Boston University
College of Arts and Sciences

Computer Science Department
111 Cummington Street, MCS 138
Boston, Massachusetts 02215
Tel: 617-353-8919 ; Fax: 617-353-6457

Reliability, Availability, Dependability and Performability: A User-centered View

Abdelsalam Heddaya*
heddaya@cs.bu.edu
Computer Science Dept.
Boston University

Abdelsalam Helal
helal@mcc.com
MCC

BU-CS-97-011†
December 4, 1996

Abstract

Reliability and availability have long been considered twin system properties that could be enhanced by distribution. Paradoxically, the traditional definitions of these properties do not recognize the positive impact of *recovery*—as distinct from simple repair and restart—on reliability, nor the negative effect of recovery, and of internetworking of clients and servers, on availability. As a result of employing the standard definitions, reliability would tend to be underestimated, and availability overestimated.

We offer revised definitions of these two critical metrics, which we call *service reliability* and *service availability*, that improve the match between their formal expression, and intuitive meaning. A fortuitous advantage of our approach is that the product of our two metrics yields a highly meaningful figure of merit for the overall *dependability* of a system. But techniques that enhance system dependability exact a performance cost, so we conclude with a cohesive definition of *performability* that rewards the system for performance that is delivered to its client applications, after discounting the following consequences of failure: service denial and interruption, lost work, and recovery cost.

Keywords: Reliability, Recovery, Availability, Dependability, Definitions.

*Some of this work was done while this author was on sabbatical leave at Harvard University.
†Available as ⟨`http://www.cs.bu.edu/techreports/97-011-reliability-def.ps.Z`⟩.

1 Introduction

What good is a fast but brittle system? What good is a reliable but slow system? What good is an available but unreliable system? We could go on, enumerating the eight binary permutations of three of the most critical descriptors of system value: *availability*, *reliability*, and *performance*, questioning whether each permutation is meaningful and desirable. But, we are interested in more than simple enumeration: we would like to examine each of these properties carefully, so as to ensure that its definition matches the expectations of the system's user. This scrutiny enables us to refine the definitions of availability and reliability to account for network and service states that affect the end-user, and to reward software recovery procedures for their positive impact on reliability. Equally importantly, we elucidate the relationships and trade-offs between these pivotal system qualities, to the point of deriving expressions that quantitatively capture the composite system features of *dependability* and *performability* [19]. Dependability combines availability and reliability, while performability adds performance to the mix. We believe that any reasonable comparison between fault-tolerant systems must attempt to measure and compare all of the above four quantities.

The subject of this paper is becoming increasingly important despite the continuous improvements in the reliability and overall quality of hardware components [11, 16]. A very large distributed computing system, being composed of a large number of computers and communication links, almost always functions with some part of it broken. Over time, only the identity and number of the failed components change. Failures arise from software bugs, human operator errors, performance overload, severe congestion, magnetic media failures, electronic component failures, or malicious subversion [7]. Additionally, scheduled maintenance and environmental disasters such as fires, floods and earthquakes, shut down portions of distributed systems.

We can achieve fault-tolerance by *recovering* from failures when they occur, or by *masking* failures on-the-fly. In the recovery approach, failed components are repaired or replaced, and once they become operational, the interrupted services are resumed by recovering as much state information as needed to enable the system to execute the services to completion. By contrast, the masking approach prepares for failures by keeping on-line redundant components that replicate the state of execution of services. Failed components can be replaced or repaired in the background, but service execution is not interrupted by failures. Failure masking is expensive, however, and is sometimes not available as a design option. Moreover, under large extents of failures (e.g., total failures), and under particular types of failures (e.g., network partitioning), failure masking may be inadequate to achieve fault-tolerance.

From the point of view of applications, it matters not what the sources of failures are, nor the design schemes employed to combat them; what matters is the end result in terms of the reliability and availability properties of the distributed system services these applications need. The widespread use of mission-critical applications in areas such as banking and OLTP, manufacturing, video conferencing, air traffic control, and space exploration has demonstrated a great need for highly available and reliable computing systems. These systems typically have their resources geographically distributed, and are required to remain *available* for use with very high probability at all times. Long-lived computations and long-term data storage place the additional burden of *reliability*.

We extend the notion of reliability to require that, either the system not fail at all for a given length of time—which is the standard definition—or that it recover enough state information after a failure for it to resume its service as if it was not interrupted. This definition, which we call *service reliability* differs from the traditional definition in that the latter does not reward recovery by accounting for its positive effect on reliabilty, as perceived by the user. We say that, for a system

to be reliable, either its failures must be rare, or the system must be capable of fully recovering from them. The difference between the full recovery required to enhance reliability, and the fast restart needed to heighten availability, is that recovery must reconstruct the state of the service after failures to what it was just before the failure. Activities interrupted by the failure can thus be resumed exactly as if no failure had occurred.

In the classical definition, a system is highly available if the fraction of its down-time is very small, either because failures are rare, or because it can restart very quickly after a failure. However, as a result of recovery activity after a repair, system services may not become accessible to users immediately after restoration to an operational state. We therefore deviate from the classical definition by stressing a more relevant measure, which we call *service availability*.

Given our views on how availability and reliability should be defined, much of the commonly construed relationships between both definitions simply vanishes. Under the classical definitions, availability is always a superior quality to reliability; it is simply an enhanced measure of reliability with lifetime being augmented by the repair process. Under our definitions, however, availability does not always relate to reliability. The superiority relationship holds only up till the occurrence of the first failure. This will become apparent in Section 5.

A reliable system is not necessarily highly available. For example, a reliable system that overcomes frequent failures by always recovering, and always completing all operations in progress, will spend a significant amount of time performing the recovery procedure, during which the system may deny new service requests. Another example is a system that is periodically brought down to perform backup procedures that facilitate future recovery. During the backup, which is done to enhance the reliability of the data storage operation, the system is not available to initiate the storage of new data.

Conversely, a system that does not use recovery can be highly available, but not necessarily reliable. For instance, a system that restarts itself quickly upon failures, without performing recovery actions, is more available than a system that performs recovery. Yet, the absence of recovery will render the system less reliable, especially if the failure has interrupted ongoing operations. Another available but unreliable system is one without any down time due to backups, because it is not backed up at all.

An important question, whose answer determines the end-user requirements of the system, is: which is more significant? For frequently submitted, short duration operations, availability may be more significant than reliability, given the very low probability of a failure interrupting the small duration of activity. Such operations are therefore better served by a high probability of being admitted into the system, than by a long time to failure. For long duration, relatively infrequently requested, services and long-running transactions, reliability represents a property more critical than availability. A highly available system can be useless for a very long duration service that is always admitted into the system, but never completes successfully because it is unable to run long enough before a failure aborts it in mid-stream.

In this paper, we quantify the effect of recovery on reliability and availability. We also address systems whose mixed workload requires both reliability and availability. In this case, a composite measure can more effectively assist the end-user in specifying the requirements of the system. We define *dependability* as the product of our refined reliability and availability, because their multiplication yields the probability that a service can be both initiated successfully, and terminated correctly. Also, we address complex systems that sustain partial states of failure (as opposed to either up or down binary states), in which case the system's performance degrades even though its full range of functionality remains intact. In these systems, reliability and availability definitions are not directly utilizable in a straightforward manner. To capture partial failures, we define system performability that rewards the system for every interval of time it is operational, at a reward rate

proportional to its level of performance during that interval.

This paper is organized as follows. In Section 2, we describe how systems fail. We enumerate the various sources of failures and characterize the failure and operational states of a distributed system. Section 3 introduces our refined definition of reliability, which we call *service reliability* and compares it to the classical definition of reliability. Similarly Section 4 presents *service availability* and contrasts it against the classical definition of availability. In Section 5, a brief derivation of dependability is presented, followed in Section 6 by a similar, more elaborate derivation of performability. Finally, a conclusion and a discussion are presented in Section 7.

2 How systems respond to failure

The rich variety of possible failure events and system responses to them gives rise to an equally wide range of system states that are relevant to its ability to accept and successfully carry out its tasks. Figure 1 shows the system states that we distinguish in this paper, and their structure. The states are hierarchically categorized as *operational* or *failed*. An operational state is further classified into *recovering* or *ready*. That is, a system can be operational but not quite ready to be accessed, as it performs recovery procedures. A ready state is classified, in turn, into *accessible* and *inaccessible*. For example, a system can be ready to accept requests, but network failure renders the system inaccessible to some, or all, potential clients. Similarly, a failed state is either *dead* or *under–repair*. An under–repair state is further subdivided into *recoverable* and *nonrecoverable*. In the former state, repair or replacement as well as recovery procedures are used to resume the state the system was in right before the failure. In the latter state, only restart is possible through repair or replacement.

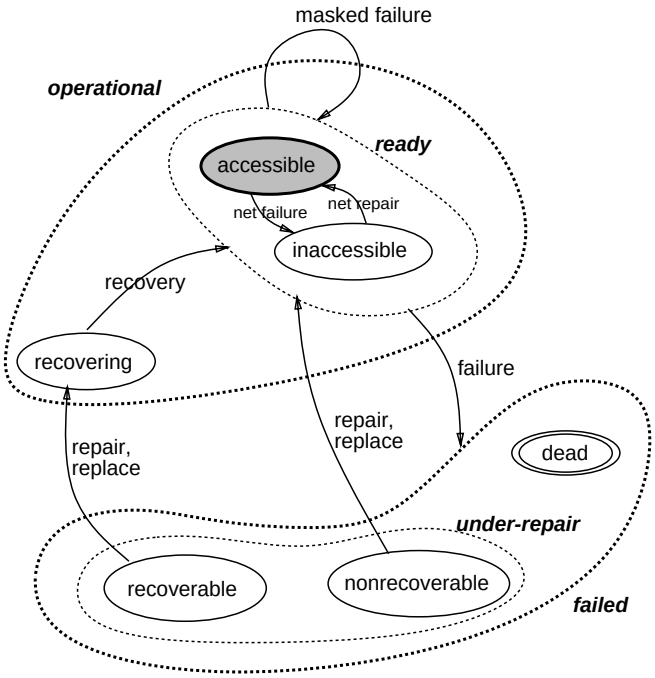

Figure 1: Failure states of a distributed system.

For a particular failure state, we define corresponding probabilistic events that take place in-

stantaneously or over a time-interval. For a state Q, $Q(t)$ denotes the event that the system is in state Q at time t, and $Q(t, t')$ denotes the event of the system being in state Q continuously in the time interval $[t, t']$. We use this notation throughout the paper.

3 Reliability

With the emergence of critical business applications such as global commerce, and with the advent of mission critical systems (like the space shuttle), the traditional definition of reliability needs to be extended to account for systems that commit to completing a system operation despite failures, once the operation is accepted by the system. In pursuit of such a commitment, a reliable system will take all needed *recovery actions* after a failure occurs, to detect it, restore operation and system states, and resume the processing of the temporarily interrupted tasks. Unfortunately, the traditional definition of reliability does not reward recovery actions. It only captures the ability of the system to operate continuously without interruption. Realizing that the field has not yet produced unified metrics to quantify recovery-enhanced reliability, we make an attempt in that direction. First, we review the standard definition of reliability then refine it and rename it as *system reliability*, R_y. Second, we propose a new definition of reliability, which we dub *service reliability*, R_s, that is similar in spirit to task-based reliability [15].

System Reliability

Reliability refers to the ability of the system to operate continuously without interruption. By tradition, *reliability* is defined as the probability that the system functions properly and continuously in the interval $[0, \tau]$, assuming that it was operational at time 0. We call this property *system reliability*, R_y. In a system without repair, $R_y(\tau)$ is the probability that the system's lifetime exceeds τ. Given the time-to-failure cumulative probability distribution function, $F(x)$, we can write $R_y(\tau) = 1 - F(\tau)$, where $F(\tau)$ is the probability that the time-to-failure is less than, or equal to, τ.

A system is perfectly reliable if it never fails. This can be attributed to the unlikely event that the constituent components are themselves perfectly reliable and the system's design suffers from no latent errors, or it can arise from the more likely event that component failures are *masked* so that they don't prevent the system as a whole from completing an ongoing service.

We refine system reliability so that: (1) it is defined starting from any point t in time, instead of from time 0, and (2) it is conditioned explicitly on the system being operational at time t. Thus, $R_y(t, \tau)$ is the conditional probability that the system does not fail in the interval $[t, t + \tau]$, given that it is operational to start with, at time t. This definition is only slightly more general than the traditional definition, but allows us to tie it smoothly with *availability*, as shown in Section 5. Formally,

$$R_y(t, \tau) = \Pr\left[ready(t, t + \tau) \setminus ready(t)\right],$$

where $ready(t, t')$ is the event of the system being continuously in the *ready* state from time t to t'.

Service Reliability

From the point of view of the service requester, it is not always necessary that the system run continuously, for the requested service to be completed successfully. Our notion of service reliability, R_s, reflects the ability of the system to complete successfully a service, even in the presence of failures, given that the system accepted the service request in the first place. Successful completion can certainly be achieved by a perfectly reliable system, but it can also be attained more realistically

and cheaply, by a system that can detect, repair, and *recover* from component failures and design errors. We define service reliability, $R_s(t, \tau)$ as the conditional probability that a request for service s that requires τ time units to complete, be successfully finished by the system at some time $t' \geq t + \tau$, given that s was properly initiated at time t. If the system experiences no failures in the interval $[t, t + \tau]$, then $t' = t + \tau$. However, we allow the system to accumulate the requisite τ units of execution time by aggregating work performed in-between failures, so long as the system is able to recover from each of these failures. When failures interrupt the processing of s, we have $t' > t + \tau$, to account for downtime, repair time, recovery time, and any work done just before the failure that is lost.

From the above discussion, it follows immediately that, in a system that does not employ recovery,

$$R_s(t, \tau) = R_y(t, \tau), \tag{1}$$

but that, when recovery's contribution to reliability is accounted for, we have

$$R_s(t, \tau) \geq R_y(t, \tau). \tag{2}$$

An accurate formalization of R_s that captures the recovery aspect of the system must account for two random phenomena: the *recoverability* of failures, and the ability of the system to accumulate at least τ time units before it fails fatally. Certain types of failures can be fatal, or unrecoverable, such as undetected malicious subversion, failures of the recovery subsystem itself, and catastrophic failures that wipe out all resources that may be used for repair or replacement. Let $U_i(t)$ denote the *uptime* between the recovery from f_{i-1}, the $(i-1)$st failure after time t, and f_i, the ith failure. If failure f_{i-1} is unrecoverable, then $U_k(t) = 0$, for all $k \geq i$. We can now define service reliability, assuming that no work is lost and that the system is restored to its full capacity, when a failure is recovered from:

$$R_s(t, \tau) = \Pr\left[\sum_i U_i(t) \geq \tau \setminus ready(t)\right]. \tag{3}$$

In other words, service reliability is the conditional probability that the system will eventually be able to accumulate enough uptime to perform the τ units of time required by service s, given that the system was available to start work on s at time t. Since not all failures are recoverable, $R_s(t, \tau)$ must decay as a function of τ. As τ increases, more failures are likely to occur, and the probability that a fatal unrecoverable failure will strike grows. Conversely, $R_s(t, 0) = 1$, always.

The conditional probability in the definition limits R_s to the case in which the system is operational at time t and the service is admitted to the system. This excludes the situation where the system is in a failure state at time t, or is recovering at time t after being repaired. This is not an arbitrary choice, for the fundamental intuition underlying reliability focuses on the system's capacity to finish what it starts.

As an example, we calculate R_s in the simple case where execution of s can be interrupted by at most one failure, then generalize our analysis to any number of failures. Figure 2 shows the case of the one failure occuring, which must necessarily happen before $t + \tau$ to have any effect on the service s that is assumed to have begun at time t. Assume that all $\{U_i(t)\}$ are independent random variables with identical stationary (time-independent) cumulative distributions $F(u) = \Pr[U \leq u]$.

$$\begin{aligned} R_s(t, \tau) &= \Pr[U_1(t) \geq \tau \text{ or } \{U_1(t) < \tau \text{ and } U_1(t) + U_2(t) \geq \tau \text{ and } f_1 \text{ recoverable}\}] \\ &= \Pr[U \geq \tau \text{ or } \{U < \tau \text{ and } 2U \geq \tau \text{ and } f_1 \text{ recoverable}\}] \end{aligned}$$

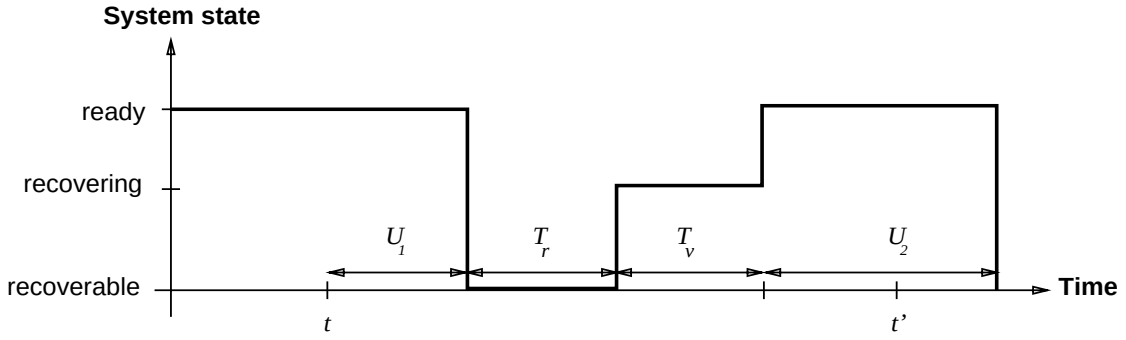

Figure 2: An example execution that is interrupted by one failure.

$$
\begin{aligned}
&= \Pr\left[U \geq \tau \text{ or } \left\{\frac{\tau}{2} \leq U < \tau \text{ and } f_1 \text{ recoverable}\right\}\right] \\
&= 1 - F(\tau) + c \cdot \left[F(\tau) - F\left(\frac{\tau}{2}\right)\right], \qquad (4)
\end{aligned}
$$

where c is the probability that a failure is recoverable[1]. Noting that $R_y(t,\tau) = 1 - F(\tau)$, we express R_s in terms of R_y and c, using $R(\tau)$ as a shorthand for $R_y(t,\tau)$ to avoid clutter.

$$R_s(t,\tau) = c \cdot R\left(\frac{\tau}{2}\right) + (1-c) \cdot R(\tau), \qquad (5)$$

when at most one failure can occur. In general, for an arbitrary number of n or fewer failures,

$$
\begin{aligned}
R_s(t,\tau) &= R(\tau) + c \cdot \left[R\left(\frac{\tau}{2}\right) - R(\tau)\right] + \cdots + c^n \cdot \left[R\left(\frac{\tau}{n+1}\right) - R\left(\frac{\tau}{n}\right)\right] \\
&= c^n \cdot R\left(\frac{\tau}{n+1}\right) + (1-c) \cdot \sum_{k=1}^{n} c^{k-1} \cdot R\left(\frac{\tau}{k}\right) \\
&\geq c^n \cdot R\left(\frac{\tau}{n+1}\right) + (1-c^n) \cdot R(\tau) \qquad (6)
\end{aligned}
$$

The above formulae yield insight into the relationship between our definition of service reliability and that of system reliability. In the case of a system that cannot recover from any failure, $c = 0$, and $R_s(t,\tau) = R(\tau)$, as would be expected. At the other extreme, a perfectly recoverable system has $c = 1$, which yields perfect service reliability: $R_s(t,\tau) \to R(0) = 1$, as $n \to \infty$, no matter how low system reliability may be. In the region that lies between these two extremes, we rely for our interpretation on the fact that $R(\tau/2) \geq R(\tau)$ always. As the failure coverage of the recovery component of a system increases, so does c, which can dramatically raise service reliability by allowing the service to aggregate progressively more numerous, and hence smaller and more likely, uptime intervals (times-to-failure).

For very long missions, where τ can be so large that $R(\tau)$ becomes negligible, our formulation quantifies the precise advantage to be gained by recovery, in rendering the effective reliability (i.e., service reliability) acceptable. Recovery can, measurably and dramatically reduce the coupling between mission duration and system lifetime.

[1]The quantity c is sometimes called *coverage*.

The above analysis and discussion illustrates how our definition of service reliability R_s recognizes the impact of repair and recovery processes on reliability, by rewarding the ability to recover from a high percentage of failures. At the same time, R_s preserves and reflects any improvements in system reliability, although the quantitative extent to which such gains contribute to R_s is inversely proportional to the quality of the recovery subsystem!

4 Availability

Availability refers to the accessibility of the system to users. A system is available if its users' requests for service are accepted at the time of their submission. Unlike reliability, availability is instantaneous. The former focuses on the duration of time a system is expected to remain in continuous operation—or effectively so in the case of recovery-enhanced reliability—starting in a normal state of operation. The latter concentrates on the fraction of time instants where the system is operational in the sense of being accessible to the end user.

System and Service Availability

The traditional definition of availability captures only overall failure and repair characteristics of the distributed system hardware. It neither captures the *readiness* of the system to perform a particular operation or service, nor the system's *accessibility* over the network. This can lead to overestimating the availability of systems that may not always be ready to accept new service requests when operational, *e.g.*, systems that use recovery to enhance reliability. During recovery, new service requests may be blocked while the system is necessarily operational and accessible. Furthermore, if the "system" is narrowly construed to contain the minimal set of server and network resources that is sufficient for proper operation, then it is still possible for the rest of the network may experience difficulties that prevent communication between clients and the system. As a result, availability can degrade well beyond the estimate of the traditional definition, so far as service *users* are concerned.

Availability, therefore, needs to be redefined for recovery-enhanced reliable systems that are internetworked with clients. In this paper, we make an attempt in that direction. We define *service availability* A_s, and distinguish it from the traditional definition of availability, which we rename as *system availability*, and denote it by A_y. With reference to the state diagram of Figure 1, we write:

$$A_y(t) = \Pr[operational(t)] \tag{7}$$

$$A_s(t) = \Pr[ready(t)] \tag{8}$$

$$= \frac{1}{t}\sum_i U_i(0,t) \tag{9}$$

where $U_i(t_1, t_2)$ is the *uptime* between the recovery from f_{i-1}, the $(i-1)$st failure in the time interval $[t_1, t_2]$, and f_i, the ith failure in the same interval. In addition, we can also define *customer availability*, which is similar to Larry Raab's *site availability* [9].

$$A_c(t) = \Pr[accessible(t)] \tag{10}$$

All three measures reflect *instantaneous availability* [20]. The definitions immediately imply that $A_c \leq A_s \leq A_y$, which is what we mean by saying that A_y overestimates availability. This arises because system availability A_y does not reward fast recovery, since the system is already considered operational once it is repaired, even though it may yet have to recover. Similarly, service availability ignores the possibility that the system may be ready, but inaccessible to some of its customers.

Assuming at most one failure can occur in the interval $[0, t]$, we can express A_y and A_s with reference to Figure 2, after setting $t \leftarrow 0$, and $t' \leftarrow t$. By following an analysis similar to that used to express R_s in Section 3, and writing $R(t)$ as a shorthand for $R_y(0, t)$.

$$A_y(t) = R(t) + [1 - R(t)] \cdot [1 - R(t - T_r)] \quad (11)$$

$$A_s(t) = R(t) + [1 - R(t)] \cdot [1 - R(t - T_r - T_v)] \quad (12)$$

The above expressions seem to indicate that availability improves as t increases, but this is only because we artificially constrained the number of failures to a maximum of one during time interval $[0, t]$. What remedies this is the measure, $\lim_{t\to\infty} A(t)$—known as the *limiting availability*—which can be shown to depend only on the mean time to fail and the mean time to repair,

$$\lim_{t\to\infty} A_y(t) = \frac{U}{U + T_r}$$

$$\lim_{t\to\infty} A_s(t) = \frac{U}{U + T_r + T_v}$$

but not on the nature of the distributions of failure times, repair times [20], or, by analogy, recovery time.

Service and customer availability as defined above are scalar measures, that necessarily induce a total ordering on any systems that are compared in these terms. Other measures that are more complex, but that lead to more detailed design insights, have been proposed in the literature [1, 3, 6, 8, 12, 13]. For example, availability can be measured in a combinatoric sense by the size, composition, and number of the various alternative sets of resources that suffice to keep the system ready to accept new service requests. Such metrics are invented to help reveal information about the impact and relative merit of certain design choices and load conditions, such as the number and placement of replicas, degree of concurrency, transaction length, data access distribution, operation mix, *etc.*The scalar and combinatoric availability metrics, however, do not conflict, since it should always be possible, and, we believe, extremely desirable, to compute the scalar metrics from the combinatoric ones. This enables fair across-the-board comparisons between whole systems, accounting for such critical phenomena as recovery and failure-prone internetworking of remote clients with the systems that serve them.

5 Dependability

The overall success of service s depends both on its correct initiation on demand *and* on its successful termination. Therefore, we define the *dependability*[2] of s, $D_s(t, \tau)$, as the probability of the conjunction of the two events, that of $ready(t)$ and that of accumulating enough work in the duration from t to $t' \geq t + \tau$ to complete the τ units of work required by s (as defined in Section 3). Let $T_s(t, \tau)$ denote the latter event. Recall that,

$$A_s(t) = \Pr[ready(t)]$$

$$R_s(t, \tau) = \Pr[T_s(t, \tau) \setminus ready(t)]$$

Bayes' law of conditional probability dictates that dependability be the product of availability and reliability, as follows:

$$D_s(t, \tau) = \Pr[ready(t) \wedge T_s(t, \tau)]$$

$$= A_s(t) \cdot R_s(t, \tau) \quad (13)$$

[2]Other researchers include into the notion of dependability additional dimensions such as security against malicious attack and safety from disasterous consequences of failure.

So we have in this definition of dependability, the happy coincidence of a single scalar metric that measures the full *initiation-to-termination probability* of successful service. We ommit discussion of *customer dependability*, except to state that it can be defined, in terms of customer availability instead of service availability, simply as $D_c(t, \tau = A_c(t) \cdot R_s(t, \tau)$.

6 Performability

The implicit assumption in the above analysis of availability and reliability is that the relevant system states are binary: either the system is up and running, or it is not. This simplistic view does hold true for systems that cannot tolerate failures, but for fault-tolerant systems, many more system states become important, one for every possible masked failure pattern. Under such partial failures, the system's *performance* degrades, even as its full range of functionality remains intact. One way to measure the consequences is to reward the system for every time unit it is ready, at a rate proportional to its performance during that interval.

The resulting metric is termed *performability*, since it combines both performance and dependability [19]. To arrive at a quantitative measure of performability, $Y_s(t, \tau)$, we reward the system for dependable performance in the time interval $[t, t']$ where $t' \gg t + \tau$. We denote the reward rate during that interval by $r_s(g(x))$, $t \leq x \leq t'$, which is the performance at time x when the system's failure configuration is $g(x)$. The traditional definition of performability [14, 19, 21] is the simple integration of the reward function.

$$\frac{1}{t' - t} \int_t^{t'} r(g(x))dx.$$

A shortcomming of this definition is that it rewards the system for every subinterval of $[t, t']$ in which it has a positive reward rate, regardless of whether this subinterval can contribute towards the successful initiation, and execution to completion, of any service of the system. We prefer to use the more realistic,

$$Y_s(t, \tau) \quad = \quad D_s(t, \tau) \cdot \frac{1}{t' - t} \int_t^{t'} r(g(x))dx, \tag{14}$$

whose mean value is

$$\overline{Y}_s(t, \tau) \quad = \quad A_s(t) \cdot R_s(t, \tau) \cdot \overline{r} \cdot A_s(t) \tag{15}$$

$$= \quad [A_s(t)]^2 \cdot R_s(t, \tau) \cdot \overline{r} \tag{16}$$

where $\overline{r}$ is the average performance[3] while the system is *ready*, and $\overline{r} \cdot A_s(t)$ is the average performance after discounting downtime. We can view this definition as dependability weighted by performance, or as performance weighted by dependability. A curious consequence of this definition is that availability has a higher impact on performability, than reliability does.

Here too, we make only quick mention of *customer performability*, which emerges by substituting customer availability A_c for service availability A_s in the above formulae.

[3]For systems whose steady state probability distribution of failure configuration is stationary, we have $\bar{r} = \sum_i \pi_i \cdot r(g_i)$, where π_i is the probability of the system being in partial failure configuration g_i. Note that this includes only configurations in which the system is *ready*, i.e., able to function, albeit at lower performance.

7 Discussion

In this paper, we revised the classical definitions of reliability and availability. We considered the positive effect of recovery on reliability, and the potentially negative effect on availability. Our revised definitions, which we call *service reliability* and *service availability* improve the match between their formal expression and intuitive meaning. A natural definition of *dependability* fortuitously results from adopting a conditional probability framework. This definition is meaningful in systems that require both high reliability and high availability, or in which trading off one for the other is an acceptable route to maximizing dependability. Air traffic control systems are a famous example [4, 10] that can benefit from our dependability metric. Finally, we reformulated the definition of *performability* as a composite measure of dependability and performance, for systems that exhibit smooth degradation in performance in response to failures, ranging from fully operational, to slowly or partially operational, to completely failed. An expression for evaluating performability is derived, that recognizes the joint contributions of unavailability, unreliability and low performance, in determining the net level of service that is deliverable to clients of a distributed system.

The ultimate test of a new modelling or evaluation framework rests on whether it influences design decisions. Our work paves the way for a systematic trade-off of reliability against availability in order to optimize dependability or performability, an approach that has proven successful in a more limited context [17, 18]. For example, consider two different recovery subsystems, V, W. V is more sophisticated, and hence has a higher coverage $c_v > c_w$. By the same token, V takes longer to recover from a failure, and incurs more overhead during normal operation, thus, $A_v < A_w$, and $\overline{r}_v < \overline{r}_w$. Clearly, if reliability is the only concern, then V is the better choice, but if overall dependability and performability are computed, it may well turn out to be more advantageous to adopt W.

The traditional definitions are valuable because of their mathematical tractability, and the large body of results and software [2, 5] that exists for them. Even though our revised definitions may prove more difficult for analytic evaluation, they are all directly measurable by simulation and system monitoring studies[4]. Finally, we note that adding to our suite of definitions, a real-time deadline interval δ, such that a request for service submitted at time t must be completed by time $t + \delta$ is a straightforward and worthwhile exercise.

Acknowledgements. We would like to thank Mark Crovella for his direct, constructive, criticism, and Mohammad Al-Ansari and Robert Carter for many provocative discussions. The members of the Oceans Group[5] provided much interesting feedback when we presented our ideas in their early form.

References

[1] B. Bhargava, A. Helal, and K. Friesen. Analyzing availability of replicated database systems. *Int'l Journal of Computer Simulation*, 1(4):393–418, Dec. 1991.

[2] Alvin M. Blum, Ambuj Goyal, Philip Heidelberger, Stephen S. Lavenberg, Marvin K. Nakayama, and Perwez Shahabuddin. Modeling and analysis of system dependability using the system availability estimator. In *Digest 24th IEEE Int. Symp. on Fault-tolerant Computing*, 1994.

[4] Admittedly, simulating for *customer* availability, dependability and performability will be more costly than for *service*-based metrics.

[5] `http://www.cs.bu.edu/groups/oceans/`

[3] B.A. Coan, B.M. Oki, and E.K. Kolodner. Limitations on database availability when networks partition. In *Proc. 5th ACM Symp. on Principles of Distributed Computing, Calgary, Canada*, 187–194, Aug. 1986.

[4] F. Cristian, B. Dancey, and J. Dehn. Fault-tolerance in air traffic control systems. *ACM Trans. on Computer Systems*, 14(3):265–286, Aug. 1996.

[5] Stacy A. Doyle and Joanne Bechta Dugan. Dependability assessment using binary decision diagrams (bdds). In *Digest 25th IEEE Int. Symp. on Fault-tolerant Computing*, 1995.

[6] H. Garcia-Molina and J. Kent. Performance evaluation of reliable distributed systems. In B. Bhargava, editor, *Concurrency Control and Reliability in Distributed Systems*, pages 454–488. Van Nostrand Reinhold, 1987.

[7] Jim Gray. A census of Tandem system availability between 1985 and 1990. *IEEE Trans. on Reliability*, 39(4):409–418, Oct. 1990.

[8] A. Helal. Modeling database system availability under network partitioning. *Information Sciences*, 83(1–2):23–35, Mar. 1995.

[9] D.B. Johnson and L.J. Raab. A tight upper bound on the benefits of replication and consistency control protocols. In *Proc. 10th ACM Symp. on Principles of Database Systems, Denver, Colorado*, May 1991.

[10] K. Kanoun, M. Borrel, T. Morteveille, and A. Peytavin. Modeling the dependability of the french air traffic control system. In *Digest 26th IEEE Ann. Int. Symp. on Fault-tolerant Computing, Sendai, Japan*, June 1996.

[11] Jean-Claude Laprie. Dependable computing: Concepts, limits, challenges. In *Digest 25th IEEE Int. Symp. on Fault-tolerant Computing*, 1995.

[12] G. Martella, B. Ronchetti, and F. Shreiber. Availability evaluation in distributed database systems. *Performance Evaluation*, pages 201–211, 1981.

[13] R. Mukkamala. Measuring the effect of data distribution and replication policies on performance evaluation of distributed database systems. In *Proc. 10th IEEE Intl. Conf. on Data Engineering*, Feb. 1989.

[14] H. Nabli and B. Sericola. Performability analysis of fault-tolerant computer systems. Technical Report 805, IRISA, Campus de Beaulieu, Rennes, France, Mar. 1994.

[15] D. K. Pradhan and N. H. Vaidya. Roll-forward and rollback recovery: Performance-reliability trade-off. In *Digest 24th IEEE Int. Symp. on Fault-tolerant Computing*, 1994.

[16] Brian Randell. Software dependability: A personal view. In *Digest 25th IEEE Int. Symp. on Fault-tolerant Computing*, 1995.

[17] R.M. Smith and K.S. Trivedi. A performability analysis of two multiprocessor systems. In *Digest 17th IEEE Int. Symp. on Fault-tolerant Computing, Pittsburgh, PA*, pages 224–229, July 1987.

[18] Ann T. Tai. Performability-driven adaptive fault tolerance. In *Digest 24th IEEE Int. Symp. on Fault-tolerant Computing*, 1994.

[19] A.T. Tai, J.F. Meyer, and A. Avizienis. *Software Performability: From Concepts to Applications*, volume 347 of *International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, London, Dordrecht, 1996.

[20] K.S. Trivedi. *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*. Prentice-Hall, 1982.

[21] K.S. Trivedi, G. Ciardo, M. Malhotra, and R.A. Sahner. Dependability and performability analysis. Technical Report ICASE 93-85, NASA Langley Research Center, Nov. 1993.