A Hierarchical Modeling and Analysis for Grid Service Reliability

Yuan-Shun Dai, *Member*, *IEEE*, Yi Pan, *Senior Member*, *IEEE*, and Xukai Zou, *Member*, *IEEE*

Abstract—Grid computing is a recently developed technology. Although the developmental tools and techniques for the grid have been extensively studied, grid reliability analysis is not easy because of its complexity. This paper is the first one that presents a hierarchical model for the grid service reliability analysis and evaluation. The hierarchical modeling is mapped to the physical and logical architecture of the grid service system and makes the evaluation and calculation tractable by identifying the independence among layers. Various types of failures are interleaved in the grid computing environment, such as blocking failures, time-out failures, matchmaking failures, network failures, program failures, and resource failures. This paper investigates all of them to achieve a complete picture about grid service reliability. Markov models, Queuing theory, and Graph theory are mainly used here to model, evaluate, and analyze the grid service reliability. Numerical examples are illustrated.

Index Terms—Grid reliability, resource management system, Markov model, queuing theory, graph theory.

1 Introduction

Grid computing [1] is a newly developed technology for complex systems with large-scale resource sharing, wide-area communication, multi-institutional collaboration, etc. [2], [3] [4], [5], [6].

The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic multi-institutional virtual organizations [4]. This is required by a range of collaborative problem-solving and resource-brokering strategies. This sharing is highly controlled by resource management system (RMS) [7], with resource providers and consumers defining what is shared, who is allowed to share, and the conditions under which the sharing occurs.

Recently, Open Grid Services Architecture [5] has enabled the integration of services and resources across distributed heterogeneous dynamic virtual organizations. A grid service is designed to complete a set of programs under the grid circumstances. The programs may need distributed remote resources. However, they initially do not know the site information of those remote resources in such a large-scale environment, so RMS plays an important role in managing the pool of shared resources, in matchmaking the programs to their requested resources, and in controlling them to access the resources through a wide-area network.

The structure and functions of the RMS in the grid have been introduced in detail in [7], [8], [9], [10]. Briefly stated, the programs in a grid service send their requests for resources to the RMS. The RMS adds these requests into the request queue [7]. Then, the requests wait in the queue for the matchmaking service of the RMS for a period of time (called waiting time) [11]. In the matchmaking service, the RMS matches the requests to the shared resources in the grid [12] and then connects the programs and their required resources. Thereafter, the programs can reach the remote resources and exchange information with them through the connections. The grid security mechanism then operates to control the resource access through Certification, Authorization, and Authentication, which constitute various logical connections to cause dynamicity in the network topology.

The above process can contain different types of failures that make a grid service unreliable. They are mainly blocking failures, time-out failures, matchmaking failures, network failures, program failures, and resource failures. When the new requests of the grid service arrive at the RMS, they cannot enter the request queue if it is full, so the blocking failure occurs [13]. Usually, the grid service may set a due time for the matchmaking service of the RMS [11], so the time-out failure occurs if the waiting time in the queue is longer than the due time of the program. If the RMS matches requests to the wrong resources, the matchmaking failure occurs [14]. Network failure may emerge when the programs are transmitting data through the network with remote resources [15]. The programs themselves are software that may contain software faults causing program failures [16]. The resources are usually heterogeneous [17], for example, they can be hardware, software, or firmware (such as database, protocol, processor, digital product, etc.) and, therefore, they may include either software or hardware faults that can induce the resource failures.

Although the developmental tools and techniques for the grid have been widely studied [1], grid reliability analysis and evaluation are not easy because of their complexity of combining various failures interacting with one another. As one of the important measures, the grid service reliability needs to be quantified, assessed, and predicted using new models and tools.

Some initial studies have been done in analyzing the grid computing reliability. Dai et al. [18] studied the service

- *Y.-S. Dai and X. Zou are with the Department of Computer and Information Science, Indiana University-Purdue University, 723 W. Michigan St. SL280, Indianapolis, IN 46202. E-mail: {ydai, xkzou}@cs.iupui.edu.*
- *Y. Pan is with the Department of Computer Science, Georgia State University, Atlanta, GA 30302-4110. E-mail: pan@cs.gsu.edu.*

Manuscript received 9 July 2005; revised 7 Aug. 2006; accepted 6 Oct. 2006; published online 21 Mar. 2007.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0232-0705.
Digital Object Identifier no. XXX

0018-9340/07/$25.00 © 2007 IEEE Published by the IEEE Computer Society

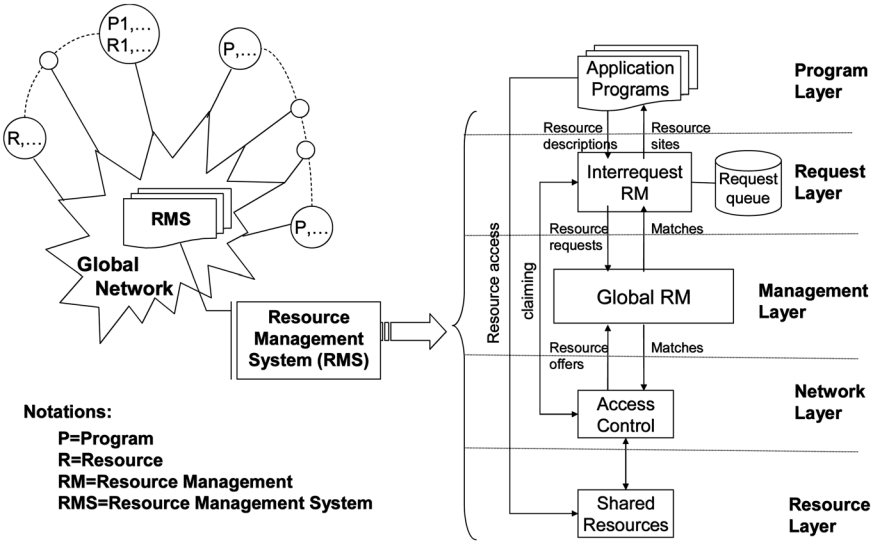


Fig. 1. The grid computing system.

reliability for a wide-area distributed system that is one of the ancestors of the grid system. The function of the control center in that model is similar to that of RMS for the grid computing. However, the reliability analysis of the control center is not exactly suitable for the RMS. Moreover, the reliability model of the subdistributed systems inherits the traditional models' characters [19], [20], [21], [22] and has certain limitations. Those traditional models have a common assumption that the operational probabilities of the nodes and links are constant. However, this assumption is unrealistic for the grid, so this assumption was relaxed in [15] by assuming that the failures of nodes and links followed their respective Poisson processes so that their operational probabilities decrease with their working time instead of the constant values. This new model is more reasonable and practical for the grid, but it only studied the network hardware failures for the grid reliability without considering other failures such as blocking failures, time-out failures, matchmaking failures, program failures, and resource failures. There are also many other reliability models for software, hardware, or small-scale distributed systems, see, e.g., [14], [23], [24], [25], [26], which cannot be directly implemented for studying grid service reliability.

The services in existing grid systems are often organized in a hierarchical fashion: They contain different program/resource/request/network/management layers that are interconnected through a set of interfaces. The reliability characteristics of some lower layers are largely independent of the layers above. From the system's perspective, the whole grid is also built from smaller, more manageable subsystems (such as the component-based approach). This characteristic of large scale systems fits naturally in the hierarchical modeling approach that is adopted by this paper.

This paper presents a new hierarchical model for the grid service reliability analysis. It also derives formulas and algorithms to effectively evaluate the grid service reliability. Section 2 describes the architecture of the grid computing with different layers and then analyzes different types of failures that may take place in grid services. Section 3 presents the hierarchical model for the grid service reliability and develops formulas and algorithms for evaluation. Numerical examples are illustrated in Section 3. Section 4 concludes this paper.

2 Grid Computing System and Failure Analysis

The grid computing system has emerged as an important new field, distinguished from conventional distributed computing systems by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.

2.1 Description of Grid Computing

The global grid system is generally depicted by Fig. 1. Various organizations [4] integrate or share their resources on the global grid. Any program running on the grid can use those resources if it can be successfully connected to them and is authorized to access them. The sites that contain the resources or run the programs are linked by the global network, as shown in the left part of Fig. 1.

The procedures for a program to use the remote resources are controlled by the RMS, which is the "brain" of the grid computing; see, for example, [7]. The RMS has five layers in general, as shown in the right-hand side of the Fig. 1. They are program layer, request layer, management layer, network layer, and resource layer, respectively.

1. *Program layer*. The program layer represents the programs of the customer's applications. The programs describe their required resources and constraint requirements. These resource descriptions are translated to the requests and sent to the next request layer.
2. *Request layer*. The request layer provides the abstraction of "program requirements" as a queue of resource requests. The primary goals of this layer are to maintain this queue in a persistent and fault-tolerant manner and to interact with the next management layer by injecting resource requests for matchmaking, claiming matched resources of the requests.
3. *Management layer*. The management layer may be thought of as the global resource allocation layer. It

has the function of automatically detecting new resources, monitoring the resource pool, removing failed/departure resources, and, most importantly, matching the resource requests of a service to the registered/detected resources. If resource requests are matched to available resources, the matched tags are sent to the next network layer.

4. *Network layer*. The network layer dynamically builds connections between the programs and resources when receiving the matched tags and controls them to exchange information through communication channels in a secure manner.
5. *Resource layer*. The resource layer represents the shared resources from different resource providers with the usage policies (such as service charge, reliability, serving time, and so forth).

Due to the hierarchical nature of the grid architecture, we find that hierarchical reliability modeling should be a good approach to handle the large-scale grid system. It will make the modeling and evaluation clear and tractable. Some layers are largely independent of one another, whereas some others closely interact. Therefore, failure analysis for this hierarchical modeling is necessary to identify these dependent and independent layers.

2.2 Failure Analysis and Grid Service Reliability

A grid service [5] is required to complete some specific programs in order to provide certain service to the users. At the "Program layer," the programs of a grid service/job initially send their requests for remote resources to the RMS. The "Request layer" will add these requests in the request queue. Please note that, in the request queue, the requests from other grid services may have also waited there, so the new requests should wait for service until the prior requests are finished. Then, the "Management layer" will try to locate the sites of the resources that match the requests. After all of the requests of those programs in the grid service are matched, the "Network layer" will build the connections among those programs and the matched resources and then control them to communicate.

It is possible to cause various types of failures on the respective layers as follows:

- *Program layer*. The programs are actually software applications that may cause software failures when running; see, e.g., [16] and [27].
- *Request layer*. When the programs' requests reach the request layer, two types of failures may occur. They are "blocking failure" and "time-out failure." Usually, the request queue [7] may have a limitation of the maximal number of requests waiting in the queue. When a new request arrives, if the queue is full, it is blocked so as to cause the blocking failure. The grid service usually has its due time set by the service monitor. If the waiting time for the requests in the queue is over the due time, the time-out failure occurs [11].
- *Management layer*. Here, "matchmaking failure" may occur, which means that the requests fail to match with the correct resources [14]. Errors like incorrectly translating the requests, registering a wrong resource, ignoring resource disconnection, and misunderstanding the users' requirements can cause such matchmaking failures.
- *Network layer*. When the programs are using the remote resources, the communication channels may be disconnected either physically or logically, which causes "network failure," especially for those transmissions of large files which require a long time [15].
- *Resource layer*. The resource shared on the grid may be software, hardware, or firmware. Hence, when using the resource, resource failures can be caused by software faults, hardware faults, or a combination of both.

By considering all the failures mentioned above, the grid service reliability can be defined by:

Grid Service Reliability. The probability that a set of programs contained by a grid service can be successfully completed: In particular, the requested resources by the programs are matched correctly and in time, the programs succeed in connecting and communicating to those resources, and both programs and resources are reliable when working.

3 Hierarchical Modeling and Evaluation for Grid Service Reliability

From the above failure analysis, we can find that the program layer, resource layer, and network layer interact heavily with one another because the programs need to access resources through the network. Therefore, the three layers cannot be viewed as independent or separated during modeling and analysis. The remaining two layers, the request layer and management layer, are relatively independent of the above three layers. By using some tricks in the modeling, the correlation can be negligible. Thus, there are three layers in this hierarchical model: 1) the Request Layer, 2) the Management Layer, and 3) the Network, Program, and Resource Layers, which will be analyzed, respectively. A numerical example will be interleaved with the analytical model for illustration.

3.1 Request Layer

This layer contains two types of failures: blocking failure and time-out failure, caused by the overflow and waiting time in the request queue. To prevent correlation from other layers, here the due time, T_d does not count all the time of the other layers but just the deadline for the waiting time of the submitted job in the request queue. It can be set by service monitors.

Suppose that the capacity of the request queue is N (the maximum number of requests). The arrival of submittals of various grid services/jobs to the RMS follows a Poisson process with the arrival rate λ_a. Different grid services/jobs may contain a batch of requests for using different resources, so the number of requests of an unknown service is a discrete random variable denoted by X. Denote the *pmf* for X to take the value x by

$$p(x) = \Pr(X = x) \quad (x = 1, 2, 3, \ldots). \tag{1}$$

Please note here that the submittals for services/jobs and the requests for different resources are two different concepts: The submittals come directly from the clients or users in accordance with the Poisson process and each submittal may ask for multiple resources, which generates a batch of resource requests in the request queue for the matchmaking service.

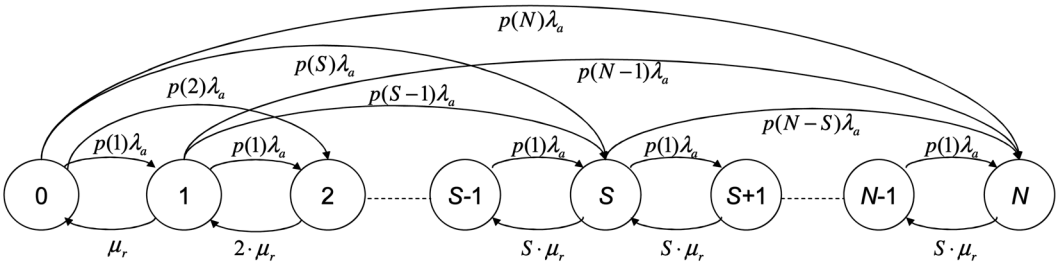


Fig. 2. Markov model for the request queue.

Usually, there are multiple RM servers to serve the requests, as shown by Krauter et al. [9, Fig. 2]. These RM servers are usually homogeneous, with similar structures, schemes, and equipment. Here, we assume S homogenous RM servers are simultaneously running to serve the requests. The service time to complete one request by each RM server is assumed governed by an exponential distribution with the parameter μ_r. Thus, such a process can be modeled by a Markov process as depicted in Fig. 2, in which state n $(n = 0, 1, \ldots, N)$ represents the number of requests in the queue. Here, the queue length n contains those requests that are being served by the S RM servers.

In Fig. 2, the transition rate from state n to state $n + x$ is $p(x)\lambda_a$ $(x = 1, 2, \ldots, N - n)$. If, at state n, the new service's requests $x > N - n$, adding all of the new service's requests will make the request queue overflow, so such new services are blocked and the RMS remains at state n with the rate $(1 - \sum_{x=1}^{N-n} p(x))\lambda_a$, which is not marked in Fig. 2. The service rate of a request by an RM server is μ_r. If $n \leq S$, the n requests can be immediately served by the S RM servers, so the departure rate of any one request is equal to $n \cdot \mu_r$. If $n > S$, only S requests are being simultaneously served by the S RM servers, so the departure rate is $S \cdot \mu_r$.

Denote q_n as the steady probability for the system to stay at state n $(n = 0, 1, \ldots, N)$. It is easy to derive the q_n by solving the following *Chapman-Kolmogorov* equations:

$$\left(\sum_{x=1}^{N} p(x)\lambda_a\right) q_0 = \mu_r q_1, \tag{2}$$

$$\left(n \cdot \mu_r + \sum_{x=1}^{N-n} p(x)\lambda_a\right) q_n = (n+1) \cdot \mu_r q_{n+1} + \sum_{y=0}^{n-1} p(n-y)\lambda_a q_y \quad (n = 1, \ldots, S-1), \tag{3}$$

$$\left(S \cdot \mu_r + \sum_{x=1}^{N-n} p(x)\lambda_a\right) q_n = S \cdot \mu_r q_{n+1} + \sum_{y=0}^{n-1} p(n-y)\lambda_a q_y \quad (n = S, \ldots, N-1), \tag{4}$$

$$S \cdot \mu_r \cdot q_N = \sum_{y=0}^{N-1} p(N-y)\lambda_a \cdot q_y, \tag{5}$$

$$\sum_{n=0}^{N} q_n = 1. \tag{6}$$

Illustrative Example. Suppose that the maximum number of requests waiting in the request queue allowed is 10, that is, $N = 10$, and there are three homogeneous RM servers serving in parallel, that is, $S = 3$. The arrival rate of various unknown grid jobs/services to the RMS is $\lambda_a = 1(\sec^{-1})$. As in (1), we suppose that the number of requests in unknown grid services is governed by a uniform distribution, that is,

$$p(x) = \Pr(X = x) = \frac{1}{b}, \quad x \in \{1, 2, 3, \ldots, b\}, \tag{7}$$

where we numerically set $b = 10$ here. Please note that the uniform distribution used here is only for illustration and other distributions can also be implemented in a similar way based on real conditions. The departure rate to complete each request by a server is $\mu_r = 0.5(\sec^{-1})$. To solve linear equations (2)-(6), we can obtain the q_n $(n = 0, 1, 2 \ldots, 10)$, as depicted in Fig. 3.

Suppose that the grid service under consideration needs to use a batch of H different resources in total, so H requests will be added into the request queue for the matchmaking. Note that the service under consideration is a known grid service that is different from the above unknown services/jobs. If a service is known, the number of requests in the service H is also determined (not a random variable). If the request queue is longer than $N - H$, not all of the H new requests can be added into the request queue, which causes the blocking failure. Hence, the probability for the blocking failure NOT to occur can be derived by

$$R_{block} = \sum_{n=0}^{N-H} q_n, \tag{8}$$

where q_n $(n = 0, 1, \ldots, N)$ is obtained by solving (2)-(6).

The H requests are supposed to be continuously added into the request queue because they are simultaneously requested by one grid service. After the H requests are added into the request queue, there is waiting time for all of them to be matched by the RM servers. Here, we adopt the rule of "first come first serve" (FCFS) for the request queue. If the waiting time is longer than a due time T_d that is preset for the matchmaking service, the time-out failure occurs. To study the time-out failure, the waiting process should be divided into three stages.

During the first stage, n requests are queuing in front of the considered H requests. If $n > S$ and $H \leq N - n$, the later H requests will be waiting for service (that is, not being served). Since the time to complete each request by an

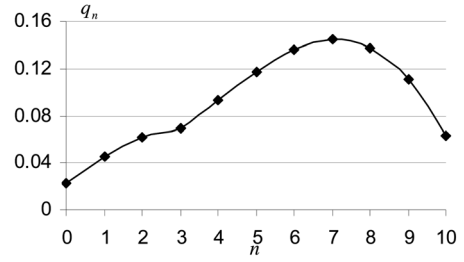


Fig. 3. Expected probability for the queue length to be n.

RM server follows exponential distribution with the parameter μ_r, the time to complete any one request of the S RM servers follows exponential distribution with the parameter $S \cdot \mu_r$. Then, the time for $n - S$ requests to be completed is a random variable, denoted by T_{n-S}, in accordance with the Erlang distribution, with the probability density function defined as

$$f_{n-S}(t) = S \cdot \mu_r e^{-S \cdot \mu_r t} \frac{(S \cdot \mu_r t)^{n-S-1}}{(n-S-1)!}, \quad t \geq 0 \text{ and } n > S. \quad (9)$$

After the $n - S$ requests are completed, the H requests of concern start entering the S RM servers one by one, which is viewed as the second stage. In this stage, a discrete time Markov chain (DTMC) is constructed, as shown in Fig. 4.

As Fig. 4 shows, the state number represents the number of considered requests that are being served in the S RM servers. The process starts from state 0 when the H considered requests immediately follow the S served requests. Then, if any request is completed, one of the H requests begins to be served, represented by state 1. The process continues till the last request of the H considered requests begins to be served, denoted as the final state k. When all of the H considered requests begin to be served, a total of $n + H - S$ requests have been completed by the parallel S RM servers. Thus, similarly to (9), the probability density function of the random variable T_{n+H-S} is

$$f_{n+H-S}(t) = S \cdot \mu_r e^{-S \cdot \mu_r t} \frac{(S \cdot \mu_r t)^{n+H-S-1}}{(n+H-S-1)!}, \quad t \geq 0 \text{ and } (n+H) > S. \quad (10)$$

Illustrative Example. Suppose that the number of requests in the job is three, that is, $H = 3$, and the due time for the three requests being matched is $T_d = 10$ seconds. Continuing the above numerical example as in Fig. 3, the probability without blocking failures can be obtained as $R_{block} = 0.6893$ by (8). Substituting parameters in (10), the probability density functions of different n are depicted in Fig. 5.

Nevertheless, the values of k (the final state) can be from 1 to $\min(H, S)$. The value of k will determine the later stage for the time-out failure analysis, so the probability for the process ending at state k, denoted by $\Pr(k)$, is required to be computed first.

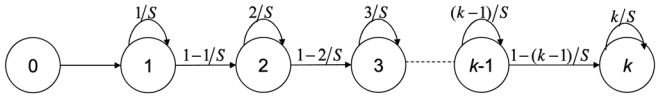


Fig. 4. DTMC for the second stage.

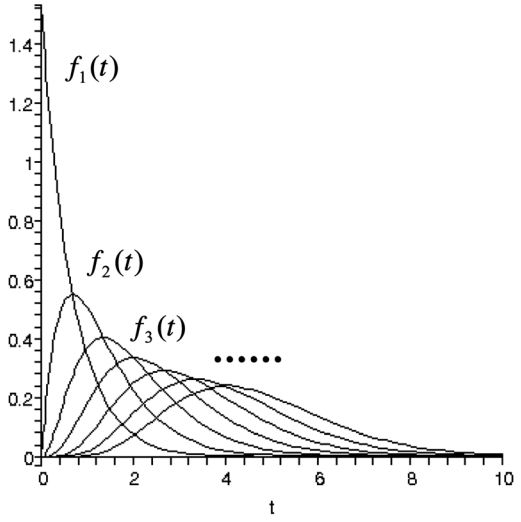


Fig. 5. The probability density functions $f_{n+H-S}(t)$.

From the DTMC of Fig. 4, it is easy to compute the $\Pr(k)$. Suppose that the number of transitions from state i to state i $(i = 1, 2, \ldots, k)$ is n_i $(n_i \leq 0)$ and then the probability for such a condition is obtained by the following function:

$$p(n_1, n_2, \ldots, n_k) = \prod_{i=1}^{k} \left(\frac{i}{S}\right)^{n_i} \cdot \prod_{i=1}^{k-1} \left(1 - \frac{i}{S}\right). \quad (11)$$

The DTMC must end at state k, $[1, \min(H, S)]$, if the following condition is satisfied:

$$n_1 + n_2 + \ldots + n_k = H - k. \quad (12)$$

Thus, the probability for the process ending at state k can be computed by the summation of all the probabilities that satisfy the condition (12) as

$$\Pr(k) = \sum_{n_1 + \ldots + n_k = H-k} p(n_1, n_2, \ldots, n_k). \quad (13)$$

Then, the process enters the third stage, during which the remaining k requests are being served in parallel. Hence, the process is modeled by the Markov process of Fig. 6, starting from state k.

Thus, the time for the process to reach the final state 0 is a hypoexponential random variable whose probability density function can be obtained by

$$g_k(t) = \sum_{i=1}^{k} (-1)^{i-1} \frac{k!}{i! \cdot (k-i)!} \cdot i \cdot \mu_r \cdot e^{-i \cdot \mu_r \cdot t}. \quad (14)$$

Then, the waiting time for all the H considered requests, before which there have been n requests, to be completed is a random variable denoted by $T_{H,n}$ whose density function can be obtained by substituting (10), (13), and (14) into

$$f_{H,n}(t) = \sum_{k=1}^{\min(H,S)} \Pr(k) \cdot f_{n+H-S}(t) \otimes g_k(t), \quad (15)$$

where "$\otimes$" represents the convolution operator of two functions.

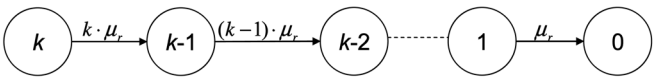


Fig. 6. The Markov process for the third stage.

Therefore, the probability for the waiting time in completing the H requests to be less than the due time, T_d, can be computed by the following formula:

$$\Pr(T_{H,n} < T_d) = \int_0^{T_d} f_{H,n}(t)dt. \tag{16}$$

Then, the expected probability for the time-out failure and blocking failure NOT to occur can be derived from

$$P_{time-block} = \sum_{n=0}^{N-H} q_n \int_0^{T_d} f_{H,n}(t)dt, \tag{17}$$

where q_n is obtained by solving (2)-(6) and $f_{H,n}(t)$ by (15). The summation in (17) between $[0, N-H]$ contains a condition that the blocking failure not to occur as analyzed by (8). Thus, in (17), $R_{time-block}$ represents the probability without time-out failures or blocking failures.

Illustrative Example. Continue the above example. From (11)-(13), we can obtain

$$\Pr(1) = \sum_{n_1=2} p(n_1) = p(2) = \frac{1}{9}, \tag{18}$$

$$\Pr(2) = \sum_{n_1+n_2=1} p(n_1, n_2) = p(0,1) + p(1,0) = \frac{4}{9} + \frac{2}{9} = \frac{2}{3}, \tag{19}$$

$$\Pr(3) = \sum_{n_1+n_2+n_3=0} p(n_1, n_2, n_3) = p(0,0,0) = \frac{2}{9}. \tag{20}$$

From (14), the PDF of hypoexponential functions $g_k(t)$ $(k = 1, 2, 3)$ is depicted in Fig. 7.

Then, substitute the values and functions of Figs. 5 and 7 and (18)-(20) into (12)-(16) to obtain the probability for the waiting time in completing the three requests before 10 seconds, that is, $\Pr(T_{3,n} < 10)$, $(n = 1, 2, \ldots, 7)$ as depicted in Fig. 8.

Then, through (17), we get the reliability without time-out failures or blocking failures:

$$R_{time-block} = 0.19414. \tag{21}$$

However, this reliability is too low, mainly due to the short queue capacity, $N = 10$, which is not realistic. Note that the abovementioned example with the short queue was just used for the purpose of illustration to make the analytical models more easily and more clearly understood.

Another more realistic example is given here with the following parameters: Queue Capacity: $N = 100$, Server Number: $S = 10$, Arrival rate: $\lambda_a = 1.4(\text{sec}^{-1})$, Departure rate $\mu_r = 0.8(\text{sec}^{-1})$, Due time: $T_d = 20$ seconds, and Uniform distribution: $b = 10$.

Suppose a grid service under consideration needs to complete three programs, say P_1, P_2, and P_3. The three programs request different resources in the grid, as given by Table 1, as well as the amount of transmitted data between the programs and the resources. Thus, there are a total six resources (R1, R2,... R6) that will be used by the grid service under consideration.

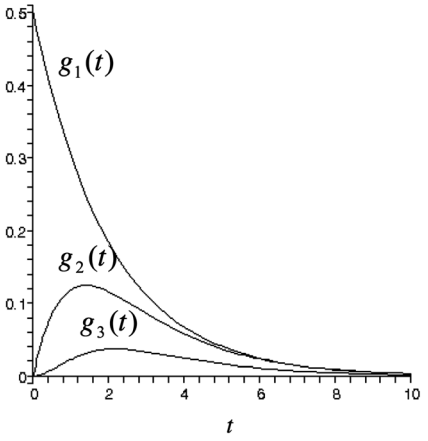


Fig. 7. The PDF of hypoexponential functions $g_k(t)$ $(k = 1, 2, 3)$.

Then, the grid service will send six requests to the RMS in order to identify and locate those resources on the grid, $H = 6$. Substituting those parameters into the formulas ((2)-(17)), they can be numerically solved by Maple to get the reliability without time-out or blocking failures:

$$R_{time-block} = 0.9809. \tag{22}$$

3.2 Management Layer

During the matchmaking process of an RM server, various protocols are implemented to translate and match the requests to their represented resources, such as disseminated and discovery protocols [9], resource trading protocols [11], etc. No matter what protocols the RM server uses, it is possible that certain requests are mismatched to the wrong resources, which causes matchmaking failure. For instance, a program is intended to use resource R1 but, due to certain ambiguous descriptions, the RM server translates the resource description into resource R2, which results in a matchmaking failure. There is another special type of mismatching failure, that is, No Match, which means the RMS cannot find any resource to match the requests. Such a case of no match can be viewed as that the requested resource is mismatched to a Null resource, which will make the service unable to continue (that is, failure) due to the resource which is lacking.

The assumptions for matchmaking failures are listed below:

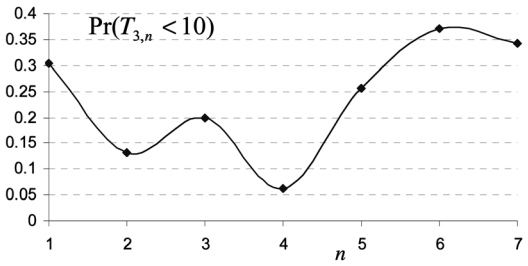


Fig. 8. The probability for the waiting time in completing the three requests before 10 seconds.

TABLE 1
Resources Used by Programs and Size of Exchanged Information

Program	Requested Resources	Exchanged information (Kbit)
P_1	R1, R2, R3	500,400,300
P_2	R3, R4,R5	200,100,150
P_3	R5,R6	400,100

1. The occurrence of matchmaking failures has failure rate $\lambda(k)$, which is a function to k (the number of faults in the grid).
2. If any matchmaking failure occurs, the program will send a feedback to RM servers and then the grid will automatically try to remove the fault that causes this failure where $p\,(0 < p \leq 1)$ is the probablity of successfully removing the fault and $q = 1 - p$ is the probability of failing to remove the fault.
3. New faults may be generated during the process and the occurrence of generating a new fault follows a Poisson distribution with a constant rate v.

According to these assumptions, we build a continuous time Markov chain (CTMC) to model this process. This Markov model, depicted in Fig. 9, is a birth-death Markov process with an infinite number of states, where state k represents k faults contained in the system.

Usually, $\lambda(k)$ is an increasing function to the number of faults k. It is designed for an RMS to be in service for a long time, especially for the Open Grid Service Architecture [5], so the above birth-death process of failures can be viewed as a long-run Markov process [26]. After running for a long time, the expected death rate, $p \cdot \lambda(k)$, will approach v given $p \neq 0$. The $\lambda(k)$ can be approximately viewed as a constant during a small enough time:

$$\lambda(k) \approx v/p.$$

Usually, the time for matchmaking some requests in a service can be viewed as small enough compared to the whole life of an open grid system. Thus, the occurrence of the matchmaking failures can be approximately modeled with a constant failure rate. Note that the failure rate can be easily estimated and dynamically adjusted if some data are recorded by the RMS, such as the number of mismatched requests, (n_f), the total number of requests, (n_t), and the expected time for completing each request, $\bar{\tau}$ (this one is not important). Then, the new value of the failure rate can be automatically updated by

$$\lambda = \frac{n_f}{n_t \bar{\tau}}$$

and, thus, the probability that all the given H requests are correctly matched to their required resources can be easily obtained by

$$R_{match} = \exp(-\lambda \cdot H \cdot \bar{\tau}) = \exp\left(-\frac{n_f}{n_t} H\right) = \exp(-\lambda_m H), \tag{23}$$

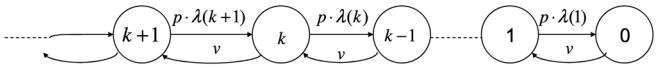


Fig. 9. CTMC for RMS reliability model.

where $\lambda_m = \frac{n_f}{n_t}$. Equation (23) is not related to the parameter $\bar{\tau}$, which is hard to exactly estimate, but n_f and n_t are much easier to count.

Illustrative Example. Suppose that the RMS records the number of mismatched requests $n_f = 100$ out of the total number of requests that have been completed $(n_t = 10,000)$. Thus, $\lambda_m = \frac{n_f}{n_t} = 0.01$ and then we get the probability for the matchmaking failure not occurring for the six requests by (23):

$$R_{match} = \exp(-0.01 \times 6) = 0.9418. \tag{24}$$

3.3 Network, Program, and Resource Layers

In the hierarchical modeling, this level is the most complicated one across the three different layers of Network Layer, Program Layer, and Resource Layer. The failures are classified as Network Failure in hardware (nodes and links) and Program and Resource Failures.

3.3.1 Network Failure

If the RMS has correctly matched the programs to their required resources, the programs are able to connect to and use those resources through the network. However, network failures may occur during this period.

Distributed network reliability for small-scale systems has been extensively studied, for example, [19], [20], [21], [22], [14, pp. 145-177]. However, those conventional models have some common assumptions: 1) The network topology is made up of physical links and nodes that are static without considering dynamic changes of components and logic structures; 2) the operational probabilities of nodes or links are constant without considering bandwidth and contention; and 3) the models only consider the hardware failures of links and processors without taking into account the software and resource failures. These assumptions need to be relaxed for grid.

In a grid network, communication between two remote sites can be logically broken even though a physical link exists between them; the authority to use some remote resources might be malformed for instance. To solve such problems, the new model proposes using a virtual structure instead of a physical structure. After matchmaking, a grid service needs to execute a set of programs and to use a set of resources. Hence, we can extract those nodes that contain these programs or resources as Virtual Nodes (VN). Then, a direct communication channel between two VNs is defined as a Virtual Link (VL) that represents not only a physical connection, but also a logical link. Also, due to the grid security mechanism, valid certificates of machines and users are necessary to access some resources so that the job submittals from different sites or users will cause different network topology. Thus, the VL is again better than the physical links by considering the security scenario. Another advantage of virtualization is to simplify the graph model for the grid covering a wide area with potentially many thousands of physical processors and cables.

Operational probabilities of VNs and VLs cannot just be set at a constant value such as 0.9, as the previous

conventional models did. Rather, the operational probabilities are affected by various conditions such as failure rate, transmitted data, available bandwidth, and operation time. Thus, our model will consider the following information which is normal in dynamic grid network: 1) For VLs, the model considers the available bandwidth, the exchanged data from different sources that contend for the bandwidth of the VLs, and the failure rate of the VLs. 2) For VNs, besides the above information for communication, the time for the VNs to execute programs/resources should also be involved. A combination of all of these conditions is much closer to the reality of a grid network and can handle the variability of a wide-area network.

In detail, different programs can exchange information of different sizes with the same resources. Denote by D_{mh} the size of information exchanged between program P_m $(m = 1, 2, \ldots, M)$ and resource R_h $(h = 1, 2, \ldots, H)$. Suppose $S(i, j)$ is the available bandwidth of the channel $VL(i, j)$ between two nodes VN_i and VN_j. The different programs may contend with common VLs' bandwidth when they are running in parallel. Denote by $D(i, j)$ the total information (bits) contending for the available bandwidth of the channel $VL(i, j)$. Then, the conditions of bandwidth and contention can be transformed into total communication time by

$$T_c(i, j) = \frac{D(i, j)}{S(i.j)}. \tag{25}$$

Denote the failure rate of the VN_n by λ_n and of the $VL(i, j)$ by $\lambda_{i,j}$. The reliability of the link for exchanging the information can be expressed by

$$R_L(i, j) = \exp\{-\lambda_{i,j} T_c(i, j)\}. \tag{26}$$

The total communication time of the node VN_j can be calculated by

$$T(j) = \sum_{i \in Q_j} T_c(i, j), \tag{27}$$

where Q_j represents the set of nodes that communicate with the node VN_j.

The reliability function of the node VN_j for communication is

$$R_c(j) = \exp\{-\lambda_j T(j)\}. \tag{28}$$

In order to evaluate the network reliability for the given programs and resources, the graph theory is implemented here. The set of VNs and VLs involved in running the given programs and exchanging information with the resources form a resource spanning tree (*RST*). The smallest dominating *RST* is called the Minimal Resource Spanning Tree ($MRST$). The detailed study of $MRST$ was given in [15].

The term "$element$" is defined here to represent both the nodes and links of the $MRST$. Assume that there are a total of K elements in an $MRST$ so that $element_i$ $(i = 1, 2, \ldots, K)$ denotes the ith element in the $MRST$. Accordingly, the communication time of the ith element is denoted by $T_w(element_i)$ and $\lambda(element_i)$ represents its failure rate. The reliability of the $MRST$ combining (26) and (28) can be simply expressed as

$$R_{MRST} = \prod_{i=1}^{K} \exp\{-\lambda(element_i) \cdot T_w(element_i)\}. \tag{29}$$

With this equation, the reliability of an $MRST$ can be computed if the communication time of all the elements is obtained. Hence, finding all the $MRST$s and determining the communication time of their elements are the first step in deriving the grid network reliability. An algorithm is presented in [15] to search the $MRST$s for a given program executed by one given VN. Repeatedly using this algorithm, all the $MRST$s to connect all the programs to their required resources can be found, respectively. *Algorithm 1* is briefly described as follows:

Step 1. Given a program, say P_m, start from a node that contains this program, search the required resources along the possible links, and record elements that compose the searching trace and their communication times.

Step 2. When all of the required resources are reached, an $MRST$ is found; record this $MRST$.

Step 3. Then, other routes are tried to search other $MRST$s until all of the $MRST$s are searched.

Step 4. Change to another node that also contains the program P_m. Repeat the abovementioned three steps until all of the nodes have P_m are developed. Save all the found $MRST$s associated with P_m into the vector.

Step 5. Change to another program and repeat the above four steps until all of the programs are explored. Then, all the vectors of $MRST(P_m)$ $(m = 1, 2, \ldots M)$ are generated.

Thus, at least one MRST of each $MRST(P_m)$ $(m = 1, 2, \ldots M)$ is reliable and then the program $MRST(P_m)$ $(m = 1, 2, \ldots M)$ can be connected to those remote resources and exchange information with them successfully through the network. If any set of the M programs is successful, then the network is reliable for the grid service to execute the required set of programs, so the grid network reliability can be written as the probability of the intersection of the set of $MRST$s of each program, which is

$$R_{net} = \Pr\left(\bigcap_{m=1}^{M} MRST(P_m)\right). \tag{30}$$

The above equation is computationally tractable (see the detailed algorithms in [15]).

3.3.2 Program and Resource Failure

In the operational phase, the program failures can be assumed to follow the exponential distributions [28]. Since the same program running on different processors may have different failure rates, the failure rate of P_i running on processing node G_j is denoted by $\lambda_s(i, j)$. Also, the processing time of P_i on G_j is denoted by $t(i, j)$. Thus, the reliability of the software program P_i running on G_j can be simply computed by

$$R_{\text{prog}}(i, j) = \exp\{-\lambda_s(i, j) \cdot t(i, j)\}. \tag{31}$$

Suppose the time for resource h to work is determined by the requested program P_i and the processing node G_j, denoted by $t(h, i, j)$. Also, we denote the failure rate of the

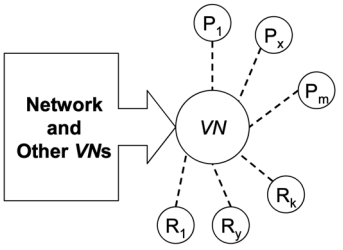


Fig. 10. VN and its subnodes of programs and resources.

resource h on the node G_j by $\lambda_r(h,j)$, which follows the exponential distribution. Thus, the reliability of resource h used by P_i and integrated on G_j can be simply expressed by

$$R_{\text{res}}(h,i,j) = \exp\{-\lambda_r(h,j) \cdot t(h,i,j)\}. \quad (32)$$

So far, the above network model only considers the hardware failures of VNs and VLs. To further increase its fidelity, program and resource failures should also be included. Thus, the programs and resources should not be drawn inside a node of the graph, as depicted by the conventional models [15], [19], [21], [22], even though its physical structure may be so. The reason is because it is hard to handle programs/resources/processors separately within one node. In order to overcome this problem, a new graph model is presented which depicts the programs and resources as subnodes of the main VN, as shown in Fig. 10. Then, the programs and resources can be logically separated with their respective failure rates and processing times, which will combine both hardware failures and software failures.

Comparing (31) and (32) to (26) and (28), we can find that they are of the same format. Therefore, it is easy to directly use the above models and algorithms for deriving the network reliability. Thus, all of the programs, resources, VNs, and VLs with their respective failure rates and processing/communication time can be integrated together as general elements of the $MRST$ analysis given by (29). Then, the reliability of combining programs, resources, and network can be derived from (30) and the algorithms of [15] for calculating (30) can be directly implemented to obtain the overall reliability, denoted by $R_{net-p-r}$.

Here, we assume that the reliability of different individual elements (program, resource, node, link), e.g., (26), (28), (29), (31), and (32), satisfy their respective exponential distributions. This assumption has been well verified on single component without repair or debug; see, e.g., [16], [26], [27]. In the grid, most components are in the operational phase without repair or debug during the service time. Though they may be repaired or debugged, usually the repair, debug, or reboot process should be performed offline, that is, not during the service time. After the repair, when the components return to the operational phase, the exponential distribution is valid again. It has been explicitly proven in [28] that a single component in the *operational phase* will follow the exponential distribution no matter what distribution it has in the *test phase*.

In addition, the time to execute a grid service is sufficiently short compared with the components' lives or degradation period. Therefore, the reliability with exponential assumption on a single component is also justified at this point. The individual component can be monitored in real time and updates the parameters dynamically for the exponential distribution. The monitored information is simple: just the number of failures over the total running time of this component which has actually been recorded by log files in today's grids. Such a dynamic updating scheme can further validate the exponential assumption, though we may relax the above assumption somewhat to allow reasonable or gradual change in the failure rate (such as wear out) because, during a short enough period of service time, the parameter cannot change too much and using the latest value should be a good approximation. However, the random and significant jump of parameters is not considered here, which is an open problem and needs further research.

Illustrative Example. Continuing the example depicted in Table 1, after knowing the sites of the resources, the programs attempt to connect to them and use them. Suppose there are six VNs that execute the three programs or contain the required resources in a redundant manner. The topology of the connections among the VNs and programs/resources is depicted in Fig. 11 using subnodes representing programs and resources. The failure rates of the nodes/links/programs/resources are marked in Fig. 11 and the available bandwidth (Kbit per second) is also marked beside the VLs.

The processing time of different programs executed by different nodes is given in Table 2 and that of resources on different nodes used by different programs is given in Table 3.

With these numerical parameters, the algorithms presented in Section 3.3 are implemented to obtain the reliability combining the network, programs, and resources:

$$R_{net-p-r} = 0.9444. \quad (33)$$

3.4 Grid Service Reliability

The failure of a grid service is actually a combination of the blocking failure, time-out failure, matchmaking failure, network failure, and program and resource failure. Those failures can be divided into three independent groups as related to three different levels of the hierarchical modeling as above: The blocking failures and time-out failures are related to the Request Layer with the request queue, the matchmaking failures are related to the management layer with RM servers, and the network, program, and resource failures are related to a lower level for execution other than the previous two levels for management.

The independence between RM Servers and the Network/Resource/Program Layers can be justified as follows: When a job is submitted to the RM servers, the RM servers automatically detect the available resources that are matched to the requests. Note that the availability here means the initially good and accessible resources. If some resources are unable to be connected due to the network problems or are detected at an unavailable state due to software/hardware failures, those resources are not fed to the next layers for calculating the $R_{net-p-r}$. Following that, the RM servers do not participate in the calculation of $R_{net-p-r}$ because the RM servers have told the programs the sites of available resources and the programs can directly access to them. Thus, the two parts can be viewed as statistically independent, considering separate failures and time difference in the model, that is, the calculation on

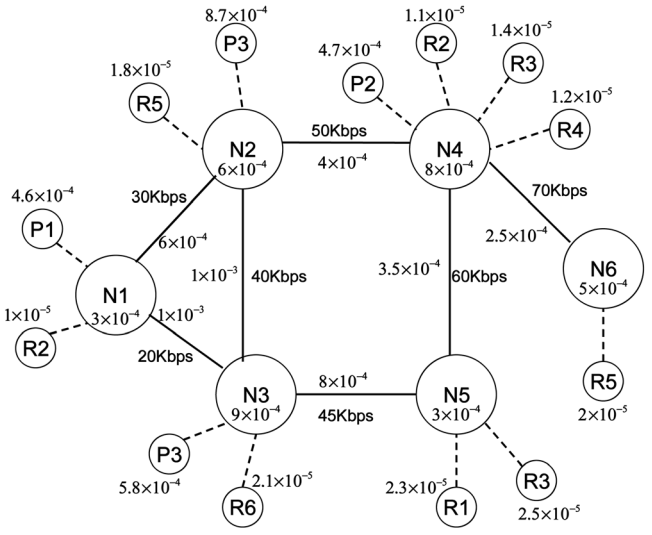


Fig. 11. Topology of the network, programs, and resources.

network reliability is not started until the end of matchmaking by RM servers.

Under the condition of independence among the three levels of the hierarchical modeling, the grid service reliability can be calculated straightforwardly by

$$R_{service} = R_{time-block} \cdot R_{match} \cdot R_{net-p-r}. \quad (34)$$

For example, substituting (22), (24), and (33) into (34), the grid service reliability is

$$R_{service} = 0.9809 \times 0.9418 \times 0.9444 = 0.8724.$$

Thus, the probability for this grid service to be successfully completed is 0.8724.

In summary, the overall outline for the grid service reliability based on the hierarchical modeling is concluded using the following steps:

Step 1. Solve (2)-(6) to obtain the q_n $(n = 0, 1, \ldots, N)$; substitute parameters into (10) to obtain $f_{n+H-S}(t)$ $(n = 0, 1, \ldots, N)$; substitute (11) into (13) to obtain $\Pr(k)$ $k \in [1, \min(H, S)]$; and derive $g_k t$ from (14).

Step 2. Substitute $\Pr(k)$, $g_k(t)$, and $f_{n+H-S}(t)$ into (15) to obtain $f_{H,n}(t)$ $(n = 0, 1, \ldots, N)$ and then substitute q_n and $f_{H,n}(t)$ into (17) to get $R_{time-block}$.

Step 3. Derive λ_m from the historical data of n_f and n_t, and substitute into (23) to get R_{match}.

Step 4. Draw the virtual network architecture by the given programs and resources, as in Fig. 10. Use *Algorithm 1* to find all MRSTs. Then, use the algorithms presented in [15] to solve (30), but note that, here, (31) and (32) are merged into the process through Fig. 10, so the result from (30) is $R_{net-p-r}$.

Step 5. Finally, substitute the outcomes from Steps 2 to 4 into (34) to obtain the $R_{service}$.

TABLE 2
Running Time (in Seconds) of Programs

Program	P1 on N1	P2 on N4	P3 on N2	P3 on N3
Time	30	35	15	13

4 Conclusion and Discussion

This paper is original in that it comprehensively and systematically studies the grid reliability, considering blocking failures, time-out failures, matchmaking failures, network failures, program failures, and resource failures in a hierarchical manner. The hierarchical modeling maps the physical and logical architecture of the grid service system and makes the evaluation and analysis clear and simple by identifying the independence among layers. Markov models, queuing theory, graph theory, and Bayesian analysis were mainly used to derive the grid service reliability. A numerical example has also been illustrated to show the procedures and effectiveness for modeling and evaluating the grid service reliability.

In our grid model, the RMS serving for the requests of grid services used one common request queue scheduled for multiple RM servers. It is also possible that each RM server has its own request queue. Actually, this condition has been covered by the general structure of our RMS model. If the request queue of an RM server does not interact with other RM servers' request queues, when the grid service requests reach this RM server's queue, it can be analyzed by our model, assuming $S = 1$ (that is, one RM server), which is a reduced case of our general RMS model. Furthermore, it is better that the interaction among different request queues is permitted, for example, if some requests of a grid service are blocked by one queue, those blocked requests can be transferred to another RM server's queue that is not full. Actually, our model has this advantage because it uses one common queue whose capacity can be set as the summation of separate queues' capacity of all RM servers. In addition, our model has another advantage: Using one common queue can balance the load to different RM servers well, that is, the unbalanced case that one RM server is idle, whereas the other has a lot of requests waiting for service will not take place.

Acknowledgments

The authors appreciate the constructive comments from Professor Dhiraj Pradhan and three anonymous reviewers who helped to significantly improve the paper.

TABLE 3
Running Time (in Seconds) of Resources

Time	R1 on N5	R2 on N1	R2 on N4	R3 on N4	R3 on N5	R4 on N4	R5 on N2	R5 on N6	R6 on N3
P_1	5	30	30	20	50				
P_2				15	30	25	20	40	
P_3							25	30	33

REFERENCES

[1] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure.* Morgan-Kaufmann, 2003.

[2] A. Kumar, "An Efficient SuperGrid Protocol for High Availability and Load Balancing," *IEEE Trans. Computers,* vol. 49, no. 10, pp. 1126-1133, Oct. 2000.

[3] S.K. Das, D.J. Harvey, and R. Biswas, "Parallel Processing of Adaptive Meshes with Load Balancing," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 12, pp. 1269-1280, Dec. 2001.

[4] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int'l J. High Performance Computing Applications,* vol. 15, pp. 200-222, 2001.

[5] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke, "Grid Services for Distributed System Integration," *Computer,* vol. 35, no. 6, pp. 37-46, June 2002.

[6] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, "Adaptive Computing on the Grid Using AppLeS," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 4, pp. 369-382, Apr. 2003.

[7] M. Livny and R. Raman, "High-Throughput Resource Management," *The Grid: Blueprint for a New Computing Infrastructure,* pp. 311-338. Morgan-Kaufmann, 1998.

[8] J. Cao, S.A. Jarvis, S. Saini, D.J. Kerbyson, and G.R. Nudd, "ARMS: An Agent-Based Resource Management System for Grid Computing," *Scientific Programming,* vol. 10, no. 2, pp. 135-148, 2002.

[9] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing," *Software—Practice and Experience,* vol. 32, no. 2, pp. 135-164, 2002.

[10] J. Nabrzyski, J.M. Schopf, and J. Weglarz, *Grid Resource Management.* Kluwer Academic, 2003.

[11] D. Abramson, R. Buyya, and J. Giddy, "A Computational Economy for Grid Computing and Its Implementation in the Nimrod-G Resource Broker," *Future Generation Computer Systems,* vol. 18, no. 8, pp. 1061-1074, 2002.

[12] Q. Ding, G.L. Chen, and J. Gu, "A Unified Resource Mapping Strategy in Computational Grid Environments," *J. Software,* vol. 13, no. 7, pp. 1303-1308, 2002.

[13] J. Pu, X. Zhang, and Z. Wu, "The Research on QoS for Grid Computing," *Proc. 2003 Int'l Conf. Comm. Techniques (ICCT '03),* pp. 1711-1714, 2003.

[14] M. Xie, Y.S. Dai, and K.L. Poh, *Computing Systems Reliability: Models and Analysis.* Kluwer Academic, 2004.

[15] Y.S. Dai, M. Xie, and K.L. Poh, "Reliability Analysis of Grid Computing Systems," *Proc. IEEE Pacific Rim Int'l Symp. Dependable Computing (PRDC '02),* pp. 97-104, 2002.

[16] M. Xie, *Software Reliability Modeling.* World Scientific, 1991.

[17] J. Basney, M. Livny, and P. Mazzanti, "Utilizing Widely Distributed Computational Resources Efficiently with Execution Domains," *Computer Physics Comm.,* vol. 140, nos. 1-2, pp. 246-252, 2001.

[18] Y.S. Dai, M. Xie, K.L. Poh, and G.Q. Liu, "A Study of Service Reliability and Availability for Distributed Systems," *Reliability Eng. and System Safety,* vol. 79, no. 1, pp. 103-112, 2003.

[19] V.K.P. Kumar, S. Hariri, and C.S. Raghavendra, "Distributed Program Reliability Analysis," *IEEE Trans. Software Eng.,* vol. 12, pp. 42-50, 1986.

[20] D.J. Chen and T.H. Huang, "Reliability Analysis of Distributed Systems Based on a Fast Reliability Algorithm," *IEEE Trans. Parallel and Distributed Systems,* vol. 3, no. 2, pp. 139-154, Feb. 1992.

[21] D.J. Chen, R.S. Chen, and T.H. Huang, "A Heuristic Approach to Generating File Spanning Trees for Reliability Analysis of Distributed Computing Systems," *Computers and Math. with Applications,* vol. 34, pp. 115-131, 1997.

[22] M.S. Lin, M.S. Chang, D.J. Chen, and K.L. Ku, "The Distributed Program Reliability Analysis on Ring-Type Topologies," *Computers and Operations Research,* vol. 28, pp. 625-635, 2001.

[23] C.R. Das and J. Kim, "A Unified Task-Based Dependability Model for Hypercube Computers," *IEEE Trans. Parallel and Distributed Systems,* vol. 3, no. 3, pp. 312-324, Mar. 1992.

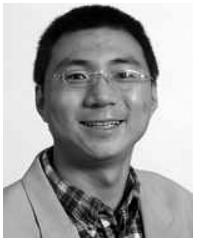
[24] D.M. Blough and A. Pelc, "Almost Certain Fault Diagnosis through Algorithm-Based Fault Tolerance," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 5, pp. 532-539, May 1994.

[25] Z.T. Kalbarczyk, R.K. Iyer, S. Bagchi, and K. Whisnant, "Chameleon: A Software Infrastructure for Adaptive Fault Tolerance," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 6, pp. 560-579, June 1999.

[26] K.S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* John Wiley & Sons, 2001.

[27] H. Pham, *Software Reliability.* Springer, 2000.

[28] B. Yang and M. Xie, "A Study of Operational and Testing Reliability in Software Reliability Analysis," *Reliability Eng. and System Safety,* vol. 70, pp. 323-329, 2000.



Yuan-Shun Dai received the bachelor's degree from Tsinghua University and the PhD degree from the National University of Singapore. He is a faculty member in the Computer Science Department at Purdue University School of Science, Indiana University-Purdue University Indianapolis. His research is in dependability, grid computing, security, and autonomic computing. He has published four books and more than 60 articles in these areas. His research was featured by *Industrial Engineer Magazine* (p. 51, December 2004). He is the program chair for the 12th IEEE Pacific Rim Symposium on Dependable Computing (PRDC '06) and the general chair for the Second IEEE Symposium on Dependable Autonomic and Secure Computing (DASC '06) and for DASC '07 as well. He also chairs many conferences and is on the editorial board of some journals; for example, he is a guest editor for the *IEEE Transactions on Reliability*, *LNCS*, the *Journal of Computer Science*, and the *International Journal of Autonomic and Trusted Computing*. He is a member of the IEEE and the IEEE Computer Society.

Yi Pan received the BEng and MEng degrees in computer engineering from Tsinghua University, China, in 1982 and 1984, respectively, and the PhD degree in computer science from the University of Pittsburgh in 1991. He is the chair and a professor in the Department of Computer Science and a professor in the Department of Computer Information Systems at Georgia State University. His research interests include parallel and distributed computing, optical networks, wireless networks, and bioinformatics. He has published more than 100 journal papers, 100 conference proceedings, and 24 books. He has served as the editor-in-chief or an editorial board member for 15 journals, including five IEEE transactions. He has organized several international conferences and workshops and has also served as a program committee member for several major international conferences such as INFOCOM, GLOBECOM, ICC, IPDPS, and ICPP. He has delivered more than 10 keynote speeches at many international conferences. He is an IEEE Distinguished Speaker (2000-2002), a Yamacraw Distinguished Speaker (2002), a Shell Oil Colloquium Speaker (2002), a senior member of the IEEE, and a member of the IEEE Computer Society.

Xukai Zou received the BS and MS degrees from Zhengzhou University and Huazhong University of Science and Technology, respectively, and the PhD degree from the University of Nebraska-Lincoln, all in computer science. He is a faculty member in the Computer Science Department at the Purdue University School of Science, Indiana University-Purdue University Indianapolis. His research is in applied cryptography, communication networks and security, and grid computing. He has published three books and more than 20 articles in these areas. He is a recipient of a US National Science Foundation Cyber Trust Award and the lead author of the book *Secure Group Communication over Data Networks* (Springer). He is the program chair and a program committee member for a number of international conferences and serves on the editorial boards and as a reviewer for many international journals and conferences. He is a member of IEEE and the IEEE Computer Society.