Digital Communications and Networks xxx (2017) 1–9

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.elsevier.com/locate/dcan

PRECESION: Progressive recovery and restoration planning of interdependent services in enterprise data centers

Ibrahim El-Shekeil *, Amitangshu Pal, Krishna Kant

Computer and Information Sciences, Temple University, Philadelphia, PA, 19122, USA

ARTICLE INFO

Keywords:
Progressive restoration planning
Enterprise data center
Genetic algorithm
Integer linear program
Multi-layer networks

ABSTRACT

The primary focus of this paper is to design a progressive restoration plan for an enterprise data center environment following a partial or full disruption. Repairing and restoring disrupted components in an enterprise data center requires a significant amount of time and human effort. Following a major disruption, the recovery process involves multiple stages, and during each stage, the partially recovered infrastructures can provide limited services to users at some degraded service level. However, how fast and efficiently an enterprise infrastructure can be recovered depends on how the recovery mechanism restores the disrupted components, considering the inter-dependencies between services, along with the limitations of expert human operators. The entire problem turns out to be NP-hard and rather complex, and we devise an efficient meta-heuristic to solve the problem. By considering some real-world examples, we show that the proposed meta-heuristic provides very accurate results, and still runs ~ 600–2800 times faster than the optimal solution obtained from a general purpose mathematical solver [1].

1. Introduction

Disruptions in enterprise data centers may occur as a result of hardware failures, operating system or software failures, intrusions, virus outbreaks, or natural disasters. For example, the 2011 Japan earthquake and tsunami damaged major data centers in Tokyo [2], and the 2012 Hurricane Sandy in the USA affected some data centers in New York due to flooding [3]. Other examples include storms and lightning that took down Google's St. Ghislain data center operations for five days in 2015 [4], and technical hiccups that affected the services of the Bank of America [5] and Amazon [6] centers for four-six days. Such scenarios or disruptions may also arise as a result of the relocation and upgrade of a data center at different sites, which require proper pre-move planning and expertise. Depending on the scale, such disruptions can either be partial, impacting only some applications, or full, impacting the entire data center. When such disruptions occur, they result in significant downtime, which may lead to a substantial financial and legal impact. According to a recent study by the Ponemon Institute, a data center outage can cost an average of $5600 per minute [7].

In light of the above, there is a growing need to optimize the post-disruption recovery and restoration process for enterprise data centers. A complete post-disruption restoration process for a large data center requires multiple stages, as backup resources are brought to the field and installed, which sometimes requires between a few weeks and several months [9]. Within this entire recovery stage, the partially restored infrastructures will still have to operate in a degraded manner and provide a partial level of service for clients.

A key design challenge of the restoration plan is to support partial business continuity, which allows applications to *progressively* come back online following failures or disruptions. Fig. 1 shows such a data center recovery process, where the critical applications or services are gradually recovered over a duration of approximately 3020 hours. Notice that the recovery processes of all services cannot be started simultaneously, because the services in an enterprise data center are typically inter-dependent. The sequencing of data center services that are gradually recovered will have a direct impact on the effectiveness of the restoration process, especially after a large-scale disaster where multiple data center services are down. Thus, the decision regarding the recovery sequence for these services plays an important role in minimizing losses by bringing the most critical applications back online.

Another challenge is to determine the number of expert workers required with the right skills to obtain the optimal uptime. The sequencing of the restoration of data center services and hiring of the appropriate number of workers with suitable skills are coupled problems, which must be solved together to achieve the optimal uptime. However, in this paper we focus on solving the sequencing problem, and assume that the number of workers and their skills are given.

In this paper, our main contributions are as follows. We characterize

* Corresponding author.
E-mail addresses: ielshekeil@temple.edu (I. El-Shekeil), amitangshu.pal@temple.edu (A. Pal), kkant@temple.edu (K. Kant).

https://doi.org/10.1016/j.dcan.2017.08.001
Received 26 February 2017; Accepted 4 August 2017
Available online xxxx

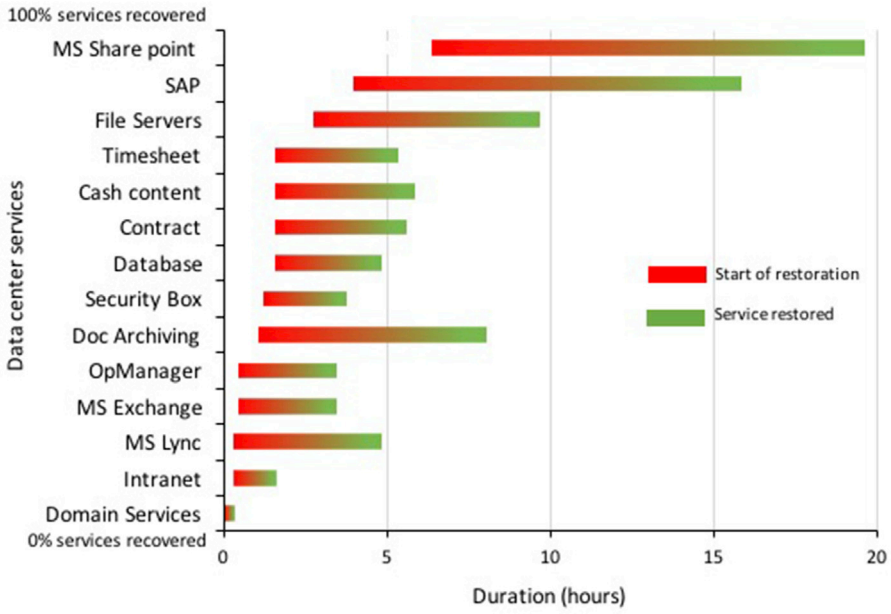

Fig. 1. The recovery and restoration timeline of data center services. **Service recovery progression from start-to-finish is shown in a red-to-green color scale.**

data center users depending on their service requirements, and divide them into different types. We focus on the post-disruption progressive data center recovery problem, with the objective of serving the maximum number of different data center user request types throughout the entire recovery process. Repairing the disrupted services requires a long time period, as well as human resources or workers. In a large data center, the availability of human resources with the desired expertise varies over time. Various data center services require different skill-sets. However, larger data centers are not populated by people with the expertise to restore all services [10,11]. Recovering different services may require different time to perform manual configuration and the actual restoration, which makes the decision process even more complex. Because the services in a data center are often interdependent, it is nontrivial to plan, evaluate and compare different recovery decisions and choose the best one. Such a complicated service recovery process involving heterogeneous worker skill sets and availability is quite different from other job scheduling problems [12–15], which have the goal of minimizing the total time to complete a set of jobs in the presence of a certain number of workers. The entire problem is not only NP-hard, but also rather complex. We propose a genetic algorithm based meta-heuristic to solve it, and show that the proposed genetic algorithm based solution is highly accurate compared to the optimal solution. This is an extension of our previous paper [16], where we maximized the up-time of different data center services considering the inter-dependencies between them and the data center servers. In contrast, in this paper, our main focus is on enhancing satisfaction levels for different types of user requests, while considering the interdependencies of various services and types of user requests with human related constraints and expertise.

The rest of this paper is organized as follows. Sections 2 and 3 address the overall problem formulation, including its complexity and the proposed meta-heuristic solution. Section 4 reports results based on real data obtained from enterprise data center environments. Section 5 discusses related work. Finally, Section 6 concludes the paper and discusses potential future work.

2. Problem description

2.1. Preliminaries and notations

The problem can be formulated using a three layer interdependency framework. Layer 1, or the *user layer,* consists of different types of user requests. Layer 2, or the *service layer*, consists of the set of services that the enterprise provides, and Layer 3, or the *server layer,* consists of the servers that need to be restored to bring back the services, as shown in Fig. 2. Each type of user request is satisfied by one or more services, which must be up in order to satisfy the corresponding type of user requests. The services depend on one or more servers to run, and thus thus the services cannot be brought online until the servers on which they depend are restored. For example, in Fig. 2, service 1 depends on servers 1 and 2; and thus, service 1 cannot be up and running until servers 1 and 2 are fully restored. Similarly, user requests of type 1 cannot be satisfied

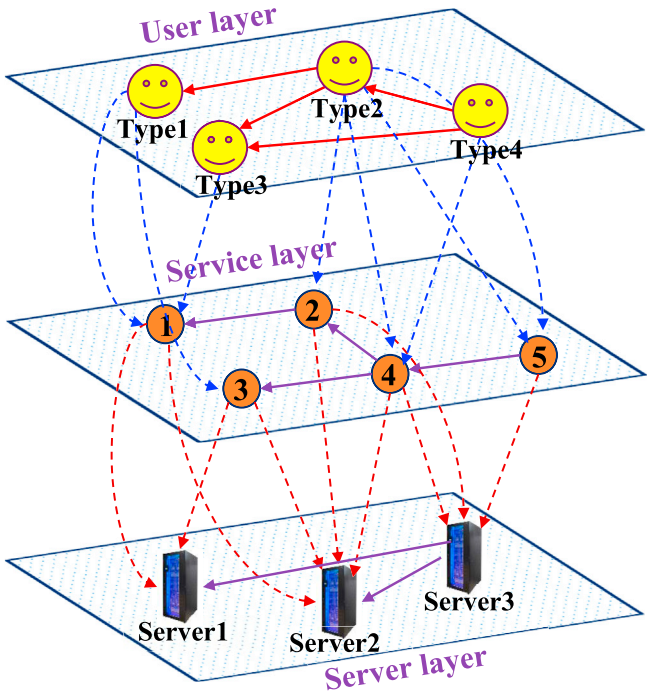

Fig. 2. Dependency graph between the server layer, service layers and user layer.

until services 1 and 3 are brought online. Such dependencies across the layers can be modeled as *inter-layer dependencies*. Table 1 shows a concrete example of types of user requests in an enterprise and the corresponding services and servers that they depend on. Assume that H_{ua} denotes whether user requests of type u are satisfied by the service a or not. Q_{ai} denotes whether the service a is dependent on the server i or not. Thus, according to Fig. 3, $H_{11} = H_{13} = 1$ as user requests of type 1 are satisfied by services 1 and 3. Similarly, $Q_{11} = Q_{12} = 1$, as service 1 depends on servers 1 and 2.

On the other hand, in layers 1, 2 and 3, there are dependencies between different types of user request, services, and servers, which we define as i*ntra-layer dependencies*. For example, the email services, human resources, and SharePoint depend on the DNS and Active Directory services, and thus these two services must be restored first before restoring others. In the server layer, the front-end web server depends on the application server, and the application server depends on the database server. If the database server is down, then the application and web servers cannot provide services. In the user layer, in order to send email, access to the network resources and authentication services needs to be up and running. Assume that S_{uv} denotes whether or not user requests of type a are dependent on user requests of type b. Similarly, P_{ab} and O_{ij} denote the dependencies between services and servers. Thus, in Fig. 3, $S_{21} = S_{23} = S_{42} = S_{43} = 1$, $P_{21} = P_{42} = P_{43} = P_{54} = 1$ and $O_{31} = O_{32} = 1$.

Assume that time is divided into *slots,* with the current slot denoted by t. Let [0, T] denote the time horizon, where T is the total time (in slots) in which all primary servers are recovered, and $0 \le t \le T$. We assume that there are M services that need to be activated, which run on a total of N servers. Assume that there are $\mathscr{U}$ different types of users. The server restoration process consists of two steps. The first is the *exclusive* stage, when an expert worker corresponding to a particular server needs to begin the restoration tasks in a dedicated fashion. The next phase is the *shared* phase, which requires some infrequent monitoring. We thus assume that a worker (or expert) becomes free after the exclusive stage, as the remaining stage can mostly be taken care of by common operators without expert intervention. Let us assume that x_i^t represents whether or not a server i is treated at or before time slot t. Similarly, e_i^t represents whether or not its exclusive stage complete, and z_i^t represents whether or not it is completely restored. For example, in Fig. 3, $x_1^t = 1$ for $t \ge 1$, $e_1^t = 1$ for $t \ge 3$, and $z_1^t = 1$ for $t \ge 5$. Similarly, we assume that y_a^t and g_u^t denote whether or not service a and user u are restored at or before time slot t, respectively. For the problem formulation, we assume that the exclusive time for server i constitutes l_i units, and the entire expected restoration time is L_i units.

We assume that the number of expert workers is significantly lower than the number of servers that need to be restored, i.e., $W \ll N$. Thus, workers must rotate in order to bring the services back online. Currently, large data centers are not populated by people who are "jacks of all trades." Instead, specialized expertise in large data centers is a growing problem, as applications and configurations are becoming very complex [10,11]. Workers do not have the expertise to work on all servers, and thus they need to be assigned one after another to progressively restore the servers, depending on their availability and expertise.

Table 1
Dependencies between user request types, servers, and services.

Users	Description	Enterprise services	Necessary servers
Type 1	Authenticate	MS Active Directory	Domain controller
	Access to network resources	DNS	DNS server
Type 2	Send emails to internal users	Microsoft Exchange	Internal email servers
Type 3	Run payroll	Human Resources	Payroll server and employee database
	Access Project	Microsoft Project	Project application and database
Type 4	Send emails to external users	Microsoft Exchange	Email forwarding servers
	Access documents in SharePoint	Microsoft SharePoint	SharePoint application and database

2.2. Problem formulation

Next, we formulate our optimization problem framework, named *progressive recovery and restoration (PRECESION)* planning. The necessary notations are listed in Table 2. The goal of PRECESION is to serve the maximum number of different types of user requests during the recovery process, i.e.,

$$\max \sum_{u=1}^{U} \sum_{t} \gamma_u g_u^t \tag{1}$$

where $\sum_t \gamma_u g_u^t$ represents the total time for which the user request type u is available. For example, assume that $T = 100$, and that the user request type u was satisfied at time slot 10. Thus, the user request remains available from time slot 10 onwards, and the total availability time of u is 90. Here, γ_u represents the weight of the type of user request u, which is proportional to the number of users typically falling into that group. We next describe the constraints, as follows:

2.2.1. Dependency constraints

Constraint (2) models the intra-layer dependencies, and states that if a user request u is dependent on a set of user request types $D(u)$, then $D(u)$ needs to be available before u becomes available. This constraint becomes nonlinear due to the presence of the *product* operator. However the constraint can be linearized by incorporating the *summation* operator, as shown in equation (2). This is explained as follows. Assume that p, q_1, q_2, q_3 are three binary variables, where p cannot be 1 if any one of q_1, q_2, q_3 is 0. Thus, $p \le q_1 q_2 q_3$, which can be linearized by writing $3p \le q_1 + q_2 + q_3$. In both cases, p can be 1 iff q_1, q_2, q_3 becomes 1.

Similarly, constraint (3) models user request-service dependencies, and states that the user request type u cannot be satisfied without restoring the set of services $D(u)$. Constraint (4) models the intra-layer dependencies, and states that if a service a is dependent on a set of services $D(a)$, then $D(a)$ needs to be ON before a can be made ON. Constraint (5) models the service-server dependencies, and state that the service a cannot be ON without restoring the set of servers in $D(a)$. Similarly, constraint (6) models the dependencies between the servers.

$$g_u^t \le \prod_{v \in D(u)} g_v^t \Rightarrow \left(\sum_{v=1}^{U} S_{uv} \right) . g_u^t \le \sum_{v=1}^{U} S_{uv} . g_v^t \ \forall u, t \tag{2}$$

$$g_u^t \le \prod_{a \in D(u)} y_a^t \Rightarrow \left(\sum_{a=1}^{M} H_{ua} \right) . g_u^t \le \sum_{a=1}^{M} H_{ua} . y_a^t \ \forall u, t \tag{3}$$

$$y_a^t \le \prod_{b \in D(a)} y_b^t \Rightarrow \left(\sum_{b=1}^{M} P_{ab} \right) . y_a^t \le \sum_{b=1}^{M} P_{ab} . y_b^t \ \forall a, t \tag{4}$$

$$y_a^t \le \prod_{i \in D(a)} z_i^t \Rightarrow \left(\sum_{i=1}^{N} Q_{ai} \right) . y_a^t \le \sum_{i=1}^{N} Q_{ai} . z_i^t \ \forall a, t \tag{5}$$

$$x_i^t \le \prod_{j \in D(i)} z_j^t \Rightarrow \left(\sum_{j=1}^{N} O_{ij} \right) . x_i^t \le \sum_{j=1}^{N} O_{ij} . z_j^t \ \forall i, t \tag{6}$$

2.2.2. Timing constraints

Constraints (7)–(10) model the start and end times of the server restoration. Constraints (7)–(8) ensure that the exclusive time of server i

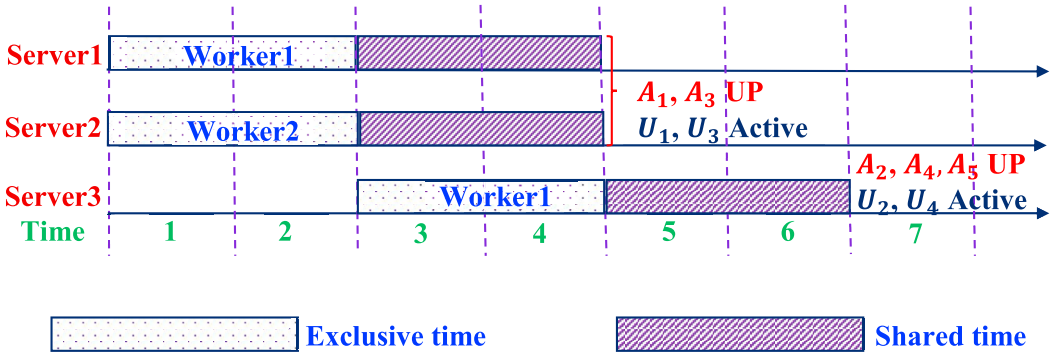

Fig. 3. Timing diagram of the worker to server assignment. $A_1 - A_5$ denote all the services, whereas $U_1 - U_4$ denote different types of user requests.

Table 2
Table of notations.

Indices		
a, b	≜	Index for services (1, …, M)
i, j	≜	Index for servers (1, …, N)
t	≜	Index for time units (1, …, T)
w	≜	Index for workers (1, …, W)
u, v	≜	Index for user request type (1, …, $\mathscr{U}$)
Variables		
x_i^t	≜	Whether or not restoration of server i has been started by a worker on or before time slot t
e_i^t	≜	Whether or not the exclusive part of server i is finished on or before time slot t
g_u^t	≜	Whether or not user request type u is satisfied at time slot t
γ_u	≜	Weight of the user request type u
z_i^t	≜	Whether or not server i is up at time slot t
y_a^t	≜	Whether or not service a is up at time slot t
l_i	≜	Expected exclusive time of server i
L_i	≜	Expected restoration time of server i
O_{ij}	≜	Whether or not server i is dependent on server j
P_{ab}	≜	Whether or not service a is dependent on service b
Q_{ai}	≜	Whether or not service a is dependent on server i
S_{uv}	≜	Whether or not user request type u is dependent on user request type v
H_{ua}	≜	Whether or not user request type u is dependent on service a
$D(a)$	≜	Set of servers or services on which service a depends
$D(i)$	≜	Set of servers on which server i depends
$\mathbb{A}_{wi}$	≜	Whether or not server i is assigned worker w
R_{wi}	≜	Whether or not worker w can restore server i

is completed after l_i time slots. Constraints (9)–(10) ensure that server i is fully restored after L_i time units.

$$x_i^t = e_i^{t+l_i} \ \forall i, \forall t = (1, 2, \ldots, T - l_i) \tag{7}$$

$$\sum_{t=1}^{T-l_i} x_i^t = \sum_{t=1}^{T} e_i^t \ \forall i \tag{8}$$

$$x_i^t = z_i^{t+L_i} \ \forall i, \forall t = (1, 2, \ldots, T - L_i) \tag{9}$$

$$\sum_{t=1}^{T-L_i} x_i^t = \sum_{t=1}^{T} z_i^t \ \forall i \tag{10}$$

2.2.3. Worker assignment constraints

Constraint (11) states that worker w is assigned to restore server i if they have the expertise to restore it. Constraint (12) ensures that all servers are assigned a worker, and thus restored by the end of the time scale T. Constraint (13) states that if worker w is assigned to restore server i and j, then their exclusive times do not overlap.

$$\mathbb{A}_{wi} \leq R_{wi} \ \forall i, \forall w \tag{11}$$

$$\sum_{w} \mathbb{A}_{wi} = 1 \ \forall i \tag{12}$$

$$x_i^t - e_i^t + x_j^t - e_j^t \leq 1 + \mathbb{M}\left(2 - \mathbb{A}_{wi} - \mathbb{A}_{wj}\right) \ \forall t, \forall w, \forall i \neq j \tag{13}$$

where $\mathbb{M}$ is a large number that is greater than the maximum value of the left-hand side. The intuition behind equation (13) is as follows. Assume that worker w is assigned to restore servers i and j. Thus, the right-hand side of equation (13) becomes 1. Now, let us assume that at any time slot t, the exclusive time for restoring servers i and j overlap, i.e., the restoration of server j starts before the exclusive times for server i is complete. This occurs if $x_i^t = x_j^t = 1$ and $e_i^t = e_j^t = 0$. Thus, $x_i^t - e_i^t + x_j^t - e_j^t = 2 \leqslant 1$, and the inequality in constraint (13) does not hold.

2.2.4. Other constraints

Constraint (14) states that if service a is ON at time slot, t then it will remain ON at all future time steps. Similarly, if a server is restored, then it remains active for all future time steps, as modeled in equation (15). Constraints (16)–(17) state similar constraints, corresponding to the server restoration start time and exclusive time, respectively. Constraint (18) states that if a user request of type u is satisfied and becomes available at time t, then it will remain available for all future time steps.

$$y_a^{t+1} \geq y_a^t \ \forall a, \forall t = (1, 2, \ldots, T - 1) \tag{14}$$

$$z_i^{t+1} \geq z_i^t \ \forall i, \forall t = (1, 2, \ldots, T - 1) \tag{15}$$

$$x_i^{t+1} \geq x_i^t \ \forall i, \forall t = (1, 2, \ldots, T - 1) \tag{16}$$

$$e_i^{t+1} \geq e_i^t \ \forall i, \forall t = (1, 2, \ldots, T - 1) \tag{17}$$

$$g_u^{t+1} \geq g_u^t \ \forall u, \forall t = (1, 2, \ldots, T - 1) \tag{18}$$

Theorem 1. *The PRECESION problem is NP-hard.*

Proof 1. *We first define the minimum latency set cover problem (MLSC), which is known to be NP-hard. Let $J = \{J_1, J_2, \ldots, J_m\}$ be a set of jobs to be processed by a factory. A job J_i has non-negative weight ω_i. Let $T = \{t_1, t_2, \ldots, t_n\}$ be a set of tools. Job J_j is associated with a non-empty subset $S_j \subseteq T$. Once the entire tool subset S_j has been installed, job J_j can be processed instantly. The factory can install a single tool at each time unit. The problem is to determine the order of tool installation so that the weighted sum of job completion times is minimized.*

We now reduce the MLSC to the PRECESION problem.We define one instance of the MLSC problem where $\omega_i = 1, \forall i$. With this, we define an instance of the PRECESION problem as follows. We assume that there are no dependencies between user request types, services, and servers.

Furthermore, assume that there is just one worker, who has the expertise to restore all the servers. The restoration times of all servers are one, with the shared time assumed to be zero, i.e., $l_i = L_i = 1$, $\forall i$. In addition, assume that $M = \mathscr{U}$ and $H_{uu} = 1$, $\forall u$. This reduction transforms an MLSC problem instance into a PRECESION instance.

2.3. An illustrative example

Let us consider an example, with $\mathscr{U} = 4$, $M = 5$, and $N = 3$. The intra and inter-layer dependencies are shown in Fig. 2. We use the AMPL solver [17] for solving the optimization problem. Fig. 3 shows the timing diagram for the restoration process, obtained by solving the problem (1). We assume that there are two workers, and worker 1 can restore servers 1 and 3, whereas worker 2 can restore servers 1 and 2. The exclusive times and shared times of all the servers are assumed to consist of two units. From Fig. 3, we can observe that all the services are dependent on A_1 and A_3. Thus servers 1 and 2 are first restored to run these two services. Server 3 is then restored, in order to run the other services. The user request types U_1 and U_3 are satisfied after bringing up the services A_1 and A_3 at time slot 4, whereas at time slot 6 other types of user requests are also satisfied, thus restoring other services.

3. A genetic algorithm based heuristic to solve the PRECESION problem

Given the NP-hardness and complexity of the problem, we propose a genetic algorithm based meta-heuristic to solve it. A genetic algorithm maintains a population of candidate solutions. Each candidate solution in the population is encoded into a structure called the *chromosome*. Each chromosome is assigned a *fitness value*, which represents the quality of the candidate solution. Better-fitted chromosomes have higher chances of surviving to the next generation. The number of chromosomes per generation is constant. As for natural life, offspring chromosomes are obtained from parent chromosomes by mainly using two operators, crossover and mutation. Some other chromosomes simply survive unaltered, while others die off. The different steps of the entire genetic algorithm are described as follows.

3.1. Chromosome structure and fitness value calculation

We define a chromosome structure by considering the sequence in which the servers need to be restored, i.e., we define a chromosome as a vector $(c_1, c_2, \ldots, c_N)$, where c_i represents the i-th server (or gene). Thus, the genes of a chromosome are the servers, and a chromosome defines the sequence of servers. We assume that there are $\mathscr{M}$ chromosomes in a *mating pool*. The fitness value of each chromosome is determined as follows. A chromosome sequence determines the order in which the servers need to be restored. We use this sequence to assign the workers to the servers, depending on their expertise and availability, as mentioned in the next paragraph, and to determine the overall satisfaction levels of different types of user requests, as obtained from equation (1), which is considered as the fitness value corresponding to that chromosome.

For the initial assignment of workers, we construct a bipartite graph of N workers and servers as follows. First, take the sequence of servers and assign them different weights depending on their precedence. For example, if the server sequence is given by S_1, S_2, and S_3 then their corresponding weights can be 3, 2, and 1, respectively. We next construct N worker-nodes, by setting W nodes and $N - W$ dummy vertices. If worker w has the expertise to restore server i, then the weight corresponding to that edge is equal to the weight of server i. All the other edges are assigned a value of 0. In this bipartite graph, we then run a maximum matching algorithm, such as the Hungarian scheme [18], to assign the workers to their corresponding servers.

After the initial assignment, the workers become free following the exclusive time of the corresponding server. Once a worker is free, they are assigned to the next possible server that can be restored (depending on those already restored) in the sequence based on their expertise. This process continues until all servers are finally restored.

3.2. Initial mating pool generation

Initially, the mating pool is generated randomly, considering the fact that chromosomes are generated to satisfy the *precedence constraints* or dependency relations between the servers. For example, in Fig. 3(a), 3→2→1 will be an invalid chromosome structure, as server 3 depends on servers 1 and 2. Thus, server 3 cannot be treated until and unless the other two servers are completely restored. We thus describe the chromosome generation process in the initial mating pool using the example in Table 3. Assume that Table 3 shows the precedence/dependency relations among the servers, which we define as the *server dependency matrix*.

Servers 1 and 2 do not depend on the other servers, i.e., all rows corresponding to these two servers are 0. Server 3 depends on server 1, server 4 depends on servers 1 and 2, and server 5 depends on servers 3 and 4. Initially, the chromosomes must be generated such that these dependency relations are maintained. To do this, we define the *candidate server set (CSS)* as the server set with all 0 rows, which consists of servers 1 and 2 in the case of Table 3. We next choose one of the servers from the CSS randomly, and this server becomes the first gene in the chromosome. We next remove the row and column corresponding to that server from Table 3. We then construct the CSS with all 0 rows from the remaining dependency matrix, and then randomly choose the next gene from the CSS. This process is repeated to generate all the genes of a chromosome. The process is then repeated further to generate all the chromosomes in the initial mating pool. This ensures that the chromosomes in the initial mating pool are consistent with the precedence relation.

3.3. Selection process

We adopt the well-known *elitism* selection mechanism, where the $\mathscr{M}_e < \mathscr{M}$ best chromosomes are placed directly in the next generation. This ensures that the best chromosomes (or solutions) in a generation are not lost in future generations. The remainder of the $\mathscr{M} - \mathscr{M}_e$ chromosomes are chosen using the *roulette wheel* selection procedure (as determined by their fitness values) to take part in crossover and mutation to produce offspring chromosomes. Notice that the elite chromosomes also take part in the crossover and mutation processes to produce offspring chromosomes in the next generation.

3.4. Crossover operation

For the crossover operation, we choose two chromosomes from the mating pool with probabilities proportional to their fitness values. In the following, we describe a two-point crossover, although in general an n-point crossover can also be used. For a two-point crossover, we first randomly generate a *cutting-point*. Assuming that the cutting point is c, for the first chromosome we retain the first c genes, and remove the rows and columns corresponding to these c genes from the dependency matrix. We next generate the CSS for the $c+1$-th gene. If the $c+1$-th gene of the second chromosome is in the CSS, then we replace the $c+1$-th gene of the first chromosome with that of the second. Otherwise, we randomly choose a server from the CSS for the $c+1$-th gene. We follow the same

Table 3
An illustration of a server dependency matrix.

	S_1	S_2	S_3	S_4	S_5
S_1	0	0	0	0	0
S_2	0	0	0	0	0
S_3	1	0	0	0	0
S_4	1	1	0	0	0
S_5	0	0	1	1	0

procedure for the other genes (from $c+2$ onwards), and repeat the same for the second chromosome.

3.5. Mutation operation

For the mutation operation, we choose a chromosome randomly, and also randomly choose a cutting-point c. We retain the first c genes, generate the CSS, and randomly choose a server from the CSS for the $c+1$-th gene. This procedure is followed for the remaining genes of the chromosome. The proposed crossover and mutation operations ensure that the selection of the chromosomes in the subsequent mating pools is consistent with the precedence relation.

The algorithm stops when the best solution of a generation does not significantly improve for a fixed number of consecutive iterations or a large predefined number of iterations is reached. When the stopping criterion is reached, the algorithm chooses the chromosome/solution with the highest fitness value.

4. Experimental results

4.1. Validating the accuracy of the genetic algorithm

We first validated the accuracy of the proposed genetic algorithm compared with the optimal solution obtained from the PuLP solver [1], which is a library for the Python scripting language to solve mathematical problems. We synthetically generated a scenario with five types of user requests, 17 services and 24 servers for this purpose. We modeled the dependencies artificially, such that on average a service depends on 1.7 services, and 1.4 servers, whereas each server depends on 1.6 other servers. On the other hand, each type of user requests depends on one type of user requests and four services. We approximated the servers storage size to be (200–300) gigabytes for application servers and (1–3) terabytes for database servers, mailbox servers, and file sharing servers. Then, we estimated the exclusive time for the restoration of servers to be (10–180) minutes, and the total restoration time to be (50–700) minutes.

We assumed that all workers are fully skilled in restoring all servers. We varied the number of workers from three to five.

Fig. 4(a) shows the average levels of satisfaction for the user request types of the data center with different numbers of workers, which is defined as the number of hours different for which user request types are served during the entire restoration process. From this figure, we can observe that the overall user satisfaction increases by just $\sim 0.5\%$ when the number of workers increases from three to five. This is because each type of user request requires multiple services, and their dependent servers need to be restored. However, with a higher number of workers, the service restoration time is dominated by the long shared time of their dependent servers. Before beginning the restoration process corresponding to any server, a worker needs to wait for its dependent servers to be completely restored. For example, in Fig. 3(b) worker 1 remains idle at time slot (3, 4) before touching server 3, as the restoration of server 1 is not finished before time slot 4. This waiting time severely limits the utilization of the workers. This can be verified from Fig. 4(b), which shows that in the synthetic scenario for more than 80% of the time a worker remains idle, especially in the case of the highest number of workers. This significantly reduces the overall worker efficiency and the level of parallelism. Because of this, the total completion time also does not vary significantly with the increase in the number of workers.

From Fig. 4(a), we can also observe that the user satisfaction obtained from the heuristic solution is 99% of that of the optimal solution, which confirms that the proposed genetic algorithm based meta-heuristic

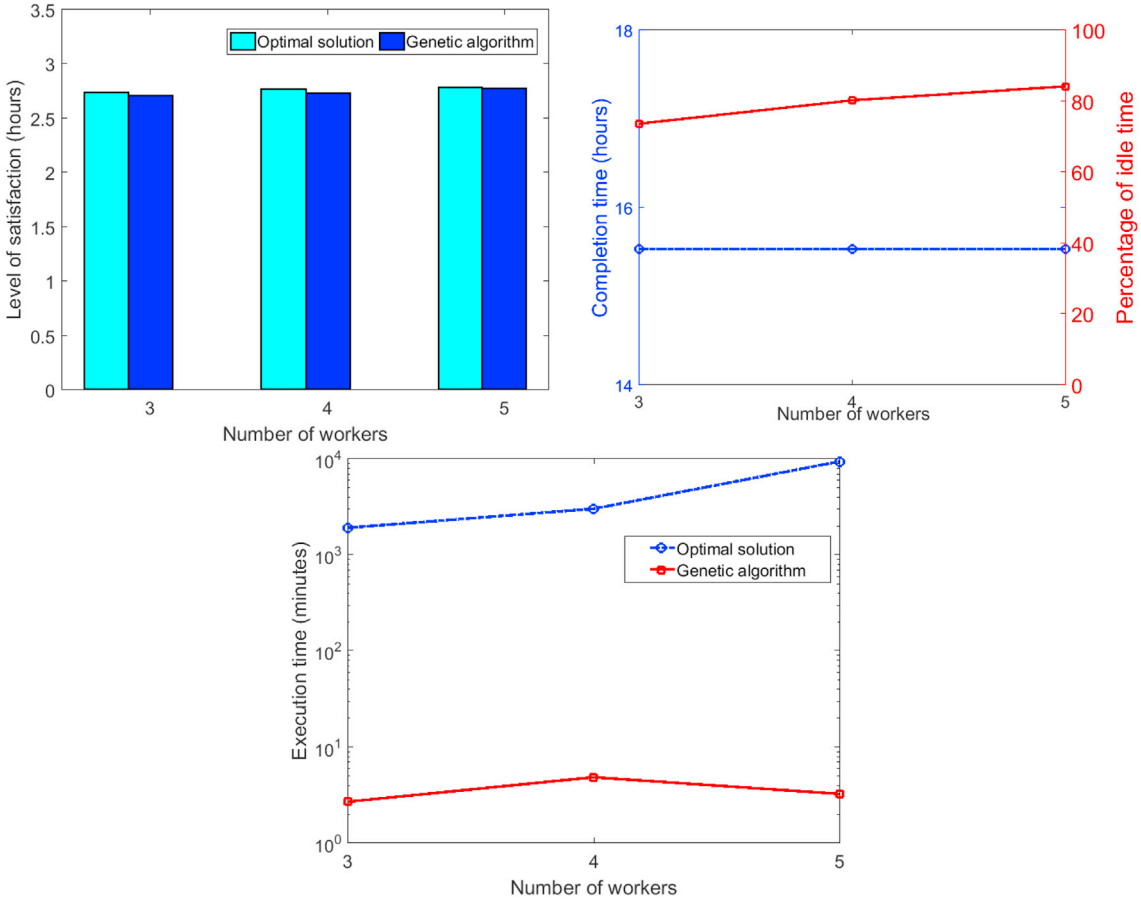

Fig. 4. (a) Comparison of the genetic algorithm and the optimal solution with different numbers of workers. (b) Variation of the total completion time and the percentage of time the workers remain idle. (c) Comparison of the execution times of the optimal solution and the genetic algorithm.

provides fairly accurate results compared to its optimal counterpart. However, in a scenario of a larger data center, some inaccuracy may arise in the genetic algorithm solution, as in the proposed meta-heuristic a worker is immediately assigned the next possible server that can be restored whenever they are free, which may not be optimal in all cases. To compare the execution times of the two schemes, we executed them on an Amazon Web Services (AWS) general purpose machine with 8 vCPU and 32 GB memory. Fig. 4(c) compares the execution time of the proposed genetic algorithm with that of the optimal solution. From this figure, we can observe that the genetic algorithm executes over 2800 times faster than the optimal solution. The execution time for the optimal solution using the PuLP solver ran beyond 150 hours.

4.2. Results obtained from a real-world data center environment

We next evaluated the performance of our proposed scheme using data obtained from a medium-size company that runs a data center for enterprise and commercial workloads. For privacy reasons, we do not identify the company name of the above-mentioned data center. We used a data set containing 15 types of user requests, 60 services, and 107 servers. The mean inter-layer dependencies are found to be 4.5 services per user request type and 1.8 servers per service, whereas each type of user request, service, and server depends on 4.5 (user request type), 2.3 (services), and 2.4 (servers), respectively.

Key services are Microsoft (MS) Windows Active Directory domain and DNS, MS Exchange for mail service, MS Lync for instant messaging service, Oracle Peoplesoft, Oracle Resource Planning, Oracle iRecruitment, Oracle Enterprise Content Management (ECM), Library management system (LMS), Decision Support (BI); a few other business services, such as intranet, timesheet, and contracts; and a few other infrastructure management service, such as WLAN registration and Solarwinds for SNMP management service. Some services, such as Solarwinds, become online once their server is restored, but require the email service to send notifications to the administrators. We decompose such services into multiple services, i.e., the Solarwinds service is up when its corresponding server is restored, whereas the notifications service of Solarwinds is restored only after both Solarwinds and MS Exchange are restored.

We considered different cases for workers/skills, and varied the number of workers in each case. Table 5 shows the different cases. First, we assumed that all workers are fully skilled in restoring all servers as Case 1, and varied the number of workers from four to 20. Then, we grouped the servers based on their vendors. Table 4 shows the different server groups. Each worker generally has the expertise to restore servers of a certain group or set of groups. For Case 2 we generated six sub-cases with the number of workers varied from two to seven, where each worker possesses a unique set of skills. After that, we chose the sub-case with four sets of skill-sets (groups) from Case 2, and created Case 3. In Case 3, we varied the number of workers for each skill-set from one to five. The detailed workers, expertise are listed in Table 5, where $\mathbb{W}_w$ denotes worker w.

Fig. 5 illustrates restoration results for Case 1, in which all workers are fully skilled in restoring all servers. We observe that the average uptimes for user request types increase with the increase in the number of workers. We notice further that this improvement does not scale with the increase in the number of workers, because it begins to saturate above a certain number of workers, due to higher completion times. The percentage of the increase in uptimes for user request types is between 6% and 9% when the number of workers increases from three to four and from four to six. This percentage falls to 2–3% when the number of workers increases to 10 and then decreases further to less than 1% when the number of workers increases further. The idle time of workers increases with the increase of the number of workers.

Fig. 6 shows the restoration results for Case 2. In this case, each worker has the expertise to restore servers of a certain group or set of groups. We can observe that the uptime, for user request types do not increase consistently with the increase of the number of workers, whereas the workers' idle time consistently increases. This is expected, because workers do not share expertise, and can only work on a limited number of servers.

Fig. 7 shows the restoration results for Case 3. As mentioned above, the first sub-case (3.1) is the same as the sub-case 2.3. Then, we increased the number of workers for each skill-set from one to five, i.e., for sub-case 3.2 two workers can restore a subset of servers, for sub-case 3.3 three workers can restore a subset of servers, and so on. We notice that the average uptime for a user request type increases with the increase in the number of workers that can restore servers.

The completion time is lower in Case 1 than in cases 2 and 3, because every worker has the skills to restore all servers. It is lowest in Case 2 compared with cases 1 and 3. Case 3 has a better completion time than Case 2, because more workers can restore the servers. Typically, workers/skill-set is not like Case 1 because it is rarely possible to find all workers with all skills. The question of how many workers with specific skills are required to obtain the best completion time remains an open problem, which is one of our future research endeavors.

Table 4
Technology and number of servers.

Group #	Technology	# of Servers
1	Oracle DB	30
2	MS SQL	8
3	.NET	28
4	Java	7
5	Microsoft	7
6	Oracle App	10
7	Others	17

Table 5
Different cases.

Cases	Sub-cases	W	Worker skills
Case 1	1.1	3	$\mathbb{W}_1, \mathbb{W}_2, \mathbb{W}_3$→Group 1-7
	1.2	4	$\mathbb{W}_1$– $\mathbb{W}_4$→Group 1-7
	1.3	6	$\mathbb{W}_1$– $\mathbb{W}_6$→Group 1-7
	1.4	8	$\mathbb{W}_1$– $\mathbb{W}_8$→Group 1-7
	1.5	10	$\mathbb{W}_1$ – $\mathbb{W}_{10}$→ Group 1-7
	1.6	12	$\mathbb{W}_1$– $\mathbb{W}_{12}$→Group 1-7
	1.7	14	$\mathbb{W}_1$ – $\mathbb{W}_{14}$→ Group 1-7
	1.8	16	$\mathbb{W}_1$– $\mathbb{W}_{16}$→Group 1-7
	1.9	20	$\mathbb{W}_1$– $\mathbb{W}_{20}$→Group 1-7
Case 2	2.1	2	$\mathbb{W}_1$→ Group 1,6,7; $\mathbb{W}_2$→ Group 2-5
	2.2	3	$\mathbb{W}_1$→Group 1–2; $\mathbb{W}_2$→Group 3–4; $\mathbb{W}_3$→Group 5-7
	2.3	4	$\mathbb{W}_1$→Group 1; $\mathbb{W}_2$→Group 2,4,5; $\mathbb{W}_3$→Group 3; $\mathbb{W}_4$→Group 6,7
	2.4	5	$\mathbb{W}_1$→Group 1; $\mathbb{W}_2$→Group 2,4,5; $\mathbb{W}_3$→Group 3; $\mathbb{W}_4$→Group 6; $\mathbb{W}_5$→Group 7
	2.5	6	$\mathbb{W}_1$→Group 1; $\mathbb{W}_2$→Group 2,5; $\mathbb{W}_3$→Group 3; $\mathbb{W}_4$→Group 4; $\mathbb{W}_5$→Group 6; $\mathbb{W}_6$→Group 7
	2.6	7	$\mathbb{W}_1$→Group 1; $\mathbb{W}_2$→Group 2; $\mathbb{W}_3$→Group 3; $\mathbb{W}_4$→Group 4; $\mathbb{W}_5$→Group 5; $\mathbb{W}_6$→Group 6; $\mathbb{W}_7$→Group 7
Case 3	3.1	4	$\mathbb{W}_1$→Group 1; $\mathbb{W}_2$→Group 2,4,5; $\mathbb{W}_3$→Group 3; $\mathbb{W}_4$→Group 6,7
	3.2	8	$\mathbb{W}_1, \mathbb{W}_2$→Group 1; $\mathbb{W}_3, \mathbb{W}_4$→Group 2,4,5; $\mathbb{W}_5, \mathbb{W}_6$→Group 3; $\mathbb{W}_7, \mathbb{W}_8$→Group 6,7
	3.3	12	$\mathbb{W}_1$– $\mathbb{W}_3$→Group 1; $\mathbb{W}_4$– $\mathbb{W}_6$→Group 2,4,5; $\mathbb{W}_7$– $\mathbb{W}_9$→ Group 3; $\mathbb{W}_{10}$– $\mathbb{W}_{12}$→Group 6,7
	3.4	16	$\mathbb{W}_1$– $\mathbb{W}_4$→Group 1; $\mathbb{W}_5$ – $\mathbb{W}_8$→Group 2,4,5; $\mathbb{W}_9$– $\mathbb{W}_{12}$→ Group 3; $\mathbb{W}_{13}$– $\mathbb{W}_{16}$→ Group 6,7
	3.5	20	$\mathbb{W}_1$– $\mathbb{W}_5$→Group 1; $\mathbb{W}_6$– $\mathbb{W}_{10}$→ Group 2,4,5; $\mathbb{W}_{11}$– $\mathbb{W}_{15}$→ Group 3; $\mathbb{W}_{16}$– $\mathbb{W}_{20}$→Group 6,7

5. Related works

5.1. Multi-layer networks

Multi-layer networks consist of multiple subsystems and layers of

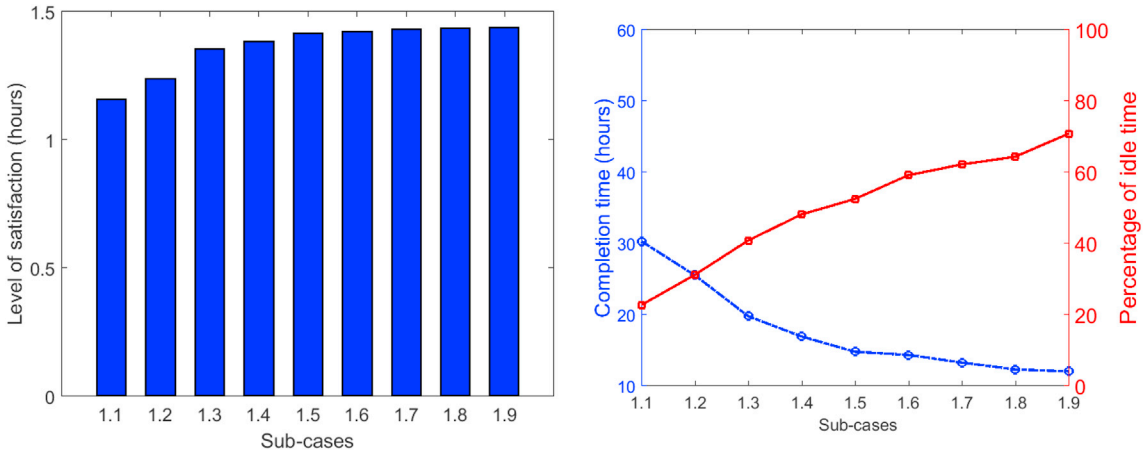

Fig. 5. Results obtained from Case 1.

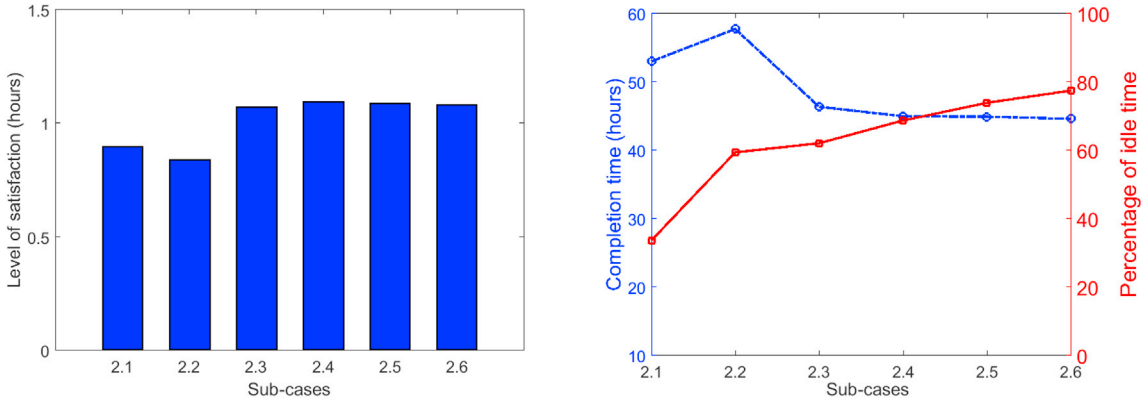

Fig. 6. Results obtained from Case 2.

connectivity or inter-dependency relations between them. Such types of networks are generally represented as graphs where different agents (or subsystems) are represented as vertices and the relations between different pairs of vertices are represented as edges [19]. In the last decade, multi-layer networks are used in different applications in different domains, such as interacting power grids [20], cascading failures and recovery in interconnected power grid and communications networks [21–24], coupled climate networks [25], interconnected transportation networks [26,27], social network between cancer researchers, their affiliations and connections [28], etc.

5.2. Operator scheduling

The job-shop scheduling problem along with operators is studied in Refs. [12–15]. Several approaches are discussed in the literature to solve this problem: in Ref. [29] The authors have proposed artificial intelligence schemes to solve this problem, whereas the authors in Ref. [30] have proposed a schedule generation scheme with an objective of minimizing the total flow time. Operator assignment problem has also been studied in the context of employee timetabling problems [25,31]. Some of these papers address problems related to the one investigated in

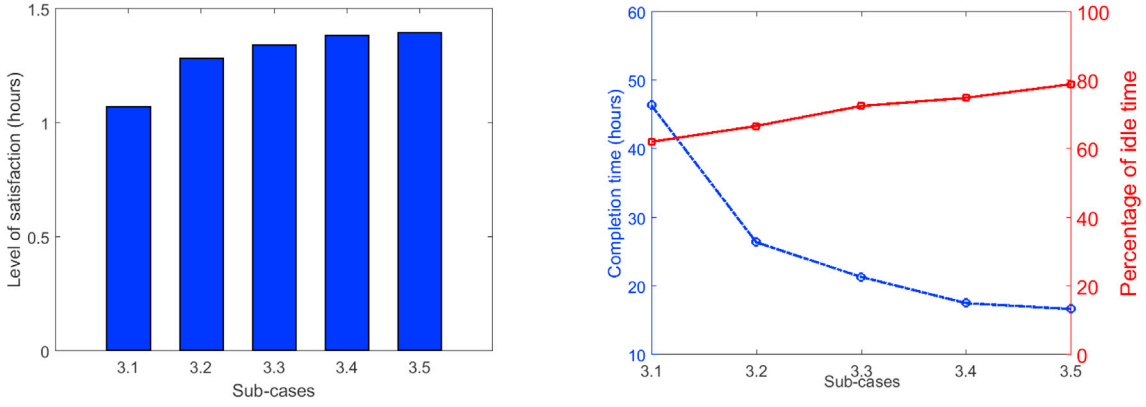

Fig. 7. Results obtained from Case 3.

this paper. In Ref. [32] the authors have studied the timetabling problem in conjunction with the job shop problem. The resource scheduling problem has been used for different applications, such as in pharmaceutical environments [33], in handicraft production [34], etc.

Our proposed scheme is significantly different from the above schemes due to several reasons. All the above scheduling schemes have tried to reduce the overall makespan or total completion time, whereas our objective is to maximize the level of satisfaction of different types of user requests during the recovery process. This needs a clear understanding and modeling the interdependencies between different servers, services and data center types of user requests by using a three-layer dependency modeling, which is unlike in the related literature. Also in our worker assignment problem, the completion time of an application is divided into exclusive and shared phases; such an environment is not considered in the above literature.

6. Conclusions and future works

In this paper, we propose a progressive data center restoration scheme in the face of large-scale disruptions with an objective of maximizing the limited service provided by the data center infrastructure during the recovery process. We propose a heuristic approach to solve this complicated problem considering the inter-dependencies of different services as well as the experts' availability. We have conducted extensive simulations on real world data center traces and shown that the heuristic approach performs quite well compared to the optimal solution.

While the paper has made a thorough analysis of the progressive recovery and restoration problem in the context of large enterprise data centers, it has simplified a number of practical concerns that arise in enterprise data center networks. For example, we assumed that when necessary servers corresponding to a service is restored, the service is up and running. However, in practice, some services may support partial load if some critical servers are restored, whereas the performance improves gradually as more servers come up. Also sometimes disruption may occur as a result of exploited vulnerabilities. Thus, it is necessary to restore and run patching to fix the vulnerabilities before services can be restored to avoid future disruptions. Additionally, some services require successive restoration such as restoring from backup and then restoring transaction logs or sometimes rebuilding the service's servers and then complete the configuration. Integrating these practical issues in our model is one of our future considerations.

References

[1] S. Mitchell, S.M. Consulting, I. Dunning, Pulp: A linear programming toolkit for python, 2011.
[2] http://iwgcr.org/japan-earthquake-puts-data-centers-and-cloud-services-at-risk/.
[3] http://www.datacenterdynamics.com/content-tracks/power-cooling/hurricane-sandy-data-center-stories-from-manhattan/72772.fullarticle.
[4] https://thestack.com/data-centre/2015/08/19/lightning-wipes-storage-disks-at-google-data-centre/.
[5] http://www.americanbanker.com/issues/176/_195/bank-of-america-website-outage-online-banking-1042932-1.html.
[6] http://aws.amazon.com/message/65648/.
[7] http://www.informationweek.com/data-center-outages-generate-big-losses/d/d-id/1097712?.
[9] http://www.drj.com/drj-world-archives/general-dr-planning/feed/Page-1.html.
[10] http://www.datacenterdynamics.com/news/the-data-center-skills-gap/75576.fullarticle.
[11] http://www.wipro.com/documents/the-impending-data-center-talent-crisis-and-how-to-avert-it.pdf.
[12] M.F. Baki, R.G. Vickson, One-operator, two-machine open shop and flow shop problems with setup times for machines and weighted number of tardy jobs objective, Optim. Methods Softw. 19 (2) (2004) 165–178.
[13] H. Kellerer, V.A. Strusevich, Scheduling parallel dedicated machines under a single non-shared resource, Eur. J. Oper. Res. 147 (2) (2003) 345–364.
[14] C.A. Glass, Y.M. Shafransky, V.A. Strusevich, Scheduling for parallel dedicated machines with a single server, Nav. Res. Logist. 47 (4) (2000) 304–328.
[15] A. Agnetis, M. Flamini, G. Nicosia, A. Pacifici, A job-shop problem with one additional resource type, J. Sched. 14 (3) (2011) 225–237.
[16] I. El-Shekeil, A. Pal, K. Kant, Progressive recovery of interdependent services in enterprise data centers, in: IEEE Resilience Week, 2016, pp. 27–32.
[17] http://ampl.com/products/solvers/.
[18] H.W. Kuhn, The Hungarian method for the assignment problem, Nav. Res. Logist. Q. 2 (1955) 83–97.
[19] M.D. Domenico, C. Granell, M.A. Porter, A. Arenas, The Physics of Multilayer Networks, arXiv preprint arXiv:1604.02021, 2016, https://arxiv.org/abs/1604.02021.
[20] C.D. Brummitt, R.M. D'Souza, E.A. Leicht, Suppressing cascades of load in interdependent networks, Proc. Natl. Acad. Sci. 109 (12) (2012) E680–E689.
[21] S. Soltan, D. Mazauric, G. Zussman, Cascading failures in power grids: analysis and algorithms, in: International Conference on Future Energy Systems, 2014, pp. 195–206.
[22] A. Bernstein, D. Bienstock, D. Hay, M. Uzunoglu, G. Zussman, Power grid vulnerability to geographically correlated failures - analysis and control implications, in: IEEE INFOCOM, 2014, pp. 2634–2642.
[23] A. Sen, A. Mazumder, J. Banerjee, A. Das, R. Compton, Identification of K most vulnerable nodes in multi-layered network using a new model of interdependency, in: IEEE INFOCOM Workshops, 2014, pp. 831–836.
[24] A. Mazumder, C. Zhou, A. Das, A. Sen, Progressive recovery from failure in multi-layered interdependent network using a new model of interdependency, in: CRITIS, 2014, pp. 368–380.
[25] O. Guyon, P. Lemaire, Pinson D. Rivreau, Cut generation for an integrated employee timetabling and production scheduling problem, Eur. J. Oper. Res. 201 (2) (2010) 557–567.
[26] A. Halu, S. Mukherjee, G. Bianconi, Emergence of Overlap in Ensembles of Spatial Multiplexes and Statistical Mechanics of Spatial Interacting Networks Ensembles.
[27] R. Parshani, C. Rozenblat, D. Ietri, C. Ducruet, S. Havlin, Inter-similarity between Coupled Networks.
[28] E. Lazega, M.-T. Jourda, L. Mounier, R. Stofer, Catching up with big fish in the big pond? multi-level network analysis through linked design, Soc. Netw. 30 (2) (2008) 159–176.
[29] T. Yamada, R. Nakano, Genetic algorithms for job-shop scheduling problems, in: Modern Heuristic for Decision Support, 1997, pp. 474–479.
[30] M.R. Sierra, C. Mencía, R. Varela, Optimally scheduling a job-shop with operators and total flow time minimization, in: CAEPIA, 2011, pp. 193–202.
[31] C. Artigues, M. Gendreau, L.-M. Rousseau, A. Vergnaud, Solving an integrated employee timetabling and job-shop scheduling problem via hybrid branch-and-bound, Comput. Oper. Res./Comput. Oper. Res. 36 (8) (2009) 2330–2340.
[32] O. Guyon, P. Lemaire, R. Pinson, D. Rivreau, Solving an integrated job-shop problem with human resource constraints, Ann. Oper. Res. 213 (1) (2014) 147–171.
[33] R. Driessel, L. Mönch, An Integrated Scheduling and Material-handling Approach for Complex Job Shops : a Computational Study, 2012.
[34] A. Agnetis, G. Murgia, S. Sbrilli, A Job Shop Scheduling Problem with Human Operators in Handicraft Production, 2014.