
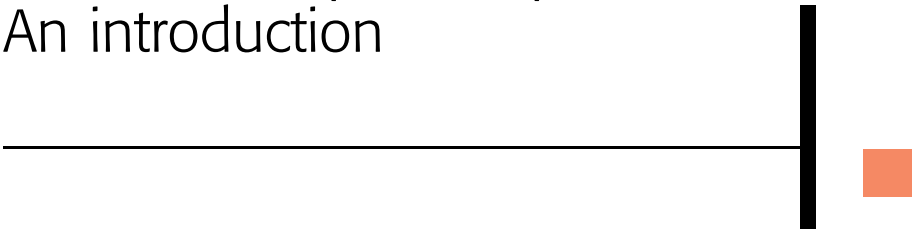


Three reliability engineering techniques and their application to evaluating the availability of IT systems: An introduction



D. Bailey
E. Frank-Schultz
P. Lindeque
J. L. Temple III

We present a brief introduction to three reliability engineering techniques: failure mode, effects, and criticality analysis; reliability block diagrams; and fault tree analysis. We demonstrate the use of one of these techniques, reliability block diagrams, in evaluating the availability of information technology (IT) systems through a case study involving an IT system supported by a three-tier Web-server configuration.

INTRODUCTION

The overall complexity of information technology (IT) systems has grown dramatically in recent years. With the increasing dependency in our society on IT systems, customer requirements for service resilience and high availability are becoming more stringent. Because of external threats and because of the risks caused by increased complexity and quality problems, the design of IT systems for high availability remains a challenge.

We have identified three phases in the evolution of high-availability systems, during which the focus shifted from component-level availability to business-process availability. In the first phase of this evolution, designing for high availability meant focusing on individual components and the hardware infrastructure (see, for example, Reference 1).

In the next phase, as the IT industry evolved, the focus shifted to delivering high-availability services from the end-user perspective. This trend demanded that attention be paid not only to the component and infrastructure level but also to applications, data, and middleware.

Now, in the third phase of this evolution, the focus is shifting to the availability of complete business processes, which may be made up of various IT processes and services. The genesis and constituent technologies of these processes vary widely, yet the

©Copyright 2008 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the Journal reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to republish any other portion of the paper must be obtained from the Editor. 0018-8670/08/\$5.00 © 2008 IBM

outcome for the user and the business depends on the entire business process.

The use of formal analysis techniques is more prevalent in industries that have produced mission-critical systems, such as defense and aerospace. Such techniques can also be put to good use in the IT industry by promoting a more rigorous approach to the design of high-availability systems.²⁻⁴ Although many sophisticated techniques are available to analyze and model the dynamic behavior of systems,^{2,3} adoption rates of these has, in our experience, been low. Key inhibitors are complexity, lack of skills, and the cost of adoption.

Three reliability engineering techniques that are sufficiently simple yet effective in the design of high-availability systems are: *failure mode, effects, and criticality analysis* (FMECA); *reliability block diagram* (RBD); and *fault tree analysis* (FTA).⁶⁻¹⁰ In this paper, we provide a brief description of these techniques and present a case study that involves the application of RBD for evaluating the availability of an IT system.

The rest of this paper is organized as follows. In the next section, we present an overview of the practice of high-availability design and describe the main steps involved in the process. In the following section, we describe each of the three reliability engineering techniques and list their benefits and limitations. In the section after that, we illustrate the use of these techniques by applying RBD to the design of a high-availability IT system based on a three-tier server configuration. The last section contains our final comments.

THE PRACTICE OF HIGH-AVAILABILITY DESIGN

In this section, we summarize the current practice of high-availability design by describing its main activities: gather availability requirements, determine the cost of outages, design the key components for high availability, make use of availability patterns, and validate the test strategy.

We use in this paper a number of terms such as availability, reliability, and resilience. *Availability* is the ability of a system to perform its required function at a random point in time. It is usually expressed as the availability ratio: $[(\text{agreed service time} - \text{unavailable time}) / (\text{agreed service time})] \times 100\%$. We say a system has high availability if it

provides service during defined periods, at acceptable or agreed upon levels, and masks unplanned outages from end users.

Reliability is the ability of a component to perform its intended function for a specified interval and under stated conditions. It is usually expressed as a probability. Simply stated, it shows the time interval over which the component is expected to work. *Resilience* is the ability of a system to recover from failure and continue to function.

Gather the availability requirements

To determine the availability requirements of a system, one must identify the main users of the system and then determine what these users expect from the system. Examples of availability requirements include:

- The days and hours a service is to be available;
- Special periods of a day when core business services must be available;
- Special considerations at weekends, month ends, vacations, etc.;
- How quickly the system must be restored and the limits on the amount of data lost in the event of failure; and
- Any degraded levels of service that might be acceptable.

Once this information is available, a decision must be made with regard to the availability strategy that is adopted for the solution. Availability requirements are too often expressed as 99.9-percent service availability during office hours, e.g., 5 a.m. to 10 p.m., or no more than 3 hours downtime per month averaged over 1 year. What these figures do not tell us is whether the system should be designed to cope with a few lengthy outages or a larger number of short outages. The availability design decisions can be quite different for these two cases.

A time line should be created that captures the key business processes and the critical periods when different processes must run. If the availability requirements identify many types of users (i.e., hundreds of types of users), the number may be too high to handle, as each type may have different availability requirements. This number can be reduced by grouping them into a more reasonable value, say, 20 user roles.

Costs associated with service outage

Cost plays an important role in determining how much redundancy can be incorporated into a system in order to enhance its availability. Thus, it might not be possible to achieve the desired availability because of cost constraints. An understanding of the real value of service availability will help designers to better select where to invest the greatest effort to provide this availability.

It is important to look at each of the services provided to the users and to assess what costs, such as financial costs, might be associated with an outage to that service. System designers need to consider items such as: lost revenue from existing business, lost opportunity for new business, loss of operational productivity, financial penalties, and loss of brand image, customer confidence, or loyalty. In addition, there are special situations, such as a major sporting event, when the service must be available “at all costs.”

The cost of a service outage may vary with the time of the day or day of the week. For instance, ATM (automated teller machine) cash services at 1 p.m. on a Friday are more critical than the same services at 2 a.m. on a Sunday.

The evaluation of costs associated with service outage should include answers to the following questions: Which services incur cost penalties if they are not available? Are any of the services time-sensitive? What is the monetary value of any outages? Does the design need to be changed to meet the cost implications of service outages or should these cost implications be reviewed and perhaps changed? These findings should be checked with the sponsor to determine whether they are correct and to agree on any possible changes required.

High-availability design of key components

This task starts with a study of the overall design of the system whose purpose is to determine the granularity of key components for which a high-availability design is required. The level of granularity selected determines the amount of effort required and the accuracy of the high-availability design outcome. It is important that availability data for the chosen components be available and that the design makes these components as independent from each other as possible.

Invariably each component can be broken down into subcomponents. A computer center can be decomposed into servers, cables, workstations, etc. It may be necessary to decompose the servers into types and then into the components that make up each type. Even a workstation can be decomposed into screen, disk, power supply, etc. In most cases, however, it will not be necessary to decompose the system to that level of detail.

A number of key decisions are made at this stage. The key components of the system are identified and their availability characteristics, such as mean time between failure (MTBF) and mean time to recover (MTTR), are specified.^{11,12} In addition, the critical components required to support the core business services are identified.

It is important to also identify out-of-line situations such as: “the mean time to replace, repair, or restart a failed component is two hours and the service downtime is limited to one hour,” or “a component is likely to fail once per week but the service must not fail more than once per month.”

For the components created specifically to provide high availability, the following questions need to be answered: Are these components required to maintain state and should they be able to recover to a point in time prior to the failure? What levels of component recovery are required (e.g., should the recovery include all data processed)? What are the recovery points, that is, where will the recovery information be coming from? What special mechanisms will be required in order to control the recovery of the component?

Make use of availability patterns

Reusable patterns provide us with a structured approach to capturing recognized, repeatable best practices. Although availability patterns are not currently used on a wide scale, we believe they will play an important role in the future practice of availability design.

There is clear evidence of the value of reusable patterns and their use throughout the industry. Patterns have focused primarily on the design and programming phases of application development, dealing with the structure and behavior of the code that supports the application.¹³

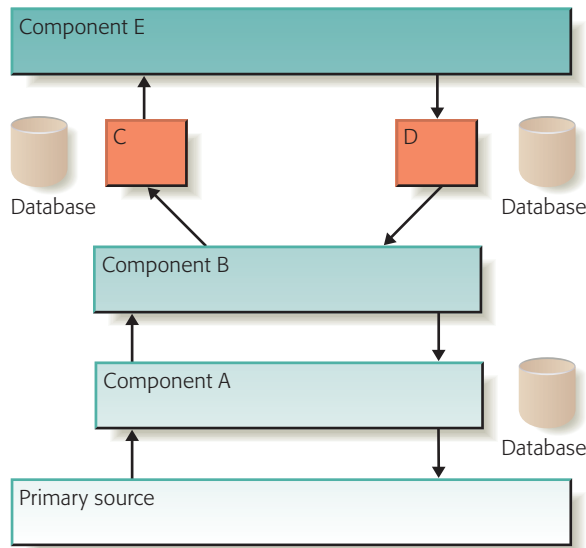


Figure 1
An application of the store-and-forward pattern

Availability patterns are component-level patterns, which identify and describe the overall structure and interactions that occur between components of a system. They enable the reuse of existing solutions when designing a system with specified availability properties. We illustrate the use of availability patterns with an example. The *store-and-forward* pattern, which is similar to existing patterns,¹⁴ is used here to demonstrate the concept.

We consider a system in which messages from a primary (possibly external) source are processed in a number of steps. If one of the processing component fails, then messages that were “in flight” when the failure occurred may be lost and have to be resent by the primary source. In order to speed up the time needed to restore the service (and therefore improve availability), rather than retrieve the lost message from the primary source, we routinely store the message at intermediate locations on the message path and, in case of failure, retrieve the message from the last storage location on the message path.

To apply this pattern the designer determines the required message storage locations and ensures that messages are committed to persistent storage at each of these locations. **Figure 1** shows an application of the store-and-forward pattern in a system consisting of components A through E. Components

A, C, and D are provided with databases for storing messages. Each one of these has the following responsibilities: 1) receive message; 2) store message (logging); and 3) forward message.

If component E fails, the recovery process retrieves the last message from C. Thus, upon the successful restart of E, messages lost in transit between C and E are retrieved from C. Messages flowing in the reverse direction are stored at D before being passed on to B.

Other uses of patterns are the identifications of anti-patterns and the analysis of failure patterns.¹⁵

Validate the test strategy for availability

A test strategy and plan are needed to ensure that the desired availability characteristics are achieved. For example, failure injection tests can be added to the test plans to confirm that recovery performs as designed.^{16,17} This is primarily a risk reduction exercise because many nontrivial high-availability systems cannot be fully tested for their availability and reliability characteristics.

Sometimes it is not possible to test an entire system with a complete end-to-end message flow. Increasingly, with e-commerce systems, the constituent parts of the system may have different owners, which can make testing the system in its entirety difficult. Nevertheless, it is important that all critical interfaces, especially external interfaces, are thoroughly tested.

Availability and resilience testing often involves the selective “destruction” of a working system, such as disabling servers, corrupting data, and so on. It is therefore unwise to assume that availability testing can be done using the same testing environments that are being used for functional or performance testing.

Testing operational recovery not only includes testing the systems recovering the service but the ability of the operational staff to follow the procedures to manage the service. Operational proving is an important step to ensure that procedures for service recovery are tested alongside the recovery of the system.

Some of the key decisions during the test stage amount to providing answers to the following

questions: Which current test practices need to be updated? How many of the nodes and components and their subsystems should undergo availability testing? Can the system, as designed, be tested in a way that ensures, with a specified level of confidence, that it will provide the expected availability of service to end users? Is the operational support adequate, or are design changes required to make it easier to manage the system?

Tools

The development and maintenance of highly available IT systems also require specialized tools, which use sophisticated mathematical algorithms to calculate failure rates and other reliability and availability measures. A number of commercial tools are available today, such as the ones from Relex Software Corporation, Isograph Ltd., and ITEM Software.^{18–20}

Because models of complex systems can become quite complex, reliability engineering tools should be used for all but the most trivial of models. Key benefits provided by such tools are: user-friendly data entry; dependency management and data consistency checking; calculation of availability predictions; ability to run stochastic simulations to predict system reliability characteristics; and a consistent format for storing models, to enable and encourage reuse. The cost of these tools may seem high, but it is not difficult to make a business case for their procurement given their labor-saving ability.

RELIABILITY ENGINEERING TECHNIQUES

Reliability engineering techniques (such as the three that we consider here) enjoy widespread use in industries such as aerospace and defense, where reliability and availability are critical. These techniques are currently being used in computer hardware design but have, unfortunately, not been widely adopted in the development of entire IT systems even though such systems are becoming more complex and more critical for the business.

Although more advanced techniques for evaluating availability have been developed,^{3,4} adoption of such techniques in IT organizations has, in our experience, been limited. We believe that the three techniques discussed here, which have been shown to be mathematically related,²¹ provide a reasonably low barrier to use by IT staff, although we recognize

that they do rely on a number of simplifying assumptions that may not always hold true for every system.

One of the key benefits resulting from the application of the techniques is that they force the analyst to follow a systematic procedure of analysis of the system. In most cases, the mere construction of the model leads to a better understanding of the system design, including aspects such as component interdependencies and reliability weaknesses.

Failure mode, effects, and criticality analysis

FMECA is a risk assessment technique for complex systems that have stringent reliability, availability, or safety requirements. The FMECA technique allows the engineer to consider how the failure modes of each system component can result in system performance problems and to ensure that appropriate safeguards against such problems are put in place. It also provides an opportunity to define the detection and recovery mechanisms that should be in place when failures do occur.

The FMECA technique can be used for the design of products (including software), processes, and services. The principal objective is to anticipate the most important design issues early in the development life cycle so that one or both of the following actions can be taken: a) modify the design to eliminate or mitigate the problem; and b) minimize the consequences by means such as early detection and well-defined recovery mechanisms.

Table 1 shows a FMECA worksheet for the function *generate payments*. The worksheet shows that there are two ways in which the function can fail: *crash* and *hang*. For each of these failure modes, the FMECA worksheet captures details about the type of failure. For example, a crash can be detected by the system monitoring software, whereas a hang is more difficult to detect and needs a manual check.

FMECA provides a number of benefits. The use of FMECA usually leads to better system design, which results in fewer critical failures and more predictable recovery from failures. It enables early identification of diagnostic tools and operational procedure requirements. It also enables earlier identification of design weaknesses and thus fewer design changes in later phases of the life cycle.

Table 1 FMECA worksheet for function *generate_payments*

FMECA Sheet									
Project Name: Payment project									
System Identifier: Payment system									
Version: 1.234									
Date: 1 January									
Item #	Functions performed by item	Failure modes	Root cause (optional)	Probability	Operational mode	Failure effects (local, higher level and end effects)	Detection method	Compensating provisions	Severity
1928	Generate payments	Crash	Programming error	Medium	Real-time input window	End-effect: None	Tivoli alert generated if process not running	Automatic restart	Low
			Programming error	Medium	Batch window	End-effect: No payments to process	Tivoli alert generated if process not running	Automatic restart	High
		Hang	Programming error	Low	Real-time input window	End-effect: None	None	None	Low
			Programming error	Low	Batch window	End-effect: No payments to process	Regular manual check on throughput	Stop and restart manually	Critical

FMECA provides input to the test strategy and test case design, allowing early identification of negative test scenarios. It also provides information that can be used (via export from most FMECA tools) to create draft fault trees that can be further analyzed using fault-tree analysis (FTA). It can also be used as a model against which to perform impact assessment when changes to the system are considered.

The FMECA involves certain limitations. It provides a static system analysis that does not consider the impact of multiple component failures, latent defects that impact timing and sequencing, or effects on redundant components (in which case RBDs and FTA can be used).

Reliability block diagram

RBDs provide a graphical means for representing availability-related system dependencies. Whereas

this can be useful in itself, the real value of this technique is realized when it is used to predict overall system availability. A detailed case study using RBD is presented in the section “Example of Reliability Block Diagram Analysis.”

RBDs can be used to predict the reliability of systems in which the reliability characteristics of the components that make up the system are known (or can be estimated to a reasonable degree of accuracy). Given the limited quantitative failure information available for software components, this technique is currently mainly used for hardware components.

RBDs provide a number of benefits. An RBD provides an intuitive graphical representation of the system from a reliability perspective. Because RBDs

can be nested, the techniques can be viewed as scalable from simple to highly complex systems.

The RBD technique can be used to identify components that are critical from a reliability point of view, and, in particular, it can be used to identify single points of failure. It can also be used to assess the impact that design changes will have on system reliability.

The RBD technique has a number of limitations. Reliability predictions are only as accurate as the failure data available for the components that make up the system. It should be noted that, even if absolute values are not very accurate, an RBD analysis can still be useful for comparing alternative design options and for performing sensitivity analysis.

RBDs assume that items fail independently of each other. This may not always be the case. If, for example, an item overheats due to the conditions that lead to its failure, then the overheating may affect the redundant component intended to provide the function of the failed component.

Fault-tree analysis

Fault-tree analysis (FTA) is a top-down, event-oriented analysis technique that provides a structured approach to identifying the basic (lowest-level) events that caused the system failure. FTA uses a graphical tree structure to represent all the events, including those caused by humans.

The event representing the failure of the entire system, which is represented as the root of the tree, is also called the *top event*; the lowest-level events, which are shown as the leaves of the tree, are called *basic events*. The nodes of the tree between the root and the leaves involve logic symbols that combine the events corresponding to the subtrees at each node. Thus, Boolean algebra and probability theory can be used to perform quantitative and qualitative analysis of the fault trees.

Figure 2 shows a simple fault tree representing the possible failures in the system supporting a Web site. If any one of the events feeding an OR gate takes place, then the event marked as the output of the OR gate is triggered. For example, if either event “BT line down” or event “AT&T line down” takes

place, then event “WAN down” is triggered. The INHIBIT gate is equivalent to an AND gate with an additional condition added (on the side of the INHIBIT icon). For example, the event “Application code failure” will be triggered only if both events “Batch cleanup failed” and “No. users > 100k” take place and, in addition, the side condition “Application allocated less than 1 GB RAM” is also true.

FTA provides a number of benefits. The top-down approach means that effort is focused on the most critical events, which are those events closest to the top of the tree (the failure of the entire system is the ultimate critical event). Because the number of top events that are analyzed and the depth to which the analysis is carried out can be controlled, the technique can be viewed as scalable with the size of the system.

FTA can take all kinds of failures into account; this differentiates it from FMECA and RBDs. It can be used as a starting point for developing fault diagnosis procedures and tools. It identifies components and events that are critical from a reliability point of view. FTA allows the evaluation of the impact that design changes will have on the system. It can be used for qualitative as well as quantitative analysis.

FTA has a number of limitations. FTA supports a single event as top event; to analyze other types of failures, additional fault trees must be developed. The level of detail, types of events included, and the organization of the tree can vary significantly from analyst to analyst. Because an FTA does not produce a unique answer, the value of an FTA depends on the skill and experience of the analyst. The accuracy of FTA results depends on data that is often difficult to obtain.

EXAMPLE OF RELIABILITY BLOCK DIAGRAM ANALYSIS

In this section we illustrate the RBD technique by using it to evaluate the availability of a Web-based application supported by a three-tier server configuration.

System overview

The system we examine supports a Web-based application and is built on the rather common three-tier server configuration illustrated in **Figure 3**. We

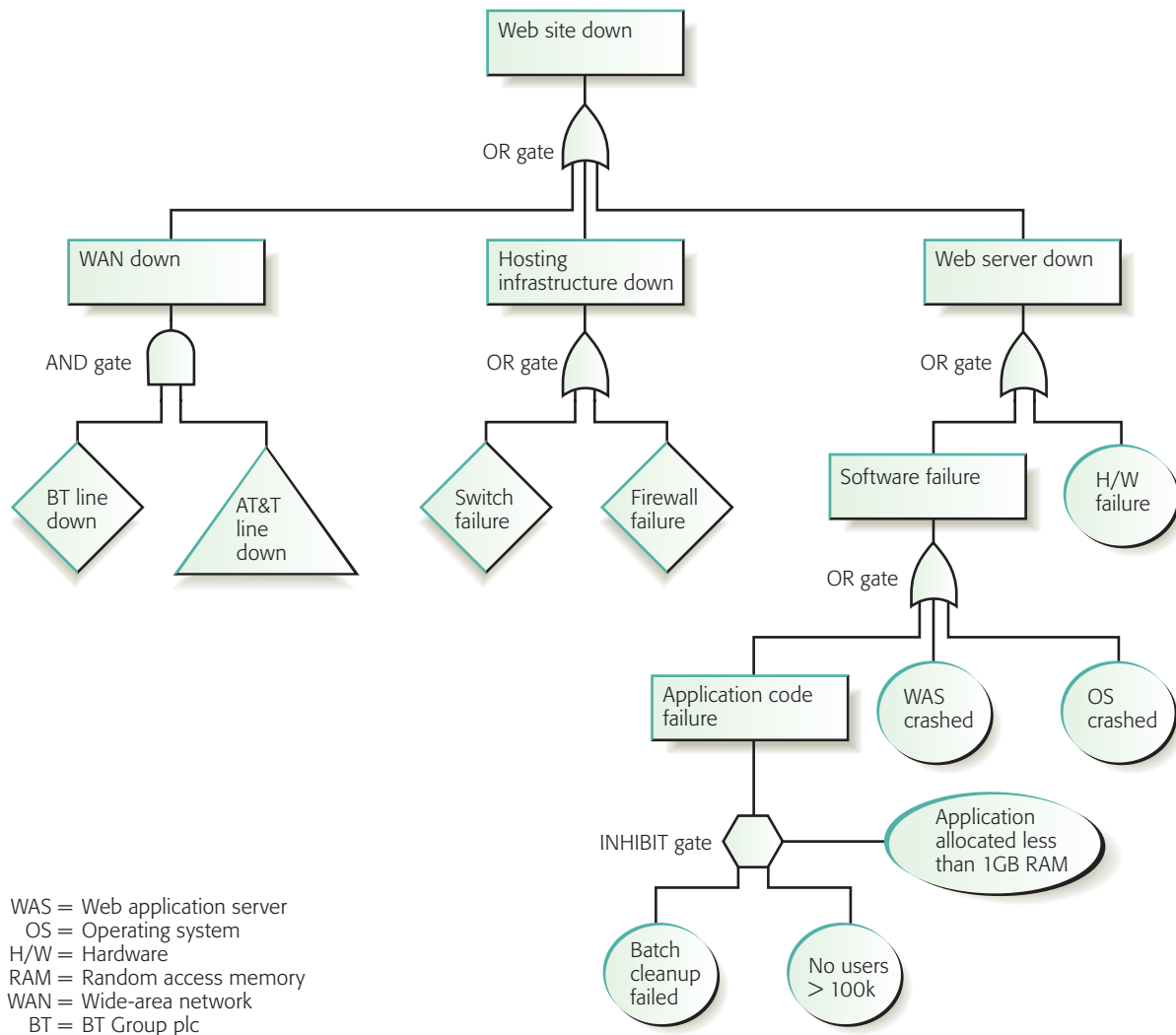


Figure 2
Fault tree for Web-based system

perform an availability analysis on the server configuration using the RBD technique.

As can be seen from the RBD at the bottom of Figure 3, the eight servers are grouped in three tiers. The first tier, which consists of nodes 1 and 2, involves Hypertext Transfer Protocol (HTTP) servers, in the role of load balancers. The second tier, which consists of nodes 3 through 6, involves Web application servers (WASs). The third tier involves nodes 7 and 8 as database (DB) servers. The RBD for the system, depicted at the bottom of the figure, consists of the 10 blocks labeled A through J, where each block corresponds to a subsystem (or tier) of the system.

If we use the letters A through J to also denote the availability of blocks A through J, then the availability of the entire system is given by the product $ABCDEFGHIJ$. Assume that the target availability for the system is 99.99 percent (which can be written as 0.9999, also known as “four nines”). If all tiers have the same availability, then in order to satisfy a target availability of 0.9999, the availability of each subsystem has to be no less than the 10th root of 0.9999, which turns out to be 0.99999 (or five nines). Because the components available today cannot deliver by themselves this level of availability, it is necessary to use redundancy in order to meet the target availability. Notice that, except for the SAN, each tier is made up of multiple parallel nodes.

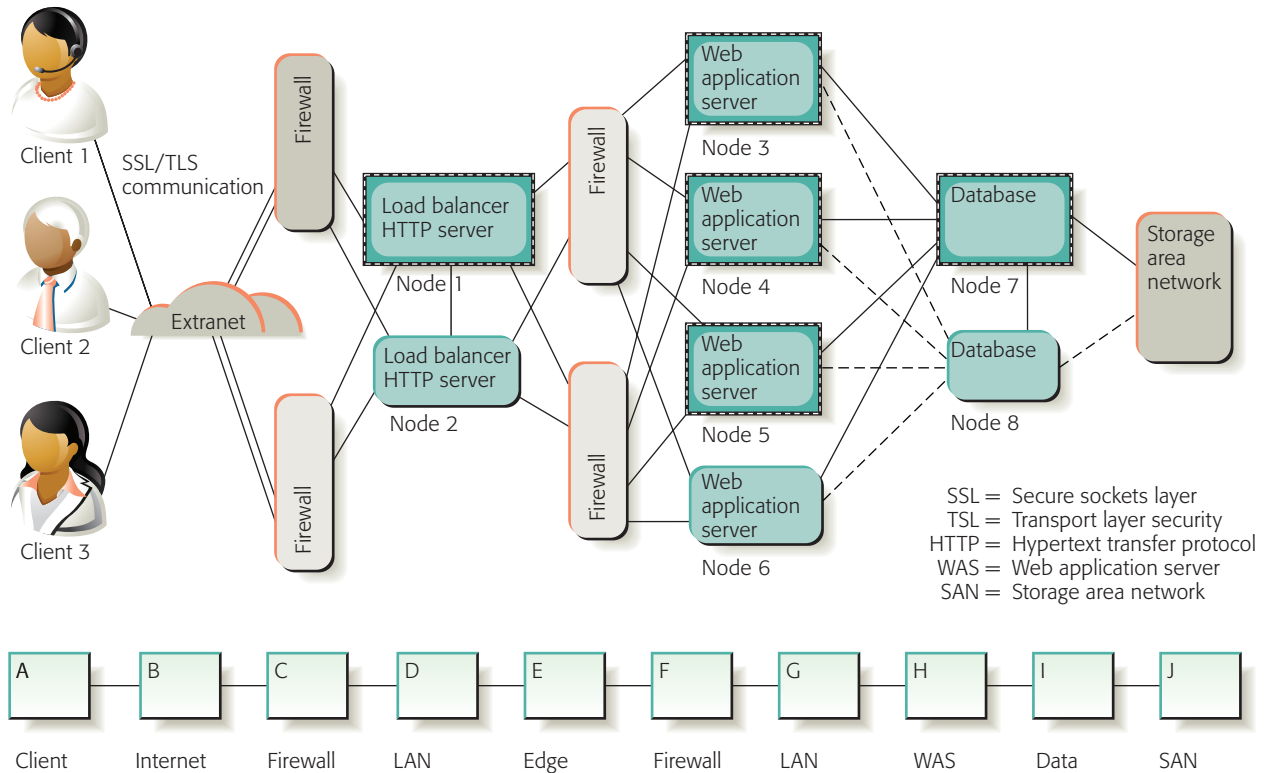


Figure 3
 A three-tier server configuration for a Web-based application and its RBD

We will take a closer look at the WAS tier and the database tier, as examples of stateless and stateful components. We make the following assumptions:

- In the WAS tier, the redundant server supports its share of the workload; this is the way n/N redundancy works. In the database tier the redundant server is standing by.
- Failures do not propagate between the servers of a tier. This assumption is optimistic, particularly when failures are driven by stressful loads when failures naturally propagate between tiers.
- The mean failover time is fast enough that no outage is recorded (i.e., work can be backed out and rerouted quickly).
- The total load in the WAS tier is never larger than three $(N-n)$ servers can handle. Workload growth and sub-linear growth of throughput with utilization (saturation) may not let this hold up. In the database tier, the assumption is that one server handles the load.
- Each server, plus software, delivers 99-percent availability. This assumption is made merely to obtain a number for this example; in a real system,

it can be calculated by developing RDBs for each node.

The WAS tier

The RBD for the WAS server tier is shown in **Figure 4**. The letter H used in this diagram is also the letter associated with the WAS server tier from the system-level RBD in Figure 3. The dashed line in the diagram indicates that three servers are required to handle the entire system load, and thus the fourth server is a redundant resource. For a configuration such as this one, when one out of the four servers is a redundant resource, we refer to it as a 1/4 tier.

The probability that this tier is unavailable is given by the binomial distribution:

$$P = \sum_{k=n+1}^N \binom{N}{k} p^k (1-p)^{N-k} \quad (1)$$

where $n = 1$, $N = 4$, and $p = 0.01$, yielding a 0.0006 probability of unavailability. The availability of the tier is $1 - 0.0006$, or 0.9994. Notice that this falls short of the availability target. The possible reme-

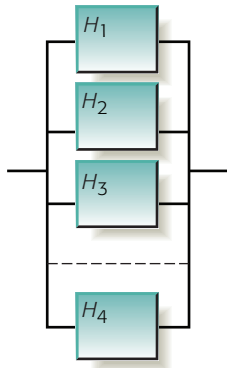


Figure 4
RBD for the WAS tier

dies are: improve the node reliability, add redundancy in the form of an additional node, or make up for the shortfall in other tiers. Otherwise one has to accept the increased risk and lower the system availability target.

Adding a node is often the most prudent solution, because high availability requires redundancy of capacity in addition to redundancy of components. In the analysis above, one of the key assumptions made was that three servers can handle the entire load. There are two factors that can work against this assumption.

The first is growth. As the workload grows, it will consume more of the available capacity. At some point, three servers will no longer be able to handle the load, but four will.

The second factor is more subtle. For capacity planning, although it is usually assumed that the throughput is linearly proportional to the utilization of the server, this is generally not true. Systems often exhibit bottlenecks, which result from increased contention in components such as memory buses, caches, and I/O devices. These cause the throughput-versus-utilization curve to flatten out (i.e., saturation) as the utilization grows beyond a certain point. Consequently, in our system, a single redundant server might be insufficient to make up for the lost capacity of a failed server.

Node availability plays an important role. If, for example, the node availability is 0.999 instead of 0.99, the availability of the solution in Figure 4 is 0.999994, which improves on the target. Because it

is often unreasonable to expect software failure rates to be low enough early in the product life, the best remedy to the situation above is to add a server. If we recalculate the availability for five nodes in total, of which two are redundant, we get an availability of 0.99999. This now meets the availability requirement for the tier. If we are aware that the load saturates at low utilization, it may be prudent to add yet another server. While this results in a 3/6 tier with an availability approaching 1 for the initial workload, this would yield a 2/6 solution with growth. This 2/6 solution has an availability of 0.999985, which approaches the target given the loss of redundancy to workload growth.

It is also important to note that, as we add redundancy to increase availability, we are dropping the utilization of the servers involved. This is often at odds with the desire to drive utilization up on distributed server farms. Finally, as we add additional servers we drive the node failure rate up, thus increasing the need for efficient error handling and increasing the operational costs.

An alternative approach is to use a small number of highly reliable components and virtualization (vertical scaling), instead of a larger number of distributed components (horizontal scaling). This approach has the advantage of sharing hardware capacity between redundant software components and results in a higher utilization of the server hardware. However, the more reliable components are more expensive, which drives up the overall cost.

One way to refine the model is to assume that the unavailability increases due to increased stress when a server fails and the remaining servers have to support a higher workload. This means that the node availability p in Equation (1) is actually a function of k . We can simplify the model by adding enough redundancy so that failures do not drive utilization of the remaining servers into the stress region. The required server capacity in that case is the capacity that supports the workload stress free. Suppose the known onset of stress is 60-percent utilization. In a 1/2 solution ($N=2$, $n=1$) the target utilization of the pair of servers before failure is 30 percent.

The database tier

In database tiers, state cannot be avoided. As a result there must be redundant capacity with access

to the data or a standby replica of the data. A highly reliable and quick mechanism is needed for the failover process. Most server platforms have such failover schemes: IBM High Availability Cluster Multi-Processing (HACMP*) for AIX*, Veritas** Cluster Server from Symantec Corporation, or IBM Sysplex for System z*. All these schemes include mechanisms for switching traffic to the redundant server, preserving the lock state, and rolling back lost work. They all require manual procedures, scripted automation, and function in the middleware to work properly. The automation scripts do not usually cover everything that needs to be done during an outage, which leads to the potential for human error.

The difference in solution availability comes down to the shape of the probability distribution of the lock holding time. It turns out that on failover there is no perceived outage most of the time. However, because there is always a finite probability of a long failover due to long lock holding time in all of these schemes, individual node availability is key to keeping down the probability of a long failover. A tighter distribution around a fast average failover will also drive availability higher. Thus, our database tier 1 is actually made up of two tiers: the operational tier I1 and the independent failover tier I2. The availability of I1 is driven by the node failure rate, as in the WAS tier H, and the failure rate of I2 is driven by the probability of a long failover interval along with the probability that the failover server is down at the time of the failure. Although the availability of I2 is usually high, it is nevertheless less than 1 and thus it lowers somewhat the operational availability of the DB tier.

Assuming a node availability of 0.99, we get an operational sub-tier availability of 0.9998. As in the WAS tier, the availability target is not met. For database servers, however, the situation is a bit more complicated. Database servers are generally more complex and more expensive than those used in the WAS tier. We usually expect higher node availability instead of counting on redundancy in order to achieve high availability. In this case, a node availability of 0.998 will achieve the goal if we assume that the failover sub-tier availability is 0.999999 (six nines).

Naturally, the standby server will drop the utilization of configured hardware by 50 percent in a $1 + N$

configuration. One difference between this and the n/N configuration of the WAS tier is that the redundant server is often kept idle until the failure occurs.

Lessons learned

Performing the RBD analysis above has taught us a number of lessons. The system availability is always lower than the availability of the weakest component or tier of the solution. In other words, tier availability targets are always higher than the end-to-end availability target. As a result, the elimination of tiers can lead to improved availability.

A three-tier server configuration results in an RBD with a higher number of tiers (blocks) when networks and other components are included.

The initial availability target for an RBD tier can be difficult to attain for a specific subsystem. Remedies include shifting the burden to other tiers, improving tier reliability through additional redundancy, or acceptance of reduced availability.

Systems with more than one redundant node are often better solutions from the availability point of view, but they naturally run at a lower utilization. A consequence of the relationship of capacity and availability is that workload growth reduces availability by consuming redundant capacity in n/N configurations.

Node reliability is a key factor in the design for availability. The amount of redundancy required to provide high availability is inverse to the component reliability. Node availability is particularly important in the database tier, where high variance in the failover time affects the probability of a perceived outage during a failover.

CONCLUDING COMMENTS

Availability has long been seen as a hardware issue. In this paper we advocated a business-oriented approach to system availability. This approach calls for availability requirements to be stated in business terms. It also requires an understanding of the cost of unavailability to determine the optimal investment that should be made to provide system availability.

Reliability engineering techniques that have been proved in other industries should be used to model

and analyze the availability characteristics of the overall system design at all stages of the development life cycle, starting during conceptual design; this will allow design changes to be made early, when the cost of making these changes is still low. The output of these techniques should also be used to improve the tools and procedures that will be used to operate the system.

While the adoption of rigorous techniques could increase the cost of analysis and design, this can be offset through the use of design patterns with known availability characteristics. Patterns will improve our ability to develop solutions that meet the availability requirements at a cost that is affordable and can be justified by balancing the additional development cost with the cost of unavailability.

The application of these techniques and patterns will also serve to improve system testing. Testing the availability and reliability of systems is known to be difficult and expensive because of the low probability of failure. It is therefore critical that failure recovery be tested by inducing failures using a risk-based approach.

The increasing use of service-oriented architectures (SOA) presents a number of opportunities and poses new challenges from a system availability perspective. A few of these are outlined below.

SOA complicates static reliability modeling by providing the ability to select and invoke services at run time. The dynamic nature of SOA will therefore place new demands on modeling techniques used to predict reliability and availability. For example, it may be justified to develop stochastic simulation models for mission-critical SOA systems.

SOA also provides the ability to assemble a system from services that have guaranteed quality of service (QoS) attributes that specify reliability characteristics. This provides very loose coupling and could allow the reliability engineer to model system availability using a clearly defined hierarchy of independent models. In practice, services are usually dependent on shared resources and very few have guaranteed availability defined in their QoS specifications. The reality of the underpinning infrastructure needs to be factored into the techniques and models used for availability prediction.

SOA provides the ability to create even more complex systems with a high degree of dynamic configurability. Intuitive approaches cannot be used to predict system availability with a sufficient degree of confidence. We conclude that the use of SOA will further increase the need for a structured approach to reliability engineering and the use of more complex modeling techniques and tools to be implemented throughout the development life cycle.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Symantec Corporation in the United States, other countries, or both.

CITED REFERENCES

1. D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, ACM, New York (1988), pp. 109–116.
2. K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*, Second Edition, John Wiley, New York (2001).
3. A. Goyal and S. Lavenberg, "Modeling and Analysis of Computer System Availability," *IBM Journal of Research and Development* **31**, No. 6, 651–664 (1987).
4. D. Wang and K. S. Trivedi, "Modeling User-Perceived Service Availability," *Proceedings of the International Service Availability Symposium (ISAS)*, Springer, Berlin, pp. 107–122 (2005).
5. M. Kaâniche, K. Kanoun, and M. Rabah, "Multi-level Modeling Approach for the Availability Assessment of e-business Applications," *Software: Practice and Experience* **33**, No. 14, 1323–1341 (2003).
6. *Procedures for Performing a Failure Mode, Effects, and Criticality Analysis*, Department of Defense, Washington, DC, MIL-STD-1629A (1980).
7. R. E. McDermott, R. J. Mikulak, and M. R. Beauregard, *The Basics of FMEA*, Productivity Press, Portland (1996).
8. M. Stamatelatos and W. Vesely, *Fault Tree Handbook with Aerospace Applications*, NASA Office of Safety and Mission Assurance, Washington, DC (2002), www.hq.nasa.gov/office/codeq/doctree/ftthb.pdf.
9. A. Birolini, *Reliability Engineering: Theory and Practice*, Fourth Edition, Springer-Verlag, New York (2004).
10. *Reliability: A Practitioner's Guide*, Intellect UK and Relex Software Corporation (2005), <http://www.intellectuk.org/>.
11. D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft, "Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies," Computer Science Technical Report UCB//

- CSD-02-1175, University of California at Berkeley (March 2002).
12. G. Candea and A. Fox, "Recursive Restartability: Turning the Reboot Sledgehammer into a Scalpel," *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, IEEE, New York (2001), pp. 125–130.
 13. J. Adams, S. Koushik, G. Vasudeva, and G. Galambos, *Patterns for e-business: A Strategy for Reuse*, IBM Press (2001).
 14. D. Powell, "Distributed Fault-Tolerance: Lessons from Delta-4," *IEEE Micro* **14**, No. 1, 36–47 (1994).
 15. J. L. Hellerstein, S. Ma, and C.-S. Perng, "Discovering Actionable Patterns in Event Data," *IBM Systems Journal* **41**, No. 3, 475–493 (2002).
 16. J. V. Carreira, D. Costa, and J. G. Silva, "Fault Injection Spot-checks Computer System Dependability," *IEEE Spectrum*, **36**, No. 8, 50–55 (1999).
 17. J. Durães, M. Vieira, and H. Madeira, "Dependability Benchmarking of Web-Servers," *Proceedings of the 23rd International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2004)*, Lecture Notes in Computer Science 3219, Springer, Berlin (2004), pp. 297–310.
 18. Relex Software Corporation, www.relexsoftware.com.
 19. Isograph Ltd., www.isograph-software.com.
 20. ITEM Software, www.itemsoft.com.
 21. M. Malhotra and K. S. Trivedi, "Power-hierarchy of Dependability-model Types," *IEEE Transactions on Reliability*, **43**, No. 3, 493–502 (1994).

Accepted for publication May 30, 2008.

Published online October 27, 2008.

Dan Bailey

IBM Bedfont, Hursley Park, Hursley SO21 2JN, UK (dan.bailey@uk.ibm.com). Dan Bailey is a certified IT Architect and the IBM U.K. Public Sector Chief Technology Officer, with more than 15 years of experience in the IT industry, taking leading roles in defining the enterprise and solution architectures for clients. Through his work on mission-critical systems, he has collated techniques for designing solutions and led the creation of an IBM method for designing highly available solutions.

Erwin Frank-Schultz

IBM UK Ltd., NHBR-1PH, North Harbour, Portsmouth Hants PO6 3AU, UK (Erwin_FrankSchultz@uk.ibm.com). Erwin is a certified IBM Executive Architect and Fellow of the Institution of Engineering and Technology. He has 20 years of experience in the architecture of IT systems, leading design authorities on large complex programs, and providing high-availability consultancy. He is the leader of the IBM Worldwide High-availability Community of Practice.

Pieter Lindeque

IBM UK Ltd., Meudon House, Meudon Avenue, Farnborough Hants GU14 7NB, UK (LINDEQP@uk.ibm.com). Pieter is an IBM Distinguished Engineer, certified Executive IT Architect, and Fellow of the Royal Academy of Engineering. He has more than two decades of experience in the architecture and delivery of large, complex system integration programs. He is operational lead of the global IBM Asset Architecture Board and an active member of the IBM Academy of Technology.

Joseph L. Temple III

IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, NY 12601 (jliitemp@us.ibm.com). Mr. Temple

is an IBM Distinguished Engineer and Master Inventor. He has worked in many areas of processor development and technical support and currently is working on client-centered engagements for the System z brand. He holds a BSEE from Lafayette College, a graduate certificate in the analysis and simulation of computer systems from Union College, and an MSEE from the University of Vermont. ■