Error Tolerant Address Configuration for Data Center Networks with Malfunctioning Devices

Xingyu Ma[†,‡], Chengchen Hu[†], Kai Chen[♮], Che Zhang[†],
Hongtao Zhang[†], Kai Zheng[§], Yan Chen[♮], Xianda Sun[†,‡]
[†]MOE KLINNS lab, Department of Computer Science and Technology, Xi'an Jiaotong University
[‡]National Laboratory for Information Science and Technology, Tsinghua University
[♮]Department of Electrical Engineering and Computer Science, Northwestern University
[§]IBM China Research Lab

***Abstract*—Address auto-configuration is a key problem in data center networks, where servers and switches encode topology information into their addresses for routing. A recent work DAC [2] has been introduced to address this problem. Without malfunctions, DAC can auto-configure all the devices quickly. But in case of malfunctions, DAC requires significant human efforts to correct malfunctions and it can cause substantial operation delay of the whole data center.**

In this paper, we further optimize address auto-configuration process even in the presence of malfunctions. Instead of waiting for all the malfunctions to be corrected, we could first configure the devices that are not involved in malfunctions and let them work first. This idea can be translated to considerable practical benefits because in most cases malfunctions in data centers only account for a very small portion. To realize the idea, we conceptually remove the malfunctions from the physical data center topology graph and mathematically convert the address configuration problem into induced subgraph isomorphism problem, which is NP-complete. We then introduce an algorithm that can solve the induced subgraph isomorphism quickly by taking advantage of data center topology characteristics and induced subgraph properties. We extensively evaluate our design on representative data center structures with various malfunction scenarios. The evaluation results demonstrate that the proposed framework and algorithm are efficient and labor-free to deal with the mapping task in the presence of error devices.

***Index Terms*—Data Center Networks, Address Configuration, Induced Subgraph Isomorphism Problem**

I. Introduction

The cloud computing is fast evolving. As the infrastructure of cloud computing, Data Center (DC) is kept enlarging its scale and it is common to contain hundreds of thousands of servers in a DC nowadays [3, 10]. For reliability and performance purposes in huge data centers, locality and topology information has been encoded into the Data Center Network (DCN) addresses. For example, in a distributed file system such as GFS [14], a chunk of data is usually replicated several times and stored on servers on different racks. To optimize the performance, people may embed locality information into IP addresses of servers such that it is easier to know how to fetch data from a nearby server rather than a remote one from the IP address. The recently proposed data center structures [1, 6]–[8, 17] go one step further to design new addresses or logical IDs for servers and switches that describe their positions in the topology. This topology information is essential for high-performance routing of these new structures. For instance, BCube [7] can build its node-disjoint parallel routing path solely based on the source and destination BCube logical IDs. PortLand [11] switches forward packets according to location information of the destination Pseudo-MAC.

While such kind of topological information embedded addressing has greatly facilitated the network performance of data centers, it also poses challenges for the operations. How to automatically configure the logical IDs is one of such challenges. A logical ID is more than an IP address, which is only required to be in an address range, but also carries topology information. Therefore, the traditional automatic address configuration schemes such as DHCP [12] are not able to work directly. In a recent work [2], Chen et al proposed DAC, a generic and automatic address auto-configuration scheme for data center networks. DAC assumes a blueprint graph (with logical IDs labeled) and a physical topology graph of the to-be-configured data center (with device IDs labeled), and performs the device-to-logical ID mapping by leveraging graph isomorphism theory [15]. DAC is the first and significant step on automatically configuration of DCN. If the physical data center is built exactly as what the blueprint defines, DAC can auto-configure all the devices very efficiently. However, DAC is subjected to several constraints when there are malfunctions[1] in a DCN.

Given data centers are usually of large scale and are with strict wiring rules (*e.g.*, BCube [7] and DCell [8]), malfunctions brought by mis-wirings are not uncommon. In case of malfunctions, DAC first reports all the malfunction-related devices and wait for manual correction of malfunctions before auto-configuration. The malfunction correction process is a great human effort and can cause substantial operation delay of the whole data center, and thus it does not make sense for majority of the well-functioning devices to wait for the long manual correction process of the malfunctions.

Xingyu Ma and Xianda Sun are guest students at MOE Key Lab for Intelligent Networks and Network Security (MoE KLINNS Lab) of Xi'an Jiaotong University during the period of this work.

This paper is in part supported by the National Natural Science Foundation of China (60903182, 60921003), the Fundamental Research Funds for Central Universities.

[1]Malfunctions in this paper refer to mis-wirings, broken NICs, broken wires and etc.

In fact, the correction process can be very difficult for large-scale DCN. Take the malfunctions of miswiring as example. Even with the knowledge of mis-wired nodes and links reported by DAC, it still leaves unknown how to connect these nodes in a right order. A simple and intuitive way to reconnect the x mis-wired nodes in a right order is trying all the connection permutations. It is normal that x is in the magnitude of tens for a DCN with hundreds of thousands of servers. If we have 20 malfunctioning devices, and each of which has only 2 connection options, the possible permutation has already reached 2^{20}. It would be the nightmare of the operators. As a result, a more efficient auto-configuration, when malfunctions exist, is a necessity rather than an option.

In this paper, we advocate a new framework ETAC (Error Tolerant Address Configuration) to configure data center addresses in the presence of malfunctioning without involving any human efforts. We aim to auto-configure the devices that are not involved in malfunctions and let them work first, instead of relying on time-consuming manual correction of malfunctions. The idea in ETAC can bring considerable practical benefit because in most cases malfunctions in data centers, if exist, only account for a very small portion [2]. This means that the majority part of data center is correct and is able to operate first. A brief comparison of ETAC and previous DAC is stated in Table I. Compared with DAC, ETAC considers the malfunctioning cases during its design and does not involve any human efforts, and therefore the total configuration time of ETAC can be much shorter than DAC if malfunctions exist.

To realize the above idea, in ETAC framework, we first abstract the physical topology into a conceptual graph and remove the malfunctions from the graph, and then we mathematically formulate address auto-configuration problem into induced subgraph isomorphism problem [16]. To seek a quick solution to the problem, we further propose a Subgraph Mapping Algorithm (SMA) that leverages the characteristics of data center topology. After that, we extensively evaluate our design on representative data center structures with various malfunction scenarios. Our evaluation results suggest that ETAC framework could efficiently realize mapping for large-scale data center structure under different error patterns.

It is worth noting that for generic graphs, induced subgraph isomorphism problem[2] is NP-complete [16], while graph isomorphism problem is in the low hierarchy of class NP[3] [15]. Therefore, the abstracted problem of ETAC is intrinsically different from the graph isomorphism in DAC and is of greater hardness. Specifically, we make the following main contributions in this paper.

- To the best of our knowledge, we are the first to define the problem of configuring the address in the malfunctioning data centers and propose a framework called ETAC to solve the problem. We further propose a solution to this problem by formulating it to induced subgraph isomorphism problem mathematically. With such a formulation, ETAC could obtain the mapping between well-functioning devices and addresses/logical IDs.
- Although the generic induced subgraph isomorphic problem is NP-complete, we design a practical subgraph mapping algorithm exploiting the features in existing data center structures and properties of the induced subgraph to configure the addresses for devices that are free from malfunctions. The algorithm is efficient: even if the DCN is in a scale of hundreds of thousands devices, it could accomplish the mapping within 5 minutes.
- The troublesome and time-consuming manual process is removed from the framework of the auto-configuration process. Therefore, the total configuration time of the well-functioning devices can be significantly reduced in ETAC compared with previous work.

The rest of the paper is organized as follows. Section II presents the problem statement and Section III proposes the subgraph mapping algorithm to solve the key subgraph isomorphic problem. We validate our design via extensive experiments and simulations in Section IV and summarize the related work in Section V. Eventually in Section VI, we conclude the paper.

TABLE I
A COMPARISON BETWEEN ETAC AND DAC.

Framework	Error-tolerance	Manual Efforts
DAC	not considered	involved
ETAC	considered	not involved

TABLE II
TABLE OF NOTATIONS.

Notations	Descriptions
$G_b/G_m/G_s$	the blueprint/physical/device graph
$\Omega_b(i)/\Omega_s(i)$	atoms for blueprint/device graph and i is the index number
Ψ_b/Ψ_s	atoms group for blueprint/device graph
$V_b(k)/V_s(k)$	injected vertexes after the $k-th$ step search on G_b/G_s

II. PROBLEM OVERVIEW

To accomplish the task to automatically configure all the devices that are not involved in malfunctions, we have two graphs as input: one is a blueprint graph and the other is a physical graph. The blueprint of the data center network is assumed to be known in advance, which defines the connections of the servers and switches, and labels each machine with a proper address or logical ID. The physical graph reflects the real connections among the machines in the data center, and it can be collected using Physical topology Collection Protocol (PCP) proposed in [2]. As an example, Figure 1 (a) is a blueprint, and characters $A-H$ in the figure are the addresses or logical IDs that should be configured to the physical machines in the corresponding locations; while Figure 1 (b) is the physical graph with physical device ID of $1-8$. In this example, we assume that there is a missing link between node[4] 7 and 8 in the physical topology as indicated

[2]Without confusion, we use "subgraph isomorphism" and "induced subgraph isomorphism" interchangeable in this paper.

[3]It is not NP-complete unless the polynomial time hierarchy collapses to its second level.

[4]In this paper, we use the terms vertex and node interchangeably.

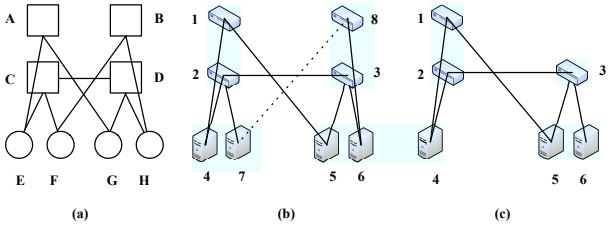

Fig. 1. The graph: (a) is the blueprint; (b) is the raw physical graph (*i.e.*, physical graph); (c) is device graph with error nodes removed(*i.e.*, device graph).

by a dotted line between node 7 and 8.

When using DAC [2] to configure the address in a malfunctioning data center, the processing framework is as shown in Figure 2 (a). A manual correction process of the malfunctions is required by DAC and this process delays the configurations of most of the devices that are in right places. Considering the example in Figure 1 (a), the malfunctioning detection will report the locations of errors, since there is a missing link between node 7 and 8 in the physical graph (the detection method is proposed in [2]). Only after all the malfunctions being corrected by a manual process, all the devices can get configured in a batch. This process forces most of the machines, even in their right places, to wait for a relatively long manual correction period.

In this paper, we construct a different ETAC framework, which is depicted in Figure 2 (b). Rather than waiting for the manual correction after the detection of malfunctions as what DAC does, ETAC first generates a third graph called device graph, which excludes the corresponding malfunctioning links and nodes. Please note that we just remove the malfunctions in the abstracted graph (*e.g.*, Figure 1 (b)) to generate the device graph (*e.g.*, Figure 1 (c)) and there is no human labors at changing the real wiring connections between devices. Continuing the example in Figure 1 (a) and (b), we now have the device graph in Figure 1 (c). All the interconnections between devices in Figure 1 (c) are guaranteed to be in accordance with the design in the blueprint, and thus Figure 1 (c) is an induced subgraph of Figure 1 (a). The next step in the proposed system is to find the injection from the well-functioning nodes in Figure 1(c) to the blueprint in Figure 1 (a), namely $\{1-6\} \rightarrow \{A-H\}$. Such a mapping problem is the key component of the proposed system and we have developed a Subgraph Mapping Algorithm to solve it. In fact, it is a induced subgraph isomorphism problem mathematically. With the notations in Table II, we give the formal definition of the problem.

Denote $G_b =< V_b, E_b >$ as the blueprint graph (Figure 1 (a)) and $G_m =< V_m, E_m >$ as the physical graph (Figure 1 (b)). Suppose subset $V_e \subset V_m$ includes all malfunctioning nodes. By removing V_e and edges E_e connected to $V_e(E_e = \{e = ij | i \in V_e \text{ or } j \in V_e\})$ from V_m and E_m, we obtain the device graph $G_s =< V_s, E_s >=< V_m \backslash V_e, E_m \backslash E_e >$. G_s, is an induced subgraph to G_b. Our task is to find a mapping from V_s to V_b, which is called the *induced subgraph isomorphism* problem.

Definition 1: A graph $G1 =< V_1, E_1 >$ is isomorphic to a subgraph of a graph $G2 =< V_2, E_2 >$, denoted by $G_1 \cong S_2 \subseteq G_2$, if there is an injection f: $V_1 \rightarrow V_2$ such that, for each pair of vertices $v_i, v_j \in V_1$, $(v_i, v_j) \in E_1$, if and only if $(f(v_i), f(v_j)) \in E_2$. Such an injection is called *induced subgraph isomorphism*.

Although speed-up techniques in DAC [2] successfully solve the graph isomorphism problem, it is not applicable to use them for the induced subgraph isomorphism problem we aim to solve in this paper. As we mentioned above, the two problems are intrinsically different. The potential isomorphism conditions, which greatly prune the search space for isomorphism problem, are not true any more for the induced subgraph isomorphism problem. For example, a possible mapping between the blueprint and the device graph must have the same degree; however, for the subgraph isomorphism problem, the degree of a node in the device graph can be less than the degree of its mapped node in the blueprint. Let us revisit Figure 1 again, Node C in Figure 1 (a) is the mapped node of Node 2 in Figure 1(c), but the degree of these two nodes are not the same. Besides the degree metric, the subgraph isomorphism problem violates many other conditions in the isomorphism problem and increases the search space a lot. In addition, many speed-up techniques for graph isomorphism, which require complete information of isomorphic graphs, are not applicable for induced subgraph isomorphism either, in case of even one node removal. Therefore, new techniques should be investigated for the induced subgraph isomorphism problem.

In the past 40 years, the induced subgraph isomorphism problem has been studied by many researchers. For general graph, the induced subgraph isomorphism is a generalization of both the maximum clique problem and maximum independent set problem, which is NP-complete [16]. Such complexity is intolerable for data center network, which has tens of thousands of nodes. We devise an algorithm, which leverages some features of data center networks and induced subgraph properties. Based on a general framework for graph structural information discovery, we have achieved an induced subgraph isomorphism mapping algorithm to be presented in the next section.

III. Subgraph Mapping Algorithm

In this section, we propose the Subgraph Mapping Algorithm (abbreviated as SMA) to map the devices that are not involved in malfunctions to the addresses/ID in the blueprint. We will first describe the top level algorithm of SMA and the challenges. With an analysis on the properties of the algorithm, we further present the detailed methods to guarantee the correctness and to accelerate the algorithm.

A. Top level algorithm

Before delving into the details, we first present the symbols used in this section. For a graph $G =< V, E >$, a *partition* of vertex set V is a group of k subsets $\Psi = \{\Omega(i) | i = 0...k-1\}$, where $\Omega(i)$ is set of vertices ($\Omega(i) \subseteq V$), $\forall i, j$, $\Omega(i) \cap \Omega(i) = \Phi$ and $\bigcup_i \Omega(i) = V$. It is worth noting in our definition that $\Omega(i)$ can be empty sets. For the ease of presentation, $\Omega(i)$ is

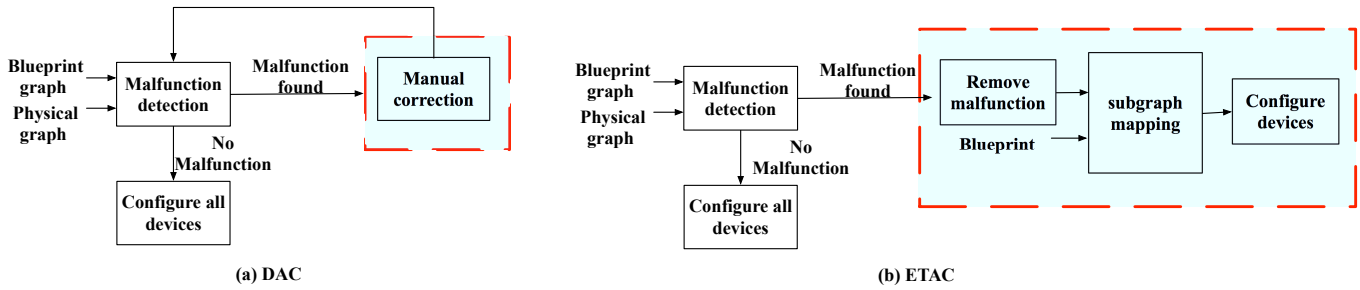

Fig. 2. The framework of DAC and ETAC.

named as an *atom*, i of $\Omega(i)$ is called the *index* and Ψ is called an *atom group* in the rest of the paper. Similarly, we use $\Psi_s = \{\Omega_s(i)|i = 0...k-1\}$ and $\Psi_b = \{\Omega_b(i)|i = 0...k-1\}$ to describe the partitions in $G_s =< V_s, E_s >$ and $G_b =< V_b, E_b >$, respectively.

The basic idea of SMA is a kind of "divide and conquer" search algorithm. It recursively partitions the graph (both G_b and G_s) into smaller atoms until each vertex in G_s finding its injection/mapping in G_b or a wrong partition of the graph in previous steps is detected. In the former case, the algorithm stops as successfully obtaining the solution; while in the latter case, the algorithm backtracks to the previous steps and partitions the graph in another direction. In each recursive step of the algorithm, the number of atoms $\Omega_s(i)$ in Ψ_s and the number of $\Omega_b(i)$ in Ψ_b are the same. If each element in $\Omega_s(i)$ can be potentially mapped to an element in $\Omega_b(j)$, we set $i = j$ and call $\Omega_s(i)$ and $\Omega_b(i)$ a *corresponding atom pair*. In each step, SMA partitions on corresponding atom pairs of the two graphs. Finally, SMA partitions all non-empty atoms $\Omega_s(i)$ in Ψ_s to single-element atoms, and each non-empty single-element atom $\Omega_s(i)$ is then injected to an single-element $\Omega_b(i)$. Thus, we can then configure the addresses of the vertices in G_s according to the addresses/ID in blueprint. It is clear that the key problems of SMA are how to partition the graph and how to decide whether the decomposition is in the right direction. Decomposition and refinement operations are designed to solve these two key issues. The top level algorithm is stated in Algorithm 1 and the description of decomposition and refinement operation is articulated below.

- *Decomposition:* Given an element v in a non-empty and non-single device graph atom $\Omega_s(i)$, such atom is decomposed into $\Omega_s(i)\backslash\{v\}$ and $\{v\}$. Other atoms remain unchanged in partition Ψ_s. Correspondingly, there will be an u in $\Omega_b(i)$ decomposing $\Omega_b(i)$ into $\Omega_b(i)\backslash\{u\}$ and $\{u\}$. For each newly emerging single-element atom in $\{v\}$ and its corresponding $\{u\}$, we call such a $v \to u$ relation a *"pair"*. The principle to select u and v in order to accelerate the algorithm will be discussed in the following subsections.
- *Refinement:* This operation is only triggered after decomposition. For the pair $v \to u$, v will split every other non-single atom $\Omega_s(j)$ into two parts according to whether elements in that atom are connected to v or not. Similar operation is also conducted for u. If there are emerging new pairs (corresponding single-element atoms), the split operation continues until there are no new-emerging pairs. During the split process, refinement will judge whether current corresponding partition is valid. Details of the judgment will be explained later.

Algorithm 1 SMA(Ψ_s, Ψ_b)

```
1:  if the non-empty atoms in Ψ_s are all single-element ones
    and are mapped to single-element atoms in Ψ_b then
2:    obtain the mapping and exit;
3:  else
4:    select a vertex v ∈ Ω_s(i); //Ω_s(i) is not a single-element
      atom
5:    for each vertex u ∈ Ω_b(i) do
6:      decomposition(Ψ_s, Ψ_b, v, u);
7:      if refinement(Ψ_s, Ψ_b)== true then
8:        SMA(Ψ_s, Ψ_b);
9:      end if
10:   end for
11: end if
```

It is worth noting that empty-set atom should be allowed for decomposition and refinement operations. During the refinement, it is possible that an empty atom might be split out in the refinement process. For pair $v \to u$, it is possible that v is connected to all elements $v_0, ...v_t$ in a certain atom $\Omega_s(j)$, while u is connected to part of elements in $\Omega_b(j)$ (*i.e.*, u is connected to $u_0, ...u_t$, but not connected to $u_{t+1}, ...u_r$), because essentially edge set E_s of induced subgraph G_s is a subset of E_b (edge set of G_b). Therefore, $\Omega_s(j)$ is split as $\{v_0...v_t\}$ and empty set $\emptyset$. $\Omega_b(j)$ is split as $\{u_0, ...u_t\}$ and $\{u_{t+1}, ...u_r\}$. $\{v_0, ...v_t\}$ is corresponding to $\{u_0, ...u_t\}$ and $\emptyset$ is corresponding to $\{u_{t+1}, ...u_r\}$. Similar situation is also applied for decomposition if $\Omega_s(i)$ only has one element, while $\Omega_b(i)$ has multiple ones. In this case, the decomposition on the single element of $\Omega_s(i)$ will incur an empty-set atom.

The framework of Algorithm 1 is a general structure, which is applicable for many structural graph problems (like induced subgraph isomorphism, graph isomorphism, and symmetric finding [5]). Based on the general structure, to support specific efficient induced subgraph isomorphism, SMA needs to exploit methodologies to validate the correctness of induced subgraph mapping and to accelerate the search process.

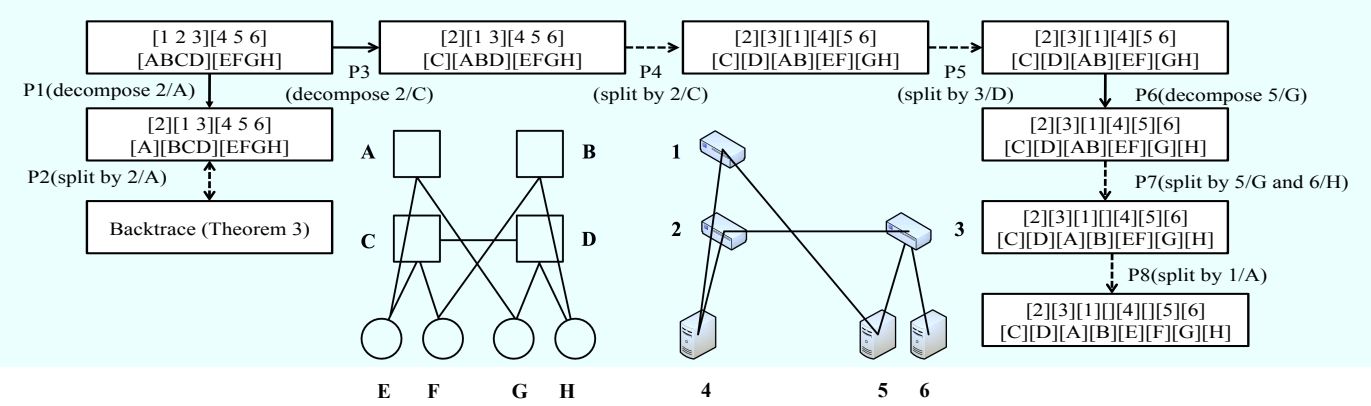

Fig. 3. A walk-through example of Subgraph Mapping Algorithm.

B. Properties analysis

In this subsection, we study the theoretic foundations for us to guarantee correctness and to speed up the algorithm. Applications and examples for these theorems are stated in the next three subsections.

The Subgraph Mapping Algorithm is essentially a search-based algorithm. We denote the depth of search as *step*, each of which comprises of two operations: decomposition and refinement. If an atom $\Omega_s(i)$ in Ψ_s and its corresponding atom $\Omega_b(i)$ in Ψ_b are both single-element atoms, we call the two atoms a *mapped node pair*. We denote all the mapped node pairs before the $k-$th step as $V_s(k) \rightarrow V_b(k)$. Regarding specific injection of v in $V_s(k)$ to u in $V_b(k)$, we represent it as $f(v) = u$.

The following Theorem 1 is a guarantee of the correctness of the algorithm.

Theorem 1: For every step (*e.g.*, $k-$th step) in the algorithm, if there are x new-emerging mapped pairs $\{v_i \rightarrow u_i | i = 0...x-1\}$ brought by the decomposition or refinement operation in the current step, we denote potential $V_s(k+1)$ as $V_s(k) \cup \{v_0...v_{x-1}\}$ and $V_b(k+1)$ as $V_b(k) \cup \{u_0...u_{x-1}\}$. The validity of the new-emerging pairs lies in the requirement: for every $v_i (i = 0...x-1)$ and every v' in $V_s(k+1)$, $(v_i, v') = 1$ if and only if $(u_i, f(v')) = 1$ (we denote $(x, y) = 1$ if x, y is connected and denote f as the injection relation). If the algorithm performs under such a requirement at every step, the final result of mapping is correct.

Proof:

Denote k as the number of step in the algorithm. We first demonstrate that before every step of search, the graph constructed by $V_s(k)$ and $V_b(k)$ are isomorphic.

For $k = 0$, the set $V_s(k)$ and $V_b(k)$ are both empty set. Hence, isomorphism is fulfilled.

Suppose when $k = t$, the graph constructed by set $V_s(t)$ and $V_b(t)$ are isomorphic. In other words, for every $v_p, v_q \in V_s(t)$, $(v_p, v_q) = 1$ in G_s if and only if $(f(v_p), f(v_q)) = 1$ in G_b.

In the $t-$th step, suppose there are x newly emerging pairs $\{v_i \rightarrow u_i | i = 0...x-1\}$. As stated in the description of Theorem, $V_s(t+1)$ is $V_s(t) \cup \{v_0...v_{x-1}\}$ and $V_b(t+1)$ is $V_b(t) \cup \{u_0...u_{x-1}\}$. For every $v_p, v_q \in V_s(t+1)$, if either v_p or $v_q \in \{v_i | i = 0...x-1\}$, according to the statement of Theorem, $(v_p, v_q) = 1$ if and only if $(f(v_p), f(v_q)) = 1$. Otherwise, both v_p and $v_q \in V_s(t)$. Therefore, the supposition guarantees that $(v_p, v_q) = 1$ if and only if $(f(v_p), f(v_q)) = 1$. Therefore, based on the supposition at $k = t$, For every $v_p, v_q \in V_s(t+1)$, $(v_p, v_q) = 1$ if and only if $(f(v_p), f(v_q)) = 1$. When $k = t+1$, the graph constructed by set $V_s(t+1)$ is isomorphic to the graph constructed by $V_b(t+1)$.

Therefore, for any step i in the algorithm, the graph constructed by $V_s(i)$ and $V_b(i)$ are isomorphic. Of course, in the final step i_final of algorithm, the partitions of V_s are all single-element atoms. The graphs constructed by $V_s(i\_final)$ and $V_b(i\_final)$ are also isomorphic. Hence, the correctness of SMA is proved. ■

In the proof of Theorem 1, to support the correctness, we leverage the properties of the induced subgraph. Furthermore, the proof stated also demonstrates that the constructed graph at each step of search is isomorphic to each other.

Apart from Theorem 1 which suffices correctness, we also investigate some necessary conditions. To better leverage the properties provided by the induced subgraph and sparsity of data center structures, we present the following Theorems, which are necessary conditions for the correctness of the algorithm.

Theorem 2: During any stage in the algorithm, for any $\Omega_s(i)$ and corresponding $\Omega_b(i)$, $|\Omega_s(i)| \leq |\Omega_b(i)|$.($|Set|$ denote the cardinality of the Set.)

Proof:

Suppose there is a stage in the algorithm, in which a $\Omega_s(i)$ and it corresponding $\Omega_b(i)$ have the relation: $|\Omega_s(i)| > |\Omega_b(i)|$. However, if there exists a valid one-to-one injection f from elements in $\Omega_s(i)$ to elements in $\Omega_b(i)$, $|\Omega_s(i)| = |\{f(v) | v \in \Omega_s(i)\}| \leq |\Omega_b(i)|$, contradicting to the supposition. ■

Theorem 2 is a natural and obvious conclusion: the device graph atoms should not be larger than their corresponding blueprint atoms.

Theorem 3: During any stage in the algorithm, for any newly emerging pair $v \rightarrow u$, denote $T_s(v)$ as the set of the index number of $\Omega_s(i)$ connected to v (Formally, $T_s(v) = \{i | \exists t \in \Omega_s(i)$ and $(t, v) = 1\}$ and $T_b(u)$ as the set of the index number of $\Omega_b(i)$ connected to u($T_b(u) = \{i | \exists t \in \Omega_b(i)$ and $(t, u) = 1\}$). The following statement should hold: $T_s(v) \subseteq T_b(u)$.

Proof:

For any step in the algorithm, if $\Omega_s(i)$ and $\Omega_b(i)$ are

corresponding atoms and such corresponding relation could finally reach a correct mapping relation, each element v' in $\Omega_s(i)$ will be mapped to an element $f(v')$ in $\Omega_b(i)$.

Therefore, suppose there is a stage in the algorithm, a new pair $v \to u$ emerges and the statement in Theorem does not hold. Therefore, there must be an $\Omega_s(i)$ and $\Omega_b(i)$, v is connected to some elements v' in $\Omega_s(i)$, while u is not connected to any elements in $\Omega_b(i)$. However, if current step could finally reach a correct mapping, there would be an $f(v')$ in $\Omega_s(i)$. Furthermore, according to the definition of induced subgraph isomorphism, if $(v, v') = 1$, then $(u, f(v')) = 1$, contradicting with the status that u is not connected to any elements in $\Omega_b(i)$. ■

Theorem 3 is primarily used for checking whether the newly emerging pairs are valid or not during refinement, supplementing Theorem 1. Intuitively, if the pair $v \to u$ is correct, the nodes connected to v should be mapped to a subset of nodes connected to u. Therefore, the subset relations on atom level should also hold.

C. Correctness and acceleration in Refinement operation

For each refinement operation, it needs to return true or false. The three Theorems are utilized for the return value of that judgment. As stated above, Theorem 1 guarantees that such judgment is valid, *i.e.*, true judgments could finally bring about a correct mapping. Theorem 2 and Theorem 3 are primarily used for pruning based on some properties provided by the induced subgraph, *i.e.*, any violations of the two Theorems will make the judgment return false.

Moreover, Theorem 3 can be better exploited for sparse graphs. First, for sparse graphs, the cardinality of $T_s(v)$ and $T_b(u)$ should be comparatively smaller and reduces the complexity of the judgment on this. Second, sparse graph will incur more opportunities of pruning via Theorem 3. These facts are obviously good news for data center structures, which are usually sparse graphs.

D. Acceleration in Decomposition operation

In line 4 of Algorithm 1, we need to select a node from a non-single atom in G_s. Actually a strategic selection of such node is much more effective than a random selection, although the latter one would not affect the correctness of algorithm. Our strategy is based on a priority rule. The neighboring nodes to the currently mapped nodes are given higher priority for selection.

Such strategy derives from an in-depth understanding of Theorem 1. As has been discussed above, Theorem 1 actually guarantees that at any steps in the algorithm, the two graphs constructed by the nodes which are currently mapped are isomorphic. Therefore, if current isomorphism can not be met, it is impossible that the final result is correct. However, connectivity at the current mapped nodes greatly influences whether Theorem 1 is effectively used for pruning or not. We use the two graphs in Figure 3 to clarify this point. For example, a current mapping is $f(1) = C, f(2) = B, f(6) = G$. It is obvious that such mapping is wrong at a glance. However, Theorem 1 will still consider current mapping is acceptable, because graphs constructed by $\{1, 2, 6\}$ and $\{C, B, G\}$ are isomorphic (actually these nodes are connectionless with each other). Comparatively, the isomorphism brought by $f(1) = A, f(2) = C, f(4) = E$ is much more indicative because these nodes($\{1, 2, 4\}$ and $\{A, C, E\}$) are connected, which makes the current isomorphism more indicative and effective.

Hence, we employ this strategy to build on the efficiency of our algorithm.

E. A walk-through example of the algorithm

To give a clear understanding of Subgraph Mapping Algorithm(SMA), a illustration is given in Figure 3. The solid-line arrow represents the decomposition operation and the dotted-line represents the split in refinement operation. P1-P8 is the sequence of procedures of this algorithm. The initial state is presented in the left-top corner: $\Psi_s = \{\Omega_s(i)|i = 0, 1\} = \{\{1, 2, 3\}, \{4, 5, 6\}\}$, $\Psi_b = \{\Omega_b(i)|i = 0, 1\} = \{\{A, B, C, D\}, \{D, E, F, G\}\}$. The initial state is according to the classification of servers and switches. Please note that the judgment of refinement is performed at all splits, but in this example we just select some for illustration purpose. In most steps, there can be several possible selection of decomposition and refinement; while in this example, we use some typical one for better illustration of the algorithm.

Suppose in P1, we first select 2 in G_s and A in G_b to decompose Ψ_s/Ψ_b with $2 \to A$. P2 uses the newly emerging pair $2 \to A$ to split other non-single atoms. Here, we demonstrate how Theorem 3 is applied. Now, $\Psi_s = \{\{2\}, \{1, 3\}, \{4, 5, 6\}\}$ and $\Psi_b = \{\{A\}, \{B, C, D\}, \{E, F, G, H\}\}$. We find that in G_s, 2 is connected 3 and 4, which belong to $\Omega_s(1)(\{1, 3\})$ and $\Omega_s(2)(\{4, 5, 6\})$ respectively. In contrast, in G_b, A is connected to E and G, which belong to $\Omega_b(2)(\{E, F, G, H\})$. Thus, $T_s(2) = \{1, 2\} \nsubseteq T_b(A) = \{2\}$, which violates Theorem 3. In this case, the judgment of refinement is returned as false and the algorithm is backtracked in P2.

In P3, decomposition using $2 \to C$ is operated with a new selection C in G_b. The following split is conducted in P4 and it is operated based on whether the elements in non-single atoms are connected to the single-element atom. For Ψ_b, C is connected to D, E, F. Accordingly, $\{A, B, D\}$ is split to $\{D\}$ and $\{A, B\}$. $\{E, F, G, H\}$ is split to $\{E, F\}$ and $\{G, H\}$. Similar operation is conducted on Ψ_s. During split in P4, a new pair $3 \to D$ emerges. Consequently, the refinement continues in P5 by using $3 \to D$ to split other non-single atoms. In this split, no atoms changes and we come to P6.

In P6, a new step of decomposition comes and we needs to first select an element in G_s. With the strategic vertex selection to get the neighboring vertices first, 5, which is connected to currently mapped 3, is selected. Decomposition using $5 \to G$ is operated.

In P7, there are two newly emerging single-element atom pairs ($5 \to G$ and $6 \to H$). We use this procedure to explain how Theorem 1 is leveraged. Before this operation, the mapping relation is that $f(2) = C$ and $f(3) = D$, we check whether $f(5) = G$ and $f(6) = H$ is allowed. (5,3)=1 and $(f(5), f(3)) = (G, D)$=1 as well. (6,3)=1 and $(f(6), f(3)) = (H, D) = 1$ too. Therefore, we could make sure

TABLE III
THE STRUCTURES FOR EVALUATIONS.

FatTree(n)	VL2(n_r, n_p)	BCube(n, k)	DCell(n, k)
F(20)= 2500	V(10,100)=27650	B(5,4)=6250	D(4,2)=525
F(40)=18000	V(20,100)= 52650	B(6,4)=14256	D(2,3)=2709
F(60)=58500	V(40,100)= 102650	B(7,4)=28812	D(3,3)=32656
F(80)=136000	V(60,100)= 152650	B(8,4)=53248	D(4,3)=221025

that graph made by nodes $\{2, 3, 5, 6\}$ in G_s and graph made by nodes $\{C, D, G, H\}$ in G_b are isomorphic to each other. Therefore, Theorem 1 judges the status before current split as valid. The split is done like stated before. Please note that empty-set atom is split to maintain the corresponding relation of atoms.

Finally, after the split in P8, all the non-empty atoms in Ψ_b are all single-element ones and then mapped to single-element atoms.

IV. EVALUATION

A. Settings

We perform the evaluations of SMA algorithm under different settings, which vary from data center network structures, sizes, the number of malfunctions and malfunction patterns. First, we investigate the relationship between the calculation time of SMA and the structure/scale of the data center. Second, we study the effect of the changes in the numbers of the malfunctions on the calculation time. Third, we study whether the locations of the malfunctions would affect the performance of the algorithm.

Four state-of-the-art data center network structures are used in the experiments, *i.e.*, FatTree [1], VL2 [6], BCube [7] and DCell [8]. The number of nodes with detailed parameter settings in these four structures is illustrated in Table III.

Two performance metrics are investigated, which are the calculation time and the consumed memory. These performance data is collected from the server, which has a Xeon 3.0GHz eight-core CPU, 8GB DRAM, 2TB disk. The programming language we use is C.

To the best of our knowledge, we are the first one to realize the induced subgraph mapping goal for data center address configuration. Until now, there is no efficient algorithm, which could configure thousand-scale malfunctioning data center in minutes. We have test brute-force induced subgraph mapping algorithm, which could not configure a 2709-scale data center within one hour. Therefore, we just present our results, which is quick enough practically.

B. Performance vs. scale

In this experiment scenario, we set the number of malfunctions to be 50 and evaluate the performance on all the 16 kinds of data center structures in Table III. To perform the evaluation, we first randomly generate the malfunctioning points and then activate the SMA. With each data center structure, the same "malfunctioning generating" and "SMA calculation" progress are repeated 100 times.

Figure 4 depicts the average calculation time of SMA under different structures/scales of the data centers. We observe from the figure that the calculation time increases as the scale of data center enlarges. However, the increase of slope is not very dramatic. In addition, all the specific experiment cases can be completed within 300 seconds, even when the number of nodes reaches 221,025 in DCell(4,3). Therefore, we argue that the average calculation time of Subgraph Mapping Algorithm is acceptable for real applications. Besides the average calculation time of Subgraph Mapping Algorithm, we also list the maximum (T_u), as well as the minimum (T_l) calculation time in 100 times experiments within the 95% confidence interval in Table IV. The unit of the calculation time in the Table IV is second. We can observe that the results for FatTree and VL2 are relatively stable and the larger fluctuation for BCube or DCell is also in an acceptable scope.

The maximum memory size (M in the table with unit of MB) is also illustrated in Table IV. We find that the larger the data center is, the larger memory size is required. The maximum memory consumptions in this evaluation scenario are 27.2MB, 26.4MB, 7.6MB and 24.4MB, respectively for FatTree, VL2, BCube and DCell.

C. Performance vs. error number

In this scenario, we test the performance of Subgraph Mapping Algorithm in the case that the number of malfunctioning nodes changes. The number of the malfunctioning nodes varies from 10 to 50, and the malfunctioning devices are randomly selected in the physical graph. Instead of performing experiments under all the 16 structures in Table III, we select four medium-size data center structures for experiment. They are FatTree(60) with 58500 nodes, DCell(3,3) with 32656 nodes, VL2(20,100) with 52650 nodes and BCube(8,4) with 53248 nodes. The calculation time of Subgraph Mapping Algorithm is shown in Figure 5. Each point in the figure is the average value over 100 times experiments. The maximum and minimum value of the calculation time among the 100-time iterations within 95% confidence interval is also recorded. The results demonstrate that the average calculation time is only slightly affected by the number of malfunctioning nodes for a fixed-size data center.

Also, we study the fluctuation of the calculation time. Over FatTree and VL2, the calculation time is about 12s and 17s respectively, with almost no fluctuation[5] when the number of malfunctioning nodes increases from 10 to 50. The fluctuation for BCube or DCell is comparatively larger, however, we still consider that it is within the acceptable range. For example, the calculation time of BCube(8,4) ranges from 41.60s to 72.96s with its average to be 46.67s. The maximal calculation time is no more than the twice of the average calculation time.

Aside from the performance results of calculation time, the memory consumptions in this evaluation are 12.6MB, 10.1MB, 7.6MB and 3.8MB over FatTree, VL2, BCube and DCell, respectively.

[5]In Figure 5, the maximum and minimum point is almost overlapped with the average point.

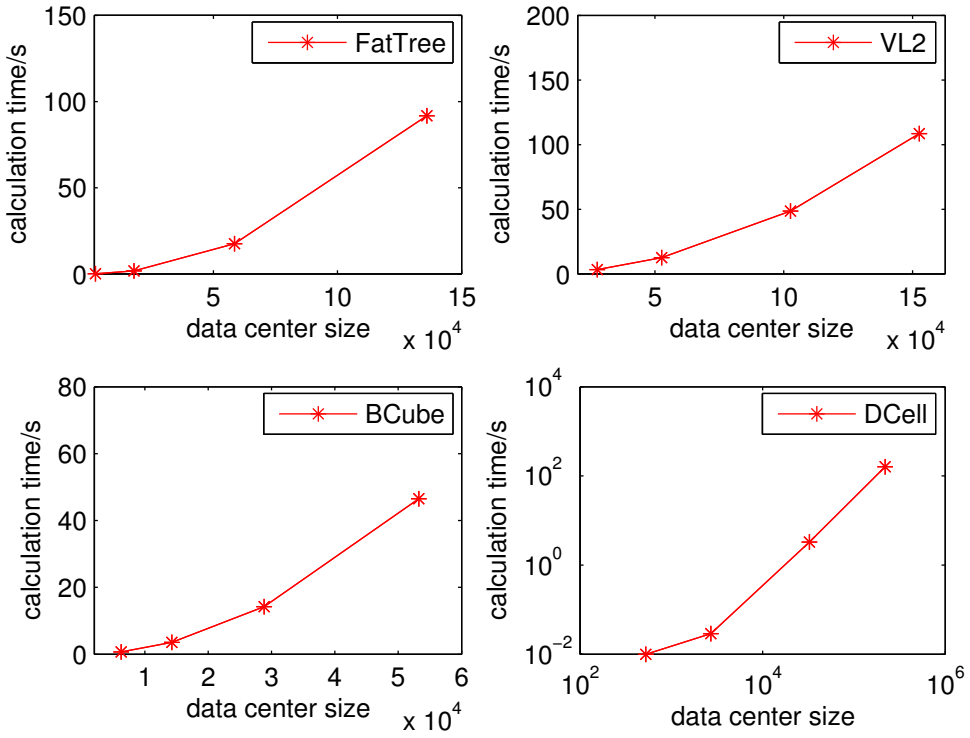

Fig. 4. The average calculation time of Subgraph Mapping Algorithm under different structures/scales of the data centers when the number of malfunctions is fixed to 50.

TABLE IV
MEMORY SIZE AND FLUCTUATION OF CALCULATION TIME UNDER DIFFERENT DATA CENTERS STRUCTURES WITH 50 MALFUNCTIONS.

FatTree				VL2				BCube				DCell			
	T_l	T_u	M		T_l	T_u	M		T_l	T_u	M		T_l	T_u	M
F(20)	0.04	0.04	0.6	V(10, 100)	3.33	3.35	5.4	B(5, 4)	0.61	1.01	1.0	D(4, 2)	0.01	0.01	0.16
F(40)	1.73	1.76	3.7	V(20, 100)	12.49	12.60	10.0	B(6, 4)	3.17	5.48	2.0	D(2, 3)	0.02	0.03	0.47
F(60)	17.35	17.50	12.6	V(40, 100)	48.44	48.89	18.8	B(7, 4)	12.53	21.67	4.3	D(3, 3)	2.70	4.39	3.6
F(80)	91.42	93.18	27.2	V(60, 100)	108.05	109.06	26.4	B(8, 4)	41.60	72.96	7.6	D(4, 3)	123.21	247.95	24.4

D. Performance vs. malfunction patterns

In the previous evaluation, the malfunction locations are randomly generated in a data center structure. To better understand whether the SMA is generic for all possible malfunction locations, we evaluate the performance by restricting the malfunctions to specific locations. In other words, we categorize errors into different patterns and investigate the possible performance variations.

To better categorize the malfunction patterns, we first classify the data center networks into switch-centric ones (*i.e.*, VL2 and FatTree) and server-centric ones (*i.e.*, BCube and DCell) [4]. For switch-centric data centers, there are usually clear "layers". The switches are usually in a hierarchical structure. For example, FatTree has four layers (cores, aggregations, edges and servers) and VL2 structure is also a four-layer structure, which involves intermediate switches, aggregate switches, ToR switches and servers. For server-centric data centers, there are usually no such clear layers for switches or servers.

First, we investigate switch-centric data center structures with malfunctions on different adjacent layers. For FatTree, we denote layer one as server, layer two as edge, layer three as aggregation and layer four as core. For VL2, we use layer one to layer four to differentiate server, ToR, aggregate and intermediate switch. We use FatTree(60) with 58500 nodes and VL2(20,100) with 52,650 nodes to carry out six groups of evaluations, *i.e.*, three over FatTree and three over VL2. In each of the group, we fix 50 malfunctions across two adjacent layers over FatTree or VL2 and repeat the experiment for 100 times. The calculation time of the experiments is illustrated in Table V. In Table V, T_l, T_a and T_u (with second as unit) are the minimum, average and maximum of the calculation time over 100 times of the experiments within 95% confidence interval. Again, the fluctuation on the calculate time is small.

For server-centric data center structure, we analyze how different proportions of error switches affect the performance. We denote α as the number of malfunctions that is in switches to all malfunctions. We use BCube(8,4) with 52,650 nodes and DCell(3,3) with 32,656 nodes as the data center structure. Also, in this experiment, the total number of the malfunctions

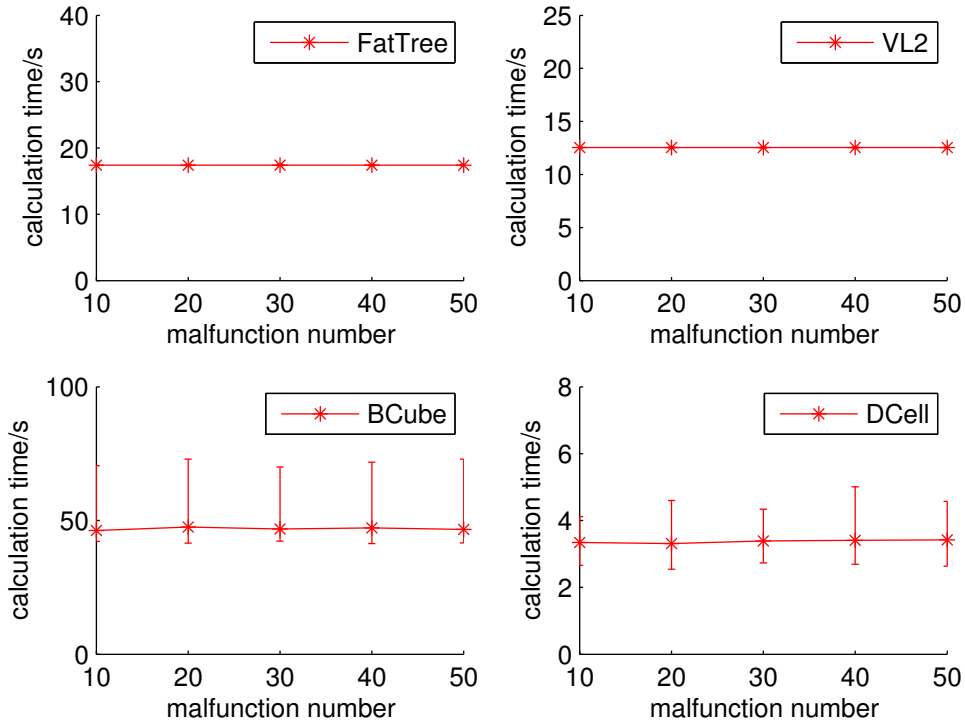

Fig. 5. The average calculation time with different malfunction numbers.

TABLE V
THE CALCULATION TIME FOR FATTREE AND VL2 WITH DIFFERENT MALFUNCTION PATTERNS.

layer	FatTree(60)			VL2(20, 100)		
	T_l	T_a	T_u	T_l	T_a	T_u
1-2	17.36	17.40	17.49	12.51	12.55	12.60
2-3	16.91	17.03	17.15	12.07	12.10	12.14
3-4	17.30	17.37	17.57	12.24	12.43	12.62

is 50. The average calculation time over 100 experiments is shown in Figure 6. The time fluctuation is again in an acceptable range.

From Table V, we find that the location of errors for switch-center data center does not affect the average calculation time much. The average performance on calculation time is almost the same for the three error patterns in FatTree or VL2. In Figure 6, we find that concerning server-centric data centers as BCube and DCell, the proportion of error switches also does not affect the calculation time. The average calculation time of SMA is about 47s over BCube(8,4) and is 3s about over DCell(3,3) under different proportions of error switches. Besides, the memory size consumed is bounded by 12.6MB, 9.9MB, 7.6MB and 3.8MB for FatTree(60), VL2(20,100), BCube(8,4), DCell(3,3), respectively.

V. RELATED WORK

The most related work of ETAC is DAC [2]. DAC malfunction handling module detects the malfunction-related devices, but it waits for all of them to be corrected before configuring the whole data center network. Correcting the malfunctions

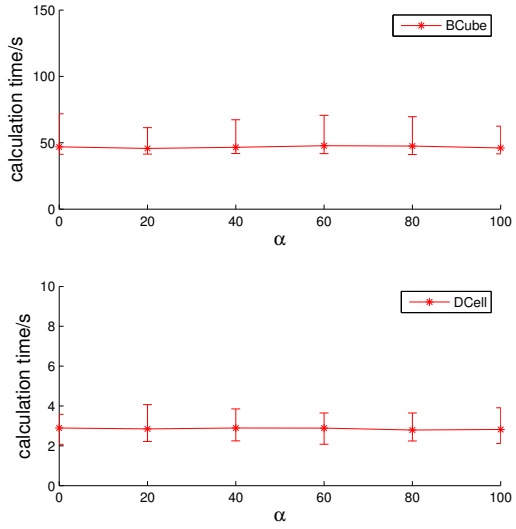

Fig. 6. The calculation time(s) for BCube(8,4) and DCell(3,3) with different malfunction patterns.

is a manual process and is time consuming. In contrast, our idea is to configure the devices of correct part and let them work first. Such simple idea involves non-trivial algorithm design and makes substantial practical benefits. DAC mapping algorithm leverages graph isomorphism theory, while our ETAC algorithm focuses on induced subgraph isomorphism problem.

PortLand [11] and DCZeroconf [9] also consider address configuration in the context of data center networking. PortLand introduces a distributed location discovery protocol

(LDP) for its PMAC address assignment. LDP assumes a multi-rooted multi-level tree topology to decide the levels of switches and then encodes this information in PMAC addresses, but it does not apply to other structures such as BCube [7] or DCell [8]. Furthermore, it does not consider the cases when the data center has malfunctions. DCZeroconf is a fully automatic address configuration mechanism to eliminate the burden of manual configurations on the IP addresses of servers and switches in data centers, however, it can not be used for non-IP data centers.

DHCP [12] is a well-known automatic configuration protocol widely used on IP networks. DHCP employs a central database for keeping track of the IP addresses that have been assigned to the network. When a new server comes, the central server will assign a new, unused IP to this server to avoid the address conflict. Zeroconf [13] could also be used for assigning IP addresses automatically. Different from DHCP, Zeroconf does not require a central DHCP server to avoid IP address conflicts. The Zeroconf enabled host first randomly picks an address and validates its availability by broadcasting queries to the network. The address will be reserved for the host if no reply shows that the address has already been occupied; otherwise the host randomly selects another address and repeats the validation process. By default, the address pool maintained in the DHCP server or the Zeroconf host does not have locality of topology embedded and is not able to work directly in data center network environments where addresses are topology-meaningful.

VI. Conclusion

In this paper, we study the automatic configuration on the addresses of the devices in data center networks with malfunctioning nodes and links. In malfunctioning data centers, previous related work configures all the devices simultaneously after a manual malfunction correction process being completed. We argue that such a manual correction period is not necessary for the devices that are not involved in any malfunctions and we aim to get these devices configured and started to work first. To achieve this purpose, we have first proposed a ETAC framework to auto-configure the addresses of devices insides a DCN with or without malfunctions and then we have modeled the mapping problem between the addresses to be configured and the physical devices as the induced subgraph isomorphic problem in mathematical formulation. In addition, we have developed an efficient induced subgraph mapping algorithm to solve the problem. By exploiting the structure characteristics of data center networks and properties of the induced subgraph, ETAC and its corresponding the mapping algorithm successfully find the solution within tens of seconds even for a data center network with scale of hundreds of thousands of devices. Also, the memory cost of the mapping algorithm is also quite limited. Again for a data center network that has more than hundreds of thousands of devices, the mapping program only occupies less than 30MB memory size.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM'08*, pages 63–74, New York, NY, USA, 2008. ACM.

[2] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu. Generic and automatic address configuration for data center networks. In *SIGCOMM*, 2010.

[3] K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, and A. Vasilakos. Survey on routing in data centers: insights and future directions. *IEEE Network*, 25(4):6 –10, july-august 2011.

[4] K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, and T. Vasilakos. Routing in data centers: Insights and future directions. In *IEEE Network Magazine - Special Issue on Cloud*, 2011.

[5] P. Darga, K. Sakallah, and I. Markov. Faster symmetry discovery using sparsity of symmetries. In *45th ACM/IEEE Design Automation Conference*, pages 149 –154, june 2008.

[6] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. Vl2: A scalable and flexible data center network. In *SIGCOMM*, 2009.

[7] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: A high performance, server-centric network architecture for modular data centers. In *SIGCOMM*, 2009.

[8] C. Guo, H. Wu, K.Tan, L.Shi, Y.Zhang, and S.Lu. Dcell: A scalable and fault tolerant network structure for data centers. In *SIGCOMM*, 2008.

[9] C. Hu, M. Yang, K. Zheng, K. Chen, X. Zhang, B. Liu, and X. Guan. Automatically configuring the network layer of data centers for cloud computing. *IBM Journal of Research and Development*, 2011.

[10] R. H. Katz. Tech Titans Building Boom. *IEEE SPECTRUM*, Feb. 2009.

[11] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM*, 2009.

[12] R.Droms. Dynamic host configuration protocol. In *RFC 2131*, 1997.

[13] B. A. S. Cheshire and E. Guttman. Dynamic configuration of ipv4 link-local addresses, August 2002. Zeroconf Working Group of the Internet Engineering Task Force (www.zeroconf.org).

[14] S.Ghemawat, H.Gobioff, and S.-T. Leung. The google file system. In *SOSP*, 2003.

[15] Wikipedia. The graph isomorphism problem. "http://en.wikipedia.org /wiki/Graph_isomorphism_problem.".

[16] Wikipedia. The induced subgraph isomorphism problem. "http:// en.wikipedia.org/wiki/Induced_subgraph_isomorphism_problem".

[17] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang. Mdcube: a high performance network structure for modular data center interconnection. In *CoNEXT '09*, pages 25–36, New York, NY, USA, 2009. ACM.