A Scheduling Model for Maximizing Availability with Makespan Constraint Based on Residual Lifetime in Heterogeneous Clusters*

Xin Jiang, Chuang Lin, Hao Yin, and Yada Hu

Department of Computer Science and Technology,
Tsinghua University, Beijing, P.R. China
jiangx05@mails.tsinghua.edu.cn,
{clin,hyin,yadandaner}@csnet1.cs.tsinghua.edu.cn

Abstract. A notable requirement of clusters is to maximize its processing performance. Lots of work in this area has been done to optimize the system performance by improving certain metric such as reliability, availability, security and so on. However, most of them assumes that the system is running without interruption and seldom considers the system's intrinsic characteristics, such as failure rate, repair rate and lifetime. In this paper, we study how to achieve high availability based on residual lifetime analysis for the repairable heterogeneous clusters with makespan constraints. First, we provide an availability model based on addressing the cluster's residual lifetime model. Second, we give an objective function about the model and develop a heuristic scheduling algorithm to maximize the availability the makespan constraint. At last, we demonstrate these advantages through the extensive simulated experiments.

Keywords: Scheduling strategy, Availability, Residual lifetime, Cluster, Makespan.

1 Introduction

With the advent of new high-speed networks, it is now possible to link together a collection of distributed, cost-effective and possibly heterogamous resources in the form of a cluster [1]. Heterogeneous cluster is the coordinated use of different types of computers, networks and interfaces to meet the requirements of widely varying application. In fact, most components of such system are different in many aspects such as total running time, processing performance, expected residual lifetime and other physical conditions with respect to circumstances. These differences have a strong impact on the task processing performance for clusters. In order to achieve high performance, some metrics, which can affect system availability, must be considered carefully. Our work is to improve the

* This work is supported by the National Natural Science Foundation of China (No.90412012, 60473086 and 60673184) and the National Grand Fundamental Research 973 Program of China (2006CB708301).

K. Li et al. (Eds.): NPC 2007, LNCS 4672, pp. 31–40, 2007.

availability of clusters without sacrificing conventional system quality of services (QoS) metrics. With QoS constraints, we address a novel scheduling strategy to endeavor to maximize the availability from the system residual lifetime's point of view.

For a job consisting of m tasks $t_1, t_2, ..., t_m$, how to allocate these tasks to each computing node is a critical problem. Some previous researches focus on the conventional measures about system qualities of service, such as the response time, finishing time, load balancing, etc. In an actual computing environment, however, it seems that how to complete the entire job successfully is more important than how to finish the job as quickly as possible. That is, the availability of system to processing tasks maybe a more important performance metric than the system's response time. For example, consider the residual lifetime of each computing node for cluster at time t, see Fig. 1. The time interval $R_i, (i = 1, 2, ..., n)$ is the residual lifetime of corresponding node at time t. For a task assignment scheme with the goal to maximize the system performance, intuitively, it may be better to allocate the task with longer execution time to the node which has longer residual lifetime. That is, under such task assignment, there's a decrease in the number of the worse cases that the node failed before the task processed by the node is finished.

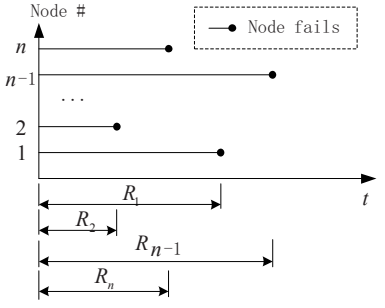

Fig. 1. Residual lifetime distribution of cluster at time t

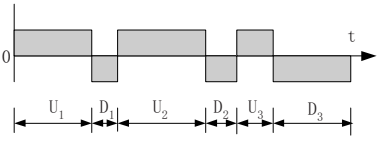

Fig. 2. A typical evolution of a failure-repair progress for a computing node

In this paper, we propose a novel model to describe the availability for repairable heterogeneous cluster based on its expected residual lifetime with the tasks' *makespan* constraints. Then we employ the model to put forward a task scheduling strategy, which can minimize the cluster availability, by allocating the tasks with longer execution time to the nodes with longer expected residual lifetime. At last, we conduct a series of simulated experiments to illustrate our task distribution strategy which is efficient to balance the availability and the *makespan* of heterogeneous clusters.

The rest of this paper is organized as follows. In section 2, we briefly introduce related work. Section 3 extends the availability model based on analyzing the

residual lifetime and then gives the objective function for optimization. We provide a scheduling algorithm to maximize the availability and with the *makespan* constraint for the cluster in section 4. Section 5 presents the simulation results of the algorithm and gives the experimental results. Section 6 concludes the whole paper and presents our future work in this field.

2 Related Work

Scheduling Strategies for cluster have been extensively studied in previous work both experimentally and theoretically. Besides widely investigated task allocations that improve system performance by optimizing the conventional performance measures, such as *makespan* or completion time [2], a few established task allocation models attach importance to system reliability, availability, safety and robustness, etc. Shatz et al deal with the task allocation with the goal of maximizing the system reliability [3]. Srinivasan et al describe a method to determine an allocation that introduces safety into a heterogeneous distributed system and at the same time attempts to maximize its reliability. Xie and Qin integrate tasks' availability requirements into stochastic scheduling so as to achieve a good balancing between system availability and throughput measured as average response time [4]. Schmidt [5] reviews results related to deterministic scheduling problems where machines are not continuously available for processing. [6] and [7] refers to a resource allocation's tolerance to uncertainty as the robustness of that resource allocation. It presents a stochastic robustness metric suitable for evaluating the likelihood that a resource allocation will perform acceptably. In [8], Dogan and Ozguner present two different cost functions which can guide a matching and scheduling algorithm to produce task assignments so that failures of the network resources will have less effect on the execution of application. Topcuoglu et al introduce two scheduling algorithms for a bounded number of heterogeneous processors with an objective to simultaneously meet high performance and fast scheduling time [9]. In [10], Hariri and Raghavendra provide two optimization algorithms for allocating the functions of a given distributed task so that the reliability is maximized and the communication delay is minimized.

In these researches introduced above, more attention is put on the metrics of reliability and availability. For instance, the maximum reliability is achieved by straightly optimizing the system reliability function, $e_{-\lambda t}$, with respect to the accumulative execution time and the intertask communication cost in [3] and [11]. An availability deficiency is defined which represents the discrepancy between the availability of a node and the availability requirements of tasks allocated to the node. By balancing the availability discrepancy and the *makespan*, a stochastic scheduling scheme is developed [4]. Our work is different from these methods above. We try to find a task allocation scheme to optimize the system availability by maximizing the objective function based on nodes' expected residual lifetime as much as possible. Furthermore, the heterogeneous cluster we considered is repairable which has been neglected by most researches above. That is, we believe that the computing node is a failure-repair system, and

alternates times and again between the up state and down state. Objectively, such failure-repair system is suitable in most actual cases.

3 Scheduling Strategy Model

3.1 Assumption and System Model

In this paper, cluster is assumed to consist of a set of heterogeneous computing nodes, $N = \{n_1, n_2, ..., n_k, k = 1, 2, ..., n\}$, connected via a high-speed interconnection network. The computing nodes only have local memory and do not share any global memory. We assume every computing node has different failure and repair rate. During an interval, all nodes might have undergone one or more failures inevitably, including either instantaneous failures or permanent failures. We suppose when the instantaneous failure occurs the node can auto-resume by re-configuring the system parameters or rebooting the machine. While the permanent failure occurs, the failed nodes are made operational by repairing or replacing. Therefore, it can be described as a Poisson stochastic process. See section 3.2. We also assume that a task can be interrupted at any time when failure occurs and continued after repair is finished.

Given a job consists of m independent tasks $T = \{t_1, t_2, ..., t_m\}$ arriving at the scheduler of cluster, it is the duty for the scheduler to allocate each task to one of the n computing nodes. Different task assignments have diverse efficiency of task processing for cluster. Successful implementation of a job requires every task of it to be finished reliably by corresponding computing nodes. At the same time, some QoS constraints, for example the *makespan*, must be considered. In our study, we present an objective cost function with the *makespan* constraint for the cluster availability. The main notations and definitions will be used in the rest of paper specified as Table 1.

Clearly, we have the equation $\tau_k = \sum_{i=1}^{m} p_{ik} e_{ik}$, and the job finishing time, *makespan*, is $Max\{\tau_k | k = 1, 2, ..., n\}$. Now, our goal is to find an optimal task

Table 1. Main notations and definitions

t_i	ith task of a job
n_i	ith computing node
X	A $m \times n$ matrix corresponding to a task assignment
p_{ij}	An element of X. It equals 1 if t_i is assigned to n_j; otherwise it equals 0
e_{ij}	Accumulative execution time for task t_i running on processor n_j
τ_k	The total computing time of node k for all tasks allocated to it
λ_i	Failure rate of the ith node
μ_i	Repair rate of the ith node
U_k	The length of the kth operation period
D_k	The length of the kth repair/replacement period
$B(\tau)$	$Prob.$(node is up, residual lifetime $> t$) at time τ
$A(\tau)$	Availability function of cluster

assignment X, under which we can maximize the system availability function $A(\tau)$ and minimize the value of *makespan*.

3.2 Residual Lifetime Modeling

In the reliability engineering [12], the expected residual lifetime is a random variable representing the remaining life of a system at time t. Considering one repairable node k of the heterogeneous cluster, each node has two-states and is assumed to undergo random failures with time elapsing independently. Simultaneously, each failure entails a random duration of repair before the node is put back into service. Also, we assume that the duration of the failing node is independent of the states of other nodes. Due to the random nature of the component failures, the Poisson process is an appropriate model to employ. Let $U_k, k \in N$ represents the length of the kth operation period, and let $D_k, k \in N$ represents the length of the kth repair/replacement time for the node as shown in Fig. 2. We denote $H(t)$ and $Y(t)$ as the probability distribution function of U_k and D_k, respectively, and λ and μ are the failure rate and the repair rate of the node respectively. As is known, $1/\lambda$ is the mean time to failure (MTTF) of the node while $1/\mu$ is the mean time to repair (MTTR) of the node. In our study, we assume that $H(t)$ is exponential distribution, then we have the lifetime distribution function of the node, $H(t) = 1 - e^{-\lambda t}$, and the survival function, $\overline{H}(t) = 1 - H(t) = e^{-\lambda t}$.

Let S_n denote the nth failure time, then $S_n = U_1 + \sum_{k=1}^{n-1}(U_k + D_{k+1})$. Obviously, the S_n sequence generates a delayed renewal process $N(t)$. The U_1 has the distribution function H. All other interarrival times have the same distribution function $H * Y$ with mean $1/\lambda + 1/\mu$, where * represents convolution. Given that the node is up at time t, the residual lifetime $W(t)$ represents as follows:

$$W(t) = S_{N(t)+1} - t. \tag{1}$$

The probability that node is up and residual lifetime is greater than ω, $\omega > 0$, at time t is given as:

$$B(t, t+\omega) = P(node\ is\ up, W(t) > \omega) = \overline{H}\,(t+\omega) + \int_0^t \overline{H}(t-x+\omega) dm\,(x)\,. \tag{2}$$

where $m(x) = \sum_{n=1}^{\infty}(H * Y)^{(n)}(x)$, and (n) is the n-fold convolution of $H * Y$ with itself. The proof of equation (2) can be seen in [12].

From the Key Renewal Theorem [13], we can give the equation:

$$B(\omega) = \lim_{t\to\infty} P(node\ is\ up, W(t) > \omega) = \frac{\int_{\omega}^{\infty} \overline{H}(x)dx}{1/\lambda + 1/\mu}. \tag{3}$$

with $\overline{H}(x) = e^{-\lambda t}$, we can calculate the probability:

$$B(\omega) = \frac{\mu e^{-\lambda\omega}}{\lambda + \mu}. \tag{4}$$

3.3 Objective Function

We define the model for each node according to equation (4), indexed by "k". Then for the node k, we have the probability that the node is up and its expected residual lifetime at time t is greater than τ_k which is given as follows:

$$B_k(\tau_k) = \frac{\mu_k e^{-\lambda_k \tau_k}}{\lambda_k + \mu_k}. \tag{5}$$

where τ_k is the total computing time of node k for all tasks allocated to it, and λ_k, μ_k are the failure rate and repair rate of node k respectively.

Now we define availability function $A(\tau)$ as the geometric mean of all $B_k(\tau_k)$, so $A(\tau)$ represents the average probability that every nodes' residual lifetime is longer than corresponding τ_k. Then, we have the equation:

$$A(\tau) = \sqrt[n]{\prod_{k=1}^{n} B_k(\tau_k)} = \sqrt[n]{exp(-\sum_{k=1}^{n} \lambda_k \tau_k) \prod_{k=1}^{n} \frac{\mu_k}{\lambda_k + \mu_k}} \tag{6}$$

Clearly, $A(\tau)$ reflects the cluster's availability of performing all tasks based on the node's residual lifetime property. We regard $A(\tau)$ as the objective function. Thus, our goal of scheduling is to find a task assignment $X = \{p_{ij} | 1 \leq i \leq m, 1 \leq j \leq n\}$ to make the value of $A(\tau)$ maximum and simultaneously reduce the *makespan* of cluster as much as possible. Without question, maximizing $A(\tau)$ is equivalent to minimizing the function $RLcost(\tau) = \sum_{k=1}^{n} \lambda_k \tau_k$, since λ_k and τ_k are constant for every node. Now, what we will do is to find a task assignment X, which can maximize the cluster performance as follows:

$$\begin{array}{l} Min\ RLcost(\tau) = \sum_{k=1}^{n} \sum_{i=1}^{m} \lambda_k p_{ik} e_{ik}; \\ Min\ makespan = max\{\sum_{i=1}^{m} p_{ik} e_{ik} | 1 \leq \lambda \leq n\}; \\ s.t. \begin{cases} \sum_{k=1}^{n} p_{ik} = 1, 1 \leq i \leq m \\ p_{ik} = 0\ or\ 1, \quad 1 \leq i \leq m, 1 \leq k \leq n \end{cases} \end{array}$$

It is a multi-object optimization problem (MOP). Finding a optimal solution of the problem is known as a NP-hard problem [14], since the optimizing objects conflict each other when optimizing them. Therefore, we develop a heuristic algorithm to find a trade-off solution which minimizes the function $RLcost(\tau)$ as well as the function *makespan*.

4 Scheduling Algorithm

In order to determine probability vector X, which corresponds to a task scheduling policy, we propose a heuristic algorithm to achieve a trade-off solution. In respect that our preferred goal is to maximize the availability, our heuristic algorithm finds the minimum value of $RLcost(\tau)$ first and then tries to satisfy the need of *makespan* constraint. The full algorithm is depicted in Table. 2.

Table 2. Trade-off algorithm

1. $p_{ij} \leftarrow 0$, for all i and j;
2. Arbitrarily order the tasks of a set $T = \{t_1, t_2, ..., t_m\}$;
3. Arbitrarily order the computing nodes of a set; $N = n_1; n_2, ..., n_n$
4. Create a set ST (ST will hold the tasks that have been allocated to nodes);
5. While ($T \neq \phi$) do
6. Get task t_i from T in turn, move it into ST;
7. Find node n_j in N, whose value of is the minimization among all of nodes';
8. if ($\sum_{s=1}^{i-1} p_{s1}e_{s1} = \sum_{s=1}^{i-1} p_{s2}e_{s2} = ... = \sum_{s=1}^{i-1} p_{sn}e_{sn}$) then Goto 10
9. if ($(\sum_{s=1}^{i-1} p_{sj}e_{sj} + e_{ij}) > max(\sum_{s=1}^{i-1} p_{s1}e_{s1}, ..., \sum_{s=1}^{i-1} p_{sn}e_{sn})$) then
   $j \leftarrow \{k|(\sum_{s=1}^{i-1} p_{sk}e_{sk} + e_{ik}) = min(\sum_{s=1}^{i-1} p_{s1}e_{s1} + e_{i1}, ..., \sum_{s=1}^{i-1} p_{sn}e_{sn} + e_{in})\}$;
10. $p_{ij} \leftarrow 1$;
11. End While;
12. Output X.

The algorithm has m rounds calculation. In each round, Pareto efficient solution is chosen and reserved for the next turn calculation, so the algorithm converges at the overall optimal solution [15].

At first, for every task t_i, step 7 endeavors to minimize the value of corresponding $RLcost(\tau)$ through finding the node n_j which has the minimum value of $\lambda_j e_{ij}$. In succession, step 8 and 9 decide if the choice of n_j can make all allocated tasks' execution time (*makespan*) minimum. If it does, we believe that n_j is the best choice for task t_i. Otherwise, we have to find a node $n_j^{'}$, which corresponds to the minimum *makespan* if allocate t_i to it. What must be noticed is that *makespan* here is the executing time for all the tasks that has been allocated. Note that step 8 ensures getting the maximum availability when all nodes *makespan* are equal. After doing that, node $n_j^{'}$ will be selected, and then task t_i will be ultimately allocated to it. It's worth noting that $n_j^{'}$ and n_j may be one and the same at some occasions.

Time complexity of this algorithm is $O(m^3n)$. However, it can be improved by sacrificing some space complexity. If we conserve the value of $\sum_{s=1}^{i-1} p_{sk}e_{sk}$ in memory for the next round computing at ith round, i.e. if we can use the ith round computing result to calculate the value of $(i+1)$th round, time complexity of the algorithm will be reduced to $O(m^2n)$.

5 Experiment and Discussion

In this section, we will use some experiential data to compute the values of $A(\tau)$ and *makespan* by a simple example. In this example, the repairable heterogeneous cluster has three computing nodes and their failure rates and repair rates are shown as Table 3. The matrix of execution times is given in Fig. 3, whose entity e_{ij} represents the execution time of t_i on node n_j. We must point out that all the data we use in our simulation are not actual figures but approximate values choosed empirically. It does not affect the simulation results, because the

$$E = \begin{bmatrix} 0.3000 & 0.1000 & 0.5000 \\ 0.7000 & 1.2000 & 0.6000 \\ 0.2000 & 0.2000 & 0.5000 \\ 0.3000 & 0.6000 & 0.8000 \\ 0.9000 & 1.0000 & 1.3000 \\ 1.2000 & 0.9000 & 1.7000 \end{bmatrix}$$

Fig. 3. Matrix of execution time for all tasks

algorithm is independent of the input figures. Our aim is only to evaluate the algorithm's performance for the proposed model.

The trade-off algorithm introduced above is running with the parameters and we achieve a task assignment X_1, as shown in Fig. 4(a). We calculate the *makespan* and $A(\tau)$ using the model introduced in section 3 under the assignment X_1. In order to identify the validity of the algorithm, we also figure out the extremes of *makespan* and $A(\tau)$ respectively. The former, X_2 (Fig. 4(b)), is achieved by maximizing the availability without considering *makespan*. And the latter, X_3 (Fig. 4(c)) is achieved by exhaustively enumerating all task assignment to find the minimum *makespan* without considering $A(\tau)$. See Table 4.

$$X_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \text{(a)} \qquad X_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{(b)} \qquad X_3 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{(c)}$$

Fig. 4. Tasks assignment with different preferences . (a) The trade-off solution (b) Maximizing the availability without considering *makespan* (c) Minimizing the *makespan* without considering availability.

Compared with X_2, the assignment X_1 reduces the availability of cluster 0.058% and improves the *makespan* about 15%. Contrarily, X_1 raises the availability 0.012% and degrades the *makespan* about 42% than X_3. We take note of that the variation range of availability is very small and reverse happens to *makespan*. The reason is that the failure rate of cluster we choosed is very small, i.e. the cluster is reliable enough and it is difficult to improve its availability, just as cluster shows in most practical situations. On the contrary, *makespan* is interrelated with the number of tasks and their execution time, which result in

Table 3. Failure rate and repair rate for all computing nodes

	n_1	n_2	n_3
λ	0.0050	0.0030	0.0070
μ	1.1900	1.2410	1.1100

Table 4. Simulation results

	$RLcost(\tau)$	$makespan$	$A(\tau)$
X_1	0.0160	1.7000	0.8665
X_2	0.0120	2.0000	0.8670
X_3	0.0167	1.2000	0.8664

the distinct fluctuation of the execution time for cluster under different task assignment.

Furthermore, we address the effect on the tradeoff algorithm by varying the numbers of nodes and tasks, and compare it with the algorithm which only maximizes availability without considering *makespan*. We suppose that the value of availability and *makespan* computed by using the tradeoff algorithm are A_1 and m_1 respectively, and corresponding values are A_2 and m_2, which computed by using the algorithm without taking into account the *makespan*, respectively. We denotes A_1/A_2 as the ΔA and $(m_2 - m_1)/m_2$ as Δm. Thus, we can acquire the variety of ΔA and Δm, as Fig. 5 and Fig. 6 shows.

Fig. 5 illustrates the variation of ΔA and Δm with the tasks number increasing from 1 to 1000, while there are 3 computing nodes. Fig. 6 illustrates the variation of ΔA and Δm with the nodes number increasing from 1 to 50, while there are 1000 tasks.

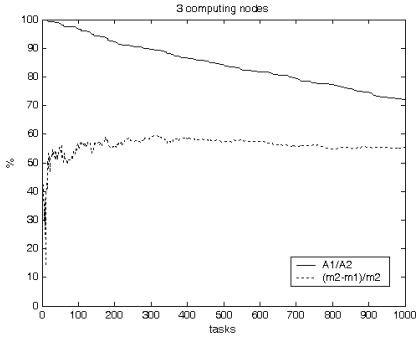

Fig. 5. 3 nodes, ΔA and ΔM vary with the number of tasks increasing

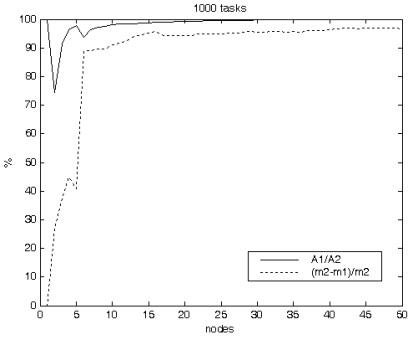

Fig. 6. 1000 tasks, ΔA and ΔM vary with the number of nodes increasing

6 Conclusion and Future Work

The residual lifetime is a very important factor for cluster availability. In this paper, we have presented task allocation to maximize availability with the *makespan* constraint. We have analyzed the residual lifetime of the computing node and addressed the stochastic model for cluster availability. In order to incorporate availability and *makespan* into task scheduling, we have proposed a trade-off scheduling algorithm. Our algorithm can improve the performance in availability as much as

possible with the *makespan* constraint. At last we have given some experimental results and simple analysis about that.

In the future, we will improve our availability model through considering the effects of tasks inter-communication channel's lifetime model which have not been involved in this paper. By applying the model and algorithm to the real cluster systems, we can evaluate its performance and improve it. We will also make use of existing algorithms, such as genetic algorithm and evolutionary algorithm, to find the solution of our model. Based on those, a more efficient algorithm would be proposed, since the algorithm represent here is not efficient enough for large scale clusters.

References

1. Alhamdan, A.A.: Scheduling Methods for Efficient Utilization of Clusters Computing Environments. PhD thesis, University of Connecticut (2003)
2. Chu, W., Holloway, L., Lan, M.T., Efe, K.: Task allocation in distributed data processing. IEEE Mag. Computer 13, 57–69 (1980)
3. Shatz, S.M., Wang, J.P., Goto, M.: Task allocation for maximizing reliability of distributed computer systems. IEEE Trans. Comput. 41(9), 1156–1168 (1992)
4. Xie, T., Qin, X.: Stochastic scheduling with availability constraints in heterogeneous clusters. IEEE Proc. Cluster Computing 9, 1–10 (September 2006)
5. Schmidt, G.: Scheduling with limited machine availability. International Computer Science Institute Technical Report (TR-98-036) (1998)
6. Ali, S., Maciejewski, A.A., Siegel, H.J., Kim, J.K.: Measuring the robustness of a resource allocation. IEEE Trans. Parallel Distrib. Syst. 15(7), 630–641 (2004)
7. Shestak, V., Smith, J., Siegel, H.J., Maciejewski, A.A.: A stochastic approach to measuring the robustness of resource allocations in distributed systems. In: ICPP '06: Proceedings of the 2006 International Conference on Parallel Processing, Washington, DC, USA, pp. 459–470. IEEE Computer Society Press, Los Alamitos (2006)
8. Dogan, A., Özgüner, F.: Reliable matching and scheduling of precedence-constrained tasks in heterogeneous distributed computing. In: ICPP '00: Proceedings of the 2000 International Conference on Parallel Processing, Washington, DC, USA, p. 307. IEEE Computer Society Press, Los Alamitos (2000)
9. Topcuouglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. 13(3), 260–274 (2002)
10. Hariri, S., Raghavendra, C.S.: Distributed functions allocation for reliability and delay optimization. In: ACM '86: Proceedings of 1986 ACM Fall joint computer conference, Los Alamitos, CA, USA, pp. 344–352. IEEE Computer Society Press, Los Alamitos (1986)
11. Srinivasan, S., Jha, N.K.: Safety and reliability driven task allocation in distributed systems. IEEE Trans. Parallel Distrib. Syst. 10(3), 238–251 (1999)
12. Aven, T., Jensen, U.: Stochastic Models in Reliability. Stochastic Modelling and Applied Probability. Springer, Heidelberg (1999)
13. Kallenberg, O.: Foundations of Modern Probability. Springer, Heidelberg (2001)
14. Papadimitriou, C.H.: Comuputational Complexity. Addison-Wesley, Reading (1993)
15. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)