Progressive Recovery of Interdependent Services in Enterprise Data Centers

Ibrahim El-Shekeil, Amitangshu Pal, and Krishna Kant
Computer and Information Sciences, Temple University, Philadelphia, PA 19122
E-mail:{ielshekeil,amitangshu.pal,kkant}@temple.edu

***Abstract*—In this paper, we discuss the progressive restoration planning in an enterprise data center environment after partial or full disruption. Repairing disrupted components takes significant amount of time and human involvement. Thus, after a major disruption, the recovery process will involve multiple stages and during each stage, the partially recovered infrastructures will be able to provide limited services at some degraded service level. However, how fast and effective an enterprise infrastructure can be recovered depends on how the recovery processes will restore the disrupted components considering the inter-dependencies in between the services, along with the limitations of expert human operators. The entire problem turns out to be NP-hard and rather complex, and we devise effective meta-heuristics to solve the problem. By taking some real-world examples, we show that the proposed meta-heuristics gives fairly accurate results compared to the optimal solution, with the ratio between the optimal and heuristic solution never exceeding 1.04.**

I. Introduction

Disruptions in data centers may occur as a result of a hardware failure, operating system or software failure, intrusion, virus outbreak or natural disaster. Natural disasters include the 2011 Japan earthquake and tsunami that damaged major data centers in Tokyo [1], and the 2012 Hurricane Sandy in USA where some data centers in New York were affected due to flood [2]. Other examples include storms or lightning that took down Google's St. Ghislain data center operations for five days in 2015 [3], or technical hiccups that affected the services of Bank of America [4] and Amazon [5] centers for 4-6 days. Such scenarios or disruptions may also arise when a data center is relocated or upgraded in a different site, which needs proper pre-move planning and expertise. These disruptions could be partial that impact some applications or full that impact the entire data center. When such disruptions occur it causes significant downtime which may lead to a substantial financial and legal impact.

In light of the above, there is a growing need to optimize the post-disruption recovery and restoration process for the 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 a few weeks to several months [6]. Within this entire recovery stage, the partially restored infrastructures will still have to operate in a degraded manner and provide some partial level of service for clients. A key design challenge of the restoration plan is to support partial business continuity, that allows applications to *progessively* come back online after failures or disruptions. Clearly 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 decision on recovery sequence of these services has an important role on minimizing the loss by bringing the most critical applications back.

This research was supported by the NSF grant CCF-1407876.

TABLE I. Dependencies in between the servers and services

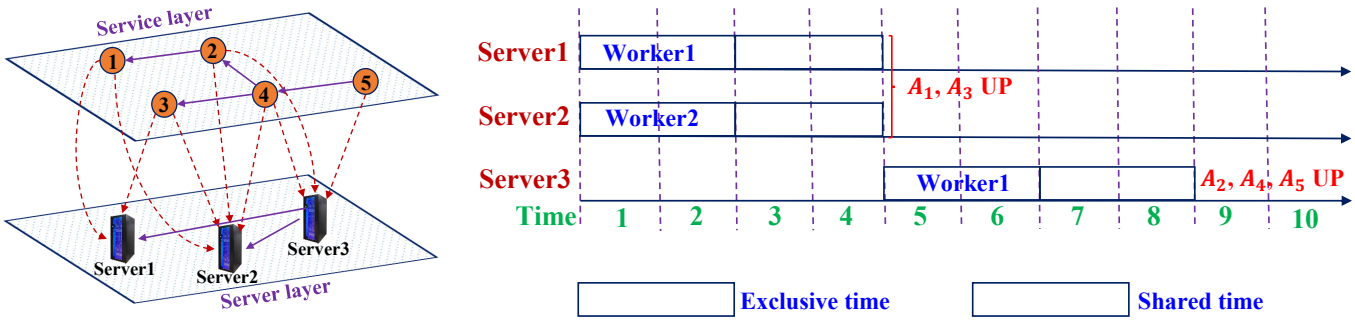
Enterprise services	Necessary servers
Microsoft Project	Web front end server, application server, database server
Microsoft SharePoint	Web front end server, application server, database server
Active Directory Authentication	Domain controller
Microsoft DNS	Domain controller
Human Resources	Application server, database server
Microsoft Exchange	Front end server (does mail routing), back end server (stores mail boxes), web client

In this paper we focus on the post-disruption progressive data center recovery problem with an objective of maximizing the up-time of the data center service in the entire recovery process. Repairing the disrupted services requires long times and human resources or workers. The availability of human resources with the desired expertise varies over time. Recovering different services may require different time to do manual configuration and actual restoration, which makes the decision process even complex. Since 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. The entire problem turns out to be NP-hard as we shall show later. We propose a genetic algorithm based meta-heuristics to solve the combined problem. With simulations we show that the ratio between the optimal solution and the proposed genetic algorithm based solution does not exceed 1.04, which confirms that the proposed heuristics provides fairly accurate results compared to its optimal counterpart. To the best of our knowledge, this is the first work addressing the data center post-disruption progressive recovery planning while considering the interdependencies of various services as well as considering the human related constraints and expertise.

Accordingly, the outline of the paper is as follows. Section II addresses the overall problem formulation including its complexity and the proposed meta-heuristic solution. Section III reports results based on real data obtained from an enterprise data center environments. Section IV discussed the related works. Finally, section V concludes the discussion and discusses potential future works.

II. Problem Formulation

We next model this problem using a 2-layer interdependency framework. Layer-1 or *service layer* consists of the set of services that the enterprise provides, whereas layer-2 or *server layer* consists of the servers that need to be restored to bring back the services as shown in Fig. 1(a). The services depend on one or more servers to run, which can be modeled



(a) (b)

Fig. 1. (a) Dependency graph in between the service layer and the server layer. (b) Timing diagram of the worker to server assignment. $A_1 - A_5$ denote all the services.

as the *Inter-layer dependencies*. Table I shows different key services of an enterprise and the corresponding servers that they are dependent on. If Q_{ai} denotes whether the service a is dependent on server i or not, then according to Fig. 1(a), $Q_{11} = Q_{12} = 1$ as service 1 depends on servers 1 and 2. On the other hand, in layer-1 and layer-2 there are dependencies in between the services as well as in between the servers respectively, which we define as *Intra-layer dependencies*. For example the email services, human resources and SharePoint depend on the DNS and Active Directory services, thus these two services needs to be restored first before restoring others. On the other hand in the server layer, the web client depends on the front and back end servers for email services, application server depends on the database server etc. Assume that P_{ab} denotes whether service a is dependent on service b or not, similarly O_{ij} denotes the dependencies in between the servers. Thus in Fig. 1(a), $P_{21} = P_{42} = P_{43} = P_{54} = 1$ and $O_{31} = O_{32} = 1$.

Assume that time is divided into *slots* which are denoted as t. Let $[0, T]$ denote the time horizon, where T is the total time in which all primary servers are recovered and $0 \leq t \leq T$. We assume that there are M services that need to be activated, which run on a total of N servers. The server restoration process consists of two steps: first is the *exclusive* stage when an expert worker corresponding to a particular server needs to install the restoration tasks in a dedicated fashion. The next phase is *shared* which needs some infrequent monitoring. We thus assume that a worker (or expert) becomes free after an exclusive stage, as the remaining stage can mostly be taken care of by common operators without an experts' intervention. Let us assume that x_i^t represents whether a server i is touched on or before time instance t or not. Similarly e_i^t represents whether its exclusive stage is over or not, and z_i^t represents whether it is completely restored or not. For example, in Fig.1(b) $x_1^t = 1$ for $t \geq 1$, $e_1^t = 1$ for $t \geq 3$, and $z_1^t = 1$ for $t \geq 5$. Similarly we assume that y_a^t denotes whether service a is restored on or before time instance t or not. For the problem formulation we assume that exclusive time of server i is l_i units, and the entire expected restoration time is L_i units. We assume that the number of expert workers are much lesser than the number of servers that need to be restored, i.e. $W \ll N$. Thus the workers need to take turns to bring the services back. The workers do not have the expertise to work on all servers, thus the workers need to be assigned one after other to progressively restore the servers depending on their availability and expertise. The necessary notations are depicted in Table II.

TABLE II. TABLE OF NOTATIONS

		Indices
a, b	$\triangleq$	Index for service's (1, ..., M)
i	$\triangleq$	Index for servers (1, ..., N)
t	$\triangleq$	Index for time units (1, ..., T)
w	$\triangleq$	Index for workers (1, ..., W)
		Variables
x_i^t	$\triangleq$	Whether or not restoration of server i has been started by a worker on or before time instance t
e_i^t	$\triangleq$	Whether or not the exclusive part of server i is finished on or before time instance t
z_i^t	$\triangleq$	Whether or not server i is up at time instance t
y_a^t	$\triangleq$	Whether or not service a is up at time instance t
l_i	$\triangleq$	Expected exclusive time of server i
L_i	$\triangleq$	Expected restoration time of server i
P_{ab}	$\triangleq$	Whether or not service a is dependent on service b
Q_{ai}	$\triangleq$	Whether or not service a is dependent on server i
O_{ij}	$\triangleq$	Whether or not server i is dependent on server j
$D(a)$	$\triangleq$	Set of servers or services on which service a depends on
$D(i)$	$\triangleq$	Set of servers on which server i depends on
$\mathbb{A}_{wi}$	$\triangleq$	Whether or not server i is assigned worker w
R_{wi}	$\triangleq$	Whether or not worker w can restore server i

With these we next formulate our optimization problem framework named *progressive restoration (PR)* with the goal to maximize the total up-time of services, i.e

$$\max \sum_{a=1}^{M} \sum_{t} y_a^t \tag{1}$$

This is because $\sum_t y_a^t$ represents the total up-time of the service a. For example, assume that $T = 100$, and the service a is up and running at time instance 10. Thus the service remains ON from time instance 10 onwards, and the total up-time of a is 90. We next describe the constraints as follows:

Dependency constraints: Constraint(2) models the intra-layer dependencies which states that if a service a is dependent on a set of service $D(a)$, then $D(a)$ needs to be ON before a can be made ON. This constraint becomes a nonlinear constraint due to the presence of the *product* operator, however the constraint can be linearized by incorporating the *summation* operator as shown in equation(2). Constraint (3) models the service-server dependency, which states that the service a cannot be ON without restoring the set of servers in $D(a)$. Similarly constraint(4) models the dependencies in between

the servers.

$$y_a^t \leq \prod_{b \in D(a)} y_b^t \implies \left(\sum_{b=1}^{M} P_{ab}\right).y_a^t \leq \sum_{b=1}^{M} P_{ab}.y_b^t \ \ \forall a,t \quad (2)$$

$$y_a^t \leq \prod_{i \in D(a)} z_i^t \implies \left(\sum_{i=1}^{N} Q_{ai}\right).y_a^t \leq \sum_{i=1}^{N} Q_{ai}.z_i^t \ \ \forall a,t \quad (3)$$

$$x_i^t \leq \prod_{j \in D(i)} z_j^t \implies \left(\sum_{j=1}^{N} O_{ij}\right).x_i^t \leq \sum_{j=1}^{N} O_{ij}.z_j^t \ \ \forall i,t \quad (4)$$

Timing constraints: Constraints(5)-(8) model the start and end time of the server restoration. Constraints(5)-(6) ensure that the exclusive time of server i is completed after l_i time units. Constraints(7)-(8) ensure that server i is fully restored after L_i time units.

$$x_i^t = e_i^{t+l_i} \ \ \forall i, \forall t = (1,2,...,T-l_i) \quad (5)$$

$$\sum_{t=1}^{T-l_i} x_i^t = \sum_{t=1}^{T} e_i^t \ \ \forall i \quad (6)$$

$$x_i^t = z_i^{t+L_i} \ \ \forall i, \forall t = (1,2,...,T-L_i) \quad (7)$$

$$\sum_{t=1}^{T-L_i} x_i^t = \sum_{t=1}^{T} z_i^t \ \ \forall i \quad (8)$$

Worker assignment constraints: Constraint(9) states that worker w is assigned to restore server i, if he has the expertise to restore it. Constraint(10) ensures that all servers are assigned a worker, and thus restored by the end of the time scale T. Constraint(11) states that if worker w is assigned to restore server i and j, then their exclusive time do not overlap.

$$\mathbb{A}_{wi} \leq R_{wi} \ \ \forall i, \forall w \quad (9)$$

$$\sum_{w} \mathbb{A}_{wi} = 1 \ \ \forall i \quad (10)$$

$$x_i^t - e_i^t + x_j^t - e_j^t \leq 1 + \mathbb{M}\left(2 - \mathbb{A}_{wi} - \mathbb{A}_{wj}\right) \ \ \forall t, \forall w, \forall i \neq j \quad (11)$$

where $\mathbb{M}$ is a large number which is more than the maximum value of the left hand side.

Other constraints: Constraint(12) states that if an service a is ON at time instance t then it will remain ON at all the future time steps. Similarly if a server is restored, then it remain active for the future time steps, which is modeled in equation(13). Constraints(14)-(15) state similar constraints corresponding to the server restoration start time and exclusive time respectively.

$$y_a^{t+1} \geq y_a^t \ \ \forall i, \forall t = (1,2,...,T-1) \quad (12)$$

$$z_i^{t+1} \geq z_i^t \ \ \forall i, \forall t = (1,2,...,T-1) \quad (13)$$

$$x_i^{t+1} \geq x_i^t \ \ \forall i, \forall t = (1,2,...,T-1) \quad (14)$$

$$e_i^{t+1} \geq e_i^t \ \ \forall i, \forall t = (1,2,...,T-1) \quad (15)$$

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

Proof: We first define the minimum latency set cover problem (MLSC) which is known to be NP-hard. Let $J = \{J_1, J_2, ..., J_m\}$ be a set of jobs to be processed by a factory. A job J_i has non-negative weight w_i. Let $T = \{t_1, t_2, ..., t_n\}$ be a set of tools. Job j is associated with a nonempty subset $S_j \subseteq T$, once the entire tool subset S_j has been installed, job 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 PR problem.We define one instance of the MLSC problem when $w_i = 1, \forall i$. With this, we define an instance of the PR problem as follows: we assume that there is no dependencies in between the services as well as in between the servers. Also assume that there are just one worker who has the expertise to restore all the servers. The restoration time of all the servers are unity, with the shared time assumed to be zero, i.e. $l_i = L_i = 1, \forall i$. This reduction transforms a MLSC problem instance into a PR instance. ■

Example: Let us consider an example with M = 5 and N = 3. The intra and inter-layer dependencies are shown in Fig.1(a). We use AMPL solver [7] for solving the optimization problem. Fig.1(b) shows the timing diagram of the restoration process obtained by solving problem(1). We assume that there are two workers, worker-1 can restore servers 1 and 3, whereas worker-2 can restore servers 1 and 2. The exclusive and shared time of all the servers are assumed to be of 2 units. From Fig.1(b) we can observe that as all the services are dependent of A_1 and A_3, thus server 1 and 2 are first restored to run these two services. Server 3 is then restored to run the other services.

A. Proposed heuristic

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 in natural life, offspring chromosomes are obtained from parent chromosomes mainly by using two operators, crossover and mutation. Some other chromosomes simply survive unaltered, while others die off. Different steps of the entire genetic algorithm is described as follows.

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 $(c_1, c_2, ..., c_N)$, where c_i represents the i-th server (or gene). Thus the genes of a chromosomes are the servers, and a chromosome gives the sequence of servers. We assume that there are $\mathcal{M}$ 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 find the total up-time of the services, 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 the 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 putting 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 zero. In this bipartite graph we then run the maximum matching algorithm like Hungarian scheme [8] to assign the workers to their corresponding servers.

After the initial assignment the workers become free after the exclusive time of the corresponding server. Whenever a worker is free, he is assigned to the next possible server that can be restored (depending on the already restored ones) in the sequence depending on his expertise. This process goes on until all the servers are finally restored.

Initial mating pool generation: Initially the mating pool is generated randomly, considering the fact that the chromosome generated satisfy the *precedence constraints* or dependency relations in between the servers. For example in Fig. 1(a) $3 \rightarrow 2 \rightarrow 1$ will be an invalid chromosome structure as server 3 depends on servers 1 and 2, thus server 3 cannot be touched until and unless the other two servers are completely restored. We thus describe the chromosome generation process in the initial mating pool using an example in Table III. Assume that Table III shows the precedence/dependency relations among the servers, which we define as the *server dependency matrix*. Servers 1 and 2 do not depend on 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, server 5 depends on servers 3 and 4. Initially the chromosomes need to be generated such that this dependency relations are maintained. To do that, we define the *candidate server set (CSS)* as the server set with all 0 rows, which are servers 1 and 2 in case of Table III. We next choose any one of the servers from the CSS randomly, this server becomes the first gene in the chromosome. We next remove the row and column corresponding to that server from Table III. We then construct the CSS with all 0 rows from the remaining dependency matrix and then choose the next gene randomly from the CSS. This process goes on to generate all the genes of a chromosome. The process is repeated for generating all the chromosomes in the initial mating pool. This ensures that the chromosomes in the initial mating pool are consistent with the precedence relation.

TABLE III. 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

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

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 two-point crossover, although in general n-point crossover can be used as well. For a two-point crossover we first generate a *cutting-point* randomly. 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, we replace the $c+1$-th gene of the first chromosome with that of the second one. Otherwise, we randomly choose a server from the CSS for the $c+1$-th gene. We follow the same procedure for the other genes (from $c+2$ onwards), and repeat the same for the second chromosome.

Mutation operation: For the mutation operation, we choose a chromosome randomly and also choose a cutting-point c randomly. We retain the first c genes, generate the CSS and choose a server from the CSS randomly 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 are consistent with the precedence relation.

The algorithm stops when the best solution of a generation does not improve significantly 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.

III. EXPERIMENTAL RESULTS

A. Validating the accuracy of the genetic algorithm

We first validate the accuracy of the proposed genetic algorithm compared to the optimal solution obtained from PuLP solver [9], which is a library for the Python scripting language to solve mathematical programs. We synthetically generate a scenario with 12 services and 30 servers for this purpose. We model the dependencies artificially such that on average a service depends on 2 services and 2.5 servers, whereas each server depends on another 2.5 servers. The exclusive and shared time of the servers are approximated from practical scenarios which typically varies from (10, 120) and (15, 300) minutes respectively. We assume that the worker's expertise to the servers follow a round robin principle. For example, in case of 3 workers we assume that $\mathbb{W}_1, \mathbb{W}_2, \mathbb{W}_3$ have the expertise of restoring servers $(S_1, S_4, S_7, \ldots)$, $(S_2, S_5, S_8, \ldots)$ and $(S_3, S_6, S_9, \ldots)$ respectively.

Fig. 2(a) shows the total up-time of the data center services with different number of workers. From this figure we can observe that the total up-time of the services increases by just $\sim$2% when the number of workers increase from 2 to 6. This is because with higher number of workers, the service restoration time is dominated by the long shared time of their dependent servers. Before starting the restoration process corresponding to any server, a worker needs to wait for its dependent servers to be completely restored. For example in Fig. 1(b) worker-1 remains idle at time instance (3, 4) before touching server 3, as the restoration of server 1 does not finish before time instance 4. This waiting time severely limits the utilization of the workers. This can be verified from Fig. 2(b) which

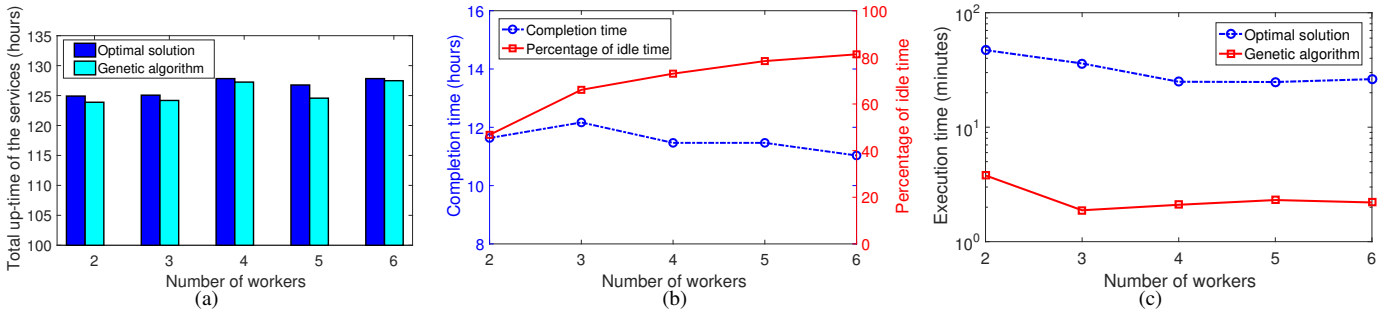


Fig. 2. (a) Comparison of the genetic algorithm and the optimal solution with different number of workers. (b) Variation of the total completion time and the percentage of time the workers remain idle. (c) Comparison of the execution time of the optimal solution and the genetic algorithm.

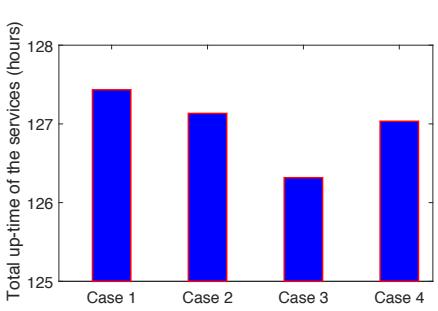


Fig. 3. Total up-time of the services for different combinations of worker expertise.

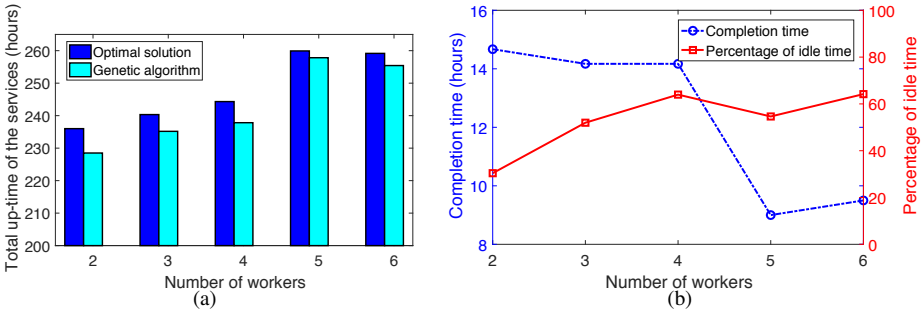


Fig. 4. (a) Comparison of the genetic algorithm and the optimal solution with different number of workers. (b) Variation of the total completion time and the percentage of time the workers remain idle.

shows that in the synthetic scenario ∼80% of the time a worker remains idle, especially in case of higher number of workers. This significantly reduces their overall efficiency and the level of parallelism. Because of that the total completion time also does not vary significantly with the increase in number of workers.

From Fig. 2(a) we can also observe that the ratio between the optimal and the heuristics solution does not exceed 1.02, which confirms that the proposed genetic algorithm based meta-heuristics provides fairly accurate results compared to its optimal counterpart. The minor inaccuracy is generated mainly in the assignment process, as in the proposed meta-heuristics a worker is immediately assigned the next possible server that can be restored whenever he is free, which may not be optimal in all cases. To compare the execution time of the two schemes, we execute them on Amazon Web Services (AWS) general purpose machine with 16 vCPU and 64 GB memory. Fig. 2(c) compares the execution time of the proposed genetic algorithm along with the optimal solution. From this figure we can observe that the genetic algorithm executes ∼12-20 times faster than the optimal solution. Infact in case of slightly larger case (with 20 services and 33 servers) the PuLP solver's execution time grows even beyond 10 hours.

The total up-time of the services highly depends on the type of workers and their expertises. Fig. 3 shows the service up-time in presence of 4 workers with various combinations of worker expertises, which we divide into following four cases:

1) **Case 1:** All workers have expertise to restore all servers.
2) **Case 2:** Workers are divided into two teams $\mathbb{T}_1$, $\mathbb{T}_2$. In $\mathbb{T}_1$ one of the workers can restore $(S_1, S_3, \ldots)$ whereas the other one can restore $(S_2, S_4, \ldots)$.
3) **Case 3:** Identical to the round-robin policy adopted in Fig. 2.
4) **Case 4:** Divide the workers into two teams $\mathbb{T}_1$, $\mathbb{T}_2$ and the servers into four groups $G_1 - G_4$. In $\mathbb{T}_1$ all the workers can restore the servers in (G_1, G_2), whereas all $\mathbb{T}_2$ workers are able to restore the servers in (G_3, G_4).

From Fig. 3 we can observe that the up-time is highest when all the workers have the expertise to restore all the servers. As their expertises become limited, the total up-time also reduces. Also the total up-time is lowest in case 3 as a particular server in this case can be only restored by a specific worker.

B. Results obtained from a real data center environment

We next evaluate the performance of our proposed scheme using data obtained from a medium size company that runs data center for running enterprise and commercial workloads: due to privacy reasons, we do not identify the company name of the above-mentioned data center. We use a data set of 20 services and 33 servers. The mean inter-layer dependencies is found to be 2 servers/service, whereas each service and server depends on 3 (services) and 1.5 (servers) respectively. Key services are Microsoft (MS) Windows Active Directory domain and DNS, MS Exchange for mail service, MS Lync for instant messaging service, SAP human resource and invoice management services, OpMeneger for SNMP network and system monitoring, EMC Documentum for file archiving and other few business services such as intranet, timesheet, contracts, etc. Few services such as OpManager becomes up once its server is restored but requires the email service to send notifications to the administrators. We decompose such services into multiple services, i.e. the OpManager service is up when its corresponding server is restored, whereas the notifications service of OpManager is restored only after both of OpManager and MS Exchange are restored. We grouped the servers based no their vendors; each worker generally has expertise to restore servers of certain vendors. The detailed

TABLE IV. WORKER EXPERTISE DATABASE

W	**Expertise Database**
2	$\mathbb{W}_1 \rightarrow$ (15 Microsoft servers), $\mathbb{W}_2 \rightarrow$ (18 others)
3	$\mathbb{W}_1 \rightarrow$ (15 Microsoft servers), $\mathbb{W}_2 \rightarrow$ (6 SAP), $\mathbb{W}_3 \rightarrow$ (12 others)
4	$\mathbb{W}_1 \rightarrow$ (15 Microsoft servers), $\mathbb{W}_2 \rightarrow$ (6 SAP), $\mathbb{W}_3 \rightarrow$ (5 EMC), $\mathbb{W}_4 \rightarrow$ (7 others)
5	$\mathbb{W}_1, \mathbb{W}_2 \rightarrow$ (15 Microsoft servers), $\mathbb{W}_3 \rightarrow$ (6 SAP), $\mathbb{W}_4 \rightarrow$ (5 EMC), $\mathbb{W}_5 \rightarrow$ (7 others)
6	$\mathbb{W}_1 \rightarrow$ (5 Microsoft Exchange servers), $\mathbb{W}_2 \rightarrow$ (4 Microsoft Lync servers), $\mathbb{W}_3 \rightarrow$ (6 Microsoft servers), $\mathbb{W}_4 \rightarrow$ (6 SAP), $\mathbb{W}_5 \rightarrow$ (5 EMC), $\mathbb{W}_6 \rightarrow$ (7 others)

worker's expertise is enlisted in Table IV, where $\mathbb{W}_w$ denotes worker w. We vary the number of workers from 2 to 6 and record the total uptime of the services.

Fig. 4(a) shows the total up-time of the services with the number of workers. From this figure we can observe that the total up-time increases by $\sim$12% as the number of workers increase from 2 to 6. Compared to the synthetic scenario, in the real data center scenario the amount of worker's idle time reduces significantly (from 80% to 60%) which reduces the completion time significantly (by $\sim$50%) as the number of workers is increased. While comparing with the optimal solution we observe that the accuracy of the genetic algorithm is no worse than $\frac{1}{1.04}$ times that of the optimal value as obtained from Fig. 4.

IV. RELATED WORKS

Multi-layer networks: Network theory is an important tool for designing complex systems, which are generally represented as graphs where different agents are represented as vertices and the relation in between different pair of vertices are represented as edges. However as the research of complex systems evolve, more realistic framework with heterogeneous vertices or edges become essential, that are manifested in the form of multi-layer networks, inter-dependent networks, networks-of-network etc. In the last decade multi-layer networks are used in different applications in different domains, such as interacting power grids [10], cascading failures and recovery in interconnected power grid and communications networks [11], interconnected transportation networks [12] etc.

Operator scheduling problem: The job-shop scheduling problem along with operators is studied extensively in the literature. Several approaches are discussed in the literature to solve this problem: in [13] the authors have proposed artificial intelligence schemes to solve this problem, whereas the authors in [14] 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 [15]. The resource scheduling problem has been used for different applications, such as in pharmaceutical environments, handicraft productions [16] etc.

Our proposed scheme is significantly different than the above schemes in a number of ways. All of the above scheduling schemes try to reduce the overall makespan or total completion time, whereas our objective is to maximize the total up-time of the services in a recovery process. This needs the clear understanding and modeling of different services and their dependent applications by using a two-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 environment is not considered in the above literature.

V. 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 overly 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.

This paper is a preliminary study to model the progressive restoration problem in the context of large enterprise data centers. To model this we have 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 comes 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. Finally, we plan to do a more thorough evaluation of our proposed method based on data from a much larger set of real-world scenarios.

REFERENCES

[1] http://iwgcr.org/japan-earthquake-puts-data-centers-and-cloud-services-at-risk/.
[2] http://www.datacenterdynamics.com/content-tracks/power-cooling/hurricane-sandy-data-center-stories-from-manhattan/72772.fullarticle.
[3] https://thestack.com/data-centre/2015/08/19/lightning-wipes-storage-disks-at-google-data-centre/.
[4] http://www.americanbanker.com/issues/176_195/bank-of-america-website-outage-online-banking-1042932-1.html.
[5] http://aws.amazon.com/message/65648/.
[6] http://www.drj.com/drj-world-archives/general-dr-planning/feed/Page-1.html.
[7] http://ampl.com/products/solvers/.
[8] H. W.Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.
[9] S.Mitchell *et al.*, "Pulp: A linear programming toolkit for python," 2011.
[10] C. D.Brummitt *et al.*, "Suppressing cascades of load in interdependent networks," *Proceedings of the National Academy of Sciences*, vol. 109, no. 12, pp. E680–E689, 2012.
[11] S.Soltan *et al.*, "Cascading failures in power grids: analysis and algorithms," in *International Conference on Future Energy Systems*, 2014, pp. 195–206.
[12] A.Halu *et al.*, "Emergence of overlap in ensembles of spatial multiplexes and statistical mechanics of spatial interacting networks ensembles," 2013.
[13] T.Yamada *et al.*, "Genetic algorithms for job-shop scheduling problems," in *Modern Heuristic for Decision Support*, 1997, pp. 474–479.
[14] M. R.Sierra *et al.*, "Optimally scheduling a job-shop with operators and total flow time minimization," in *CAEPIA*, 2011, pp. 193–202.
[15] O.Guyon *et al.*, "Cut generation for an integrated employee timetabling and production scheduling problem," *European Journal of Operational Research*, vol. 201, no. 2, pp. 557–567, 2010.
[16] A.Agnetis *et al.*, "A job shop scheduling problem with human operators in handicraft production," pp. 3820–3831, 2014.