy – the gap between maximum and actual entropy. Thus, it actually indicates *decreasing* entropy. The general Theil index given multiple events $x_1, \ldots, x_n$, where x_i is the value of the ith event in the distribution, $\bar{x}$ is the mean value of all events in the distribution and N is the population (3.5) is defined as

$$T_j = T_{\alpha=1} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i}{\bar{x}} \times ln\frac{x_i}{\bar{x}}\right). \tag{3.5}$$

The following section introduces the HPC dataset to which the Gini and Atkinson indices will be applied. Following that, the resulting coefficients and Lorenz curves generated by that dataset will be reported. Finally, various conclusions will be drawn relative to these results.

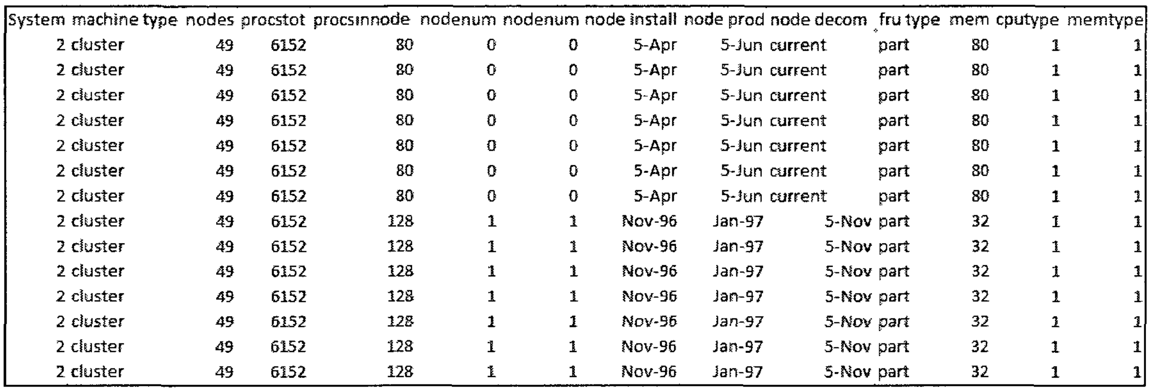
3.6 Application to HPC Data and Results

3.6.1 Introduction to the Dataset

This section demonstrates the utility of inequality indexing by generating Gini and Atkinson coefficients for each of the 23 HPC machines housed at Los Alamos National Laboratory from 1996 to 2005. The dataset containing this information was released in 2005 as a collection of CSV files and was initially analyzed by Schroeder et al from Carnegie Mellon University's Parallel Data Laboratory [99]. In that initial study, a number of statistical conclusions are drawn in a similar fashion to our team's later work on the BlueGene/L machine at Lawrence Livermore National Laboratory, which was discussed in Chapter 2.

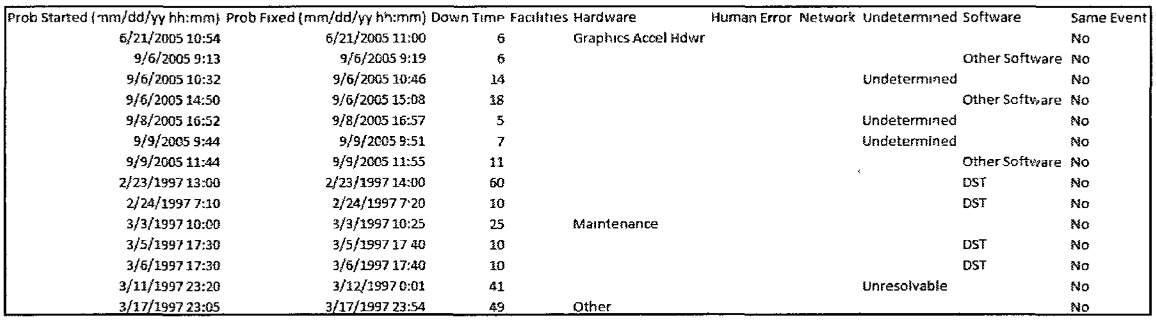
Gini, Atkinson (with parameter 0.99) and Atkinson (with parameter 0.20) coefficients will now be generated for each system contained in the log file. Doing so will both demonstrate the viability of inequality index coefficients as a metric for comparing HPC system failure inequality as well as provide finer-grain information regarding the inequality stemming from each machine's failure behavior during the time frame covered in the log file.

The nearly 3 MB file contains 23,739 failure events from 23 of LANL's systems, accrued over a nine-year period. To maintain confidentiality, each system is labeled using a number system from 2 to 24 (systems "2", "3", etc), with System 2 being the oldest system recorded in the log and System 24 being the youngest. Each failure record contains the time when a failure was first reported (labeled 'prob started'), the time when the issue causing the failure was resolved ('prob fixed'), the total system downtime (in minutes) resulting from the failure, the system and node affected by the failure, the type of workload running on the node at the time of the failure and the root cause of the failure. Screenshots representing the left and right halves of the original CSV file (taken from the same sample) are shown in Figures 3.3 and 3.4, respectively.



System	machine type	nodes	procstot	procsinnode	nodenum	nodenum	node install	node prod	node decom	fru type	mem	cputype	memtype
2	cluster	49	6152	80	0	0	5-Apr	5-Jun	current	part	80	1	1
2	cluster	49	6152	80	0	0	5-Apr	5-Jun	current	part	80	1	1
2	cluster	49	6152	80	0	0	5-Apr	5-Jun	current	part	80	1	1
2	cluster	49	6152	80	0	0	5-Apr	5-Jun	current	part	80	1	1
2	cluster	49	6152	80	0	0	5-Apr	5-Jun	current	part	80	1	1
2	cluster	49	6152	80	0	0	5-Apr	5-Jun	current	part	80	1	1
2	cluster	49	6152	80	0	0	5-Apr	5-Jun	current	part	80	1	1
2	cluster	49	6152	128	1	1	Nov-96	Jan-97	5-Nov	part	32	1	1
2	cluster	49	6152	128	1	1	Nov-96	Jan-97	5-Nov	part	32	1	1
2	cluster	49	6152	128	1	1	Nov-96	Jan-97	5-Nov	part	32	1	1
2	cluster	49	6152	128	1	1	Nov-96	Jan-97	5-Nov	part	32	1	1
2	cluster	49	6152	128	1	1	Nov-96	Jan-97	5-Nov	part	32	1	1
2	cluster	49	6152	128	1	1	Nov-96	Jan-97	5-Nov	part	32	1	1
2	cluster	49	6152	128	1	1	Nov-96	Jan-97	5-Nov	part	32	1	1

Figure 3.3 Leftmost Columns Sample, LANL Failure Dataset



Prob Started (mm/dd/yy hh:mm)	Prob Fixed (mm/dd/yy hh:mm)	Down Time	Facilities	Hardware	Human Error	Network	Undetermined	Software	Same Event
6/21/2005 10:54	6/21/2005 11:00	6		Graphics Accel Hdwr					No
9/6/2005 9:13	9/6/2005 9:19	6						Other Software	No
9/6/2005 10:32	9/6/2005 10:46	14					Undetermined		No
9/6/2005 14:50	9/6/2005 15:08	18						Other Software	No
9/8/2005 16:52	9/8/2005 16:57	5					Undetermined		No
9/9/2005 9:44	9/9/2005 9:51	7					Undetermined		No
9/9/2005 11:44	9/9/2005 11:55	11						Other Software	No
2/23/1997 13:00	2/23/1997 14:00	60						DST	No
2/24/1997 7:10	2/24/1997 7:20	10						DST	No
3/3/1997 10:00	3/3/1997 10:25	25		Maintenance					No
3/5/1997 17:30	3/5/1997 17:40	10						DST	No
3/6/1997 17:30	3/6/1997 17:40	10						DST	No
3/11/1997 23:20	3/12/1997 0:01	41					Unresolvable		No
3/17/1997 23:05	3/17/1997 23:54	49		Other					No

Figure 3.4 Rightmost Columns Sample, LANL Failure Dataset

Failures are detected by an automated monitoring system that pages operations staff whenever a node is down. The operations staff then creates a failure record in the database specifying the start time of the failure and the system and node affected. They then turn the node over to a system administrator for repair. Upon repair, the system administrator notifies the operations staff, who then put the node back into the job mix and fill in the end time of the failure record. If the system administrator was able to identify the root cause of the problem, he provides operations staff with the appropriate information for the 'root cause' field of the failure record. Otherwise the root cause is specified as 'undetermined'. Note that the dataset covers each of the 23 systems only during their time in a production environment, and testing/debugging as well as maintenance time periods are not covered.

Though it covers multiple systems, the LANL dataset is similar to the BG/L dataset in that various sources of failure are broken up into facilities. As in Chapter 2, 'facilities' here refers to the various components of the system acting as possible origins for the failures that occur on that system. The facilities themselves are however not identical to those found in the BG/L dataset. Instead of the ten facilities represented there, in the LANL data there are only five: *HARDWARE, SOFTWARE, NETWORK*, *HUMAN ERROR*, and *UNDETERMINED.*

Further, instead of the machine or operating system autonomously generating the facility of origin in the log file, here an actual human being records the suspected facility in the log, which is later investigated and confirmed or changed at subsequent operations staff meetings. The log also contains a column for noting whether the current failure event is due to the previously listed event, again decided by a human operator, not the system itself. Again, it is important to note that the dataset contains information from multiple machines, which is unlike the previously-used BG/L data (which contained information from only one system).

Each entry in the log file contains numerous columns. Specifically, those columns are the name ("2", "3", etc.) and type (cluster, graphics card, etc.) of each system containing a logged event, the number of nodes housed in that system, the total number of processes running on the machine as well as the number of processes running on each node in the machine at the time of the failure, the machine node reporting the failure, the installation date and production commission and decommission dates of that node, the purpose of that node, the start and end dates of the failure in question, the total downtime in minutes caused by the failure in question, and finally the facility of origin for that failure. This analysis pays specific attention to the 'system' and 'down time' columns of each entry –inequality coefficients are generated for system down time for each of the 23 machines found in the file. All of these failures are caused by one of the five facilities reported in the log. Recall that the scale- and location-invariant nature of inequality indexing allows us to compare entries reported by multiple machines with one another, regardless of node count or system type. This is why the LANL dataset and its 23 machines were chosen for this case study.

The following section reports the resulting inequality coefficients for each of the 23 systems found in the log file. Coefficients were generated via the Gini and Atkinson indices, with two parameter variations utilized for the Atkinson index: $\epsilon = 0.99$, which generates an Atkinson index coefficient utilizing all events found in the file, and $\varepsilon = 0.20$, which was chosen in accordance with the findings from [103], which found that "redistributions at the lower end of the distribution also have a greater impact on mortality."

Recall the formulation of the Gini index from (3.3). For this dataset, then, the Gini index coefficient is computed with respect to $D(x_i)$, the amount of system downtime (in minutes) caused by an individual failure x_i, $i = 1, \dots, n$. Here, x_1 represents the earliest failure record located in the dataset and x_n represents the latest, with μ representing the average downtime caused by each failure event located in the file. The cumulative distribution function $F(i)$, representing the amount of downtime caused by all failures up to i (3.6), is then:

$$F(i) = \sum_{j=1}^{i} D(x_i) \, . \tag{3.6}$$

The Atkinson index coefficient is also calculated with respect to the downtime caused by each failure record located in the log file. Using the conventions from (3.4), the Atkinson index coefficient is calculated with respect to $y_i = D(x_i)$, with y_1 representing the down time caused by the earliest failure recorded to the log file and y_n representing the downtime caused by the latest. N represents the total number of failures recorded to the dataset and μ represents the average amount of system downtime caused by each failure.

Results were generated for the 20th, 40th, 60th, 80th, and 100th percentile of events found in the LANL data relative to each of the 23 systems. Each system's failures were grouped according to their time of occurrence, with the first event in each group being the one generated by the earliest failure, and the last event being the one generated by the latest. These percentiles then represent the lifetime of each system as represented in the dataset. The resulting coefficients will illustrate the level of inequality in the downtime caused by each system failure as each of these systems age. Further, Lorenz curves were created for each of the 23 systems. Lastly, the 23 resulting Gini coefficients were averaged in relation to each percentile. All of these results are reported in section 3.6.2. Section 3.6.3 will draw conclusions from these results.

3.6.2 Results

Tables 3.1 through 3.23 display the per-percentile inequality coefficients generated for each of the 23 systems found in the LANL failure dataset. Figures 3.5 through 3.73 show bar graphs showing the resulting impact $D(x_i)$ of each failure x_ι exhibited by the system, histograms showing the frequency of system failures at various levels of impact are also given (note that the log of these values has been taken to better illustrate differences in failure activity between each system) and a Lorenz curve for each system, representing the overall level of inequality in the downtime resulting from each failure incurred by that system.

Following the generation of inequality coefficients, bar graphs, histograms and Lorenz curves for all 23 systems, the resulting Gini, Atkinson (ε =0.20), and Atkinson (ε=0.99) coefficients were grouped according to percentile and subsequently averaged. The resulting values located in Table 3.24 represent the average fluctuation in inequality for the down time caused by *all* failures across *all* systems as these systems age.

Conclusions based on these results will be drawn in the following section.

Table 3.1 Inequality Coefficients, System 2, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
2	20th	1421	211,800	0.6010276	0.534944	0.1286560
2	40th	2842	518,491	0.6731329	0.6346716	0.2116812
2	60th	4263	687,349	0.6908012	0.6356018	0.2292204
2	80th	5684	794,993	0.6732575	0.5996397	0.2147699
2	100th	7105	1,008,366	0.6827248	0.6047009	0.2245452

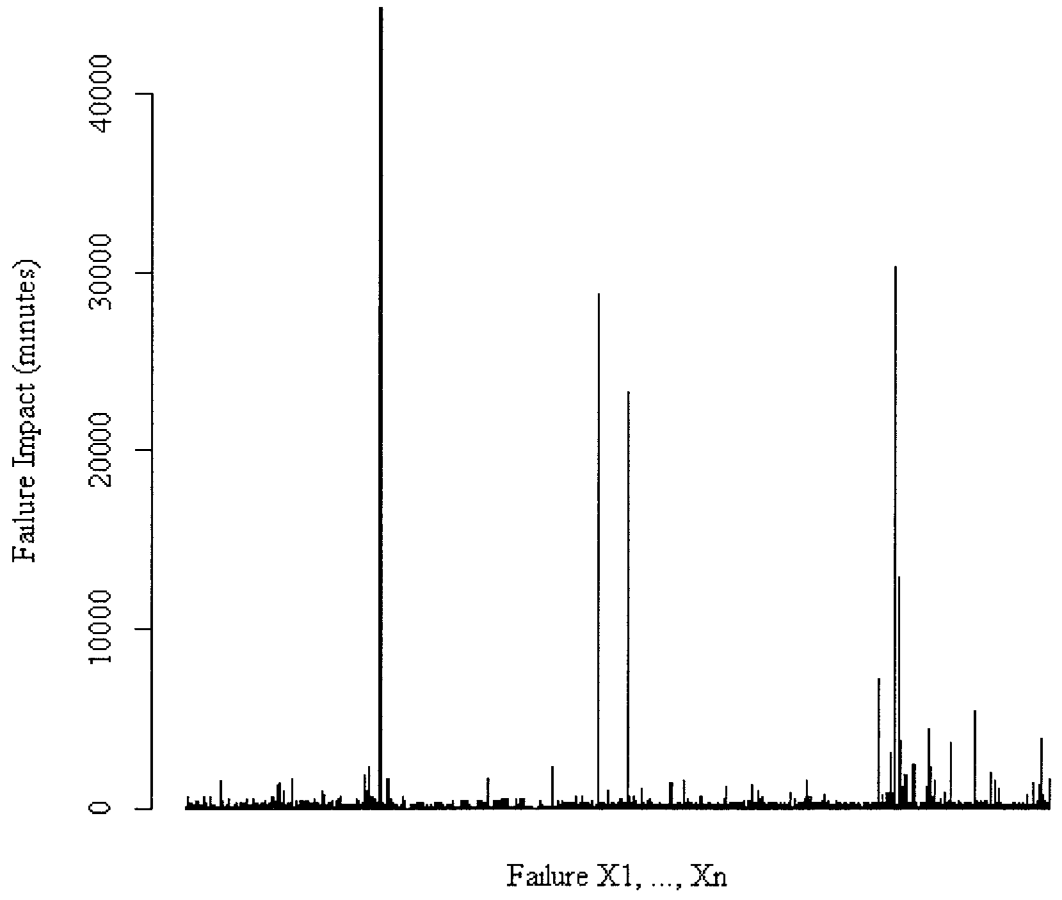


Figure 3.5 Impact of Each Failure, System 2, LANL Data

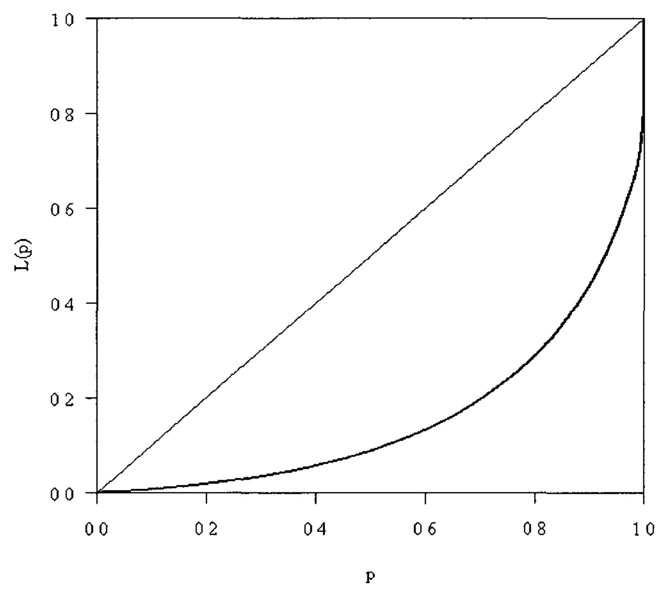


Figure 3.6 Lorenz Curve, System 2, LANL Data

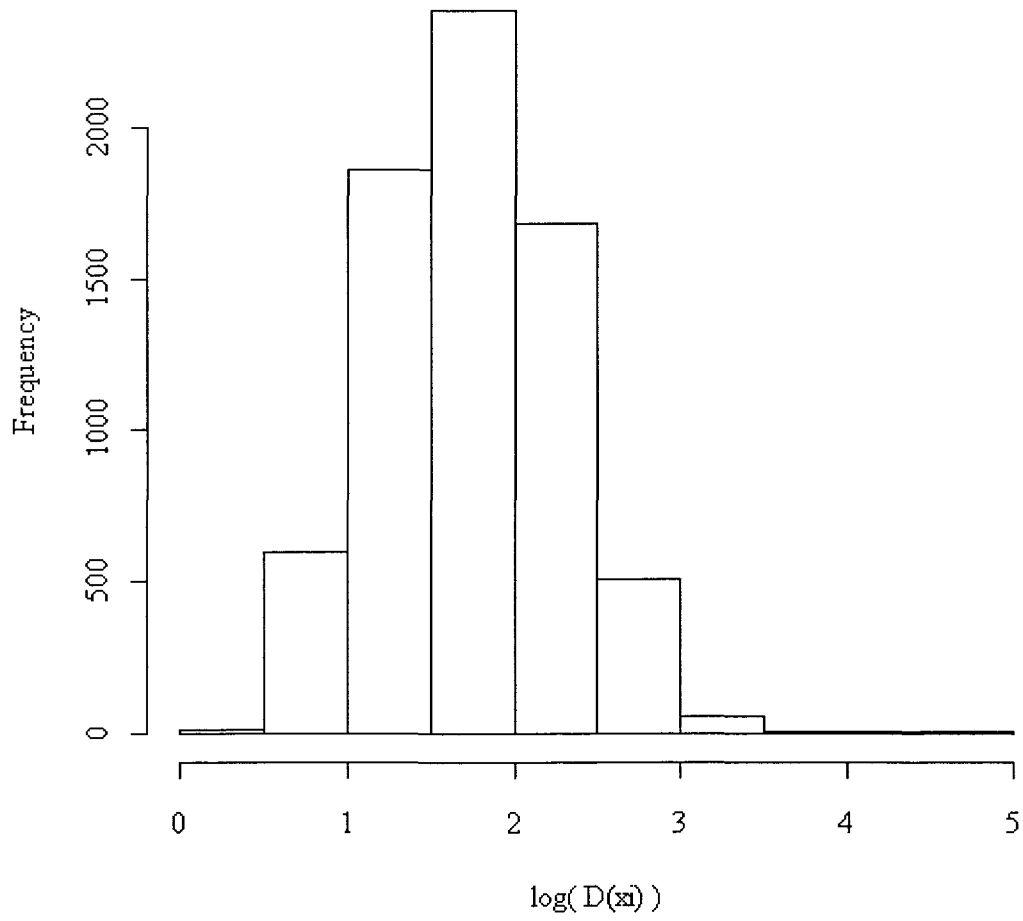


Figure 3.7 Failure Impact Histogram, System 2, LANL Data

Table 3.2 Inequality Coefficients, System 3, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
3	20th	58	5,736	0.6523049	0.5747176	0.1595362
3	40th	122	13,383	0.6474038	0.5700684	0.1510210
3	60th	184	19,382	0.6375234	0.5412378	0.1517705
3	80th	236	33,489	0.7124483	0.6315552	0.2135558
3	100th	299	40,386	0.6840931	0.5940145	0.1941182

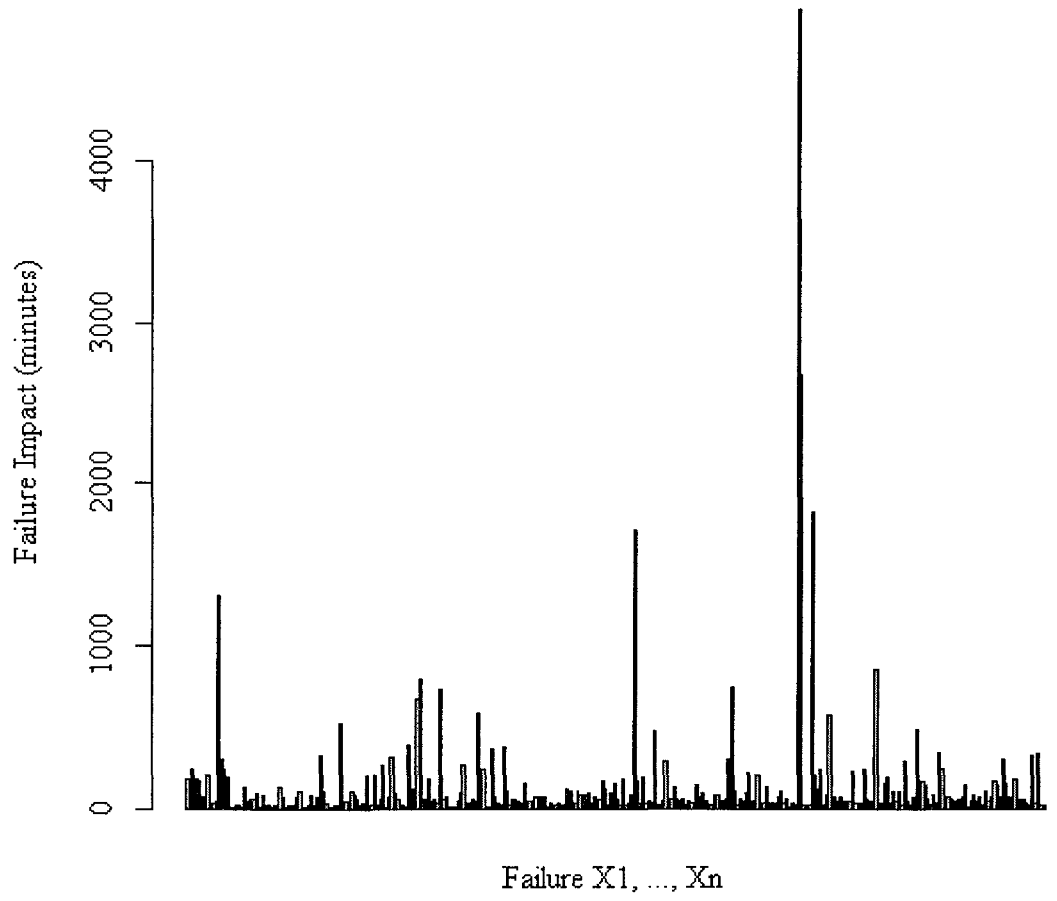


Figure 3.8 Impact of Each Failure, System 3, LANL Data

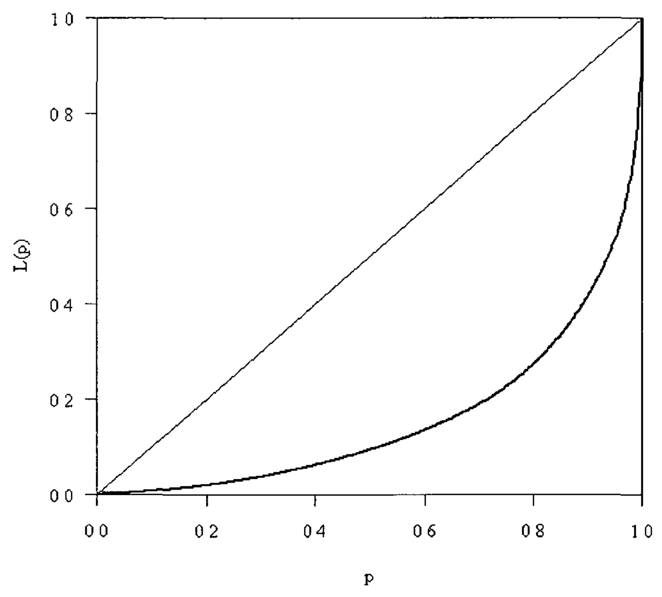


Figure 3.9 Lorenz Curve, System 3, LANL Data

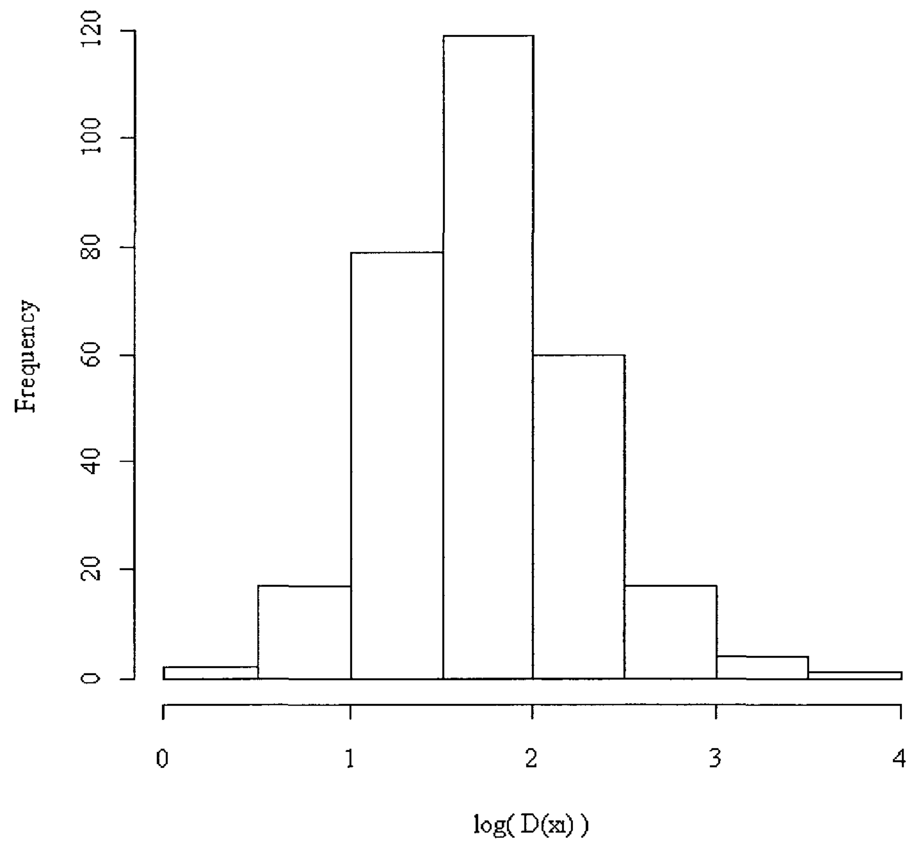


Figure 3.10 Failure Impact Histogram, System 3, LANL Data

Table 3.3 Inequality Coefficients, System 4, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
4	20th	63	5,759	0.5286301	0.4269491	0.09448456
4	40th	118	11,171	0.5802441	0.4862753	0.1187666
4	60th	190	18,896	0.5478803	0.4409907	0.1046655
4	80th	230	27,713	0.605446	0.5069202	0.1428101
4	100th	299	40,101	0.6151453	0.5224879	0.1475057

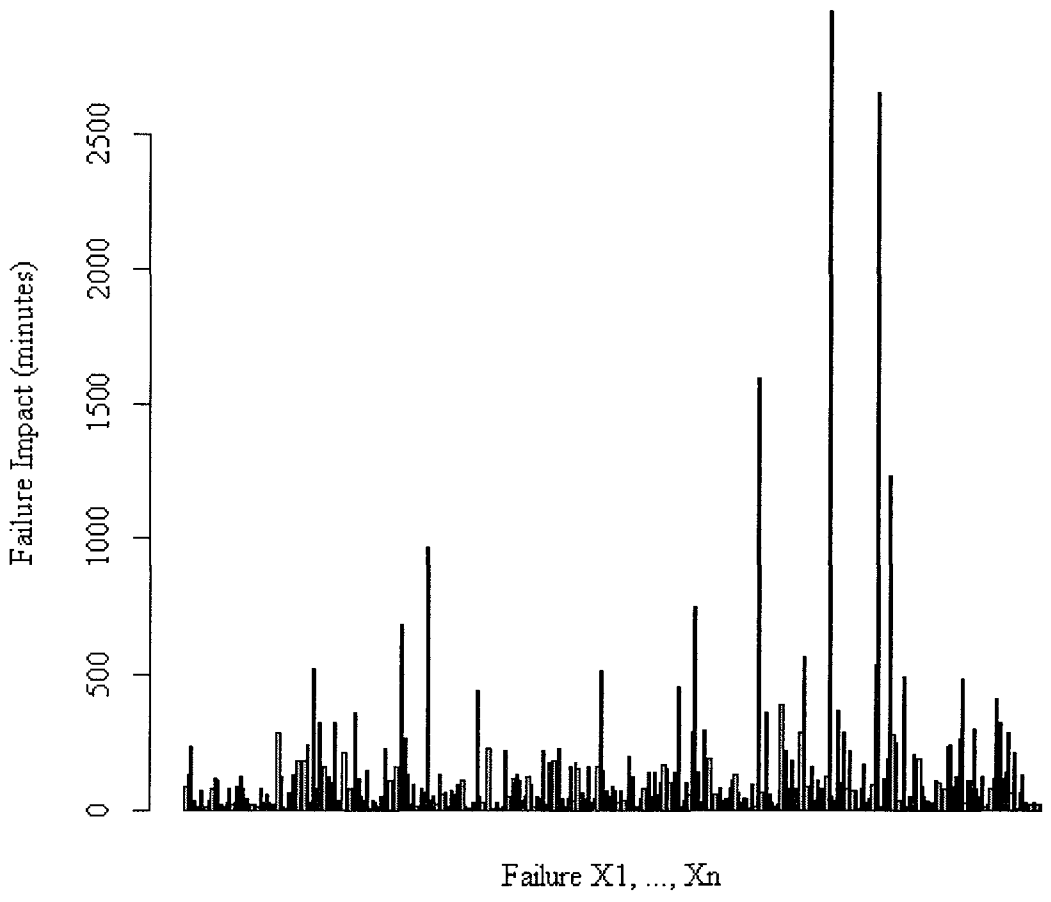


Figure 3.11 Impact of Each Failure, System 4, LANL Data

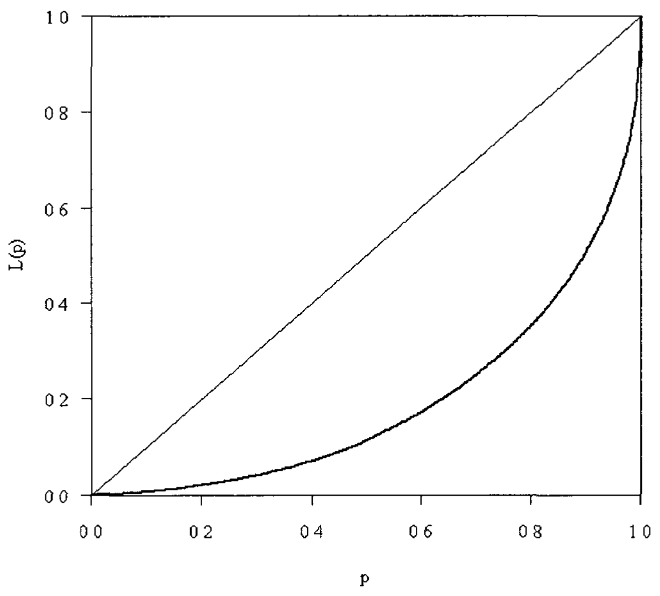


Figure 3.12 Lorenz Curve, System 4, LANL Data

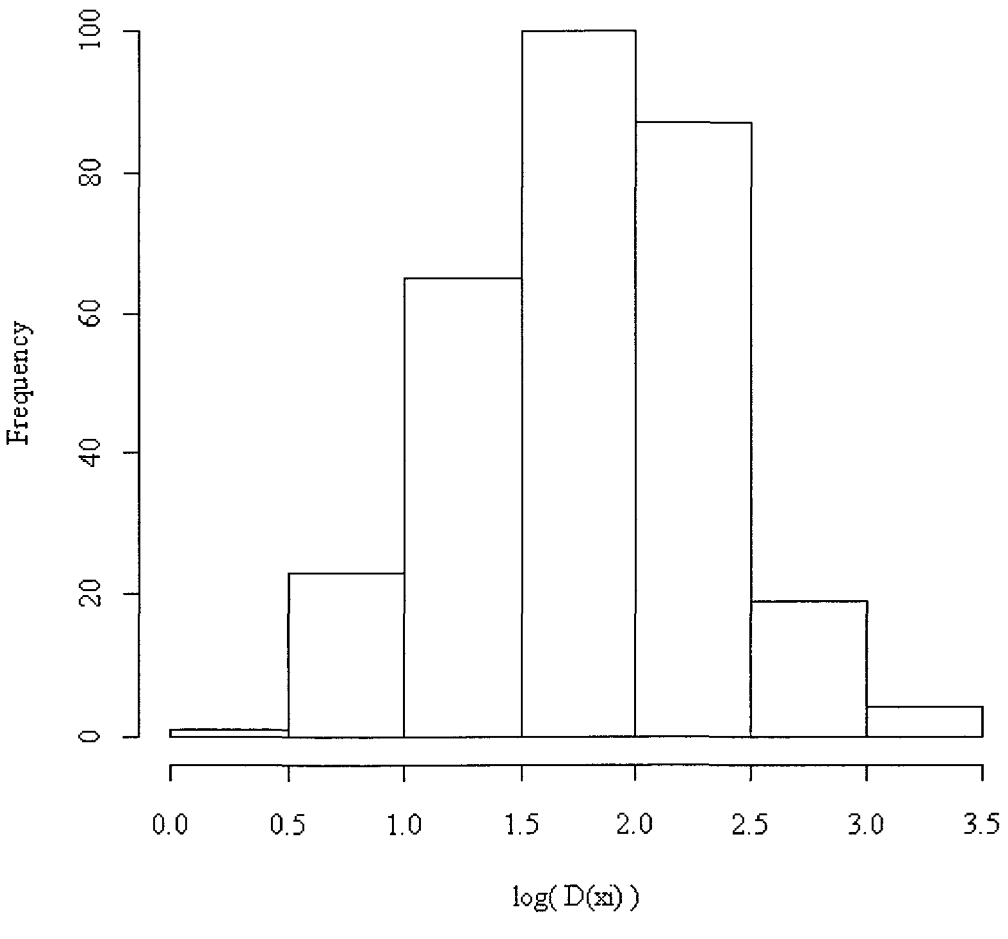


Figure 3.13 Failure Impact Histogram, System 4, LANL Data

Table 3.4 Inequality Coefficients, System 5, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
5	20th	59	10,683	0.6821483	0.5886865	0.2028295
5	40th	124	20,104	0.6740514	0.5828212	0.1817055
5	60th	181	26,212	0.6593811	0.5582976	0.1723399
5	80th	246	36,974	0.6702198	0.5747256	0.1787636
5	100th	305	44,641	0.6501178	0.550562	0.1651481

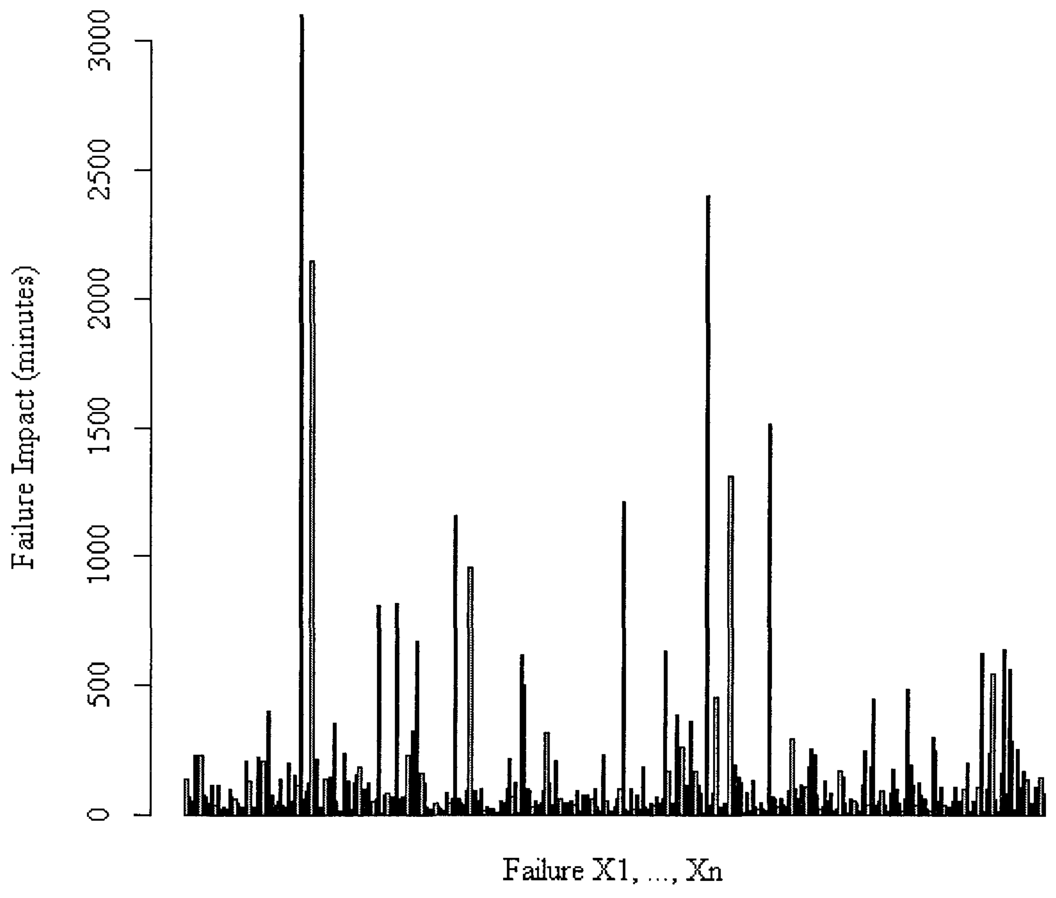


Figure 3.14 Impact of Each Failure, System 5, LANL Data

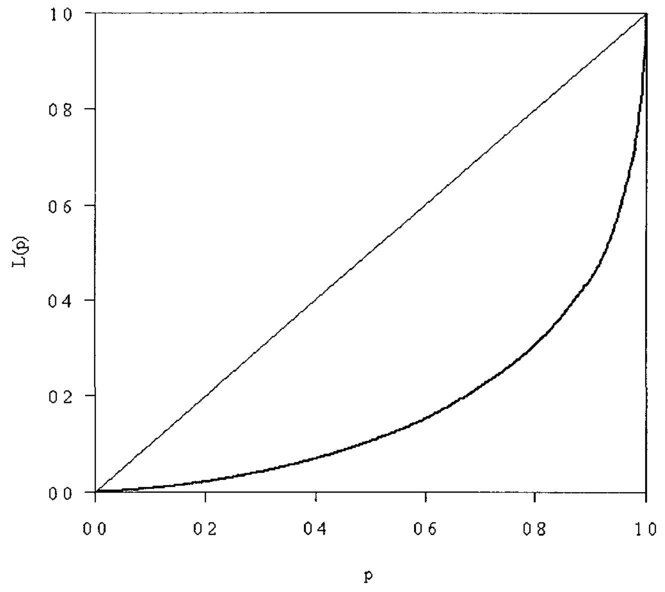


Figure 3.15 Lorenz Curve, System 5, LANL Data

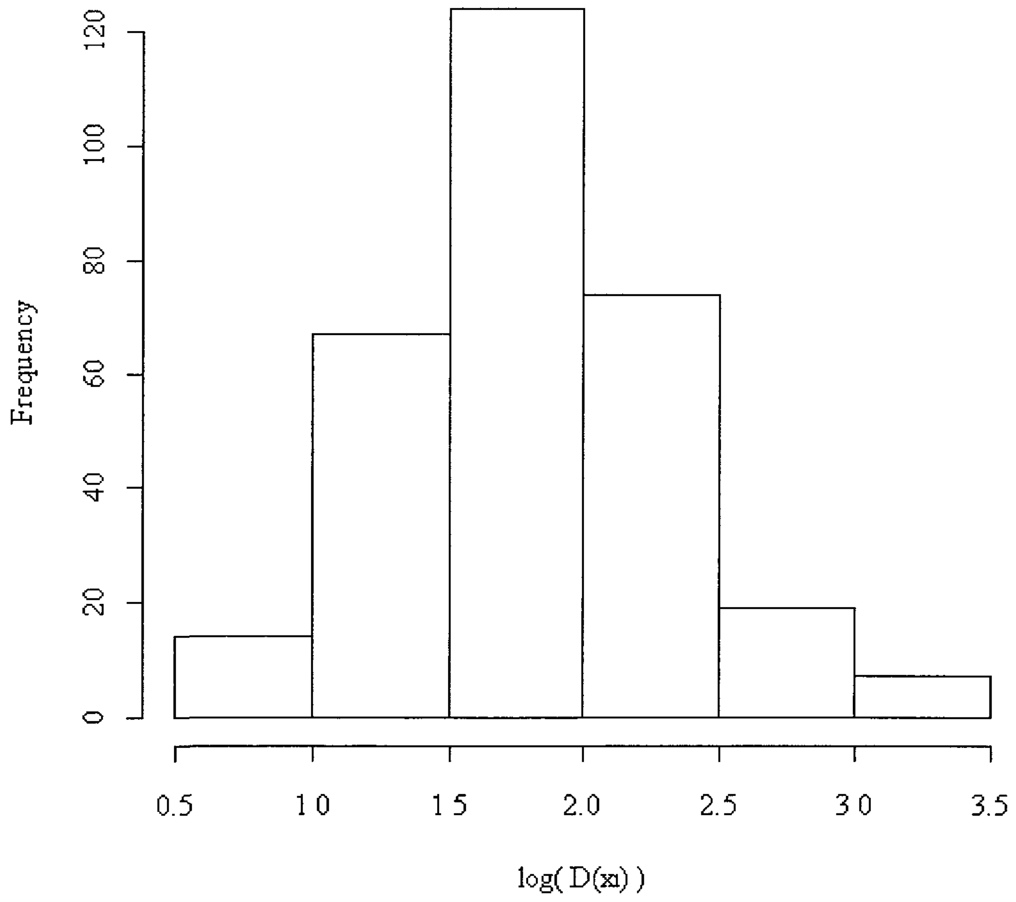


Figure 3.16 Failure Impact Histogram, System 5, LANL Data

Table 3.5 Inequality Coefficients, System 6, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
6	20th	17	2,607	0.6746124	0.6037954	0.1850318
6	40th	27	3,806	0.676484	0.5857101	0.1882174
6	60th	35	4,302	0.6902305	0.6058226	0.1843178
6	80th	45	4,633	0.676087	0.5748184	0.1837073
6	100th	64	7,178	0.6535421	0.5566991	0.1592298

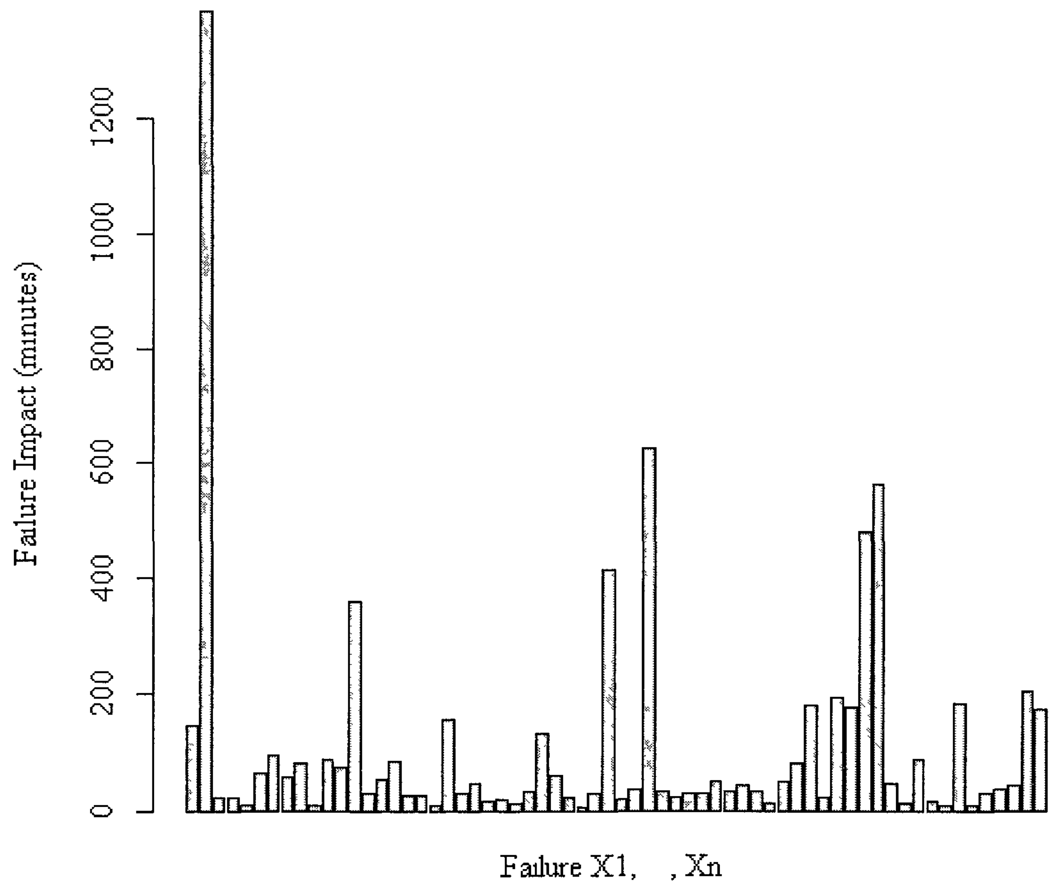


Figure 3.17 Impact of Each Failure, System 6, LANL Data

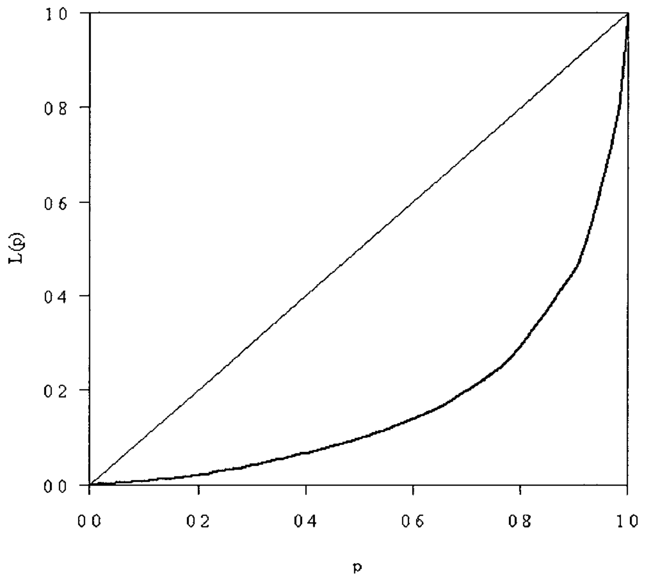


Figure 3.18 Lorenz Curve, System 6, LANL Data

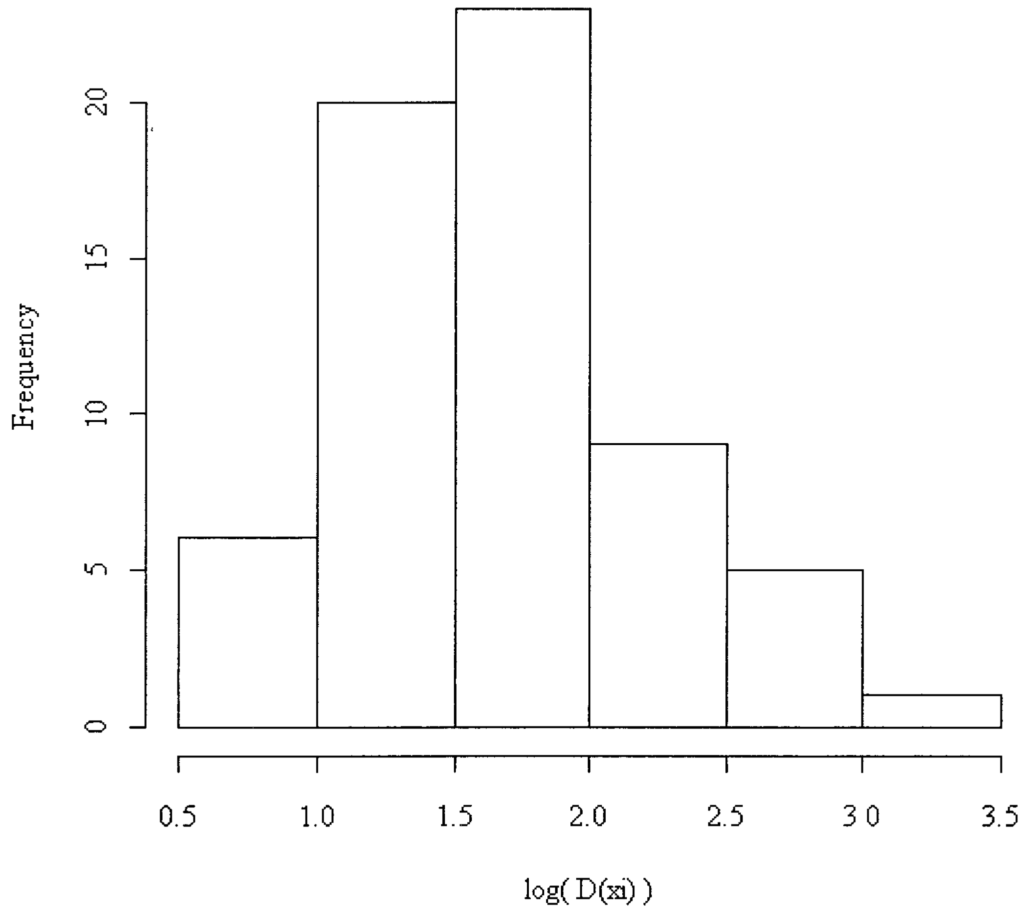


Figure 3.19 Failure Impact Histogram, System 6, LANL Data

Table 3.6 Inequality Coefficients, System 7, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
7	20th	25	2,902	0.2843005	0.1302336	0.03110032
7	40th	54	7,430	0.4077613	0.2752189	0.05774089
7	60th	77	12,075	0.4426404	0.3044237	0.0680330
7	80th	104	18,002	0.4624144	0.3183379	0.0761484
7	100th	129	34,323	0.6428636	0.5383517	0.1940597

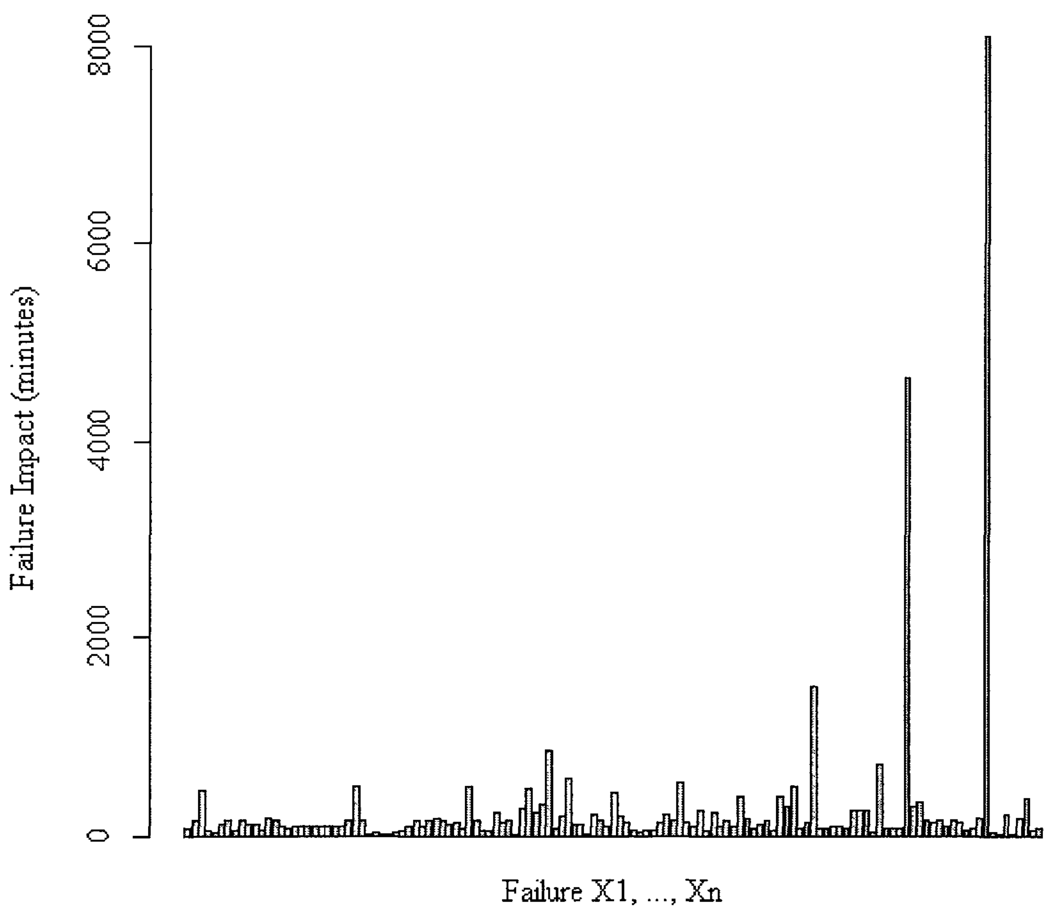


Figure 3.20 Impact of Each Failure, System 7, LANL Data

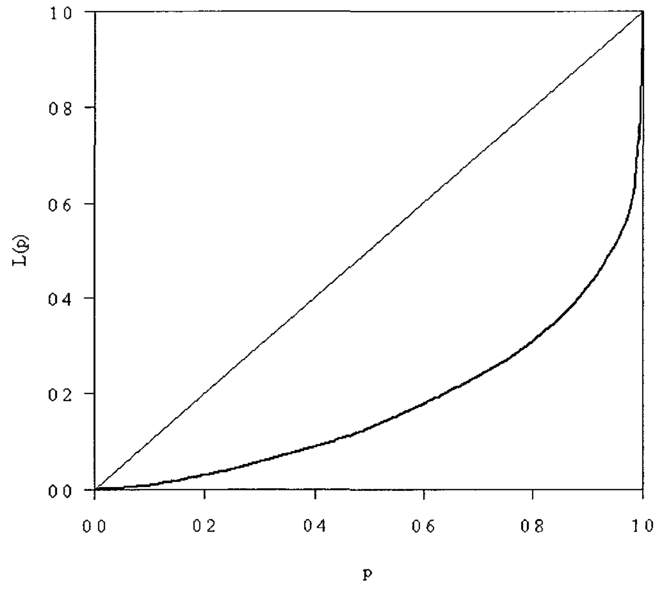


Figure 3.21 Lorenz Curve, System 7, LANL Data

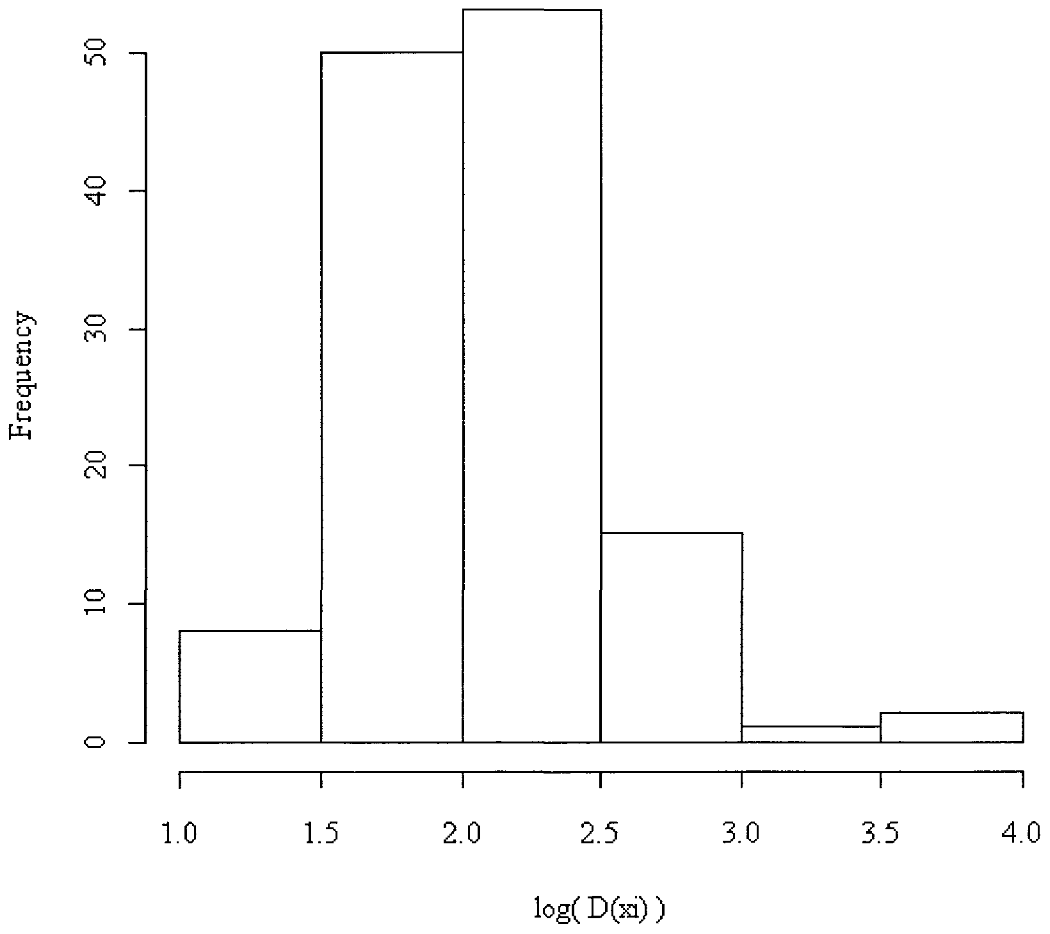


Figure 3.22 Failure Impact Histogram, System 7, LANL Data

Table 3.7 Inequality Coefficients, System 8, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
8	20th	89	208,409	0.8794655	0.9185997	0.3480385
8	40th	190	328,866	0.8680784	0.9003143	0.3357564
8	60th	285	499,239	0.8493329	0.8862067	0.3141345
8	80th	386	836,041	0.8240501	0.8720414	0.2832965
8	100th	475	977,672	0.8083147	0.8593525	0.2694886

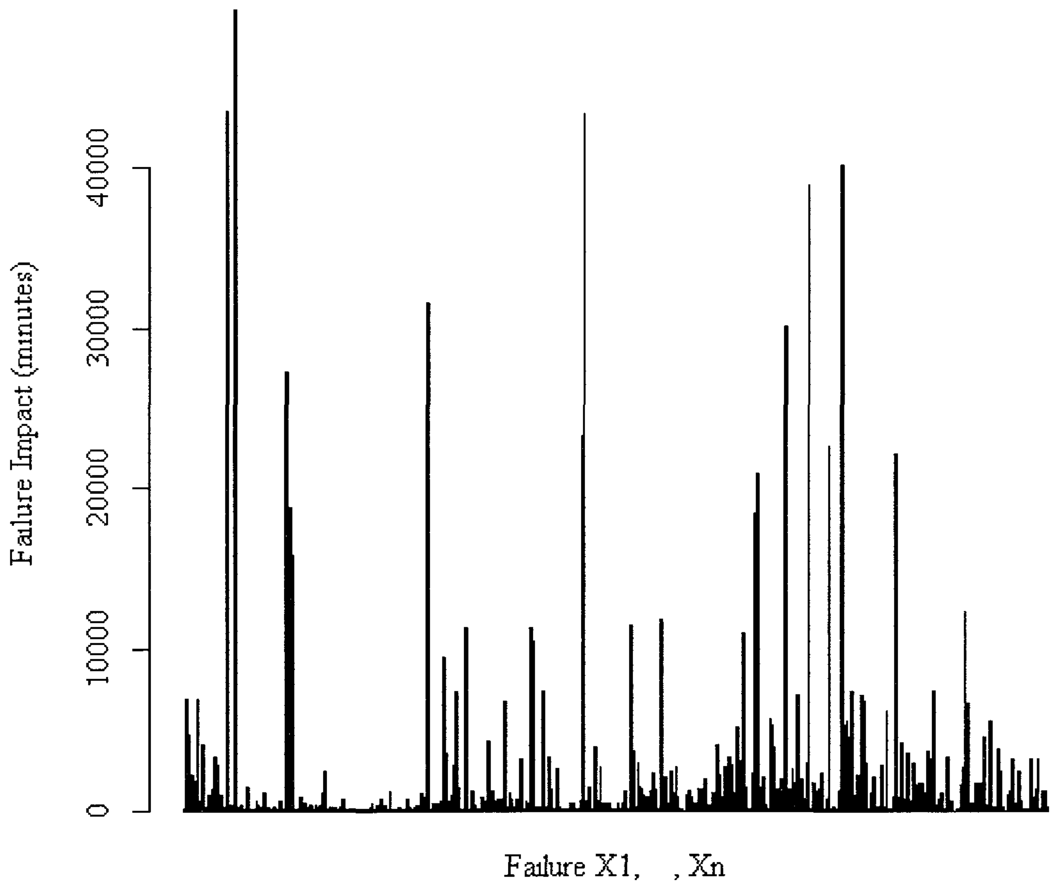


Figure 3.23 Impact of Each Failure, System 8, LANL Data

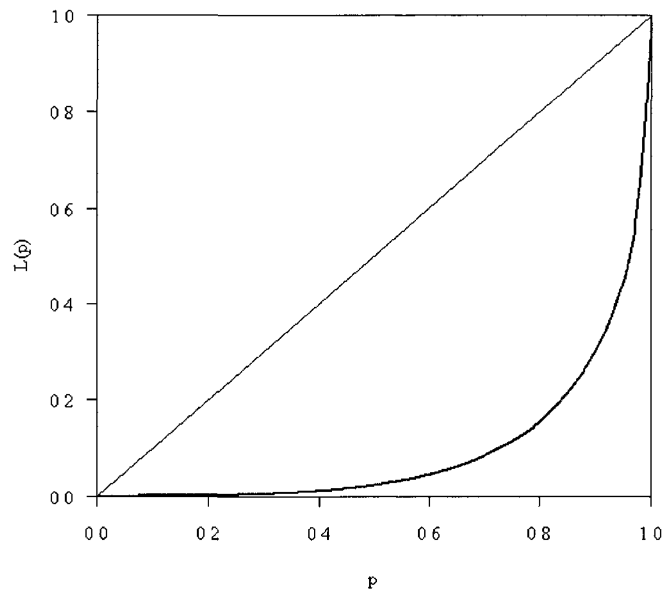


Figure 3.24 Lorenz Curve, System 8, LANL Data

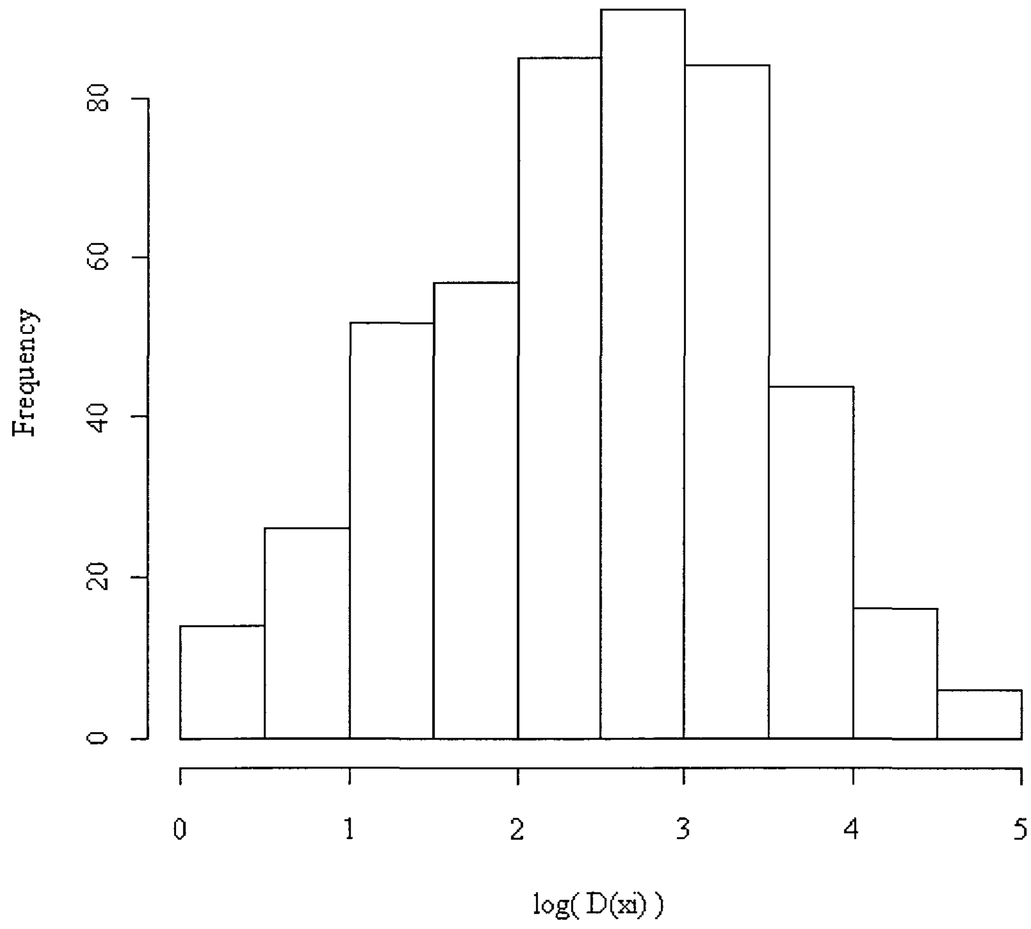


Figure 3.25 Failure Impact Histogram, System 8, LANL Data

Table 3.8 Inequality Coefficients, System 9, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
9	20th	56	249,154	0.9172414	0.9878632	0.4167057
9	40th	112	266,823	0.9346124	0.9999993	0.4590362
9	60th	168	274,393	0.9462083	1	0.490213
9	80th	224	281,007	0.9510054	1	0.5069014
9	100th	280	290,964	0.9506572	1	0.5108189

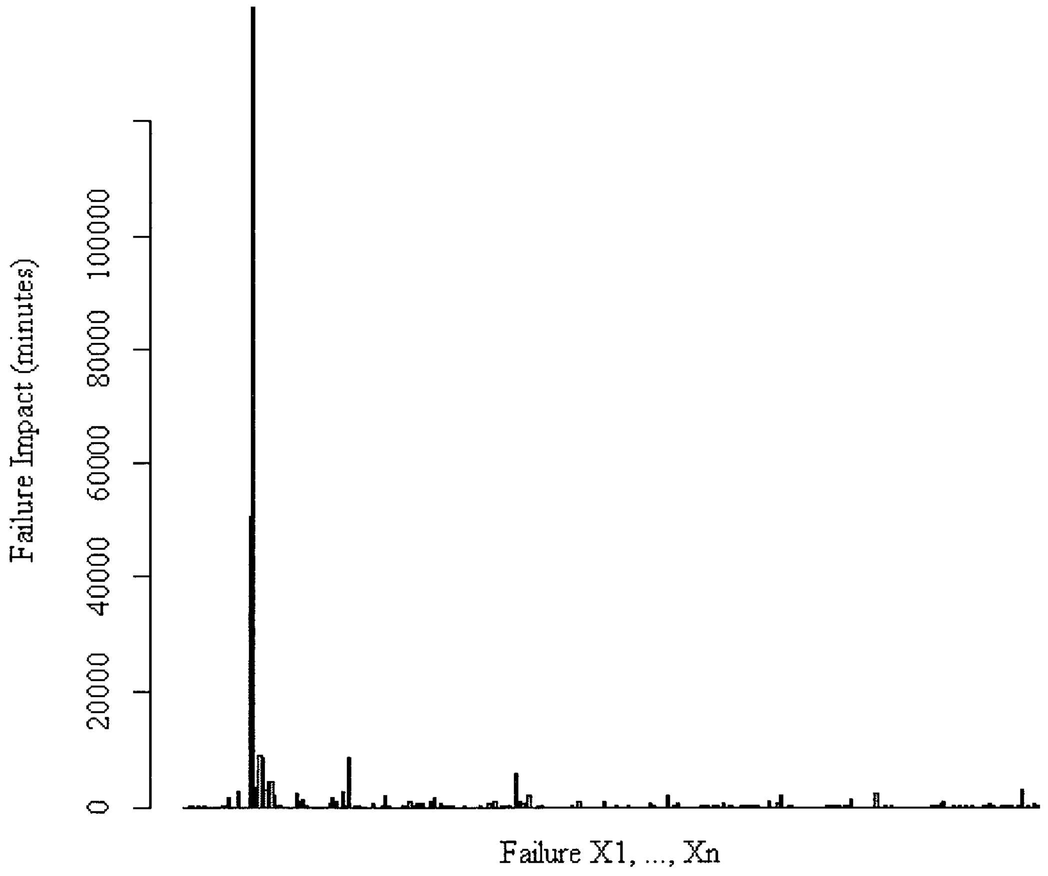


Figure 3.26 Impact of Each Failure, System 9, LANL Data

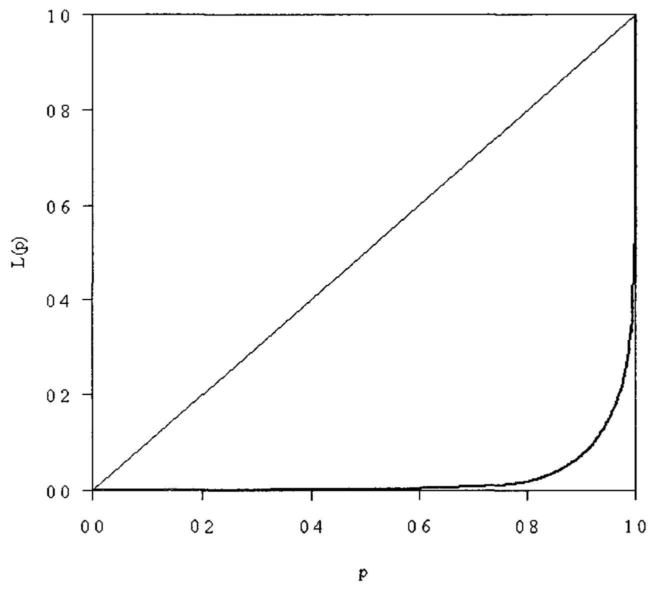


Figure 3.27 Lorenz Curve, System 9, LANL Data

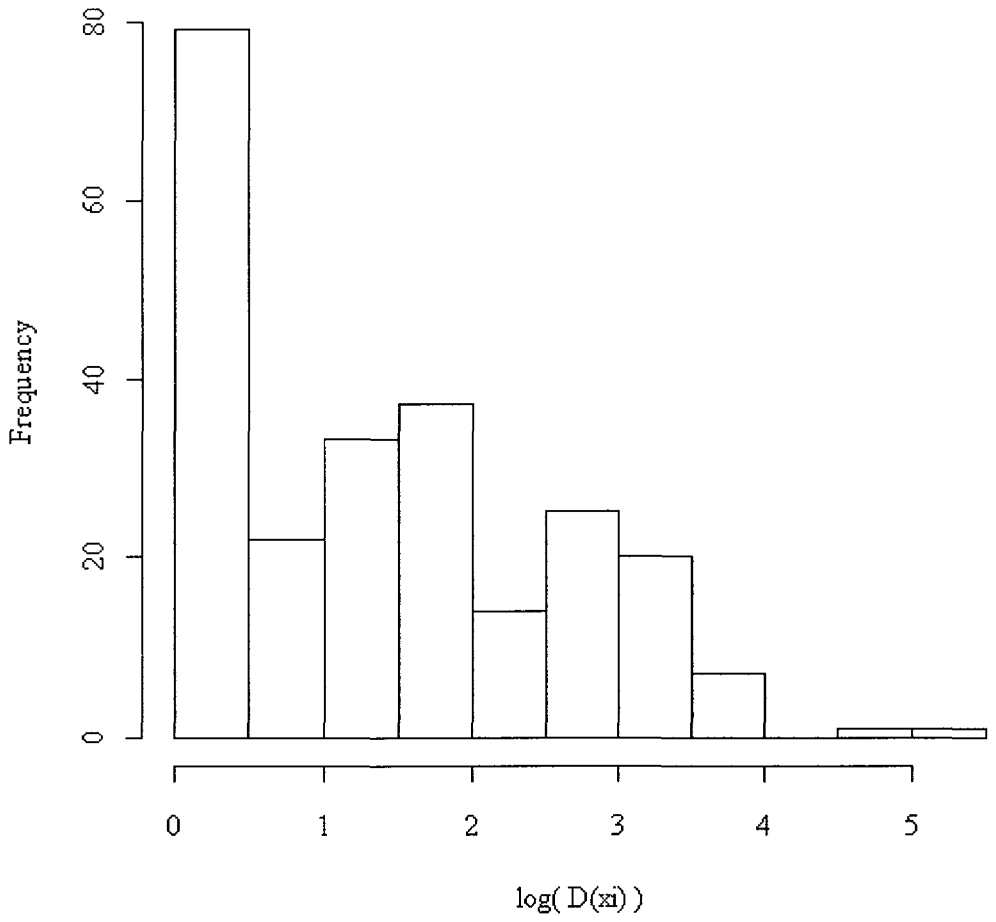


Figure 3.28 Failure Impact Histogram, System 9, LANL Data

Table 3.9 Inequality Coefficients, System 10, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
10	20th	47	269,867	0.868898	0.9822825	0.3491037
10	40th	94	285,196	0.9093874	0.999917	0.4094456
10	60th	141	300,167	0.9231305	0.9999997	0.4360462
10	80th	190	389,218	0.9141034	0.9999993	0.4118522
10	100th	237	392,114	0.926879	0.9999999	0.4352492

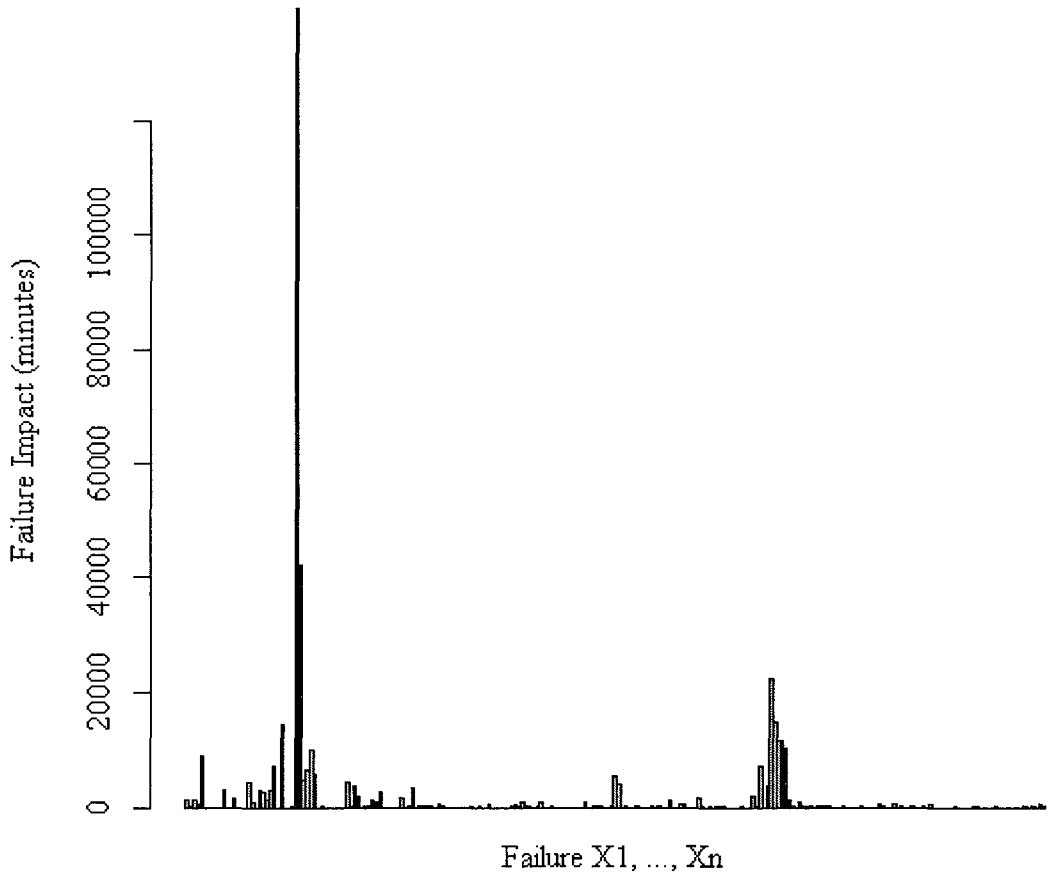


Figure 3.29 Impact of Each Failure, System 10, LANL Data

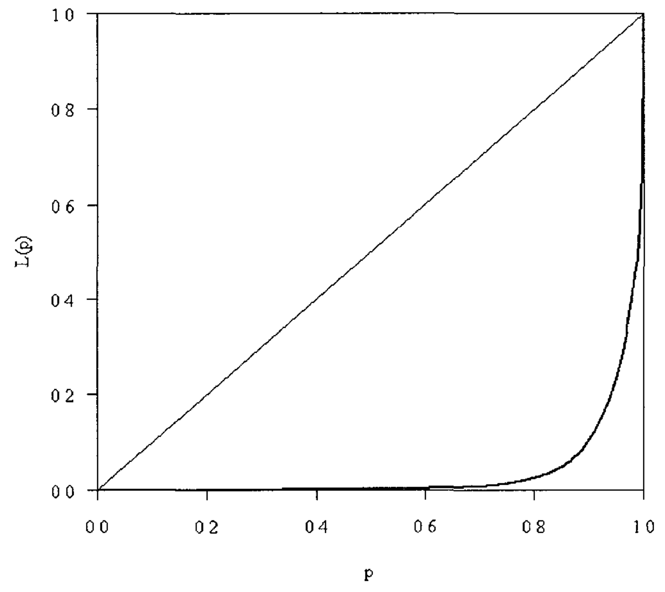


Figure 3.30 Lorenz Curve, System 10, LANL Data

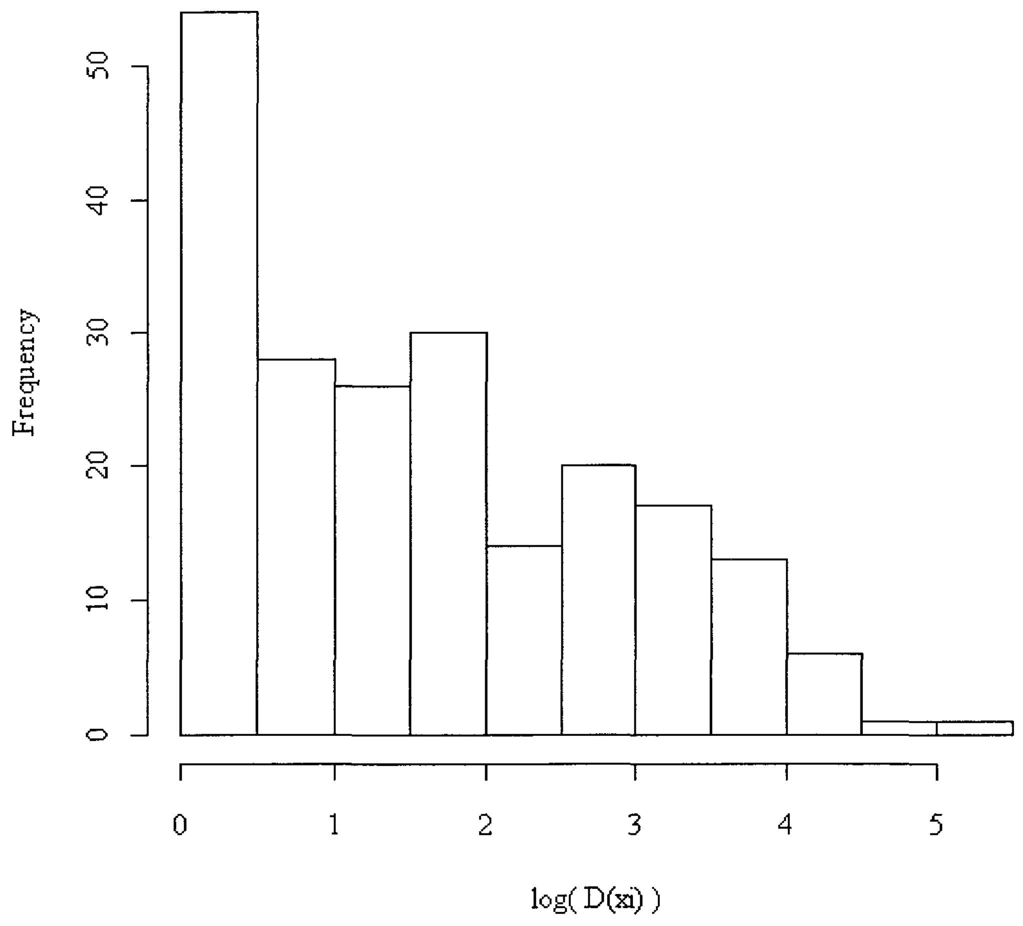


Figure 3.31 Failure Impact Histogram, System 10, LANL Data

Table 3.10 Inequality Coefficients, System 11, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
11	20th	53	356,159	0.901521	0.9824175	0.3898342
11	40th	109	374,986	0.9318934	0.999996	0.4497052
11	60th	159	406,813	0.9343652	1	0.459456
11	80th	218	412998	0.9450112	1	0.4855938
11	100th	268	418,993	0.9494329	1	0.4992231

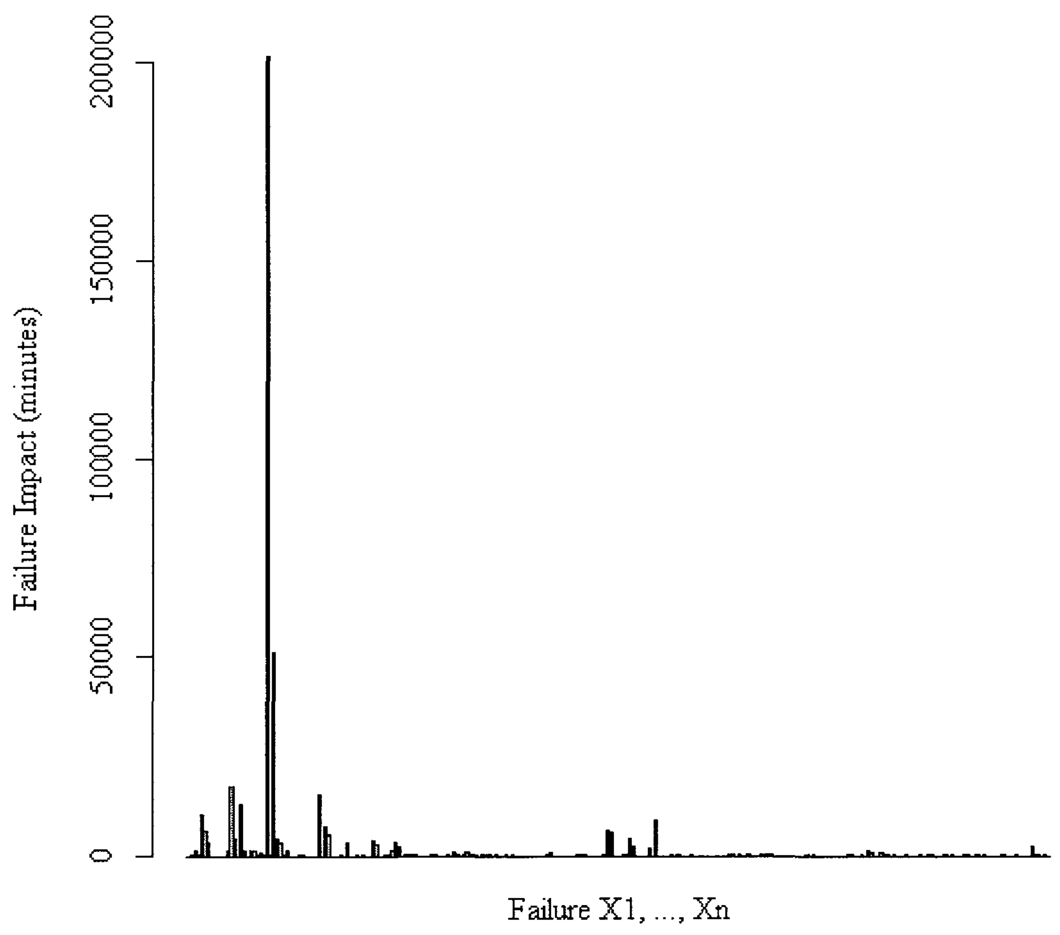


Figure 3.32 Impact of Each Failure, System 11, LANL Data

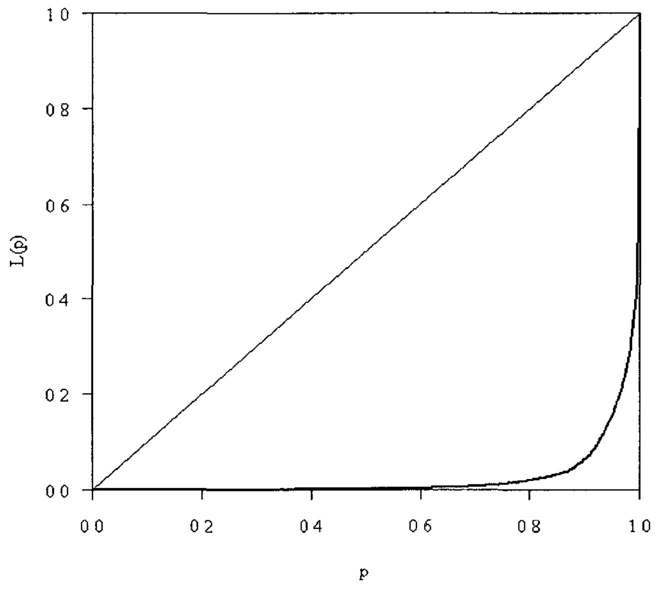


Figure 3.33 Lorenz Curve, System 11, LANL Data

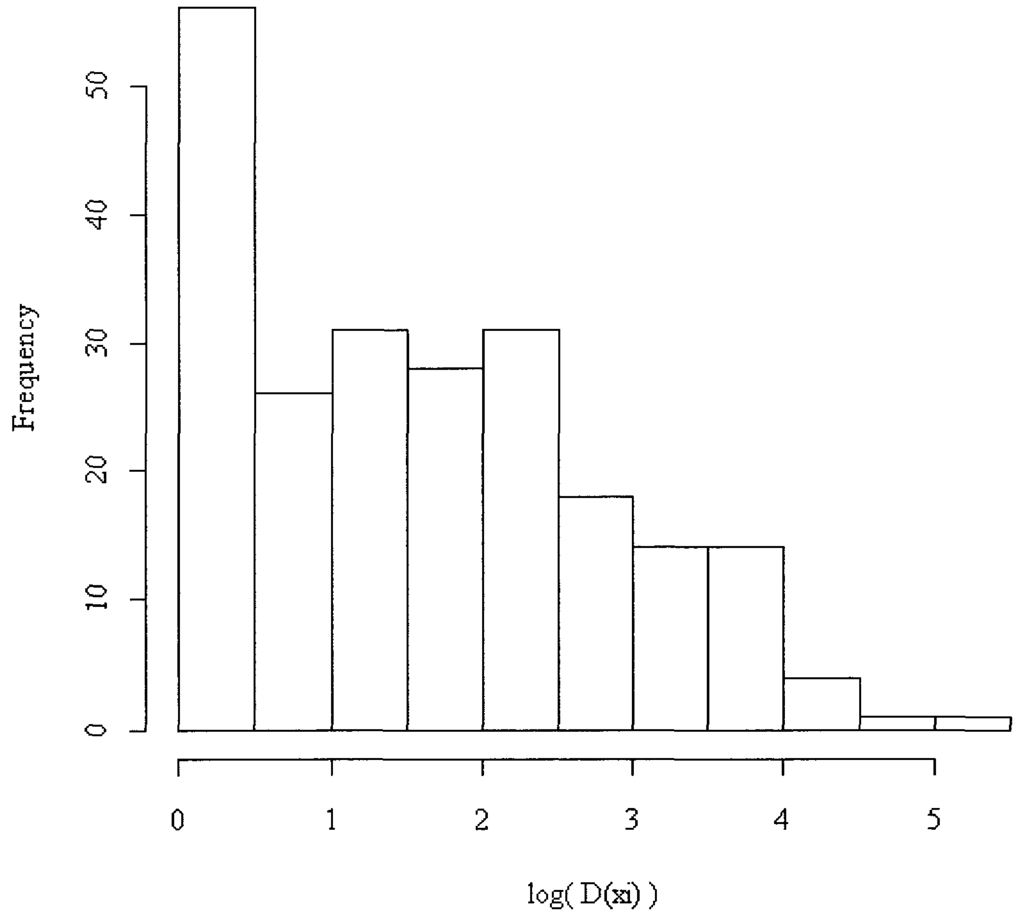


Figure 3.34 Failure Impact Histogram, System 11, LANL Data

Table 3.11 Inequality Coefficients, System 12, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
12	20th	51	242,749	0.9010614	0.9691547	0.4003552
12	40th	107	254,317	0.9353616	1	0.4663913
12	60th	153	283,519	0.9339544	1	0.4650438
12	80th	204	292,791	0.938812	1	0.4790362
12	100th	259	299,825	0.9407131	1	0.488347

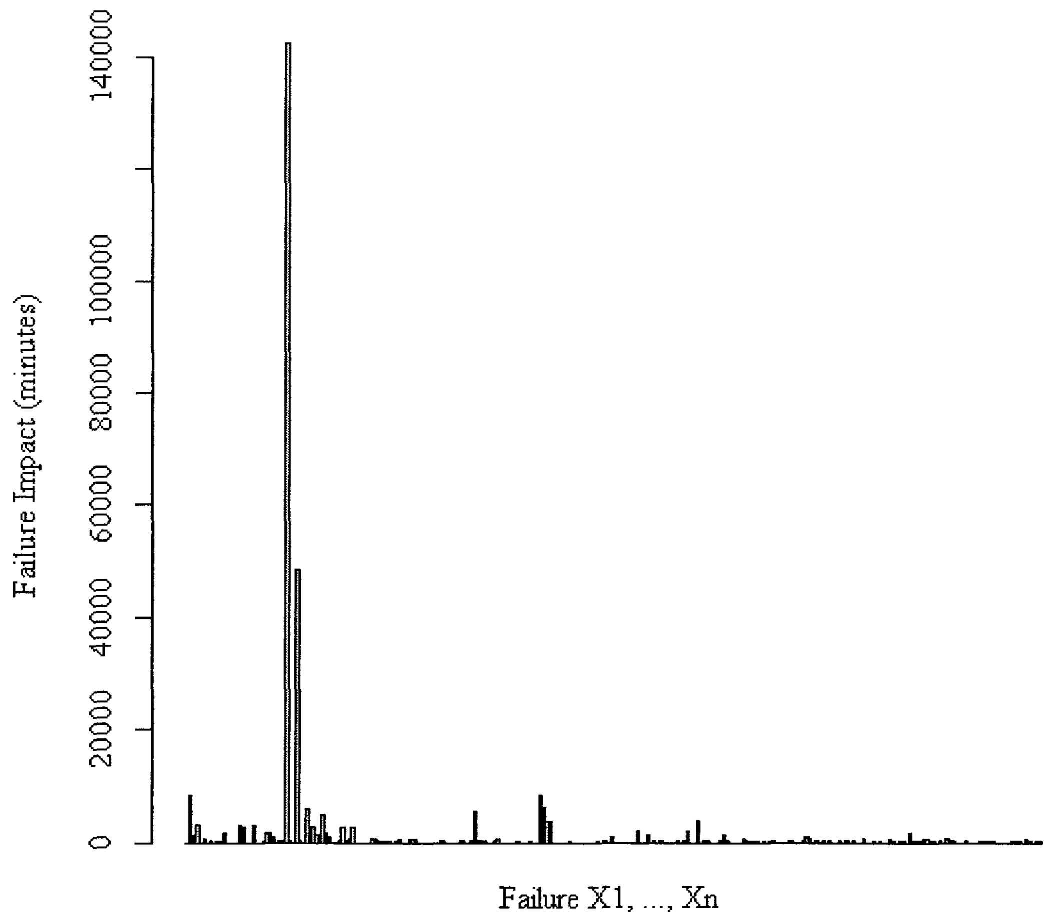


Figure 3.35 Impact of Each Failure, System 12, LANL Data

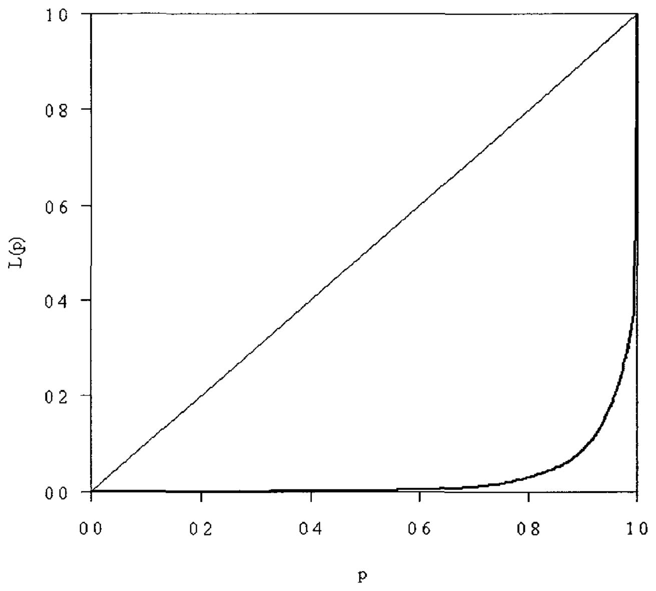


Figure 3.36 Lorenz Curve, System 12, LANL Data

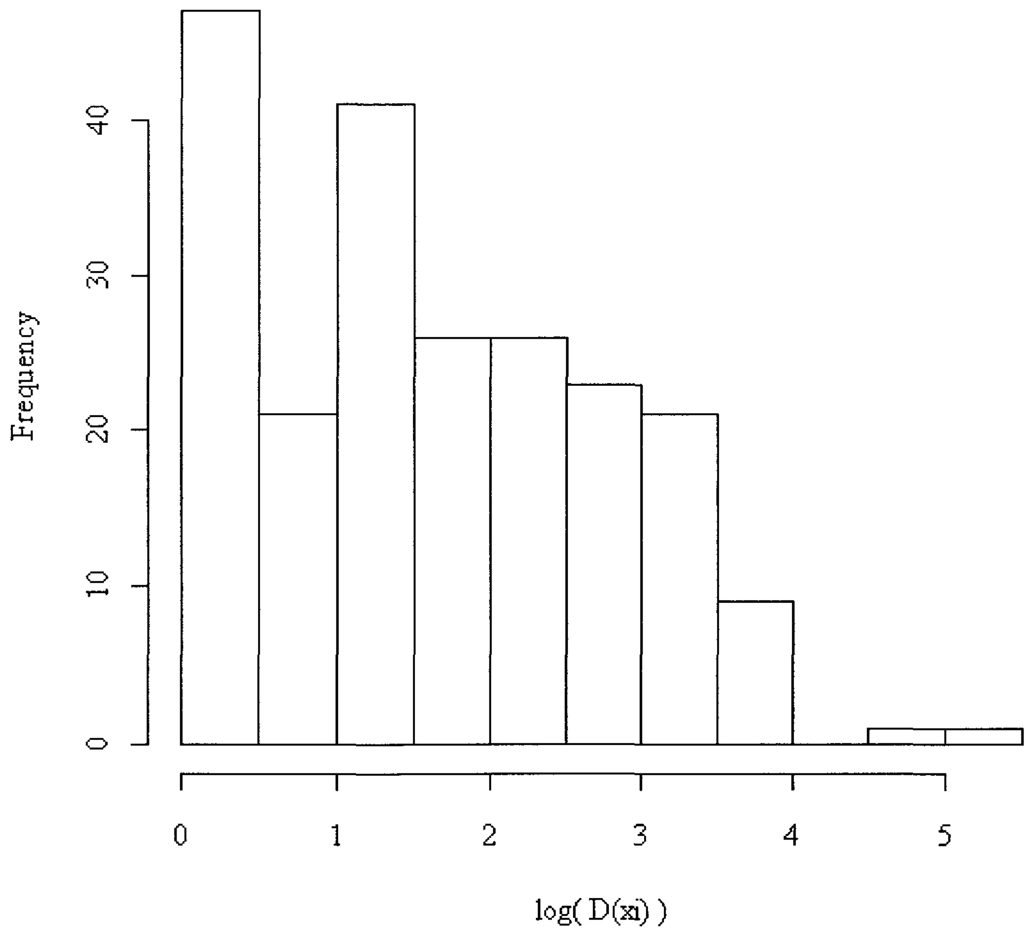


Figure 3.37 Failure Impact Histogram, System 12, LANL Data

Table 3.12 Inequality Coefficients, System 13, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
13	20th	39	359,810	0.8414183	0.9467759	0.3249069
13	40th	81	379,558	0.8931308	0.973137	0.3926667
13	60th	115	394,761	0.9074928	0.9999994	0.4176841
13	80th	159	403,719	0.9239984	1	0.4471762
13	100th	201	441,075	0.9494189	1	0.4596139

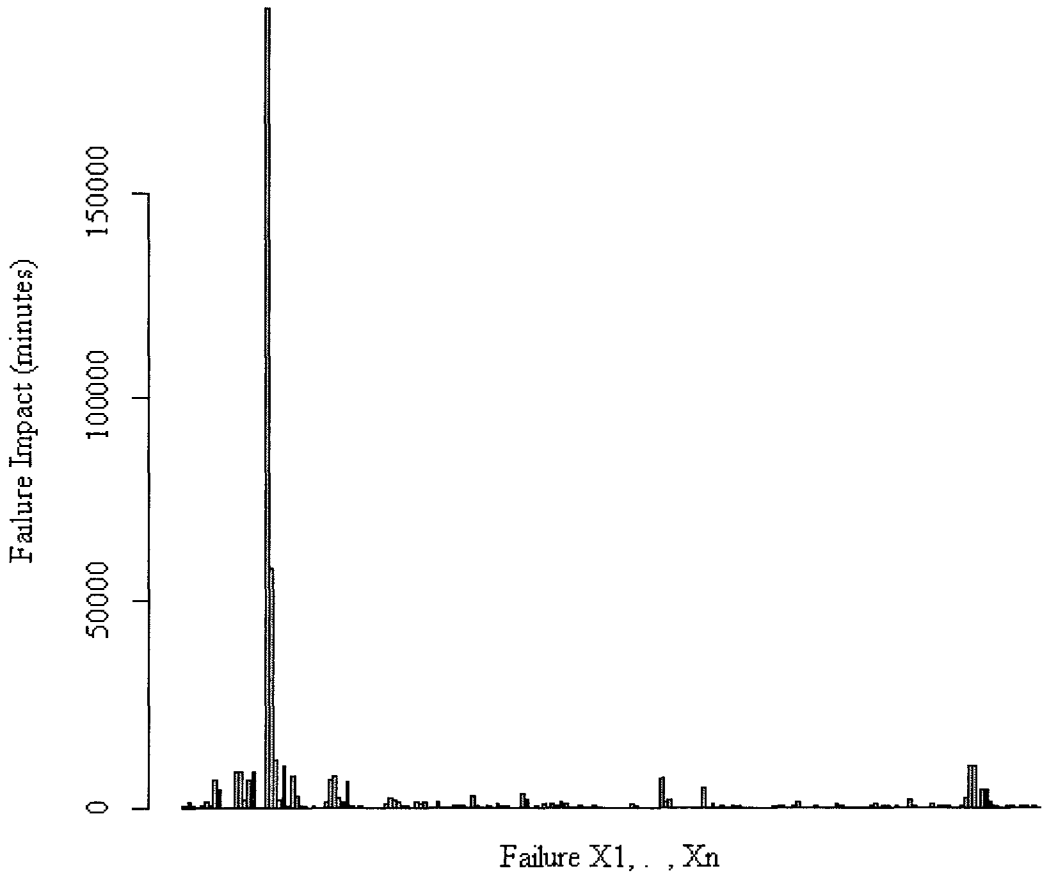


Figure 3.38 Impact of Each Failure, System 13, LANL Data

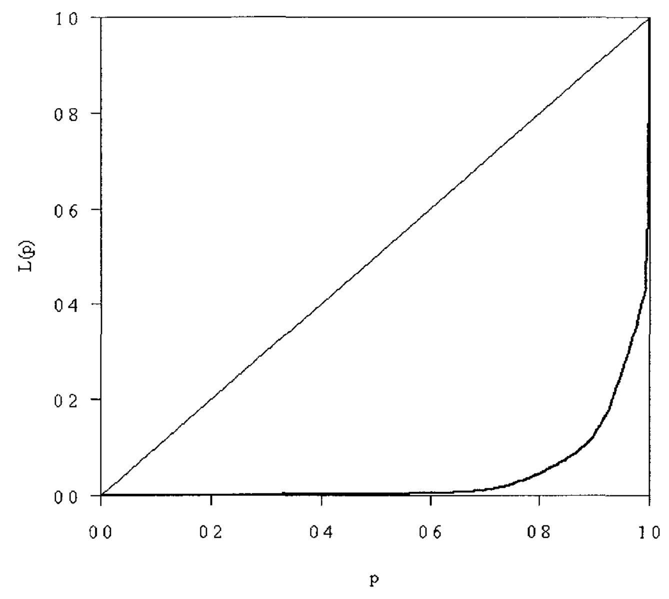


Figure 3.39 Lorenz Curve, System 13, LANL Data

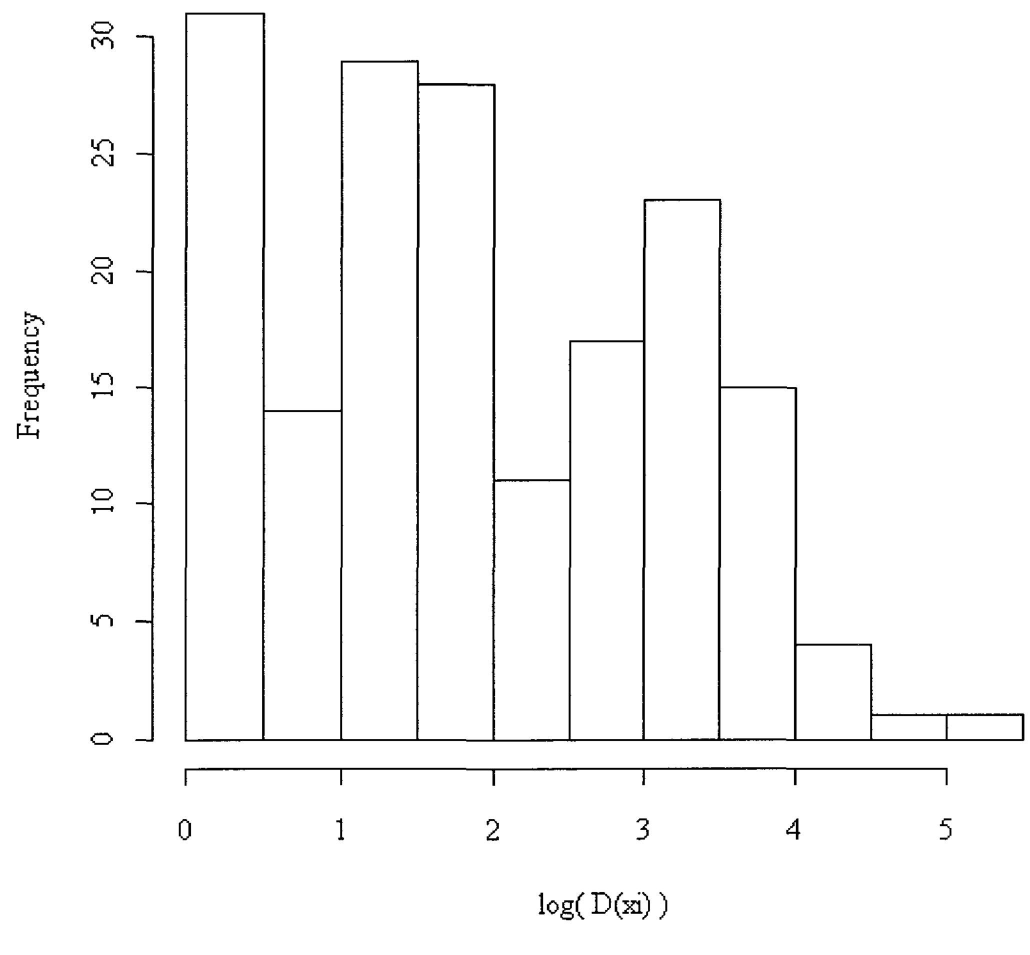


Figure 3.40 Failure Impact Histogram, System 13, LANL Data

Table 3.13 Inequality Coefficients, System 14, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
14	20th	24	29,061	0.829304	0.9784576	0.3094202
14	40th	56	697,322	0.9253138	0.9942794	0.4290339
14	60th	80	704,071	0.942445	0.9948515	0.4664413
14	80th	102	728,867	0.9469915	0.997793	0.4771013
14	100th	125	738,907	0.9514942	0.9993597	0.4909886

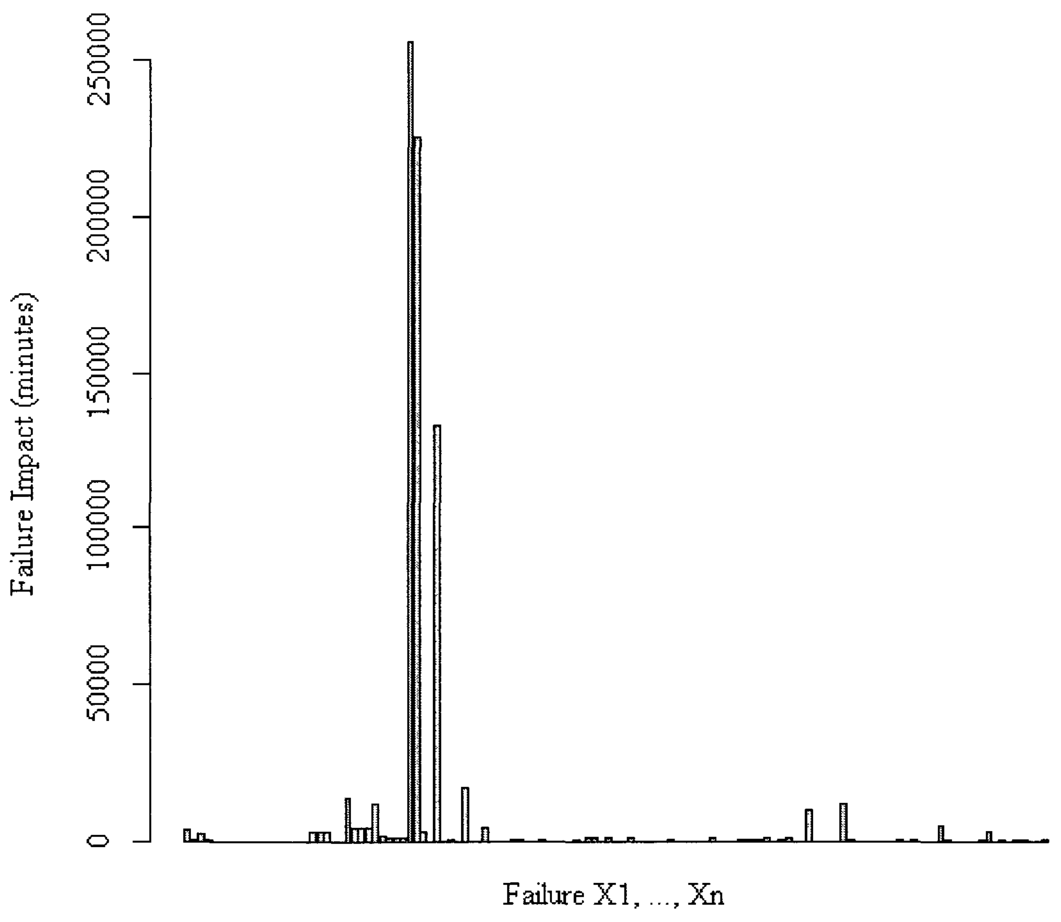


Figure 3.41 Impact of Each Failure, System 14, LANL Data

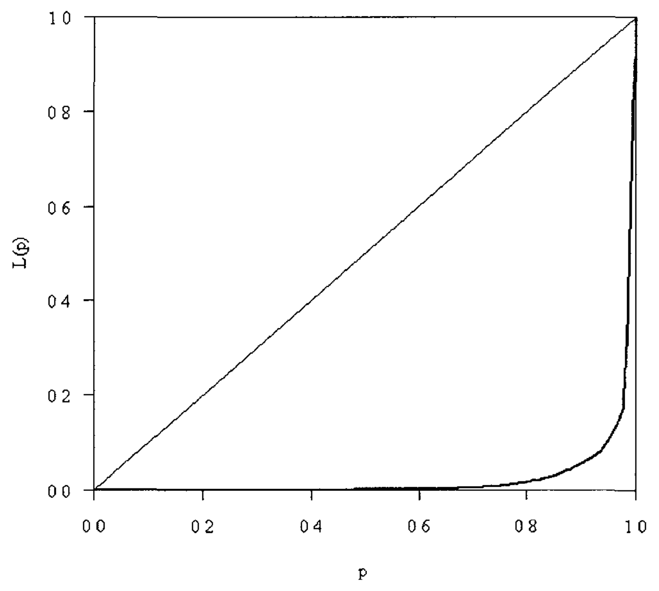


Figure 3.42 Lorenz Curve, System 14, LANL Data

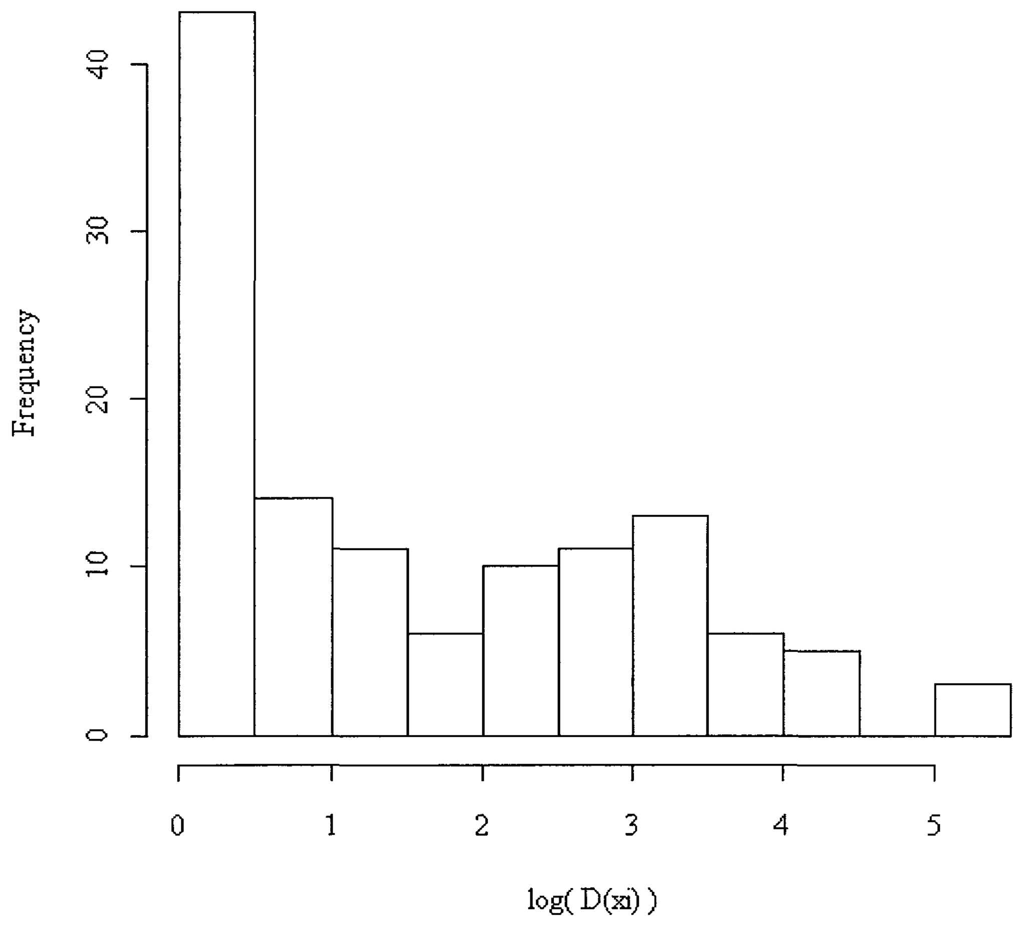


Figure 3.43 Failure Impact Histogram, System 14, LANL Data

Table 3.14 Inequality Coefficients, System 15, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
15	20th	11	391	0.5607998	0.4344859	0.1257257
15	40th	28	4,579	0.7633997	0.7228064	0.2411162
15	60th	36	5,118	0.7457014	0.6893286	0.2306536
15	80th	44	6,046	0.724897	0.6711294	0.2138146
15	100th	54	7,470	0.7406217	0.6840907	0.2201631

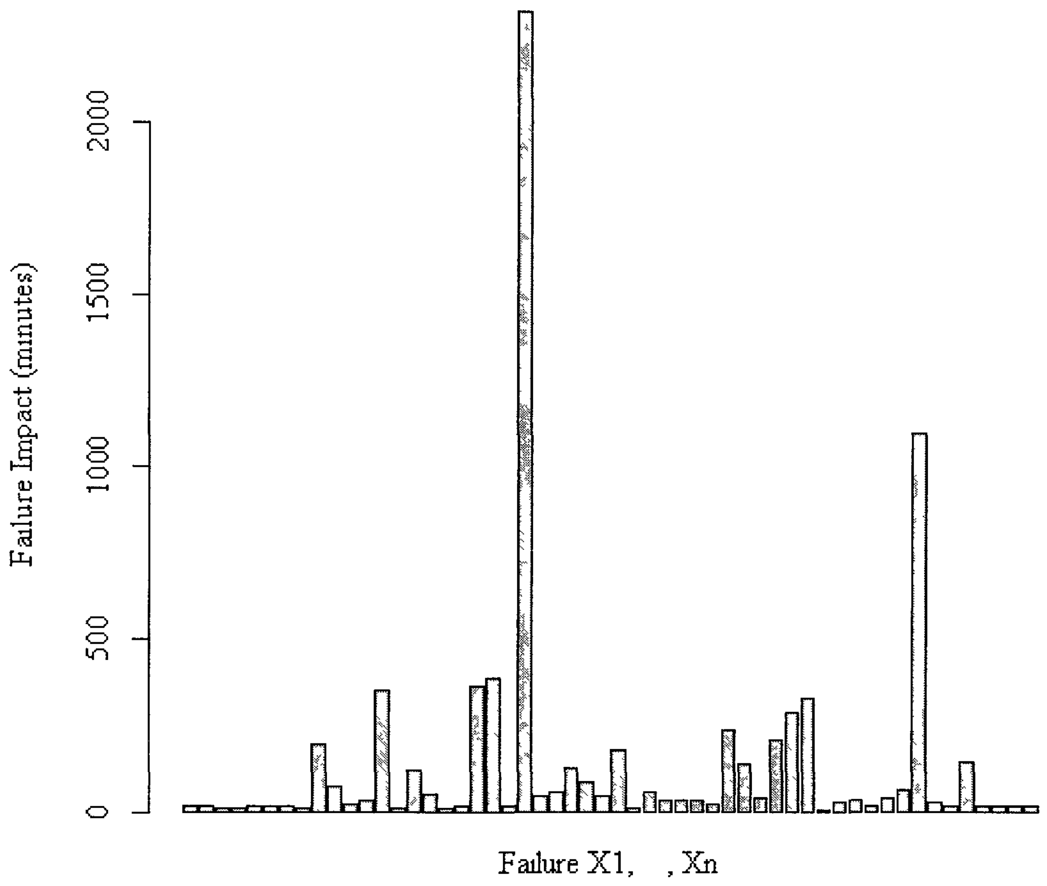


Figure 3.44 Impact of Each Failure, System 15, LANL Data

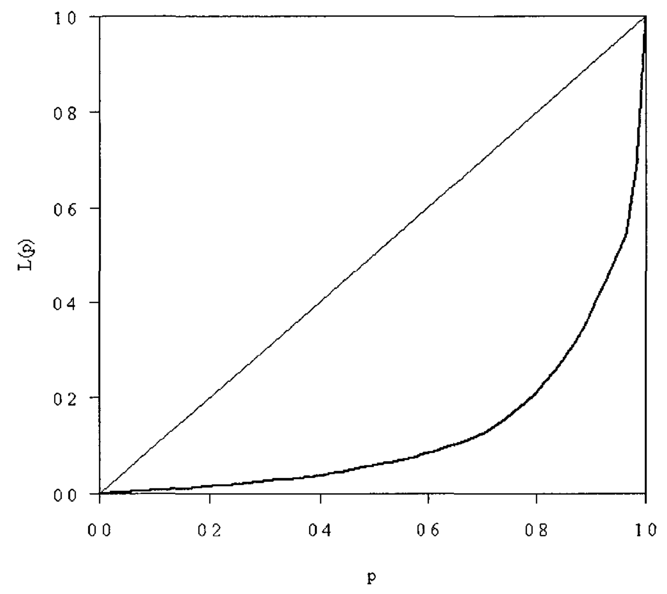


Figure 3.45 Lorenz Curve, System 15, LANL Data

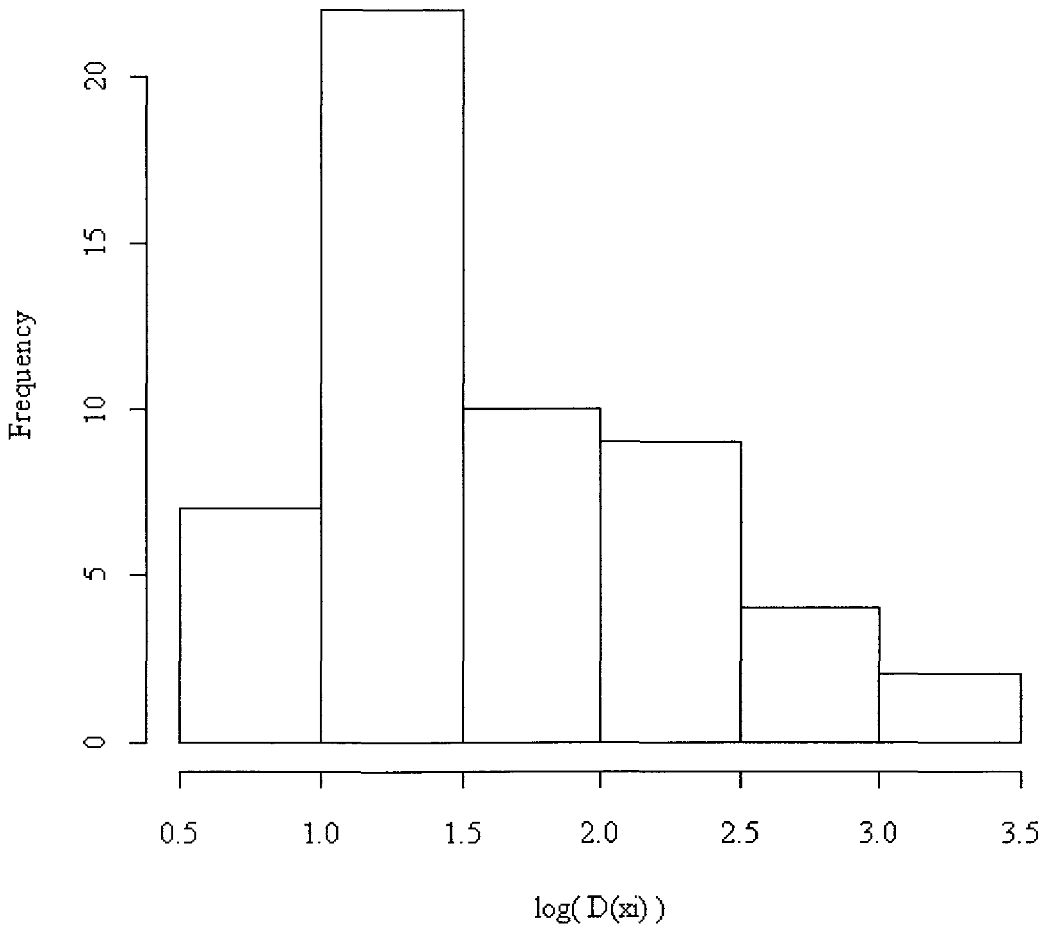


Figure 3.46 Failure Impact Histogram, System 15, LANL Data

Table 3.15 Inequality Coefficients, System 16, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
16	20th	540	61,494	0.700608	0.6923546	0.2129928
16	40th	1,080	149,227	0.7370584	0.7315113	0.2465563
16	60th	1,614	664,537	0.9022084	0.8946175	0.5427902
16	80th	2,147	1,219,242	0.9317911	0.924417	0.5807508
16	100th	2,680	1,445,023	0.9215625	0.911852	0.5371212

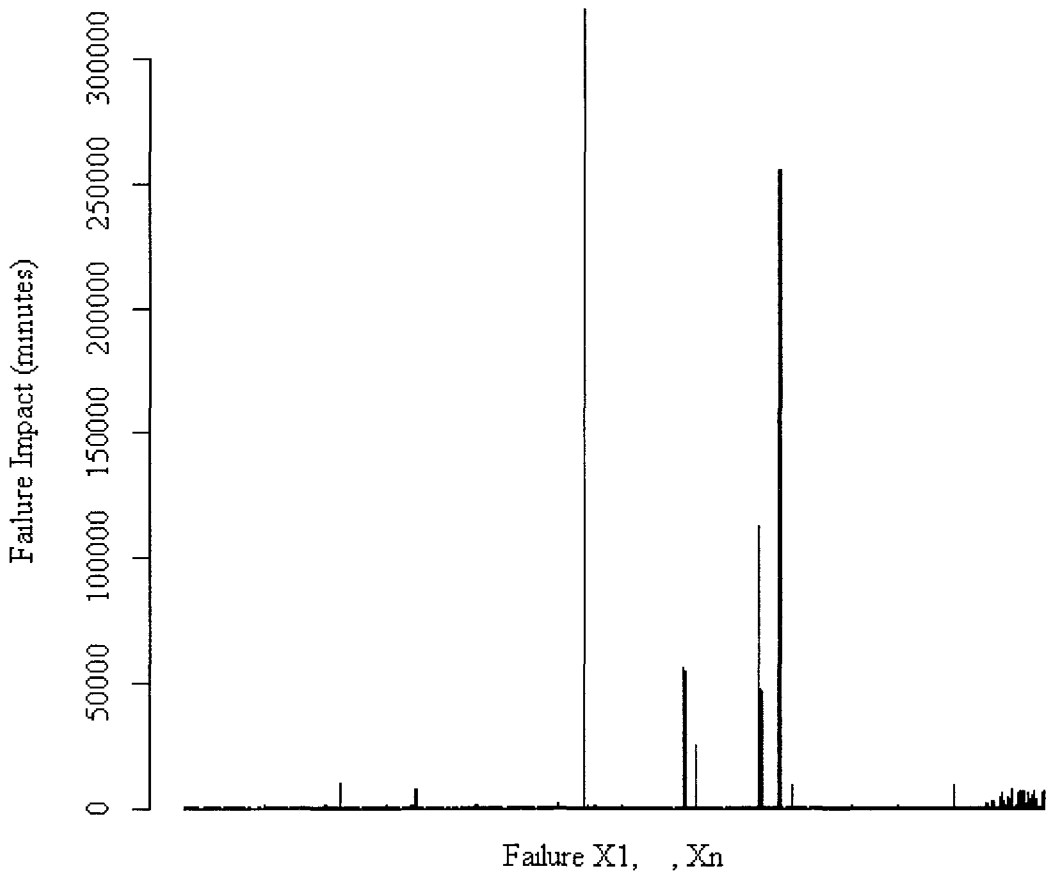


Figure 3.47 Impact of Each Failure, System 16, LANL Data

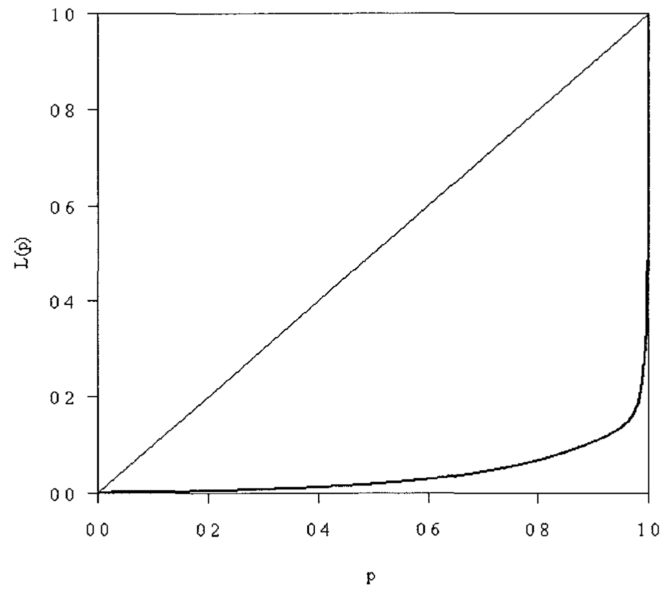


Figure 3.48 Lorenz Curve, System 16, LANL Data

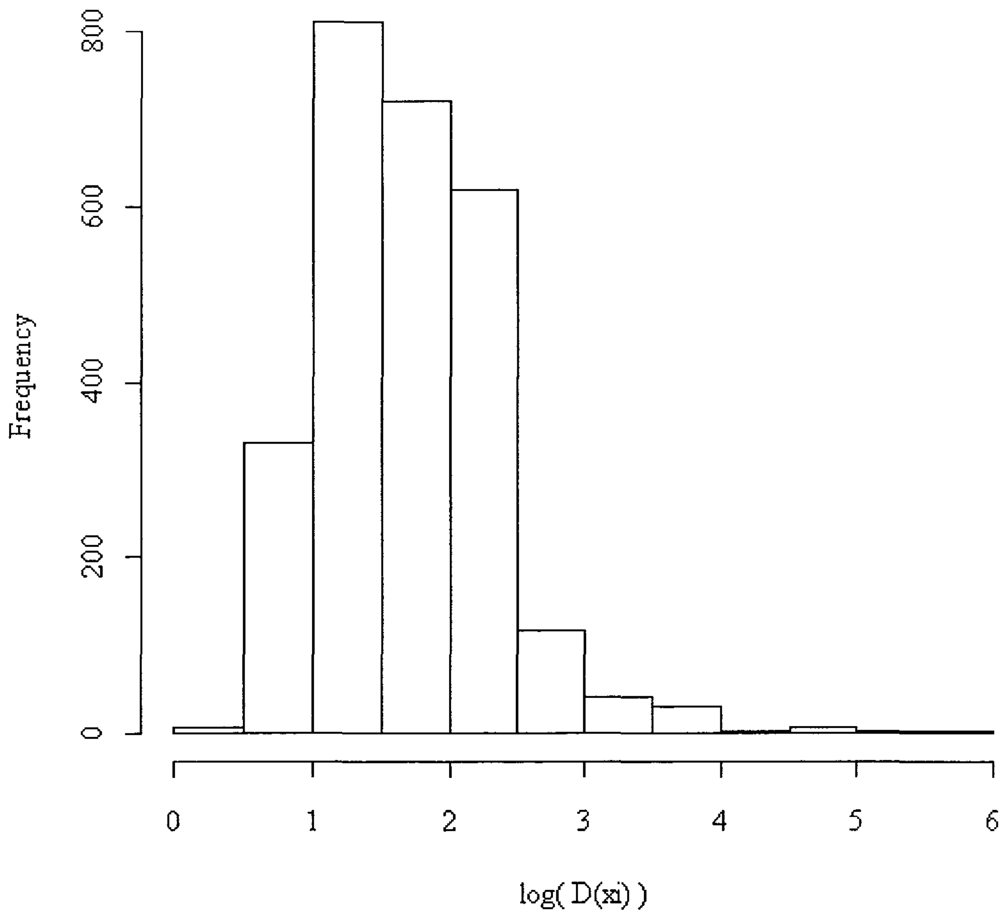


Figure 3.49 Failure Impact Histogram, System 16, LANL Data

Table 3.16 Inequality Coefficients, System 17, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
17	20th	22	3,260	0.6989682	0.6530687	0.1836791
17	40th	49	5,434	0.7013287	0.6423195	0.1851869
17	60th	77	10,232	0.7175553	0.6857478	0.1917146
17	80th	104	12,881	0.72673	0.68572	0.2003400
17	100th	126	15,651	0.7847755	0.7355964	0.2148792

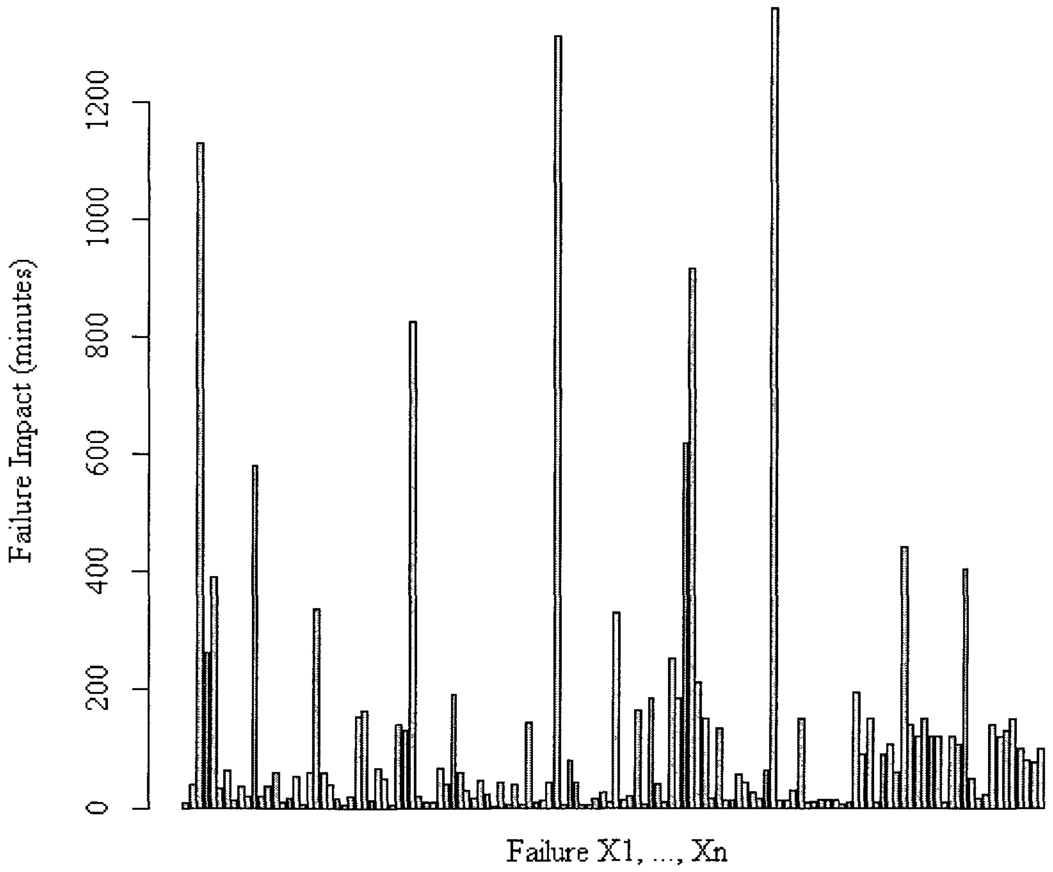


Figure 3.50 Impact of Each Failure, System 17, LANL Data

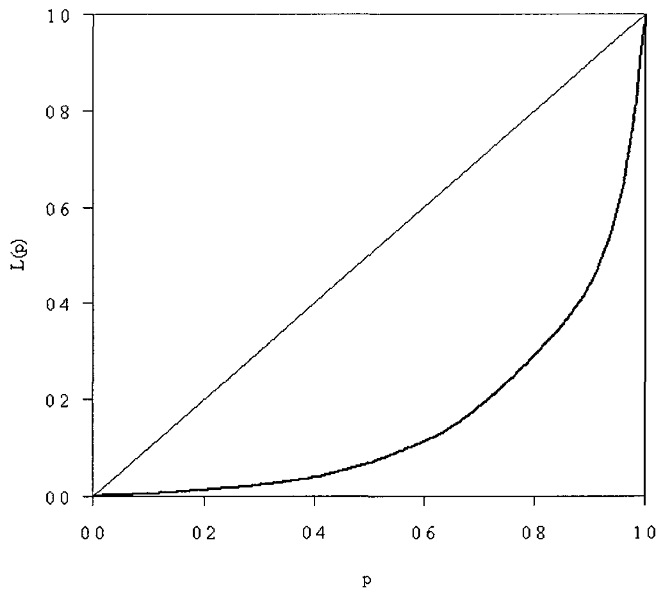


Figure 3.51 Lorenz Curve, System 17, LANL Data

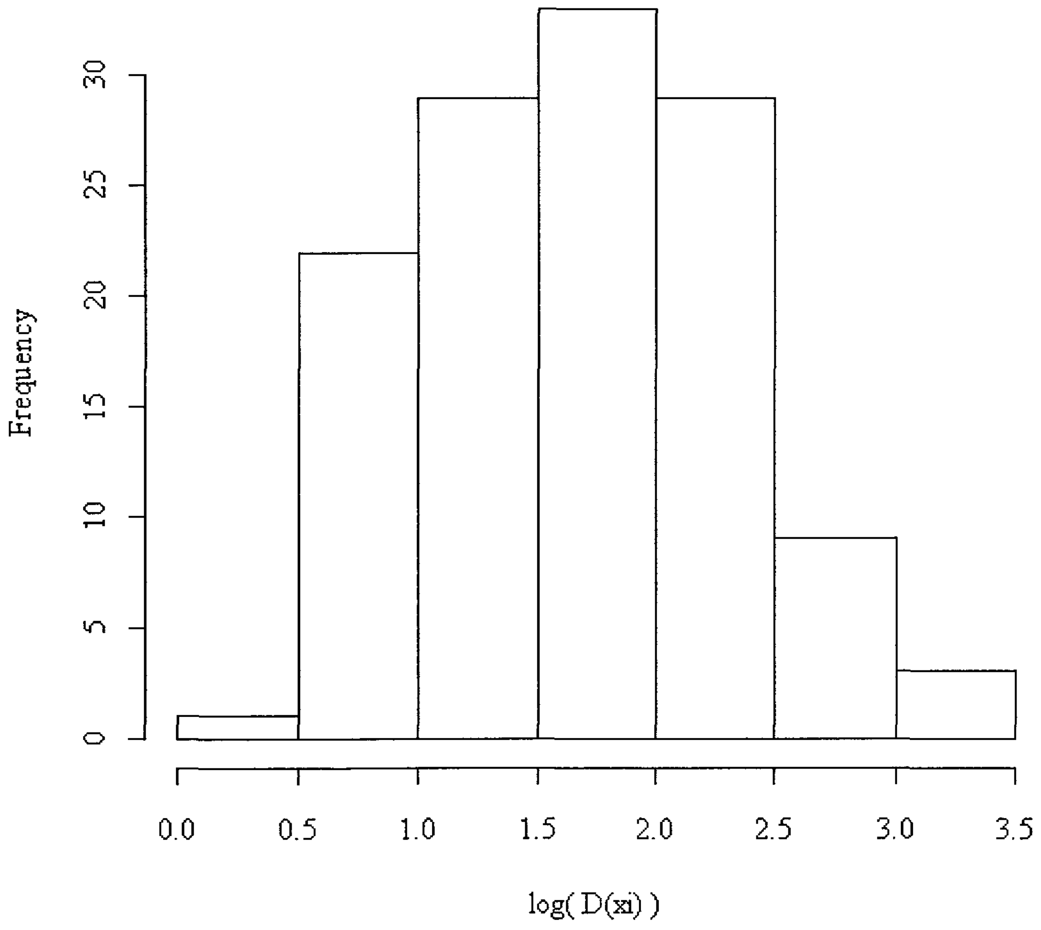


Figure 3.52 Failure Impact Histogram, System 17, LANL Data

Table 3.17 Inequality Coefficients, System 18, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
18	20th	797	196,049	0.7050154	0.6213237	0.2172665
18	40th	1,614	362,808	0.712082	0.6282973	0.2339089
18	60th	2,378	466,939	0.7095604	0.6257881	0.2253477
18	80th	3,224	577,409	0.709114	0.6251526	0.2219993
18	100th	3,997	716,641	0.7092562	0.6276465	0.2229577

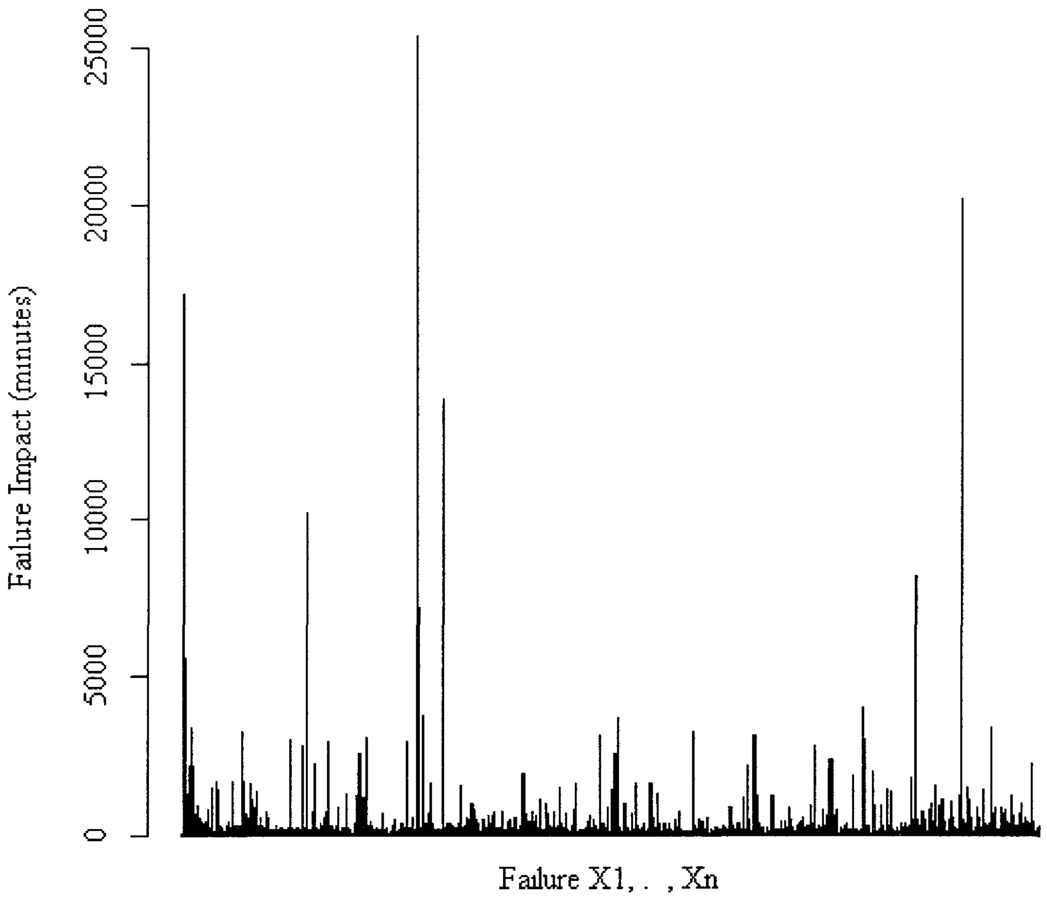


Figure 3.53 Impact of Each Failure, System 18, LANL Data

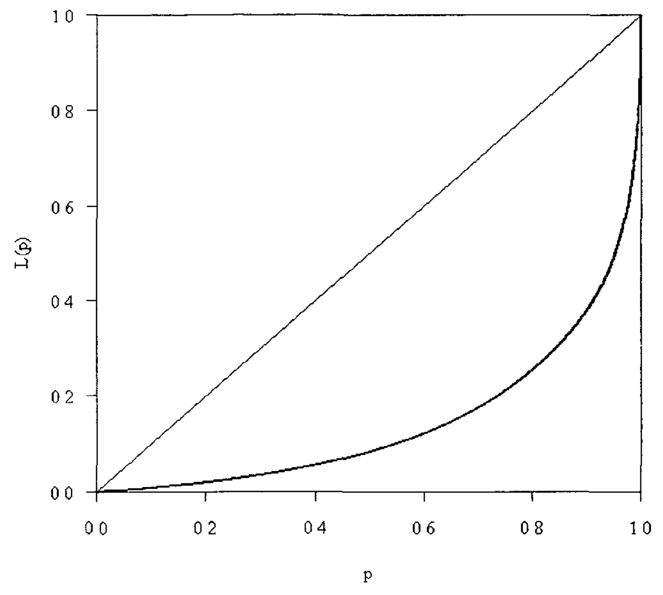


Figure 3.54 Lorenz Curve, System 18, LANL Data

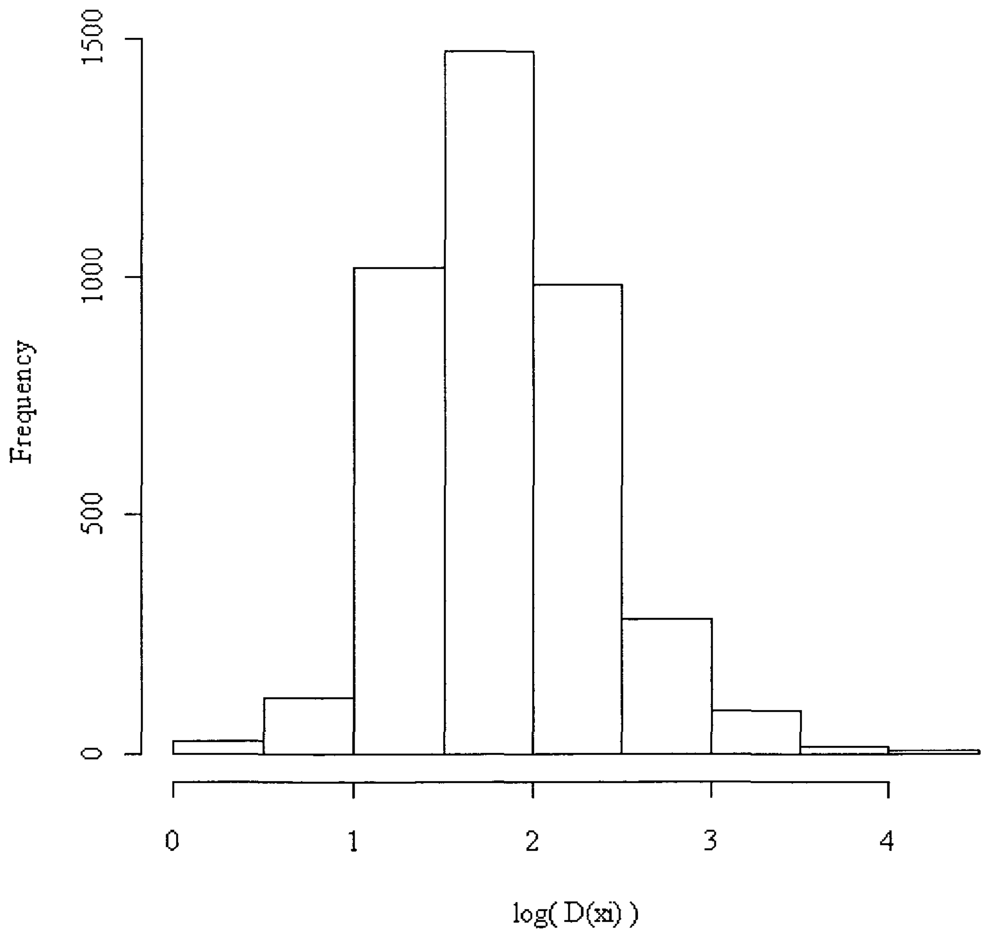


Figure 3.55 Failure Impact Histogram, System 18, LANL Data

Table 3.18 Inequality Coefficients, System 19, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
19	20th	661	179,209	0.7520437	0.6696452	0.2678224
19	40th	1,298	278,133	0.7269844	0.6355074	0.2520972
19	60th	1,992	377,358	0.7296928	0.640358	0.2476286
19	80th	2,656	493,133	0.7152188	0.6226461	0.2286325
19	100th	3,284	681,077	0.7409367	0.6561462	0.2670949

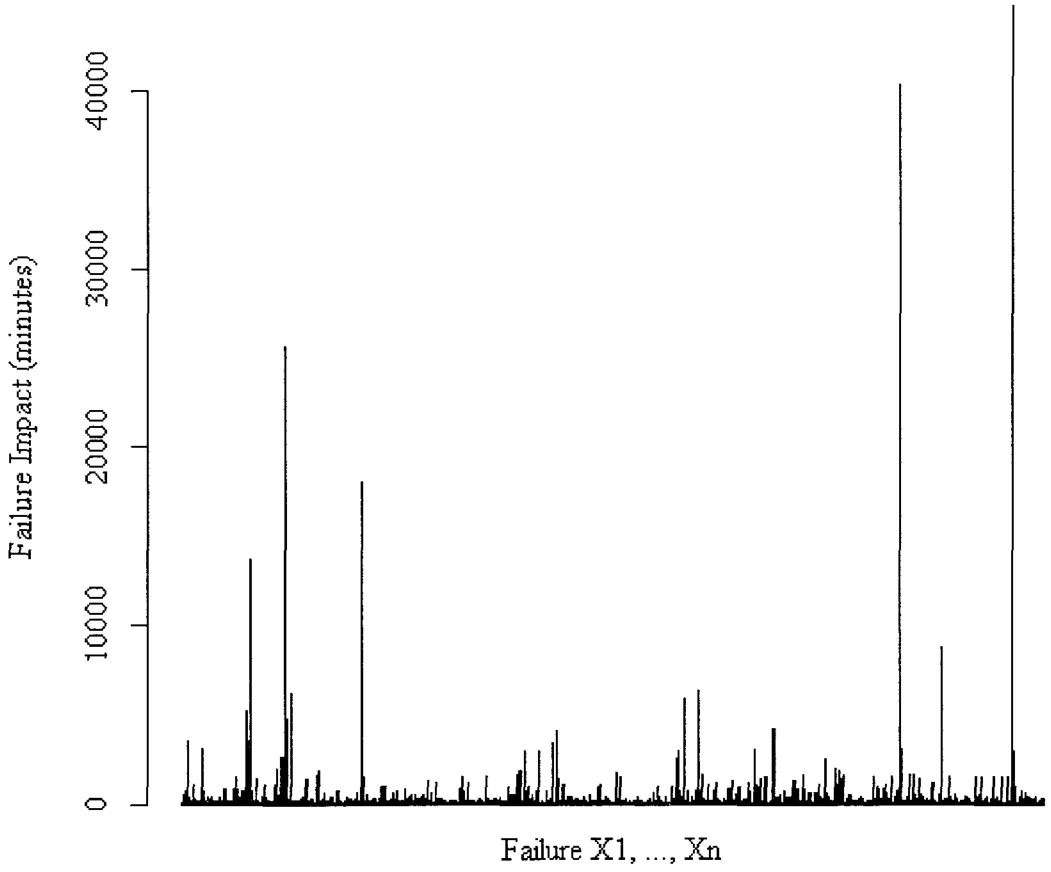


Figure 3.56 Impact of Each Failure, System 19, LANL Data

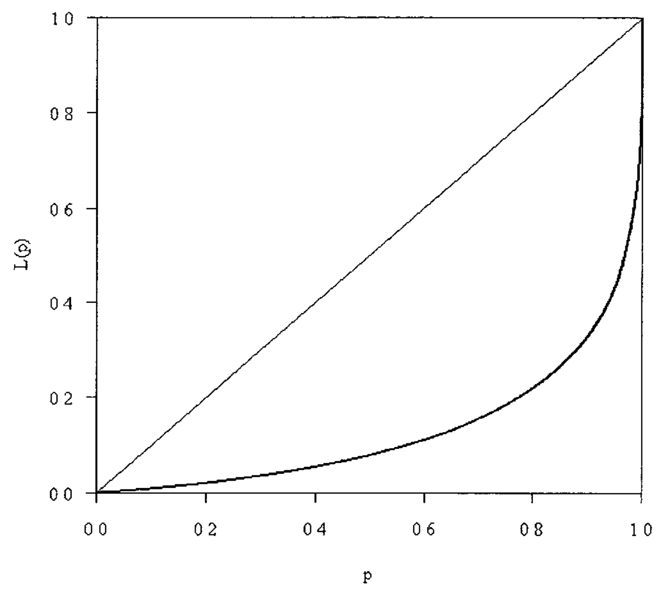


Figure 3.57 Lorenz Curve, System 19, LANL Data

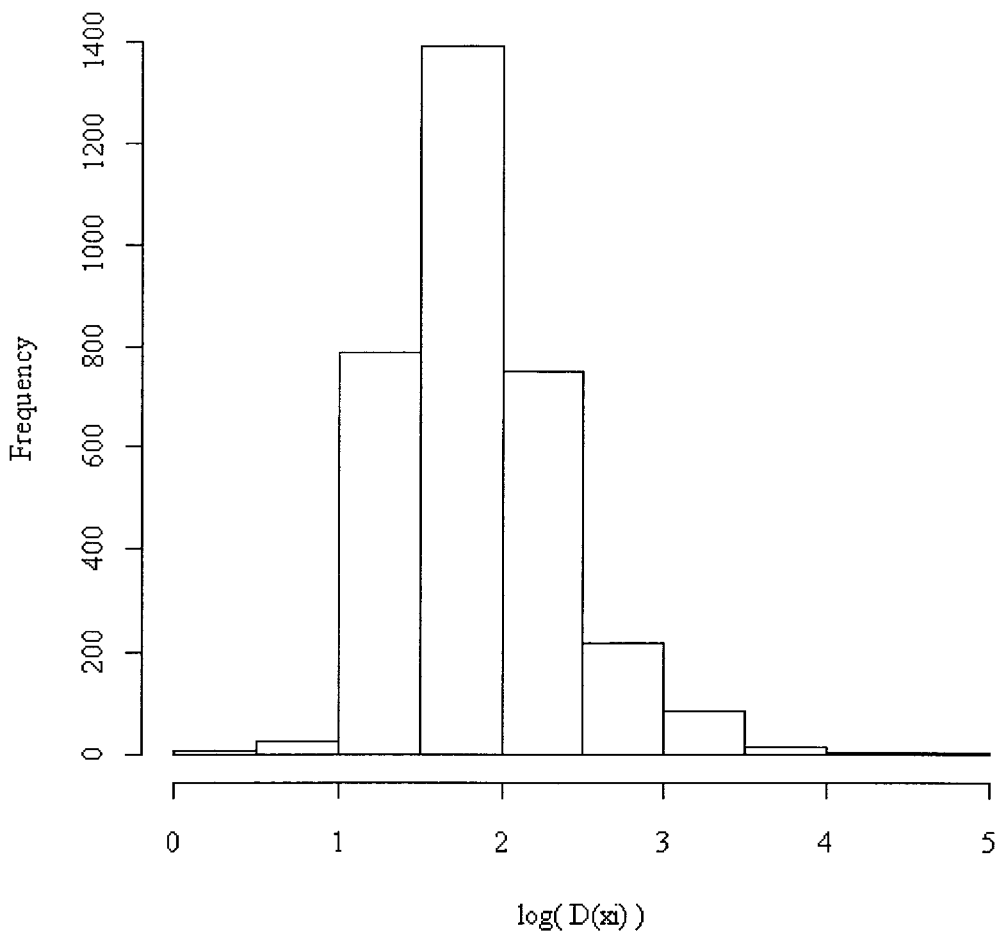


Figure 3.58 Failure Impact Histogram, System 19, LANL Data

Table 3.19 Inequality Coefficients, System 20, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
20	20th	494	147,549	0.7598128	0.6832824	0.2785598
20	40th	1,021	249,951	0.730096	0.6446679	0.2780654
20	60th	1,478	386,880	0.7377354	0.6552135	0.2886511
20	80th	1,995	464,709	0.7131277	0.6286357	0.2632639
20	100th	2,478	520,493	0.7138073	0.6316302	0.2574603

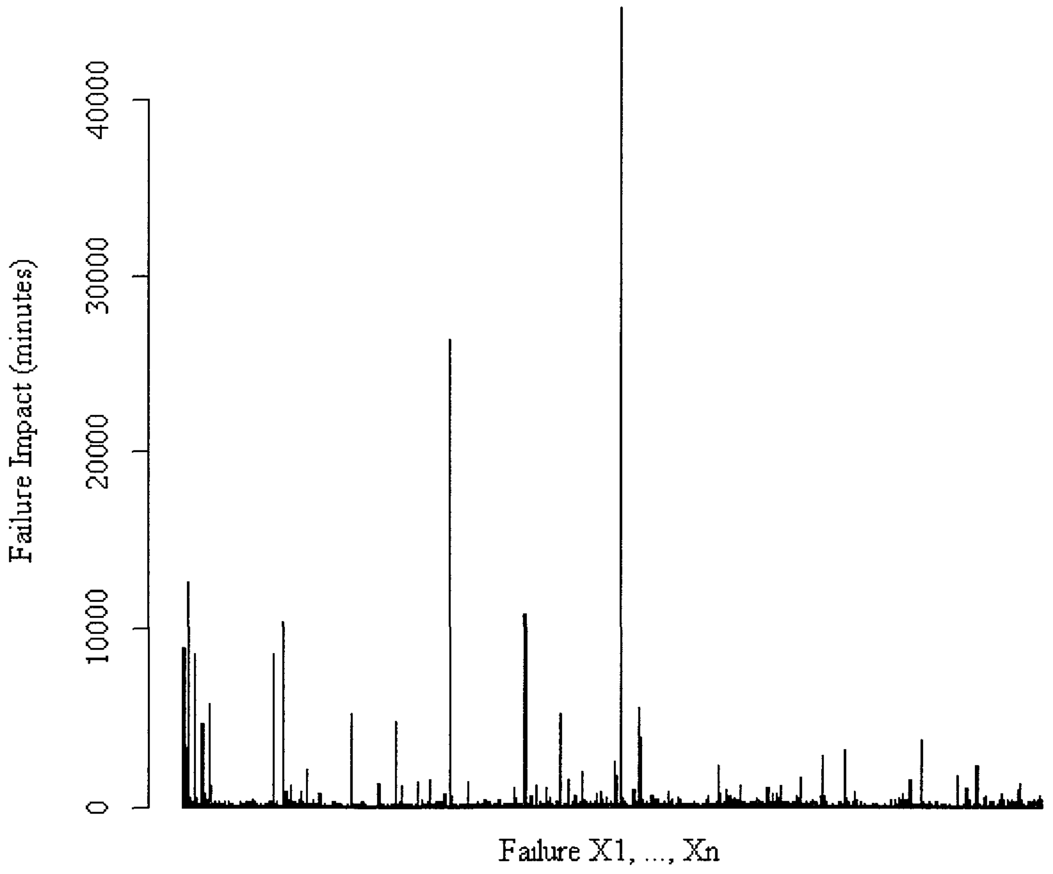


Figure 3.59 Impact of Each Failure, System 20, LANL Data

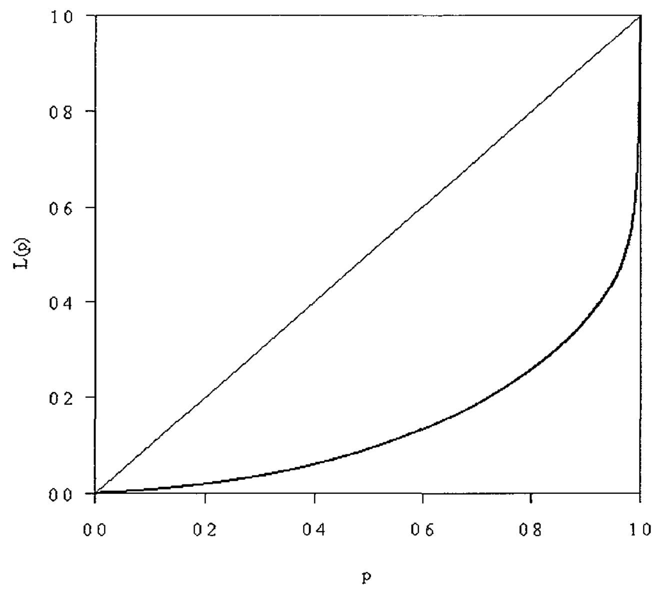


Figure 3.60 Lorenz Curve, System 20, LANL Data

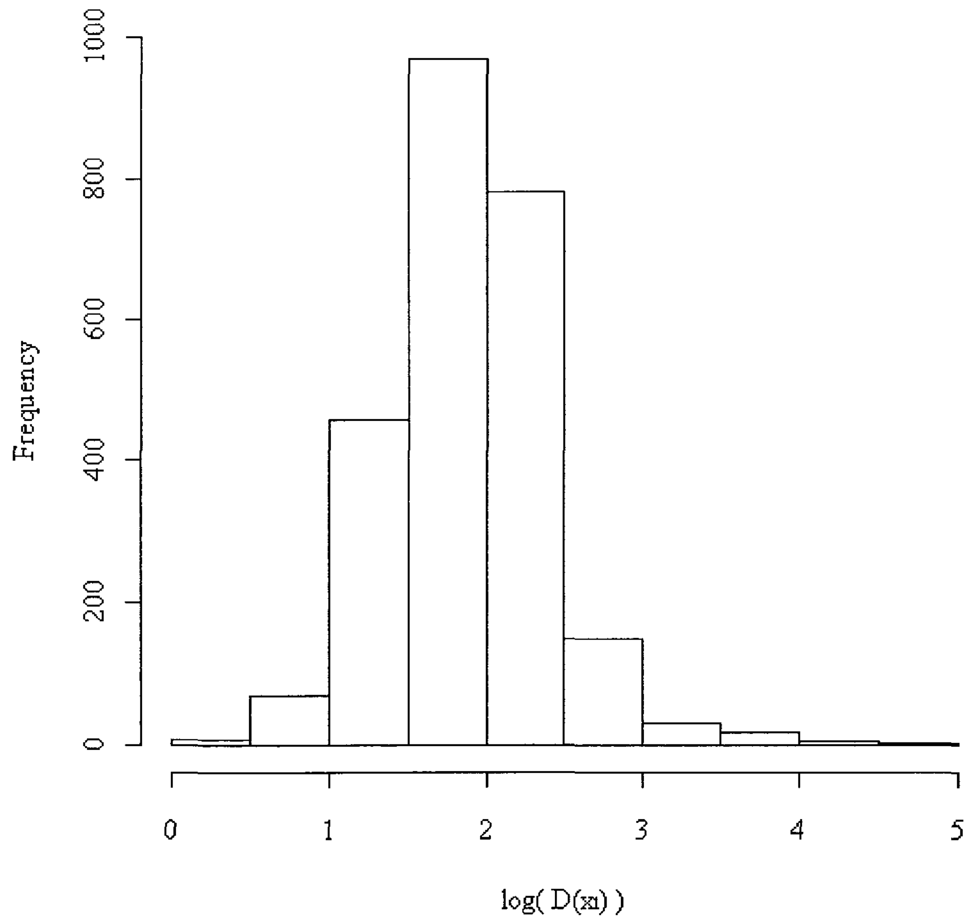


Figure 3.61 Failure Impact Histogram, System 20, LANL Data

Table 3.20 Inequality Coefficients, System 21, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
21	20th	28	4,122	0.5584494	0.4520249	0.1100806
21	40th	44	6,724	0.5513155	0.4406514	0.1041251
21	60th	73	9,985	0.526417	0.4201482	0.09500196
21	80th	96	14,147	0.5109527	0.3972765	0.0897491
21	100th	110	15,807	0.4956176	0.3767154	0.0845924

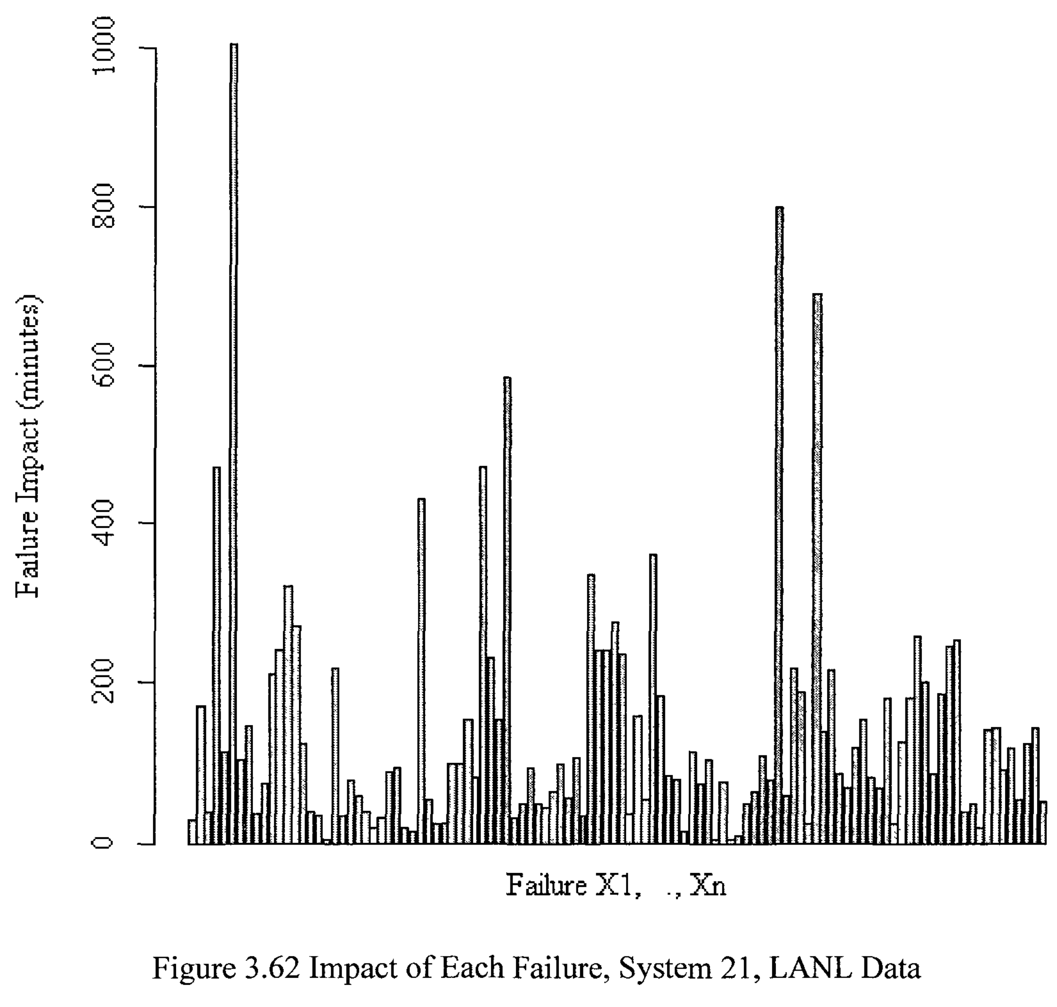


Figure 3.62 Impact of Each Failure, System 21, LANL Data

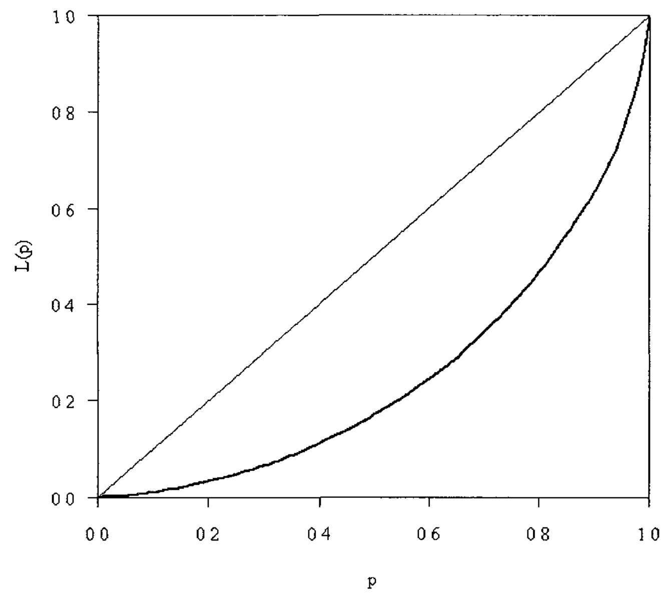


Figure 3.63 Lorenz Curve, System 21, LANL Data

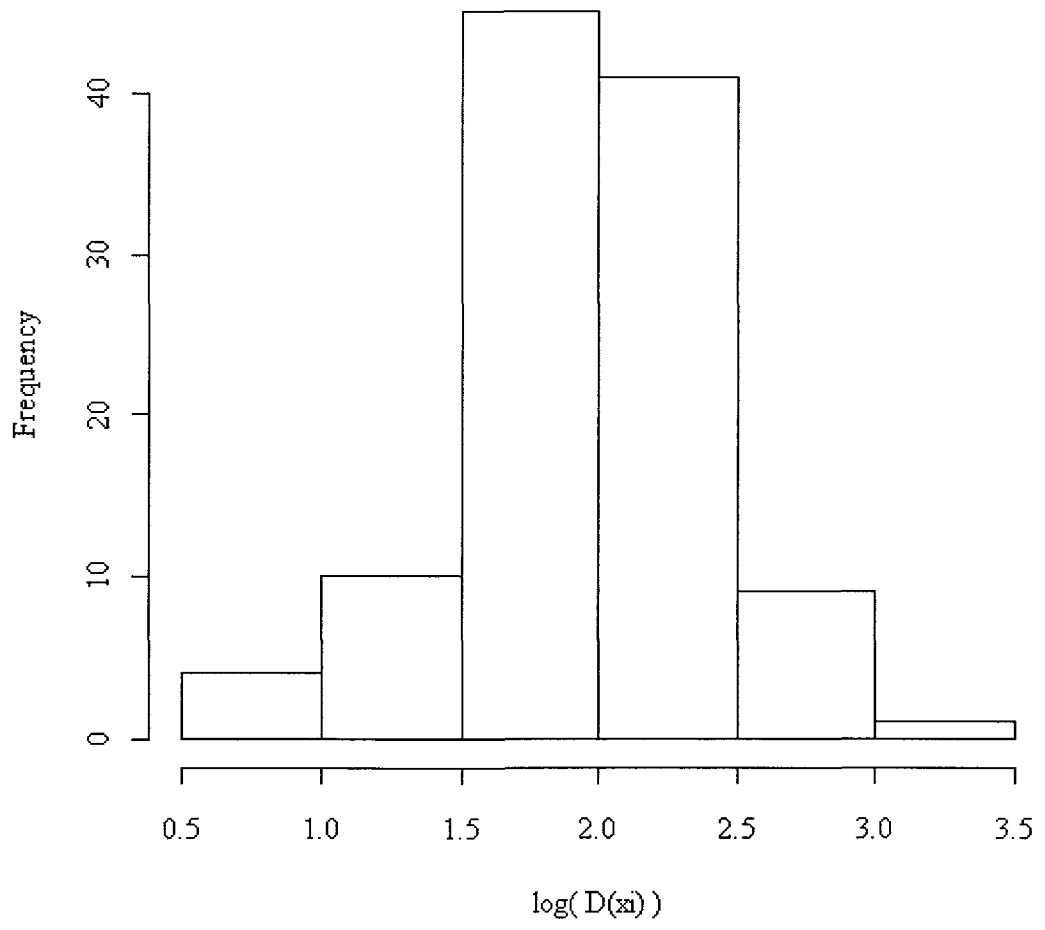


Figure 3.64 Failure Impact Histogram, System 21, LANL Data

Table 3.21 Inequality Coefficients, System 22, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
22	20th	52	10,816	0.4742298	0.3459984	0.0911673
22	40th	98	17,558	0.413181	0.2724191	0.06959792
22	60th	162	66,477	0.7286914	0.6479295	0.2960288
22	80th	208	78,659	0.6937795	0.600729	0.2632218
22	100th	246	84,556	0.6699327	0.5698679	0.2503528

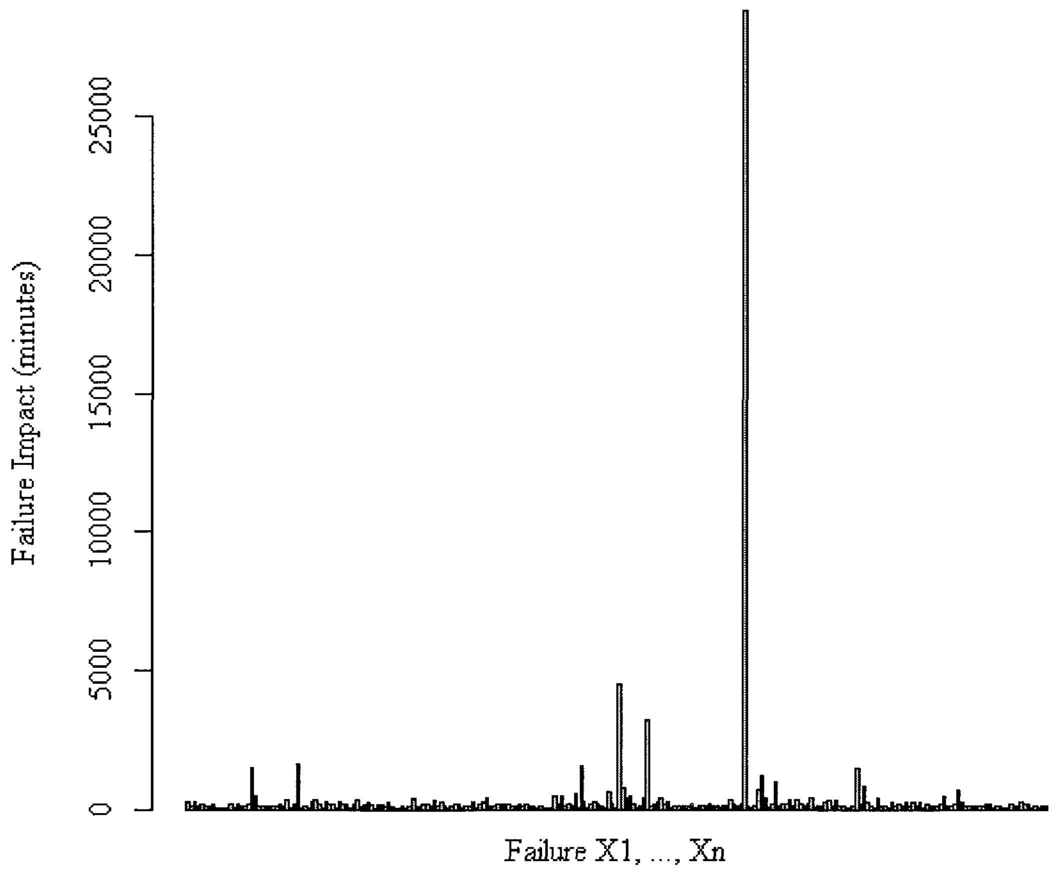


Figure 3.65 Impact of Each Failure, System 22, LANL Data

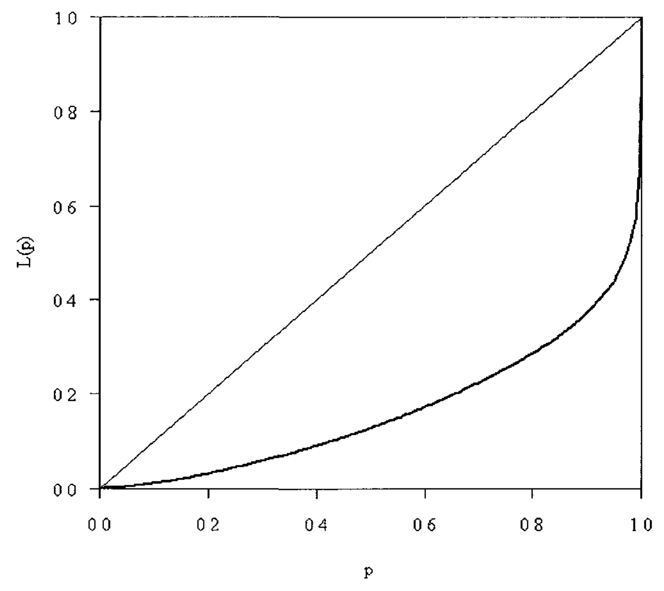


Figure 3.66 Lorenz Curve, System 22, LANL Data

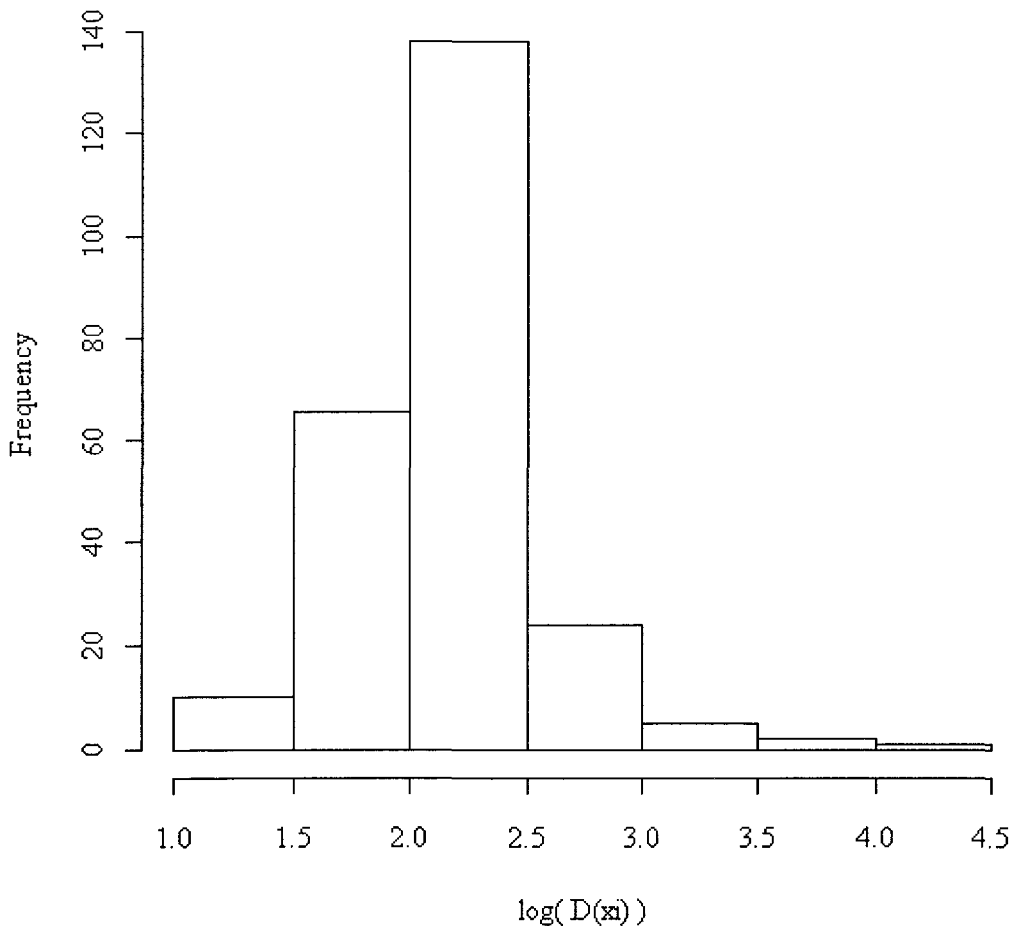


Figure 3.67 Failure Impact Histogram, System 22, LANL Data

Table 3.22 Inequality Coefficients, System 23, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
23	20th	114	16,049	0.5291957	0.4541011	0.09828172
23	40th	218	32,484	0.589893	0.523773	0.1262523
23	60th	343	50,474	0.5910107	0.5233588	0.1288740
23	80th	452	70,531	0.57899	0.5117965	0.1215052
23	100th	564	85,743	0.591682	0.5148623	0.1284186

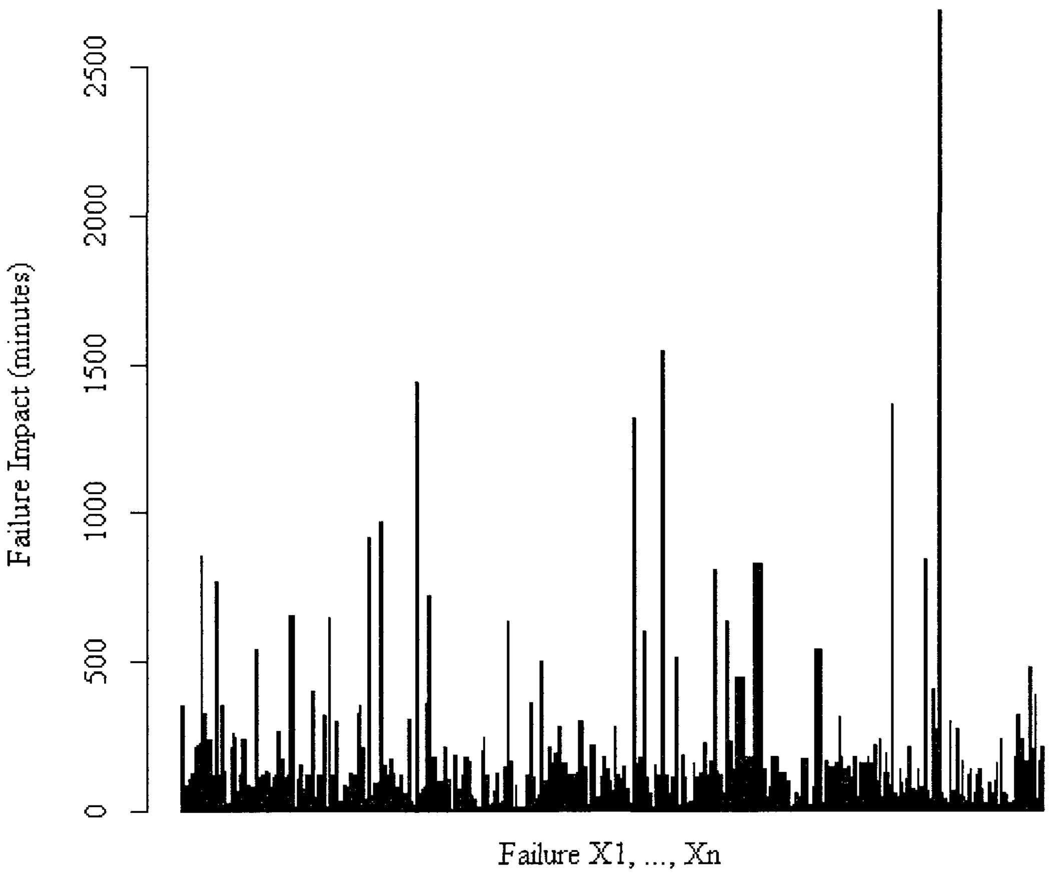


Figure 3.68 Impact of Each Failure, System 23, LANL Data

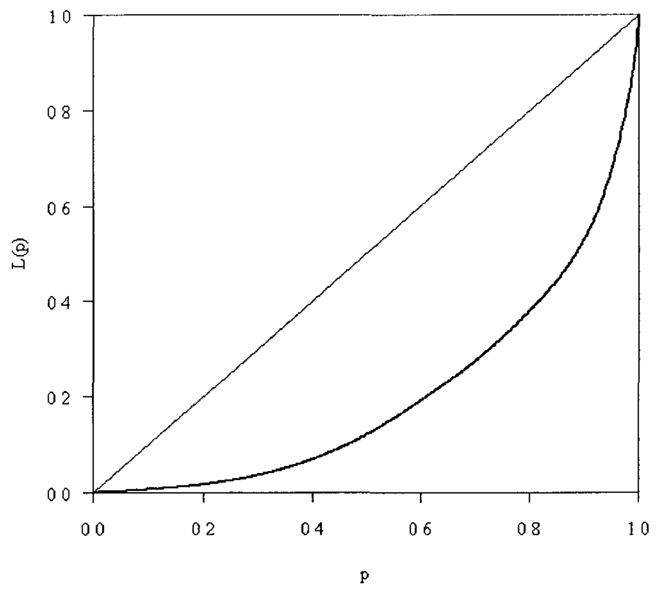


Figure 3.69 Lorenz Curve, System 23, LANL Data

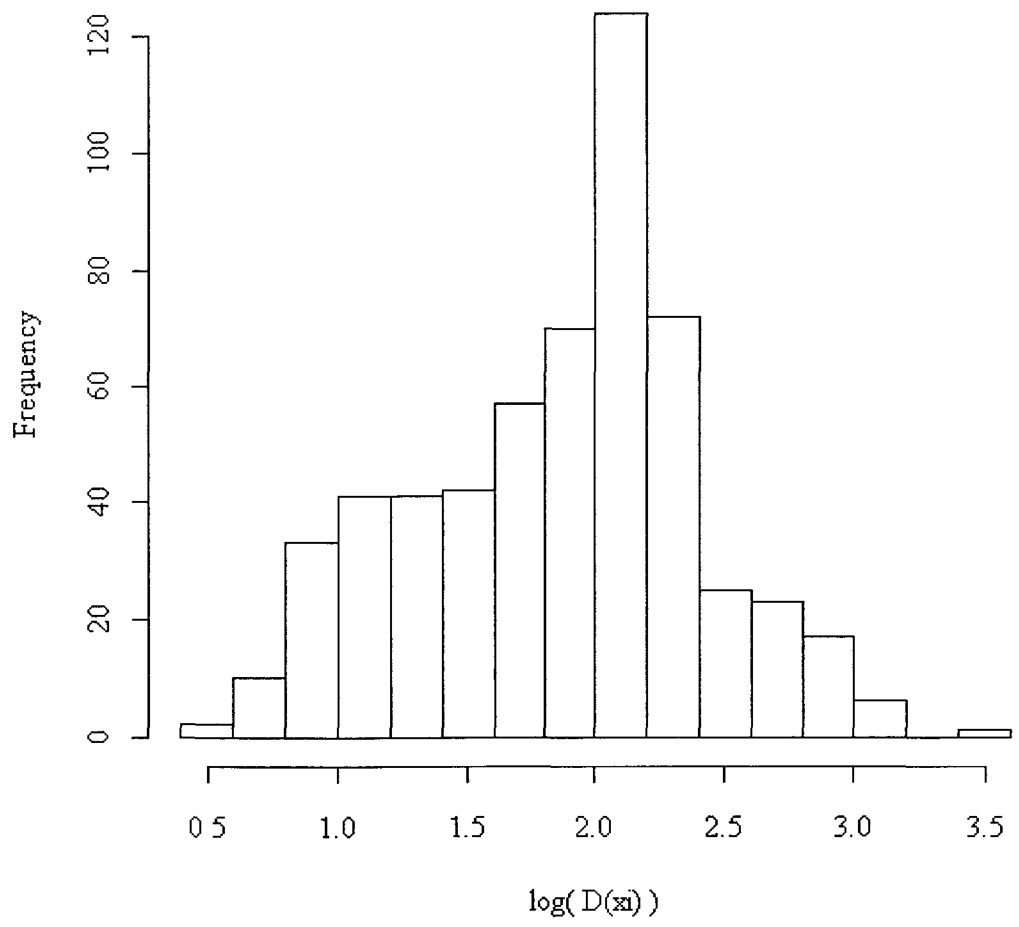


Figure 3.70 Failure Impact Histogram, System 23, LANL Data

Table 3.23 Inequality Coefficients, System 24, LANL Data

System	Percentile	Failures	Downtime (min.)	Gini	Atk (.99)	Atk (.20)
24	20th	32	6,658	0.4370119	0.2876526	0.06265949
24	40th	59	10,882	0.484666	0.3301613	0.07753442
24	60th	93	45,331	0.8146349	0.7538131	0.3706069
24	80th	126	52,208	0.786939	0.714698	0.3425476
24	100th	155	58,177	0.7704078	0.6930412	0.3240548

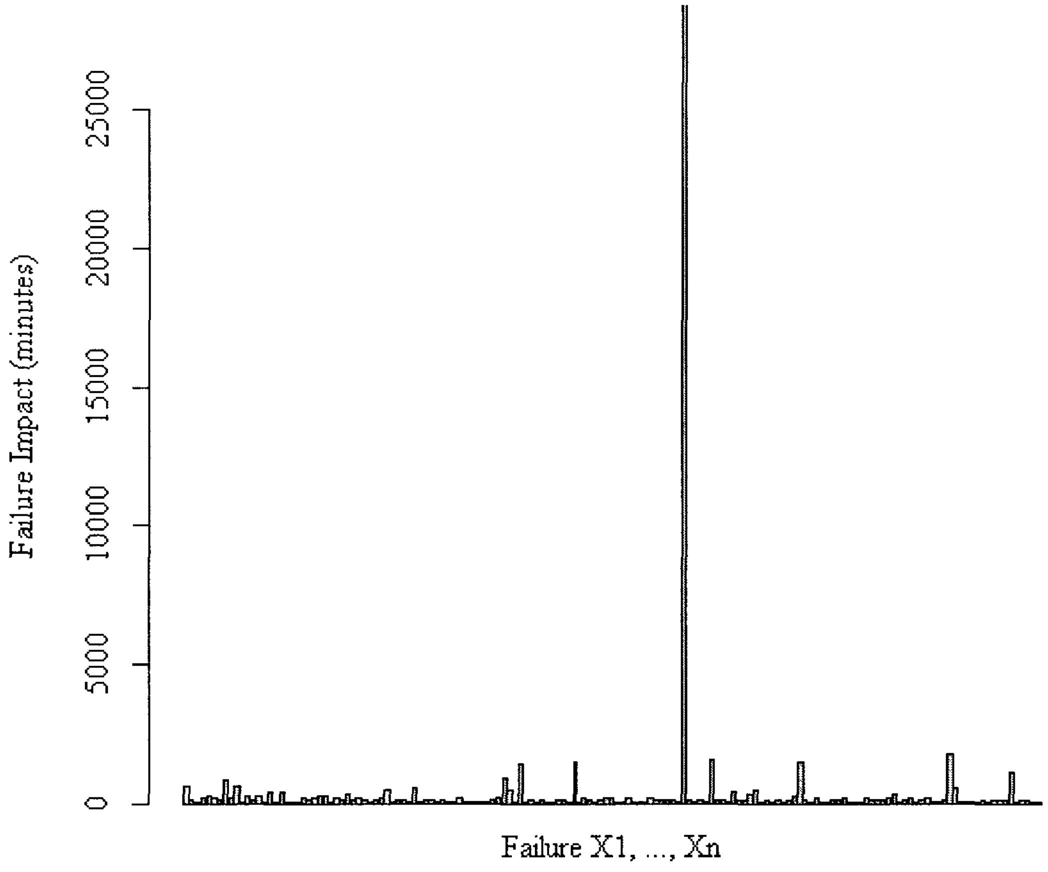


Figure 3.71 Impact of Each Failure, System 24, LANL Data

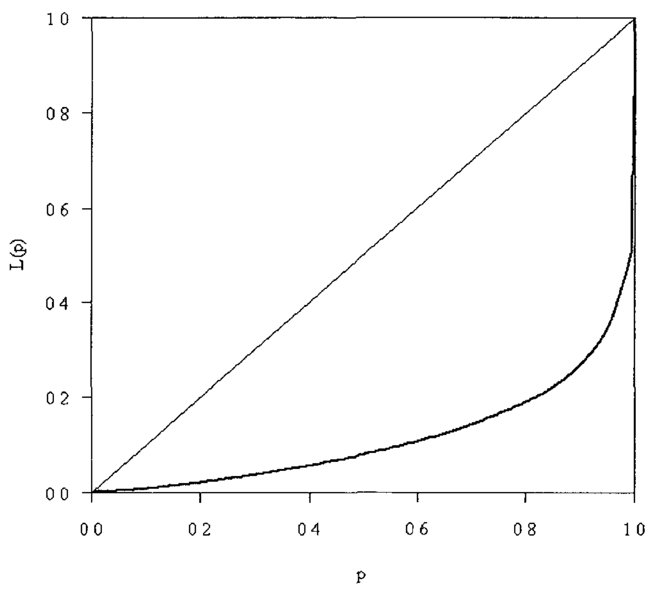


Figure 3.72 Lorenz Curve, System 24, LANL Data

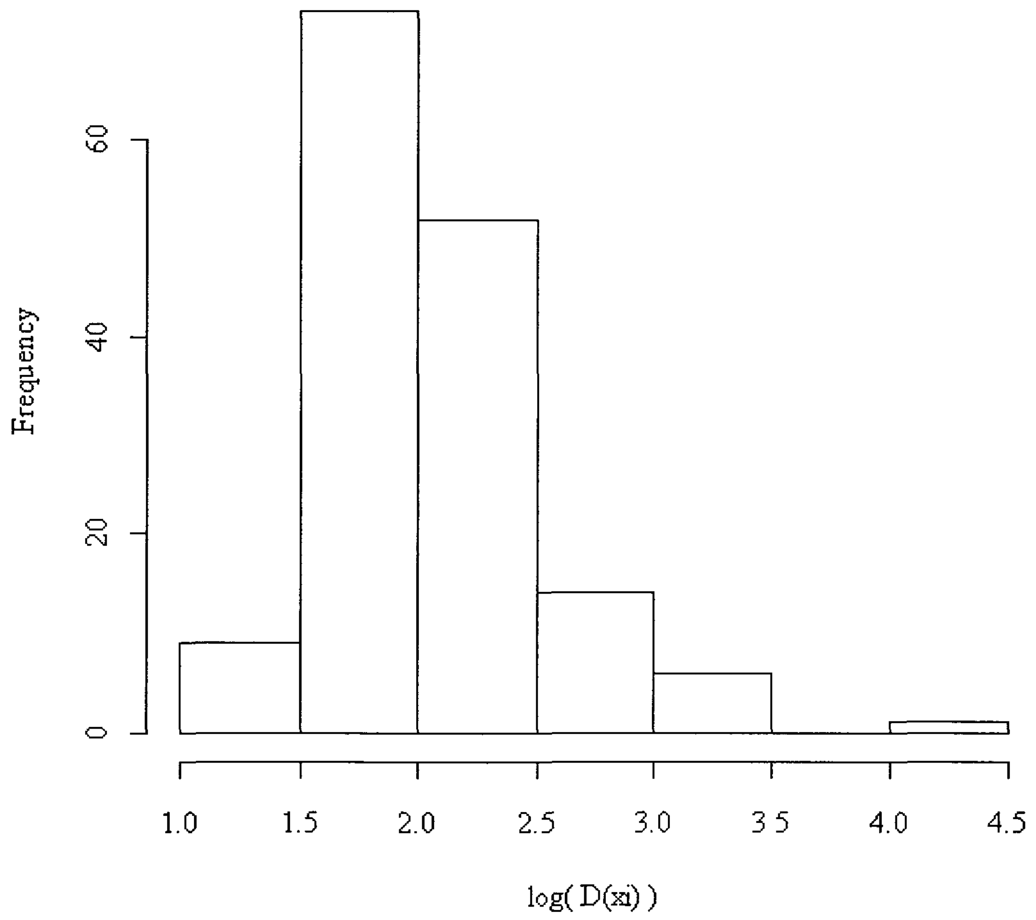


Figure 3.73 Failure Impact Histogram, System 24, LANL Data

Table 3.24 Average Gini and Atkinson Coefficient per Percentile, LANL Data

Percentile	Avg Gini Coefficient	Avg Atk. ($\epsilon = 0.20$)	Avg Atk. ($\epsilon = 0.99$)
20th	0.68426383	0.21688	0.648644
40th	0.715950435	0.246331	0.677153
60th	0.748810878	0.285942	0.717554
80th	0.753712383	0.28811	0.715567
100th	0.762782465	0.29328	0.722912

The following section concludes this chapter by drawing conclusions from these results and subsequently suggesting the introduction of the time-dependent failure index for HPC system volatility.

3.6.3 Conclusions

The average inequality coefficient per percentile results found in Table 3.24 show that as these systems age, they exhibit an increase in the level of inequality in the downtime caused by each incurred failure. That is, as HPC systems age, their failure activity becomes more volatile.

Significance testing verified the legitimacy of this result. When testing the null hypothesis H_0 that the resulting Gini coefficient averages represented the same value versus the alternate hypothesis H_a that the values significantly differed, a resulting p-value of 0.01455 confirmed the alternate hypothesis at the 95% confidence level. A multiple R-squared value of 0.8967 further proved that the resulting Gini coefficient averages successfully model the behavior of the systems in the given percentiles.

When testing the null hypothesis H_0 that the resulting Atkinson coefficient averages at parameter 0.20 represented the same value versus the alternate hypothesis H_a that the values significantly differed, a resulting p-value of 0.02465 confirmed the alternate hypothesis at the 95% confidence level. A multiple R-squared value of 0.8546 further proved that the resulting Atkinson coefficient averages at parameter 0.20 successfully model the behavior of the systems in the given percentiles.

When testing the null hypothesis H_0 that the resulting Atkinson coefficient averages at parameter 0.99 represented the same value versus the alternate hypothesis H_a that the values significantly differed, a resulting p-value of 0.02904 confirmed the alternate hypothesis at the 95% confidence level. A multiple R-squared value of 0.8384 further proved that the resulting Atkinson coefficient averages at parameter 0.99 successfully model the behavior of the systems in the given percentiles.

This result suggests that system age significantly affects the volatility of the failure behavior exhibited by a given HPC system with regards to failures with both large and small impacts on downtime. The Atkinson coefficients confirm this result. There is clearly more inequality reported by the Atkinson ($\varepsilon = 0.99$) result than the Atkinson ($\varepsilon = 0.20$) result

Clearly, larger failures have a more significant effect on the level of equality contained in the dataset than smaller failures. For example, System 13 exhibits a single failure with an impact of 195195 minutes, or 5.648 days. This dwarfs the impact of all other failures encountered by the system and results in a 100th percentile Gini coefficient of 0.8384 and a 99th percentile Atkinson ($\epsilon = 0.99$) coefficient that was rounded to 1 via the software. However, the Atkinson ($\epsilon = 0.20$) coefficient, which lowered this and all other values, was only 0.4596139. The resulting inequality coefficients for all systems were most affected by the largest failures, such as this one.

To better represent the behavior of these and other systems, we suggest the derivation of a time-dependent inequality measure. The resulting equation – the Failure index (FI) – will generate coefficients that can then be used as a measure of HPC system volatility in time. It may be used in conjunction with existing metrics such as Mean Time Between Failure (MTBF) and the 'nines' measure of system reliability to better illustrate the failure activity of a given HPC system. The FI has roots in the Atkinson index and via the time-dependent generation of a scale- and location-invariant inequality coefficient it allows for a normalized representation of HPC system failure volatility in time.

Whereas the Gini and Atkinson index coefficients in this chapter were calculated with respect to the impact of each failure located in the record, an FI coefficient is generated with respect to the cumulative downtime caused by all failures in a given time interval. As explained in the next chapter's introduction, the Failure Index is less affected by single large failures such as the one encountered by System 13.

The following chapter gives a mathematical introduction of the FI. Chapter 5 contains a case study in which FI coefficients are generated for the same LANL dataset used to formulate the results in this chapter. Those coefficients are then analyzed and compared with the information in this chapter to yield conclusions relative to the failure volatility of aging HPC systems. Those conclusions along with suggested future work are given in Chapter 6.

CHAPTER 4

A FAILURE INDEX FOR HPC APPLICATIONS

4.1 The Failure Index

The provision of resilient exascale HPC applications is a complicated and multifaceted effort requiring input from and coordination between computer scientists, mathematicians, application developers and pure scientists from various government laboratories, corporations and academic institutions. Presently there exist calls for such collaboration [95] and further there exists the suggestion of increased cross-entity standardization in the vocabulary, algorithms, and log file formats utilized in the larger HPC research and development community. To facilitate such standardization, the community must both re-examine existing metrics and their meaning in tomorrow's failure-rich computing environments and also develop new statistics appropriate to the study of HPC application resilience [96, 97].

This chapter formally introduces such a value – the Failure Index (or FI) – for HPC system volatility. The FI is introduced here from a mathematical standpoint. Importantly, in this chapter the FI is introduced at a high level and in relation to a continuous function wherein time acts as the independent variable. Our real-world FI coefficient generation and analysis in Chapter 5 will take place on discrete, rather than continuous data, requiring numerical integration techniques. Specifically, FI coefficients will be generated for the LANL system reliability dataset previously visited in Chapter 3.

A metric such as the FI was first suggested in our work detailing the generation of pertinent resilience-related application information found in Chapter 2. As shown in Chapter 3, the Gini and Atkinson inequality indices provide the suggested normalized view into an aspect of HPC application behavior. Specifically, such inequality indices generate coefficients relative to the level of inequality found in the individual failures incurred by an HPC system. Further, the scale- and location-invariant nature of these values allows for the comparison of multiple HPC systems regardless of size.

Chapter 3 introduced those concepts and demonstrated their ability to capture this information. An analogous index is now present that, while similar in structure to the above, serves to capture the level of volatility in *total* system downtime relative to the age of a given HPC system.

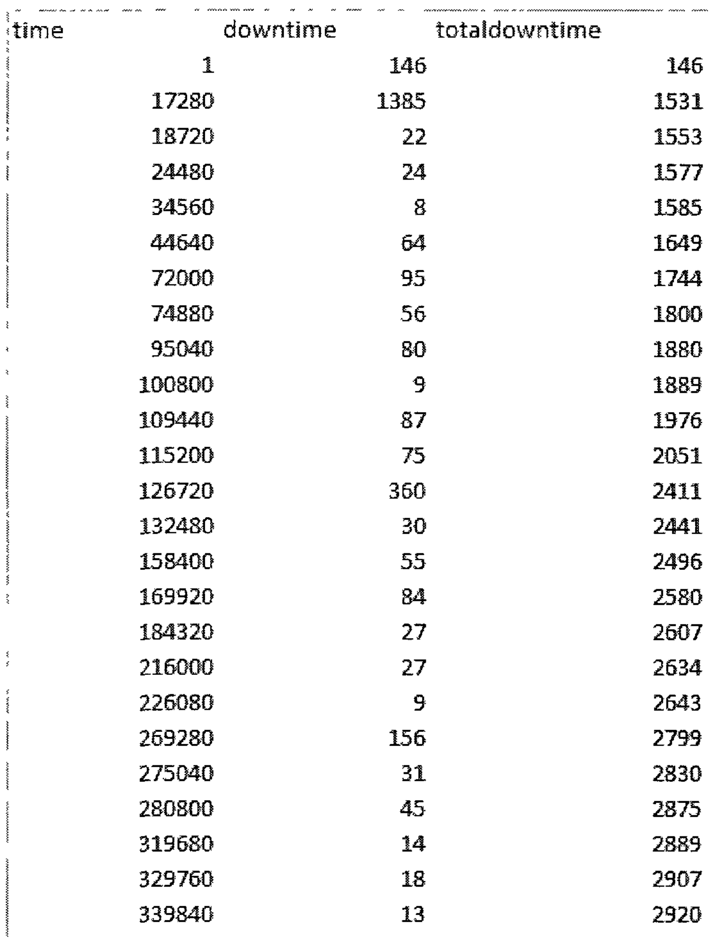
It is important to note the difference between the information captures by inequality metrics such as the Gini and Atkinson indices and the information captured by the Failure Index. Existing inequality indices are calculated with respect to discrete weighted events. The Failure Index is calculated with respect to *cumulative time-dependent data*. That is, where the Gini or Atkinson indices capture the level of inequality in multiple individual failure events, the Failure Index captures the *overall level of volatility exhibited by the system in time.*

The information captured by the FI was approximated in Chapter 3 via the calculation of Gini and Atkinson coefficients relative to various percentiles of a system's total lifespan. Using the FI, such percentiles are not needed, as its time-dependent nature allows the FI to examine the volatility in cumulative system downtime with respect to the age of the system. The Failure Index is mathematically defined as

$$FI_{\epsilon}\big(F(T)\big) = 1 - \frac{1}{F(T)} \left(\frac{1}{T} \int_{1}^{T} F(t)^{1-\epsilon} dt \right)^{\frac{1}{1-\epsilon}} . \tag{4.1}$$

Here, $F(t)$ represents the cumulative distribution function of system downtime resulting from failure, and $F(T)$ represents the cumulative system downtime at a given time T. As the Failure Index is constructed as a time-dependent implementation of the Atkinson index, it also contains the parameter ε, $0 \leq \varepsilon < 1$, that allows the Failure Index to place emphasis on various segments of a distribution, with parameters closer to (but not equal to) 1 generating FI coefficients giving greater weight to the larger elements of the distribution than parameters closer to 0. An FI coefficient, then, captures the level of volatility in total system downtime at time T using the given parameter, with higher levels of volatility yielding FI coefficients closer to 1 and lower levels of volatility yielding FI coefficients closer to 0. Equation (4.1) presents the novel mathematical contribution of this dissertation, and all resulting analysis in Chapter 5 is performed using this model.

Using the terminology from the LANL dataset used in Chapters 3 and 5, existing inequality indices such as the Gini or Atkinson index generate coefficients which measure the level of inequality contained in the ”*downtime*” column. The Failure Index, however, generates coefficients which measure the level of volatility in the *“total downtime”* column with respect to the *“time”* column. Figure 4.1 shows these three columns from System 6’s reliability dataset, here represented in minutes.

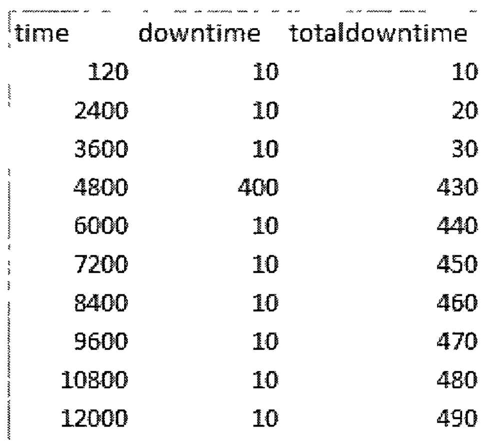


time	downtime	totaldowntime
1	146	146
17280	1385	1531
18720	22	1553
24480	24	1577
34560	8	1585
44640	64	1649
72000	95	1744
74880	56	1800
95040	80	1880
100800	9	1889
109440	87	1976
115200	75	2051
126720	360	2411
132480	30	2441
158400	55	2496
169920	84	2580
184320	27	2607
216000	27	2634
226080	9	2643
269280	156	2799
275040	31	2830
280800	45	2875
319680	14	2889
329760	18	2907
339840	13	2920

Figure 4.1 “Time”, “Downtime” and “Total Downtime” Columns, LANL Data

In Chapter 5, Failure Index coefficients will be generated for all 23 systems contained in the LANL dataset. Like existing inequality indices such as the Gini and Atkinson index, the Failure Index produces scale- and location-invariant coefficients, allowing for the comparison of various machines regardless of size.

FI coefficients are less sensitive to the volatility caused by a single abnormally strong or weak failure and more sensitive to stretches of volatile or non-volatile failure behavior in time. To illustrate this, consider the datasets given in Figure 4.2 and 4.3.

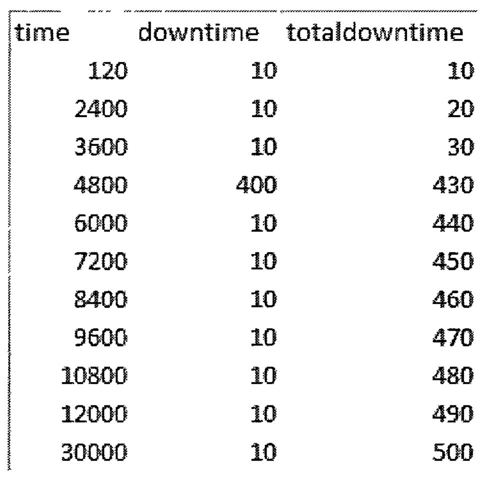


time	downtime	totaldowntime
120	10	10
2400	10	20
3600	10	30
4800	400	430
6000	10	440
7200	10	450
8400	10	460
9600	10	470
10800	10	480
12000	10	490

Figure 4.2 Sample Data 1, FI Coefficients vs. Atkinson Coefficients

The above dataset generates an Atkinson index coefficient ($\varepsilon = 0.99$) of 0.7030416 for *"down time"*. Clearly, a high level of inequality in *"down time"* exists due to a single failure causing a down time of 400 minutes. Likewise, the Failure Index coefficient ($\varepsilon = 0.99$) for the above dataset is 0.8281679, which represents a substantial amount of failure volatility in the given time frame.

The system depicted in Figure 4.2 runs for a total of 12000 minutes, or 200 hours. But what if the system ran for another 300 hours before encountering another 10 minute failure? That situation is represented in Figure 4.3.



time	downtime	totaldowntime
120	10	10
2400	10	20
3600	10	30
4800	400	430
6000	10	440
7200	10	450
8400	10	460
9600	10	470
10800	10	480
12000	10	490
30000	10	500

Figure 4.3 Sample Data 2, FI Coefficients vs. Atkinson Coefficients

The Atkinson index coefficient (ε = 0.99) reduces slightly to a value of 0.6905922, as more equality is introduced to the dataset when a tenth failure with an impact of 10 minutes is accounted for. However, the Failure Index coefficient (ε = 0.99) *substantially* drops, to a value of 0.3925665. This is because the overall failure volatility of the system has substantially lowered due to the 300-hour non-failure period encountered by the system. Failure Index coefficients take such times into consideration. Standard inequality coefficients such as those generated by the Gini and Atkinson indices do not.

The source code used to generate these coefficients via the R statistical software package is located in Appendix B.

4.2 The GT Index

In addition to FI coefficients, the analysis in Chapter 5 will also generate coefficients using the GT Index, first proposed by Kaminskiy et al in 2008 [98]. The GT is another modified version of the Atkinson index for use particularly in the evaluation of systems with repairable parts (for which an HPC application or system certainly qualifies – HPC system nodes can be replaced and HPC application processes can be rebooted). The GT index models the distribution in question as a Poisson process (PP) and measures that distribution's trend toward and away from a heterogeneous Poisson process (HPP) in time.

In using the GT index the 0-to-1 system utilized by the Failure Index is discarded in favor of a scheme containing values ranging from -1 to 1. An *improving* system, noted by values less than 0, denotes a system with a decreasing rate of occurrence of failure (ROCOF). Likewise, those values more than zero denote a *failing* system (one with increasing ROCOF). Values closer to 1 or -1 improve or fail more rapidly than values closer to 0. GT coefficients will allow us to examine changes in failure rate with respect to time, which when combined with FI coefficients will better describe the failure behavior of a given system. The GT index is defined as

$$GT(\, N(T)\,) \;=\; 1 - \frac{2\int_0^T N(t)dt}{T \times N(T)}\,. \tag{4.2}$$

Here, $N(T)$ represents the number of failure events occurring in the given time interval $[0, T]$. This is an important difference between the GT and Failure indices. The GT is constructed in accordance to the number of events in a given time interval, whereas the FI is constructed with respect to the cumulative impact of those failures. The GT is associated with failure *rates*, and the FI is associated with failure *volatility*. The GT says nothing about the impact of a given failure.

Recall that the GT index models the underlying distribution as a Poisson process. Here, an improving Poisson process is defined as one with decreasing ROCOF, while a deteriorating process has increasing ROCOF. A heterogeneous Poisson process has a constant ROCOF. Improving Poisson processes yield GT coefficients closer to -1, while deteriorating Poisson processes yield GT coefficients closer to 1.

In Chapter 5, GT coefficients will be calculated for all time intervals for which FI coefficients are generated. The source code for generating GT coefficients via the R statistical software package can be found in Appendix C.

CHAPTER 5

USING THE FAILURE INDEX: A CASE STUDY

5.1 Introduction

In this chapter we generate Failure Index coefficients for each of the 23 Los Alamos National Laboratory systems. A detailed description of this dataset can be found in the introduction to Chapter 3.

In this analysis, FI coefficients are generated with parameters $\epsilon = 0$, $\epsilon = 0.20$ and $\epsilon = 0.99$. Further, a GT coefficient is generated for each system using Equation (4.2). The GT coefficient represents the system's rate of occurrence of failure (ROCOF) with respect to time. Increases in ROCOF generate positive GT coefficients. Likewise, decreases in ROCOF generate negative GT coefficients. Coefficients closer to 1 or -1 contain higher increases or decreases in ROCOF than coefficients closer to 0.

In addition, this analysis plots the total downtime exhibited by the system versus the age of the system. Much like a Lorenz curve, a perfectly non-volatile system would generate a 45-degree line on such a graph. Failure Index coefficients capture fluctuations toward and away from this line.

Further, FI ($\epsilon = 0$) and GT coefficients are plotted in relation to the age of the system. This shows fluctuations in failure volatility and failure rate in time.

Long stretches of time without failure activity lower FI coefficients. Rapid and highly unequal failure activity increases FI coefficients. Section 4.1 reviews the information collected by the Failure Index relative to the information captured by inequality indices such as the Gini and Atkinson.

The following section displays all results from this case study. Tables 5.1 through 5.23 show the FI coefficients generated for each system using parameters 0, 0.20 and 0.99, in addition to the GT coefficients generated by each system. Figures 5.1 through 5.69 plot the total downtime exhibited by each system with respect to time, the FI coefficient efficient with respect to time and the GT coefficient with respect to time for each of the 23 systems. Tables 5.24 through 5.26 illustrate the number of LANL systems with increasing and decreasing ROCOF, the FI coefficient averages for those systems and the FI coefficient averages for all 23 systems. Section 5.3 will draw conclusions from these results.

5.2 Results

Table 5.1 FI Coefficients and ROCOF, System 2, LANL Data

System	FI ($\varepsilon = 0$)	FI ($\varepsilon = 0.20$)	FI ($\varepsilon = 0.99$)	GT (N(T))
2	0.3488647	0.3739617	0.560606	-0.2452266

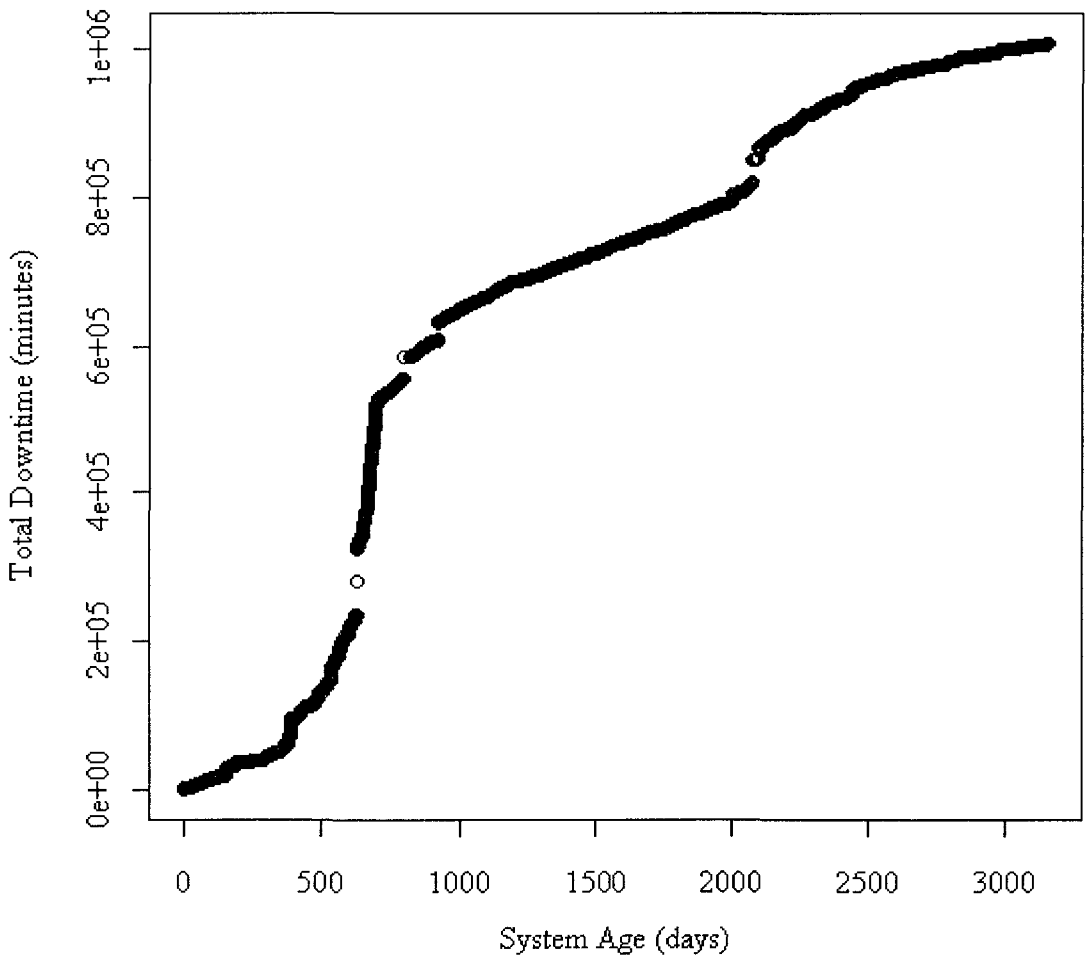


Figure 5.1 Total Downtime vs. System Age, System 2, LANL Data

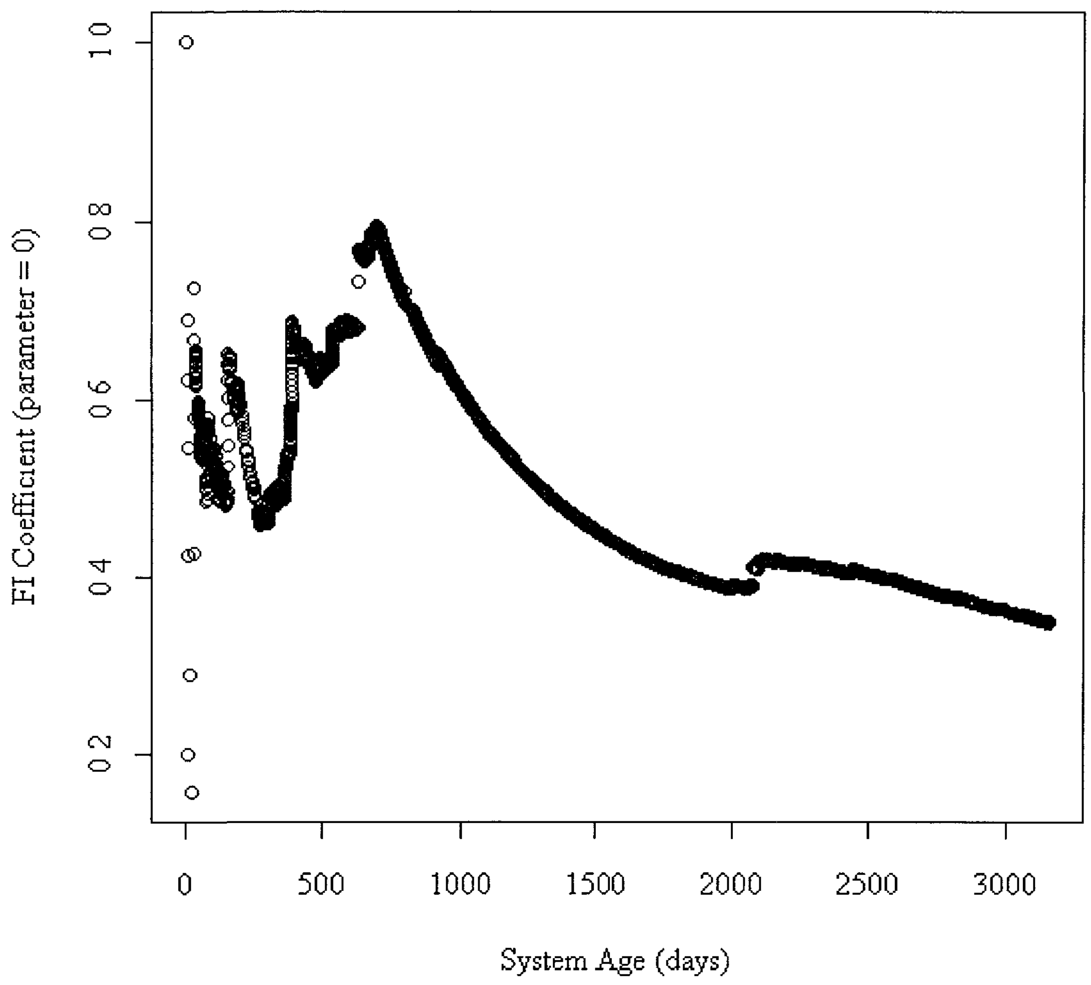


Figure 5.2 FI Coefficient vs. System Age, System 2, LANL Data

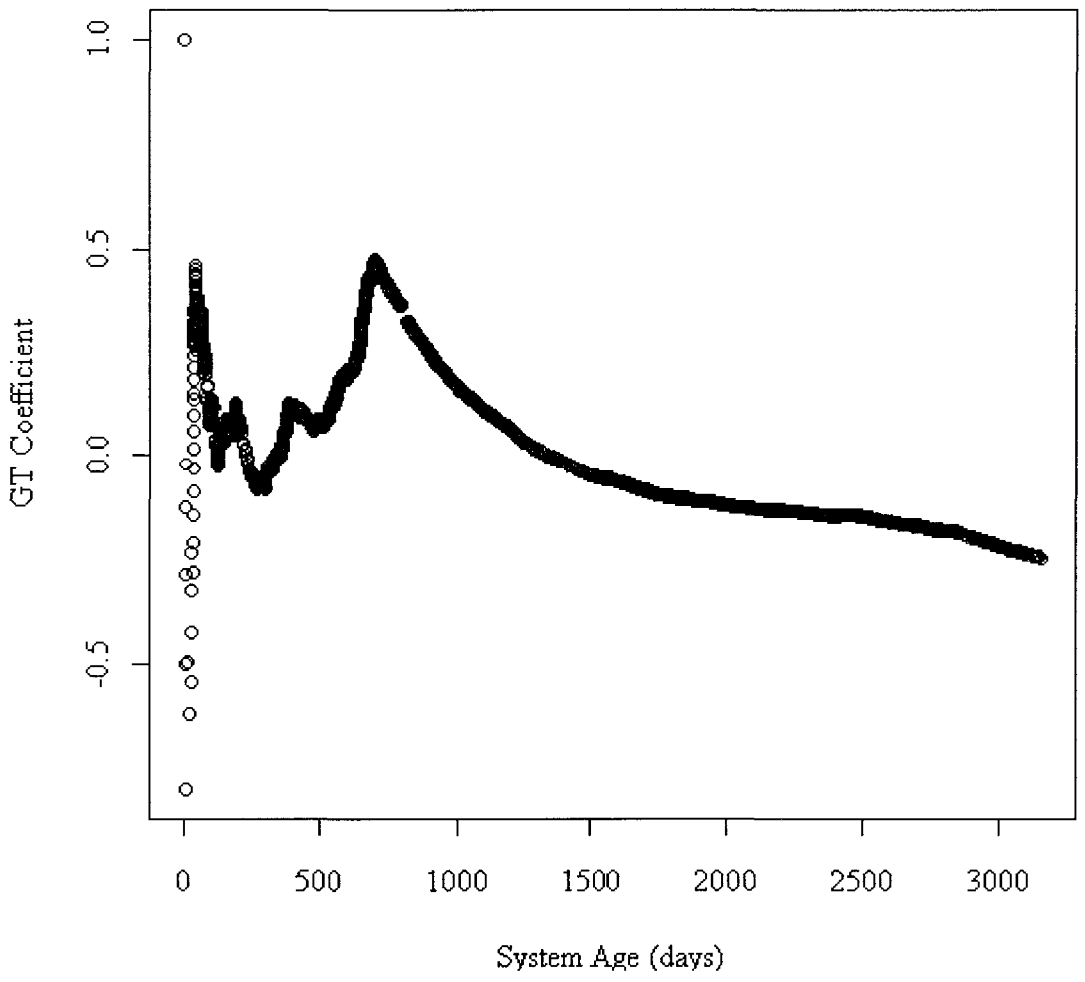


Figure 5.3 ROCOF vs. System Age, System 2, LANL Data

Table 5.2 FI Coefficients and ROCOF, System 3, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
3	0.4762693	0.4928603	0.6238533	-0.1400224

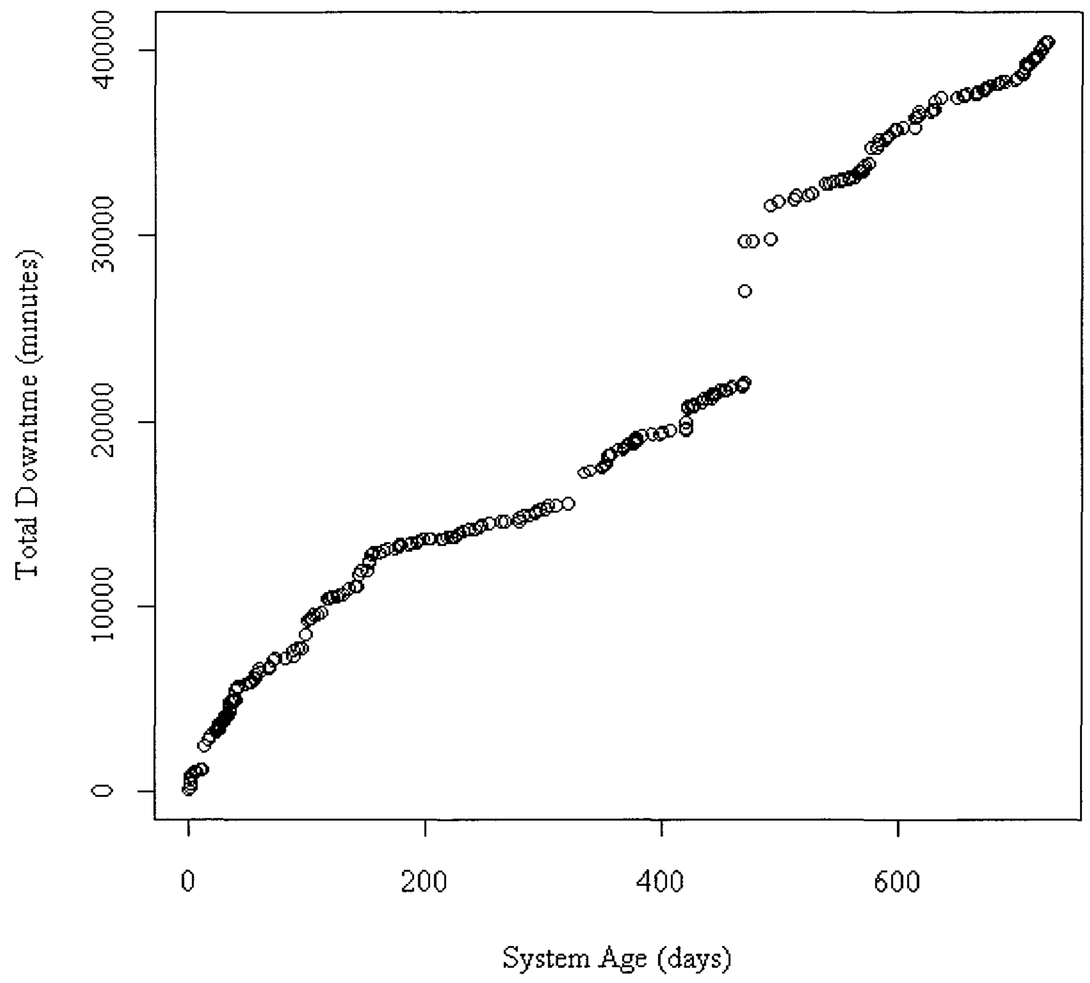


Figure 5.4 Total Downtime vs. System Age, System 3, LANL Data

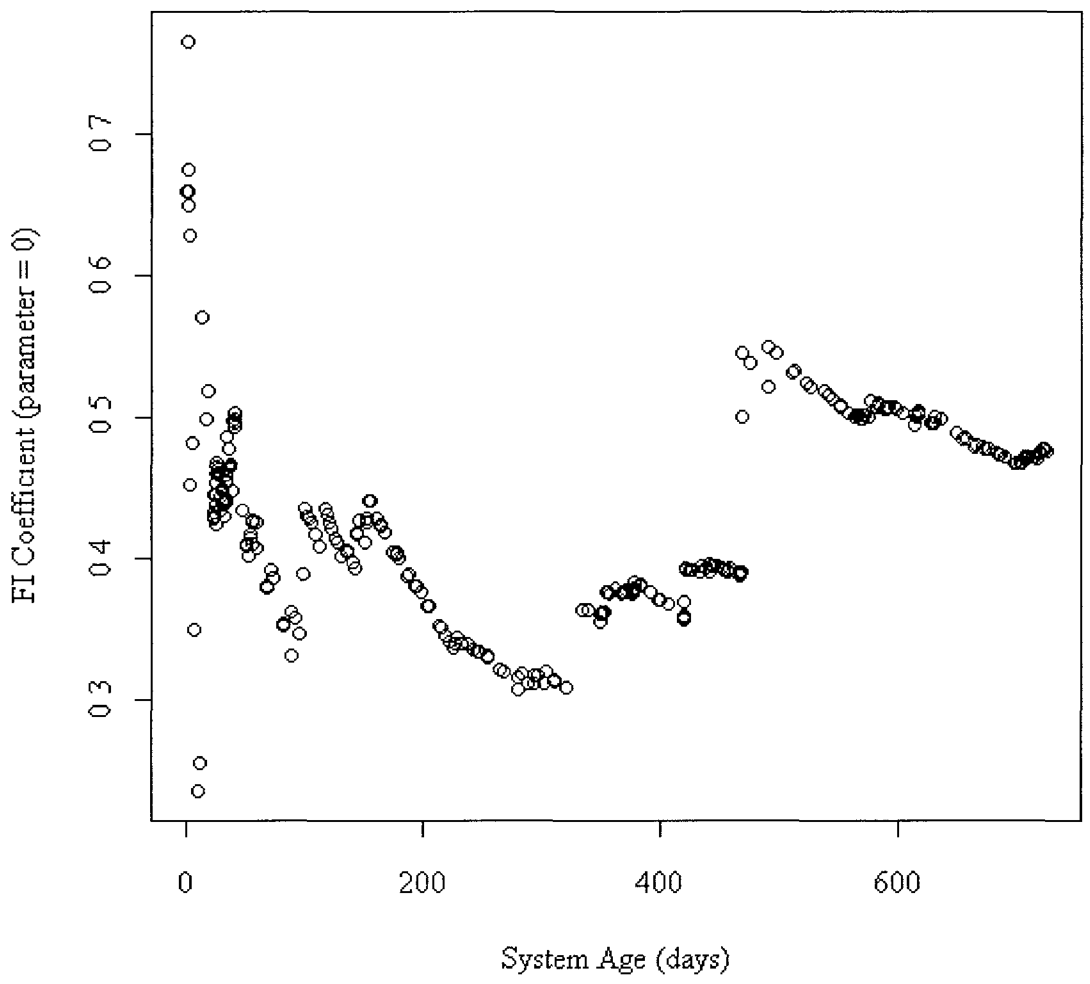


Figure 5.5 FI Coefficient vs. System Age, System 3, LANL Data

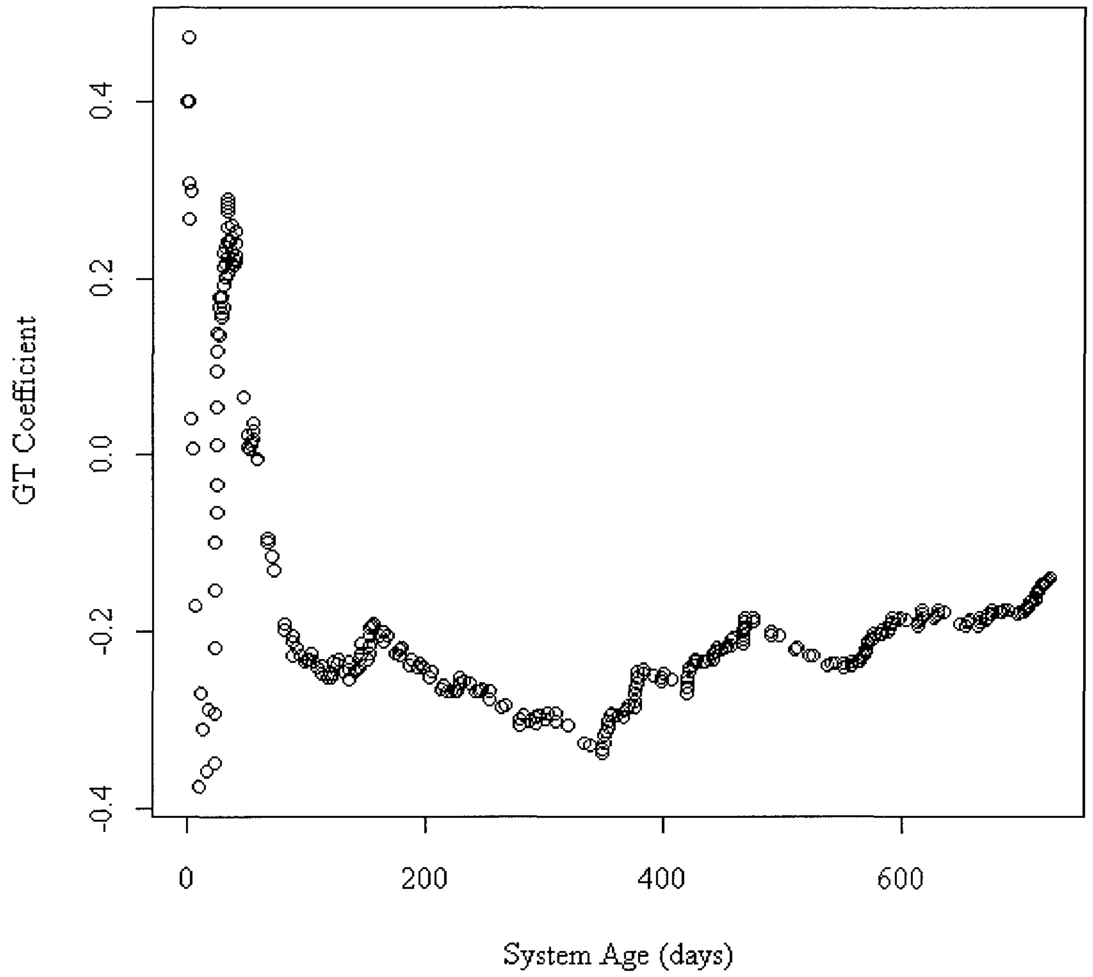


Figure 5.6 ROCOF vs. System Age, System 3, LANL Data

Table 5.3 FI Coefficients and ROCOF, System 4, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
4	0.5312802	0.5510522	0.6861621	-0.1069729

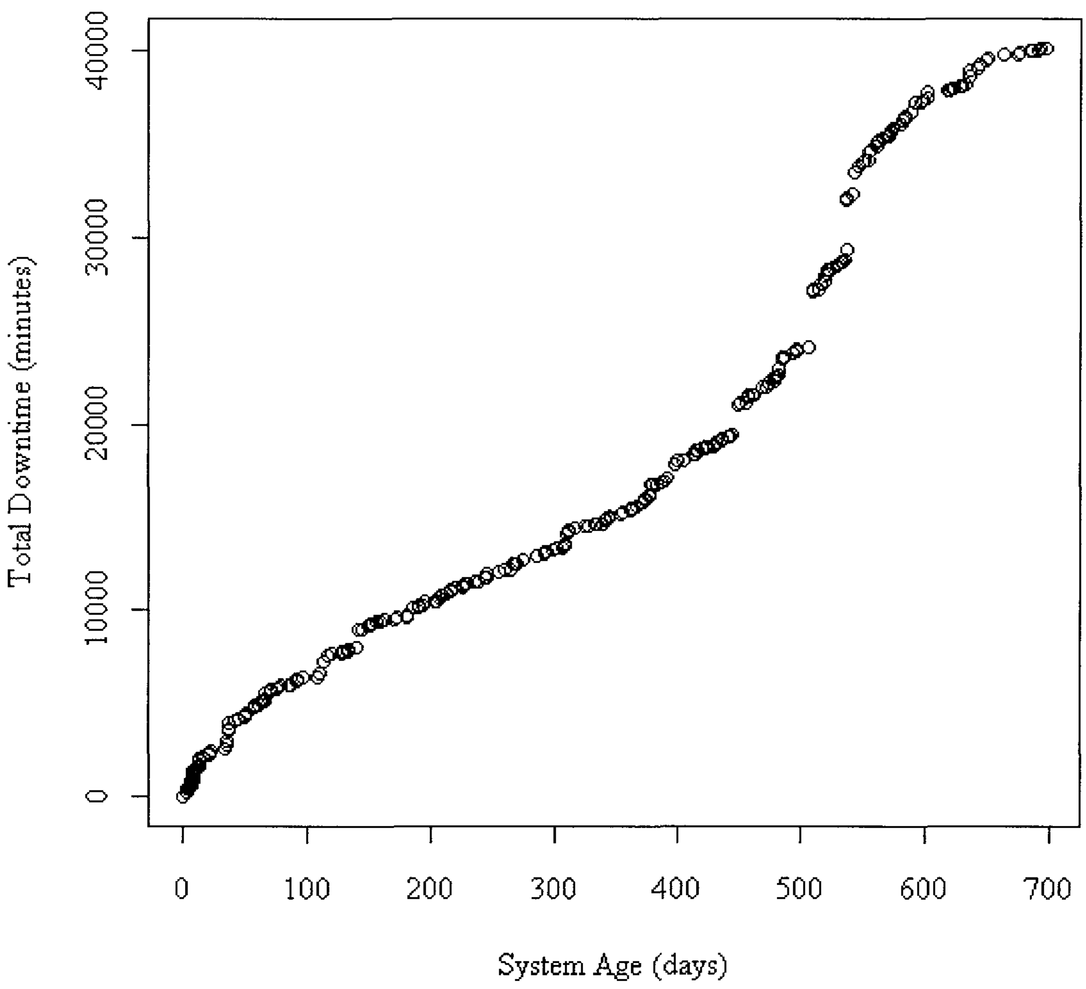


Figure 5.7 Total Downtime vs. System Age, System 4, LANL Data

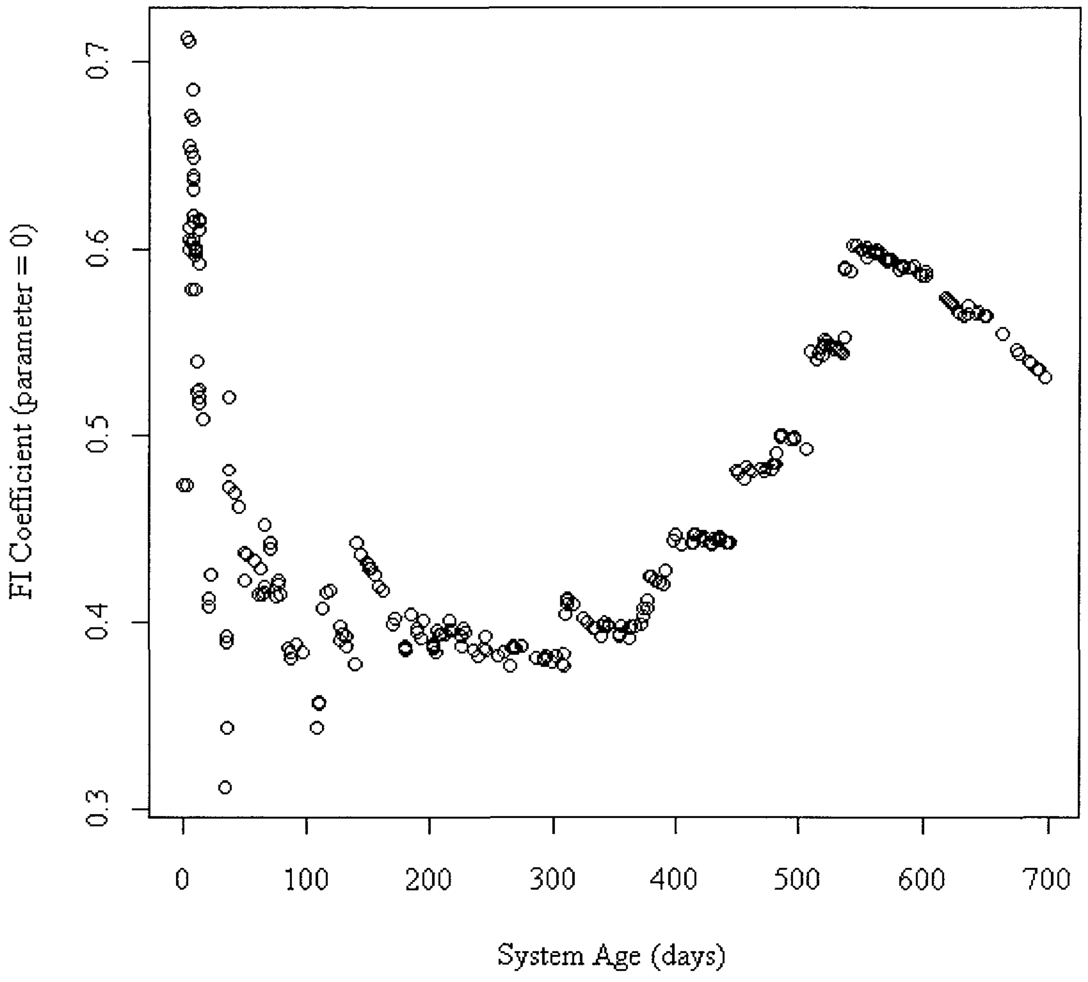


Figure 5.8 FI Coefficient vs. System Age, System 4, LANL Data

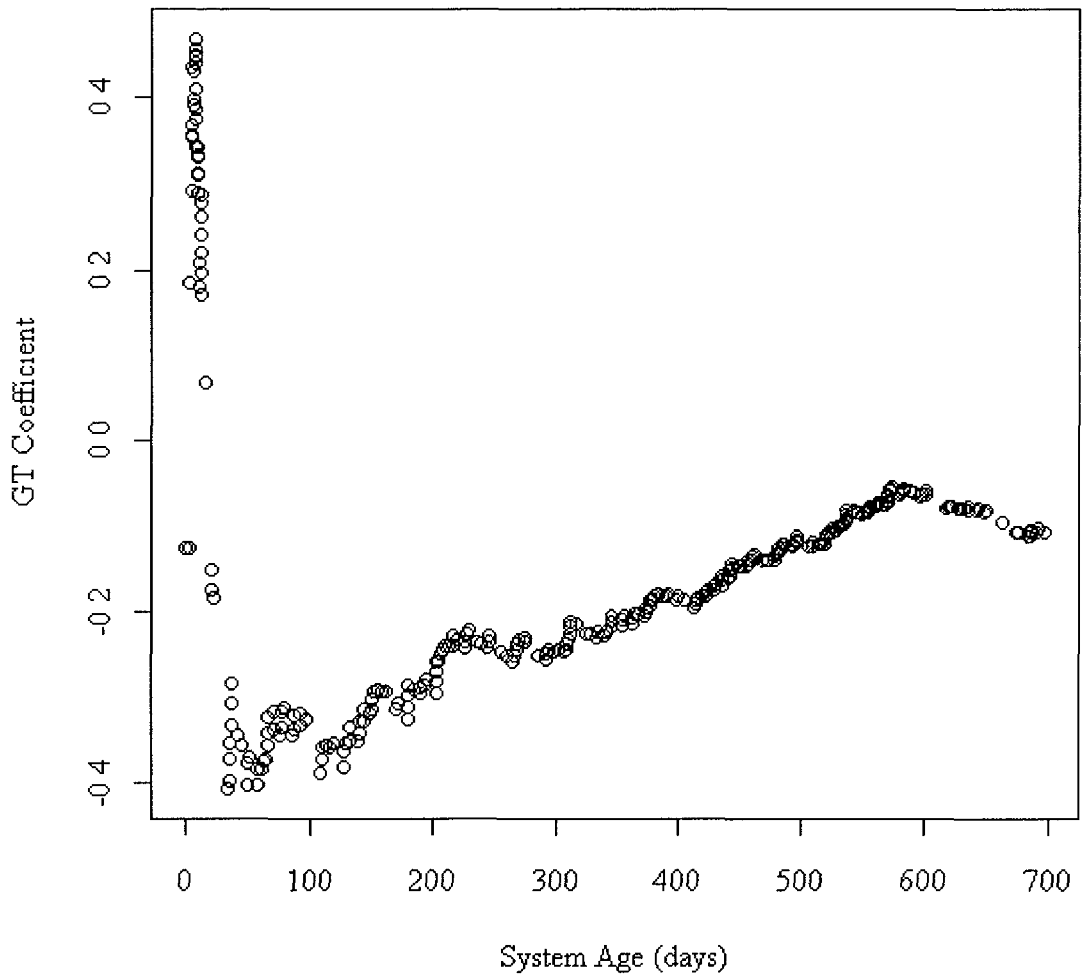


Figure 5.9 ROCOF vs. System Age, System 4, LANL Data

Table 5.4 FI Coefficients and ROCOF, System 5, LANL Data

System	FI ($\varepsilon = 0$)	FI ($\varepsilon = 0.20$)	FI ($\varepsilon = 0.99$)	GT (N(T))
5	0.4517466	0.4683794	0.6250355	-0.05701952

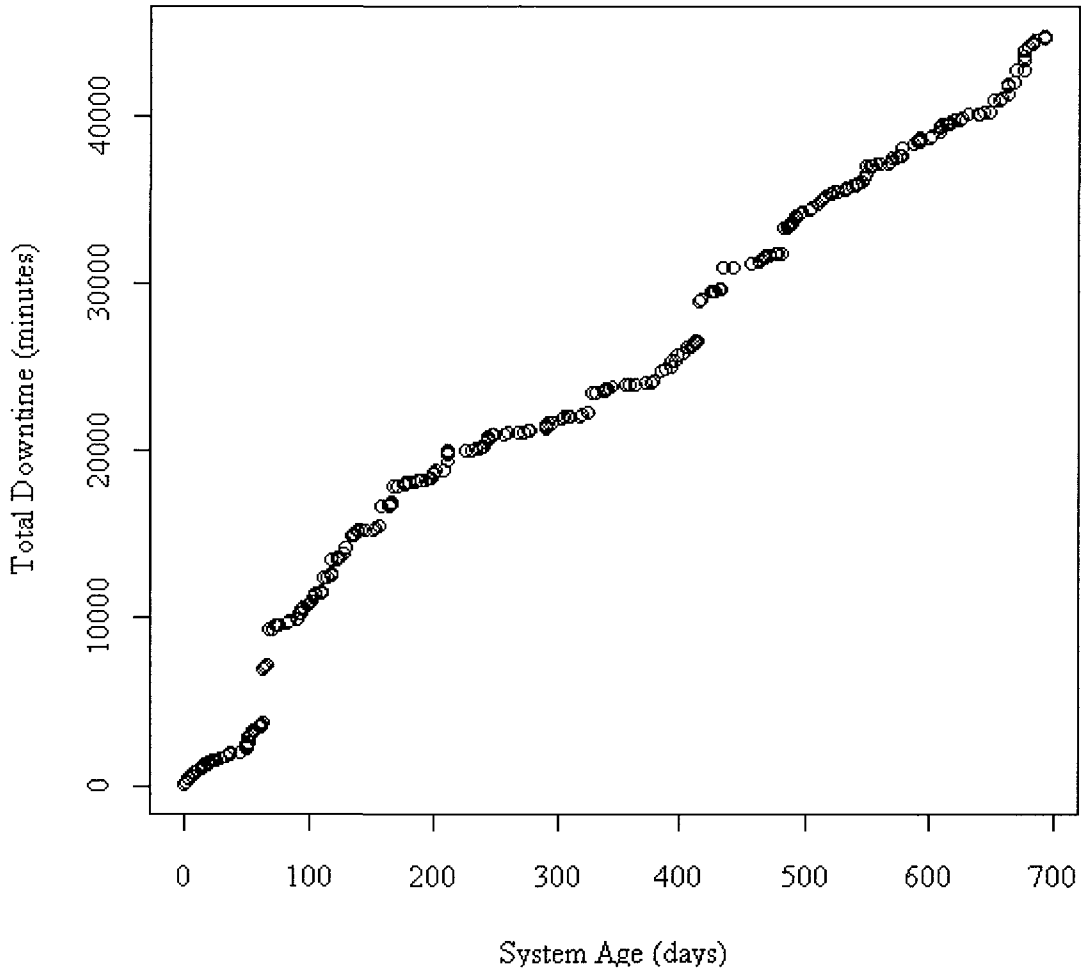


Figure 5.10 Total Downtime vs. System Age, System 5, LANL Data

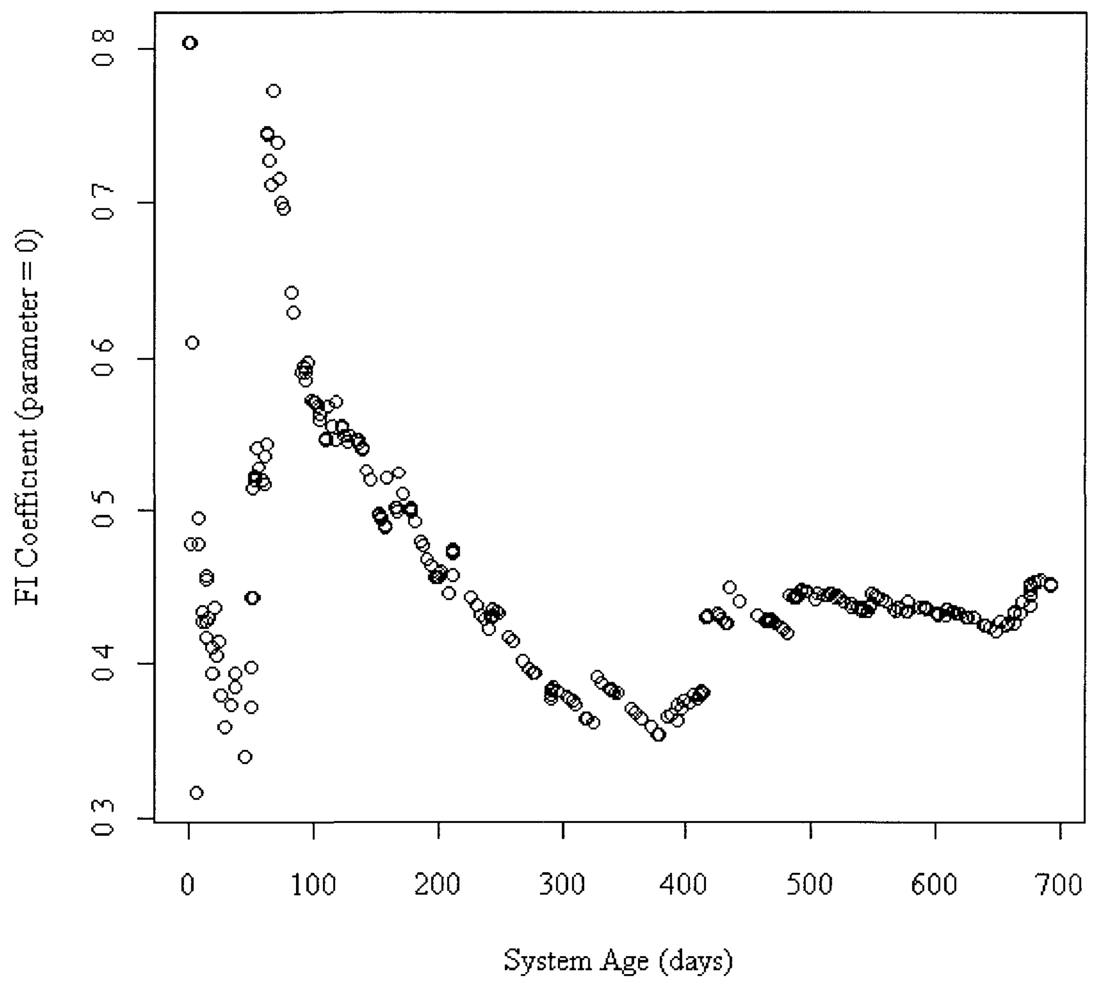


Figure 5.11 FI Coefficient vs. System Age, System 5, LANL Data

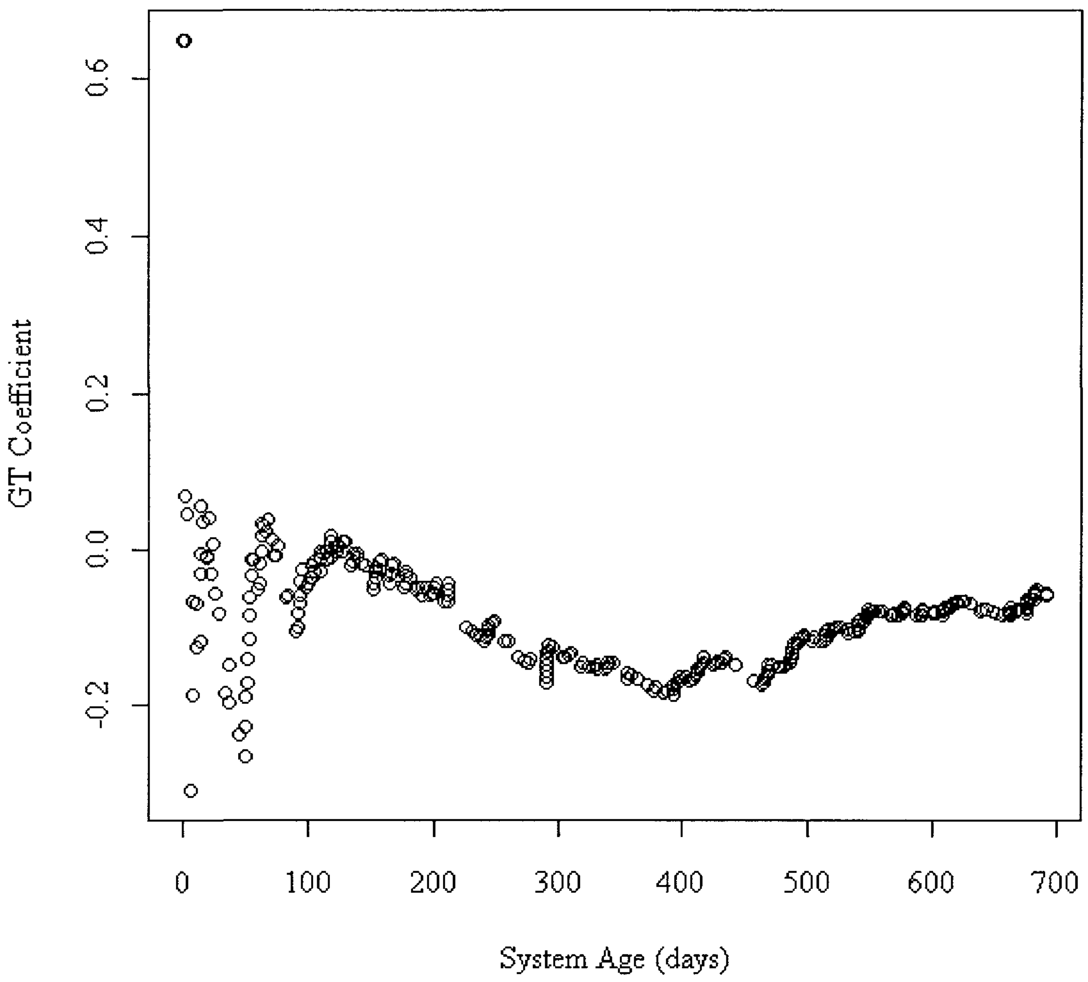


Figure 5.12 ROCOF vs. System Age, System 5, LANL Data

Table 5.5 FI Coefficients and ROCOF, System 6, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
6	0.4661334	0.4750499	0.5761766	-0.009286395

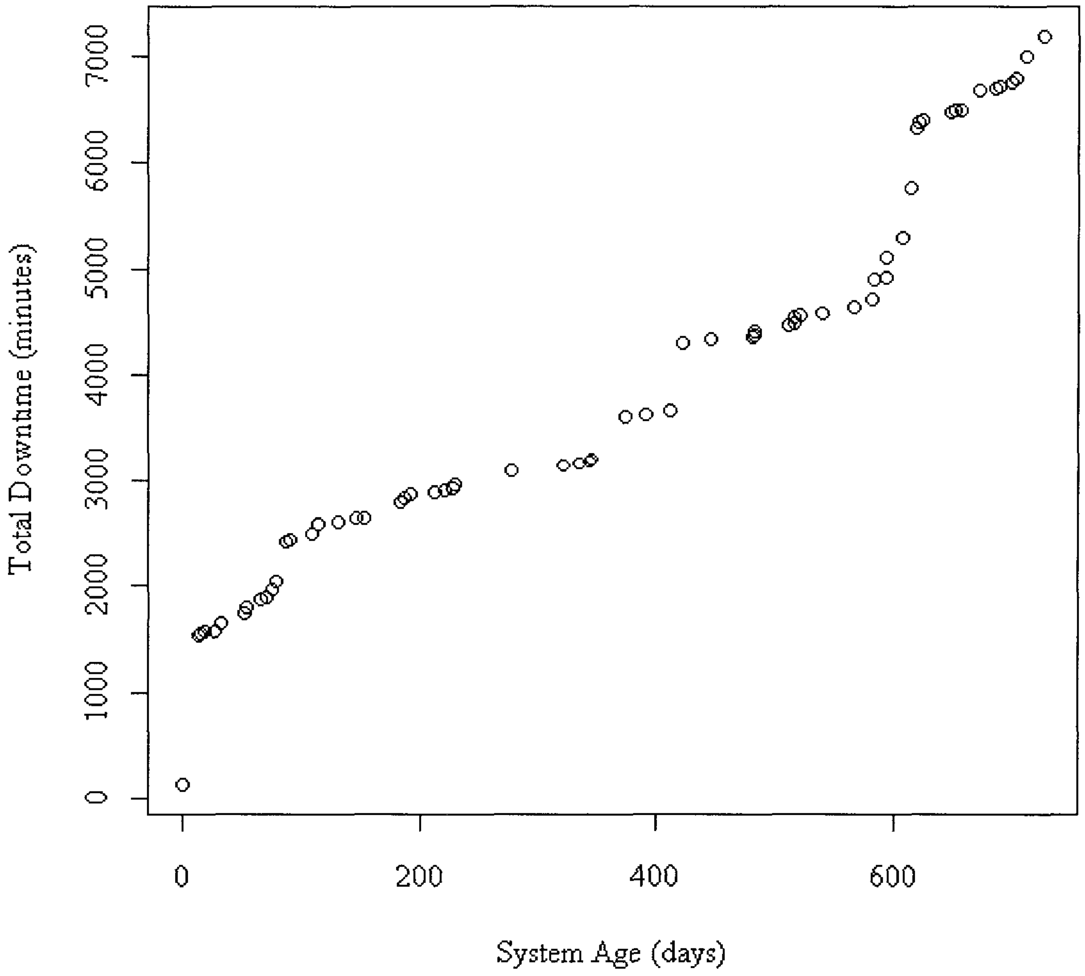


Figure 5.13 Total Downtime vs. System Age, System 6, LANL Data

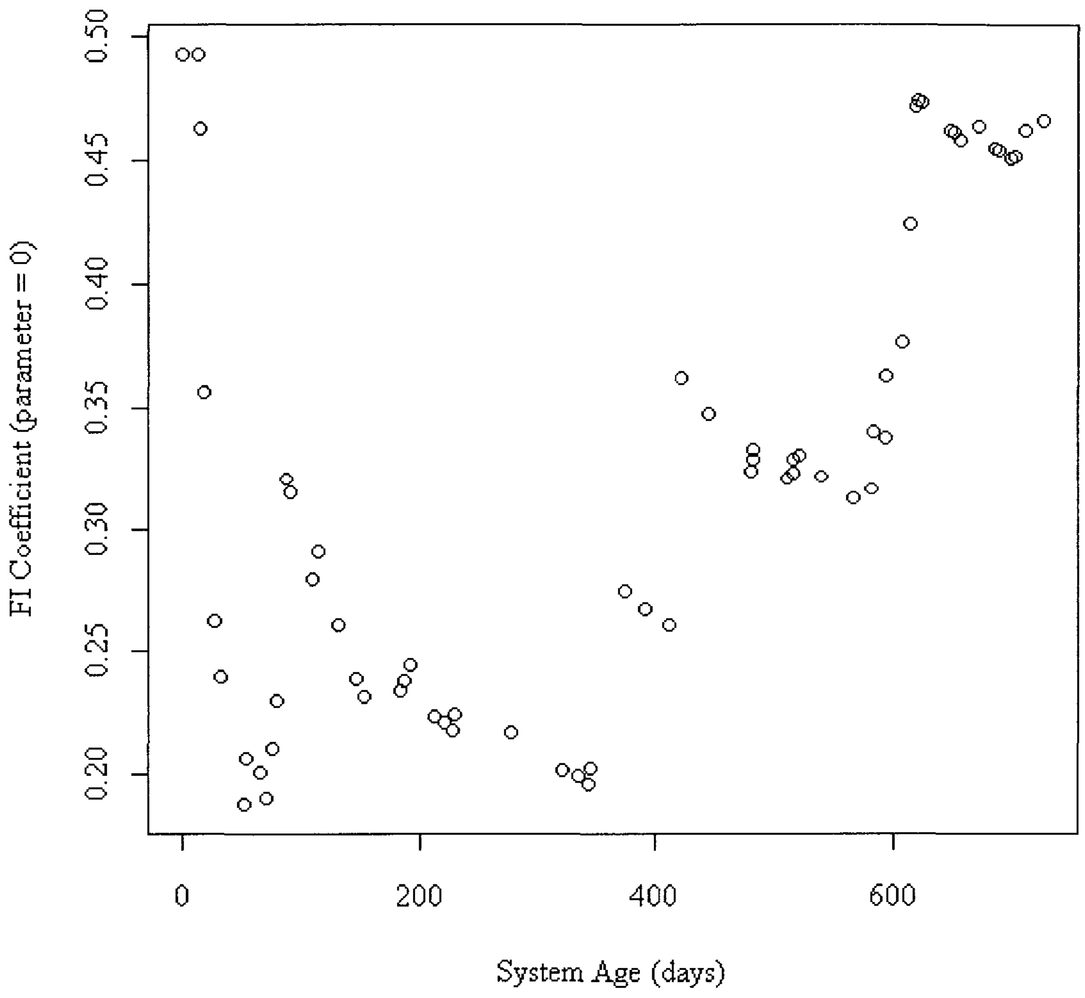


Figure 5.14 FI Coefficient vs. System Age, System 6, LANL Data

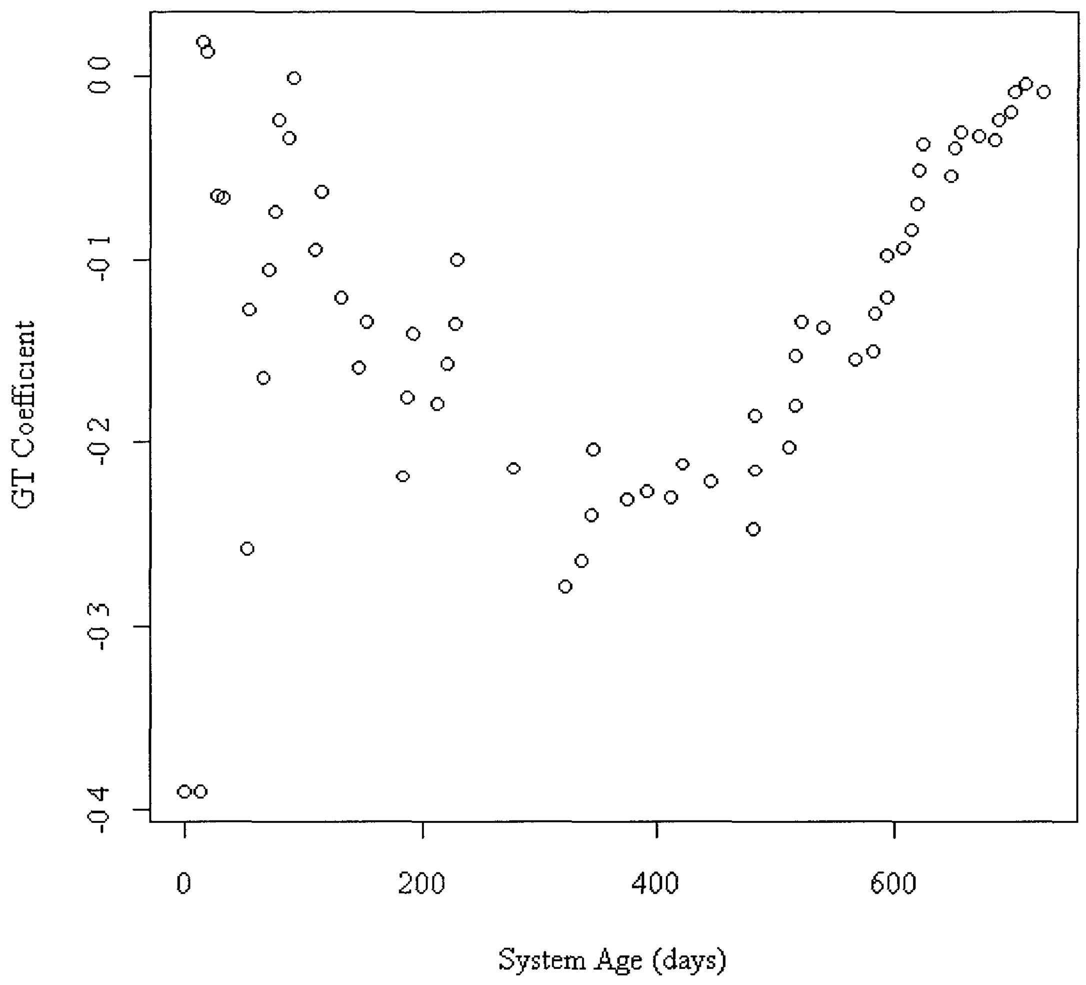


Figure 5.15 ROCOF vs. System Age, System 6, LANL Data

Table 5.6 FI Coefficients and ROCOF, System 7, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
7	0.5590928	0.5754688	0.6686179	-0.2554706

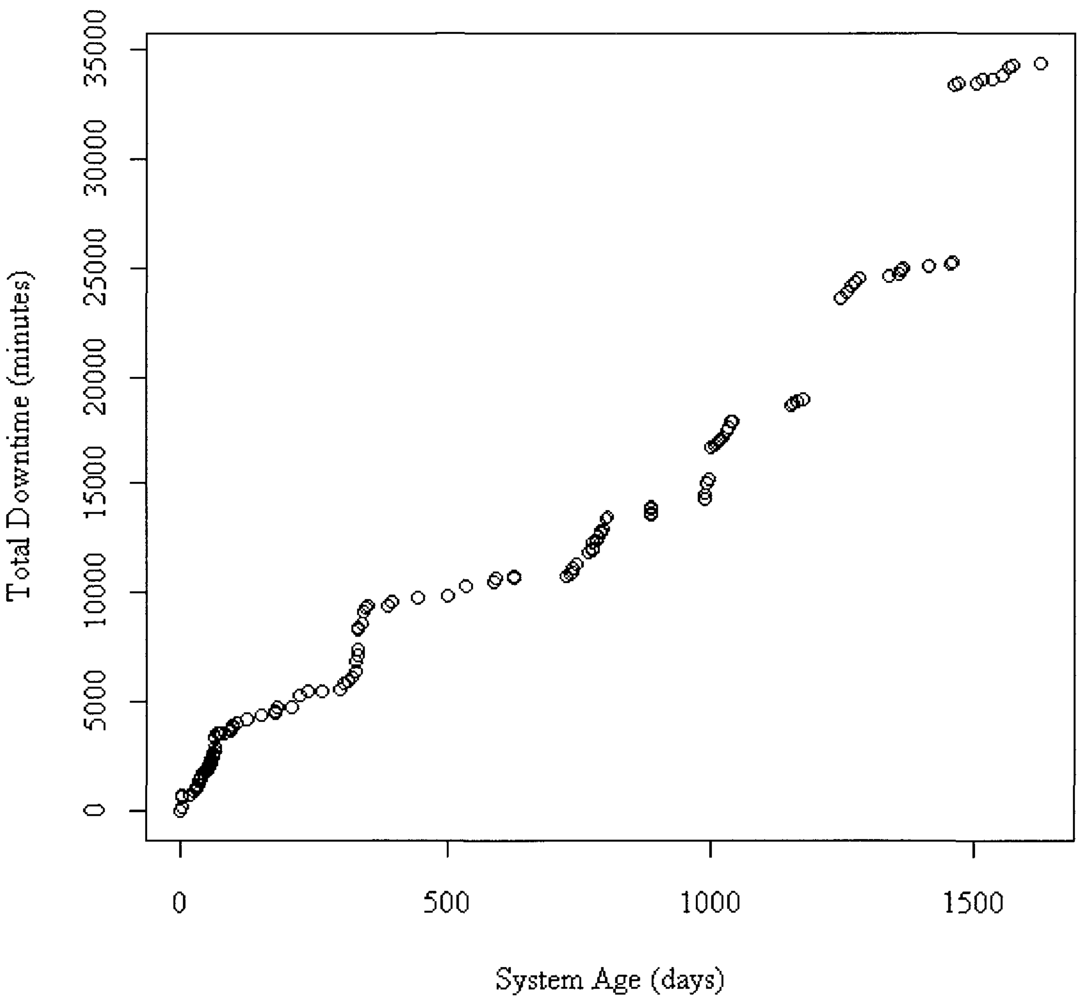


Figure 5.16 Total Downtime vs. System Age, System 7, LANL Data

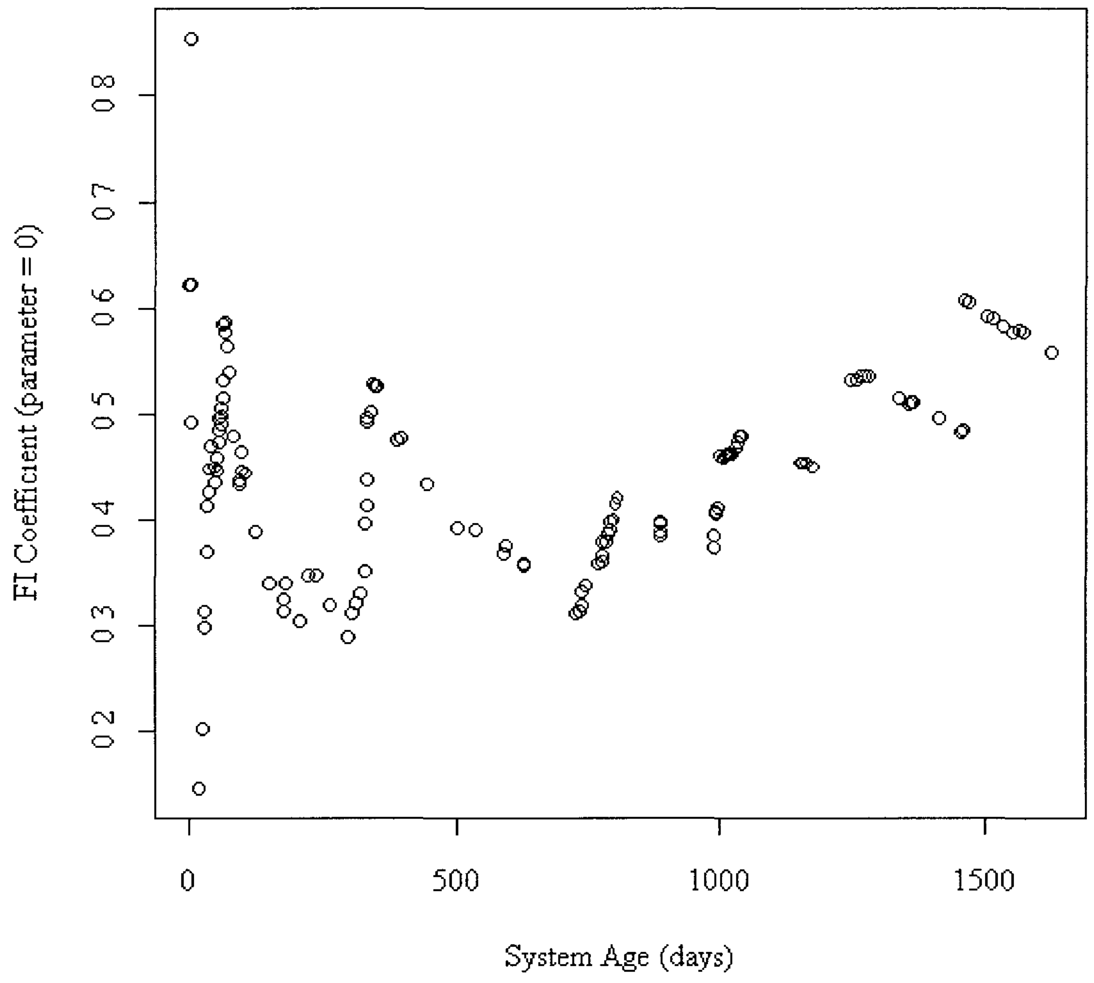


Figure 5.17 FI Coefficient vs. System Age, System 7, LANL Data

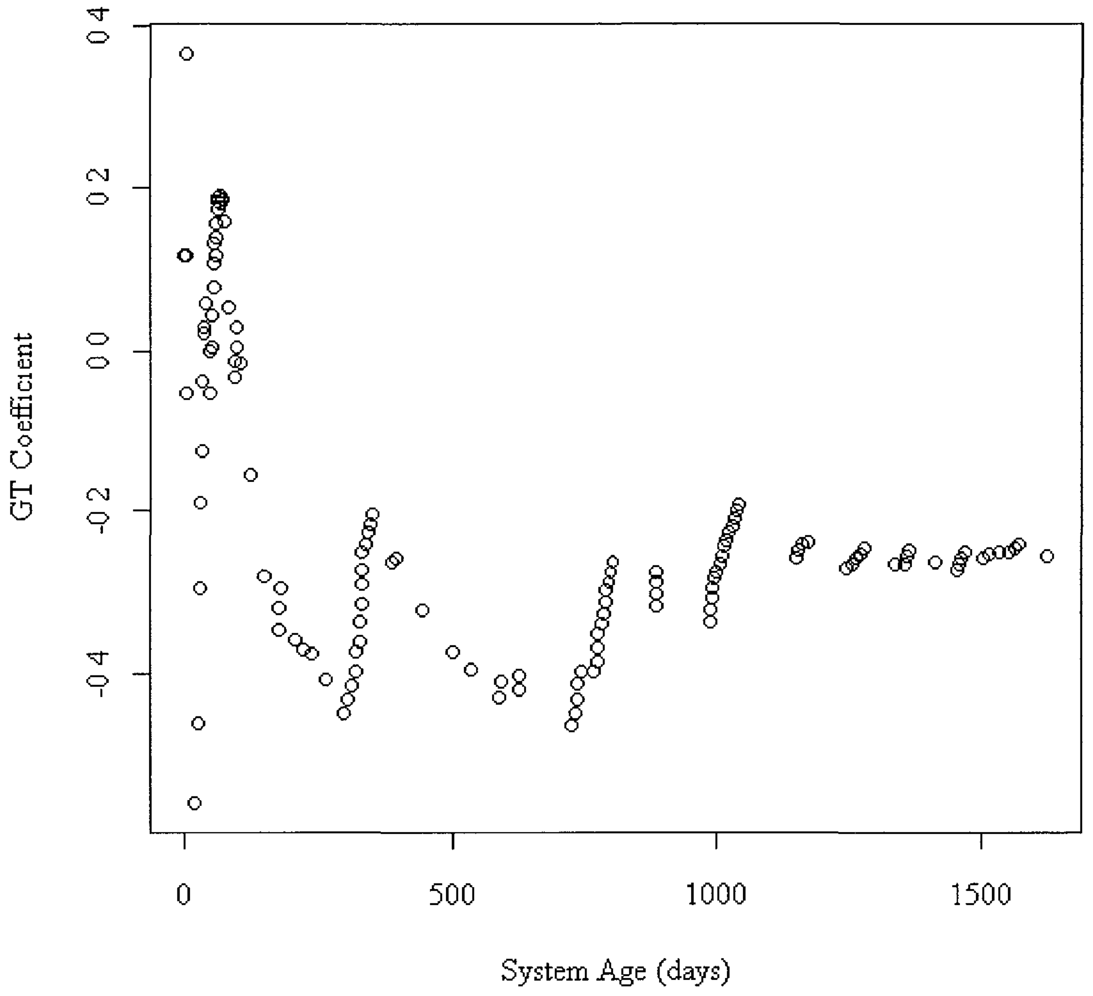


Figure 5.18 ROCOF vs. System Age, System 7, LANL Data

Table 5.7 FI Coefficients and ROCOF, System 8, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
8	0.5743319	0.5965732	0.7276079	0.1277645

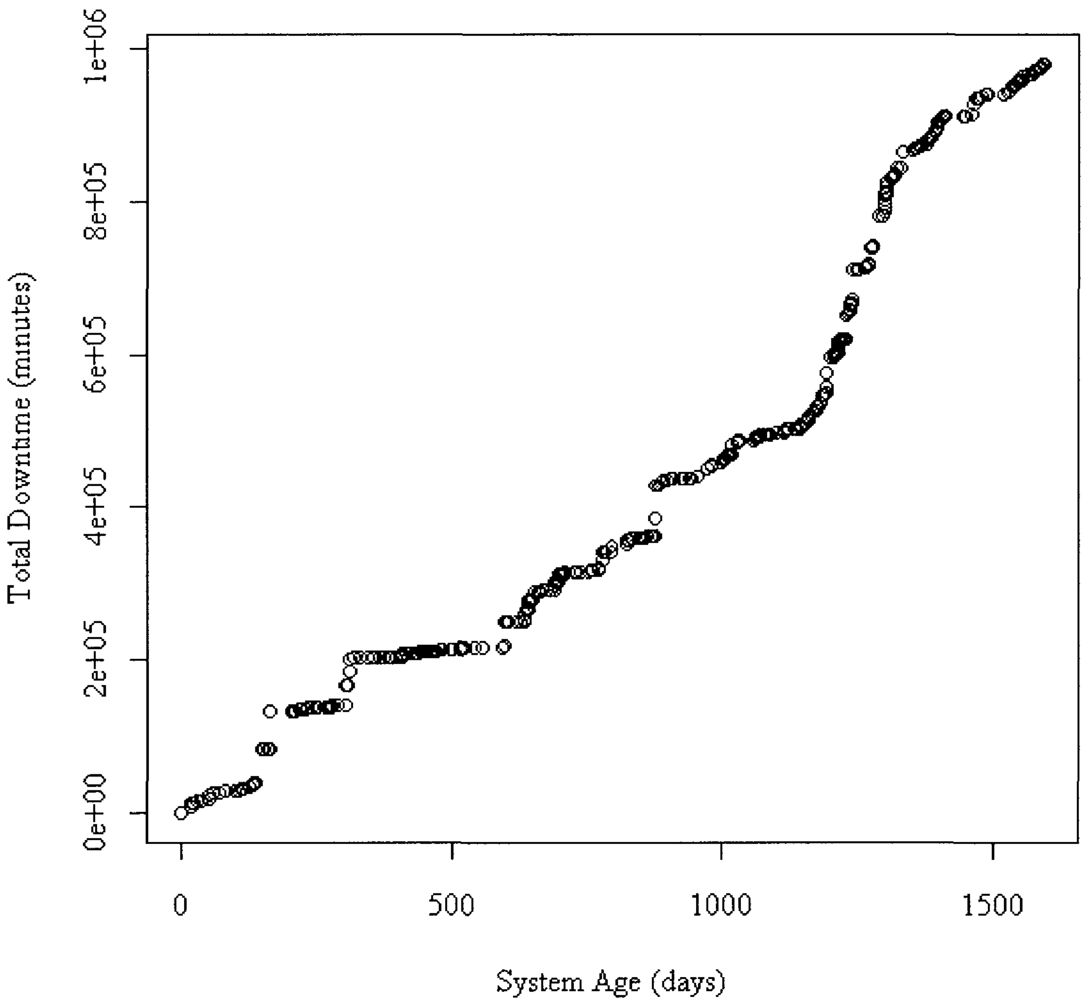


Figure 5.19 Total Downtime vs. System Age, System 8, LANL Data

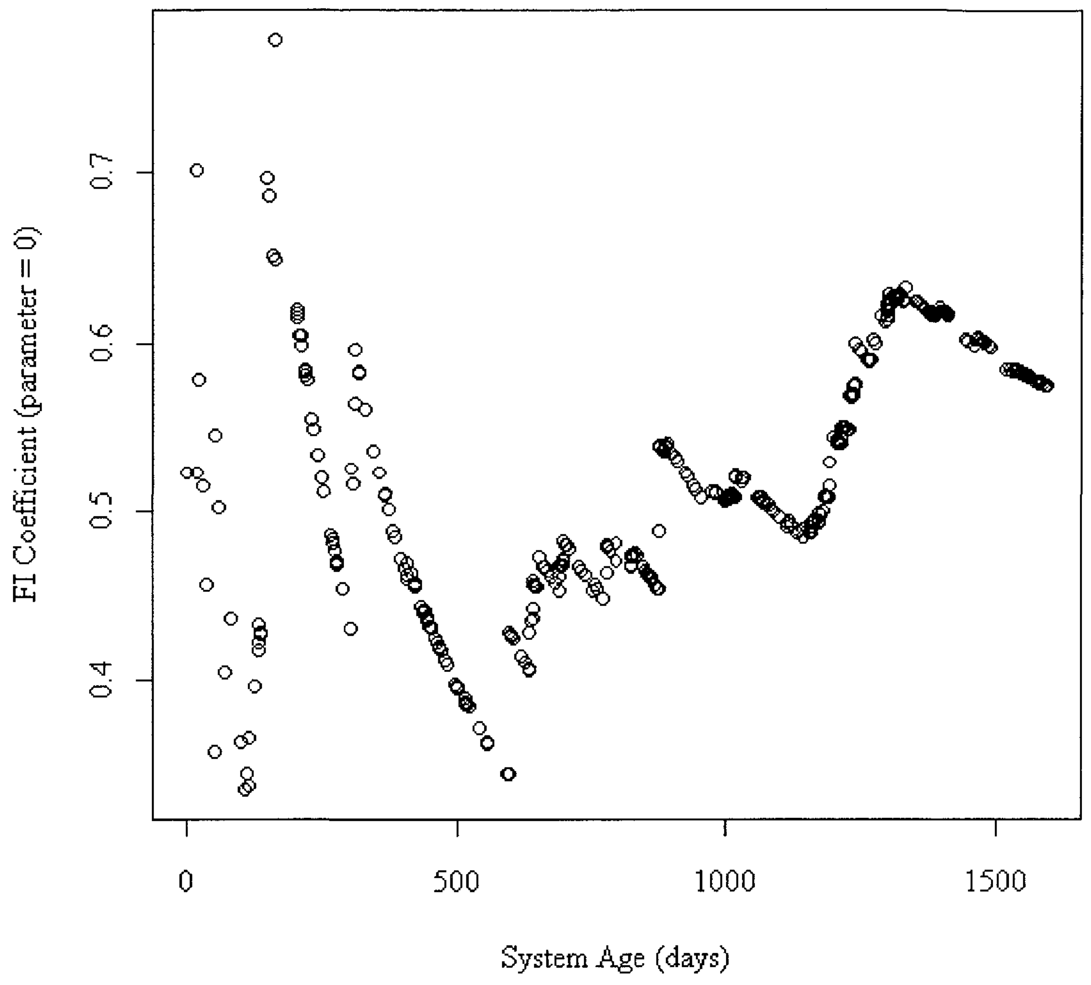


Figure 5.20 FI Coefficient vs. System Age, System 8, LANL Data

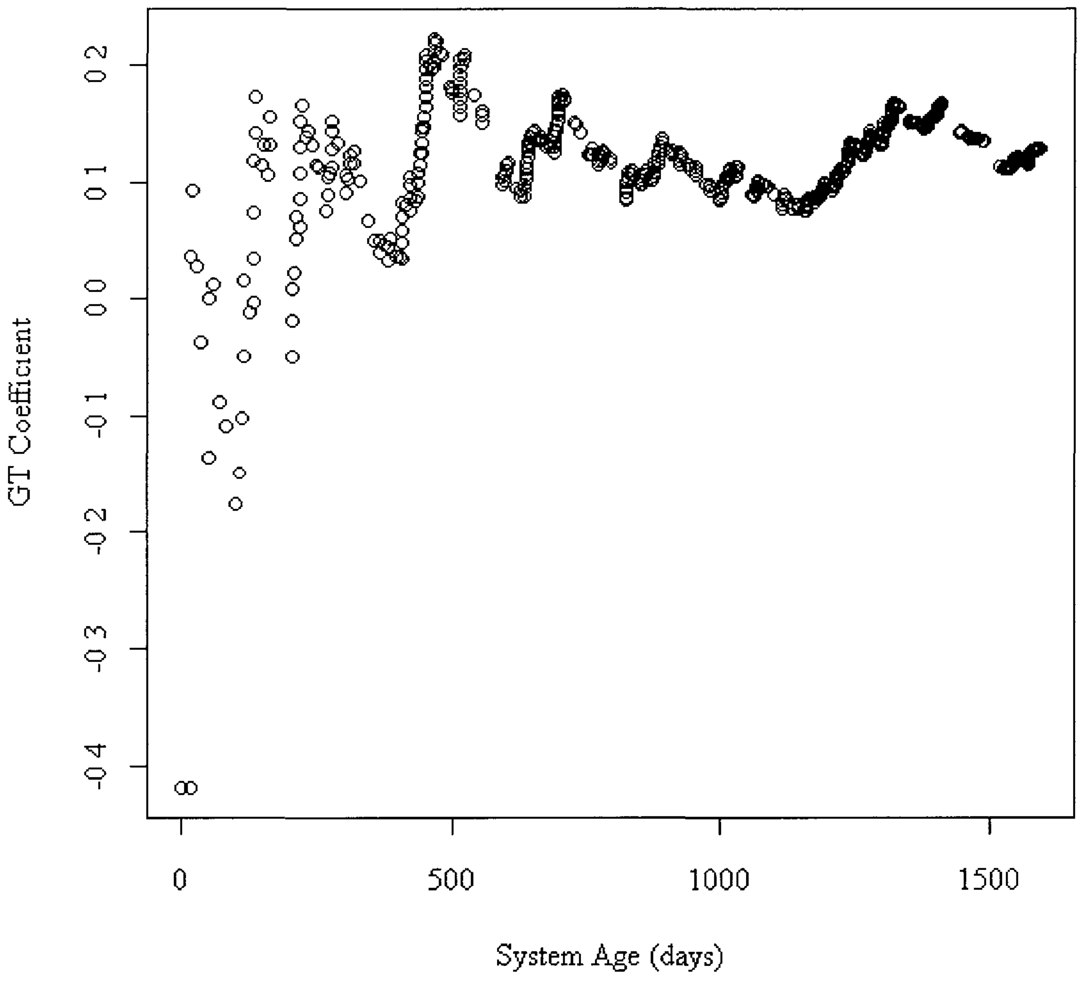


Figure 5.21 ROCOF vs. System Age, System 8, LANL Data

Table 5.8 FI Coefficients and ROCOF, System 9, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
9	0.3801783	0.4195316	0.7901992	0.4110598

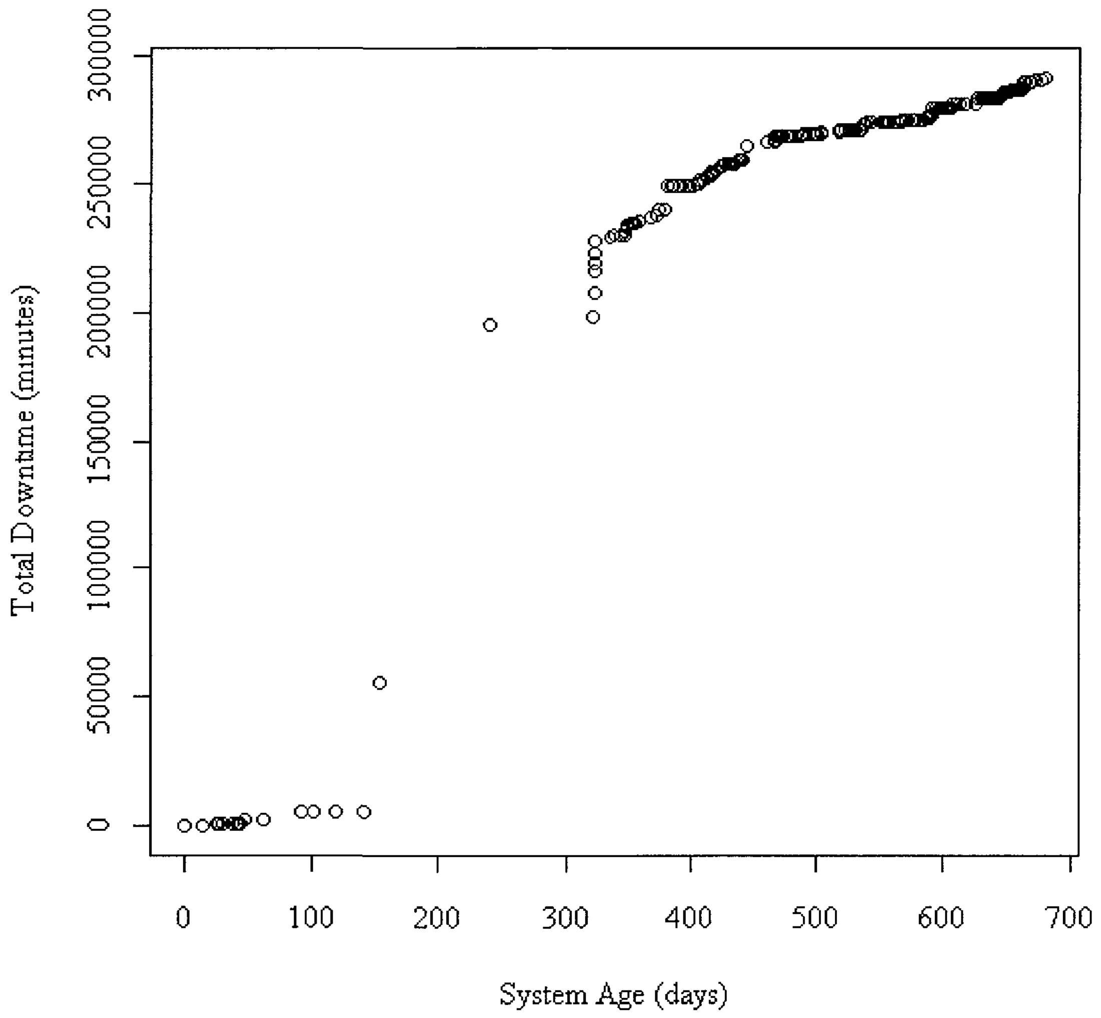


Figure 5.22 Total Downtime vs. System Age, System 9, LANL Data

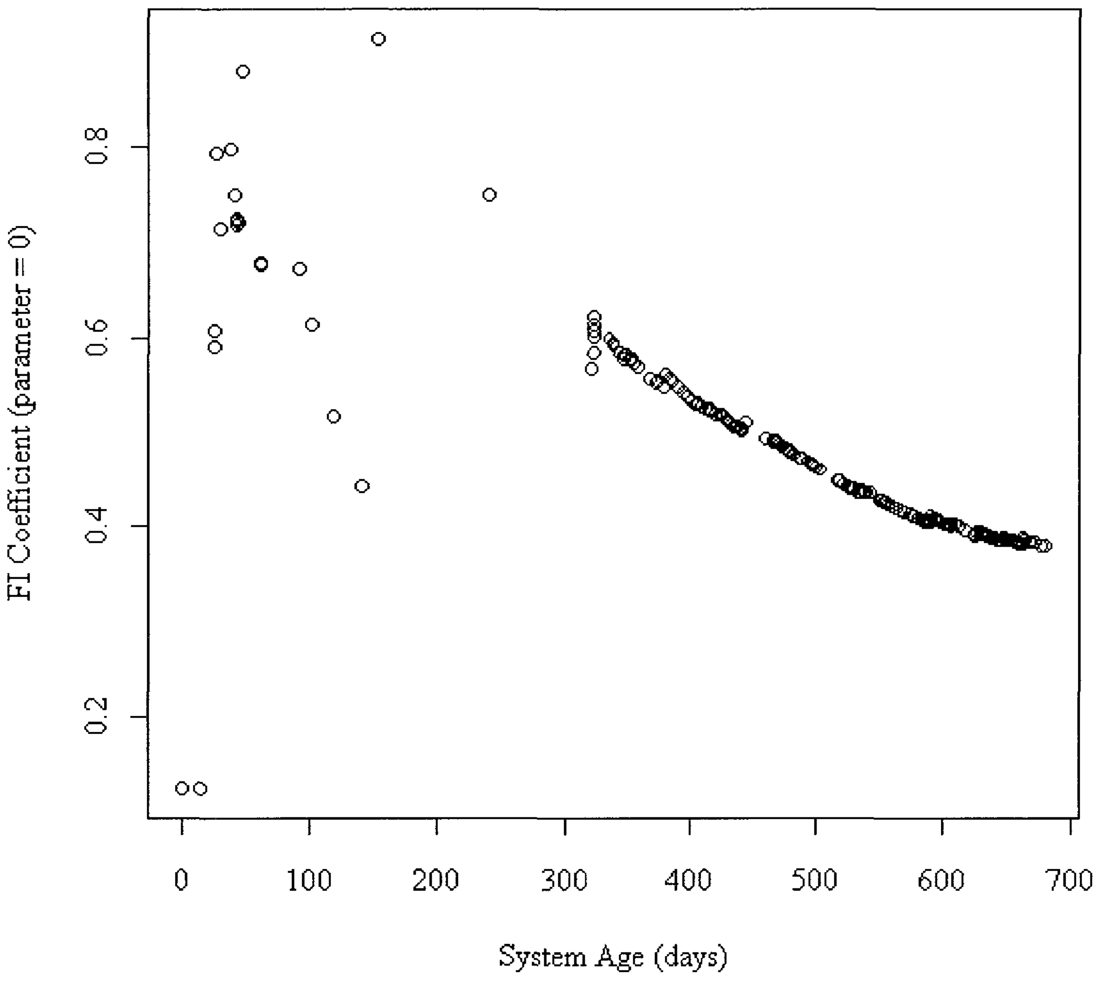


Figure 5.23 FI Coefficient vs. System Age, System 9, LANL Data

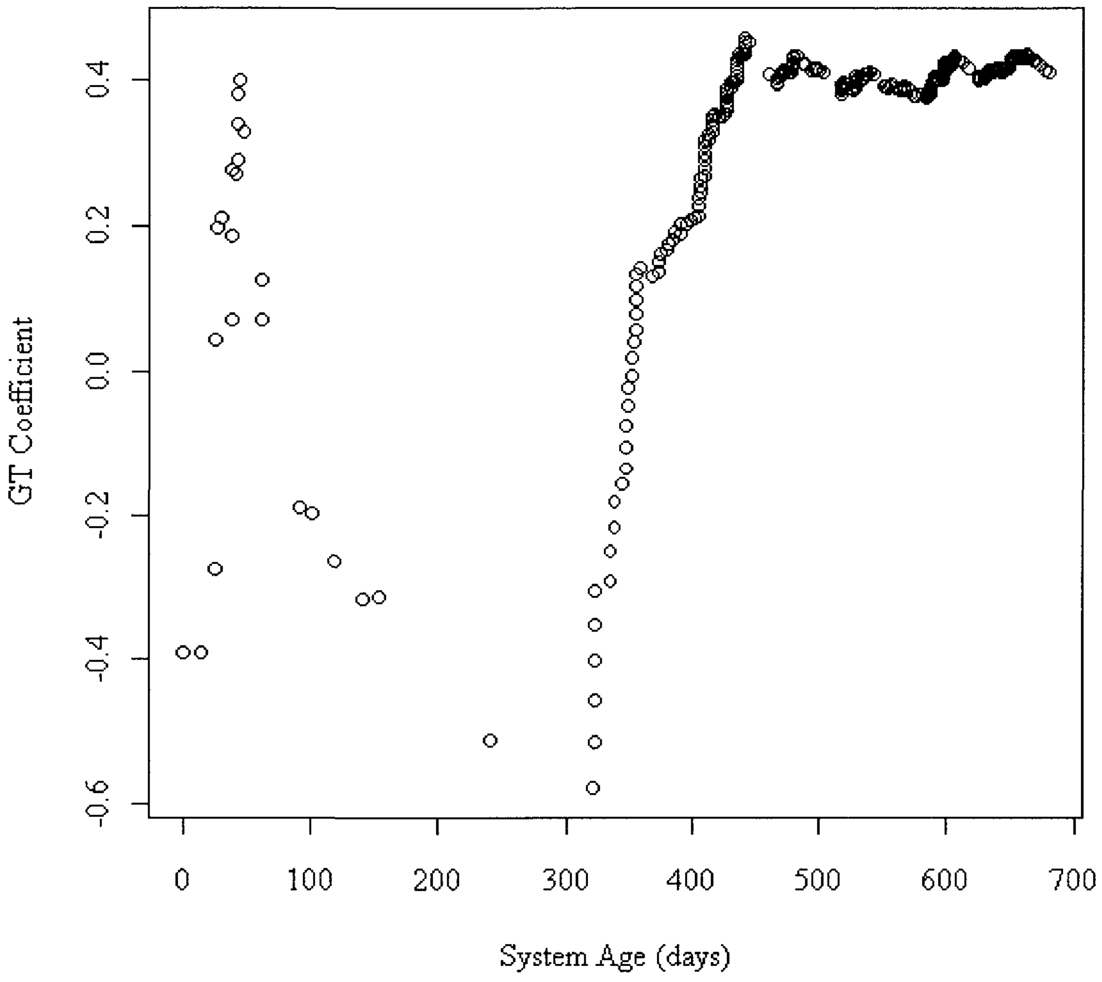


Figure 5.24 ROCOF vs. System Age, System 9, LANL Data

Table 5.9 FI Coefficients and ROCOF, System 10, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
10	0.4393362	0.4668898	0.6757968	0.3073982

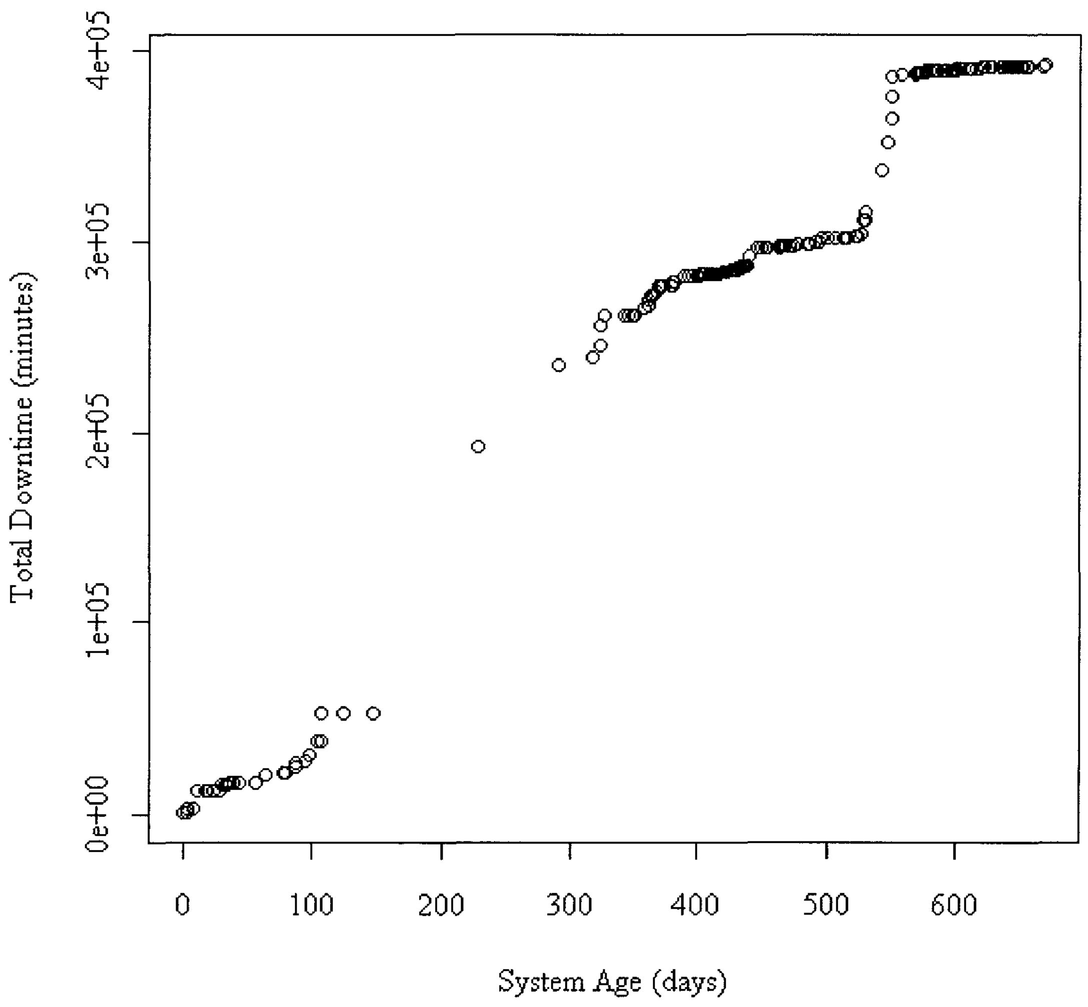


Figure 5.25 Total Downtime vs. System Age, System 10, LANL Data

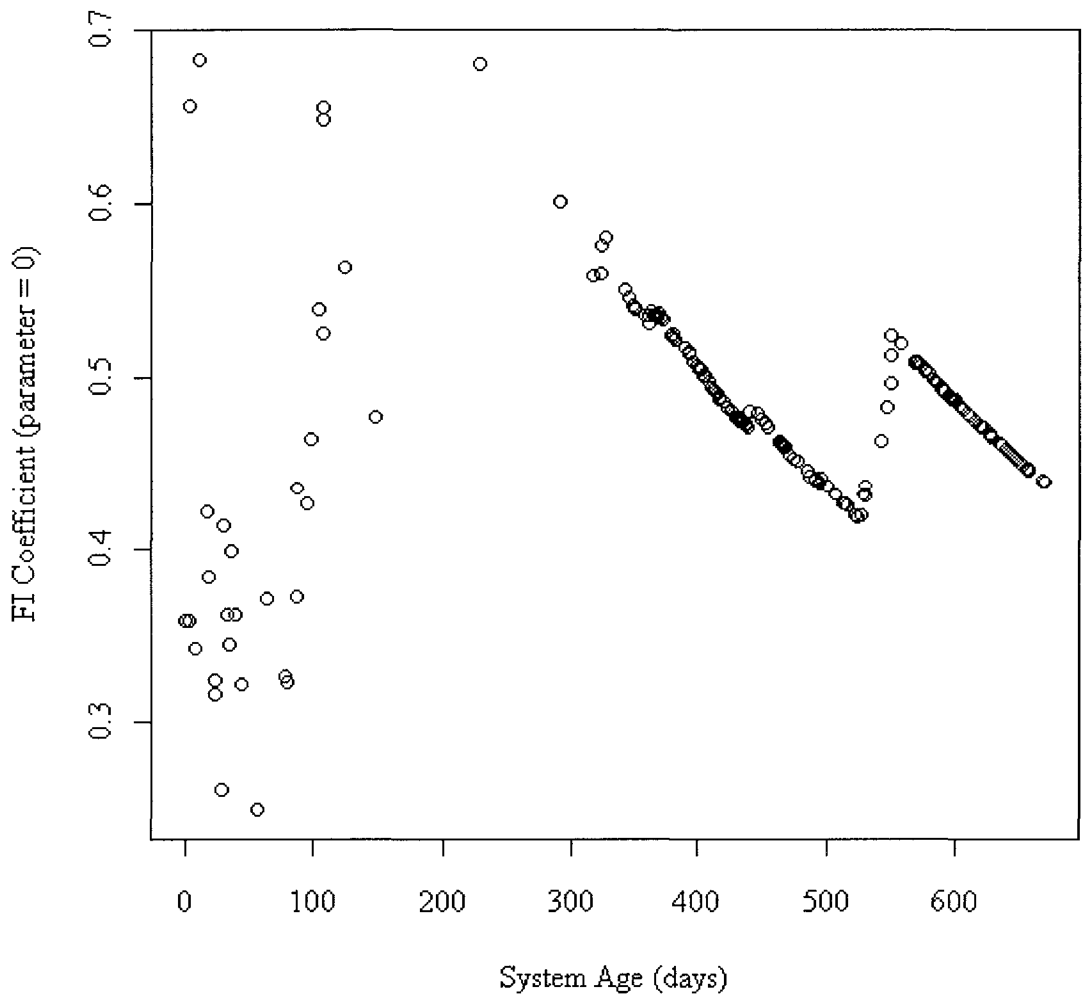


Figure 5.26 FI Coefficient vs. System Age, System 10, LANL Data

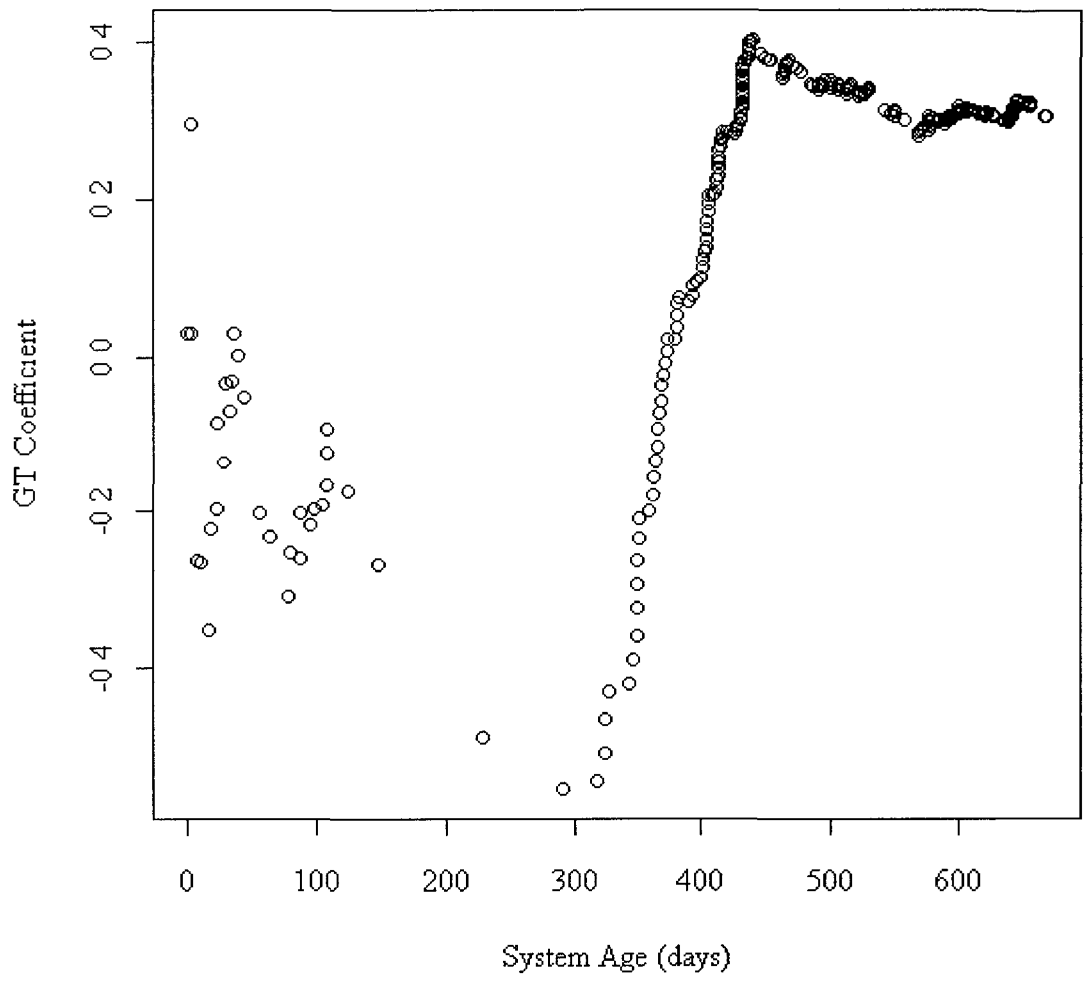


Figure 5.27 ROCOF vs. System Age, System 10, LANL Data

Table 5.10 FI Coefficients and ROCOF, System 11, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
11	0.3228945	0.3503598	0.6125574	0.3016312

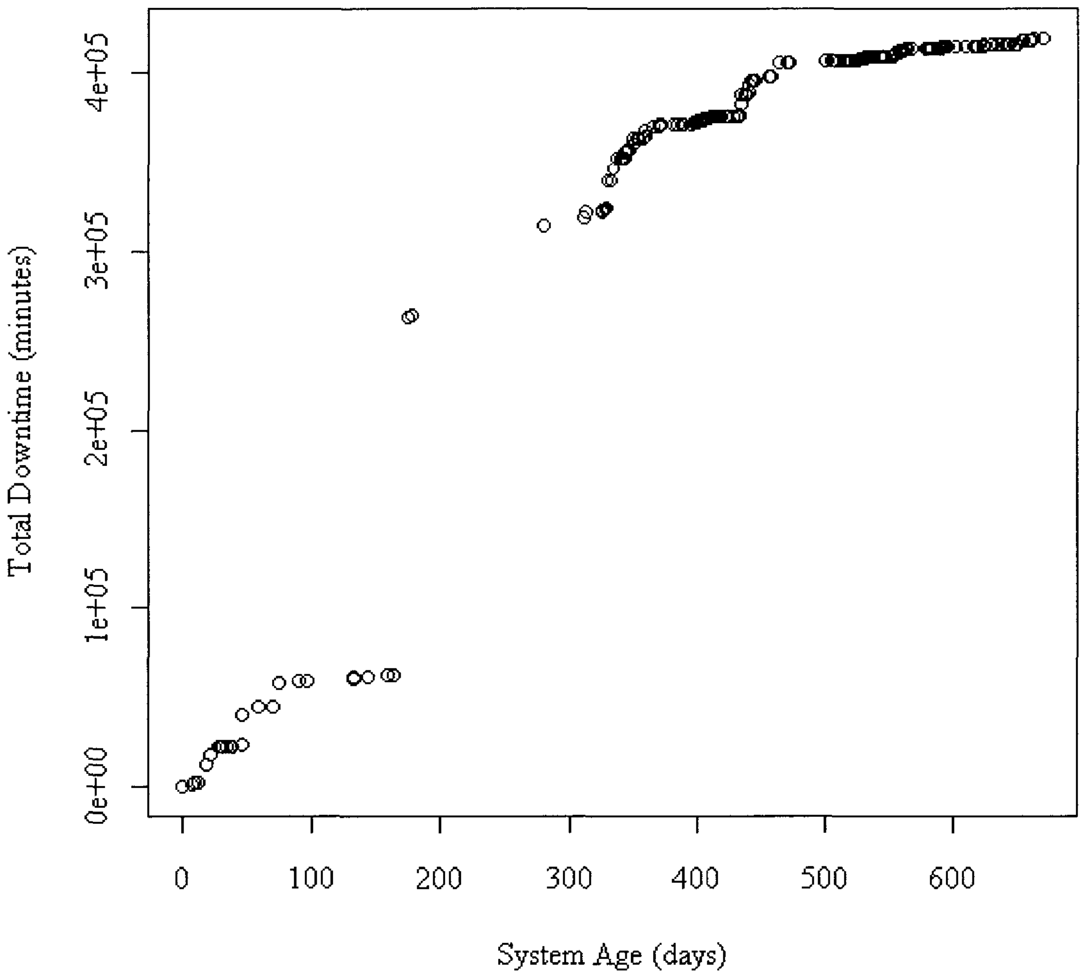


Figure 5.28 Total Downtime vs. System Age, System 11, LANL Data

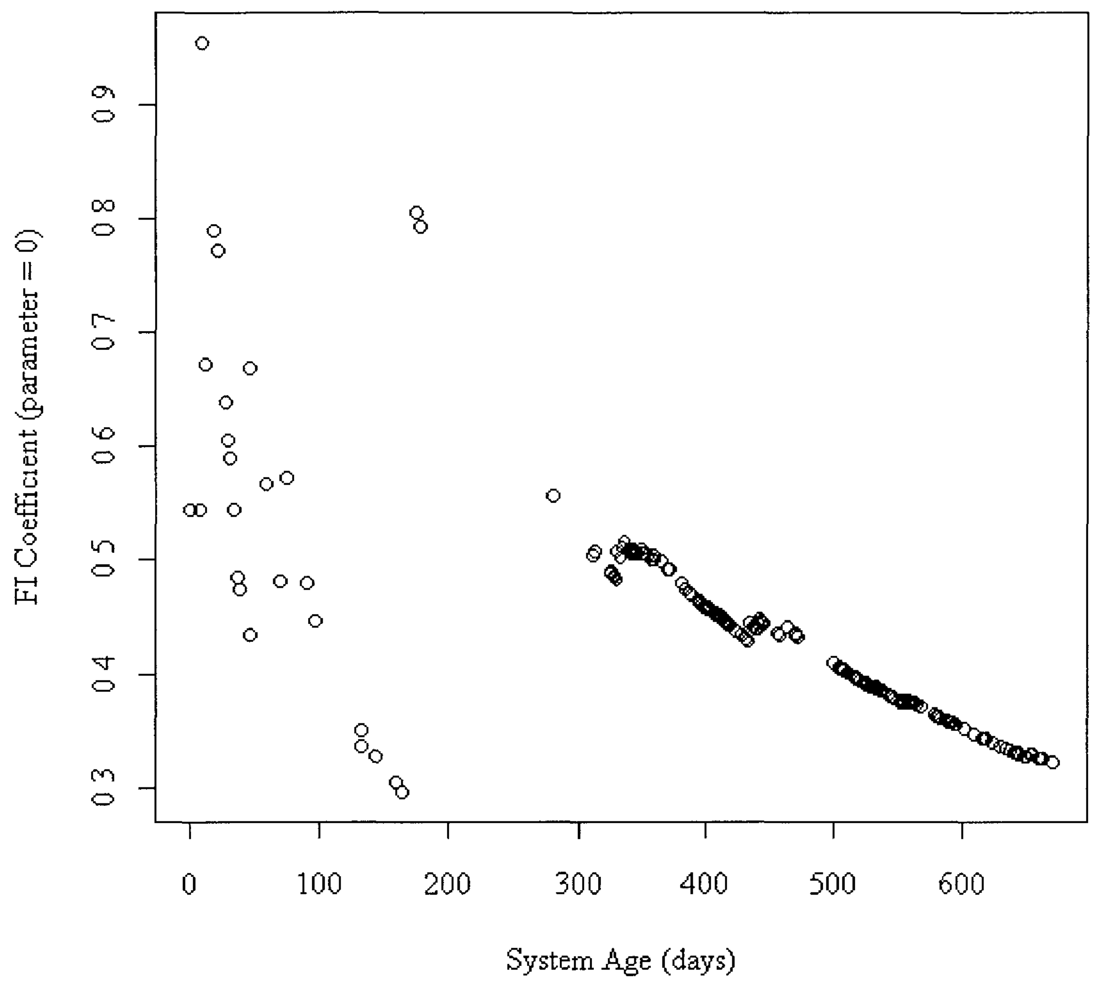


Figure 5.29 FI Coefficient vs. System Age, System 11, LANL Data

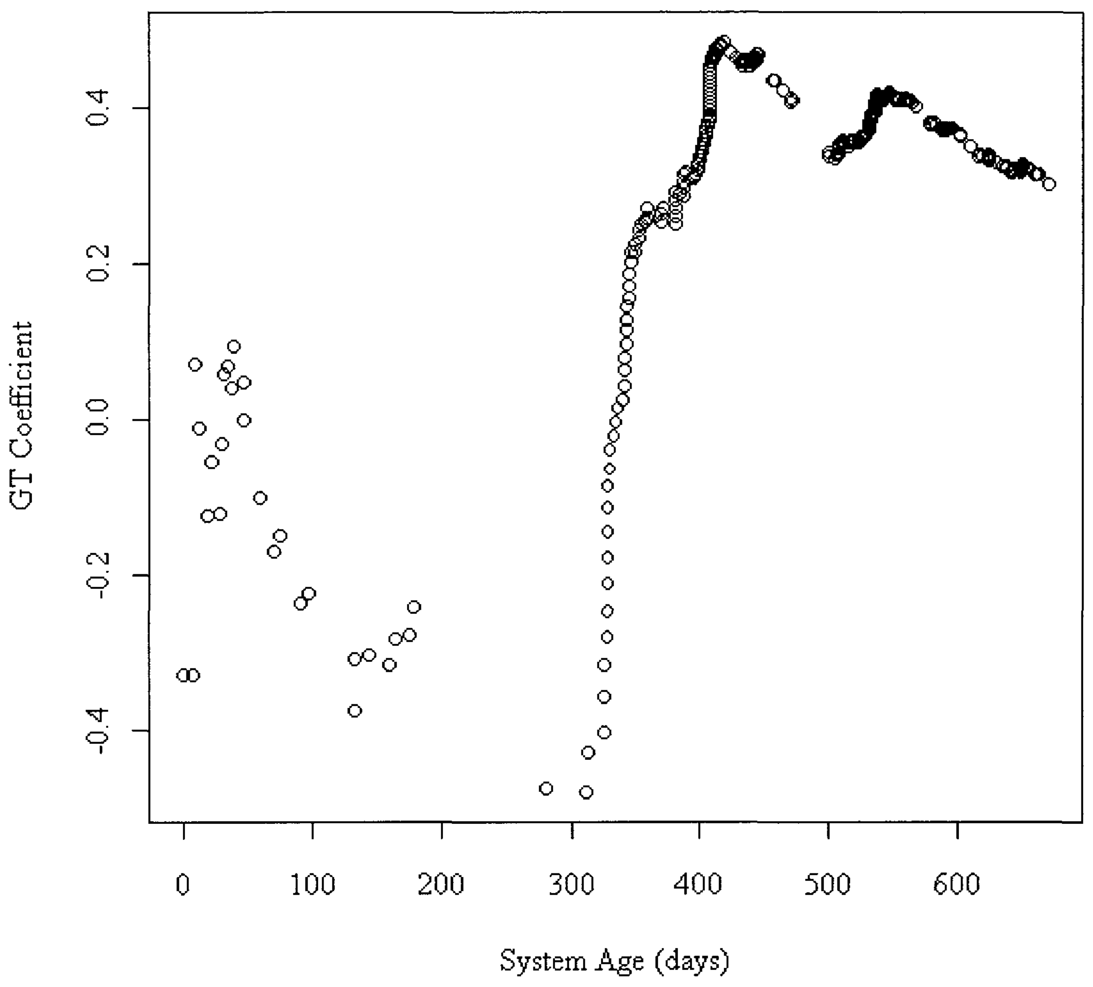


Figure 5.30 ROCOF vs. System Age, System 11, LANL Data

Table 5.11 FI Coefficients and ROCOF, System 12, LANL Data

System	FI ($\varepsilon = 0$)	FI ($\varepsilon = 0.20$)	FI ($\varepsilon = 0.99$)	GT (N(T))
12	0.3989725	0.4317884	0.666791	0.3937295

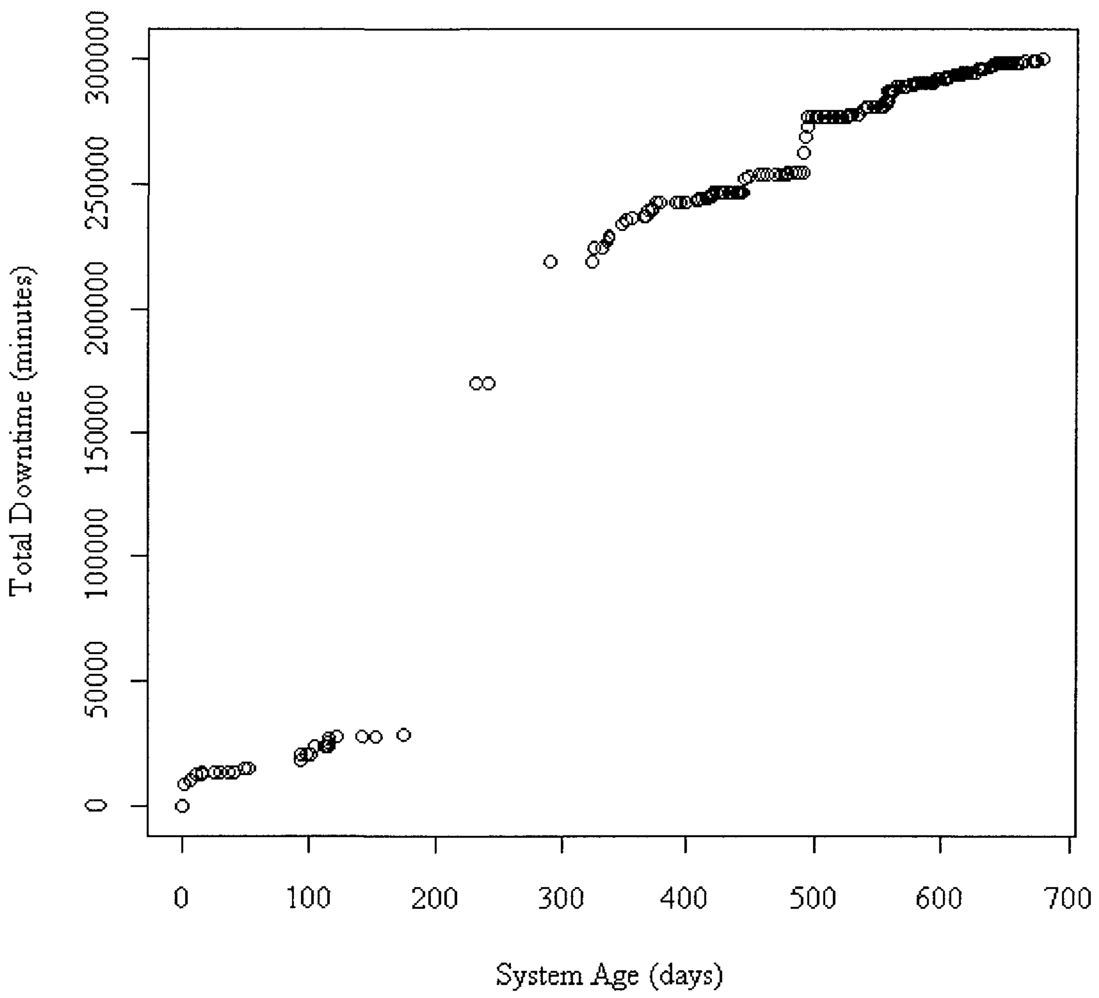


Figure 5.31 Total Downtime vs. System Age, System 12, LANL Data

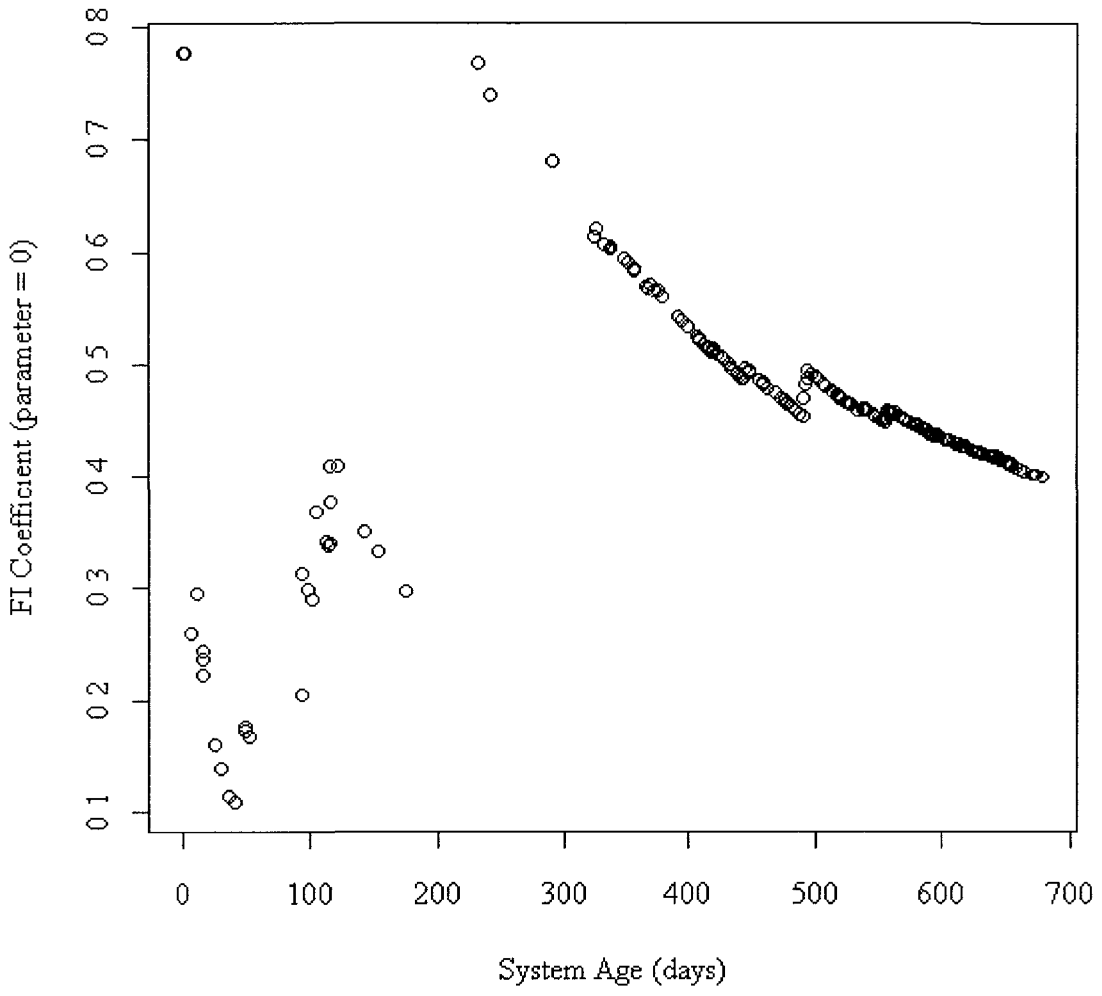


Figure 5.32 FI Coefficient vs. System Age, System 12, LANL Data

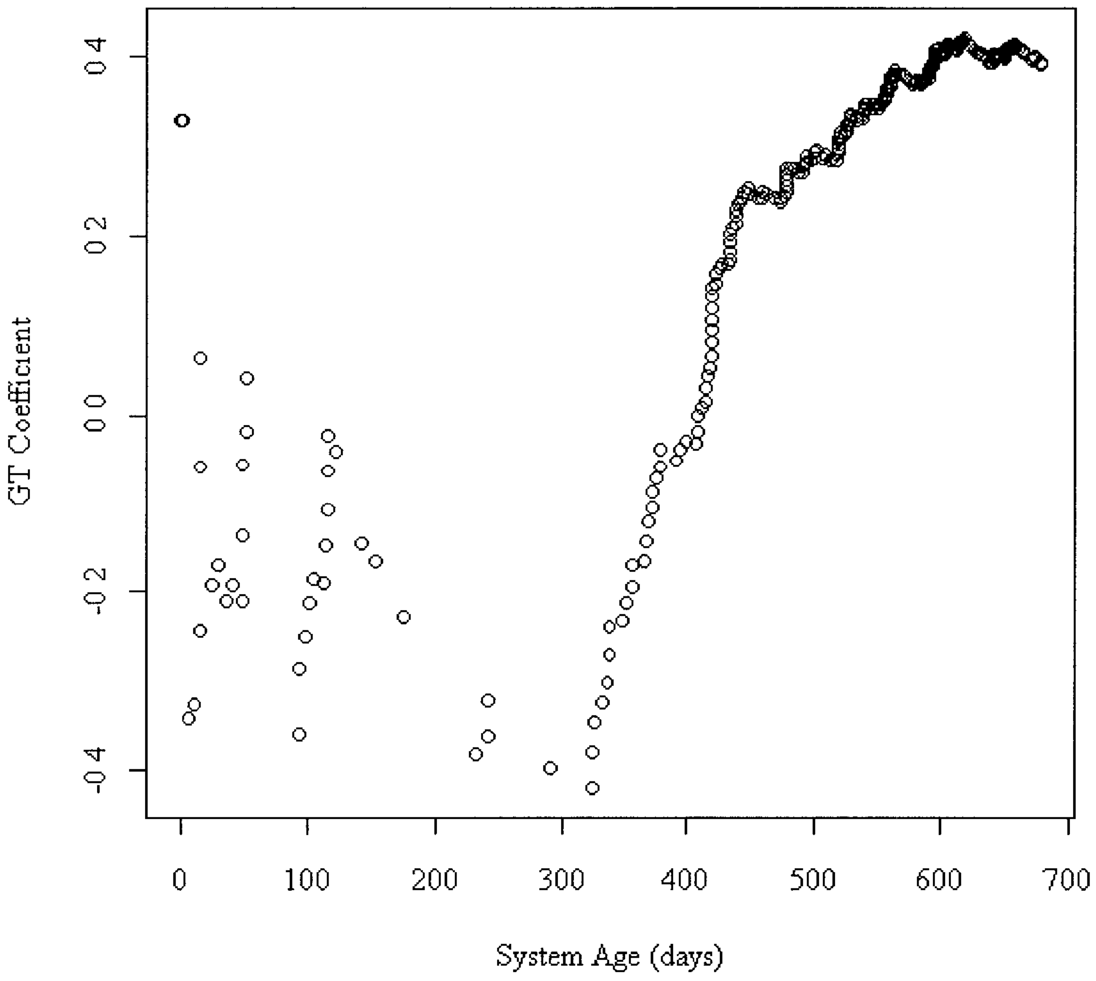


Figure 5.33 ROCOF vs. System Age, System 12, LANL Data

Table 5.12 FI Coefficients and ROCOF, System 13, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
13	0.3546629	0.3805183	0.6209388	0.298536

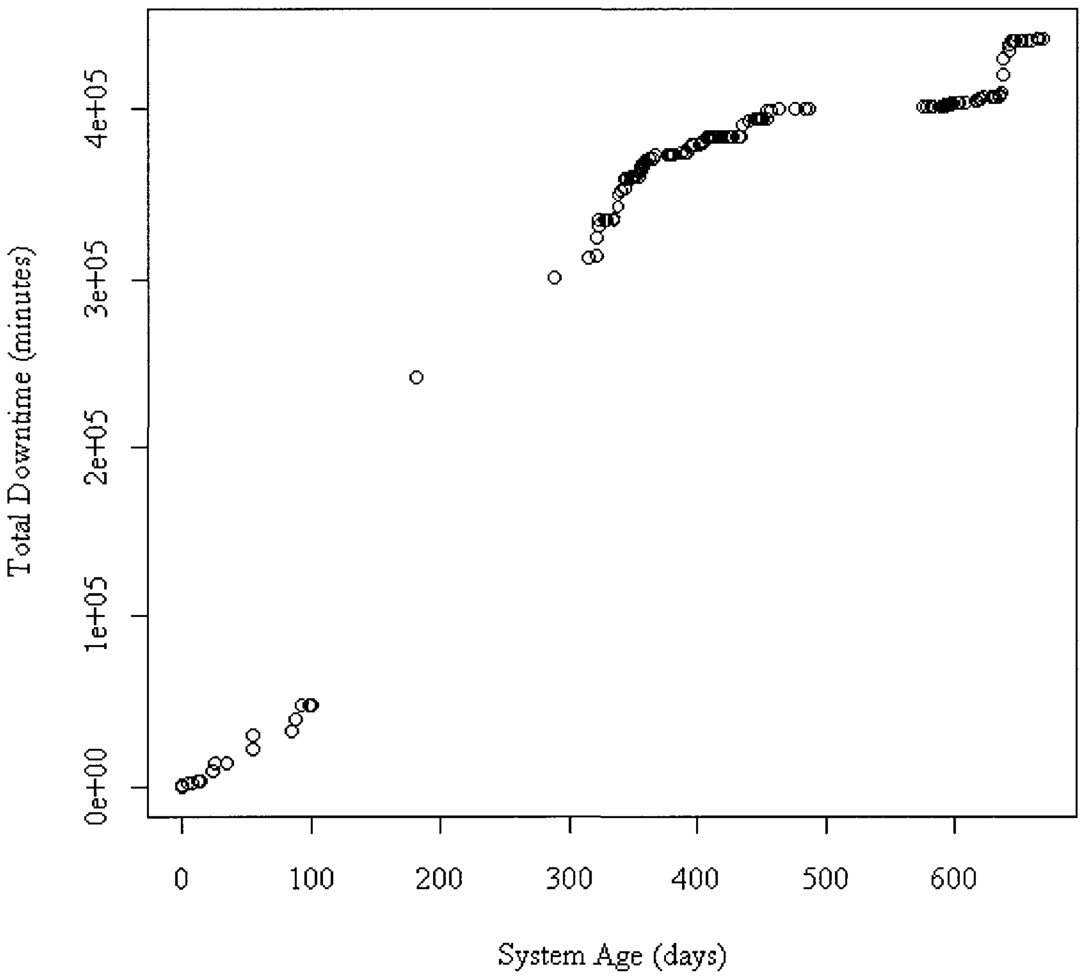


Figure 5.34 Total Downtime vs. System Age, System 13, LANL Data

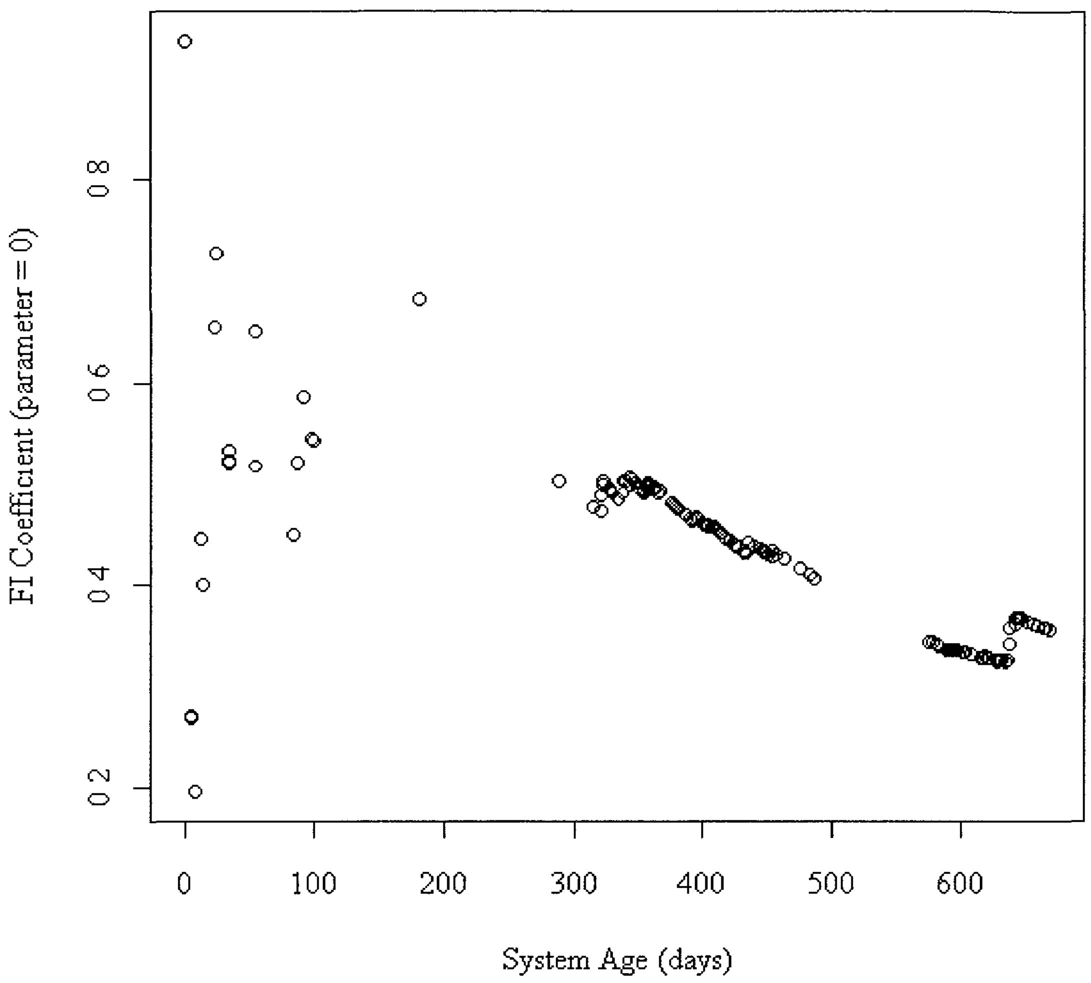


Figure 5.35 FI Coefficient vs. System Age, System 13, LANL Data

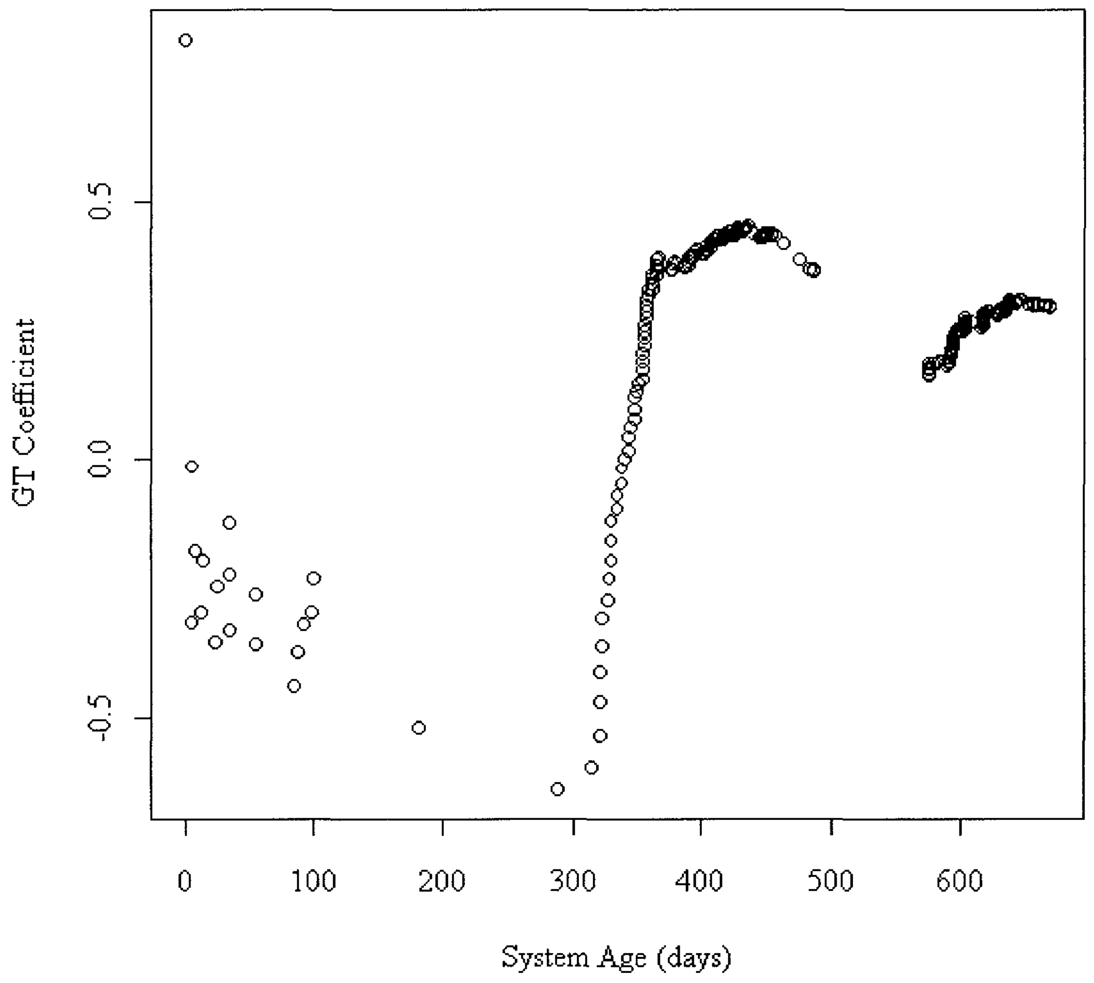


Figure 5.36 ROCOF vs. System Age, System 13, LANL Data

Table 5.13 FI Coefficients and ROCOF, System 14, LANL Data

System	FI ($\varepsilon = 0$)	FI ($\varepsilon = 0.20$)	FI ($\varepsilon = 0.99$)	GT (N(T))
14	0.3113004	0.3501405	0.7133446	0.1522754

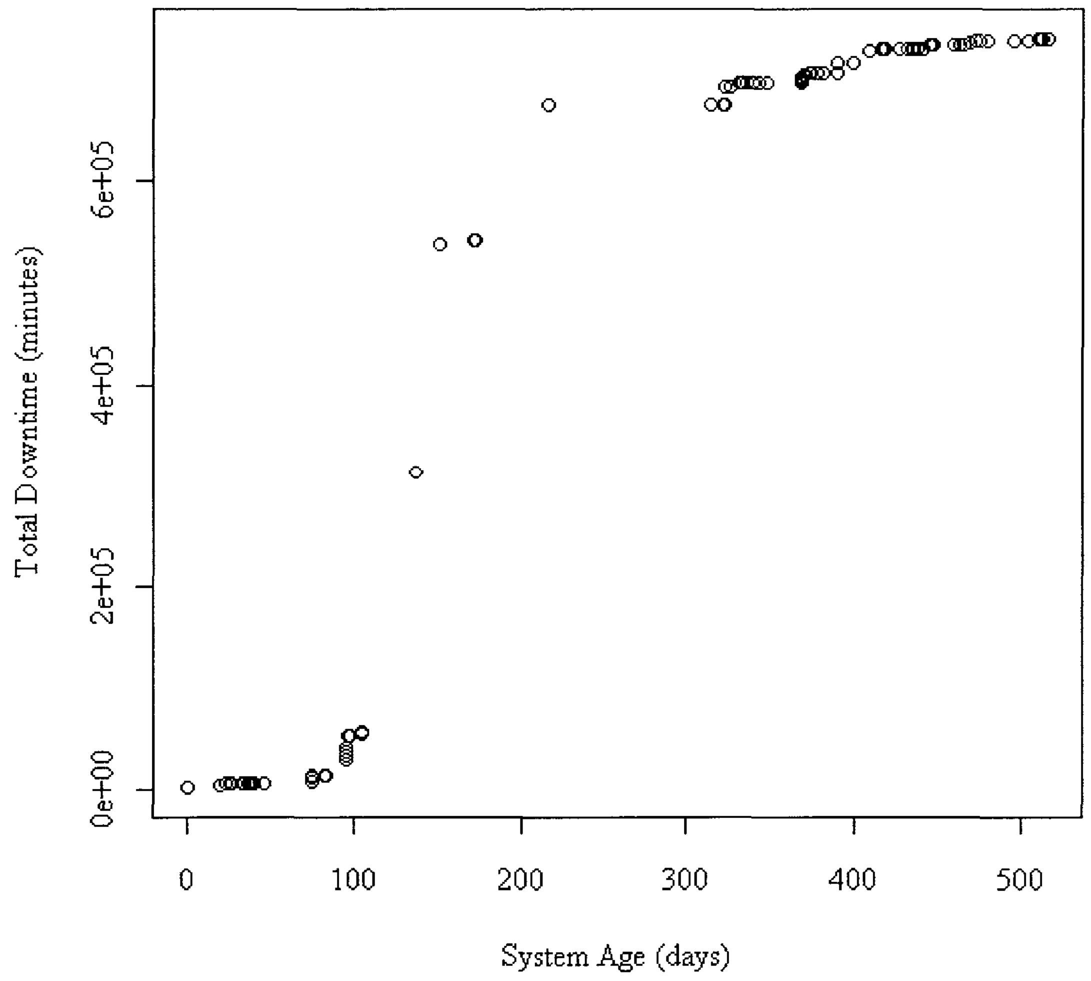


Figure 5.37 Total Downtime vs. System Age, System 14, LANL Data

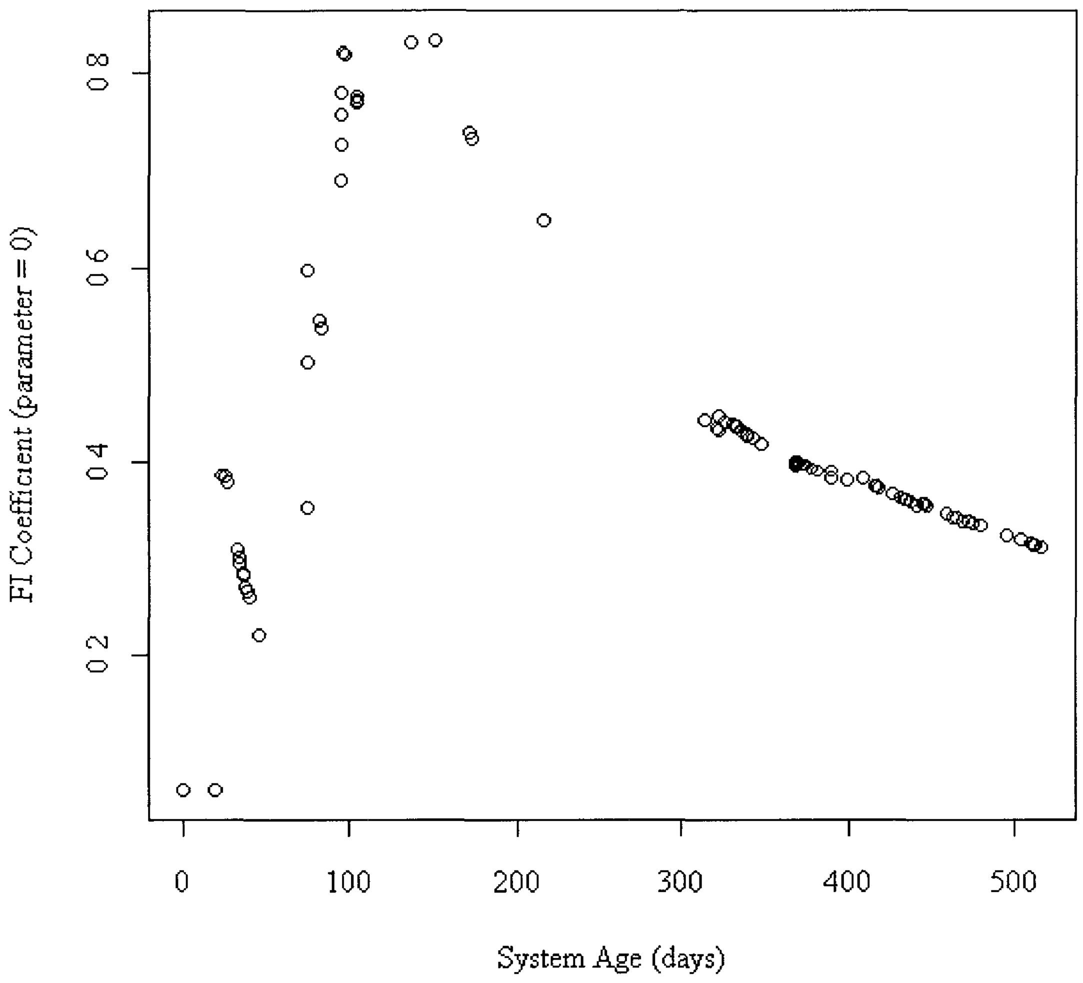


Figure 5.38 FI Coefficient vs. System Age, System 14, LANL Data

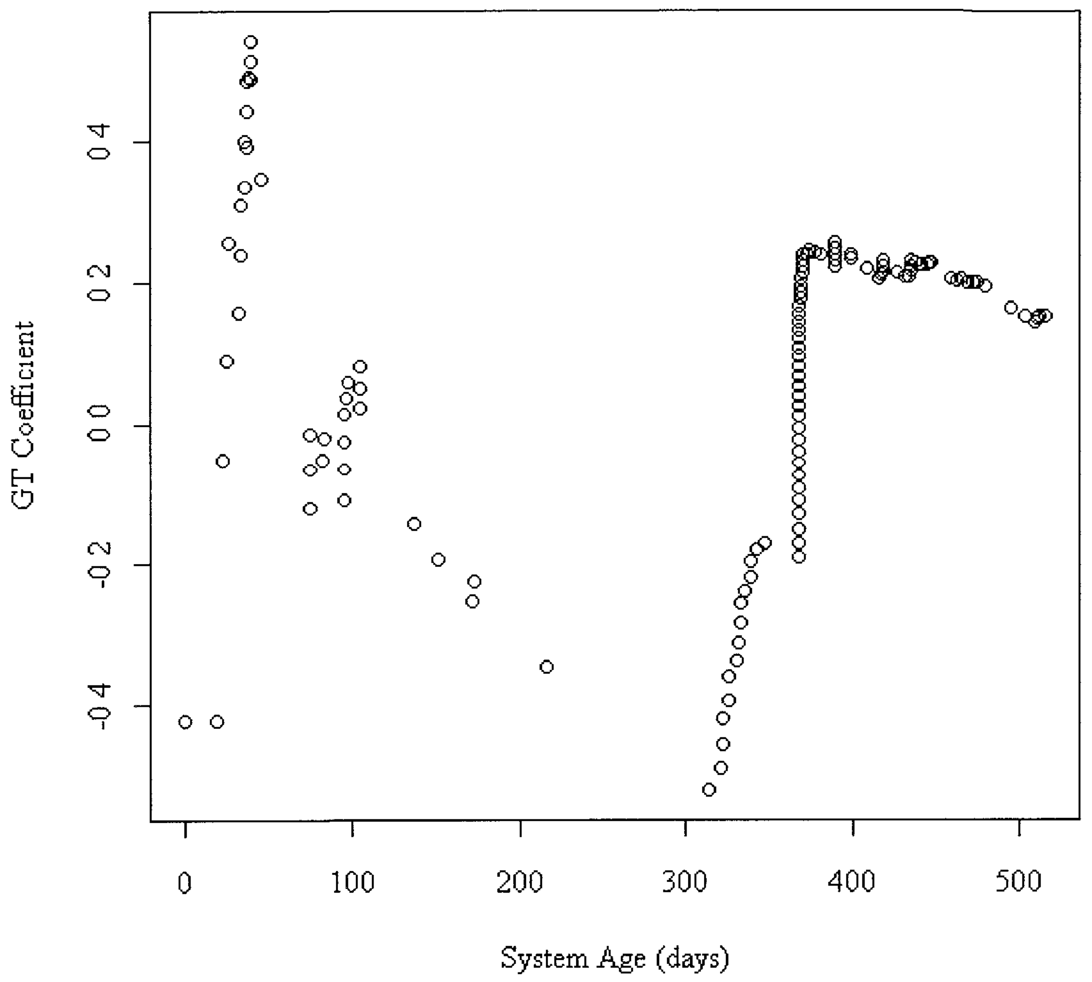


Figure 5.39 ROCOF vs. System Age, System 14, LANL Data

Table 5.14 FI Coefficients and ROCOF, System 15, LANL Data

System	FI ($\varepsilon = 0$)	FI ($\varepsilon = 0.20$)	FI ($\varepsilon = 0.99$)	GT (N(T))
15	0.3203539	0.3445422	0.6678887	-0.3574969

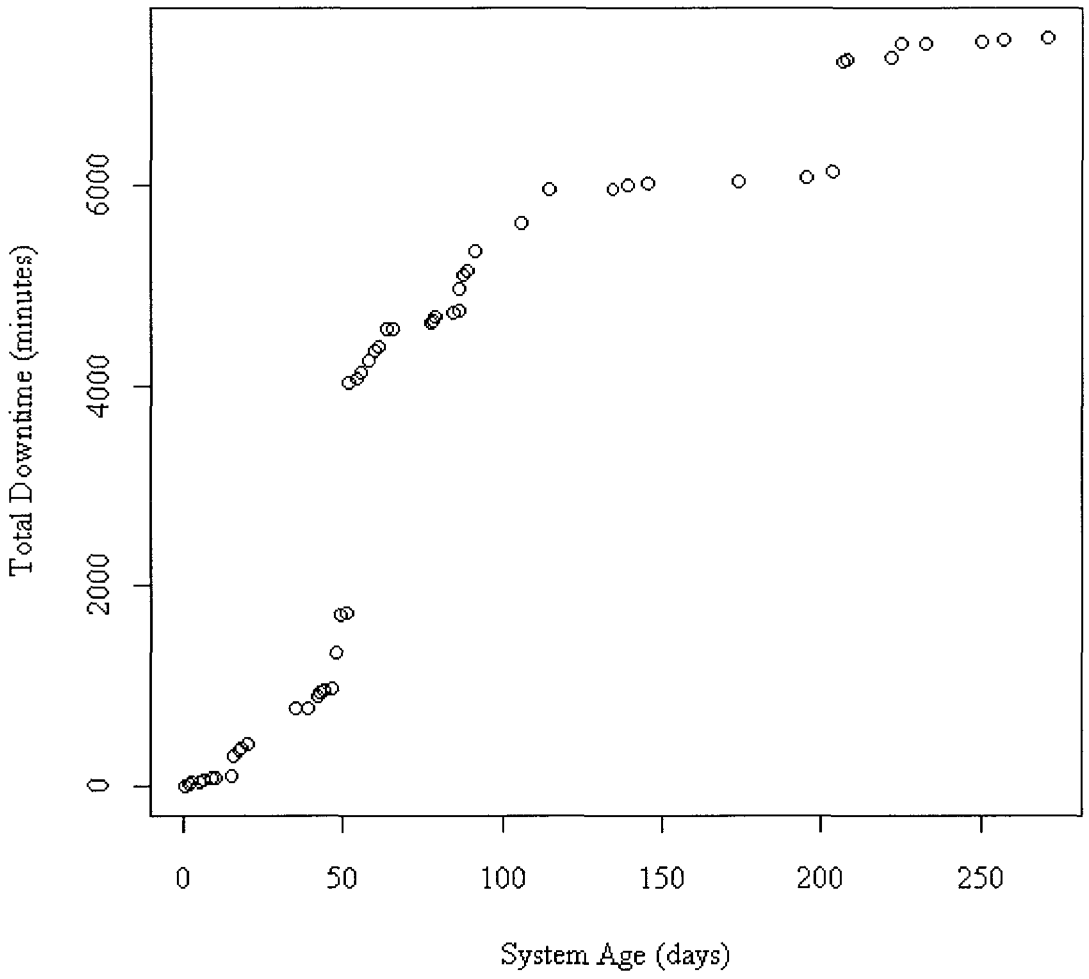


Figure 5.40 Total Downtime vs. System Age, System 15, LANL Data

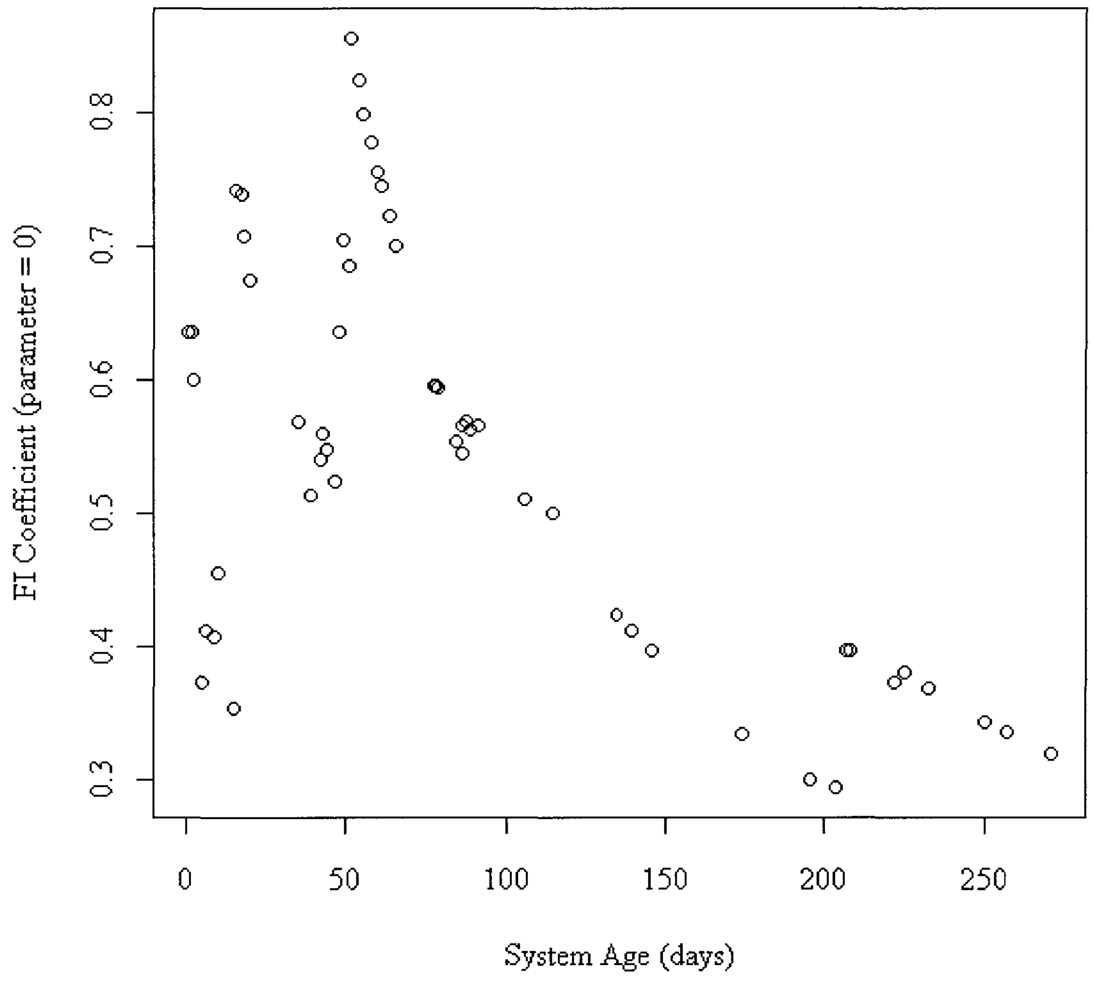


Figure 5.41 FI Coefficient vs. System Age, System 15, LANL Data

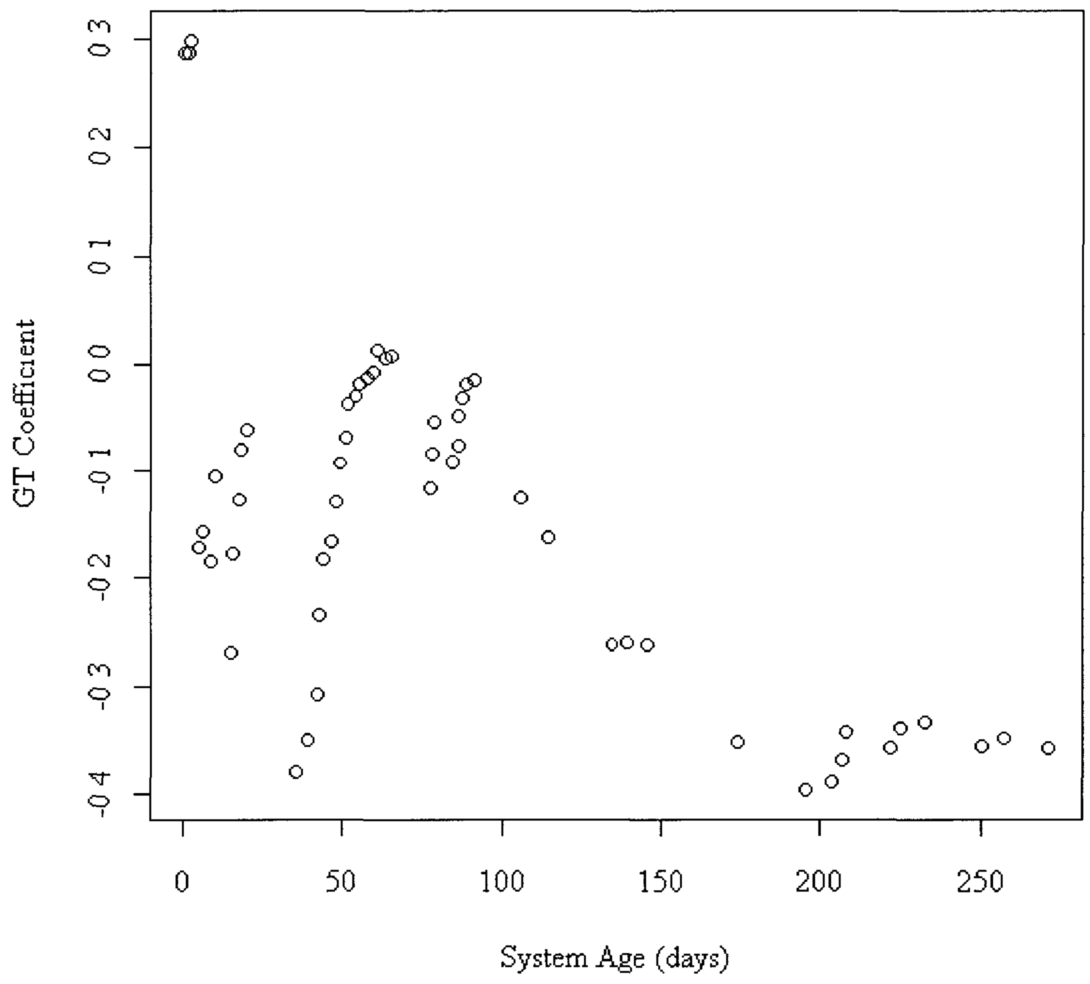


Figure 5.42 ROCOF vs. System Age, System 15, LANL Data

Table 5.15 FI Coefficients and ROCOF, System 16, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
16	0.506235	0.5470675	0.7952932	-0.1423502

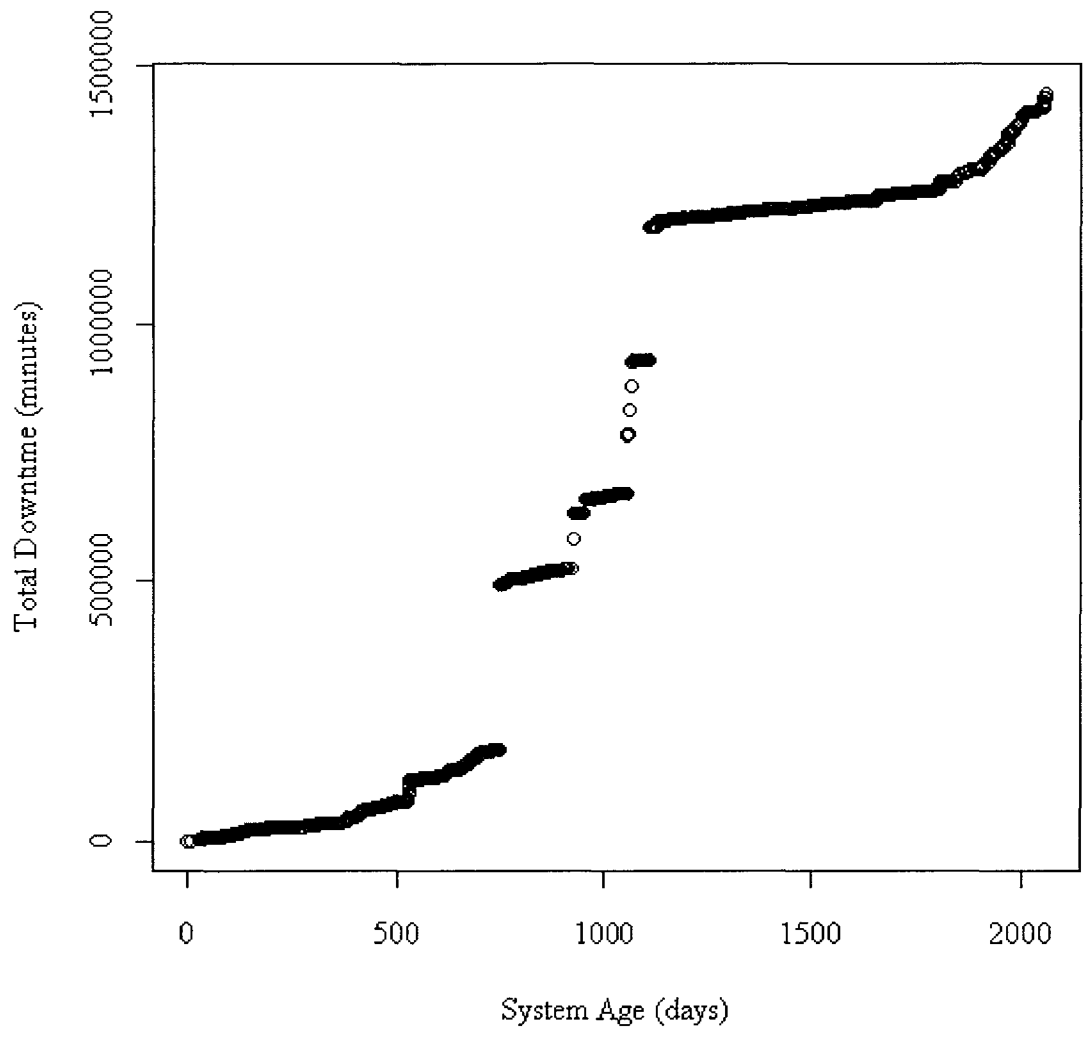


Figure 5.43 Total Downtime vs. System Age, System 16, LANL Data

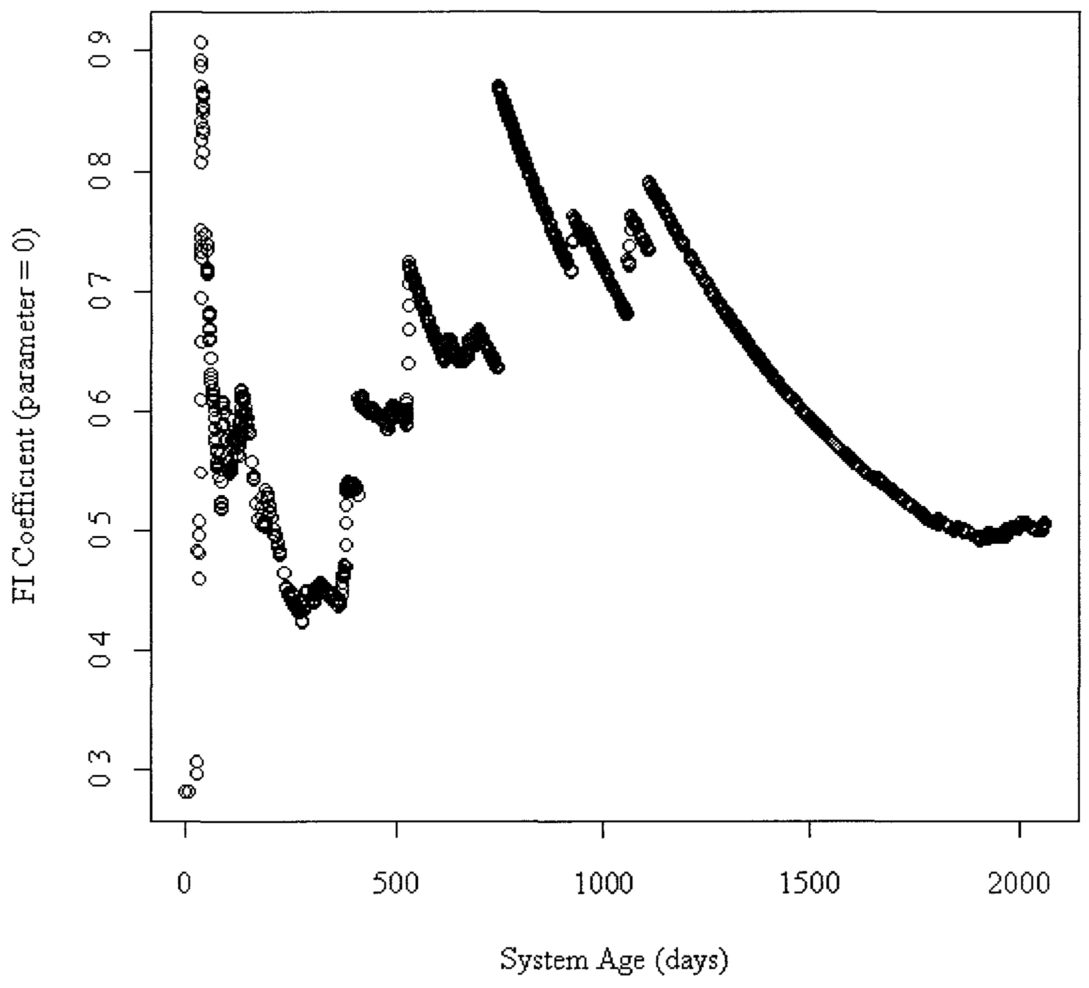


Figure 5.44 FI Coefficient vs. System Age, System 16, LANL Data

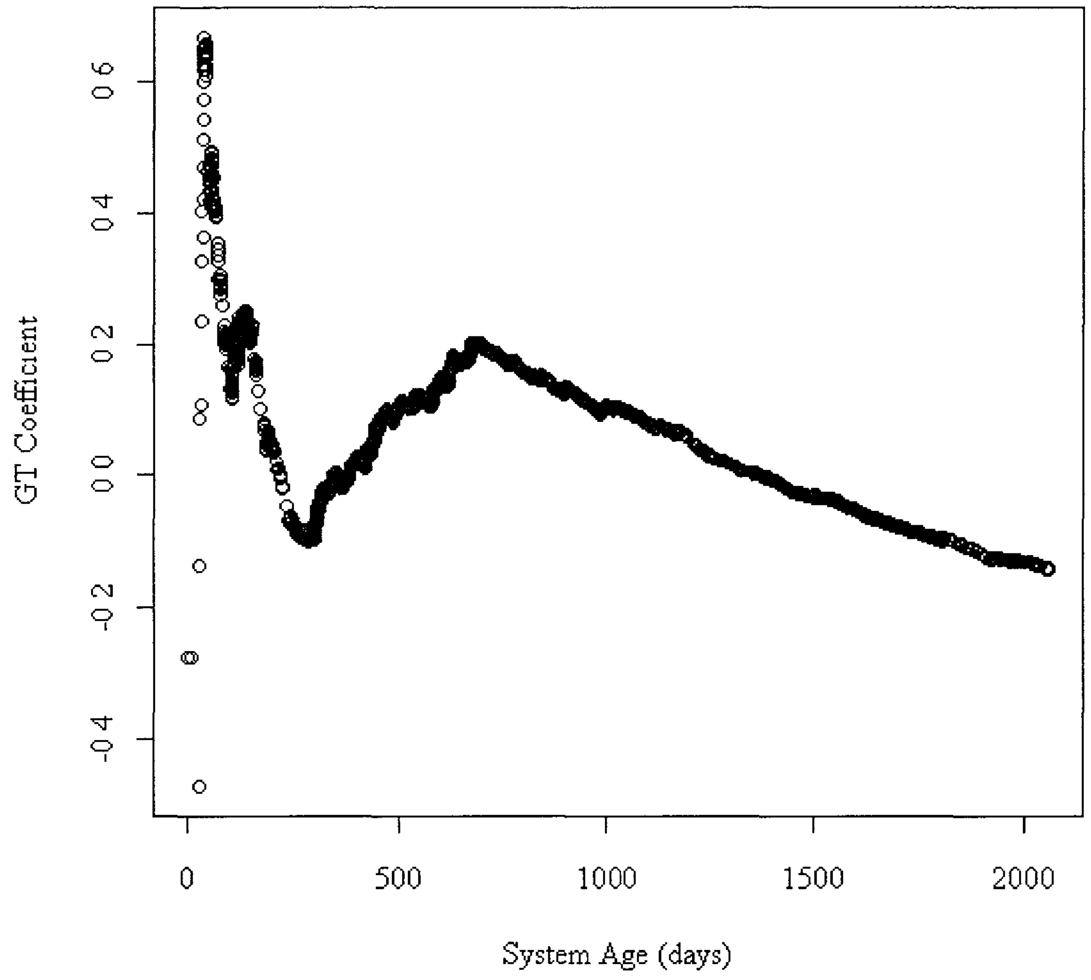


Figure 5.45 ROCOF vs. System Age, System 16, LANL Data

Table 5.16 FI Coefficients and ROCOF, System 17, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
17	0.4431006	0.4693957	0.6992799	-0.1168528

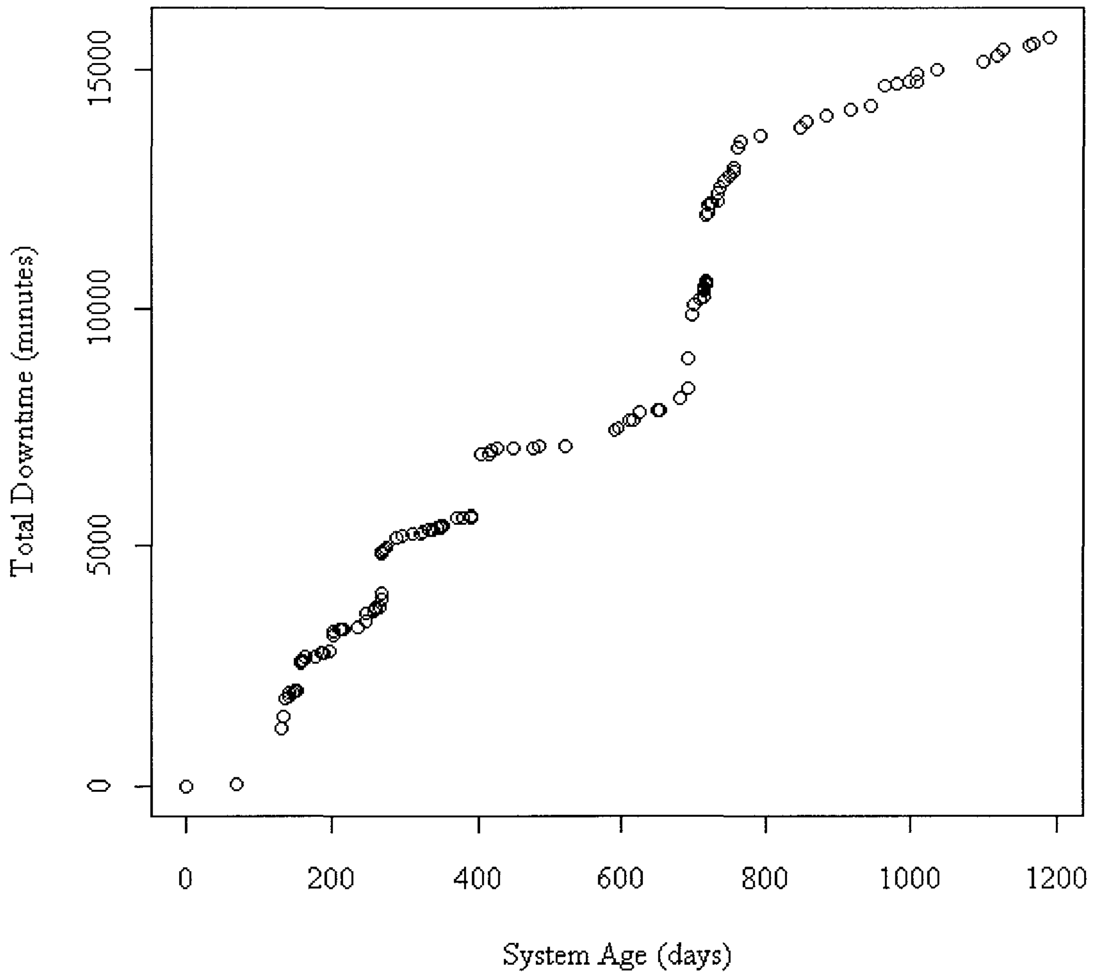


Figure 5.46 Total Downtime vs. System Age, System 17, LANL Data

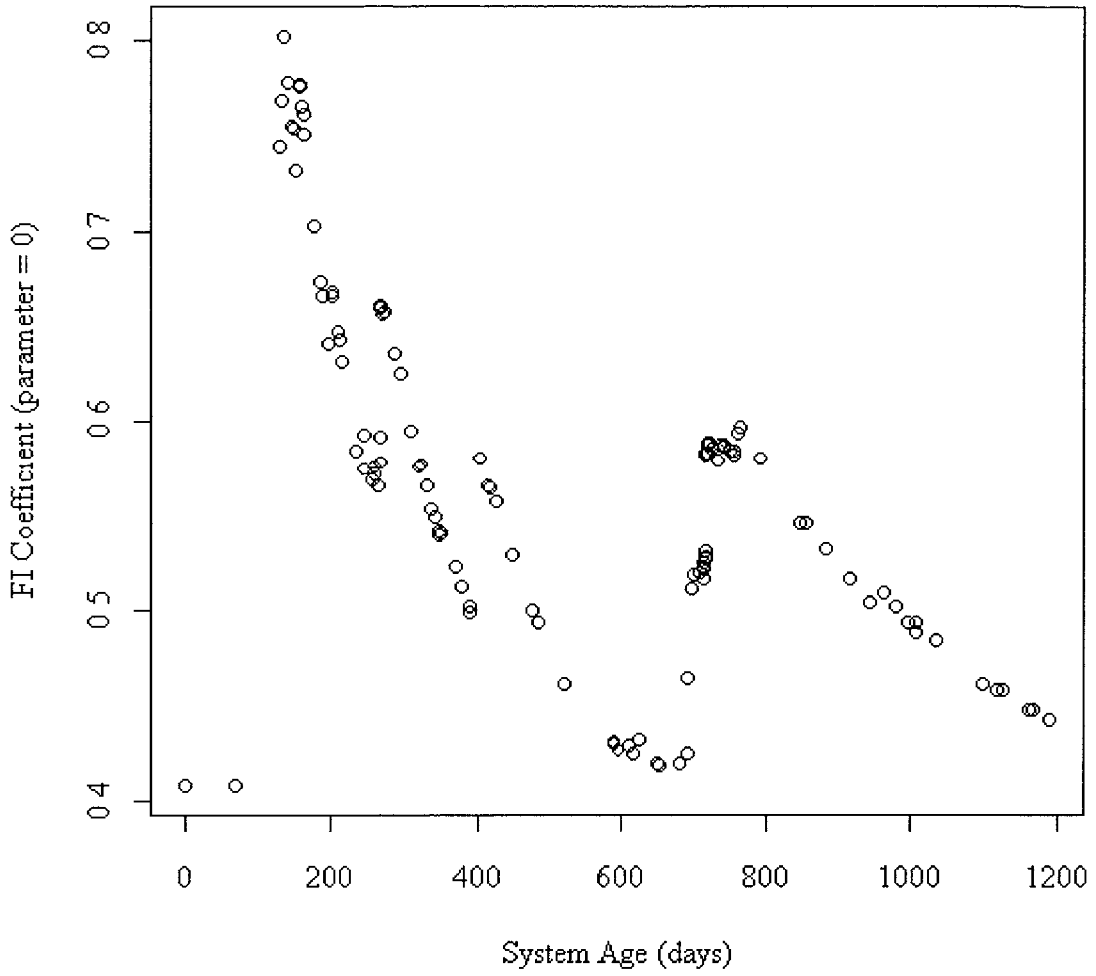


Figure 5.47 FI Coefficient vs. System Age, System 17, LANL Data

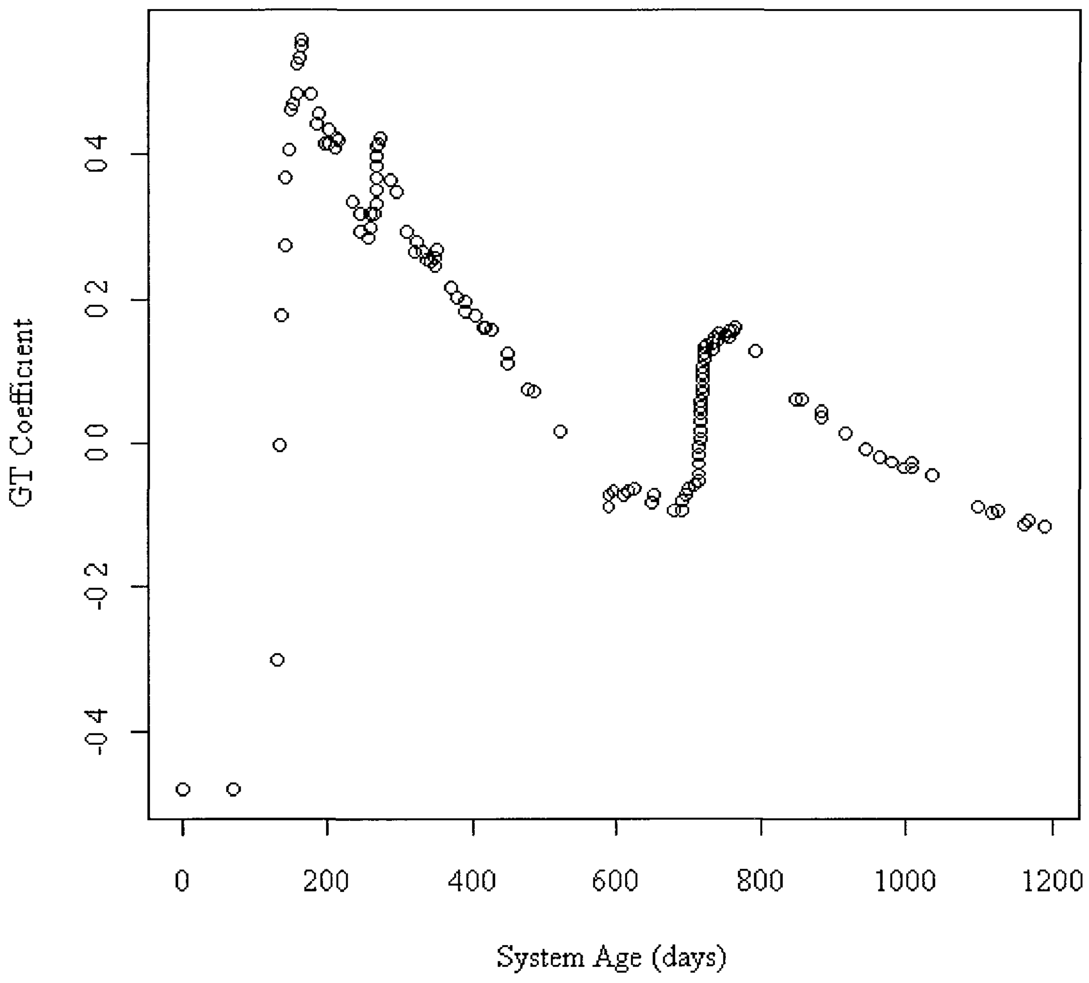


Figure 5.48 ROCOF vs. System Age, System 17, LANL Data

Table 5.17 FI Coefficients and ROCOF, System 18, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
18	0.4424562	0.4581639	0.5760466	-0.01011574

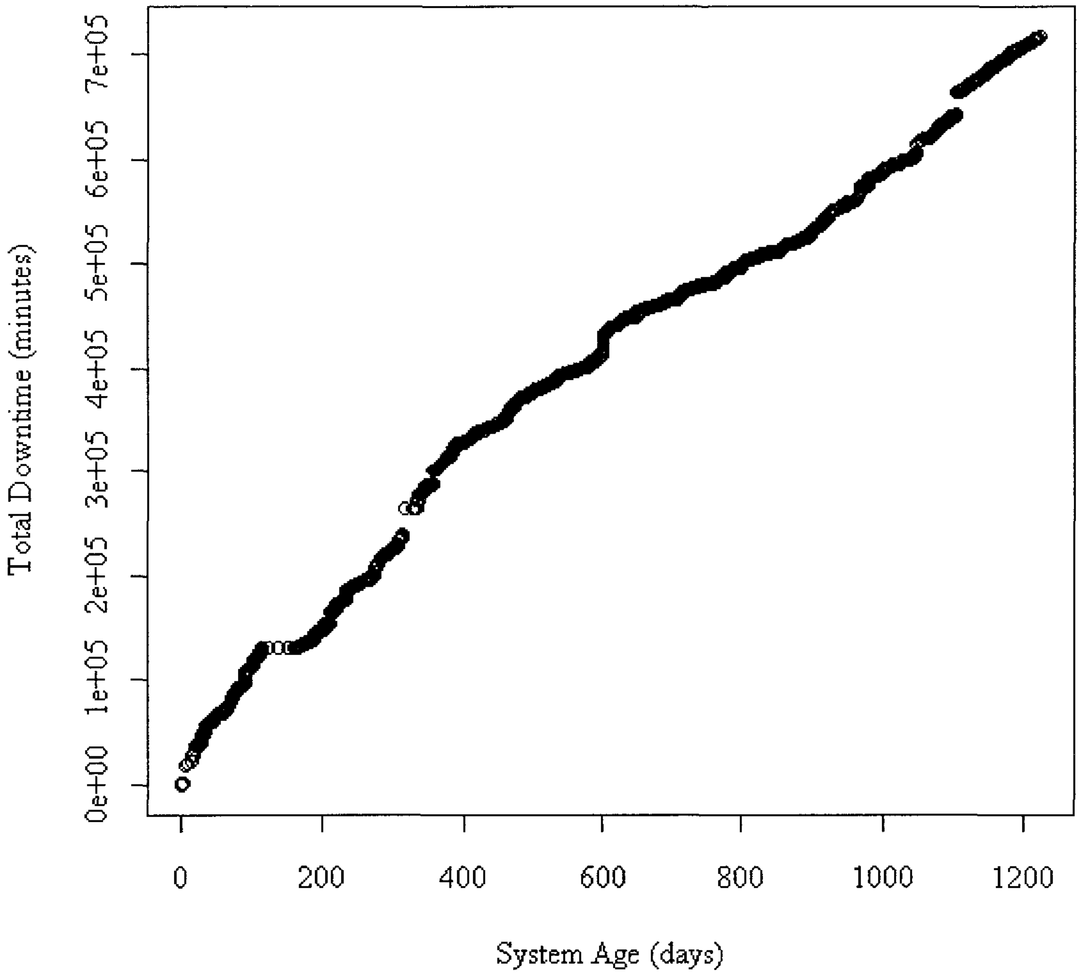


Figure 5.49 Total Downtime vs. System Age, System 18, LANL Data

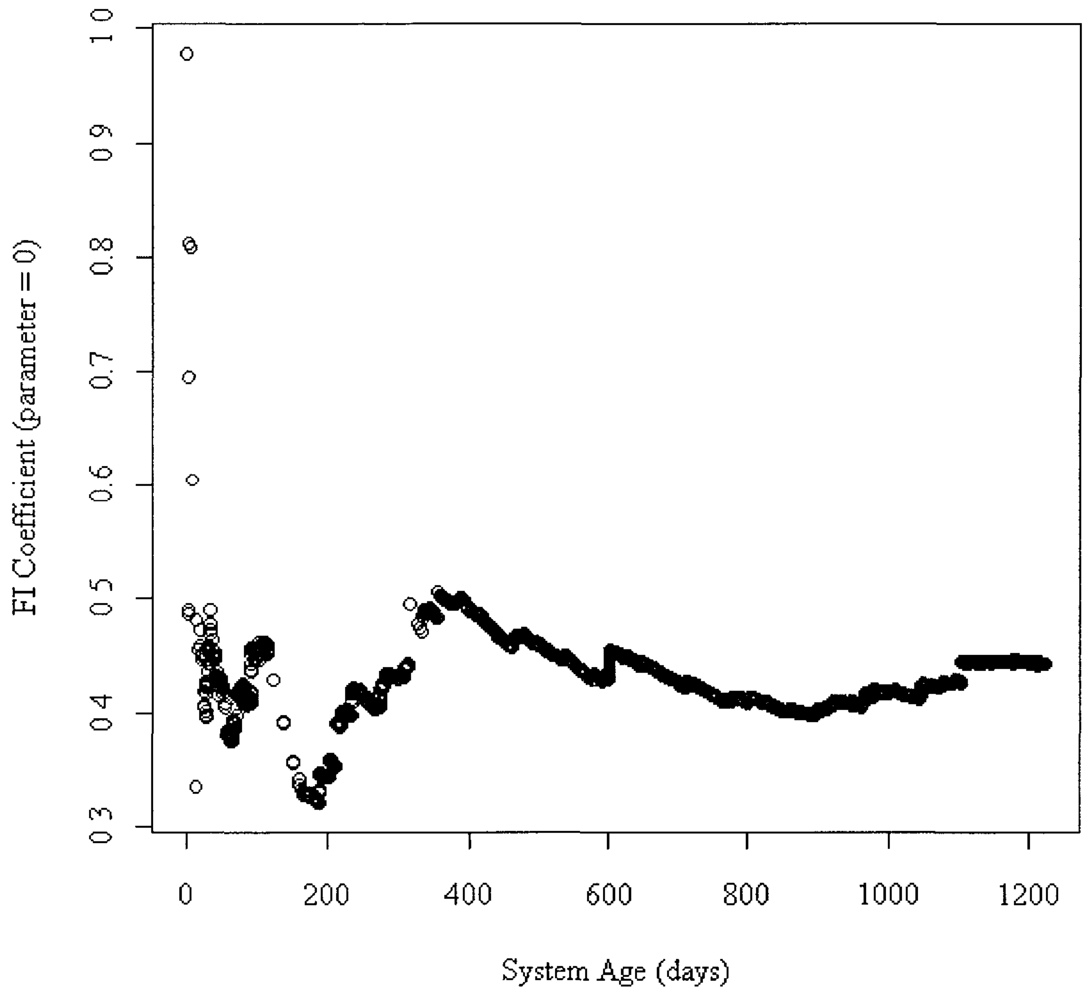


Figure 5.50 FI Coefficient vs. System Age, System 18, LANL Data

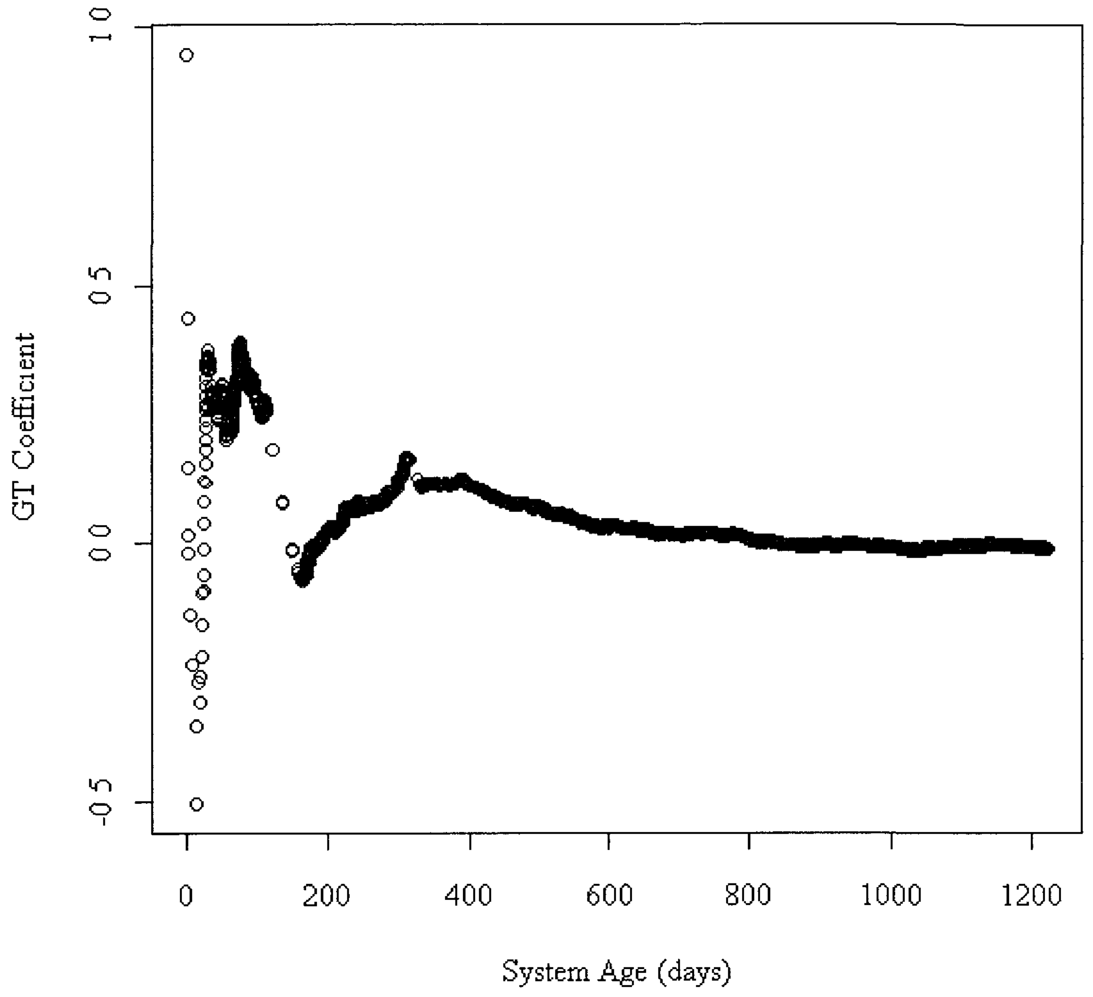


Figure 5.51 ROCOF vs. System Age, System 18, LANL Data

Table 5.18 FI Coefficients and ROCOF, System 19, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
19	0.4978905	0.5130915	0.6412173	-0.02191605

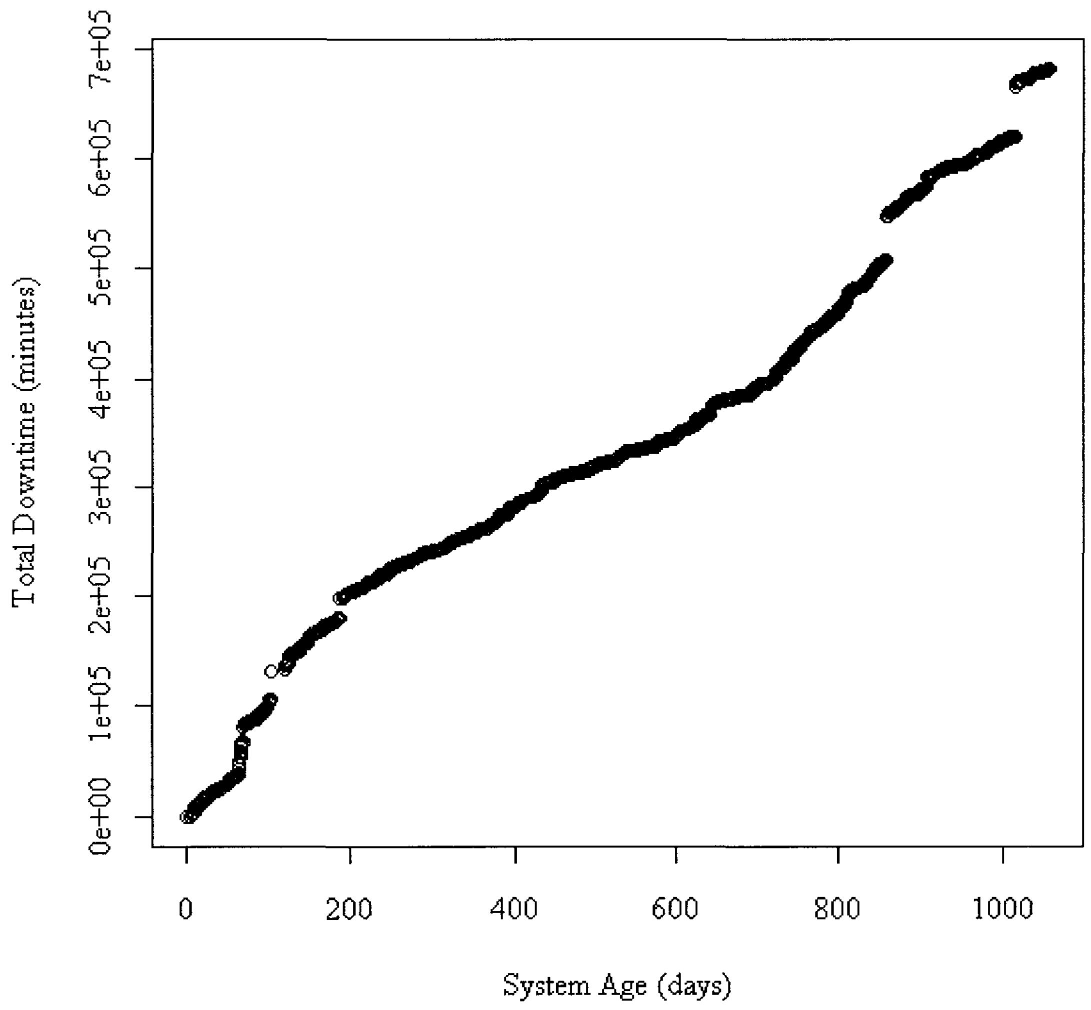


Figure 5.52 Total Downtime vs. System Age, System 19, LANL Data

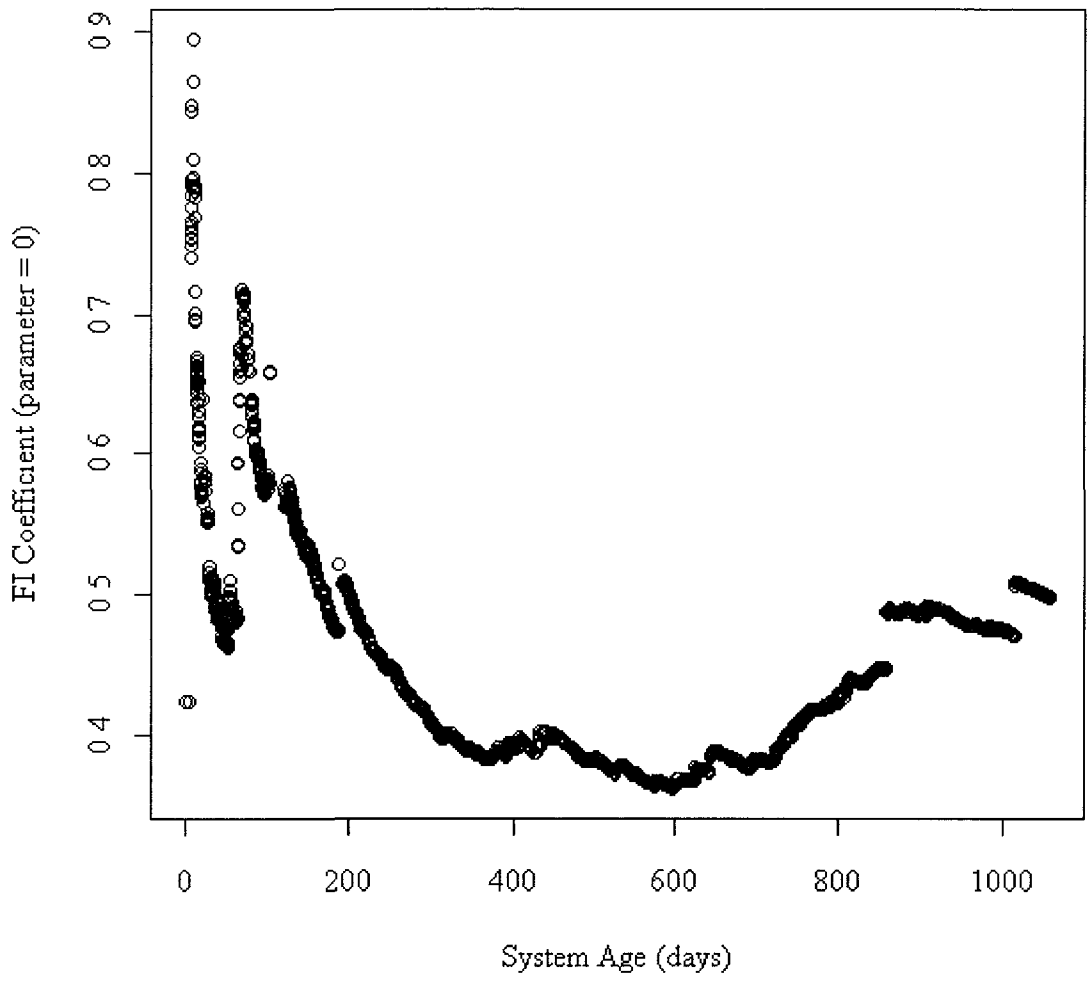


Figure 5.53 FI Coefficient vs. System Age, System 19, LANL Data

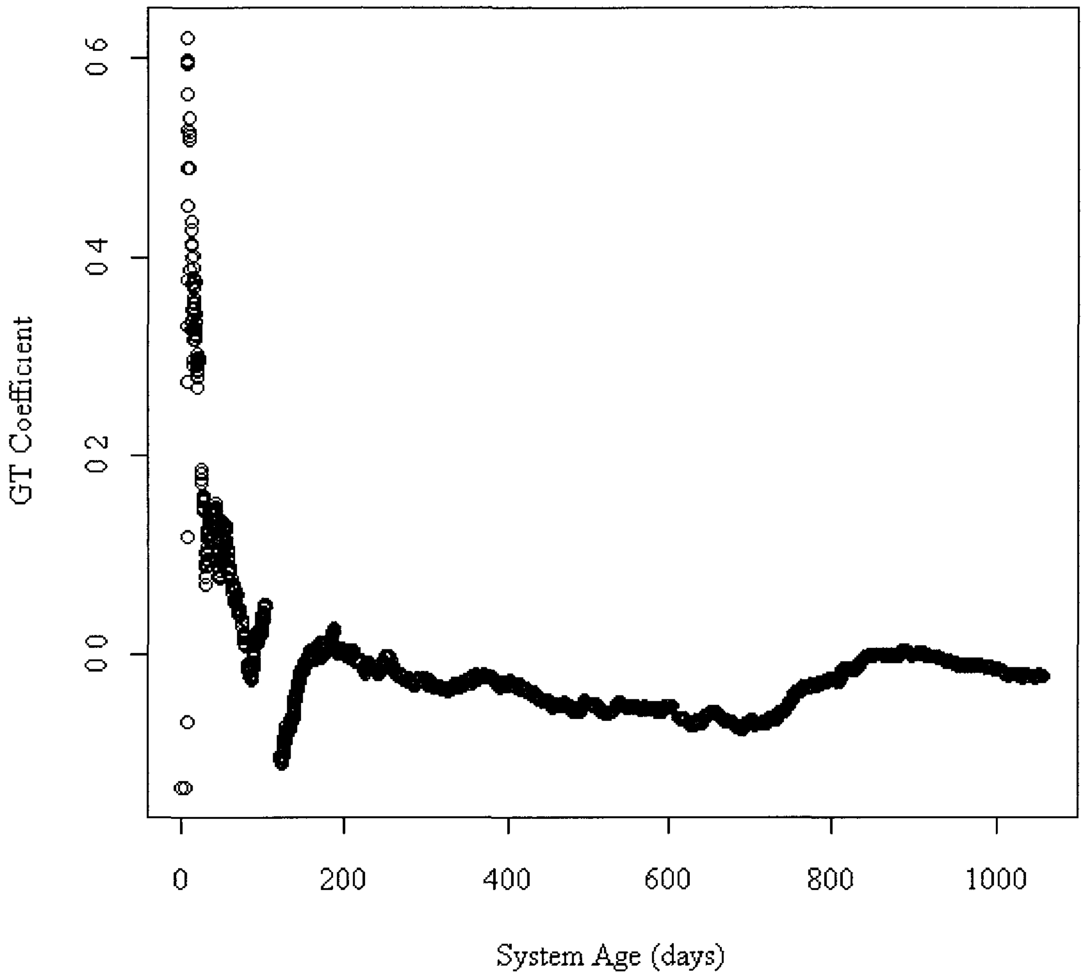


Figure 5.54 ROCOF vs. System Age, System 19, LANL Data

Table 5.19 FI Coefficients and ROCOF, System 20, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
20	0.2358755	0.2441450	0.3392873	-0.3663878

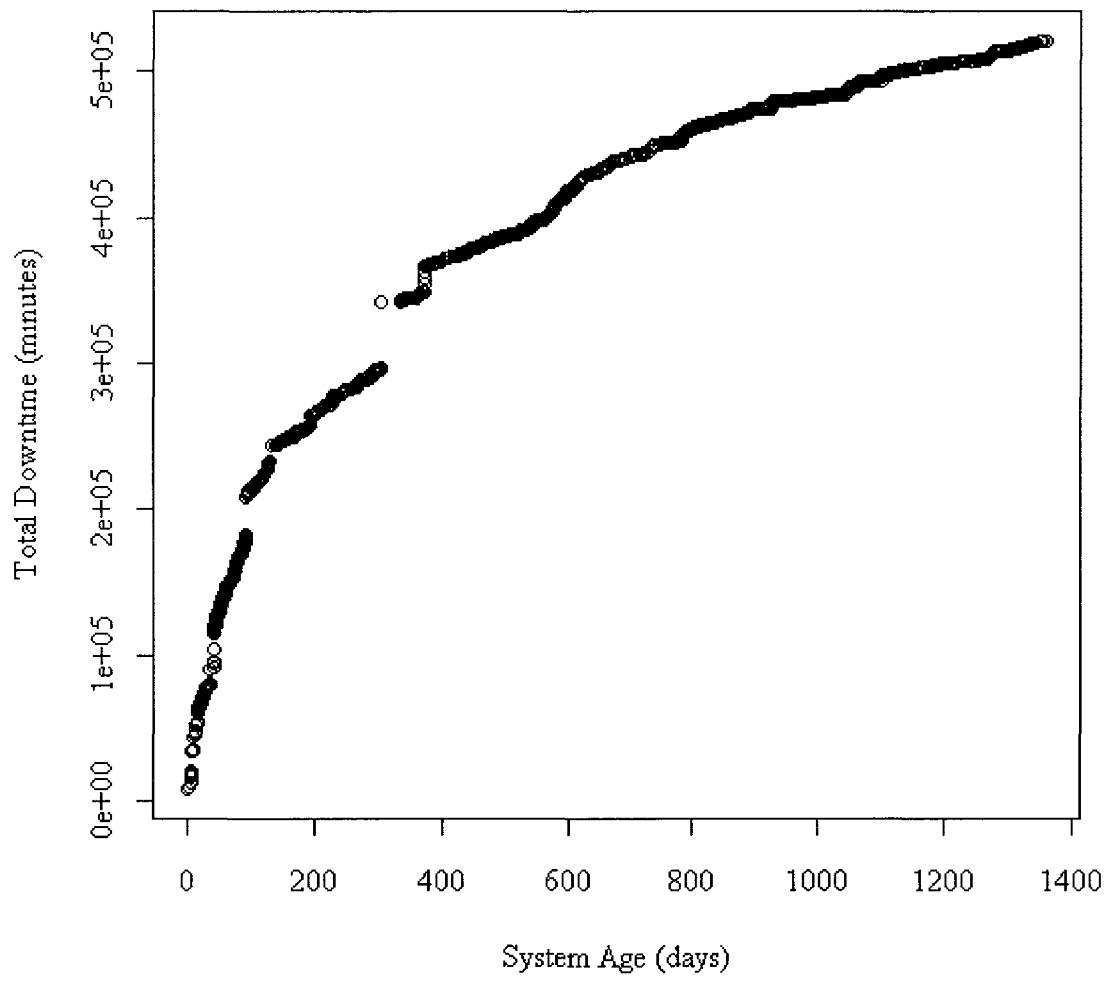


Figure 5.55 Total Downtime vs. System Age, System 20, LANL Data

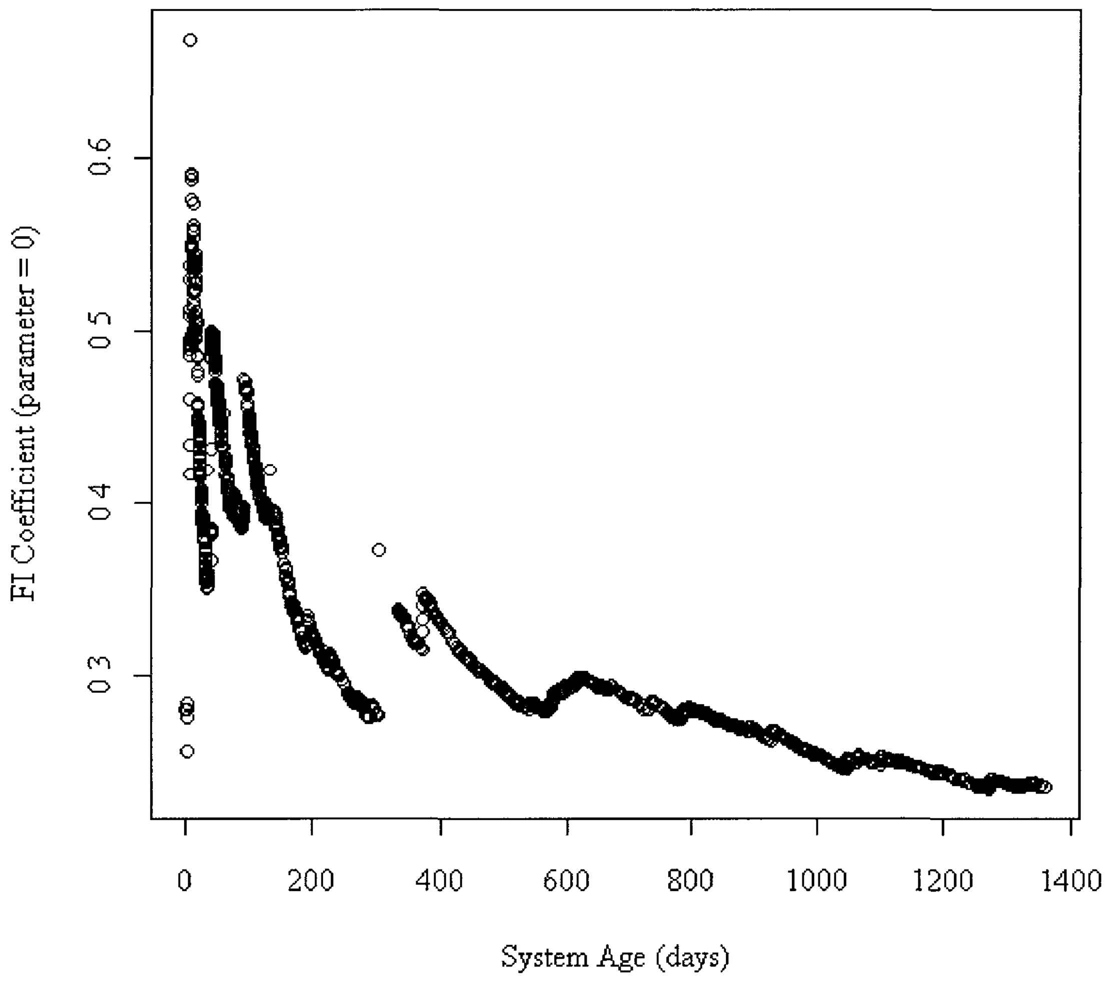


Figure 5.56 FI Coefficient vs. System Age, System 20, LANL Data

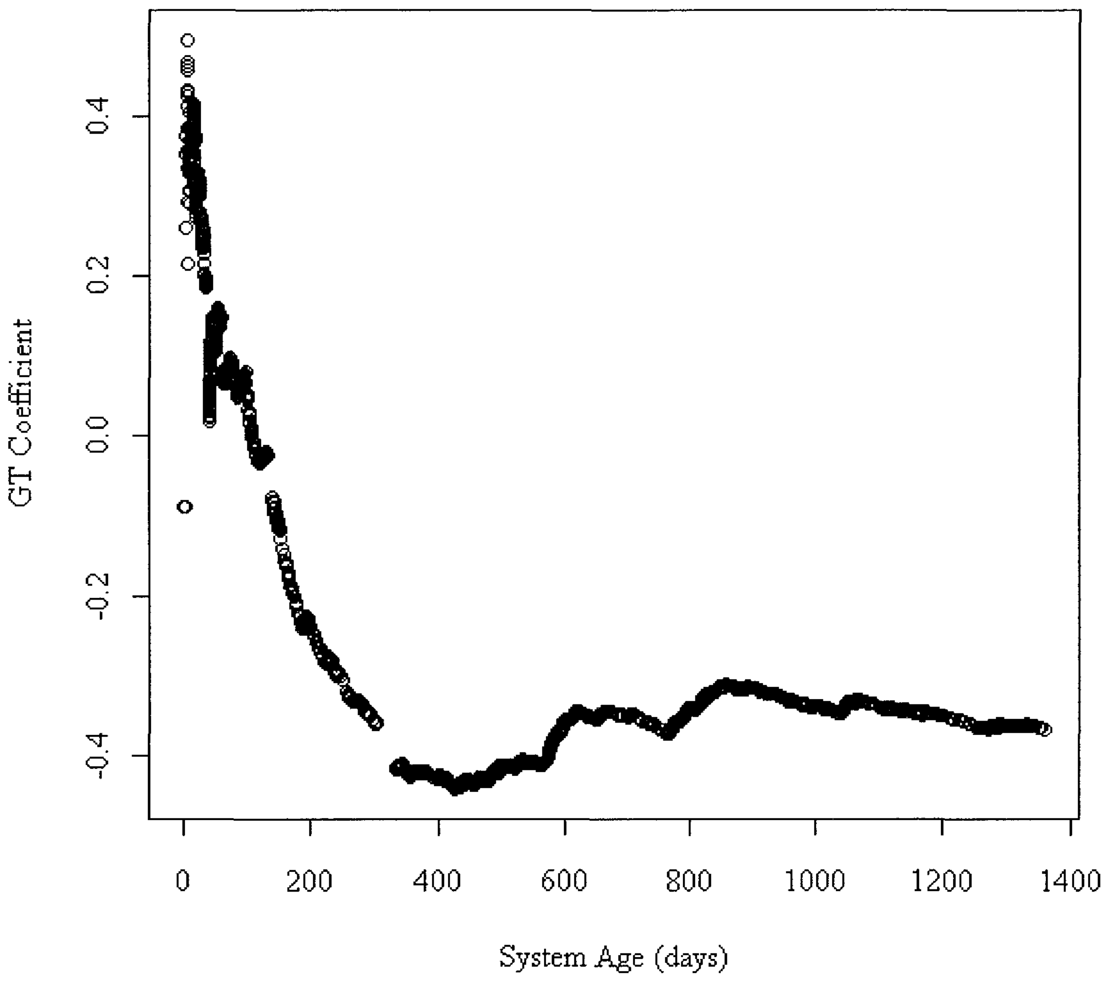


Figure 5.57 ROCOF vs. System Age, System 20, LANL Data

Table 5.20 FI Coefficients and ROCOF, System 21, LANL Data

System	FI ($\varepsilon = 0$)	FI ($\varepsilon = 0.20$)	FI ($\varepsilon = 0.99$)	GT (N(T))
21	0.3521442	0.3752301	0.821141	-0.2562533

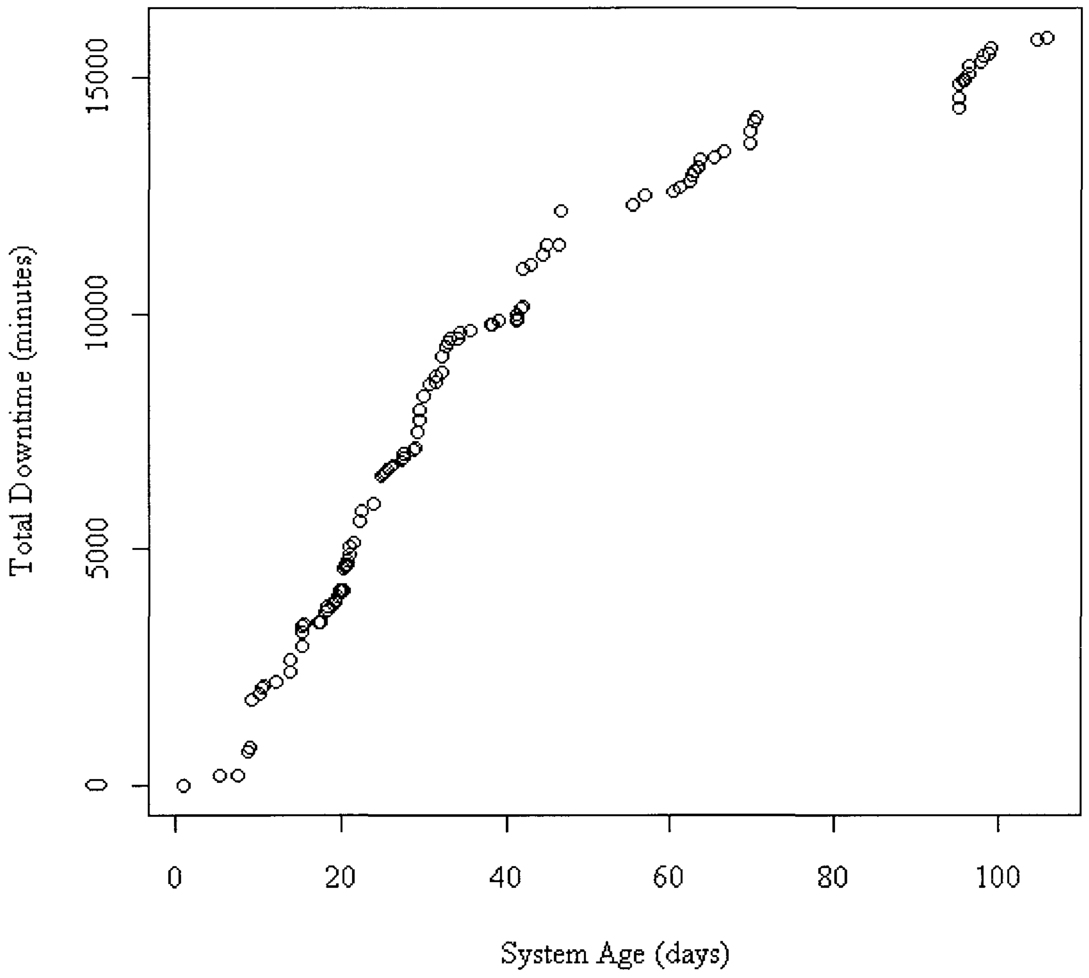


Figure 5.58 Total Downtime vs. System Age, System 21, LANL Data

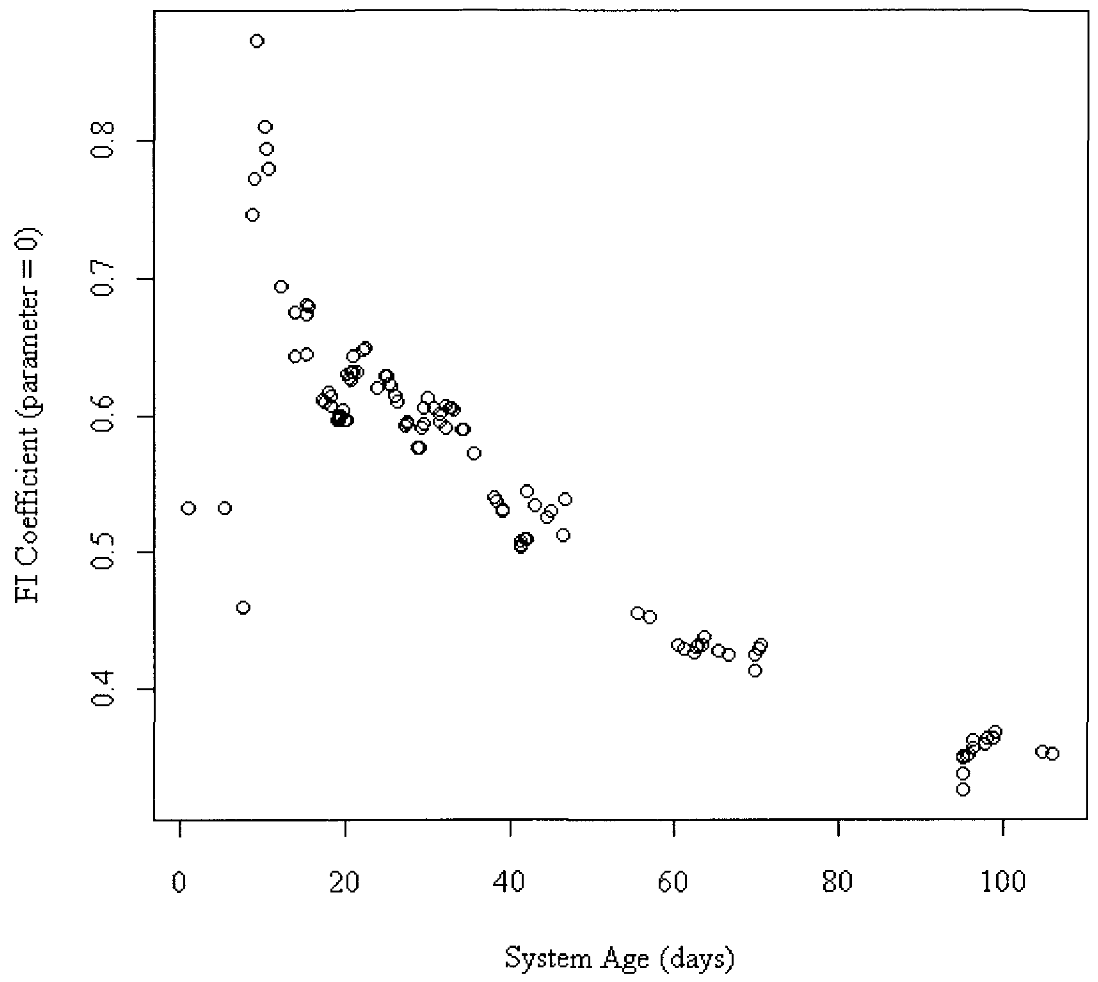


Figure 5.59 FI Coefficient vs. System Age, System 21, LANL Data

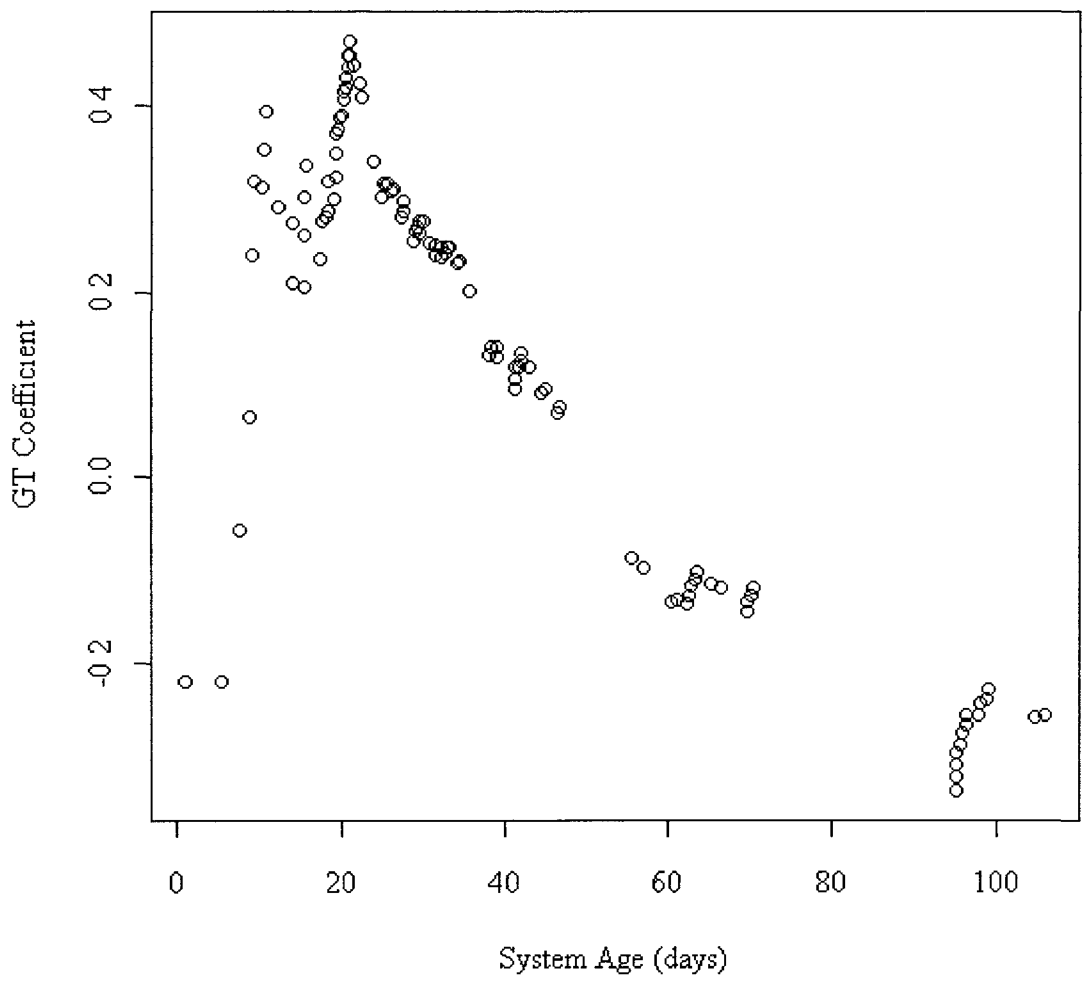


Figure 5.60 ROCOF vs. System Age, System 21, LANL Data

Table 5.21 FI Coefficients and ROCOF, System 22, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
22	0.4140757	0.443951	0.6050619	-0.1168699

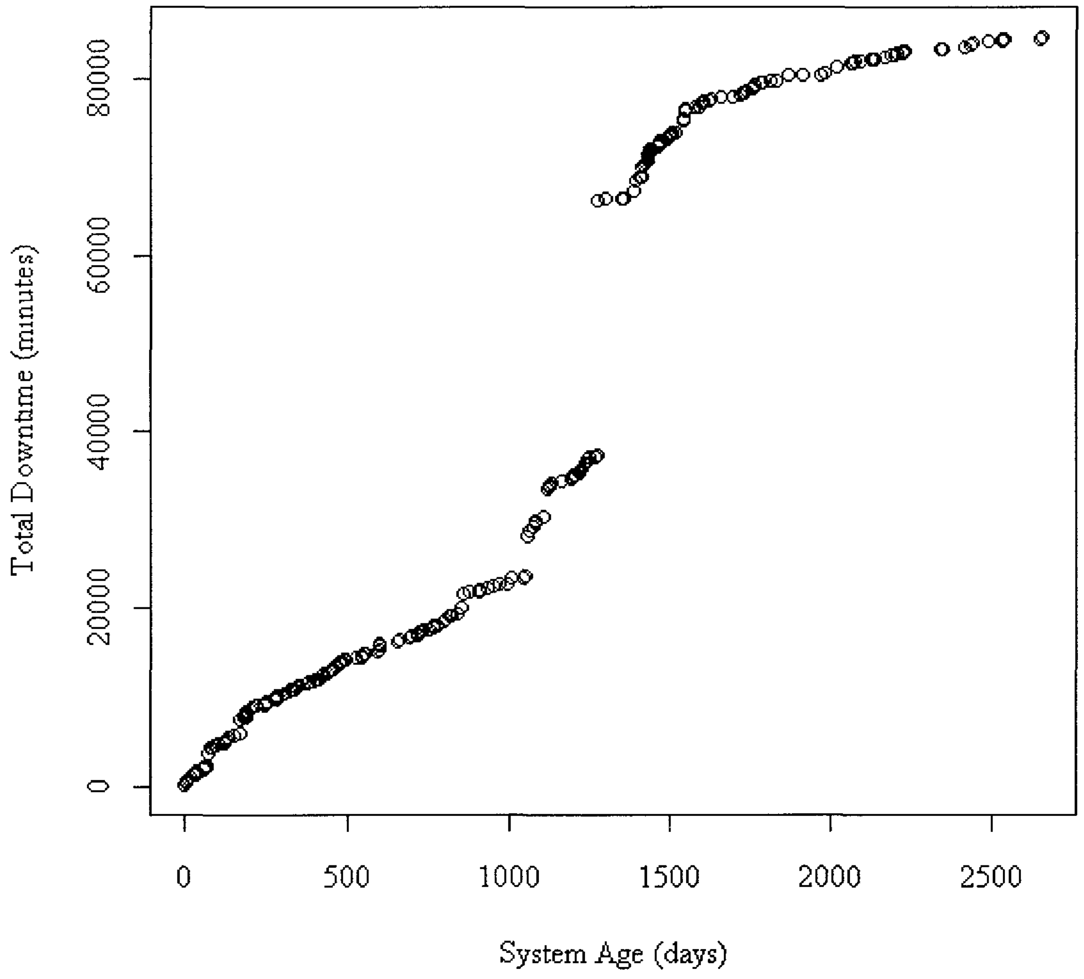


Figure 5.61 Total Downtime vs. System Age, System 22, LANL Data

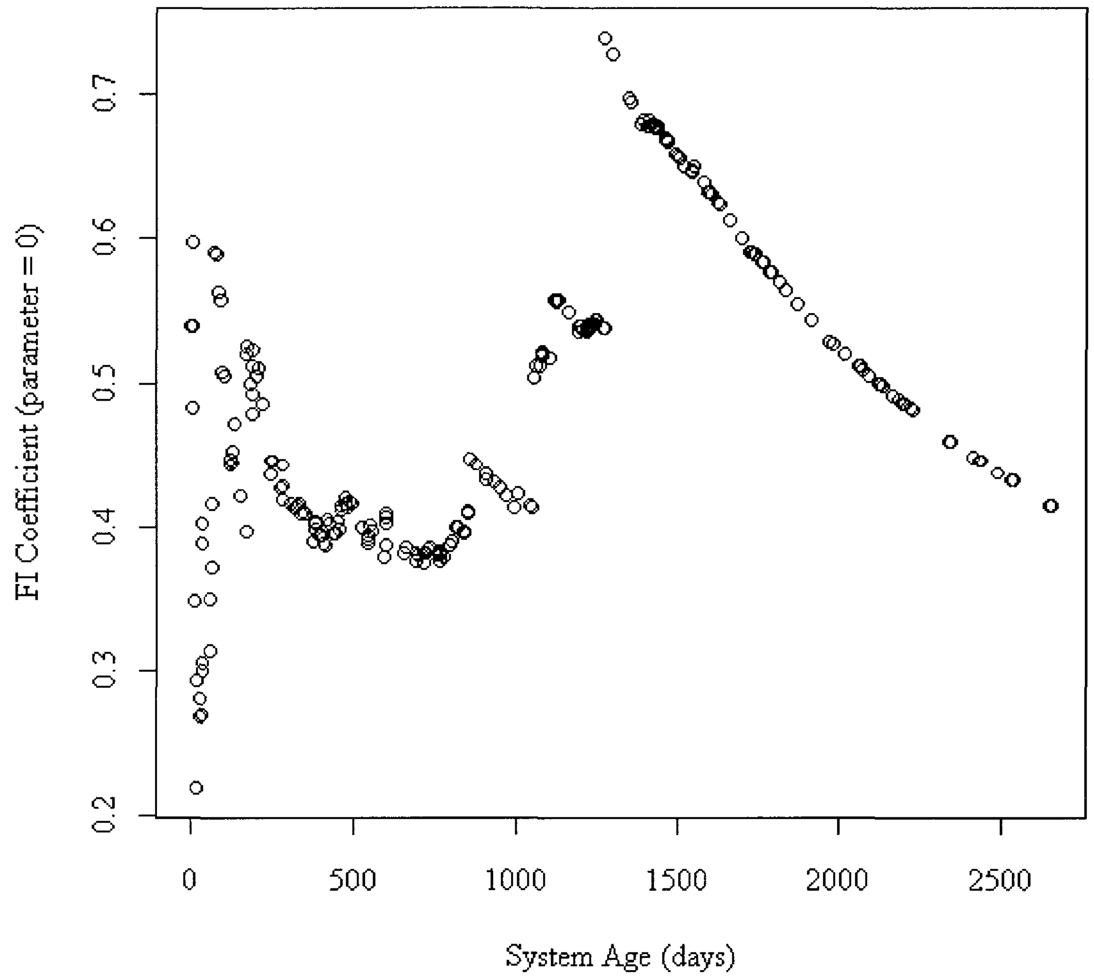


Figure 5.62 FI Coefficient vs. System Age, System 22, LANL Data

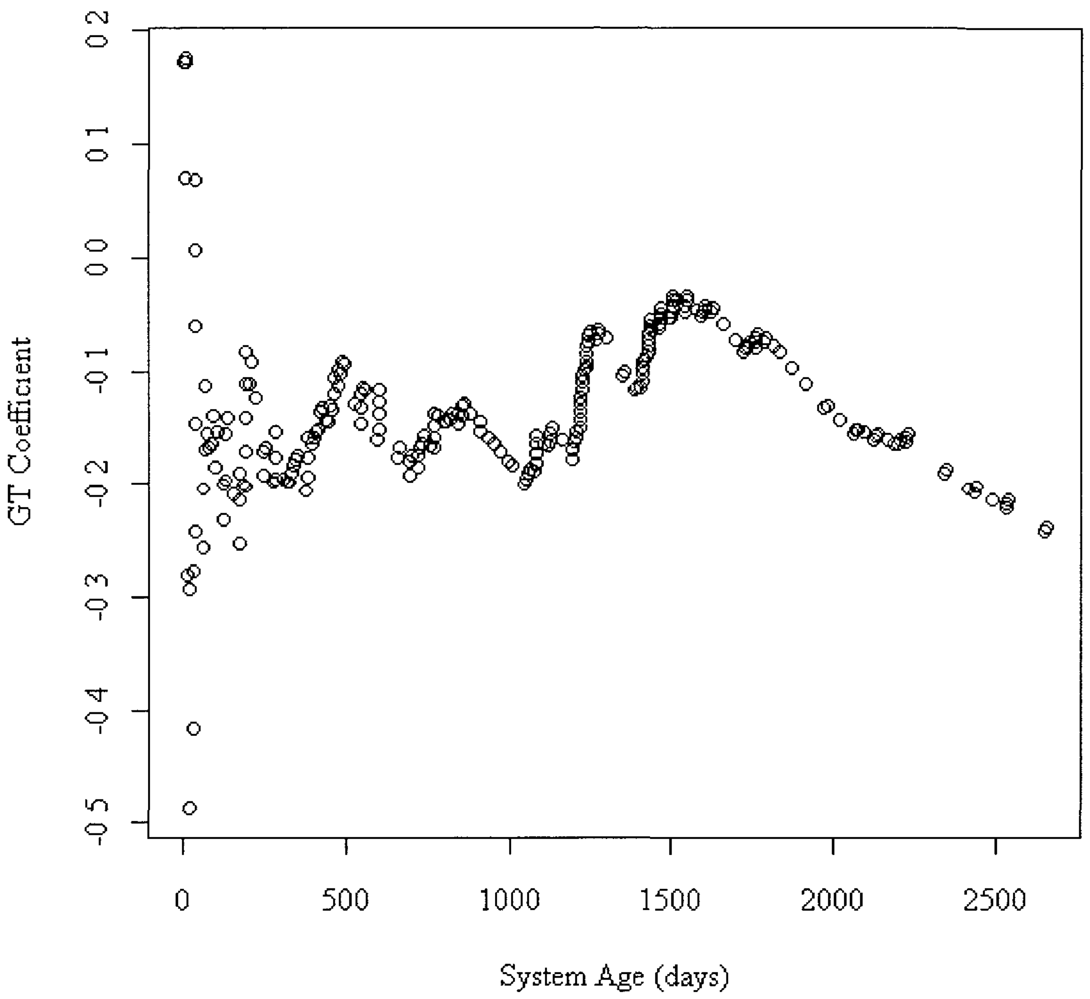


Figure 5.63 ROCOF vs. System Age, System 22, LANL Data

Table 5.22 FI Coefficients and ROCOF, System 23, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
23	0.4729281	0.4977982	0.6634825	-0.003102837

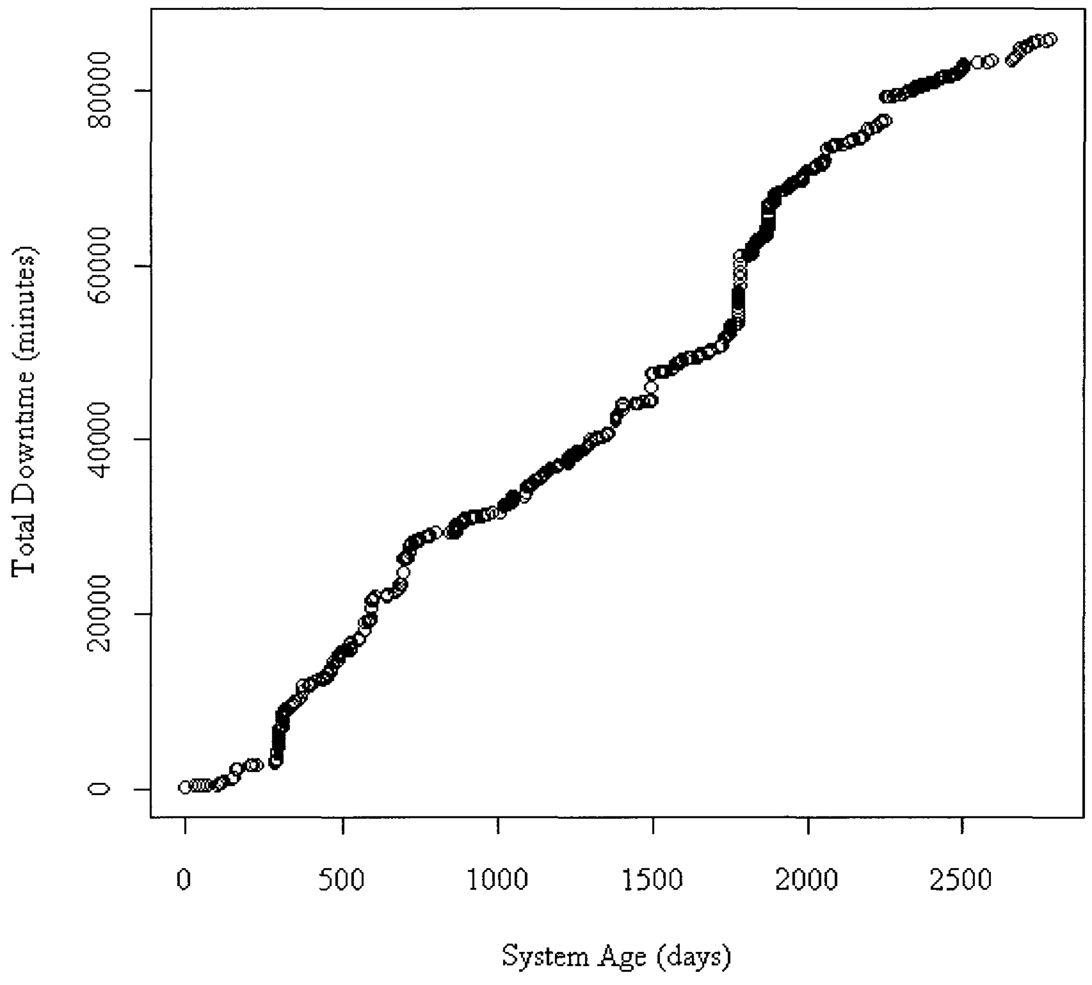


Figure 5.64 Total Downtime vs. System Age, System 23, LANL Data

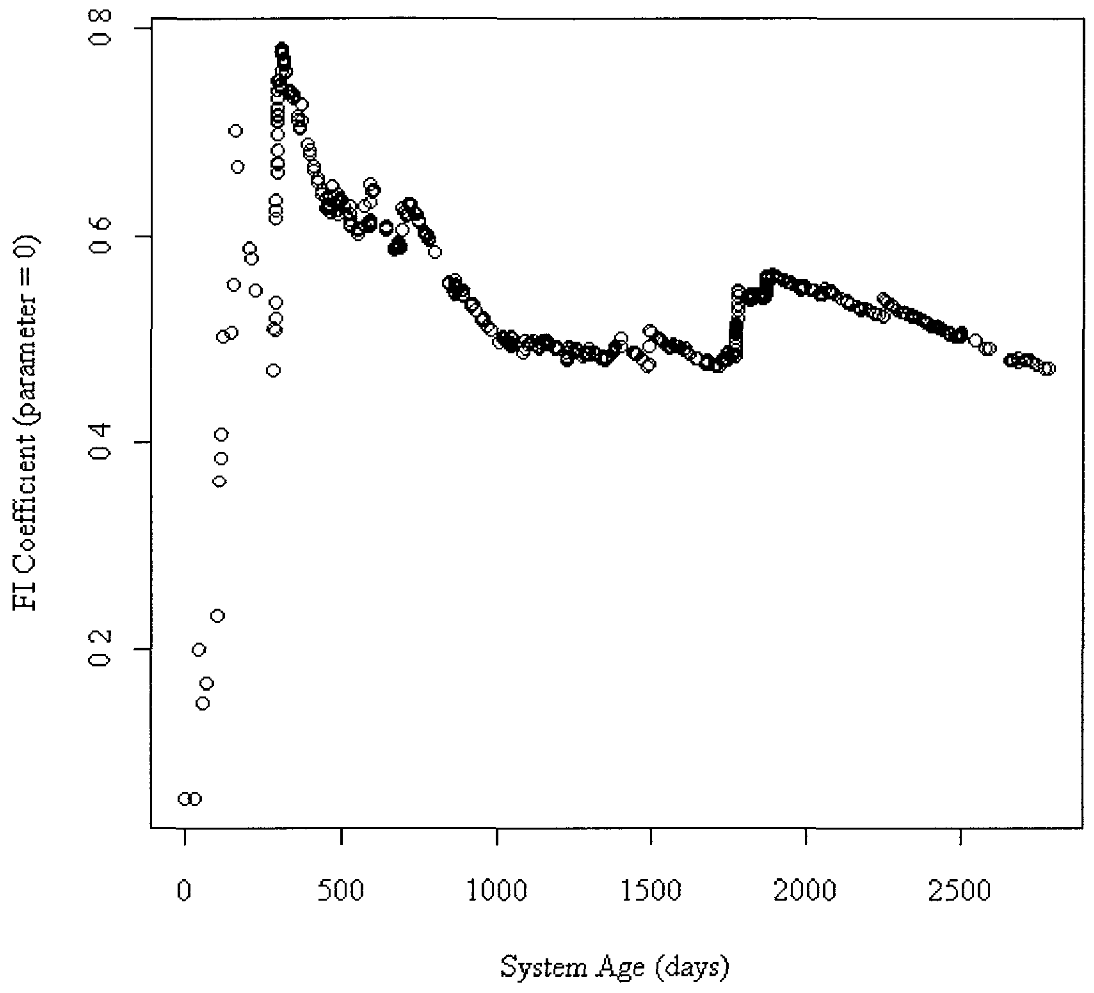


Figure 5.65 FI Coefficient vs. System Age, System 23, LANL Data

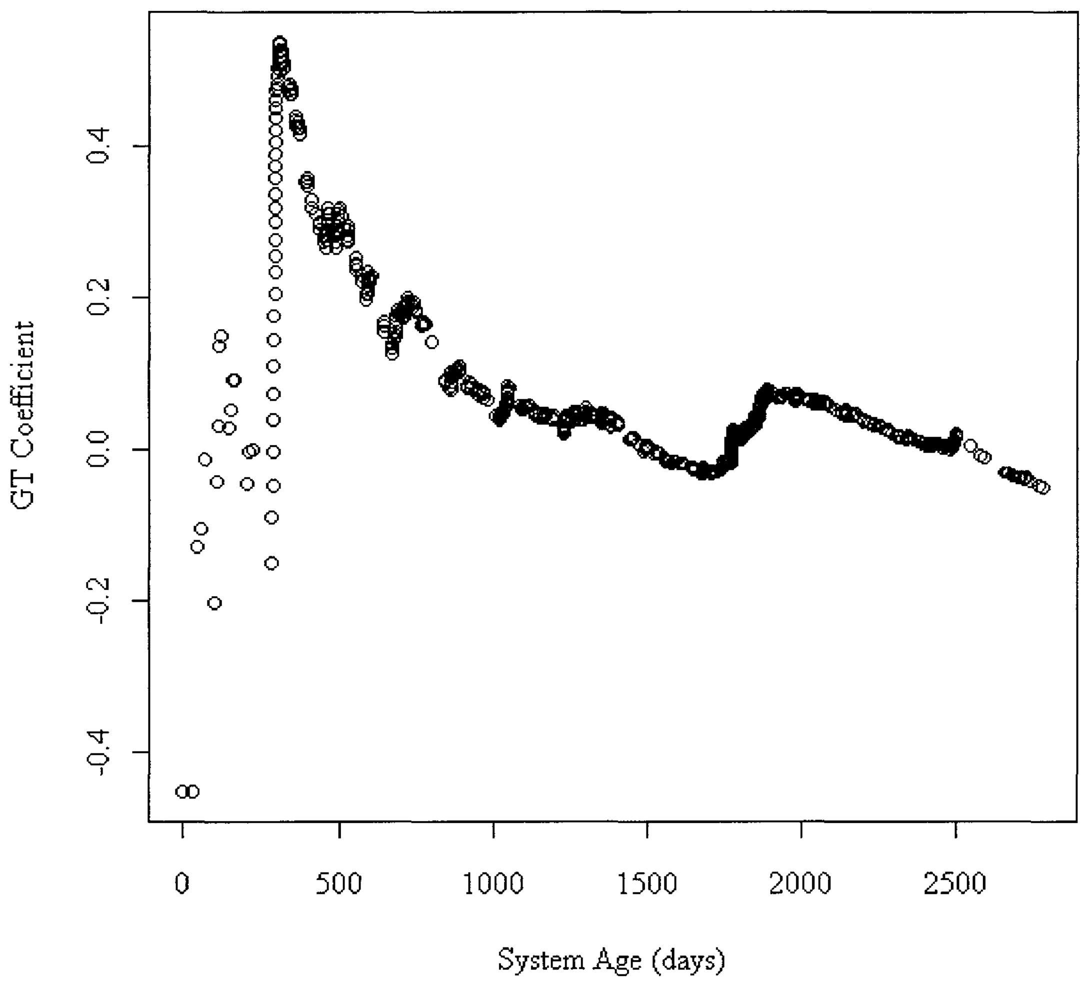


Figure 5.66 ROCOF vs. System Age, System 23, LANL Data

Table 5.23 FI Coefficients and ROCOF, System 24, LANL Data

System	FI (ε = 0)	FI (ε = 0.20)	FI (ε = 0.99)	GT (N(T))
24	0.3328124	0.3583177	0.514077	-0.3460315

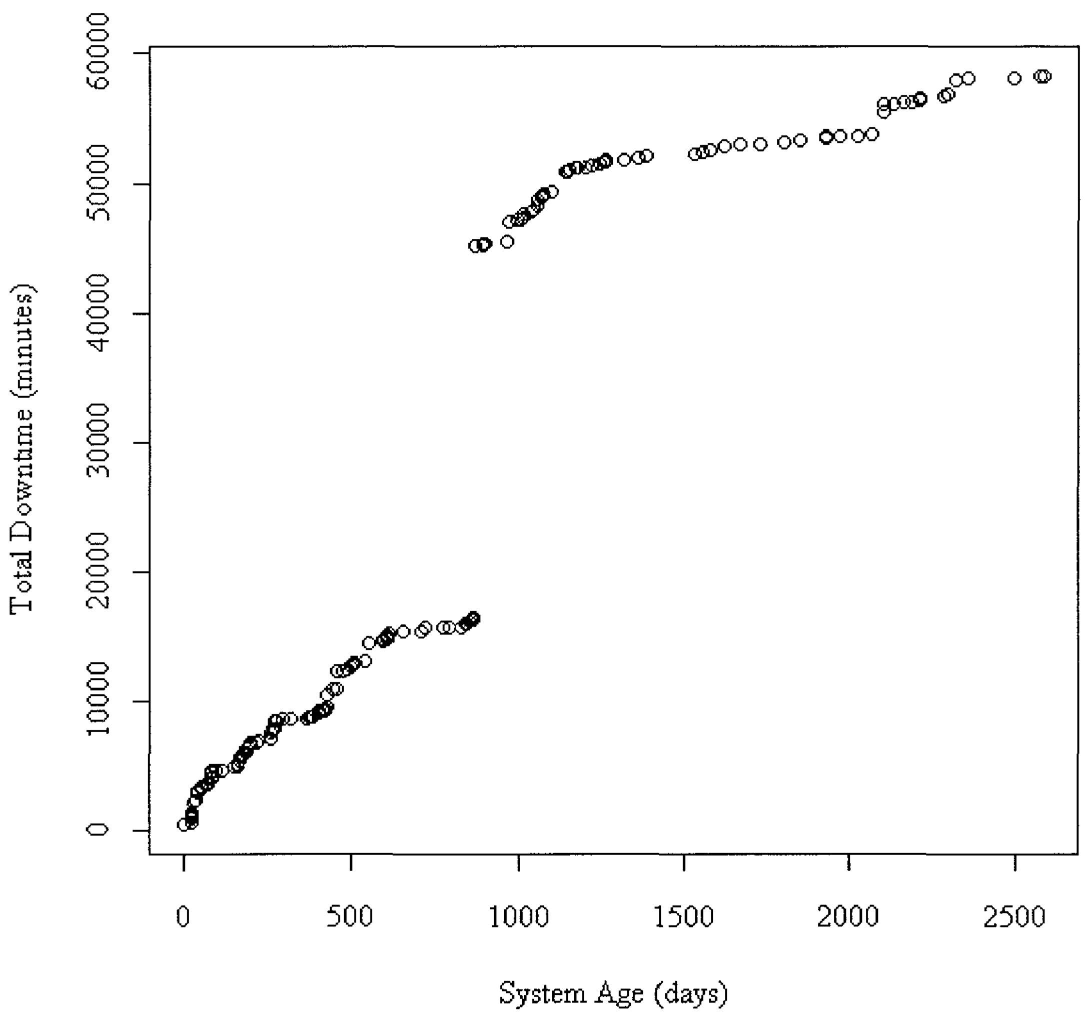


Figure 5.67 Total Downtime vs. System Age, System 24, LANL Data

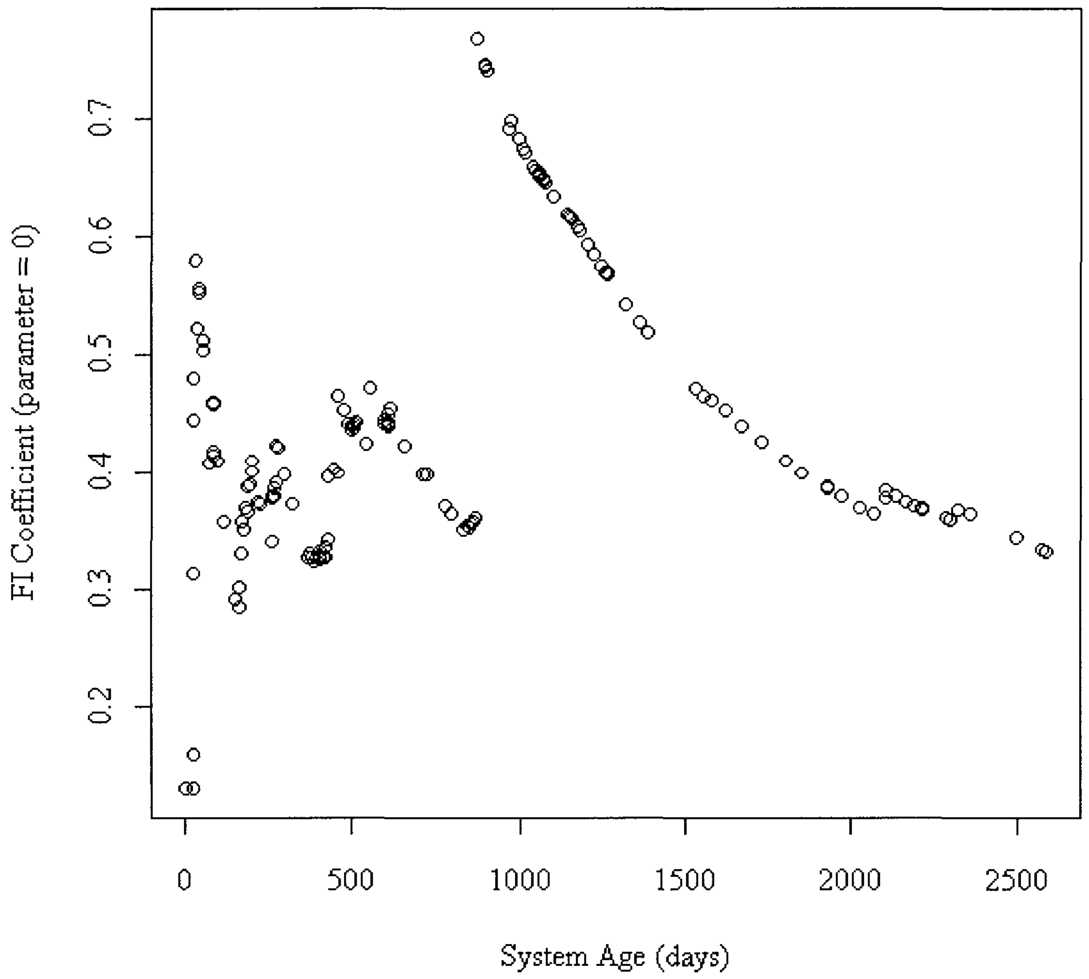


Figure 5.68 FI Coefficient vs. System Age, System 24, LANL Data

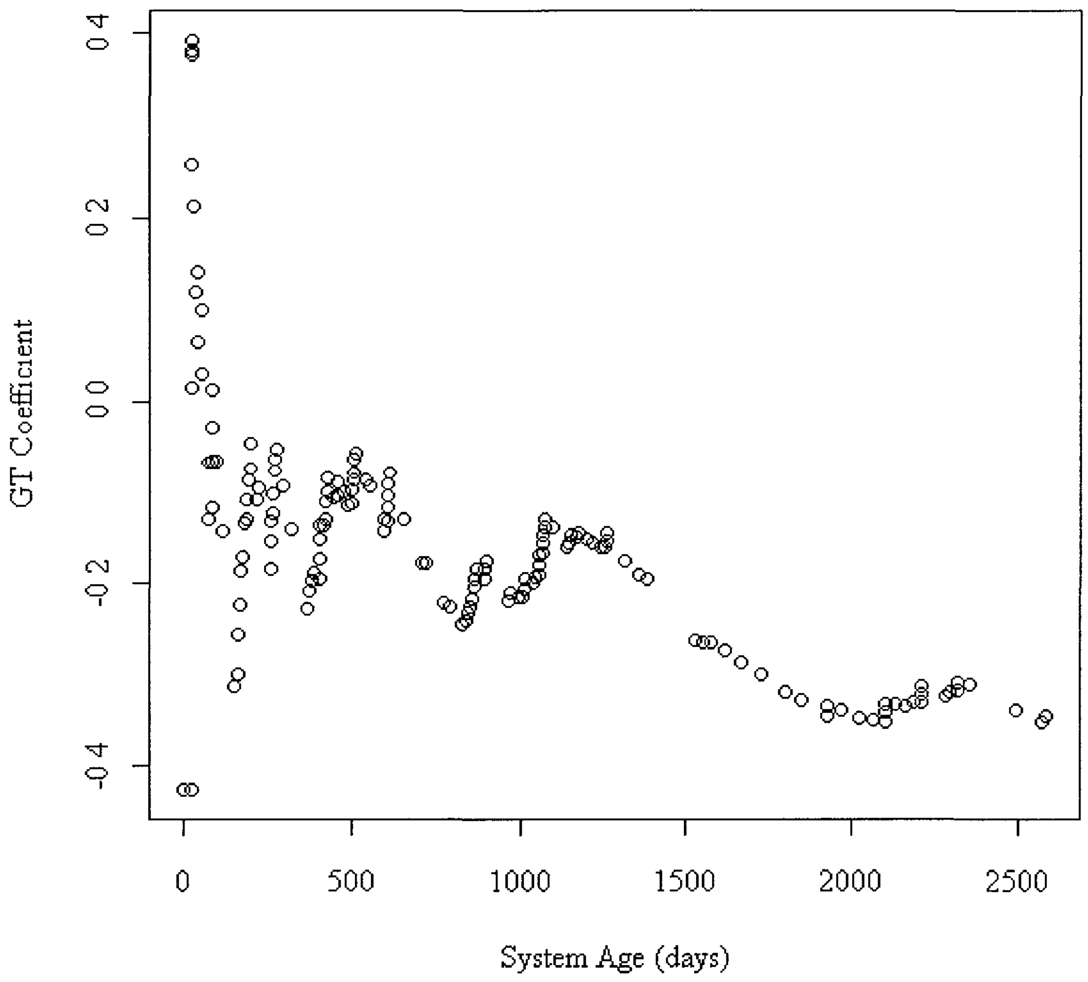


Figure 5.69 ROCOF vs. System Age, System 24, LANL Data

Table 5.24 Number of Systems with Increasing vs. Decreasing ROCOF, LANL Data

Increasing ROCOF	Decreasing ROCOF
16	7

Table 5.25 FI Coefficient Averages, Increasing vs. Decreasing ROCOF, LANL Data

Systems With:	FI ($\varepsilon = 0$)	FI ($\epsilon = 0.20$)	FI ($\epsilon = 0.99$)
Increasing ROCOF	0.405614	0.42934	0.632599
Decreasing ROCOF	0.430932	0.455128	0.659323

Table 5.26 Coefficient Averages, All Systems, LANL Data

Index	FI ($\varepsilon = 0$)	FI ($\epsilon = 0.20$)	FI ($\epsilon = 0.99$)	GT (N(T))
Avg.	0.418823	0.442795	0.646542	0.052074

5.3 Conclusions

Sixteen of the 23 systems exhibit increasing ROCOF. Seven of the systems have decreasing ROCOF. However, increasing or decreasing ROCOF has little impact on the failure volatility of a given system, as systems with increasing ROCOF generated an FI ($\epsilon = 0.99$) coefficient of 0.632599 and systems with decreasing ROCOF generated an FI ($\epsilon = 0.99$) coefficient of 0.659323, values which are not significantly different.

On average, the systems generated an FI ($\varepsilon = 0$) coefficient of 0.418823, an FI ($\epsilon = 0.20$) coefficient of 0.442795 and an FI ($\epsilon = 0.99$) coefficient of 0.646542. Similar to the relationship reported in the inequality index results from Chapter 3, this suggests that the higher failure volatility exists amongst the larger failures incurred by a system, and less failure volatility exists amongst the smaller failures.

Large periods of non-failure behavior significantly impact the failure volatility (and as such the resulting FI coefficient) of a given system. System 21, for example, contains a significant period of no failure activity toward the end of its lifetime, resulting in an FI ($\epsilon = 0.99$) coefficient of 0.821141. This was the highest FI coefficient reported by any system. Likewise, System 20, which contains the most regular time intervals between each failure, reported an FI ($\epsilon = 0.99$) coefficient of 0.3392873, the lowest of any system.

In conclusion, the amount of time between successive failures significantly contributes to the overall failure volatility exhibited by an HPC system. Such time periods are not accounted for using traditional inequality indices such as the Gini or Atkinson index. The FI, however, does capture these time periods, and results show that systems with more regular time intervals between system failures exhibit lower levels of volatility than those with prolonged periods of highly unequal time between successive failures. FI coefficients are less affected by single large failures than Gini or Atkinson coefficients, though the level of inequality in the impact of each failure exhibited by the system does still play a role in the volatility reported by a Failure Index coefficient.

The relationship between system volatility as captured by FI coefficients and system failure rate as captured by GT coefficients can be seen in the plots of FI and GT coefficients in relation to time. A system's ROCOF levels as the system ages, while system volatility continues to fluctuate in the latter days of a system's lifespan.

The following chapter closes the dissertation by drawing results from the entirety of the work and proposing future uses of the Failure Index. The source code used to generate the FI and GT coefficients reported in this chapter are given in Appendix B and Appendix C, respectively. A list of the acronyms used in this dissertation is given in Appendix A.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this dissertation we introduced and demonstrated a Failure Index for High Performance Computing applications. The FI is a time-dependent implementation of the Atkinson Index and captures the failure volatility exhibited by an HPC system in a given period. It is less affected by single large failures and more affected by fluctuations in failure strength over time and irregular time intervals between successive failures. Existing inequality indices such as the Gini and Atkinson index do not take time into consideration. System failure rates level as HPC system's age, while system volatility does not.

Background information was given in the first two chapters in order to demonstrate the need for such a metric, including a general introduction to HPC resilience, a look at previous work in statistically analyzing large HPC datasets, and a means to generate application-specific data from a large-scale HPC application.

Inequality indices were then introduced from a statistical standpoint and their applicability was demonstrated via application to a dataset containing failure information from 23 machines housed at Los Alamos National Laboratory from 1996 to 2005.

The FI was then mathematically introduced in Chapter 4, along with an overview of the GT Index, which captures a system's rate of occurrence of failure (ROCOF). In Chapter 5, the FI and GT Indices were applied to the LANL dataset, with results showing time plays an important role in the failure volatility exhibited by a given system. Results also show that a system's ROCOF and its level of failure volatility are not statistically related.

6.2 Future Work

Suggested future work in this area consists largely of implementing the Failure Index in real time on an existing suitably-scaled HPC system. Specifically, it is suggested that an FI module be created for the SLURM platform, which in its current form is much like the proposed Resilience Framework mentioned in the first chapter of this dissertation.

This effort would require a large dedication to both application programming and tuning as well as interface development. However, the utility of a real time FI is clear – one would be able to react to changes in application failure volatility in accordance with real time fluctuations of a system's FI coefficient, which when combined with existing metrics such as MTBF and the 'nines' system reliability allows for greater flexibility in how to proactively prepare for the failure behavior exhibited by a given system.

Future work also includes combining the FI with the real-time application metric generation software Gilgamesh, developed by the author and discussed in Chapter 2. This would allow for the generation of live per-application FI coefficients immediately after the extraction of relevant failure-related data from that application. This would allow for the real time monitoring of an application's failure volatility.

APPENDIX A

LIST OF ACRONYMS

ASCI – Advanced Scientific Computing Initiative

BG/L – Blue/Gene L

BLCR – Berkeley Laboratory Checkpoint/Restart

Blobs – Binary Large OBjectS

C/R – Checkpoint/Restart

CIF – Cumulative Intensity Function

CLI – Command Line Interface

CMU – Carnegie Melon University

CPU – Central Processing Unit

CTD – Collection of Training Data

DOE – Department of Energy

FC() – Failure Classification Function

FENCE – Fault-Aware ENabled Computing Environment

FI – Failure Index

FLOPS – Floating Point Operations Per Second

FPM – Failure Probability Model

GUI – Graphical User Interface

HPC – High Performance Computing

K-S – Kolmogorov-Smirnov

LAM – Local Area Multicomputer MPI

LANL – Los Alamos National Laboratory

LLNL – Lawrence Livermore National Laboratory

MPI – Message Passing Interface

MTBF – Mean Time Between Failure

MTTF – Mean Time To Failure

MTTR – Mean Time To Repair

OSS – Open|SpeedShop

ROCOF – Rate of Occurrence of Failure

RRE – Resilience-Related Event

SEP – Similar Event Prediction

SGI – Silicon Graphics, Inc.

TTF – Time To Failure

TTR – Time To Repair

APPENDIX B

FAILURE INDEX SOURCE CODE

```
#FISCRIPT.R -- Computes a time-dependent Failure Index coefficient for
the given dataset
#AUTHOR: Clayton Chandler, Louisiana Tech University

#load the necessary libraries
library(caTools)

#NOTE: dataset must be ran before running script.
#NOTE: parameter must be set before running script.

#set the necessary values, assuming data has already been loaded
idx = 2:length(data$time)
T <- as.double(data$time[idx][(length(data$time)-1)])
FT <- as.double(data$totaldowntime[idx][(length(data$downtime)-1)])

#get the mean values for both time and downtime
EFT <- as.double(mean(data$totaldowntime))
ET <- as.double(mean(data$time))

#find the integral using the trapezoid rule and the given parameter
INTEGRAL <- as.double( (data$time[idx] - data$time[idx-1]) %*%
((data$totaldowntime[idx]^(1-parameter)) + (data$totaldowntime[idx-
1]^(1-parameter))) / 2)

#Multiply the integral by (1 / T)
INSIDEPRODUCT <- as.double(INTEGRAL * (1/T))

#Obtain the Inside Product to the (1 / 1-parameter) power
POWER <- as.double(INSIDEPRODUCT ^ as.double(1/(1-parameter)) )
```

```
#Multiply the result by 1/E(T)

OUTSIDEPRODUCT <- as.double(POWER * as.double(1/FT) )


#Take one minus this to get final result

FI <- as.double(1 - OUTSIDEPRODUCT)


#Return the resulting FI coefficient

return (as.double(FI))
```

APPENDIX C

GT INDEX SOURCE CODE

```
#GTSCRIPT.R -- Computes a GT Index coefficient for the given dataset
#AUTHOR: Clayton Chandler, Louisiana Tech University

#load the necessary libraries
library(caTools)

#NOTE: dataset must be ran before running script.

#NOTE: parameter must be set before running script.

#set the necessary values, assuming data has already been loaded
idx = 2:length(data$quantile)
T <- as.double(data$quantile[idx][(length(data$quantile)-1)])
N  <-  as.double(data$numberofevents[idx][(length(data$numberofevents)-
1)])

#find the integral using the trapezoid rule and the given parameter
GTINTEGRAL  <-  as.double(  (data$quantile[idx]  -  data$quantile[idx-1])
%*% (data$numberofevents[idx] + data$numberofevents[idx-1]) / 2)

#Obtain the numerator
GTNUMERATOR <- as.double(GTINTEGRAL * 2)

#Obtain the denomenator
GTDENOMENATOR <- as.double(T * N)

#Get the fraction
GTFRACTION <- as.double(GTNUMERATOR / GTDENOMENATOR)
```

```
#Get the coefficient

GT <- as.double(1 - GTFRACTION)


#Return the resulting GT coefficient

return (as.double(GT))
```

REFERENCES

[1] R. Haring, IBM T.J. Watson, Personal communications, July 2005.

[2] N. Taerat, N. Nakisinehaboon, C. Chandler, J. Elliot, C. Leangsuksun, G. Ostrouchov, and S. L. Scott, "Using Log Information to Perform Statistical Analysis on Failures Encountered by Large-Scale HPC Deployments," in *Proceedings of the 2008 High Availability and Performance Computing Workshop*, 2008.

[3] I. Philp, "Software failures and the road to a petaflop machine," in HPCRI: 1st Workshop on High Performance Computing Reliability Issues, in *Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA-11)*, IEEE Computer Society, 2005.

[4] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *SOSP 2003: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pp. 29–43, 2003.

[5] "Advanced configuration & power interface," http://www.acpi.info/, 2004, last accessed on November 12, 2011.

[6] H. Song, C. Leangsuksun, and R. Nassar, "Availability modeling and analysis on high performance cluster computing systems," in *First International Conference on Availability, Reliability and Security*, pp. 305–313, 2006.

[7] S. Rani, C. Leangsuksun, A. Tikotekar, V. Rampure, and S. Scott, "Toward efficient failure detection and recovery in hpc," in High Availability and Performance Computing Workshop, 2006.

[8] S. Sankaran, J. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman, "The LAM/MPI checkpoint/restart framework: System-initiated checkpointing," in *Proceedings, LACSI Symposium*, October 2003.

[9] J. Duell, "The design and implementation of Berkeley Lab's Linux checkpoint/ restart," Tr, Lawrence Berkeley National Laboratory, 2000.

[10] C. Hsu and W. Feng, “A power-aware run-time system for high-performance computing,” in *SC ’05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” in *Symposium on Operating Systems Principles*, pp. 164–177, 2003.

[12] S. Chakravorty, C. Mendes, and L. Kale, “Proactive fault tolerance in large systems,” in HPCRI: 1st Workshop on High Performance Computing Reliability Issues, in *Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA-11)*, IEEE Computer Society, 2005.

[13] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo, “BlueGene/L failure analysis and prediction models,” in *The International Conference on Dependable Systems and Networks 2006*, pp. 425–434, 2006.

[14] S. Fu and C.-Z. Xu, “Exploring event correlation for failure prediction in coalitions of clusters,” in *SC ’07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pp. 1–12, 2007.

[15] S. Chakravorty, C. Mendes, and L. Kale, “Proactive fault tolerance in mpi applications via task migration,” in *The International Conference on High Performance Computing*, 2006.

[16] “Advanced configuration & power interface”, http://www.acpi.info/, 2005, last accessed on November 12, 2011.

[17] R. Sahoo, A. Oliner, I. Rish, M. Gupta, J. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam, “Critical event prediction for proactive management in large-scale computer clusters,” in *KDD ’03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 426–435, 2003.

[18] A. Oliner, R. Sahoo, J. Moreira, M. Gupta, and A. Sivasubramaniam, “Fault-aware job scheduling for bluegene/l systems,” in *International Parallel and Distributed Processing Symposium*, 2004.

[19] T. Liu, Z. Ma, and Z. Ou, “A novel processs migration method for mpi applications,” in *Symposium on Dependable Computing*, pp. 247-251, 2000.

[20] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” In *Symposium on Operating Systems Principles*, pp. 164–177, 2003.

[21] A. B. Nagarajan and F. Mueller, "Proactive fault tolerance for hpc with xen virtualization," Technical Report TR 2007-1, Dept. of Computer Science, North Carolina State University, 2007.

[22] H. Song, C. Leangsuksun, R. Nassar, N. R. Gottumukkala, S. Scott, and A. Yoo, "Availability modeling and analysis on high performance cluster computing systems," in *Proceedings of the International Symposium on Frontiers in Availability, Reliability and Security (FARES)*, April 2006.

[23] N. R. Gottumukkala, C. Leangsuksun, R. Nassar, and S. L. Scott, "Reliability-aware resource allocation in HPC systems," in *Proceedings of the IEEE International Conference on Cluster Computing*, 2007.

[24] G. Vallee and C. Morin, "A Framework for High Availability Based on Single System Image," in *Proceedings of High Availability and Performance Computing Workshop (HAPCW) 2005*, October 2005.

[25] Y. Liu and C. Leangsuksun, "Reliability-aware checkpoint/restart scheme: A performability trade-off," in *Proceedings of IEEE International Conference on Cluster Computing (Cluster) 2005*, September 26-30, 2005.

[26] A. Tikotekar, C. Leangsuksun, and S. L. Scott, "On the survivability of standard MPI applications," in *Proceedings of 7th LCI International Conference on Linux Clusters: The HPC Revolution 2006*, May 2-4 2006.

[27] Y. Zhang, M. S. Squillante, A. Sivasubramaniam, and R. K. Sahoo, "Performance implications of failures in large-scal cluster scheduling," Job Scheduling Strategies for Parallel Processing, vol. 3277/2005, pp. 233–252, 2005.

[28] T. Y. Lin and D. P. Siewiorek, "Error log analysis: Statistical modeling and heuristic trend analysis," IEEE Transactions on Reliability, vol. 39, no. 4, pp. 419–432, 2000.

[29] I. Lee and R. K. Iyer, "Analysis of software halts in tandem system," in *Proceedings 3rd Intl. Software Reliability Engineering*, pp. 227–236, October 1992.

[30] K. Goseva-Popstojanova and K. S. Trivedi, "Failure correlation in software reliability models," IEEE Transactions on Reliability, vol. 49, no. 1, pp. 37–48, 2000.

[31] T. Heath, R. P.Martin, and T. D.Nguyen, "Improving cluster availability using workstation validation," in *Proceedings of the 2002 ACM SIGMETRICS Conference*, pp. 217–227, 2002.

[32] D. Eager, J. Zahorjan, and E. Lazowska, "Speedup versus efficiency in parallel systems," IEEE Transactions on Computers, vol. 38, no. 3, pp. 408–423, 1989.

[33] V. Kumar and A. Gupta, "Analysis of scalability of parallel algorithms and architectures: a survey," in *Proceedings of the 5th international Conference on Supercomputing*, pp. 396–405, June 1991.

[34] S. Shatz, J.-P.Wang, and M. Goto, "Task allocation for maximizing reliability of distributed computer systems," IEEE Transactions on Computers, vol. 41, no. 9, pp. 1156–1168, 1992.

[35] D. Nicol and F. Willard, "Problem size, parallel architecture, and optimal speedup," Parallel and Distributed Computing, vol. 5, no. 4, pp. 404–420, 1988.

[36] J. W. Young, "A first order approximation to the optimum checkpoint interval," Communications of the ACM, vol. 17, no. 9, pp. 530–531, 1974.

[37] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," Future Gener. Comput. Syst., vol. 22, no. 3, pp. 303–312, 2006.

[38] R. Geist, R. Reynolds, and J. Westall, "Selection of a checkpoint interval in a critical-task environment," Transactions on Reliability, vol. 37, no. 4, pp. 395–400, 1988.

[39] K. F. Wong and M. Franklin, "Distributed computing systems and checkpointing," in *Proceedings of the 2nd International Symposium on High Performance Distributed Computing*, pp. 224–233, 1993.

[40] Y. Ling, J. Mi, and X. Lin, "A variational calculus approach to optimal checkpoint placement," IEEE Transactions on Computers, vol. 50, no. 7, pp. 699-707, 2001.

[41] T. Ozaki, T. Dohi, and H. Okamura, "Distribution-free checkpoint placement algorithms based on min-max principle," IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 2, pp. 130-140, 2006.

[42] J. Stearley, "Defining and measuring supercomputer reliability, availability, and serviceability (ras)," in *Proceedings of the 2005 Linux Clusters Institute Conference*, 2005.

[43] J. S. Plank and M. G. Thomason, "The average availability of parallel checkpointing systems and its importance in selecting runtime parameters," in *International Symposium on Fault-Tolerant Computing*, June 1999.

[44] S. I. Feldman and C. B. Brown, "Igor: a system for program debugging via reversible execution," in *Proceedings of the 1988 ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging*, pp. 112–123, 1989.

[45] N. H. Vaidya, "A case for two-level distributed recovery schemes," in *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pp. 64–73, 1995.

[46] J. Plank, J. Xu, and R. Netzer, "Compressed differences: An algorithm for fast incremental checkpointing," Technical Report CS-95-302 (TP CS-95-302), The University of Tennessee at Knoxville, 1995.

[47] A. C. Palaniswamy and P. A. Wilsey, "An analytical comparison of periodic checkpointing and incremental state saving," ACM SIGSIM Simulation Digest, vol. 23, no. 1, pp. 127–134, 1993.

[48] M. Harchol-Balter and A. B. Downey, "Exploiting process lifetimes distributions for dynamic load balancing," Transactions on Computer Systems, vol. 15, no. 3, pp. 253–285, 1997.

[49] R. F. deMello and L. J. Senger, "A new migration model based on the evaluation of processes load and lifetime on heterogeneous computing environments," in the *16th Symposium of Computer Architecture and High Performance Computing*, pp. 222–227, 2004.

[50] J. Cao, Y. Li, and M. Guo, "Process migration for mpi applications based on coordinated checkpoint," in *ICPADS '05: Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS'05)*, pp. 306–312, 2005.

[51] X.-H. Sun, Z. Lan, Y. Li, H. Jin, and Z. Zheng, "Towards a fault-aware computing environment," in *Proceedings of the 2008 High Availability and Performance Computing Workshop*, 2008.

[52] F. Salfner, M. Schieschke, and M. Malek, "Predicting failures of computer systems: a case study for a telecommunication system," *Parallel and Distributed Processing Symposium 2006*, pp. 8-12, April 2006.

[53] J. T. Daly, "A model for predicting the optimum checkpoint interval for restart dumps," in *International Conference on Computational Science*, pp. 3–12, 2003.

[54] T. J. Hacker and Z. Meglicki, "Using queue structures to improve job reliability," in *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, pp. 43–54, 2007.

[55] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance-computing systems," in *Proceedings of the 2006 International Conference on Dependable Systems and Networks (DSN- 2006)*, June 2006.

[56] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," in *International Parallel and Distributed Processing Symposium*, April 2008.

[57] Z. Xue, X. Dong, S. Ma, and W. Dong, "A survey on failure prediction of large-scale server clusters," *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007*, vol. 2, pp. 733–738, July 30 2007-Aug. 1 2007.

[58] W. Kang and A. Grimshaw, "Failure prediction in computational grids," *Simulation Symposium*, 2007, pp. 275–282, March 2007.

[59] N. Ye, *The Handbook of Data Mining*. Lawrence Erlbaum, 2003.

[60] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E.Moreira, S.Ma, R. Vilalta, and A. Sivasubramaniam, "Critical event prediction for proactive management in large-scale computer clusters," in *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 426–435, ACM, 2003.

[61] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A meta-learning failure predictor for blue gene/l systems," in *International Conference on Parallel Processing, 2007*, pp. 40–40, September 2007.

[62] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "An adaptive semantic filter for blue gene/l failure log analysis," in *IEEE International Parallel and Distributed Processing Symposium, 2007*, pp. 1–8, 26-30 March 2007.

[63] G. Hamerly and C. Elkan, "Bayesian approaches to failure prediction for disk drives," in *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 202–209, 2001.

[64] J. Pearl, "Causality: Models, Reasoning, and Inference". IIE Transacations, vol. 34, no. 6, pp. 583-589.

[65] J. Becklehimer, C. Willis, J. Lothian, D. Maxwell, and D. Vasil, "Real time health monitoring of the cray xt3/xt4 using the simple event correlator (sec)," in *Proceedings of Cray User Group 2007*, 2007.

[66] I. Philip, "Software Failures and the Road to a Petaflop Machine," in HPCRI: 1st Workshop on High Performance Computing Reliability Issues, in *Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA-11)*, 2005.

[67] H. Song, C. Leangsuksun, and R. Nassar, "Availability Modeling and Analysis on High Performance Cluster Computing Systems," in *First International Conference on Availability, Reliability, and Security*, pp. 305-313, 2006.

[68] "Los Alamos National Laboratory Operational Data to Support and Enable Computer Science Research," http://institutes.lanl.gov/data/fdata/, last accessed on November 1st, 2011.

[69] B. Schroeder and G. Gibson, "A Large-Scale Study of Failures in High Performance-Computing Systems," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, June, 2006.

[70] F. Salfner and M. Malek, "Using hidden semi-markov models for effective online failure prediction," *26th IEEE International Symposium on Reliable Distributed Systems, 2007*, pp. 161–174, October 2007.

[71] N. Taerat, N. Nakisinehaboon, C. Chandler, J. Elliot, C Leangsuksun, G Ostrouchov, and S. L. Scott, "Using Log Information to Perform Statistical Analysis on Failures Encountered by Large-Scale HPC Deployments," in *Proceedings of the 2008 High Availability and Performance Computing Workshop*, 2008.

[72] "Sisyphus Project," http://www.cs.sandia.gov/~jrstear/sisyphus/, last accessed on November 1st, 2011.

[73] "Open|SpeedShop," http://www.openspeedshop.org, last accessed on November 11th, 2011.

[74] S.-Z. Zhang, N.-H. Yang, and X.-K. Wang, "Construction and application of bayesian networks in flood decision supporting system," *2002 International Conference on Machine Learning and Cybernetics, 2002*, vol. 2, pp. 718–722, 2002.

[75] M. L. Wong and K. S. Leung, "An efficient data mining method for learning bayesian networks using an evolutionary algorithm-based hybrid approach," IEEE Transactions on Evolutionary Computation, vol. 8, no. 4, pp. 378–404, Aug. 2004.

[76] A. Tikotekar, C. Leangsuksun, and S. L. Scott, "On the survivability of standard mpi applications," in *LCI International Conference on Linux Clusters: The HPC Revolution*, May 2006.

[77] "DPCL," http://dpcl.sourceforge.net/, last accessed on July 20th, 2011.

[78] "DynInst," http://www.dyninst.org/, last accessed on July 20th, 2011.

[79] G.Almasi, R. Bellofatto, J. Brunheroto, C. Cascaval, J. G. Castonos, L. Ceze, P. Crumlev, C. C. Erway, J. Gagliano, D. Lieber, X. Martorell, J. E. Moreira, A. Sanomiva and K. Strauss, "An Overview of the Blue Gene/L System Software Organization," Euro-Par 2003, *Parallel Processing2003*, pp.543-555, 2003.

[80] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A Meta-Learning Failure Predictor for Blue Gene/L Systems," *International Conference on Parallel Processing 2007*, pp. 40-47, 2007.

[81] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer and M. Snir, "Toward Exascale Resilience," in the International Journal of High Performance Computing Applications, vol. 23, no. 4, pp. 374-388, November 2009.

[82] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, A. Trefethen and M. Valero, "The International Exascale Software Project: A Call to Cooperative Action by the Global High-Performance Community," in the International Journal of High Performance Computing Applications, vol. 23, no. 4, pp. 309-322, November 2009.

[83] C. Chandler and N. DeBardeleben, "Toward Resilient High Performance Applications through Real-Time Reliability Metric Generation and Autonomous Failure Correction," *18th International Symposium on High Performance Distributed Computing*, 2009.

[84] C. Chandler, N. DeBardeleben and C. Leangsuksun, "Resilience Analysis of High Performance Computing Applications via Bayesian Pattern Recognition," (Poster) *The National Workshop on HPC Resilience*, August 2010.

[85] J. L. Gastwirth, "A General Definition of the Lorenz Curve," in Econometrica, vol. 39, pp. 1037-1039, November 1971.

[86] J. L. Gastwirth, "The Estimation of the Lorenz Curve and Gini Index," in Review of Economics and Statistics, vol 54, no. 3, pp. 306-316, August 1972.

[87] T. G. Holland, G. D. Peterson and A. Gonzalez, "A Cross-National Analysis of How Economic Inequality Predicts Biodiversity Loss," in Conservation Biology, vol. 23, no. 5, pp. 1304-1313, October 2009.

[88] E. Deconinck, E. Zhang, D. Coomans and Y. V. Heyden, "Classification Tree Models for the Prediction of Blood-Brain Barrier Passage of Drugs," in the Journal of Chemical Information and Modeling, pp. 1410-1419, February 2006.

[89] G. Henriques and R. Patel, "NAFTA, Corn, and Mexico's Agricultural Trade Liberalization," International Relations Center, February 26, 2009.

[90] B. Chandra, S. Mazumdar, V. Arena and N. Parimi, "Elegant Decision Tree Algorithm for Classification in Data Mining," in *the Third International Conference on Web Information Systems Engineering*, December 11, 2002.

[91] N. Mimoto, "Statistical Inference for Densities and Related Indices", Thesis, University of Western Ontario, 2008.

[92] E. Cutrini, "Using Entropy Measures to Disentangle Regional from National Localization Patterns," Regional Science and Urban Economics, vol. 39, no. 2, pp. 243-250, November 2009.

[93] T. Taerat, N. Naksinehaboon, C. Chandler, J. Elliot, C. Leangsuksun, G. Ostrouchov, and S. L. Scott, "Using Log Information to Perform Statistical Analysis on Failures Encountered by Large-Scale HPC Deployments," in *Proceedings of the 2008 High Availability and Performance Computing Workshop*, 2008.

[94] D. Van de gaer, N. Funnell and T. McCarthy, "Statistical inference for two measures of inequality when incomes are correlated," in Economics Letters, vol. 64, no. 3, pp. 295-300, September 1999.

[95] A. Tikotekar, "A Component-Based Survivability Framework for HPC Applications," Master's Thesis, Louisiana Tech University, Ruston, LA, USA, Mar. 2007.

[96] T. Liu, "High Availability and Performance Linux Cluster," Master's Thesis, Louisiana Tech University, Ruston, LA, USA, May 2004.

[97] V. K. Munganuru, "On Planned Downtime Reduction for High Performance Cluster Systems," Master's Thesis, Louisiana Tech University, Ruston, LA, USA, Feb. 2006.

[98] M. P. Kaminskiyy and V. Krivtsov, "A Gini-Type Index for Aging/Rejuvenating Objects," in Mathematical and Statistical Models and Methods in Reliability, Statistics and Industry and Technology, pp. 133-140, 2010.

[99] B. Schroeder and G. A. Gibson, "A Large-Scale Study of Failures in High-Performance Computing Systems," *the International Conference on Dependable Systems and Networks* 206, 2006.

[100] "The CIA World Factbook, 2009," https://www.cia.gov/library/publications/the-world-factbook/, last accessed on November 22nd, 2011.