A Hierarchical Modeling of Availability
in Distributed Systems

Salim Hariri
Syracuse University
ECE Department, Syracuse, New York 13244

Hasan B. Mutlu
AT&T Bell Laboratories
Naperville, Illinois 60566

ABSTRACT

Distributed computing systems are attractive due to the increase in availability, throughput, and resource sharing. Modeling and evaluation of such computing systems is an important step in the design process of distributed systems. In this paper, we propose a two-level hierarchical model to analyze the availability of distributed systems as perceived by their users. At the higher level (user level), the availability of the tasks (processes) is analyzed in terms of the availability of the system components. At the lower level (component level), detailed Markov models are developed to analyze the component availabilities. These models take into account the hardware/software failures, congestion and collisions in communication links, allocation of resources, and the redundancy level. A systematic approach is developed to apply the hierarchical model to evaluate the availability of computing systems and communication links. We also present the availability analysis of some of the services provided by the Unified Workstation Environment (UWE) currently being implemented at AT&T Bell Laboratories.

1. Introduction

Advances in computer networking and very large scale integrated (VLSI) devices have led to an increased interest in distributed computing systems. Distributed systems consist of several computers (could be a workstation, a mainframe, or a supercomputer) that interact and cooperate by a message passing. These systems are perceived by the users as logical time-shared computing systems and provide an increase in throughput, reliability, fault-tolerance, resource sharing, extensibility, and a cost-effective computing environment [1]. Designing and engineering such systems are difficult and more complex than the ones experienced in centralized computing systems because of the usage of heterogeneous dispersed components. Consequently, performance metrics such as reliability, availability, throughput, delay, and error rate are important parameters to evaluate and optimally engineer the design of cost-effective distributed systems.

This paper addresses the reliability/availability issues in a distributed computing environment. Although we present a hierarchical model for availability analysis, the same model can be used to analyze reliability in such systems. Availability measures the probability that the system is functioning properly at a given time $t$, whereas the system could have failed and repaired several times before $t$. Modeling availability of distributed systems is more complex than modeling uniprocessor systems. In such systems, several computers cooperate on processing a service submitted by a user and the availability of this service depends on several factors (hardware and software failure rates, locations of these computers with respect to each other and the user requesting the service). Our approach for evaluating the availability of distributed systems is based on a hierarchical modeling scheme. At the lowest level (component level), we provide models to evaluate the availabilities of computing systems and communication links. At the higher level (system level), we present models to evaluate process (program) availability with respect to one site (user), expected process availability, and the availability of a set of processes (programs).

2. Modeling Availability at the System Level

2.1 Process Availability

Process availability can be defined as the probability that the process is functioning properly at a given time. Modeling process availability in a distributed computing environment is complicated because process execution involves the cooperation and interaction of several types of resources. For example, Figure 1 shows an example of a distributed system in which a process, say $P$, would require accessing three processes running on three separate computers.
Such a process could be an application program to transfer a fund from one account to another and interacts with three resources (software modules, computers, devices, etc.): one software module (request handler) to manage and supervise the processing of user requests, second process (program such as fund transfer) is needed to transfer fund from one account to another in an atomic manner, and the third resource is a database to store user accounts. To increase the availability of executing processes in this environment, double redundancy is introduced as shown in Figure 1. Process $P$ that is invoked from site $e_1$ can be processed successfully if the computers $c_1$, $c_2$, and $c_3$, the communication links $t_1$ and $t_2$, and software modules used by $P$ are all running properly. Furthermore, if computer $c_3$ or link $t_2$ is down, it is still possible to run $P$ using the copy of the fund transfer program available at computer $c_4$.

![Figure 1. Process Execution in a Distributed Computing Environment.](image)

Some applications with real-time constraints or excessive delays could prevent a process from achieving its goals properly. Hence, highly congested links in point-to-point networks and high collision rates in broadcast-based networks should be considered in modeling the link availability. Consequently, process availability depends on the 1) availabilities of computers; 2) availabilities of software modules; 3) availabilities of communications links; 4) degree of congestion and collision (delays); 5) allocation of resources; and 6) redundancy level. In our proposed hierarchical model, the computer availability module takes into consideration the failure rates caused by both hardware and software. Furthermore, the link availability model considers failures caused by hardware as well as those caused by excessive delays (item 4 mentioned above). The main idea behind lumping items 1 and 2, and 3 and 4 is to capture the effects of the parameters mentioned above by a tree (referred to as process spanning tree PST). The allocation of resources and their redundancy level are modeled by the existence of several PSTs that can be used to run process $P$ with respect to user 1 as shown in Figure 2. Thus, process $P$ is available if at least one PST is in operational state (all nodes and links of that tree are functioning properly). Hence, the availability of process $P$ can be evaluated by determining the probability that at least one PST is operational. That is,

$$A(P, \text{user 1}) = P_i \left( \text{at least one } \text{PST}_i \text{ is up} \right) = P_i \left( \bigcup_{i=1}^{i=3} \text{PST}_i \right)$$

where $A(P, \text{user 1})$ denotes the availability of process $P$ with respect to user 1.

![Figure 2. Process Spanning Trees.](image)

One approach to evaluate the probability described by the above equation is to modify the set of trees (PSTs) into another equivalent set of mutually exclusive PSTs. For example, the events corresponding to the union of the three PSTs associated with process $P$ can be decomposed into the following three events: first event occurs when PST$_1$ is up, second event occurs when PST$_2$ is up and PST$_1$ is down, and third event occurs when PST$_3$ is up and both PST$_1$ and PST$_2$ are down. Hence, the availability of process $P$ can be written as

$$A(P, \text{user 1}) = P_i (\text{PST}_1) + P_i (\text{PST}_2 \cap \text{PST}_1) + P_i (\text{PST}_3 \cap \text{PST}_1 \cap \text{PST}_2)$$

where PST$_i$ denotes that PST$_i$ is in the failure state. The term corresponding to the first event is $A_i A_2 A_3 A_1 A_3$, where $A_i$ denotes the availability of
component $i$. In the second event, $PST_2$ is up which results in having all the components of $PST_2$ being operational, and $PST_1$ is down which occurs when either $t_2$ or $c_3$ has failed. Hence, the term corresponding to the second event is $A_cA_tA_cA_tA_t(1-A_tA_c)$. In a similar manner, the term corresponding to the third event is $A_cA_cA_cA_cA_c(1-A_cA_t)$. The process availability with respect to a user at site $c_1$ is the sum of the terms associated with these three events. That is,

$$A(P, user_1) = A_cA_tA_cA_tA_t(1-A_tA_c) + A_cA_tA_cA_tA_t(1-A_cA_t)$$

The process availability evaluated above was with respect to users at site $c_1$. Similar analysis can be applied to evaluate process availability with respect to users at site $c_2$. A detailed analysis of evaluation algorithms to evaluate similar reliability/availability measures can be found in [2, 3]. These algorithms can be directly adopted to evaluate the measures of the hierarchical availability model presented in this paper.

### 3. Modeling Availability at the Component Level

The availability metrics, developed in the previous section, model the availability perceived by user’s processes and are described in terms of node and link availabilities. In this section, we present models to evaluate the component availabilities used in evaluating availability measures at the system level. At the component level, detailed models can be used to analyze availability. In what follows, we apply Markovian models to evaluate component availabilities. Other methods such as SHARPE [4], Extended Petri Nets [5], Simulation [6] can also be used at this level.

#### 3.1 Point-to-Point Communication Link Availability Model

Packet-switched data networks are constructed from switching nodes interconnected by trunks. At a switching node, the data is stored in one or more first-come-first-serve (FCFS) queues from which they are eventually taken for transmission over the trunk. The existence of a physical communication path which functions correctly and in addition a non-congested communication path are of interest. The congestion appears as an equivalent loss of availability of data transmission services offered to users, and consequently it must be considered in assessing the network performance.

##### 3.1.1 Notation and Definitions

- $\lambda_p$ physical failure rate.
- $\lambda_c$ congestion failure rate.
- $\lambda_{cp}$ failure rate that takes the system from the congested state to the physical failure state.
- $\mu_p(\mu_c)$ physical (congestion) failure repair rate.

#### 3.1.2 Congestion

When a packet is transmitted through the packet-switched network, it can be delayed at intermediate nodes or trunks along its route, as well as at the final destination node due to congestion. The following indicates congestion at the node or trunk for Datakit® Packet-Switched Data Network: Too many requests for virtual circuits; Controller is busy; Trunk is busy and No free buffers in the node. At each node, there is a queue of packets for each outgoing channel. If the rate at which packets arrive and queue up exceeds the rate at which packets are transmitted, the queue size grows without bound and the delay experienced by a packet becomes infinite. The growth of the queue lengths at various nodes consequently causes low throughput. Ideally, the congestion control techniques should limit queue lengths at the nodes, avoiding the collapse of traffic. Thus, the model should be able to capture the impairment of the nodes and the trunks due to congestion and reflect it in the availability measurement.

##### 3.1.3 Trunk Availability Model Construction

The availability $A(t)$ is the probability that at the time $t$, the component is in the operational state and is not congested. In other words,

$$A(t) = P_c(\text{physically correct operation and not congested})$$

A three-state Markovian model is developed to evaluate the node and the trunk availability metric. The states of the component are as follows:

- **state 0** — correct operation state. The state in which the trunk physically operates correctly and is not congested.
- **state 1** — physical failure state. The state in which the link (trunk) failure occurs due to a hardware or software failure.
- **state 2** — congestion state. The state of physically correct operation, but the component is unable to provide data transmission services due to congestion.

With these states, the component transition diagram is represented by a Markov model as shown in Figure 3.
The transition from state 0 to state 1 is caused by physical failure (hardware/software), while the transition from state 1 to state 0 occurs after repairing the component. The transition from state 0 to state 2 is caused by the impairment of the component due to congestion, while the transition from state 2 to state 0 is caused by recovering from the congestion state. The occurrence of physical failure in state 2 will cause the system to go to state 1. The transition diagram covers all possible states and transitions that a component is likely to experience. Note that the transition from state 1 to 2 does not appear on the diagram since the removal of the detected faults in state 1 will not make the system enter a congestion state. Although one could think of this transition possibility as a valid point to be considered, in reality, there will always be a brief moment that the transition will first go to state 0 then to state 2.

In practice, the probability that a continuous-time Markov chain will be in state 1 at time $i$ is evaluated by a limiting value which is independent of the initial state. Hence, if we solve the system for its steady-state (i.e., if we let $i$ approach $\infty$), the following time-independent component availability metric is obtained:

$$A = \frac{\mu_p (\mu_e + \lambda_{sp})}{(\mu_e + \lambda_{sp}) (\mu_p + \lambda_p) - \lambda_e (\lambda_p - \lambda_{sp})}.$$  (3.2)

Notice that as $\lambda_p = \lambda_{sp}$ or $\lambda_e = 0$, equation (3.2) is reduced to

$$A = A_A A_C \text{ (series system)}$$  (3.3)

$$A_C = \frac{\mu_p}{\mu_e + \lambda_{sp} + \lambda_e}.$$  (3.4)

From a practical point of view, $A_A$ and $A_C$ could be interpreted as the availability of component $i$ due to physical failures (e.g., hardware or software) and congestion respectively. In reality, it is possible to consider that $\mu_e >> \lambda_e$, $\mu_p >> \lambda_p$ and $\lambda_p > \lambda_{sp}$. Then, equation (3.2) is again reduced to

$$A = A_A A_C \frac{\mu_p}{(\mu_p + \lambda_p) (\mu_e + \lambda_{sp} + \lambda_e)}.$$  (3.5)

It is interesting to note that as we omit the failures due to the congestion (i.e., $\lambda_p$ and $\lambda_{sp}$), equation (3.2) and (3.5) are reduced to a two-state Markovian model (i.e., equation (3.3)).

### 3.2 Host Availability Model

#### 3.2.1 Notation and Definitions

$\lambda_h$  hardware failure rate from the healthy state.

$\lambda_{sf}$ (permanent/transient) software failure rate from the healthy state.

$\lambda_{pf}$  permanent software failure rate from the transient software failure state.

$\mu_h$  hardware failure repair rate from the hardware failure state.

$\mu_{sf}$  software failure repair rate from the permanent software failure state.

### 3.2.2 Design of the Model

The host availability depends not only on hardware failures, but also on the failures caused by software. Consequently, the availability $A(i)$ of a host is the probability that at time $i$, it is in a state of physically correct operation and does not have any software problems. In other words,

$$A(i) = P_s \{ \text{no hardware and software failures} \}.$$  (3.6)

A 4-state Markovian model is developed to evaluate the host availability metric. The states of the component are as follows:

1. **state 0** — correct operation state. The state in which the host operates correctly and has no software failure.
2. **state 1** — hardware failure state. The state in which the host failure occurs due to a hardware or environmental problem. Typical environmental problems are power outages or errors caused by improper operation of the component.
3. **state 2** — transient software failure state. The state in which the host failure occurs due to a transient software failure.
4. **state 3** — permanent software failure state. The state in which the host failure occurs due to a permanent software failure.

![Transition Diagram for the 4-State Markovian Model](image_url)

Figure 4 shows a 4-state Markov model that takes into account both software and hardware failures. The transition from state 0 to state 1 is caused by a hardware or environmental problem, while the transition from state 1 to state 0 is caused by removing the failure. Likewise the transition from state 0 to state 2 is caused by a transient software failure, while the transition from state 2 to state 0 occurs after recovering from the transient software...
failure. The occurrence of permanent software failure in state 2 will cause the system to go to state 3. Partial restoration of system software can lead to a transition to state 2 from which a fast recovery can move the system back to its operational state. Finally, the transition from state 0 to state 3 is caused by a permanent software failure, whereas the transition from state 3 to state 0 takes place by removing the permanent software failure which requires complete reset and reload of the host’s system software.

The transition diagram covers all possible states and transitions that a component is likely to experience. Note that the transition from state 1 to 3 does not appear on the diagram since the removal of the detected faults in state 1 will not make the system enter a software failure state.

By analyzing the system for the steady-state availability, when \( i = \infty \), the following host availability metric is obtained:

\[
A = \frac{\lambda_1 \lambda_2 \mu_1 \mu_2 + \mu_1 \lambda_2 \mu_2 + \mu_2 \lambda_1 \mu_1}{(\lambda_1 + \lambda_2)} \quad (3.7)
\]

It is interesting to note that as we omit the failures due to the software (i.e., \( \lambda_1 \) and \( \lambda_2 \)), equation (3.7) is reduced to a two-state Markovian model. That is,

\[
A = \frac{\mu_1 + \lambda_2}{\mu_1 + \lambda_2} = \text{MTBFH} + \text{MTTRH} \quad (3.8)
\]

The above models are the basis to estimate the availabilities for all components in a distributed computing environment. If one wants to model the availability of a component caused by only hardware failure, equation (3.8) is used. For example, the availability of a storage component in a distributed environment can be modeled using equation (3.8). A host such as a UNIX-based system can be modeled using equation (3.7) to account for hardware and software failures.

4. A Two-Step Approach for Availability Analysis in Distributed Systems

The objective of the approach to be presented in this section is to analyze the availability of the services and tasks provided by a distributed computing environment with respect to the user site. To reduce the complexity of this analysis, we develop a two-step procedure. In the first step, we group the resources of a given distributed system according to their access requirements. The purpose of the classification of resources is to be able to carry the analysis on small sets of resources instead of considering all resources. In the second step, we apply the models and techniques discussed in the previous sections to evaluate the availability metrics of the given distributed system.

4.1 Classification of Distributed System Resources

The type of activities and processes in distributed systems will be outlined by analyzing the activities of a general purpose distributed system currently being developed at AT&T Bell Laboratories. The purpose of the UWE project is to investigate seamless distributed computing environments. In this environment, a user would have a single account with a single physical login directory across all machines, called community. Also, the user view the system as one logical machine in which all its resources are accessible to the user regardless of his/her physical location. Users are identified within the community with user_id and community_name/user_id to those outside the community. The Remote File Sharing (RFS) and Remote Execution (REXEC) facilities have provided the capability of sharing file resources over the network and executing arbitrary commands on any machine, respectively. For example, for each resource to be shared, a machine is selected to contain the resource. Then, this machine advertises the resource to the rest of the machines within the community.

Figure 5 shows one implementation of the UWE paradigm which consists of 6 3B2s: 2 310s with 4 Megabytes of memory each, 2 400s with 4 Megabytes each, and 2 600s with 8 Megabytes of memory. The communication network implements both Datanet and Ethernet. Table 1 shows the allocation of the resources...
over the workstations in UWE. The services and the tasks to be provided and executed by the UWE are outlined in the next subsections.

4.1.1 Login Process

The purpose of the login server is to splice users to the least loaded workstation when they request login to the environment. This is done through a process, called monitor process, which collects load information from the machine on which it is running. The set of core machines, which refers to hanar, hakar, kavar, and hakar, form a ring. The load information obtained by the monitor processes are circulated among the core machines such that each one knows the load status of other workstations. The login server reads the data from the monitor process and uses it to determine where the user should be logged in. The user is spliced to the least loaded machine, hanar is the primary login server while hakar is the backup if hanar is in a failure state. This process is aimed at serving the users who access the community through terminals. The users who own workstations will be logged into their stations.

4.1.2 Command Execution

The commands that are frequently used can be divided into four groups: 1) Simple: ls, date, mv, rm, cd, pr, cat; 2) Complex: find, cpio, nroff, spell, cp, cc; 3) Editor commands and, 4) Miscellaneous: rmail, mail, mailx, post. The complexity of these commands depends also on the number and type of parameters used in these commands. Also, their execution times vary substantially depending on whether or not the commands involve local or remote access to their needed resources. For example, read and send mail involves remote access in the current implementation of the UWE. When a user reads his/her mail, the mail command accesses the user's mailbox in /usr/mail. This directory is owned by hanar and remotely mounted by all other machines in the community. This allows the users to see the same mail regardless of which machine they are on. However, to send mail, the complexity of the command varies whether it is initiated from a core machine or not. If the send mail is issued from a core machine, the UUCP command will queue up the job in /usr/spool/uucp directory. This directory owned by hanar and mounted by all the other core machines via RFS. The other scenario is for workstations outside the core set which can not mount /usr/spool/uucp via RFS. They send files to be mailed to the hanar using UUCP and then hanar sends these files on their behalf in a transparent manner.

4.1.3 Resource Sets

The processes and commands that are outlined in the previous discussion depend on their executions on the services provided by a set of main resources. Analyzing the availability as well as the performance of a distributed system can be described in terms of the availability or performance of these resources. For example, a print command executed in UWE involves the use of two resources: the user_file system and the print_server. First, the _user_file_to_be_printed_is copied so it can be processed locally at the machine which has originated the command by a distributed printer spooler (UNISON). Once the file has been processed by UNISON and ready to be printed, the print_server is called to format and then route that file to the required printer. Also, the print_server sends the user information about the status of the file to be printed. Hence, the print command can be executed successfully if these two resources are accessible and their associated processes can run without malfunctioning until completion. In other words, the availability of the print command can be described in terms of the availabilities of the two resources involved in that command. That is,

\[ A(\text{print}) = A(\text{user files}) \cdot A(\text{print server}) \]

To serve the users who access the community through terminals. The users who own workstations will be logged into their stations.

<table>
<thead>
<tr>
<th>RESOURCE SET (RS)</th>
<th>HOST(S)</th>
<th>RESOURCE NAMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS_1</td>
<td>hanar</td>
<td>mail, uucp, uucp, dmd, exptools, printer_server</td>
</tr>
<tr>
<td>RS_2</td>
<td>hakar</td>
<td>man, wwb, news, rfsname, 630</td>
</tr>
<tr>
<td>RS_3</td>
<td>hanar, hakar</td>
<td>rfs_name_server, login_server,</td>
</tr>
<tr>
<td>RS_4</td>
<td>hanar, hakar</td>
<td>user_files</td>
</tr>
<tr>
<td>RS_5</td>
<td>remote hosts</td>
<td>post database, linear, md tools</td>
</tr>
</tbody>
</table>

The main resources that are used to execute the processes and services provided by UWE as well as their locations are shown in Table 1. Since the software failures are incorporated in the computer availability, then the
availability of a resource running on a host, say $x_j$, with respect to a user at site, say $x_i$, can be measured by the probability of accessing host $x_j$ from host $x_i$ successfully. This means that the availability of the resources of a distributed system with respect to site $x_j$ can be described in terms of the probability of accessing the hosts running these resources from site $x_j$ without any failures. Consequently, this can be used to reduce the analysis to a few Resource Sets (RS) such that the resources in each set have the same access requirement. For example, the resources uucppub, uucp, print_server (printer), and mail are stored at handar and therefore belong to one set, say RS1. If all the resources shown in Table 1 are grouped into sets based on their access requirements, the analysis can be reduced to studying the availability of accessing the hosts of the five sets shown in Table 2. Although RS3 and RS4 (shown in Table 2) have the same host set (handar and hakar), we did not combine their resources in one set and that can be explained in the following. In RS3, the primary login-server runs handar while its back-up copy runs on hakar when handar is not available. This access requirement is different from that of the resources of RS4. In RS4, the user files are partitioned evenly over two computers (users 2, 3, and 5 are allocated to handar while users 6, 7, and 8 are allocated to hakar) as shown in Table 1. If handar is not available, users 2, 3, and 5 cannot access their files, whereas users 6, 7, and 8 can access their files stored at hakar.

In what follows, we will present an approach to evaluate the availability of accessing a resource set with respect to any site in the distributed system. Also, if the probability of using the resources in each set is known, we can evaluate the system-level availability as described in Section 2.

4.2 Evaluating UWE's Availability Metrics

To evaluate the availability of a resource with respect to one site, say oacak workstation, the set of process spanning trees (PSTS) should be determined. Then, the availability of having at least one PST operational can be evaluated using the approach presented in Sections 2 and 3. In what follows, we will determine the availability of three resources such as print_server, user_files, and a print process with and without redundant resources with respect to a user located at oacak workstation.

4.2.1 Case 1: Without Redundant Resources

4.2.1.1 Access Availability of the print_server

The PSTs that connect a user located at oacak host to the print_server which is allocated at handar are as follows:

$X_{oacak}X_{handar}X_{oacak}$

$X_{oacak}X_{hakar}X_{handar}X_{oacak}$

The availability of the print_server service can be evaluated in a similar method to that described in Section 2 and is shown below as,

$A(print\_server, oacak) = A_{oacak}A_{handar}A_{oacak} + A_{handar}A_{oacak}A_{hakar}A_{handar}A_{oacak}A_{hakar}$

where $A_i$ denotes the unavailability of component $i$ which is equal to $(1 - A_i)$.

The availability of any resource in RS1 with respect to site oacak is equal to the availability of the print_server because they have the same set of PSTs. Namely,

$A(print\_server, oacak) = A(mail, oacak) = A(uucppub, oacak) = A(uucp, oacak).$

4.2.1.2 Access Availability of the user files

In the current implementation of UWE, all user files are evenly distributed across two hosts: handar and hakar and thus the probability of storing the files of a user at one of these two computer is 0.5. Consequently, the PSTs must be determined for both scenarios (user files are stored at either handar or at hakar). The availability of accessing a file is:

$A(user\_files, oacak) = 0.5 (A_{handar}A_{hakar} + A_{hakar}A_{handar}) + 0.5 (A_{oacak}A_{hakar} + A_{hakar}A_{oacak}).$

4.2.1.3 Availability of the print process

In some environments, a distributed system would be designed to perform certain important processes and thus the services provided by these processes are essential and their availabilities must be evaluated. As an example, we will evaluate the availability of a process that involves printing a document in UWE. The execution of that process goes through a sequence of processes such as, accessing the file system so that file(s) to be printed can be accessed by UNISON locally at the host executing the print command. Once UNISON's process is completed, system software (RFS, REXEC, etc.) are then used to access the print_server which in turn performs the necessary processes to drive the printer(s) specified by the print command. Hence, the print_process can be defined as a sequence of two processes: accessing the file system and accessing the print_server. The availability of the print command is therefore the probability of achieving these two processes successfully at a given time. That is,

$A(print, oacak) = A(user\_files, oacak) . A(print\_server, oacak).$

The availabilities of these two processes have been evaluated in the previous two subsections and thus the print availability is just the product of these two measures.

4.2.2 Case 2: With Redundant Resources

In distributed computing environment, the inherent redundant resources can be used to increase availability, fault-tolerance, and minimize the average response time. For example, if we assume hakar workstation is replica of handar. That means hakar will execute all processes routed to handar when it is down or heavily loaded. Similarly, handar can substitute for hakar when it is down.
or to improve the performance of the environment. By applying the same steps discussed in Case 1, the availabilities of the resources studied in Case 1 after introducing double redundancy to the resources available at hanar and hakar computers can be evaluated.

4.2.3 Availability Improvement Factor (AIF)
The improvement in availability because of introducing redundancy to the resources of a distributed system should be evaluated and could be used as a cost measure for the availability improvement. This measure can be defined as the ratio between the availability increase to the maximal increase (which occurs when availability becomes one). Namely,

\[ AIF = \frac{A(\text{with redundancy}) - A(\text{without redundancy})}{1 - A(\text{without redundancy})} \]

The availabilities of the resources discussed previously (print_server, user_files, and print_process) have been evaluated along with their associated AIFs. Table 3 shows the availabilities of the invoking the print-process with respect to all sites and the availability improvement factors obtained because of introducing double redundancy to the UWE's resources. In this analysis, we assume that the availabilities of the UWE components are as follows: Host Availability = 0.998; Node Availability = 0.99; Link Availability = 0.97; and Ethernet Availability = 0.97.

Table 3. Availability of the Print-Process in UWE.

<table>
<thead>
<tr>
<th>SITE</th>
<th>AVAILABILITY WITH REDUNDANCY</th>
<th>AVAILABILITY IMPROVEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>hanar</td>
<td>0.99800</td>
<td>0.99800</td>
</tr>
<tr>
<td>hakar</td>
<td>0.99396</td>
<td>0.99674</td>
</tr>
<tr>
<td>harvar</td>
<td>0.99285</td>
<td>0.99674</td>
</tr>
<tr>
<td>hadar</td>
<td>0.99179</td>
<td>0.99580</td>
</tr>
<tr>
<td>har</td>
<td>0.99253</td>
<td>0.99550</td>
</tr>
<tr>
<td>uca</td>
<td>0.91197</td>
<td>0.95950</td>
</tr>
<tr>
<td>sou</td>
<td>0.9612</td>
<td>0.96866</td>
</tr>
<tr>
<td>Til</td>
<td>0.92963</td>
<td>0.95814</td>
</tr>
<tr>
<td>Tilson</td>
<td>0.89272</td>
<td>0.95814</td>
</tr>
<tr>
<td>Til25</td>
<td>0.85728</td>
<td>0.92010</td>
</tr>
</tbody>
</table>

E(A) = 0.96259
E(A) = 0.98007

5. Conclusions
Several models to analyze the availability of the services provided to the users of a distributed computing environment has been proposed. The presented models provide not only the one-index representation of critical component availability measures, but also assist the process of designing, and engineering the overall distributed computing environment. The availability approach presented in this paper can also be used to identify bottlenecks and critical components; determine the sensitivity of system availability to hardware and software failures; and optimize the process of improving the availability of a distributed computing environment. In addition, since the failure and repair rates can be calculated for each component in the distributed computing environment, the reliability, maintainability, and other time-dependent measures can also be computed using the approach presented in this paper. Furthermore, the links and the nodes of process spanning trees can be labeled in a manner such that the effect of some other performance metrics (e.g., delay, throughput) can be modeled. Consequently, the same techniques can be adopted to evaluate these combined measures.

6. Acknowledgements
The authors like to thank D. T. Klein and W-M. Chen for their valuable information about the current implementation of UWE.

7. References