A Framework for Characterization and Analysis of Software System Scalability

Leticia Duboc
Dept. of Computer Science
University College London
WC1E 6BT
United Kingdom
l.duboc@cs.ucl.ac.uk

David S. Rosenblum
Dept. of Computer Science
University College London
WC1E 6BT
United Kingdom
d.rosenblum@cs.ucl.ac.uk

Tony Wicks
Fortent Ltd
80-110 New Oxford Street
WC1A 1HB
United Kingdom
t.wicks@fortent.com

ABSTRACT

The term *scalability* appears frequently in computing literature, but it is a term that is poorly defined and poorly understood. The lack of a clear, consistent and systematic treatment of scalability makes it difficult to evaluate claims of scalability and to compare claims from different sources. This paper presents a framework for precisely characterizing and analyzing the scalability of a software system. The framework treats scalability as a *multi-criteria optimization problem* and captures the dependency relationships that underlie typical notions of scalability. The paper presents the results of a case study in which the framework and analysis method were applied to a real-world system, demonstrating that it is possible to develop a precise, systematic characterization of scalability and to use the characterization to compare the scalability of alternative system designs.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*design studies, measurement techniques, modeling techniques, performance attributes*; D.2.8 [**Software Engineering**]: Metrics—*product metrics*; D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Design, Economics, Measurement, Performance

1. INTRODUCTION

"I examined aspects of scalability, but did not find a useful, rigorous definition of it. Without such a definition, I assert that calling a system 'scalable' is about as useful as calling it 'modern'. I encourage the technical community to either rigorously define scalability or stop using it to describe systems." —Mark D. Hill, *What is Scalability?* [15]

As this quotation suggests, *scalability* is a term that appears frequently in computing literature, but it is a term that is poorly defined and poorly understood. It is an important attribute of computer systems that is frequently asserted but rarely validated in any meaningful, systematic way. Many papers employ phrases such as "X is a scalable system" or "Y is not a scalable approach" in order to convey some intuition about the system or approach at hand, but such phrases leave little more than the vague and ill-formed impression that "X is good" or "Y is bad".

There have been attempts at providing better definitions of the term in the years since the above quotation appeared [13, 25, 7, 8]. Although the quotation is 17 years old, it still seems as valid today as when Hill first stated it. However, the need for a rigorous treatment of scalability arguably has become even more critical because computing technology seems to be approaching fundamental physical limits [31]. Consequently, scalability of software systems will become an increasingly important concern, as developers no longer will be able to rely on exponential improvements in computer technologies to rescue them from poor design decisions.

The first obstacle to addressing this concern is the lack of a consistent and systematic treatment of scalability, which makes it difficult to evaluate claims of scalability, to compare claims from different sources, and to approach the problem of scalability analysis in a uniform manner across different system designs and application domains. Indeed, there is little consensus as to what the term actually means, giving rise to a range of competing, and at times inconsistent views of what it means for a system to be scalable. Additionally scalability is, in general, a highly complex attribute of systems requiring sophisticated analysis techniques.

This paper presents a general framework for characterizing and analyzing the scalability of a software system. The framework attempts to resolve the intuitions, ambiguities and inconsistencies underlying the use of the term in computing. It precisely captures the dependency relationships that underlie typical notions of scalability and describes necessary steps for scalability analysis. The framework is designed to be instantiated for particular scalability properties of interest and can accommodate both the analysis of measurements of implemented systems and predictions of scalability from formal system models. The analysis approach presented in this paper treats scalability as a multi-criteria optimization problem and uses precepts of microeconomics to support the analysis.

The paper is organized as follows: Section 2 discusses the different contexts and understandings of the term scalability found in the literature. Section 3 presents our framework and an analysis method based on analytical tools from multi-criteria optimization. Section 4 presents results from a case study in which we applied the framework and analysis method to a real-world system, demonstrating that it is possible to develop a precise, systematic characterization of scalability and to use the characterization to compare the scalability of alternative system designs. Section 5 reviews the current state-of-the-art of scalability research. Section 6 concludes the paper and discusses future work.

2. BACKGROUND

Scalability has been discussed in many domains, from parallel computing [23] to software processes [24] to requirements prioritization [2] . A fairly typical illustration of the way the term is used in computing literature is provided by the specification of SAP, a well known protocol studied in networking [14]. The roughly 5500-word document contains exactly three occurrences of the word "scalability". The first occurrence is in the abstract of the document:

> *This document describes version 2 of the multicast session directory announcement protocol, Session Announcement Protocol (SAP), and the related issues affecting security and scalability that should be taken into account by implementors.* [14]

The second occurrence is in a paragraph in the middle of the document; here is the complete paragraph:

> *A SAP announcer periodically multicasts an announcement packet to a well known multicast address and port. The announcement is multicast with the same scope as the session it is announcing, ensuring that the recipients of the announcement are within the scope of the session the announcement describes (bandwidth and other such constraints permitting). This is also important for the scalability of the protocol, as it keeps local session announcements local.* [14]

The third occurrence is within a heading entitled "Scalability and Caching" for a section containing a free-form discussion of desiderata that neither uses the term nor precisely defines its intended meaning.

The authors of this document are extremely talented network protocol designers, and thus their claims about scalability arguably can be taken on trust. But what exactly are their claims? What is it whose scalability is being claimed? Is it the whole protocol? The caching strategy of the protocol? What system property should be measured to evaluate its scalability? Is it message delay? Message throughput? What is it whose scaling is considered important? Is it something in the design of the protocol? Something in the execution environment of the protocol?

Like the above example, numerous are the cases in the computing literature in which scalability is mentioned in passing in this fashion or simply claimed outright but never fully defined or justified. To overcome this problem, a few authors have attempted to define rules of thumb with varying degrees of rigor:

> *Scalability means not just the ability to operate, but to operate efficiently and with adequate quality of service, over the given range of configurations.* [17]

Terms such as "efficient"and "adequate" are too subjective to be of use. Somewhat better is the following:

> *An architecture is scalable ... if it has a ... linear (or sub-linear) increase in physical resource usage as capacity increases ...* [8]

Looking for an increase at most linear in resource usage as demands on the system increase may seem like a reasonable rule of thumb at first blush, but it is easy to think of exceptions to this rule. For instance quicksort, a backbone algorithm of many information processing systems, has super-linear time complexity of $O(n \log n)$ in the average case.

Contributing further to the challenges of terminology and uniformity, scalability concerns have been mentioned in the context of specific software qualities:

> *"Scalability is the ability of a system to continue to meet its response time or throughput objectives as the demand for the software functions increases."* [30]
>
> *"However, a RAID5 system still has problems with its scalability ... the MTTF of RAID5 is inversely proportional to the square of the number of HDDs."* [38]
>
> *"The limited scalability of existing multicast simulation methods is primarily due to the large amount of state maintained by the simulators, which is often on a high order of the input size... This state requires a proportional amount of memory in the simulator."* [37]
>
> *"The table above gives cost-equivalent key sizes ... The time to break is computed assuming that Wiener's machine can break a 56-bit DES key in 100 seconds, then scaling accordingly. The 'Machines' column shows how many NFS sieve machines can be purchased for $10 million..."* [29]

The use of the term scalability in the context of performance, as in the first quote, is the most commonly found context. However, performance is just one of many possible indicators of scalability. The second quote, for example, uses mean-time-to-failure to estimate the system availability as the number of hard disk drives scales. The third one refers to the amount of memory used by a simulation system as a function of its input size. Note that the fourth quotation points to a table that indicates the time required to break encryption keys using a varied number of NFS sieve machines. In this case, the performance metric *time* serves as a proxy for security. Therefore, scalability is also analyzed with respect to other system qualities, such as reliability, availability, resource usage, dependability and security.

What the cited quotations have in common is a sense that a system needs to tolerate variation or *scaling* in some characteristic affecting its execution. For this reason, we advocate that *any system analysis conducted with respect to a variation over a range of environmental or design qualities is a scalability analysis.* Furthermore, it is apparent that there is no single, universal notion of scalability, but rather its characterization is highly domain- and system-specific.

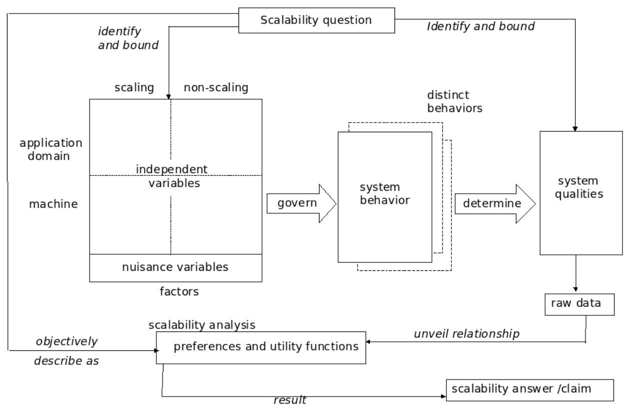

Figure 1: The scalability Framework.

Thus, determining the critical scaling characteristics, the critical execution characteristics that must be measured in order to judge scalability, and what constitutes sufficient "tolerance", are all things that *system stakeholders* ultimately must decide in the context of system requirements, since they will know best what kinds of demands will be placed on a system and which system characteristics are most critical to maintain as demands change.

Even when developers claim to have considered scalability, they have often catered only for the boundary cases of environmental and design characteristics. However, boundary cases will often change. Furthermore, the same system may be deployed at different clients, each one with its own limits. Finally, boundary cases are no basis for understanding the system behavior on other points of the scaling range. Analyzing scalability requires, instead, a clear and complete picture of the operational impact that the variation of environmental and design characteristics will have on the system. For this reason we state that scalability analysis should unveil the relationship between the operational characteristics of the system and the full range of its environmental and design characteristics—in an explicit and continuous form. Such understanding is useful for guiding design decisions that are more likely to support evolving requirements and multiple deployments, objectively stating or judging scalability claims and comparing claims from different sources.

Based on our extensive reading of the literature and on our own understanding of scalability, we next define a framework that gives stakeholders a systematic way of describing and analyzing scalability characteristics of software systems.

3. THE SCALABILITY FRAMEWORK

We define scalability as *a quality of software systems characterized by the causal impact that scaling aspects of the system environment and design have on certain measured system qualities as these aspects are varied over expected operational ranges.* If the system can accommodate this variation in a way that is acceptable to the stakeholder, then it is a scalable system. More precisely, we define scalability analysis as a form of experimental design that is used to reveal the causal relationship underlying the notion of scalability in terms of certain *factors* and certain *dependent variables* [28]. Figure 1 depicts the framework we have designed to reveal this relationship.

3.1 Scalability Variables

Factors represent characteristics of the application domain and the machine that will affect the system behavior, such as volume of input data, rate at which work arrives at the system, number of concurrent system users, maximum cache size, maximum thread pool size, number of nodes in a server cluster, algorithm selection and cost. These characteristics can be classified as scaling and non-scaling. The former represent characteristics that can potentially *scale* in a scalability analysis, whether within or between executions of the system. The later are characteristics that are either fixed on the scalability analysis or vary in a nominal scale.

The subset of the factors that can be manipulated for the scalability analysis are called *independent variables.* We refer to the scaling characteristics belonging to the independent variables as *scaling dimensions.* Scaling dimensions represent the scaling aspects of both the application domain and the machine, such as the number of concurrent users or available nodes in a cluster. We refer to the complement set as non-scaling variables, which are characteristics of the application domain or machine that are either set to fixed levels or varied in a nominal scale in an attempt to enable the system to deal with the scaling dimensions. They could be, for example, the selection of a sorting algorithm or the RAM of the machine. *Nuisance variables* are characteristics of the application domain and machine that cannot be manipulated for the scalability experiment, such as the processor speed if the infrastructure is already decided and cannot be changed. A particular set of values selected for the machine design variables is called a *machine configuration.*

The dependent variables represent aspects of the system behavior that are affected by changes in the independent variables. They typically correspond to software metrics related to performance, cost, maintenance, reliability, security and operational constraints. Examples are peak and average values of throughput, storage consumption, operating costs and failures per unit of time.

While the dependent variables typically represent metrics of traditional system characteristics such as performance, resource usage, reliability and so on, it is the analysis of the dependent variables *in the presence of variation to the scaling dimensions* that turns an ordinary quality analysis into a scalability analysis. And thus it is vague to refer simply to "the scalability of a system"; instead one must refer to "the scalability with respect to throughput", or "the scalability with respect to latency and memory consumption".

Once the factors and the dependent variables influencing the system scalability have been identified, developers should define an objective and quantifiable way to measure the dependent variables against the scaling dimensions. In this way, the relationship between the operational characteristics of the system and the full range of it scaling dimensions can be unveiled in an explicitly and continuous form, characterizing the system's scalability. The raw data values for the variables can be empirically collected or estimated through models, and statistical techniques can be used to fit the curves that characterize this relationship. Section 3.2 discusses an analysis that treats scalability as a *multi-criteria optimization problem.*

3.1.1 Selecting the Variables for Analysis

The selection of independent and dependent variables will be unique to the scalability questions and system at hand.

Examples of scalability questions are: (1) For a given machine configuration, how do the dependent variables behave or degrade as the variables representing the scaling dimensions increase? (2) How do the dependent variables behave when one accounts for the probability distribution of the scaling dimensions? (3) Given two or more candidate machine configurations, which one produces the most favorable scalability with respect to the dependent variables?[1] (4) Given two or more candidate machine configurations, what are the "crossing points", that is, the points where one configuration becomes more favorable than another? (5) Given some analyzed behavior of dependent variables over analyzed ranges of scaling dimensions, how are the dependent variables expected to behave if the dimensions are extrapolated beyond the bounds between which they were analyzed?

Once the variables of interest have been identified, the stakeholder must establish bounds defining their expected range of values. The bounds will reflect expected variation and growth in the application domain, technical and economic feasibility in the machine design, and acceptable bounds on required system qualities. Note that the machine may be designed to cope with variation in a scaling dimension by *dynamically* varying some quality of the design, such as the size of a thread pool. However, such dynamic variation can be performed only within static limits embodied in the machine, and it is these static limits that are the bounds for the independent variables representing the machine.

3.1.2 Examples

As demonstrated in this section, our framework is expressive enough to unify different notions of scalability that have appeared in the literature. We argue that by recasting a scalability concern according to the framework, it is easier the express and justify scalability claims[2].

1. Google search engine: The Google architecture is a large cluster of commodity processors over which the load of search queries is distributed [3].

Scaling dimensions:
Number of queries per second, number of machines in the cluster
Non-scaling variables:
Available bandwidth, network round-trip time
Dependent variables:
Response time for a query, bandwidth usage, I/O devices usage

Claim: Google is scalable with respect to *response rate* because it is able to maintain a sub one-second response time as the *number of queries per second* scale, by varying the *number of commodities machines* in the cluster.

2. Distributed systems: Jogalekar and Woodside address the question of a scalability measure for distributed heterogeneous service systems [18]. One of the scalability concerns discussed in the paper investigates whether the extra investment in maintaining and administering multiple copies of the connection manager or database can be justified against the increase of the "power" metric (ratio between throughput and response time).

[1]We will shortly describe ways of systematically determining when an analyzed behavior is "favorable".

[2]We only exemplify the chosen set of variables and state a simplified scalability claim, as the information we used is limited to what has been published.

Scaling dimensions:
Number of active clients, Number of replicated databases, number of connection managers, CPU speed
Dependent variables:
Power, cost in dollars

Claim: The system is scalable from one configuration to another if the *power per invested dollar* can be improved (or maintained) by evolving the system configuration (*number of databases* and *connection managers*).

3. Parallel algorithms: Kumar and Rao have developed a scalability metric which relates the problem size to the number of processors required for an increase in speedup in proportion to the number of processors used [27].

Scaling dimensions:
Problem size, Number of processor units
Non-scaling variables:
Work distribution scheme, Presence/absence of shared memory, Diameter of the network, Relative speed of the communication network
Dependent variables:
Efficiency, speedup

Claim: A parallel algorithm can be claimed scalable if *efficiency* can be maintained between 0 and 1 for an increasing *number of processor units*, provided that the *problem size* is also increased.

4. Commercial software: Masticola and Bondi discuss a large-scale commercial server-based software product [26]. By migrating the system from *C#* to *Java*, developers hoped to increase the number of users supported by the system.

Scaling dimensions:
Number of users, number of collaborating servers
Non-scaling variables:
Server design (C# vs. Java)
Dependent variables:
Processors usage, bandwidth usage, I/O devices usage

Claim: The design is scalable from C# to Java if it can increase (1) the *number of users* supported on each "standard server" by an one order of magnitude and (2) the *total number of users* through use of collaborating servers, while maintaining active *resource usage.*

3.2 Scalability Analysis

In developing our scalability framework and working with the case study system described in Section 4, it quickly became apparent that scalability can be treated quite naturally as a *multi-criteria optimization problem* in which several, possibly conflicting quantitative criteria are to be optimized simultaneously. In our case, the dependent variables quantify the criteria to be optimized, and the tradeoffs among the dependent variables result in different choices of machine configuration. The stakeholder's scalability goals in respect to different measurable characteristics of the system are quantified by *preference* functions over the value of each dependent variable. The developer can then use these preference values to measure the "satisfaction level" of the stakeholder with respect to individual quality goals. As a measure of the overall satisfaction with the system, we use a *utility* function that combines the values of the different dependent variables. By plotting preferences and utility values against the scaling dimensions, it is possible to understand trends between the aspects of the system environment/design and the operational characteristics of the system, which can be used to characterize the system's scalability.

3.2.1 *Preferences and Pareto Optimality*

In many multi-criteria optimization problems, an solution that maximizes all objectives may not exist. For instance, it may be possible to improve the throughput at the cost of memory consumption due to increased buffer size, or to reduce memory consumption at the cost of decreased throughput. This kind of situation is called *Pareto optimal* [33].

In a Pareto optimal situation it is not possible to improve any one objective without compromising another. More formally, let $X = \{x_1, \ldots, x_k\}$ be a set of possible *outcomes*, each representing a set of values for i competing *objectives* (to which the dependent variables of our framework correspond). Let $f_j : X \to \mathbb{R}$ be functions quantifying the *preference* for objective j in an outcome in X, with $1 \leq j \leq k$. The preference functions must be defined in such a way that the values representing the outcomes for a particular objective are mapped to a range of values in which a greater value equates with greater preference. For instance, the identity function can be used as a preference function for a quality such as average throughput, since higher throughput is typically preferred to lower throughput. However, for a quality such as memory consumption, lower consumption is preferred to higher consumption, and so the preference could be computed as the reciprocal of the value of memory consumption. In general, any arbitrarily complex function mapping outcomes to real numbers can be used to capture one's full understanding of the nature of the preference for a particular objective in the set of outcomes.Then, X is Pareto optimal if and only if there exists no y in X satisfying the following two conditions: (1) for all $x_m \neq y$, $f_{j_0}(y) > f_{j_0}(x_m)$ for some objective j_0, and (2) for all $x_m \neq y$, $f_j(y) \geq f_j(x_m)$ for all objectives j other than j_0. In other words, there is no one outcome y that maximizes all preferences.

Typically the Pareto optimum will be a *frontier* of points whose shape indicates the trade-off between objectives. In our framework, each point represents a single system behavior produced by the execution of a single machine configuration, with all executions using the same values for the environment variables. Dependent variables define the axes of the plot, with each behavior represented as a point according to the values it produces for the dependent variables.

To illustrate these ideas, consider a hypothetical example in which we are measuring the dependent variables *average response time* and *average disk usage* against the scaling dimension *average number of simultaneous users*. Figure 2.a illustrates hypothetical alternative behavioral outcomes for these characteristics produced by five hypothetical machine configurations for a given value of the scaling dimension. Given that it is reasonable to try to *minimize* the value of both characteristics, we define the following preference functions over the set of outcomes: *average available disk* for disk usage and *average throughput* for average response time. The outcomes produced by Configurations A, B and C are Pareto optimal and thus define the Pareto frontier depicted in figure 2.a. In particular, none of these three configurations is universally preferable to the other two, since none of them maximizes both preferences. However, Configurations D and E are inferior in both preferences to A, B and C and thus can be rejected from further consideration.

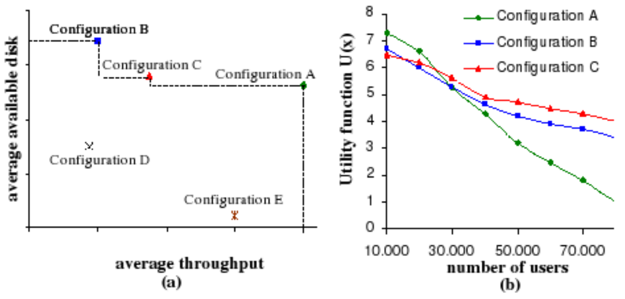

Figure 2: Comparing Configurations.

3.2.2 *Utility Functions*

The choice between alternative outcomes in a Pareto frontier is not straightforward. To resolve the tradeoffs inherent in a Pareto optimal situation, we may use a *utility function* to transform a vector of preference values into a single scalar value. A commonly used model of utility is *objective weighting*, in which the utility function $U(X)$ is a linear combination of the preference values $f_j(X)$, each preference value weighted by a corresponding weighting factor λ_j [11]. In particular, $U(X) = \lambda_1 f_1(X) + \ldots + \lambda_k f_k(X)$. The weights represent the extent to which an increase of one quality will be accepted as the others decrease. The choice of weights must reflect the goals of the stakeholders, who must decide how they want to prioritize particular system qualities. Each combination of weights defines a different utility function, and the chosen weights reflect a set of goals defined by the stakeholder. Systematic selection of appropriate functions based on requirements goals is subject of future research.

When assigning weights to disparate system qualities such as disk consumption and throughput, there is a danger of producing an ill-defined utility function due to wide variation in the range of possible values for each preference. Therefore, the preference values $f_j(X)$ should first be *normalized* to a common range before being used in a utility function. For instance, we may define a function $\hat{f}_j(X)$ for each objective j that normalizes the preference values $f_j(X)$ to a value between 0 and 1:

$$\hat{f}_j(X) = \frac{f_j(X) - f_{j,min}}{f_{j,max} - f_{j,min}} \qquad (1)$$

where $f_{j,min}$ and $f_{j,max}$ are, respectively, the lowest and highest possible preference values for objective j. If the lowest and highest possible values cannot be established in a meaningful way, then the lowest and highest values observed in all outcomes in X can be used.

Finally, the comparison between configurations is dealt with by plotting the utility of their outcomes against each analyzed value of the scaling dimensions. Figure 2.b plots utility curves for the three hypothetical Pareto-optimal configurations of Figure 2.a against the average number of simultaneous users. The configuration that maximizes utility for the expected range of the scaling dimensions is the one that should be selected. In this example, Configuration C quickly achieves maximum utility at around 30,000 users.

3.3 Steps for Scalability Analysis

A scalability analysis may be used with different goals, such as developing a scalable system, objectively judging scalability claims or comparing claims from different sources. Here we present a summary of the steps we propose for the analysis in the context of software development: (1) Define the system's scalability goals in terms of the scaling aspects of the system environment/design and its desired system qualities. (2) Select controllable and quantifiable variables to represent the characteristics of the system environment/design and the system qualities that are likely to impact the system's scalability. (3) Identify domain, technological and economical ranges for the scaling dimensions and roughly estimate their effect on the system qualities of interest. (4) Define preferences and utility functions to measure the dependent variables against the scaling dimensions, design variables and constraints. (5) Clearly express the objective of the scalability analysis in terms of the chose variables and functions so that its results can be unambiguously communicated to stakeholders. (6) Design and implement the system taking into consideration the scalability goals and progressively collecting metrics to estimate the preference and utility values for the full range of scaling dimensions. (7) Draw clear and precise conclusions about the scalability of the system with respect to the stakeholder's goals.

4. CASE STUDY

Intelligent Enterprise Framework (IEF) is a system that handles large volumes of data, builds adaptive profiles of business entities within the data, and generates alerts to notify users of behavior that appears unusual. Its data analysis comprises the following stages: validation, preprocessing, loading, migration, profiling, eventing and alerting. In the preprocessing stage, IEF takes a batch of transactions as input and replaces the various original business entity identifiers with simplified *surrogate keys*. Normally, a transaction contains a heterogeneous mixture of identifiers for business entities, such as account, branch, customer, transaction code and transaction type. The surrogate key assignment is performed by a critical subsystem, the *Surrogate Key Server* that allocates a uniform system of internal identifiers to the heterogeneous mixture of business entity identifiers.

In the company's early implementation of the SK Server (year 2000), the creation/lookup of surrogate keys was recognized as a major barrier to scalability. The implementation consisted of an *in-memory cache* of previously mapped entity-ID/surrogate-key pairs, which incurred a very high storage overhead, increasing both memory footprint and garbage collection activity. As the number of distinct business entities grew during the lifetime of the system, available memory eventually was exhausted, bringing the system to a halt. The stakeholder's key scalability concern is to support an ever-growing number of distinct business entities in the overnight batches of transactions, while maintaining throughput and the system's memory footprint within the bounds defined in the system requirements. The problem was corrected in a new design that uses a separate *file-based* surrogate key lookup for the assignment of keys on large sets of distinct business entities, such as accounts.

The main multi-criteria trade-off was in terms of memory usage and throughput. The file-base design reduced memory footprint by using a lookup file in disk. However, because this file was serially searched, the time to replace the keys grew with the number of entities, affecting throughput. The memory-based design, on the other hand, improved throughput by using a HashMap, at the cost of memory consuption.

The case study consisted of running execution experiments to analyze and compare the scalability of the two designs, from the validation to the migration stages. The designs were executed on a test machine containing twin Intel(R) Xeon(TM) processors, 2.40 GHz CPUs, 2GB of memory and 4x60GB disk with RAID. Harness tests were run until the machine reached its physical limits or until the system could no longer comply with its requirements. The memory-based version of the system, which used a memory cache only implemented the cache as a Java Hashtable object allocated in the Java Virtual Machine (JVM) heap. At the time, the preprocessor stage was expected to run on a JVM with maximum available memory of 500MB, which imposed a hard upper limit on the number of entity-key pairs that could be cached. The file-based design restricts the amount of heap used to cache entity-key pairs, by using the memory lookup only for small sets of distinct entities, such as branch and transaction type. Switching from one design to another was a matter of setting a parameter in the system configuration. The system is implemented in Java and has 1,556 classes and 326,293 lines of code.

In both designs, the size of the maximum JVM heap was set to 500MB to mimic the environment at the time the first version was on operation. The number of threads handling the data migration to the operational data store was configurable. Thus, we studied a total of six machine configurations in the case study, by varying the number of migration threads (one, three and five) and alternating the lookup design (file-based and memory-based). Due to space constraints, we discuss only two configurations in the remainder of this section, one for the memory-based design and the other for the file-based design, both with five threads.

4.1 Framework Instantiation

After conferring with the stakeholder, we determined that average throughput, memory usage and disk usage should be the system qualities to be measured in the case study as dependent variables. Throughput and memory usage were primary concerns, as the system must be able to complete its analysis overnight and without running out of memory. Disk usage was also chosen as a dependent variable of interest because in adopting a file-base lookup, the usage of space holding the keys is shifted from memory to disk, and the stakeholder needs to analyze if this shift would impose a scalability problem. Throughput was measured as the average number of transactions per second, memory usage as the footprint of the JVM heap size and disk usage as the percentage of available disk space used. Network usage is a less important property that can also affect the system throughput. However, because network usage not directly related with the algorithm for creating and looking up surrogate keys, it was assumed to be a nuisance variable.

We therefore instantiated our framework as follows:

Scaling dimensions:
Number of distinct business entities, number of concurrent threads
Non-scaling variables:
Memory- vs file-based design, JVM memory size
Nuisance variables:
Network bandwith
Dependent variables:
Throughput, memory usage, disk usage

The stakeholder has declared that for a typical bank, the system should be able to process batches of 30 million transactions and deal in total with over 50 million business entities: 30 million accounts, 20 million customers, 1 thousand branches, 115 transaction codes and 130 transaction types.

The distribution of business entities was estimated by using a sample of 15-months' worth of data. The estimated number of business entities involved in transactions on a given batch is in average 36.66 million million with standard deviation of 5 million. We therefore set the desired average and acceptable bounds for the system qualities of interest and the likely distribution of business entities for the data analysis system as following:

- *Number of distinct business entities*: 0 - 50 million, with an average number per batch of 36.66 million and a standard deviation of roughly 5 million entities.
- *Number of concurrent threads*: 1 - 5 lookup threads
- *Average throughput*: 100 transactions/second [3]
- *Memory usage*: 0 - 500MB
- *Disk usage*:0 - 24GB

4.2 Measurement of System Qualities

In order to measure the system qualities of the two configurations, we ran jobs on the input data that produce daily summaries of the transactions. The experiment consisted of running each configuration against an increasing number of distinct entities. For such, we generated 5 sets of 5 million transactions containing, respectively, the following rough numbers of distinct entities: 1.5 thousand, 6 thousand, 50 thousand, 500 thousand and 5 million. Each experiment was run two to three times, individual measurements were recorded periodically, and the average number was used. Although the experiment cannot prove that the new design scales to the required production data load, it is sufficient to demonstrate the different scalability characteristics of the two designs.

We measured the machine configurations behavior for the datasets described above and plotted the measurements in a number of data plots. Figure 3.a shows plots of JVM heap size against the number of distinct entities for the memory-based and the disk-based configurations. In the memory-based configuration, heap usage reaches 500MB, which was the maximum memory available to the JVM at the time. Therefore, the JVM runs out of memory at just a little over 4 million distinct entities in memory, bringing the system to a halt. In the disk-based configuration, heap usage is bound to around 20MB, as just the business entities with few distinct values are held in memory (roughly 1200 entities). Figure 3.b plots throughput against the number of entities. In the memory-based configuration, as the JVM runs out of memory, the throughput goes to zero because of virtual memory swapping. In the disk-based configuration, throughput decreases slowly, as the number of entities to be searched for a lookup increases. Disk usage, although a stakeholder concern, was 1%–2% of the total available space in both configurations, proving not to be a problem. Due to space constraints, we omit the disk usage plots.

[3]Harness tests were run in a machine that was substantially less powerful then the one normally used in the production environment. Based on the experience of the testing group at Fortent, we aimed for an average of at least 100 transactions/second for the testing machine.

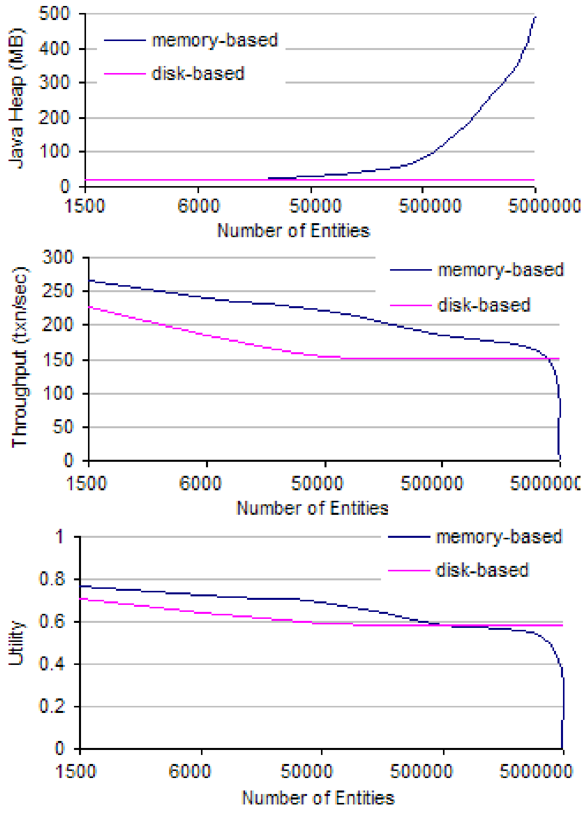

Figure 3: Memory vs. File Design

4.3 Scalability Analysis

For the case study, the stakeholder was interested only in identifying the preferred machine configuration. As described in Section 3.2, we defined preference functions for each measured quality, in particular throughput (unchanged from its original value), available heap (total memory minus memory usage) and available disk (total disk minus disk usage), so that a higher value represents a more preferred outcome; note that the bounds established by the requirements for the system qualities serve as the bounds for these corresponding preference functions. As also described in Section 3.2, we then computed normalized values of the measured data according to Equation 1 with respect to the lower and upper bounds of each quality. We enhanced the normalization so that when a preference value was lower than the required lower bound, then a penalty value of -1 was used instead of the computed normalized value.

More formally, the normalized preference of a throughput value x is defined by $\hat{t}(x)$[4]:

$$\hat{t}(x) = \begin{cases} -1, & \text{if } x < 100, \\ \frac{x-100}{400-100}, & \text{otherwise.} \end{cases} \quad (2)$$

The normalized preference of a heap usage value y is defined by $\hat{h}(y)$:

$$\hat{h}(y) = \begin{cases} -1, & \text{if } y > 500, \\ \frac{500-y}{500-0}, & \text{otherwise.} \end{cases} \quad (3)$$

[4]A upper-limit of 400 transactions/second (close to the maximum observed throughput) was used for mathematical convenience.

And the normalized preference of a disk usage value z is defined by $\hat{d}(z)$:

$$\hat{d}(z) = \begin{cases} -1, & \text{if } z > 24, \\ \frac{24-z}{24-0}, & \text{otherwise.} \end{cases} \quad (4)$$

Given these normalized preference functions, we then defined a utility function reflecting the stakeholder's goals for optimizing the measured system qualities. In particular, the stakeholder declared that it is 10 times as important to optimize throughput and to optimize memory usage as it is to optimize disk usage, since disk storage is becoming increasingly cheaper. This results in the following preference function $U(x, y, z)$:

$$U(x, y, z) = \frac{10\hat{t}(x) + 10\hat{h}(y) + \hat{d}(z)}{21} \quad (5)$$

Note that we have chosen to normalize the weights so that they sum to one.

Figure 3.c shows a plot of Formula 5 against the measured range of distinct entities for the memory-based configuration and the disk-based configuration, with no extrapolation beyond the measured range. The plot shows that the utility of the memory-based lookup design exceeds that of the file-based one for most of the measured range, but drops to zero after a critical point of approximately 4 million distinct business entities. The file-based lookup implementation has its utility dropping only slightly and very smoothly from roughly 0.7 to 0.6 as the number of distinct entities increase. No extrapolation was necessary to show the superiority of the file-base lookup configuration over the other one. As in average a batch has to handle in average of 36.66 million with standard deviation of 5 million business entities in the production deployment and the JVM memory, at the time, was limited to 500MB, a utility of zero disqualifies the memory-based lookup solution from consideration.

In hindsight, the file-based design may appear to be obviously superior to the memory-based design, but this was not at all obvious when the memory-based design was first developed. In fact, if the designs had been compared only in terms of the load at the time the memory-based system was first being developed, then the memory-based design would have been selected instead of the file-based design. Only by doing a proper analysis over the full range of the scaling dimensions are we able to select the most scalable design.

Being a retrospective study, the analysis was eased by the fact that the scalability requirements were well-known and both designs were implemented, so that reliable metrics could be collected and analyzed. We also knew that the file-based lookup solution could handle the batches with 30 million transactions and 50 million distinct entities required at the time. However, as the number of entities has been increasing beyond the expected range in the past years, the disk design has now reached its limits in terms of scalability and is being currently re-implemented with the support of our scalability framework.

Although the preference and utility functions provided by the stakeholder were sufficient to show the superiority of the disk-based design, we recognize that other analyses may require more complex preference and utility functions. Finally, for its retrospective nature, we could not assess the overhead of such analysis in the development lifecycle. These issues currently are being tackled by another case study.

5. RELATED WORK

Like ourselves, other authors have questioned the meaning and use of the term scalability and have attempted to provide better definitions [15, 32, 7, 34]. One particular area in which scalability enjoys precise characterization is parallel computing, where scalability metrics such as *fixed-size speedup* and *fixed-time speedup* have been established and accepted [17]. However, because of fundamental differences between parallel computing and other classes of systems, such metrics cannot be applied broadly. Another area where scalability receives a more systematic treatment is model checking, where the state space size can be taken as an indicator of the scalability of the techniques used to tackle the so-called state explosion problem [12].

Measurement and prediction of certain qualities represented in our framework as dependent variables have been studied for many years, particularly in the area of performance evaluation, where well established models such as queuing networks, Petri nets and stochastic process algebras are used to estimate and compare the performance of one or more machine configurations. Many of these solutions propose extensions to standards and notations such as MDA, UML, architecture description languages (ADLs), OWL-S and WSOL to incorporate annotations for performance prediction [35]. Others recognize that system models evolve, and some of them extend the Software Performance Engineering (SPE) process to deal with this problem [6]. Finally, there are works that use monitoring to model and predict performance [9]. These approaches can be specialized for different kinds of systems or technologies, such as component-based systems and middleware, by exploiting knowledge about the technology to aid performance prediction [36]. All of these approaches can be applied with our framework to produce the raw measurements of behavioral qualities needed for subsequent scalability analysis.

Outside the realm of performance analysis, the most notable related works are found in the field of architecture evaluation. A number of methods have been developed to evaluate system qualities at the architecture level, such as: Scenario-based Architecture Analysis Method (SAAM) [20], Architecture Tradeoff Analysis Method (ATAM) [22], Cost Benefit Analysis Method (CBAM) [21], Active Reviews for Intermediate Designs (ARID) [10] and Scenario-Based Architecture Reengineering (SBAR) [4]. These methods generally identify the quality goals of interest and then highlight the strengths and weaknesses of the architecture to meet the desired goals. Depending on the method, the evaluation explicitly addresses a single quality or multiple quality goals. The latter are the ones that might be perceived as overlapping with our work. The main difference lies in the fact that none of them treats the *scaling* of system characteristics affecting the qualities of interest in an explicit form. ATAM and CBAM, for example, propose the use of growth and exploratory scenarios, but neither provide guidance on how to devise these scenarios nor treat them differently from use case scenarios in the analysis process. Growth and exploratory scenarios represent snapshots of the system at some time in the future, catering possibly for analysis of boundary cases. Nevertheless, we claim that to truly understand the scalability of a system one needs to unveil the relationship between the operational characteristics of the system and the full range of its environmental and design characteristics, in an explicit and continuous form. In any case, these and other methods could be adapted or used to analyze scalability according to our framework. Also related are works that use utility theory for allocating system resources. Bennani and Menasce, for example, propose a solution that dynamically redeploys the servers of a data center in order to maximize some global utility function [5]. These, however, normally are targeted at performance metrics and do not provide an explicit treatment of scalability.

Specifically on scalability, previous papers attempted to adapt scalability notions for parallel computing to other classes of systems, particularly distributed systems. Jogalekar and Woodside defined an effectiveness metric computed as a function of throughput and quality of service [17]. In other work, Jogalekar and Woodside concentrated on finding the most economical path for evolving a system by relating a notion of the *power* of the system to the cost of obtaining this power [19]. Both papers are limited to the particular domain of distributed systems and limited to analyzing scalability with respect to performance. More recent work has targeted narrow classes of systems and technologies, or are limited to performance as the scalability metric of interest. For instance, Masticola et al. used early life-cycle models to analyze the scalability of component-based systems in terms of performance metrics [26]. Steen et al. described a systematic approach to guide the distribution and application of scaling techniques, such as replication and caching [32]. Weyuker and Avritzer create a *Performance Non-Scalability Likelihood* (PNL) metric in order to assess the expected loss in performance for a given workload. In contrast, our framework attempts to establish a uniform notion of scalability that can be applied in a wide variety of application domains and can support analysis of scalability with respect to a wide variety of system qualities. A nice work on scalability was done by Weinstock and Goodenough [34]. They also discuss the lack of a uniform definition of scalability and propose an audit to help to expose issues that, if overlooked, can lead to scalability problems. Such audit could be used in the context of our framework to highlight possible issues that will lead to the appropriate selection of variables.

6. CONCLUSION AND FUTURE WORK

Scalability is a frequently claimed attribute of software systems. This paper has discussed the imprecise use of the term in computing, the lack of consensus on the meaning of the term, and the potential problems that ensue as a result. In an attempt to improve the precision and utility of uses of the term, we have defined a framework for characterizing and analyzing the scalability of software systems. The framework can be supported with instantiation methods and analysis techniques and tools to deal with the particular characteristics of different problem domains. For instance, on another large case study, we are currently using goal-oriented requirements engineering to elicit the scalability requirements, select the variables and derive the preference and utility functions, while the raw data are being collected by using instrumentation and profiling tools. The paper presented a method for comparative scalability analysis based on microeconomics theory and illustrated it with a case study drawn from the financial services industry.

The analysis approach described has limitations that we intend to address in future work. For instance, we currently do not deal with variables that are not numeric or linearly ordered, with co-dependent variables, or with intransitive preferences. There exist sound mathematical techniques that can be used in such situations, such as preference logics [1]. Additionally, more research effort should go into establishing methods for deriving preference and utility functions that are meaningful and suitable to express the causal relationships between factors and dependent variables in a scaling system. In, Oslo and Rodgers have done some preliminary work in this direction, using the WinWin negotiation model to derive preference functions [16].

There are several additional future directions that can be explored, particularly in the field of requirements engineering. Scalability should be dealt with as early as the requirements phase. However, to the best of our knowledge, there is no definitive work on requirements and scalability. We are currently applying and investigating the suitability of goal-oriented requirements engineering as a means to identify scalability goals and break them down into lower-level requirements that can be used to identify variables for the analysis and define preferences and utility functions. Additionally, it is frequently the case that system measurement must be performed in restricted environments that do not adequately present the full range of expected characteristics and workload of a production environment. In such cases it would be desirable to extrapolate the results of scalability analysis to the expected production range of scaling dimensions. However, extrapolation must be done with extreme care and must be informed by knowledge of the underlying computational phenomena of the machine (such as the computational complexity of the key algorithms and data structures that affect the measured system qualities). We also intend to explore scalability of software tools and techniques, which is a particularly important and frequently raised concern in software engineering research and practice, but one that begs for more systematic treatment.

7. ACKNOWLEDGMENTS

This work was partially supported by the European IST FET programme in project SENSORIA (IST-2005-016004). Leticia Duboc is funded under a studentship from UCL. David Rosenblum holds a Wolfson Research Merit Award from the Royal Society. The authors gratefully thank Rami Bahsoon, Antonio Carzaniga, Sebastian Elbaum, Wolfgang Emmerich, José Faro, Anthony Finkelstein, Gorka Guardiola, Emmanuel Leitier, Damon Wischik and Alex Wolf for discussions of this work, and the anonymous referees for their useful comments.

8. REFERENCES

[1] H. Andréka, M. Ryan, and P.-Y. Schobbens. Operators and laws for combining preference relations. *Journal of Logic and Computation*, 12(1):13–53, 2002.

[2] P. Avesani, C. Bazzanella, A. Perini, and A. Susi. Facing scalability issues in requirements prioritization with machine learning techniques. In *Proc. RE '05*, pages 297–306, Washington, DC, USA, 2005. IEEE Computer Society.

[3] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, March–April 2003.

[4] P. Bengtsson and J. Bosch. Scenario-based software architecture reengineering. In *Proc. ICSR '98*, page

308, Washington, DC, USA, 1998. IEEE Computer Society.

[5] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Proc. ICAC '05*, pages 229–240, Washington, DC, USA, 2005. IEEE Computer Society.

[6] A. Bertolino and R. Mirandola. Software performance engineering of component-based systems. In *Proc.* 4th *WOSP*, pages 238–242, New York, USA, 2004. ACM Press.

[7] A. B. Bondi. Characteristics of scalability and their impact on performance. In *Proc.* 2nd *WOSP*, pages 195–203. ACM Press, 2000.

[8] G. Brataas and P. Hughes. Exploring architectural scalability. In *Proc.* 4th *WOSP*, pages 125–129. ACM Press, 2004.

[9] M. Caporuscio, A. D. Marco, and P. Inverardi. Run-time performance management of the Siena publish/subscribe middleware. In *Proc.* 5th *WOSP*, pages 65–74, New York, USA, 2005. ACM Press.

[10] P. C. Clements. Active reviews for intermediate designs, 2000. SEI, Carnegie Mellon University CMU/SEI-2000-TN-009.

[11] H. Eschenauer, J. Koski, and A. Osyczka. *Multicriteria Design Optimization: Procedures and Applications*. Springer-Verlag, 1990.

[12] E. M. C. O. Grumberg and D. A. Peled. *Model Checking*. The MIT PRess, 1999.

[13] D. B. Gustavson. The many dimensions of scalability. In *Proc.* 39th *IEEE Computer Society Int'l Computer Conference*, pages 60–63, February 1994.

[14] M. Handley, C. Perkins, and E. Whelan. Session announcement protocol. RFC 2974, Network Working Group, Internet Engineering Task Force, Oct. 2000. Accessed from http://www.faqs.org/ftp/rfc/pdf/rfc2974.txt.pdf on 3 April 2006.

[15] M. D. Hill. What is scalability? *ACM SIGARCH Computer Architecture News*, 18(4):18–21, 1990.

[16] H. P. In, D. Olson, and T. Rodgers. Multi-criteria preference analysis for systematic requirements negotiation. In *Proc. COMPSAC '02*, pages 887–892, Washington, DC, USA, 2002. IEEE Computer Society.

[17] P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589–603, 2000.

[18] P. P. Jogalekar and C. M. Woodside. A scalability metric for distributed computing applications in telecommunications. In *Proc.* 15th *Int'l Teletraffic Congress (ITC-15)*, volume 2a, pages 101–110, 1997.

[19] P. P. Jogalekar and C. M. Woodside. A scalability metric for distributed computing applications in telecommunications. In *Proc. Fifteenth Int'l Teletraffic Congress (ITC-15)*, volume 2a, pages 101–110, 1997.

[20] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Softw.*, 13(6):47–55, 1996.

[21] R. Kazman, J. Asundi, and M. Klein. Quantifying the costs and benefits of architectural decisions. In *Proc. ICSE '01*, pages 297–306, Washington, DC, USA, 2001. IEEE Computer Society.

[22] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The architecture tradeoff analysis method. In *Proc. ICECCS '98*, pages 68–78, 1998.

[23] V. Kumar and A. Gupta. Analysis of scalability of parallel algorithms and architectures: a survey. In *Proc.* 5th *Int'l Conference on Supercomputing*, pages 396–405, New York, USA, 1991. ACM Press.

[24] M. Laitinen, M. E. Fayad, and R. P. Ward. Thinking objectively: The problem with scalability. *Communications of the ACM*, 43(9):105–107, 2000.

[25] E. A. Luke. Defining and measuring scalability. In *Proc. Scalable Parallel Libraries Conference*, pages 183–186. IEEE Press, October 1993.

[26] S. Masticola, A. B. Bondi, and M. Hettish. Model-based scalability estimation in inception-phase software architecture. In *Proc. ACM/IEEE* 8th *MoDELS/UML*, pages 355–366, 2005.

[27] V. N. Rao and V. Kumar. Parallel depth first search. part i. implementation. *Int. J. Parallel Program.*, 16(6):479–499, 1987.

[28] J. R. Ruthruff, S. Elbaum, and G. Rothermel. Experimental program analysis: a new program analysis paradigm. In *Proc. ISSTA'06*, pages 49–60, New York, USA, 2006. ACM Press.

[29] R. Silverman. A cost-based security analysis of symmetric and asymmetric key lengths, 2000. RSA Laboratories Bulletin 13.

[30] C. U. Smith and L. G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley Publishing Company, September 2001.

[31] The 2020 Science Group. Towards 2020 Science. technical report, Microsoft Corporation, Mar. 2006. Accessed from http://research.microsoft.com/towards2020science/ on 1 July 2006.

[32] M. van Steen, S. van der Zijden, and H. J. Sips. Software engineering for scalable distributed applications. In *Proc. 22nd Int'l Computer Software and Applications Conference*, pages 285–293, 1998.

[33] H. R. Varian. *Intermediate Microeconomics: A Modern Approach*. W. W. Norton, 6th edition, 2003.

[34] C. B. Weinstock and J. B. Goodenough. On system scalability, 2006. SEI, Carnegie Mellon University CMU/SEI-2006-TN-012.

[35] M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, and J. Merseguer. Performance by unified model analysis (PUMA). In *Proc.* 5th *WOSP*, pages 1–12. ACM Press, 2005.

[36] X. Wu and M. Woodside. Performance modeling from software components. In *Proc.* 4th *WOSP*, pages 290–301, New York, USA, 2004. ACM Press.

[37] D. Xu, G. F. Riley, M. H. Ammar, and R. Fujimoto. Enabling large-scale multicast simulation by reducing memory requirements. In *Proc. PADS '03*, page 69, Washington, DC, USA, 2003. IEEE Computer Society.

[38] H. Yokota. Performance and reliability of secondary storage systems. In *Proc. of 4th WMSCI*, pages 668–673, 2000.