

Modeling Application Availability

J. Singh
Quantitecture, Inc.
j.singh@quantitecture.com

Keywords: Application Availability, Distributed Systems, Virtualization, Software as a Service (SaaS), Business Continuity Planning

Abstract

Managing hundreds of business applications in an enterprise with a quantitative view to their availability is a challenge that requires a combination of quantitative methods and capturing knowledge from across the organization. A unified methodology modeling availability of applications in an enterprise is proposed. The proposal provides technology management of the enterprise with tools to continuously manage the evolution of their technology infrastructure.

1. INTRODUCTION

The concepts behind increasing the availability of systems are hardly new. They started with companies like Tandem and Stratus and have become more prevalent in the decades since. How to design a highly available system is well understood – for one example, see [Schmidt 2006]. The techniques for managing hundreds of business applications in an enterprise with a quantitative view to their availability are less well formalized.

Gartner Group [Plummer et al. 2008] sees three major trends with implications for availability. First, driven by the Going Green imperatives, virtualization technologies permit parts of many different applications to share the same hardware, introducing new single-points-of failure into the corporate infrastructure. Second, Software as a Service (SaaS) extends what used to be a corporate infrastructure into one that spans across corporate lines. Third, as consumers change the way decisions and technology choices are made by IT, the focus of availability will shift from components of the IT infrastructure to business applications.

The mathematics behind the techniques for assessing the availability of systems is well understood for some cases but not for others and we present a unified framework for it.

The notion of a hierarchical network to model a complex set of components is developed. This notion is developed from

the bottom up, introducing the nodes in the model first and building up from there.

2. TERMINOLOGY

Consider a system S . p_S , the availability of S is the probability that S is up and running.

Since p_S is a number that needs to be as close to 100% as possible, like $(1 - 10^{-n})$, counting the nines is a customary way of referring to n as the unit of availability. A 99.999% availability is referred to as *five nines* availability.

Another common way to refer to availability, in increasing order of availability, is as A or AA or AAA. We bring the two terminologies together and refer to five nines availability as 5A, AAA availability as 3A, etc. Integrating the two terminologies gives a quantitative meaning to an AAA rating: 8.76 hours down time per year, or 40 minutes per month.

3. COMPOSITE NODES

Understanding the availability of an application consists of understanding its components and how they combine to deliver the application functionality. Two types of composite nodes are introduced in this section and the techniques for computing their availability developed. These nodes represent the availability of their components and provide a mechanism for aggregating the availability of the components to the application level.

3.1. Simple Aggregate Nodes

Consider a system S consisting of two components, C and D , both being required for S to function (and be considered available). If the two components are independent of each other, $q_C = (1 - p_C)$ and $q_D = (1 - p_D)$, then

$$\begin{aligned} p_S &= p_C \times p_D \\ &= (1 - q_C - q_D + q_C \times q_D) \end{aligned}$$

We get the result

$$q_S = q_C + q_D - q_C \times q_D$$

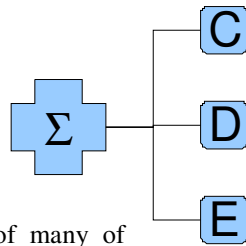
If q_C and q_D are small numbers, their product can be ignored and we get the result

$$q_S = q_C + q_D$$

This formula is the basis of computing the availability of the system. Expressing it in number of As is useful since it will be used often

$$n_s = -\log_{10}(q_s) = -\log_{10}(10^{-n_C} + 10^{-n_D})$$

This result can be extended to a system with any number of elements.



In modeling a system composed of many of components, we characterize each component in terms of its availability. A *Simple Aggregate Node* is defined to represent the aggregate according to the above formulas.

3.1.1. Estimating Availability of Components

The formulation above expresses n_s in terms of n_C and n_D . Where do n_C and n_D come from? For component C,

$$p_C = \frac{MTBF}{MTBF + MTTR}$$

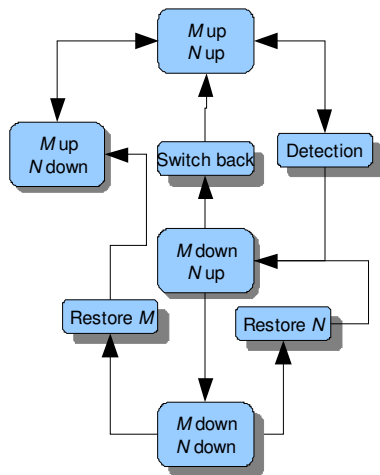
MTBF figures can be obtained from vendors. MTTR is driven by the Service Level Agreement (SLA) of the data center.

3.2. Fail-over Nodes

Systems concerned with high availability often have fail-over components incorporated into the design. In this section, we develop the model for such a system.

We denote the failing component as M and the switch-to component as N . The fail-over process encompasses a number of different scenarios depending on how it is configured. One such scenario is shown here; the states where the system is unavailable are shown with a shadow. The “ M down N up” and “ M up N down” states are shown with a smaller shadow because the risk in these states is higher.

Other variants of the state diagram:



- Switch-back not required. Once M has failed over to N , it becomes the primary. When M is repaired, it becomes the secondary.
- The fail-over components are arranged in a voting cluster, as is the case with RAID disks.
- Etc.

All of the possible state diagrams can be characterized using three parameters of the Service Level Agreement:

1. Availability of the fail-over process p_{FO} (and $q_{FO} = 1 - p_{FO}$). q_{FO} can be anything from sub-second/yr (as in the case of mirrored disks) to several hours/yr (when it includes conference calls with key players to decide whether a fail-over should be initiated and possibly travel to an access point to do so *plus* the time it takes to complete the fail-over: applying log files to shadow databases, switching DNS entries, changing configuration files, etc.).
2. Availability of switch-back p_{SB} (and $q_{SB} = 1 - p_{SB}$). q_{SB} is the time the system is expected to spend per year in the switch back state. It includes the time when the primary node is under repair *plus* the time after repair waiting for the right opportunity to switch the configuration back.
3. Availability of the Restore process p_R (and $q_R = 1 - p_R$). q_R is the time per year expected to be spend rebuilding the system. This may involve requisitioning database archives from off-site locations, building new servers and configuring them, and anything else required to bring the system on-line again.

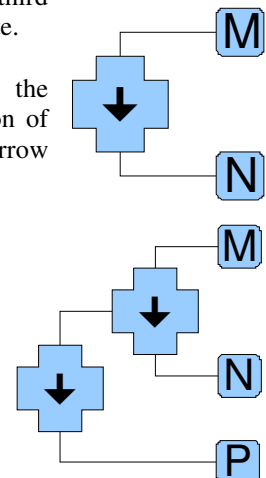
The technique for modeling the availability of a system that has a fail over mechanism built into it is to compute the availability of this composite element and use it in assessing the availability of the overall system.

$$q_s = q_{FO} + q_M q_N (1 + q_R) + q_M q_{SB}$$

In deriving this expression, we assume that the times for restoring M and N are the same. The first term represents the detection state, the second term represents the bottom three shadowed states and the third term represents the switch back state.

A *Fail-Over Node* is used in the network to represent this collection of components. The direction of the arrow indicates the direction of fail-over.

More complex fail-over scenarios can be represented by combining such nodes. For example, in a situation where M fails-over to N



which fails-over to P can be represented as shown.

3.2.1. Estimating Fail-over Parameters

The formulation above expresses q_S in terms of q_{FO} , q_{SB} and q_R . Where do q_{FO} , q_{SB} and q_R come from? They are driven by the Service Level Agreement (SLA) of the data center or the support organization. In some instances, the type of support needed may be too new to estimate the parameters. If this is the case, a walk-through of the process provides an initial guess that can be refined as experience builds up.

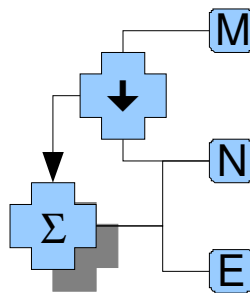
3.3. Real World Usage

In this section, we consider some examples from the real world to see if the nodes defined thus far are sufficient for modeling real world systems. Some are straightforward and fit into the structure already established. Others (just 1) require slight modifications to nodes already defined.

A trading system that has to be highly available while the market is open but the requirements are less stringent at other times. This scenario is the same as other scenarios, only that the service level agreement (SLA) applies just to market hours.

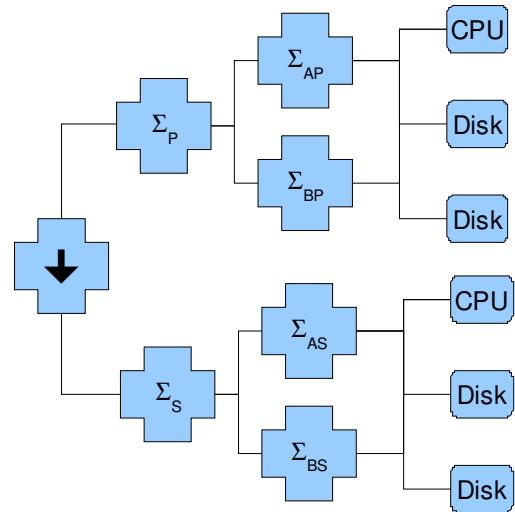
How should multiple tiers of functions, some of which need to be more available than others, be modeled? For example when taking trading orders is very important but the availability requirements for a reporting function are less stringent. This scenario consists of two calculations, one for the higher availability functions and the other for the lower availability functions. Thus, two aggregate nodes are required to model the SLA. A new type of node is not required.

How about a situation with shedding load in the event of a disaster? The primary SLA is maintained by failing over but a secondary SLA is compromised, perhaps because the fail-over configuration has just enough capacity to keep the core business running. The primary SLA is a fail-over node. The secondary SLA, shown with a shadow, is a standard Aggregate node with one difference: the fail-over state of the primary component is one of its components. This is not a new type of node; it recognizes that the fail-over node has another output which indicates it being in a fail-over state.



3.3.1. Real World Usage: Virtualization

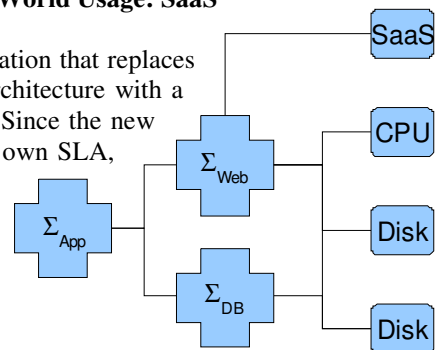
Consider an enterprise with 2 applications, A and B, one built on a Apache-MySQL stack and the other on a Microsoft stack. Each application has 2 servers for redundancy, each server hosting a web server as well as a database server. What does the configuration look like once the servers of the two applications are virtualized into just 2 servers?



Here, Σ_{AP} represents the primary web server and database server for application A. The diagram is slightly abridged: aggregation nodes representing the Apache server and MySQL server, components that would represent their availability, have been omitted here but would need to be represented in real life.

3.3.2. Real World Usage: SaaS

Consider an application that replaces a function in its architecture with a SaaS component. Since the new component has its own SLA, it can be represented similar to other base components.



4. SYSTEM REQUIREMENTS

In this section, we present key characteristics of a system that might be used for modeling the network of nodes described above and how it might be used.

In large organizations, all components are not controlled by the application owner. The decisions about where a particular part of the architecture is hosted are often decoupled from the technical design. For example, a database group may be empowered to make decisions on how the database is configured independently from how the authentication solution is constructed. It is important, therefore, to allow for *distributed data entry* – each group limited to doing the entry for nodes under its control. The policy for determining who can edit what part of the network must be independently configurable.

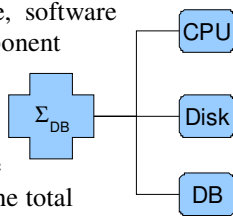
The system evolves over time and it is important to be able to model the impact of making a change before making it. In other words, we envision a need to have the “current” configuration and also to have future “proposed” configurations in the same model so this impact can be analyzed.

Finally, the modeling system has to provide a set of reports to its users to help them assess the overall availability of the system and provide a view into the “weak links” in system availability.

5. USAGE RESULTS

The techniques proposed in this paper were used to model the availability of an application consisting of a financial portfolio management application. The application framework was distributed – rich client software running at the users desktop connected over the internet to web services which retrieved data from a set of databases. Some observations from the exercise:

1. The model started with hardware components at the base and built software and process components at higher layers.
2. In addition to the hardware, software components required a component to represent the state of the software, as shown. The DB node represents failures due to software failures and Σ_{DB} represents the total database.
3. At higher levels in the model, specific failure modes of the application became more important. For example, the application could miss its availability targets because required data feeds were late or because there were certain types of data elements that caused the application run a long time and be unavailable for other tasks. Components representing such “data quality” elements were not part of the original model but were easily added.



6. SUMMARY

We have developed the mathematics for calculating the availability of systems. The mathematics is quite simple; the key to its usage is building a network representing the system.

The collection of the data on which to base the calculations can be a challenge. A system for collection of this data has to allow entry by members of a large organization, and consolidate that information into a query and reporting tool. The requirements for such a system have been identified. An early implementation is available. We are seeking suitable partners to test the proposed methodology.

References

Plummer, Daryl C.; Smulders, Charles; Fiering, Leslie; Natis, Yefim V.; Mingay, Simon; Driver, Mark; Fenn, Jackie; McLellan, Laura and Wilson, Debbie. 2008. “Gartner's Top Predictions for IT Organizations and Users, 2008 and Beyond: Going Green and Self-Healing” ID Number: G00154035.

Schmidt, Klaus. 2006. High Availability and Disaster Recovery: Concepts, Design, Implementation. Springer-Verlag, New York, Berlin & Heidelberg.

Biography

Dr. Singh is the principal at Quantitecture (<http://www.quantitecture.com>), a consulting practice on performance, scalability and availability.

Dr. Singh is a former Director of Software Development at Fidelity Investments, where he served in line and staff roles over a 14 year career. During that time, he had a substantive role in the performance of 4 major projects: two with 7-figure budgets and two with 8-figure budgets.

Previously, he held line and staff positions for 11 years at Computervision, a CAD/CAM company. There he managed a system performance group for 2 years and went on to invent new products that enjoyed considerable market success.

In addition to his consulting practice, he writes on System performance, scalability and availability. He received his PhD in Electrical Engineering from Syracuse University.